**Funkcije**

Jedinica: 10 od 19

**Rezime**

* Funkcije su jedan od ključnih elemenata OOP-a;
* Postoje ugrađene i korisnički definisane funkcije;
* Funkcija se može definisati bilo gde u kodu, a poželjno je na početku;
* Funkcije primaju parametre i vraćaju rezultat;
* Funkcije mogu imati i podrazumevane vrednosti parametara;
* Promenljive deklarisane u funkciji nisu dostupne van funkcije;
* Da bi promenljive bile dostupne van funkcije, potrebno je korišćenje ključne reči global;
* Funkcijama se smanjuje veličina koda;
* Funkcijama se povećava portabilnost koda.

U ovoj lekciji upoznaćemo se sa pojmom [funkcija](https://www.link-elearning.com/linkdl/opisPojma.php?id=139107) u PHP-u. Do sada smo se, iako još uvek nismo obradili taj pojam, dosta susretali sa funkcijama u PHP-u. Zapravo, većina naredbi koje smo do sada izvršili bile su funkcije.

Nakon sledećeg opisa, lako ćemo prepoznati kada smo se tačno u dosadašnjem radu koristili funkcijama, po njihovoj simptomatičnosti, odnosno, karakterističnoj formi.

Pre svega, da prvo razjasnimo šta je uopšte funkcija. Funkcija je neka programska struktura, koja je u stanju da na osnovu određenih primljenih parametara ili bez njih uradi neki posao i na osnovu urađenog posla vrati rezultat.

Razloga za korišćenje funkcija u programiranju ima zaista mnogo. Ali, osnovne pozitivne osobenosti funkcije su:

* + [enkapsulacija](https://www.link-elearning.com/linkdl/opisPojma.php?id=139095),
  + smanjenje redundantnosti koda,
  + brzina kodiranja,
  + lakše održavanje
  + pouzdanost

Veoma je čest slučaj da se pojavi potreba za učestalom upotrebom jednog istog bloka koda u jednom programu. Ukoliko funkcije ne bi postojale, ovaj kod bismo, u najboljem slučaju, morali da kopiramo na svako mesto gde nam je potreban.

Npr. u kodu, na više mesta ispisujemo jedan isti tekst. Recimo, zvezdice u narednom primeru:

<?php  
echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";  
echo "<br> naslov <br>";  
echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";  
echo "<br> Neki tekst <br>";  
echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";  
?>

Očigledno je da se deo koji iscrtava zvezdice ponavlja tri puta, a u programiranju postoji pravilo da isti kod ne bi trebalo da se ponavlja.

Sledeći to pravilo, dolazimo do zaključka da je neminovno rešiti problem ponavljanja zvezdica iz prethodnog koda, a najefikasniji način da to uradimo jeste upotreba funkcije.

Tako bi verzija prethodnog koda sa upotrebom funkcije izgledala ovako:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | // creating function stars  function stars(){      echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<br>";  }    //calling function stars  stars();  echo "title <br>";  //calling function stars  stars();  echo "Some text"; |

U prethodnom kodu razlikujemo dve celine. Jedna je definisanje funkcije, a druga pozivanje funkcije.

Kada je u pitanju definicija funkcije, u PHP-u moraju biti ispoštovana neka pravila. Pre svega, funkcija mora biti započeta ključnom rečju **function**. Zatim, mora posedovati naziv, za koji, što se tiče notacije, važe ista pravila kao i za promenljive (osim znaka $ ispred naziva). Nakon naziva, u malim zagradama, potrebno je naglasiti prijem parametara (kroz listu promenljivih odvojenih zarezima) ili zagrade ostaviti prazne, ukoliko funkcija ne prima parametre. I, konačno, samo telo funkcije, odnosno, kod koji funkcija izvršava i koji treba postaviti u vitičaste zagrade, odmah nakon prijema parametara:

function mojaFunkcija($parametar1, $parametar2){  
      echo "ovo je telo";  
}

Ovakav pristup ne samo da omogućava da se smanji veličina koda već i dovodi do centralizacije koda, odnosno, jednostavne manipulacije raznim mehanizmima koji se u kodu izvršavaju putem funkcija.

Recimo da primer sa zvezdicama još uvek nije prikazao pravu moć funkcija, i da je još uvek lakše kopirati jedan isti red nekoliko puta. Ali, šta ako bismo želeli da naš izlaz sa zvezdicama ima dinamički sadržaj? Na primer:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Beograd \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Tekst o Beogradu  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Pariz \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Tekst o Parizu  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* London \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Tekst o Londonu

U ovom slučaju, morali bismo, osim kopiranja zvezdica, da u svakom redu ispisujemo i naziv grada i da brišemo i dodajemo zvezdice na kraju reda, kako bi bile poravnate.

Kod bi izgledao ovako:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Beograd \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";  echo "<br>Text about Belgrade<br>";  echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Pariz \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";  echo "<br>Text about Paris<br>";  echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* London \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";  echo "<br>Text about London<br>";  ?> |

Upotrebom funkcije, ovaj kod bi se mogao rešiti na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?php  function stars( $cityName )  {  for( $i=0; $i<35; $i++ )  echo "\*";  echo " " . $cityName . " ";  for( $x= ( $i + 2 ) + strlen ( $cityName ); $x < 90; $x++ )  echo "\*";  }    stars( "Belgrade" );  echo "< br > Text about Belgrade < br >";  stars( "Paris" );  echo "< br > Text about Paris < br >";  stars( "London" );  echo "< br > Text about London < br >";  ?> |

U slučaju tri grada, koliko ih ima u primeru, postavlja se pitanje svrhe ovakve funkcije. Ali, već posle nekoliko novih gradova, kod bi se umnogostručio kada ne bi bilo ove funkcije.

Osim toga, šta ukoliko bismo poželeli da u jednom trenutku smanjimo broj zvezdica kojima je oivičen naziv grada?

U ovoj funkciji, za to bi trebalo promeniti samo dve vrednosti, dok bi, bez funkcije, trebalo menjati kod onoliko puta koliko ima gradova.

**Tipovi funkcija**

Ono što je još zanimljivo u prethodnom primeru jeste da u okviru funkcije stars(), pozivamo još jednu funkciju strlen(), koja vraća dužinu nekog stringa. U našem slučaju, to je naziv grada. Funkcija strlen() nije funkcija koju smo ručno napisali, već je implementirana u PHP, kao njegov sastavni deo, iz čega zaključujemo da PHP ima dve vrste funkcija, **korisnički definisane** i **unutrašnje** (ugrađene) funkcije koje su deo PHP jezika.

Korisnički definisane funkcije su one koje sami kreiramo (poput funkcije **stars**) i one se kreiraju korišćenjem ključne reči function, kao u prethodnom i sledećem primeru:

|  |  |
| --- | --- |
| 1  2  3 | function add($num1,$num2){      return $num1 + $num2;  } |

Funkcija koju smo upravo napisali prihvata dva parametra i sabira njihove vrednosti. Nakon toga uz pomoć naredbe **return** funkcija vraća dobijeni rezultat na mesto poziva.

Funkcija može biti napisana bilo gde u kodu, ali je ustaljena praksa da se sve funkcije pišu na početku koda, pre ostalih naredbi. Takođe, funkcija može biti napisana i u posebnom fajlu. Tada je potrebno da se ovaj fajl uključi na svaku stranu koja će koristiti njegove funkcije.

Funkcije mogu biti pozvane iz bilo kog dela PHP koda. Nakon poziva funkcije, funkcija preuzima prosleđene parametre, ukoliko ih ima, izvršava određene naredbe i vraća dobijenu vrednost programu.

|  |  |
| --- | --- |
| 1  2  3  4 | function add($num1, $num2){   return $num1 + $num2;  }  echo "The sum of 5 and 2 is  " . add(5, 2); |

U gornjem primeru koda funkcija se prvo definiše, ali se poziva tek kasnije, u okviru echo naredbe. Prilikom poziva funkcije prosleđuju joj se prave vrednosti parametara (5 i 2). Funkcija preuzima ove vrednosti, smešta ih u promenljive $num1 i num2, obavlja zadate operacije (sabira ih) i dobijenu vrednost vraća echo naredbi.

Moguće je napraviti i funkciju koja može, a ne mora da primi parametre, a da pritom ne dođe do greške. U tu svrhu, postavljaju se **podrazumevane vrednosti parametara**:

|  |  |
| --- | --- |
| 1  2  3  4 | function add($num1 = 3, $num2 = 4){   return $num1 + $num2;  }  echo "Result is " . add(); |

Promenljive koje se koriste unutar tela funkcije nazivaju se **lokalne** promenljive. Njima ne može da se pristupa izvan funkcije. Ukoliko je, ipak, potrebno da se jednoj istoj promenljivoj pristupa i iz tela funkcije i izvan njega, ovu promenljivu treba deklarisati kao **globalnu**, upotrebom ključne reči *global*. Globalne promenljive postoje i izvan funkcija u kojima su definisane.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function format\_name( $first\_name ,$last\_name){      global $name;      $name = $first\_name." - ".$last\_name;  }  format\_name( "Peter", "Andersen" );  echo $name; |

Promenljiva $name je kreirana unutar funkcije, ali je upotrebom ključne reči global deklarisana kao globalna promenljiva, što znači da će postojati i izvan funkcije. Prilikom poziva funkcije, promenljiva $name biva kreirana i dodeljena joj je neka vrednost. Ovu vrednost je moguće koristiti kasnije u kodu (echo naredba).

Funkciji je moguće proslediti parametre bilo kog tipa. Prilikom definisanja funkcije unutar zagrada se navode parametri koje funkcija koristi. Prilikom poziva funkcije, potrebno je proslediti joj konkretne vrednosti za parametre koje očekuje. Ove vrednosti mogu biti: brojevi, tekst, promenljive, nizovi, pa čak i izrazi koji imaju neku vrednost. Pogledajte sledeću funkciju:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function add\_numbers($numbers){      for($i=0; $i < sizeof( $numbers ); $i++){          @$sum = $sum + $numbers[ $i ];      }      return $sum;  } |

Ova funkcija sabira brojeve koji su joj prosleđeni kao niz. Ukoliko se pri pozivu funkcije ne prosledi niz, doći će do greške. Zbog toga je uvek dobro proveriti da li su prosleđene vrednosti odgovarajućeg tipa. Na primer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | function add\_numbers($numbers = 0){      $sum = 0;      if(is\_array($numbers)){          for( $i=0; $i < sizeof( $numbers ); $i++){              $sum = $sum + $numbers[$i];          }          return $sum;      }else{          echo 'Error with function argument!';          exit();      }  }  echo add\_numbers(); |

Svaka funkcija očekuje određeni broj parametara. Ukoliko je, pri pozivu, prosleđeno manje vrednosti nego što funkcija zahteva, funkcija koristi vrednost NULL za one koje nedostaju. Ukoliko prosledite više vrednosti, funkcija ih ignoriše.

U PHP-u je moguće postaviti podrazumevane vrednosti za parametre, koje će se koristiti ukoliko se ne proslede sve vrednosti parametara prilikom poziva.

|  |  |
| --- | --- |
| 1  2  3  4  5 | function add\_2\_numbers( $num1=1, $num2=1 ){      $total = $num1 + $num2;      return $total;  }  echo add\_2\_numbers(10); |

Analizirajući ponašanje koda, iznad uočavamo da je kao vrednost parametra $num1 preuzet broj 10, koji je i prosleđen pri pozivu funkcije. Kako drugi parametar nije prosleđen, preuzeta je podrazumevana vrednost 1. Zbog ovakvog ponašanja funkcija vraća vrednost 11.

Kada prosledite neku promenljivu funkciji, u stvari se prosleđuje njena vrednost. Bilo kakva izmena koju izvršite nad prosleđenom promenljivom neće uticati na originalnu vrednost. Na primer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function add\_1($num1){      $num1 = $num1 + 1;  }  $orig\_num = 3;  add\_1($orig\_num);  echo $orig\_num; |

Prilikom poziva, funkciji *add\_1()* se prosleđuje vrednost sadržana u promenljivoj $orig\_num. Naredbe unutar funkcije ne utiču na samu promenljivu, pa će naredba echo ispitati broj 3, tj. prvobitnu vrednost promenljive *$orig\_num*.

U nekim slučajevima će biti potrebno da funkcija promeni vrednost neke promenljive kreirane izvan funkcije. Tada se umesto vrednosti promenljive prosleđuje njena referenca pomoću operatora **&** na sledeći način:

|  |  |
| --- | --- |
| 1  2  3 | function add\_1(&$num1){      $num1 = $num1 + 1;  } |

Prilikom poziva ovako definisane funkcije prosleđuje se referenca do promenljive (pokazivač) umesto vrednosti koja je smeštena u njoj. Sada će promene unutar tela funkcije uticati na originalnu vrednost. Naredbe:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function add\_1(&$num1){      $num1 = $num1 + 1;  }  $orig\_num = 3;  add\_1($orig\_num);  echo $orig\_num; |

Rezultat na strani će sada biti broj 4.

**Anonimous funkcije**

Ponekad funkciju treba definisati generički u samom kodu. U tom slučaju, funkcija neće imati ime, već će se samo aktivirati kroz kod. Ovakve funkcije nazivaju se još i anonimous funkcije.

|  |  |
| --- | --- |
| 1  2  3  4 | <?php  $myAninimousFunction = create\_function('$a,$b','return $a + $b;');  echo $ myAninimousFunction(2, 4);  ?> |

Parametri funkcije create\_function() predstavljaju niz, prvi član niza predstavlja njene argumente, dok je drugi član samo telo funkcije. Funkcija create\_function() je kada je komplikovanije telo funkcije vrlo usporeno, pa se iz tog razloga izbegava njeno korišćenje zbog usporavanja rada web stranice.

[Anonimus funkciju](https://www.link-elearning.com/linkdl/opisPojma.php?id=139102) takođe možemo definisati i na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4 | $f = function(){      echo "Hello";  };  $f(); |

**Dinamički poziv funkcije**

PHP omogućava da promenljivoj dodelimo naziv funkcije kao string, a nakon toga tu istu promenljivu koristimo za pozivanje funkcije. Pogledajmo primer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  function helloFunction()  {     echo "Hello<br />";  }  $function\_holder = "helloFunction";  $function\_holder();  ?> |

Dakle, funkcija je pri kreiranju dobila naziv „helloFunction”. Nakon toga je promenljivoj $function\_holder dodeljen string sledećeg sadržaja „helloFunction”, odnosno sadržaj koji je identičan nazivu funkcije. Na kraju, funkciju više ne moramo pozivati samo pomoću njenog imena, već i pomoću novokreirane promenljive: $function\_holder().

Funkciju je moguće kreirati i unutar tela već postojeće funkcije. Primer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | function func1(){      function func2(){          echo 'Test';      }      func2();  }  func1(); |

Rezultat na strani je: Test.

**Šta od ponuđenog nije osobenost funkcije?**

 Enkapsulacija

 Smanjenje redundantnosti koda

 Blokada upotrebe globalnih promenljivih

 Brzina kodiranja

**Šta vraća funkcija strlen()?**

 Dužinu nekog stringa

 Tip prosleđenog podatka

 Trenutno ulogovanog korisnika

 Ništa, funkcija ne postoji

**Vežba 1**

***Problem:***

Potrebno je napraviti funkciju koja prihvata string i vraća njegove karaktere u obrnutom redosledu (na primer, string „moj string” postaje “gnirts jom”).

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php  function str\_reverse($str){      $rez = "";      for($i=strlen($str)-1; $i>=0; $i--){          $rez.=$str[$i];      }      return $rez;  }  $text = "my string";  echo str\_reverse($text);  ?> |

Funkciju koja obavlja posao zahtevan postavkom zadatka nazvaćemo: str\_reverse. Kako funkcija treba da obradi string koji joj je prosleđen, postavićemo jedan argument: $str. Nakon završenog posla, funkcija će vratiti rezultat u vidu stringa. Mogli bismo pretpostaviti da će promenljiva koja će predstavljati rezultat rada naše funkcije u početku biti prazan string, a kasnije verovatno neki kompleksniji niz karaktera. Na osnovu navedenog pišemo sledeći kod:

|  |  |
| --- | --- |
| 1  2  3  4 | function str\_reverse($str){      $rez = "";      return $rez;  } |

PHP stingove može obrađivati poput nizova. To znači da je PHP u stanju da prebroji karaktere u stringu i da na osnovu pozicije dobavi odgovarajući karakter iz stringa. Ovakve oznake nazivaju se indeksi i u PHP-u indeksiranje podrazumevano počinje nulom.  
  
Kreirajmo sada promenljivu $text i izvršimo poziv funkcije:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function str\_reverse($str){      $rez = "";      return $rez;  }  $text = "my string";  echo str\_reverse($text); |

String koji funkcija obrađuje u našem slučaju je: „my string”. To znači da ako bismo želeli da kroz ovaj string prođemo upotrebom for petlje, moramo utvrditi broj karaktera u stringu da bismo za svaki string izvršili po jednu iteraciju. To možemo uraditi funkcijom strlen(). Ipak, indeksiranje počinje nulom, što znači da je najveći indeks našeg stringa za jedan manji od broja karaktera jer prebrojavanje vršimo od 1, a indeksiranje od 0. Zato postavljamo sledeću petlju, kojom ispisujemo karakter koji odgovara iteraciji:

|  |  |
| --- | --- |
| 1  2  3 | for($i=strlen($str)-1; $i>=0; $i--){          $rez.=$str[$i];      } |

Konačno je naš zadatak potpuno završen i funkcija spremna za korišćenje.

|  |
| --- |
| **Napomena** U primeru se radi sa nizovima kako bi se ispunila njegova svrha, te je ova vežba namenjena nešto naprednijim polaznicima. Nizovi će biti obrađeni u narednoj lekciji. Ukratko, oni su sredstvo da se u jednu promenljivu smesti više vrednosti. Vrednostima se pristupa pomoću kombinacije uglastih zagrada i pozicije vrednosti. |

**Vežba 2**

***Problem:***

Postoji sledeći ulaz u aplikaciju:

define( "SQUARE", 0 );  
define( "RECTANGLE", 1 );    
define( "CIRCLE", 2 );    
define( "PI", 3.14 );

Potrebno je napraviti funkciju koja će kao parametre prihvatati tip geometrijskog oblika (kvadrat, pravougaonik ili krug ) i vrednosti a i b. U zavisnosti od tipa geometrijskog oblika izračunava se njegova površina i vraća rezultat.

Potrebno je testirati funkciju.

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <?php  define("SQUARE", 0);  define("RECTANGLE", 1);  define("CIRCLE", 2);  define("PI", 3.14);  function area($type, $a, $b = 0){      $rez = 0;      switch($type)          {              case SQUARE:                  $rez = pow($a, 2);                  break;              case RECTANGLE:                  $rez = $a \* $b;                  break;              case CIRCLE:                  $rez = pow($a, 2) \* PI;                  break;          }      return $rez;  }  echo area(CIRCLE, 4);  ?> |

Ovaj zadatak je odličan primer kombinovanja znanja iz različitih, do sada obrađenih oblasti PHP-a. Najpre smo na već poznat način kreirali konstante koje će u nastavku programske logike biti korišćene, a nakon toga smo kreirali i pozvali funkciju. Funkcija radi sa tri parametra, od kojih samo poslednji poseduje podrazumevanu vrednost što ovaj parametar čini opcionim.  
  
Uz pomoć switch izjave proveravamo vrednost parametra $type i utvrđujemo kojoj konstanti odgovara. U zavisnosti od ove vrednosti, vršimo izračunavanje po sledećim formulama:

|  |  |
| --- | --- |
| 1  2  3 | $rez = pow($a, 2);  $rez = $a \* $b;  $rez = pow($a, 2) \* PI; |

Sada je naš rezultat kreiran i može biti vraćen na mesto poziva funkcije.