

Univerzitet u Beogradu
Fakultet organizacionih nauka
Laboratorija za elektronsko poslovanje

SKRIPTA IZ PREDMETA INTERNET TEHNOLOGIJE

-PHP-

SADRŽAJ

1	Uvod.....	4
1.1	PHP OKRUŽENJE.....	7
2	Osnove PHP koda.....	9
2.1	Osnovna sintaksa, promenljive i operatori.....	9
2.2	Osnovni tipovi u PHP-u.....	13
2.3	Opseg vazenja promenljive.....	18
2.4	Operatori i kontrolne strukture.....	20
2.4.1	Aritmeticki operatori	20
2.4.2	Inkrementiranje, dekrementiranje i operator dodeljivanja	20
2.4.3	Operatori poredjenja	21
2.4.4	If-else kontrola toka.....	22
2.4.5	Switch	23
2.4.6	Ternarni operator	24
2.4.7	Petlje	25
3	Funkcije.....	27
3.1	Ugrađene funkcije za rad sa varijablama	32
4	Stringovi.....	34
4.1	strtoupper()	36
4.2	strtolower().....	36
4.3	strlen()	36
4.4	strpos()	37
4.5	str_replace().....	37
4.6	substr()	38
4.7	str_split().....	39
5	Nizovi	40
5.1	array_key_exists().....	43
5.2	in_array()	44
5.3	array_push()	44
5.4	array_pop()	46
5.5	unset()	47
5.6	Sortiranje nizova	48
5.6.1	sort()	49
5.6.2	asort()	49
5.6.3	ksort()	50
5.7	count()	51
5.8	Višedimenzionalni nizovi	52
6	Izdvojeni koncepti i mogućnosti PHP	57
6.1	Date	57

6.2	Include i Require	58
6.3	PHP napredne funkcije za rukovanje fajlovima	59
6.4	Regularni izrazi	60
6.5	Superglobals.....	62
6.6	PHP napredne funkcije za rukovanje cookies	64
6.6.1	Slanje header-a	66
6.7	PHP napredne funkcije za sesijama.....	67
6.8	PHP napredne funkcije za upravljanje greškama	73
7	Uvod u forme.....	78
7.1	O formama uopšteno	78
7.2	Metode slanja i prihvatanja podataka	79
7.3	Provera metode pristupa dokumentu / skripti	83
7.4	Forme i srpski znakovi	83
7.5	Prikaz i obrada unutar jednog dokumenta.....	84
7.6	Elementi za unos podataka	86
7.7	Text box polje	86
7.8	Text area	86
7.9	Hidden polje.....	87
7.10	Checkbox	88
7.11	Upload fajla	89
8	Dinamički linkovi.....	91

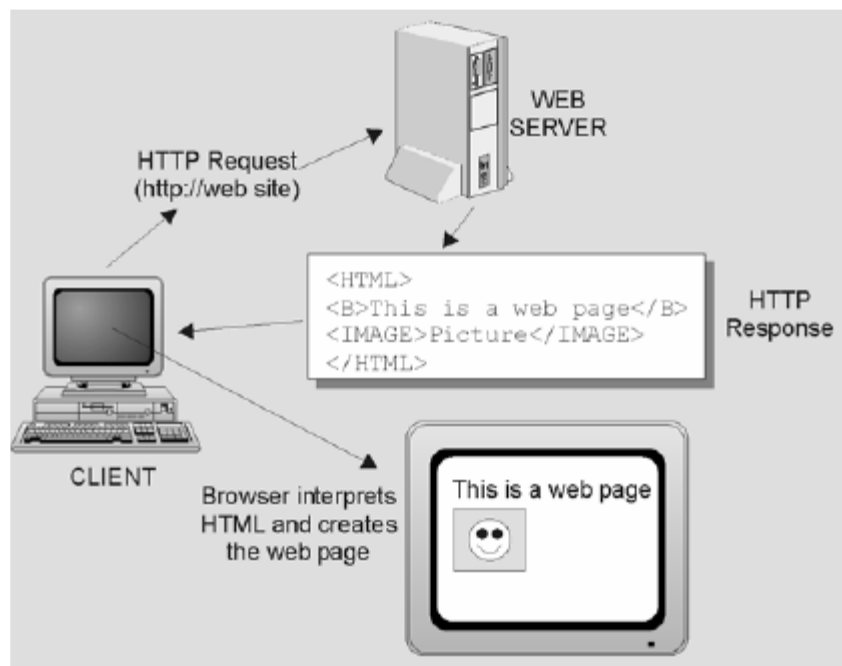
1 Uvod

Server - side skripting je veb server tehnologija koja omogućava da se korisnički zahtevi obrađuju pomoću skripti koje se izvršavaju na serverskoj strani kako bi se generisale dinamičke stranice. Najčešće se koristi da bi se interaktivne veb stranice povezale sa bazama podataka, radi identifikacije korisnika, ažuriranja sadržaja i sl. Razlika između skriptovanja na serverskoj i korisničkoj strani je u tome što se kod korisničke strane skripte izvršavaju u veb browser-u korisnika. Glavni predstavnik client-side grupe jezika je JavaScript. Kod pisan u JavaScriptu je obično umetnut u HTML stranicu i izvršava se tek u klijentovom pretraživaču.

Osnovna razlika između PHP i HTML stranica je u načinu na koji veb server upravlja njima.

Kada veb serveru stigne zahtev za HTML stranicom, veb server preduzima sledeće operacije:

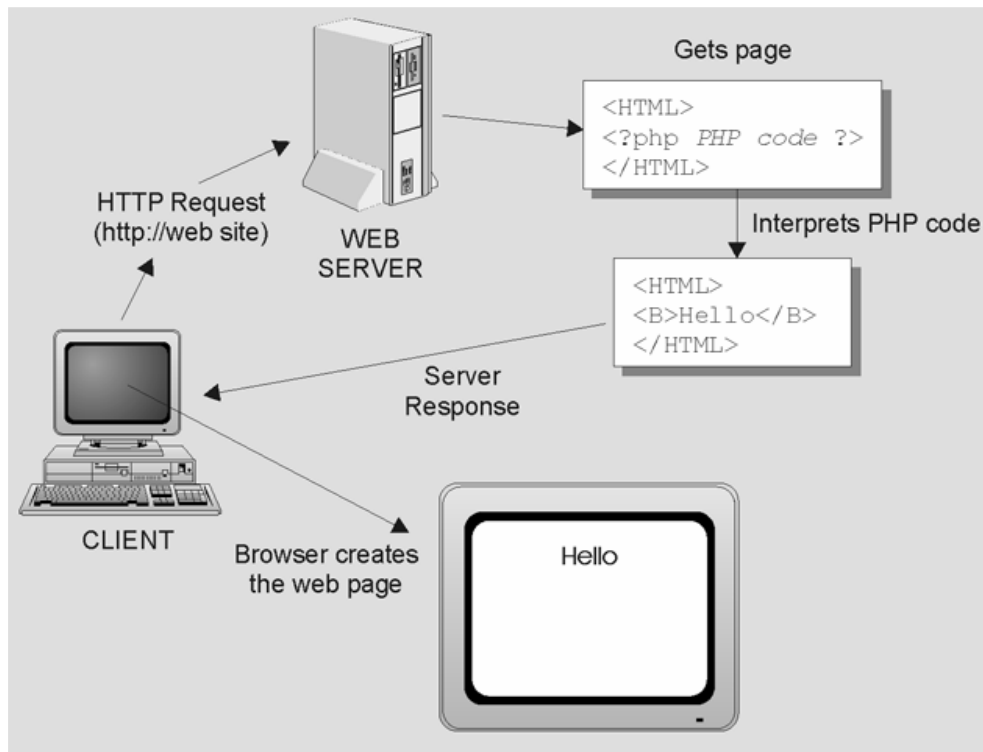
- Čita zahtev
- Pronalazi stranicu na serveru
- Šalje stranicu ka browser-u



Slika 1 Obrada zahteva za HTML stranicom

Kada veb server dobije zahtev za PHP stranicom, umesto slanja statičke HTML stranice, server preduzima određene akcije u skladu sa PHP kodom:

- Čita zahtev
- Pronalazi stranicu na serveru
- Obradjuje sve instrukcije koje se nalaze u PHP fajlu, i na taj način kreira ili modifikuje stranicu
- Šalje novu stranicu ka browser-u



Slika 2 Obrada zahteva za HTML stranicom

Server-side obrada omogućava:

- Smanjuje količinu saobraćaja između server i klijenta
- Otklanja problem kompatibilnosti browser-a
- Obezbeđuje različite tipove informacija za klijente
- Poboljšava sigurnost aplikacije

Možda bi bilo najbolje malo detaljnije razmotriti razlike između njih na jednom trivijalnom primeru. Neka je na primer, potrebno napisati skriptu koja prikazuje tačno vreme negde na stranici. Prvo pitanje koje treba postaviti je 'Koje je vreme tačno vreme?'. Ono na serveru ili ono na klijentovom računaru? Ove dve skripte neće nikada prikazati isti rezultat.

Primer – prikaz vremena kod klijenta

```

<html>
  <head>
    <script language="JavaScript">
      var v = new Date()
      document.write(v.getHours())
      document.write(":")
      document.write(v.getMinutes())
      document.write(":")
      document.write(v.getSeconds())
    </script>
  </head>
  <body>
  </body>
</html>
  
```

Primer – prikaz serverskog vremena

```
<?php
$str_vreme= date("H:i:s");
echo $str_vreme;
?>
```

PHP

PHP je open source jezik koji se koristi za razvoj server-side aplikacija, kao i dinamičkog Web sadržaja. PHP dozvoljava interakciju sa velikim brojem relacionih baza podataka kao što su MySQL, Oracle, IBM D2, Microsoft SQL Server, PostgreSQL i SQLite. Koristi se za upravljanje dinamičkim sadržajem, praćenjem sesija, zapravo PHP vam omogućava da uz pomoć njega napravite čitave e-commerce sajtove. PHP radi na većini operativnih sistema današnjice, kao što su UNIX, Linux, Windows i Mac OS i može da interaguje sa većinom Web servera.

Rasmus Lerdorf je izbacio prvu verziju PHP-a davne 1994 godine. Personal Home Page Tools je predstavio PHP 1995. godine, a dve godine kasnije su razvoj nastavila dva Izraelska programera. Od 1999. godine se zasniva na Zend engine-u. PHP je prvo bio akronim za Personal Home Page Tools, da bi kasnije promenio naziv u Hypertext Preprocessor.

PHP je stekao popularnost zbog svoje jednostavnosti i sintakse nasleđene iz programskog jezika C. Tokom vremena jezik se proširivao i sticao mogućnosti za objektno orijentisano programiranje, naročito od verzije 5.0. Nalikuje jeziku C++ u smislu da dozvoljava i čisto-proceduralno programiranje ali omogućava i korišćenje klasa i drugih koncepata objektno orijentisanog programiranja (nasleđivanje, apstraktne metode, interfejsi itd.).

Danas, PHP je instaliran na više od 20 miliona sajtova i preko million web servera. Poslednja realizovana stabilna verzija je 7.2 iz novembra 2017.god.

1.1 PHP okruženje

Najjednostavnija varijanta za instaliranje PHP-a jeste preko XAMPP-a. XAMPP omogućava da se u jednom koraku instaliraju i PHP kao i veb server i neke dodatne softvere koji mogu biti od koristi prilikom razvoja aplikacije.

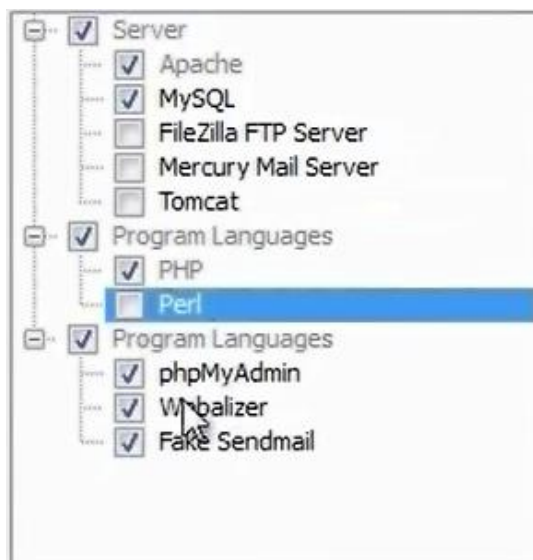
XAMPP predstavlja skraćenicu za:

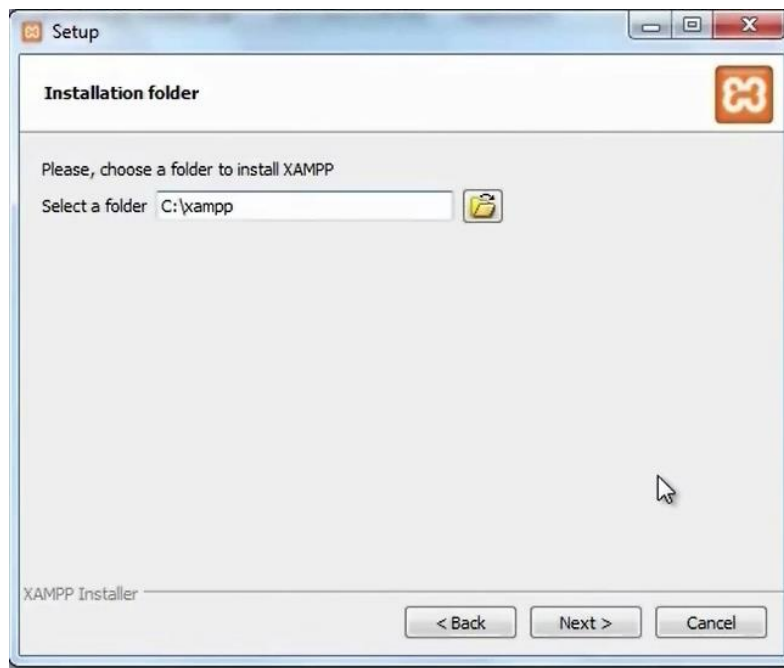
- X - Da ne zavisi od operativnog sistema
- A - Apache
- M - MySQL
- P - PHP
- P - Perl

Osnovna ideja koja stoji iza XAMPP-a jeste da postoji jedan instaler, i da se uz njegovu pomoć instaliraju sve neophodni softveri i paketi za razvoj veb aplikacije. Dodatno, pored toga što se dobijaju svi instalirani softveri, oni su već konfigurisani za korišćenje.

Na adresi <https://www.apachefriends.org/download.html> treba preuzeti XAMPP za odgovarajući operativni sistem, i nakon toga jednostavno pokrenete XAMPP, koji će u pozadini već biti pripremljen za korišćenje.

Proces instaliranja se svodi na klasičan način (next, next, next..), pri čemu se može odrediti lokacija gde će XAMPP da bude instaliran na lokalnoj mašini. (Pogledati slike u nastavku)





URL lokalnog veb servera je <http://localhost> ili 127.0.0.1

2 Osnove PHP koda

2.1 Osnovna sintaksa, promenljive i operatori

U PHP fajlu, blok koji je okružen jezičkim strukturama `<?php i ?>` se smatra PHP kodom i izvršava se, a ostatak van tih znakova se smatra tekстом koji jednostavno treba da se ispiše na standardni izlaz, bez interpretiranja. Ako se PHP kod izvršava na serveru koji podržava skraćenice, PHP blok može početi i sa `<?>`, a završiti se sa `?>` tagom. PHP kod se može napisati bilo gde u okviru `<body>` taga HTML dokumenta. U nekim slučajevima, PHP, kod se piše u okviru `<head>` taga.

Sa regularnim .html fajlovima, HTTP server (npr. Apache) samo prenosi sadržaj stranice u browser. Server ne pokušava da razume ili obrađuje HTML fajl – to je posao browser-a. Fajlovi sa .php ekstenzijom se “skeniraju” i obrađuje se kod.

PHP fajl sadži standardne HTML tagove i PHP skript kod. Sledi primer dva prosta PHP koda koji na ekranu browser – a ispisuju “Zdravo svete!”.

```
<html>
<head> <title> Primer outputa Zdravo svete </title> </head>
<body>
<?php
echo "Zdravo svete!";
?>
</body>
</html>
```

```
-----
<html>
<head> <title> Primer outputa Zdravo svete </title> </head>
<body>
<?php
print ( 'Zdravo svete' );
?>
</body>
</html>
```

Kao što iz primera vidi, svaka linija PHP koda mora da se završi tačkom – zarez. Tačka – zarez se koristi da bi se jedan set instrukcija razdvojio od drugog seta. Postoje dve osnovne komande koje služe za prikazivanje teksta na browser – u: **echo** i **print**. Kao što je već rečeno, PHP kod se nalazi u okviru HTML koda i izvršava se na serveru. Tek kada se PHP kod izvrši na serveru, output se šalje do browser - a klijenta koji izvršava HTML kod i prikazuje PHP kod koji je već izvršen na serveru, tj. prikazuje i rezultat PHP koda i rezultat HTML koda. Ako se na primer u PHP skripti koristi komanda **echo** “
”, dešava se sledeće:

- Veb server izvršava echo “
” kod.
- Rezultat tog koda će biti
.
- Zatim, Veb server šalje browser – u klijenta izvršeni kod.

- Kada browser klijenta vidi `
`, kao rezultat PHP koda koji treba da prikaže, razumeće da je rezultat PHP koda ustvari HTML kod koji browser treba da izvrši. Browser će izvršiti HTML kod i konkretno u ovom primeru, kursor će preći u sledeći red.

Pomoću echo naredbe u PHP skriptama, mogu se pisati bilo koji HTML kodovi koje browser treba da izvrši.

Ukoliko se pogleda page source stranice (Desni klik na browser, pa opcija “View page source”) može se uočiti da se nigde ne vidi php kod koji je otkucan, a to je upravo iz razloga što se php izvršava na serverskoj strani, a browser samo prihvata i prikazuje odgovor sa servera.

Sledeći kod:

```
<?php
    phpinfo();
?>
```

ispisuje stranicu sa velikim brojem informacija o instaliranim softverima kao sto su php verziji, informacije o sistemu,, build date, konfiguracija Apache servera, kao i koje ekstenzije za php su instalirane i mnoge druge korisne informacije.

Jedna od uloga “phpinfo()” koji je ugrađen u PHP-u jeste upravo to, da predstavlja “Hello World” primer za testiranje okruženja u kojem PHP radi.

Još jedna od stvari koju je veoma važno napomenuti jeste to da PHP ima veoma dobru dokumentaciju koja je svima dostupna: <http://php.net/manual>.

Komentari

Najčešće korišćen, za jednu liniju koda:

```
<?php
    // echo 'Hello world';
?>
```

Za veći broj linija koda treba koristiti:

```
<?php
    /*
        echo 'Hello world';
        echo 'Hello world';
        echo 'Hello world';
        echo 'Hello world';
    */
?>
```

Ili se mogu koristiti Shell Style komentari:

```
<?php
    # echo 'Hello world';
?>
```

Kada je reč o varijablama (promenljive) onda u tom slučaju PHP treba posmatrati kao case sensitivity jezik:

```
<?php
    $caseSensitive = 'Nismo';
    $CaseSensitive = 'isti.';

    echo $caseSensitive;
    echo $CaseSensitive;
```

U prethodnom primeru nazivi promenljivih su različiti, onda ih i PHP tako posmatra i pretpostavlja da one sadrže dve različite vrednosti.

Međutim ukoliko bi ova se ova priča posmatra na primeru klase u tom slučaju PHP nije case sensitivity. Primer:

```
<?php

class Fakultet
{
    ...
}

$c1 = new Fakultet;
$c2 = new FAKULTET;
?>
```

PHP klase nisu case sensitive, što znači da nije bitno da li se naziv klase napiše velikim ili malim slovima, PHP će to posmatrati kao istu klasu.

Isto važi i za PHP funkcije, koje takođe nisu case sensitive, već ih PHP posmatra na isti način kao što to radi i sa klasama. Primer:

```
<?php

function nazivFakulteta()
{
    ...
}

nazivfakulteta();
NazivFakulteta();
```

```
NAZIVFAKULTETA();  
?>
```

2.2 Osnovni tipovi u PHP-u

Tip - Integer

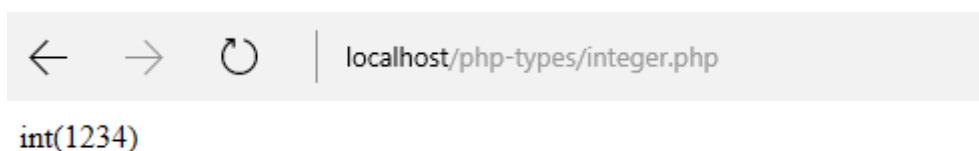
Postoji više tipova integer-a koje PHP dozvoljava. Sve varijable u PHP-u počinju znakom “\$”.

Kod:

```
<?php
$standardniInteger = 1234;

var_dump($standardniInteger);
?>
```

Ukoliko bi pokrenuli ovaj naš kod rezultat u browser-u bi bio sledeći:

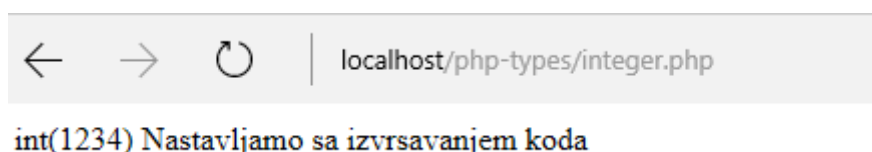


Sada se postavlja pitanje šta je “**var_dump**”. Predstavlja funkciju koja je ugrađena u php i koja kao parametar prima neku promenljivu i ispisuje vrednost, to jest ispisuje kojeg tipa je ta promenljiva kao i njenu vrednost. Ova funkcija zapravo može da bude vrlo korisna u slučajevima kada niste sigurni šta ćete da dobijete kao rezultat izvršavanjem nekog dela koda. Tako da se često koristi u svrhe testiranja i debugovanja, ali samo u slučaju nekih sitnih stvari. Važno je napomenuti da funkcija `var_dump` ispiše to što je zadato i nastavi sa izvršavanjem koda, međutim ukoliko je potrebno da se prekine sa izvršavanjem koda (u slučajevima testiranja ili traženja greške), onda se može iskoristiti funkcija “**die**” u kombinaciji sa funkcijom `var_dump (die(var_dump()))`. Primeri:

```
<?php
$standardniInteger = 1234;

var_dump($standardniInteger);
echo ' Nastavljamo sa izvorsavanjem koda ';
?>
```

Rezultat:

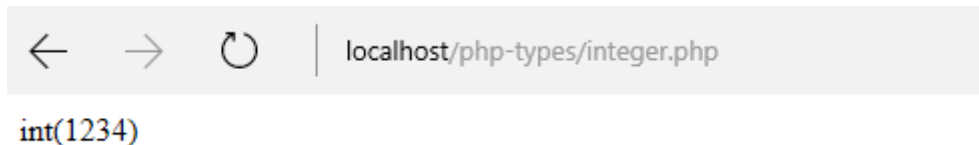


```
<?php

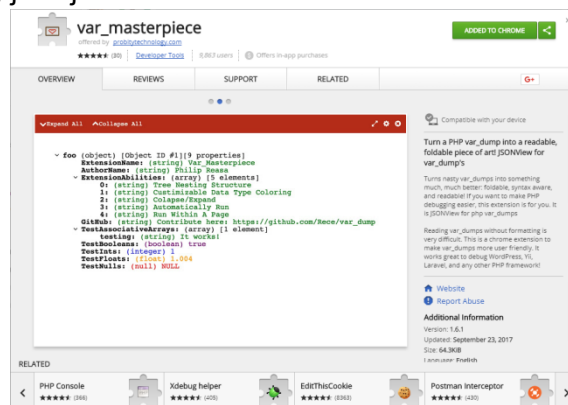
$standardniInteger = 1234;

die(var_dump($standardniInteger));
echo ' Nastavljamo sa izvršavanjem koda ';
?>
```

Rezultat:



U Google Chrome postoje brojne ekstenzije koje mogu biti od velike koristi prilikom programiranja, kada je reč o `var_dump` funkciji ekstenzija `var_masterpiece` formatira sadržaj tako da on bude čitljiviji, jasniji i kako biste se lakše snalazili.



Takođe pored standardnih integer-a PHP omogućava da se korsite i oktalni, heksadecimalni i binarni brojevi.

```
<?php

$oktalniBroj = 01234; // Svi oktalni brojevi pocinju sa '0'

$heksadecimalniBroj = 0xABC; // Svi heksadecimalni brojevi pocinju sa '0x'

$binarniBroj = 0b1000; // Svi binarni brojevi pocinju sa '0b'

var_dump($oktalniBroj);
echo "<br/>";
var_dump($heksadecimalniBroj);
echo "<br/>";
var_dump($binarniBroj);
?>
```

Rezultat:

```
← → ↻ | localhost/php-types/integer.php  
int(668)  
int(2748)  
int(8)
```

Tip - Decimal/ Floating Point

Pored tipa Integer postoji i tip float koji se koristi za decimalne brojeve. Jednostavan primer korišćenja ovog tipa:

```
<?php  
    $floatBroj = 1.234;  
  
    var_dump($floatBroj);  
?>
```

Tip - Boolean

Tip boolean je veoma jednostavan za korišćenje, ima samo dve dopustive vrednosti, to su "true" ukoliko je neki uslov zadovoljen ili ukoliko je nesto ispravno, ili "false" u slučaju da nešto nije ispunjeno ili zadovoljeno. Primer:

```
<?php  
    $bool = true;  
    var_dump($bool);  
?>
```

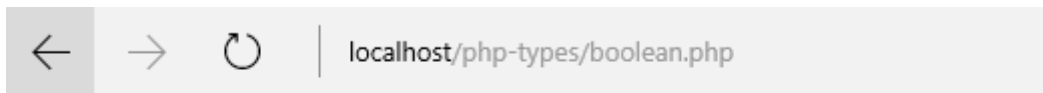
Rezultat:

```
← → ↻ | localhost/php-types/boolean.php  
bool(true)
```

Interesantno je to da se za svaka vrednost promenljive indirektno može predstaviti preko boolean-a, to jest da vidi se da je true/false u zavisnosti od toga da li neka promenljiva ima vrednost ili nema. Primer:

```
<?php  
    $imaVrednost = 1234;  
    var_dump((bool)$imaVrednost);  
?>
```

Rezultat:



`bool(true)`

Operacija **(bool)\$imaVrednost** predstavlja operaciju kastovanja, koja označava da se jedna promenljiva nekog tipa pretvara u drugu koja je nekog drugog tipa. (Napomena: kastovanje nije uvek moguće izvršiti, nisu svi tipovi kompatibilni, takođe prilikom kastovanja nekih tipova neophodno je da se stavi eksplicitno kastovanje kao u ovom slučaju dok kod drugih nije potrebno to raditi već sam kompajler je sposoban da to odradi za nas.).

Ukoliko je vrednost neke promenljive prazan string, nula ('0') ili ukoliko je neki niz prazan, u tim slučajevima vrednost jedne ovakve operacije, to jest boolean vrednost neke promenljive će biti **false**.

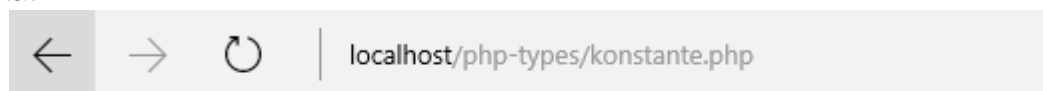
Konstante

Konstanta se definiše pomoću funkcije "define" koja kao prvi parametar prima naziv konstante koji se nalazi u okviru jednostrukih navodnika, a drugi parametar predstavlja vrijednost te konstante. Neko pravilo jeste da u PHP-u se konstanta definiše velikim slovima i da svaka reč u nazivu konstante bude odvojena underscore-om ("_"). Pogledajmo primer:

```
<?php
    define('NOVA_KONSTANTA', "Ovo je nasa nova konstanta");

    echo NOVA_KONSTANTA;
?>
```

Rezultat:



`Ovo je nasa nova konstanta`

Jedna od važnih osobina konstanti zbog kojih se one i koriste jeste to što se njima može pristupiti iz bilo kog dela koda. Što naravno može biti i dobra i loša stvar, zavisno od konkretnog problema i logike.

Na samom kraju priče oko osnovnih tipova u PHP-u potrebno je videti na koji način se može odrediti tip neke promenljive. Primer:

```
<?php
    define('KONSTANTA', "Ja sam konstanta");
    $integer = 123;
    $float = 0.45;
    $string = "Ja sam string";
    $bool = true;
?>
```

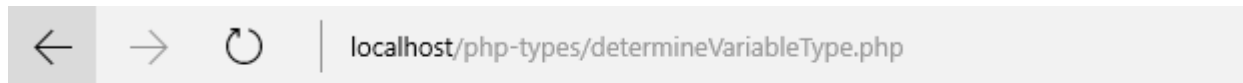

U PHP-u se nalaze neke ugrađene funkcije *is_int*, *is_float*, *is_string*, *is_bool* i *defined*.

Primer:

```
<?php
    define('KONSTANTA', "Ja sam konstanta");
    $integer = 123;
    $float = 0.45;
    $string = "Ja sam string";
    $bool = true;

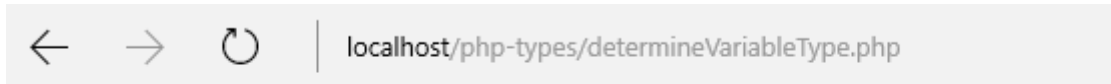
    echo is_int($integer);
    echo is_float($float);
    echo is_string($string);
    echo is_bool($bool);
    echo defined('KONSTANTA');
?>
```

Rezultat:



11111

Vrednost "1" označava da je izraz tačan. Ukoliko bi se izmenile funkcije *is_float* i *defined* na: *is_float(\$integer)* i *defined('NOVA_KONSTANTA')* rezultat bi bio sledeći:



111

Dakle, ukoliko je izraz tačan ili ispravan da će gore navedene funkcije vratiti vrednost "1" u suprotnom neće vratiti ništa

Dva specijalna tipa

- NULL - promenljive kojima nije dodeljena vrednost, koje su nedefinisane ili kojima je izričito dodeljena vrednost NULL.
- Resurs - neke ugrađene funkcije (npr. za rad sa bazama podataka) vraćaju promenljive tipa resurs, koje predstavljaju spoljne resurse (veza sa bazom podataka).

Konverzija tipa podataka

Operator za konverziju omogućava privremenu promenu tipa promenljive ili vrednosti.

Postupak se odvija identično kao u C-u. Primer:

```
<?php
    $broj = 10; //integer
    $realbroj = (double)$broj; //double
?>
```

2.3 Opseg vazenja promenljive

U PHP-u postoji nekoliko mogućih opsega važenja:

- Ugrađene globalne promenljive koje su vidljive u celom fajlu.
- Konstante koje imaju globalnu vidljivost i mogu da se koriste izvan funkcija.
- Globalne promenljive, deklarisanе u skriptu, vidljive su svuda u skriptu, ali ne i unutar funkcija.
- Promenljive koje napravite unutar funkcije, a deklarirate kao statičke promenljive, ne vide se izvan funkcija, ali čuvaju svoju vrednost između dva izvršavanja funkcije.
- Promenljive koje napravite unutar funkcije lokalne su za funkciju i prestaju da postoje kada prestane izvršavanje funkcije.

Od verzije PHP-a 4.1 pa nadalje, za neke posebne promenljive, važe drugačija pravila za opseg važenja. To su superglobalne promenljive, koje se svuda vide, i u funkcijama i van njih.

Primeri:

```
<?php

$a = 1; /* globalno vidljiva promenljiva u čitavom fajlu*/

function test()
{
    echo $b; /* lokalna promenljiva, vidljiva samo unutar funkcije*/
}

?>
```

```
<?php

$broj1 = 1;
$broj2 = 2;

function saberi()
{
    global $broj1, $broj2;
    $broj2 = $broj1 + $broj2;
}

?>
```

Jos jedan način da se pristupi globalnim promenljivama jeste da se koristi predefiniati niz u php-u koji se naziva: "\$GLOBALS[]"

```
<?php
    $broj1 = 1;
    $broj2 = 2;

    function saberi()
    {
        $GLOBALS['broj2'] = $GLOBALS['broj1'] + $GLOBALS['broj2'];
    }
?>
```

Pored pomenutih promenljivih često korišćene jesu i statičke promenljive. One takođe postoje samo unutar opsega neke funkcije ili fajla ali ne gube svoju vrednost kada izvršavanje programa napusti opseg u kojem su one definisane. Primer:

```
<?php

function ispisi()
{
    $a = 0;
    echo $a;
    $a++;
}
?>
```

U ovom slučaju stalno bi se ispisivala nula pri pozivu ove funkcije iz razloga što promenljiva nestaje iz memorije čim se ova funkcija završi i zato se pri svakom pozivu ove funkcije kreira nova promenljiva “a”.

Da bi se to ispravilo treba koristiti statičku promenljivu, to jest:

```
<?php
function ispisi()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

U ovom slučaju promenljiva “a” pošto je statička samo će šrvi put da se inicijalizuje, to jest samo će tada da se upiše u memoriju i ostaje u njoj sve do kraja rada programa. Na taj način pri svakom novom pozivu ove funkcije ispisaće za jedan broj veći od prethodnog puta.

2.4 Operatori i kontrolne strukture

2.4.1 Aritmetički operatori

Kada je rec o aritmetičkim operatorima kao i u skoro svakom programskog jeziku standardni aritmetički operatori jesu sabiranje ('+'), oduzimanje ('-'), deljenje ('/') i množenje ('*'). Pored njih postoje još dva koja mogu biti od koristi i često se koriste. Prvi jeste za određivanje ostatka prilikom deljenja ('%'), a drugi za stepenovanje nekog broja ('**'). Primer:

```
<?php
    echo (15 + 4) . '<br/>';
    echo (15 - 4) . '<br/>';
    echo (15 * 4) . '<br/>';
    echo (15 / 4) . '<br/>';
    echo (15 % 4) . '<br/>';
    echo (15 ** 4) . '<br/>';
?>
```

U nastavku su dati rezultati:

19
11
60
3.75
3
50625

2.4.2 Inkrementiranje, dekrementiranje i operator dodeljivanja

Dva veoma bitna i cesto koriscena operatora jesu operatori za inkrementiranje(uvecavanje vrednosti za jedan) i dekrementiranje (smanjivanje vrednosti za jedan). Ova dva operatora se izgledaju: '++' i '--' respektivno. Pišu se neposredno pre ili posle neke promenljive, sa tim sto postoji razlika da li smo ih stavili ispred ili iza neke promenljive, ukoliko stoje pre neke promenljive oni će prvo da izmene vrednost te promenljive a onda ce da urade nesto sa tom promenljivom. Međutim, ukoliko stoje nakon promenljive prvo će da iskoriste staru vrednost promenljive, pa tek onda da promene vrednost promenljive. Lakše ćemo to razumeti na primeru:

```
<?php
    $promenljiva = 5;

    echo ++$promenljiva . '<br/>';

    echo $promenljiva++ . '<br/>';

    echo $promenljiva . '<br/>';
```

```
?>
```

Rezultat:

```
6
6
7
```

Operator dodeljivanja omogućuje samo skraćeni zapis nekog dela koda

```
<?php
    $promenljiva = 5;

    $promenljiva = $promenljiva + 2;

    echo $promenljiva . '<br/>';

    $promenljiva += 3;

    echo $promenljiva . '<br/>';
?>
```

Rezultat:

```
7
10
```

Umesto znaka '+' može se korsititi neki drugi gore navedeni simbol (to jest: -, *, /) ili ukoliko je reč o stringovima, to jest spajanju stringova može se koristiti i operator '===' ukoliko je potrebno da spojiti dva stringa.

2.4.3 Operatori poredjenja

Ovi operatori se koriste kada je potrebno uporediti dve promenljive, to jest kada se upoređuju vrednosti i tipovi. Kada se upoređuje da li dve promenljive imaju istu vrednost koristimo operator "==". Ukoliko se upoređuje to da li su i iste vrednosti i istog tipa koristi se operator "===", primer:

```
<?php
    var_dump(7 == 7);
    var_dump(7 === 7);
    var_dump(7 == "7");
    var_dump(7 === "7");
?>
```

Rezultat je sledeći:

```
bool(true)
bool(true)
bool(true)
bool(false)
```

Ukoliko je potrebno proveriti da li su neke promenljive različite koriste se operatori "!=" i "!==" koji se ponašaju na isti način kao i prethodna dva samo u negaciji.

Pored provere da li su dve promenljive jednake postoje i operatore za proveru da li je neka promenljiva manja, veća, manja jednaka ili veća jednaka od neke druge promenljive. Primer:

```
<?php
    var_dump(8 > 8);
    var_dump(8 >= 8);
    var_dump(4 < 2);
    var_dump(4 <= 3);
?>
```

Rezultat:

```
bool(false)
bool(true)
bool(false)
bool(false)
```

2.4.4 If-else kontrola toka

Pomoću if-else kontrole kontroliše se u kom smeru ili koji deo koda će se izvršiti u zavisnosti od toga da li je neki uslov ispunjen ili ne. Primer:

```
<?php
$array = ['Filip', 'Marko', 'Edis'];

if(count($array) > 0)
{
    echo "U nizu se nalaze " . count($array) . " elemenata.";
}
else
{
    echo "Niz je prazan.";
}
?>
```

Rezultat bi bio sledeći:

U nizu se nalaze 3 elemenata.

If-else odlučuje da ili se ide u jednom smeru ili u drugom, međutim ukoliko treba ići u trećem smeru koristi se “**elseif**”. Primer:

```
<?php

$array = ['Filip', 'Marko', 'Edis'];
$count = count($array);

if($count == 1)
{
    echo "U nizu se nalazi jedan elemenat.";
}
elseif($count > 1)
{
    echo "U nizu se nalazi vise elemenata.";
}
else
{
    echo "Niz je prazan.";
}

?>
```

Rezultat je sledeci:

U nizu se nalazi vise elemenata.

2.4.5 Switch

Primer:

```
<?php

$array = ['Filip', 'Marko', 'Edis'];
$count = count($array);

switch($count)
{
    case 1:
        echo 'U nizu se nalazi jedan element.';
        break;
    case 2:
        echo 'U nizu se nalaze dva elementa.';
        break;
    case 3:
        echo 'U nizu se nalaze tri elementa.';
        break;
    default:
```

```

        echo 'Niz je prazan.';
    }
?>

```

Rezultat:

U nizu se nalaze tri elementa.

Vazna napomena jeste da u okviru svakog slucaja stoji “**break**”, u suprotnom moze se desiti da udjemo u vise slucaja (case-ova) sto nikako ne zelimo. Pogledajmo na primeru:

```

<?php
    $array = ['Filip'];
    $count = count($array);

    switch($count)
    {
        case 1:
            echo 'U nizu se nalazi jedan element.';
        case 2:
            echo 'U nizu se nalaze dva elementa.';
        case 3:
            echo 'U nizu se nalaze tri elementa.';
        default:
            echo 'Niz je prazan.';
    }
?>

```

Rezultat:

U nizu se nalazi jedan element.
 U nizu se nalaze dva elementa.
 U nizu se nalaze tri elementa.
 Niz je prazan.

2.4.6 Ternarni operator

Ternarni operator da se if-else kontrola toka kraće, lepše i jednostavnije zapiše. Sintaksa je sledeća: (**uslov**) ? **AkoJeZadovoljen** : **AkoNijeZadovoljen**;

Primer:

```

<?php

    $array = ['Filip', 'Marko', 'Edis'];
    // Prvi nacin
    /*if(count($array) > 0)
    {
        echo "Niz ima: " . count($array) . " elemenata.";
    }

```



```

    }
    else
    {
        echo "Niz je prazan.";
    }*/
    // Drugi nacin
    echo (count($array) > 0) ? "Niz ima: ".count($array)." elemenata."
        : "Niz je prazan.";
?>

```

Rezultat:

Niz ima: 3 elemenata.

2.4.7 Petlje

While-petlja

While petlja služi za izvršavanje istog segmenta koda sve dok je neki uslov zadovoljen. Kad taj uslov više nije zadovoljen while petlja prestaje sa izvršavanjem tog određenog dela koda i nastavlja se sa izvršavanjem ostatka koda koji se nalazi nakon while petlje. While petlja se piše na sledeći način:

```

while(uslov)
{
    // kod koji treba da se izvršava sve dok je uslov zadovoljen
}

```

```

<?php
    $i = 0;
    while($i < 5)
    {
        echo 'Ova poruka ce se ispisati pet puta.';
        $i++;
    }
?>

```

Rezultat će biti:

Ova poruka ce se ispisati pet puta.
 Ova poruka ce se ispisati pet puta.
 Ova poruka ce se ispisati pet puta.
 Ova poruka ce se ispisati pet puta.
 Ova poruka ce se ispisati pet puta.

For - petlja

Kao i sve ostale petlje i for petlja služi za istu svrhu a to je da određeni dio koda izvršimo više puta, to jest sve dok je neki uslov zadovoljen

```
for(uslov1; uslov2; uslov3)
{
}
```

Tako da bi prikaz gorenapisane while petlje preko for petlje izgledao:

```
<?php
    for($i = 0; $i < 5; $i++)
    {
        echo 'Ova poruka ce se ispisati pet puta.';
    }
?>
```

Postoje tri načina da se prekine izvršavanje bloka koda:

- break - iskakanje iz petlje
- continue - iskakanje na novu iteraciju petlje
- exit - prekidanje izvršavanja celog PHP skripta

Alternativan način za pisanje kontrolnih struktura:

- Početna vitičasta zagrada ({) zamenjuje se dvotačkom (:)
- Završna vitičasta zagrada (}) zamenjuje se novom ključnom rečju, koja će biti endif, endswitch, endwhile, endfor ili endforeach, u zavisnosti koja upravljačka struktura je u pitanju.

Pogledajmo to na primeru:

```
<?php
    if ($stanje == 0) :
        echo "Nemate dovoljno novca.";
        exit;
    endif;
?>
```

3 Funkcije

Funkcija predstavlja deo koda koji je zadužen za obavljanje određenog zadatka. Razlog zbog kojeg se koriste funkcije jeste da određeni deo koda jednom napisan može da se poziva kada god je potreban u programu, na taj način se optimizuje kod.

Sintaksa koja se koristi prilikom pisanja PHP funkcija je sledeća:

```
<?php
    function nazivFunkcije($prviParametar, $drugiParametar, ...)
    {
        Kod koji funkcija treba da izvrši..
    }
?>
```

Važno je još napomenuti da u PHP-u za razliku od nekih drugih programskih jezika prilikom pisanja funkcija nije potrebno napomenuti tip povratne vrednosti funkcije (Napomena: Pogledati šta znači “weakly typed language”, naročito kada je php u pitanju).

Primer funkcije:

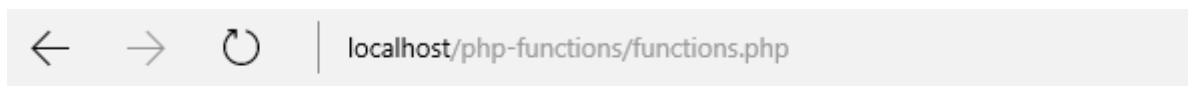
```
<?php
    function porukaDobrodoslice()
    {
        echo 'Dobrodosli!';
    }
?>
```

Poziv ove funkcije:

```
<?php
    function porukaDobrodoslice()
    {
        echo 'Dobrodosli!';
    }

    porukaDobrodoslice();
?>
```

Rezultat:



Dobrodosli!

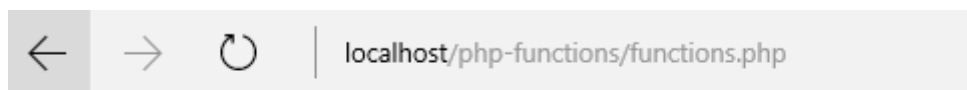
Funkcija sa parametrima:

```
<?php
    $imeKorisnika = 'Filip';
    $prezimeKorisnika = 'Furtula';

    function porukaDobrodoslice($ime, $prezime)
    {
        echo 'Dobrodosli ' . $ime . ' ' . $prezime . '!';
    }

    porukaDobrodoslice($imeKorisnika, $prezimeKorisnika);
?>
```

Rezultat:



Dobrodosli Filip Furtula!

Napomena, umesto spajanja stringova, moglo je da se uradi i na sledeći način:

```
echo "Dobrodosli $ime $prezime !";
```

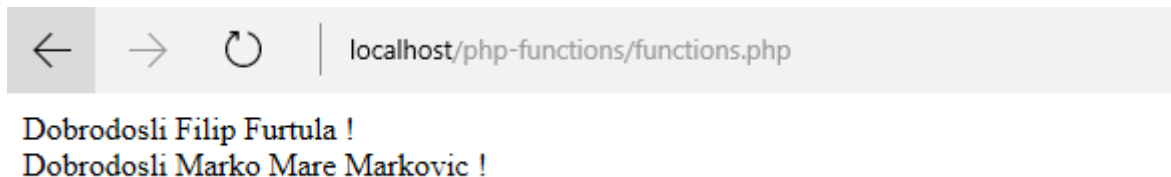
Još jedna veoma bitna stvar kod funkcija u PHP-u jeste mogućnost korišćenja "default parametra". Primer:

```
<?php
    $imeKorisnika = 'Filip';
    $prezimeKorisnika = 'Furtula';

    function porukaDobrodoslice($ime, $prezime, $srednjeIme = '')
    {
        echo "Dobrodosli $ime $srednjeIme $prezime !";
        echo "<br/>";
    }
```

```
porukaDobrodoslice($imeKorisnika, $prezimeKorisnika);  
porukaDobrodoslice("Marko", "Markovic", "Mare");  
?>
```

Rezultat:



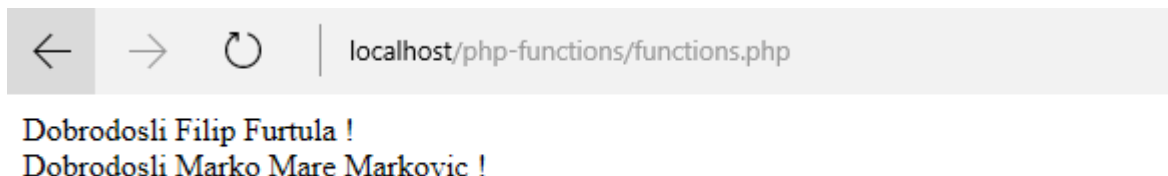
localhost/php-functions/functions.php

Dobrodosli Filip Furtula !
Dobrodosli Marko Mare Markovic !

Sledeća stvar koja je veoma bitna kod funkcija jeste povratna vrednost. Primer:

```
<?php  
$imeKorisnika = 'Filip';  
$prezimeKorisnika = 'Furtula';  
  
function ispisiPoruku($poruka)  
{  
    echo $poruka;  
    echo "<br/>";  
}  
  
function generisiPoruku($ime, $prezime, $srednjeIme = '')  
{  
    return "Dobrodosli $ime $srednjeIme $prezime !";  
}  
  
$poruka1 = generisiPoruku($imeKorisnika, $prezimeKorisnika);  
$poruka2 = generisiPoruku("Marko", "Markovic", "Mare");  
  
ispisiPoruku($poruka1);  
ispisiPoruku($poruka2);  
?>
```

Rezultat će biti identičan:



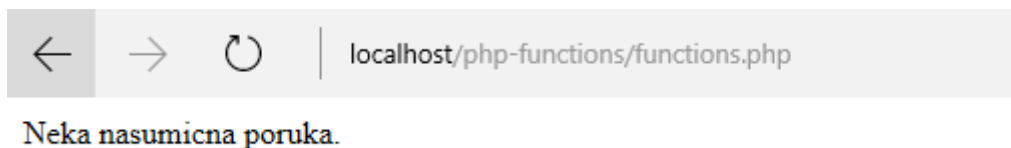
localhost/php-functions/functions.php

Dobrodosli Filip Furtula !
Dobrodosli Marko Mare Markovic !

Još jedna veoma zanimljiva stvar kada su PHP funkcije u pitanju jesu funkcije kao varijable. Zapravo je stvar u tome da je moguće sačuvati naziv funkcije unutar varijable, a onda uz pomoć varijable pozivati tu istu funkciju, primer:

```
<?php
function ispisiPoruku($poruka)
{
    echo $poruka;
    echo "<br/>";
}
$nazivFunkcije = 'ispisiPoruku';
$nazivFunkcije("Neka nasumicna poruka.");
```

Rezultat koda je sledeci:



← → ↻ | localhost/php-functions/functions.php

Neka nasumicna poruka.

Takođe je bitno napomenuti i pojam definisanosti i životnog ciklusa neke promenljive kada je reč o funkcijama. Životni ciklus neke promenljive u okviru funkcije traje onoliko koliko traje i sama funkcija, dok oblast definisanosti funkcije može da varira. Primer:

```
<?php

$name = "Petar";

function setName()
{
    $name = "Dusan";
    echo $name; // prvi slucaj
    echo "<br/>";
}

setName();
echo $name; // drugi slucaj

?>
```

U prvom slučaju pošto je pozicija u okviru funkcije ispisaće se “Dusan” iz razloga što je postavljena vrednost parametra “name” i isto ga tu ispisujemo u okviru funkcije. To je lokalna promenljiva te funkcije i definisana je i ima životni ciklus samo u okviru te funkcije, nakon završetka funkcija se briše iz memorije i nestaje. Međutim ukoliko se posmatra drugi slučaj na ekranu će se ispisati “Petar”, iz razloga što se funkcija “setName” završila i sve lokalne promenljive koje su definisane u okviru nje su izbrisane iz memorije, takođe promenljiva “name” u okviru funkcije je definisana samo u okviru vitičastih zagrada te funkcije. Primer:

Dusan
Petar

U prethodnim primerima smo videli kako se prosleđuju parametri funkciji i kako funkcija može da vrati neki rezultat, to se odnosilo na prenos parametara preko vrednosti. Dakle funkciji treba dati neku vrednost (podatak), funkcija iskoristi i vrati neki rezultat, ali u tom slučaju funkcija ne može da izmeni vrednost tog podatka u memoriji. Kada se koristi prenos preko vrednosti, funkcija koristi kopiju podataka da bi odradila neki zadatak i po potrebi vraća neki rezultat koji je ona generisala. Međutim ukoliko je potrebno pozvati neku funkciju, koja treba sama da izmeni vrednost nekog podatka koji je prosleđen, tada se mora koristiti prenos parametara preko reference. Funkciji se ne daje kopija vrednosti sa kojom treba da radi već joj se prosleđuju reference na memorijsku lokaciju, gde se neki podatak nalazi. Na taj način funkcija može da pristupi toj lokaciji i da izmeni vrednost tog podatka. Za prenos podataka preko reference koristi se karakter “&”, primer:

```
<?php
function dodajText(&$string)
{
    $string .= 'reference.';
}

$s = 'Prenos preko, ';
dodajText($s);
echo $s;
?>
```

Tako da bi rezultat bio: “**Prenos preko, reference.**”

Ili ukoliko bi se koristio sa brojevima:

```
<?php

$a = 4;
test ($a);

function test(&$a)
{
    $a = 5;
}

echo $a;
?>
```

Tako da bi rezultat bio: “**5**”.

PHP

3.1 Ugrađene funkcije za rad sa varijablama

Php poseduje niz ugrađenih funkcija za rad sa varijablama:

gettype()

Ova funkcija vraća vrednost tipa promenljive. Moguće vrednosti:

- "double"
- "string"
- "array"
- "object"
- "class"
- "unknown type"

```
<?
$user_input= 2;
if (gettype ($user_input) == "integer") {
    $age = $user_input;
}
?>
```

settype()

Funkcija postavlja vrednost tipa podataka:

- array,
- double, integer,
- object
- string.

```
<?
$a = 7.5;
settype($a, "integer");
echo $a;
?>
```

isset() i unset()

unset() se koristi za brisanje promenljive i oslobađanje memorijske lokacije koju ona zauzima.

isset() proverava da li je promenljivoj zadata vrednost.

```
<?
$productID = "432BB";
if (isset($productID)) {
    echo("This will print");
}
unset($productID);
if (isset ($productID)) {
    echo("This will NOT print");
}
?>
```

is...() functions

Funkcija ispituje tip promenljive:


```
is_int(), is_long(), is_double(), is_real(), is_string(), is_array(),  
is_object()
```

```
<?  
$ProductID = "432BB";  
if (is_string ($ProductID)) {  
    echo ("String");  
}  
?>
```

4 Stringovi

Stringovi kao tip podataka u php-u predstavljaju niz karaktera koji je veoma znacajan i nad kojim mozemo raditi razlicite operacije i pozivati razlicite funkcije koje su ugradjene u sam php. U prethodnim primerima su kod stringova korišćeni jednostruci i dvostruki navodnici, naravno postoji značajna razlika među njima. Ukoliko se koriste jednostruki navodnici onda će se videti identičan sadržaj koji se nalazi između njih, to jest ukoliko se stavi neka promenljiva unutar njih php će ispisati naziv te promenljive, a ne njenu vrijednost. Primer:

```
<?php
    echo 'Moje ime je Filip';
?>
```

Rezultat:

← → ↻ | localhost/php-types/stringovi.php

Moje ime je Filip

```
<?php
    $name = 'Filip';
    echo 'Moje ime je $name';
?>
```

Rezultat:

← → ↻ | localhost/php-types/stringovi.php

Moje ime je \$name

Takođe ukoliko postoji potreba da se koristi jednostruki navodnik u okviru stringa koristi se karakter “\” ispred tog jednostrukog navodnika kako bi on znao da ga posmatra kao običan karakter, a ne kao kraj stringa. Pogledajmo primer:

```
<?php
    echo 'Da l\' bi bio ljubazan da mi dodas ranac?';
?>
```

Rezultat:

← → ↻ | localhost/php-types/stringovi.php

Da l' bi bio ljubazan da mi dodas ranac?

Za razliku od jednostrukih navodnika ukoliko bi se koristili dvostruki navodnici onda bi rezultati bili drugačiji, kada su prethodni primeri u pitanju, primer:

```
<?php
    echo "Moje ime je Filip";
?>
```

```
<?php
    $name = 'Filip';
    echo "Moje ime je $name";
?>
```

```
<?php
    echo "Da l' bi bio ljubazan da mi dodas ranac?";
?>
```

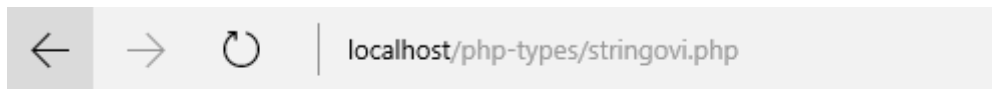
Rezultati:



Takođe u okviru duplih navodnika treba povezati promenljive sa ostatkom teksta ukoliko se koriste vitičaste zagrade. Primer:

```
<?php
    $mesto = 4;
    echo "Takmicenje pocinje {$mesto}og Marta";
?>
```

Rezultat:



Takmicenje pocinje 4og Marta

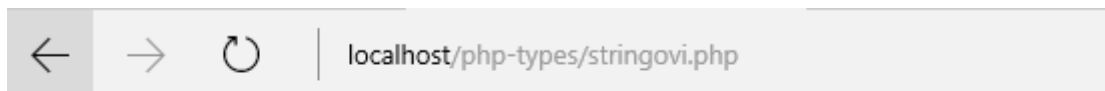
U narednom tekstu će biti objašnjene neke od funkcija koje postoje za rad sa stringovima.

4.1 strtoupper()

Reč je o funkciji koja svako slovo prosleđenog stringa postavlja na veliko slovo, primer:

```
<?php
    $malaSlova = 'ova recenica je napisana malim slovima.';
    echo $malaSlova;
    echo "<br/>";
    $velikaSlova = strtoupper($malaSlova);
    echo $velikaSlova;
?>
```

Rezultat:



ova recenica je napisana malim slovima.
OVA RECENICA JE NAPISANA MALIM SLOVIMA.

4.2 strtolower()

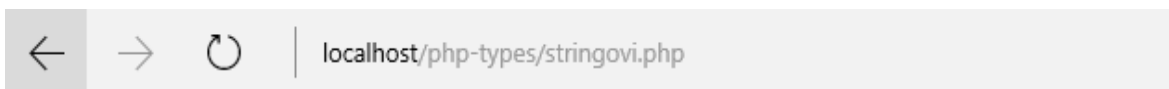
Ova funkcija radi na sličnom principu kao prethodna, samo sto umesto da pretvara slova u velika ona radi obrnut proces i pretvara sva slova u mala za neki prosleđeni string.

4.3 strlen()

Ova funkcija omogućava određivanje dužinu nekog stringa, što često može biti od koristi. Primer:

```
<?php
    $recenica = 'Fakultet organizuje zurku u prostorijama citaonice.';
    $brojSlova = strlen($recenica);
    echo "Recenica: '$recenica' ima: $brojSlova karakter";
?>
```

Rezultat:



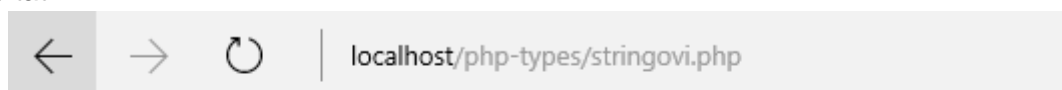
Recenica: 'Fakultet organizuje zurku u prostorijama citaonice.' ima: 51 karakter

4.4 strpos()

Reč je o funkciji koja vraća indeks, gde se nalazi neki karakter ili niz karaktera za koji se ispituje da li se nalazi u datom stringu. Ova funkcija prima 2 obavezna parametra i jedan opcioni parametar. Prvi parametar koji prima jeste string u okviru kojeg se ispituje da li se nalazi neki karakter ili niz karaktera. Drugi parametar jeste upravo taj karakter ili niz karaktera koji se traži. Dok treći karakter predstavlja mogućnost da se funkciji kaže od kojeg indeksa u stringu počinje pretraga za datim karakterom ili nizom karaktera. Važno je jos napomenuti da funkcija ukoliko dati string ne sadži karakter ili niz karaktera koji se traži vraća kao rezultat prazan string. Primer:

```
<?php
$recenica = 'Fakultet organizuje zurku na fakultetu.';
// Indeks reci 'zurka'
echo 'Pozicija: ' . strpos($recenica, 'zurku');
echo "<br/>";
// Pocni pretragu od 25og indeksa
echo 'Pozicija: ' . strpos($recenica, 'tet', 25);
echo "<br/>";
// Nije pronadjeno
echo 'Pozicija: ' . strpos($recenica, 'Univerzitet');
?>
```

Rezultat:



Pozicija: 20

Pozicija: 34

Pozicija:

4.5 str_replace()

Reč je o funkciji koja vrši neku vrstu izmene našeg stringa, kao prvi parametar prima karakter ili niz karaktera koji se već nalaze u stringu i koje se menjaju. Drugi karakter predstavlja vrednost kojom se menja prvi parametar. Treći parametar predstavlja string nad kojim se vrše izmen izmene i kao u prethodnom primeru i ovde postoji još jedan parametar koji je opcioni i omogućava da se toj funkciji prosledi promenljiva. Funkcija će u toj promenljivoj da upiše koliko puta je izvršena izmena u tom stringu. Očigledno je u pitanju prenos preko reference, to jest u pitanju su pokazivači. Primer:

```
<?php
    $recenica = 'on i samo on.';
    $nakonIzmene = str_replace('on', 'oni', $recenica, $brojZamena);
    echo "Nakon izmene: '$nakonIzmene' broj zamena: $brojZamena";
?>
```

Rezultat:

← → ↻ | localhost/php-types/stringovi.php

Nakon izmene: 'oni i samo oni.' broj zamena: 2

4.6 substr()

Ova funkcija omogućava da iz određenog stringa izvučemo samo neki njegov deo, to jest da izvuče podstring iz nekog stringa. Prvi parametar koji ova funkcija prima jeste čitav string iz kojeg se izvlači neki njegov podstring, drugi parametar predstavlja od indeksa od kojeg se kreće. Treći opcioni parametar - ukoliko se ne navede onda će se kao rezultat dobiti podstring od zadatog indeksa pa do kraja niza, dok ukoliko se definiše ond predstavlja dužinu podstringa. Poslednja dva parametra mogu biti negativna, primer:

```
<?php
    $recenica = 'Testiranje funkcije substr() u php-u.';
    echo substr($recenica, 4);
    echo '<br/>';
    echo substr($recenica, -4);
    echo '<br/>';
    echo substr($recenica, 4, 15);
    echo '<br/>';
    echo substr($recenica, 4, -5);
?>
```

Rezultat:

← → ↻ | localhost/php-types/stringovi.php

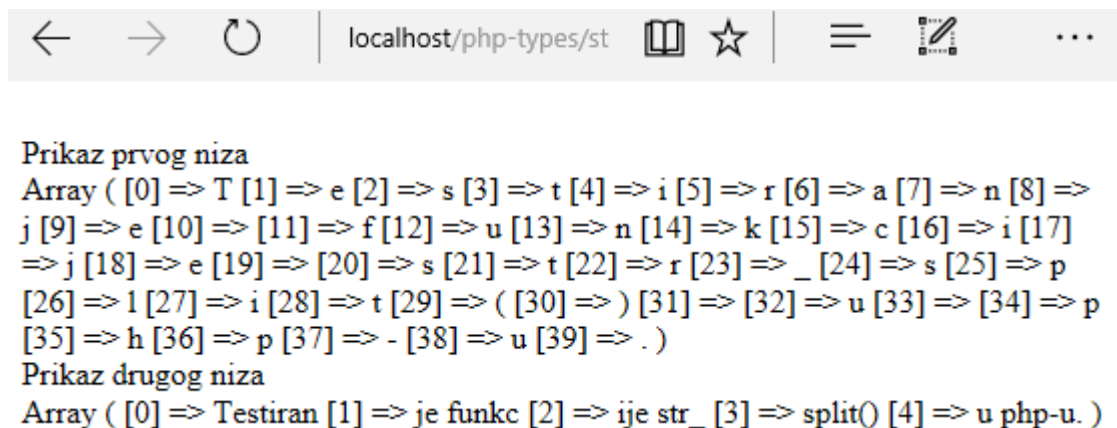
iranje funkcije substr() u php-u.
p-u.
iranje funkcije
iranje funkcije substr() u p

4.7 str_split()

Ova funkcija omogućava da se neki string veoma jednostavno pretvori u niz karaktera. Prvi i jedini obavezan parametar koji ova funkcija zahteva jeste sam string koji zelimo da pretvorimo u niz. Drugi opcioni parametar jeste koliko karaktera treba ima svaki od elemenata tog niza (po default-u jedan karakter ukoliko se ne navede opcioni parametar). Konkretan primer vezan za ovu funkciju:

```
<?php
    $recenica = 'Testiranje funkcije str_split() u php-u.';
    $nizKaraktera = str_split($recenica);
    echo "<br/>Prikaz prvog niza<br/>";
    print_r($nizKaraktera);
    $nizKaraktera = str_split($recenica, 8);
    echo "<br/>Prikaz drugog niza<br/>";
    print_r($nizKaraktera);
?>
```

Rezultat:



```
Prikaz prvog niza
Array ( [0] => T [1] => e [2] => s [3] => t [4] => i [5] => r [6] => a [7] => n [8] =>
j [9] => e [10] => [11] => f [12] => u [13] => n [14] => k [15] => c [16] => i [17]
=> j [18] => e [19] => [20] => s [21] => t [22] => r [23] => _ [24] => s [25] => p
[26] => l [27] => i [28] => t [29] => ( [30] => ) [31] => [32] => u [33] => [34] => p
[35] => h [36] => p [37] => - [38] => u [39] => . )
Prikaz drugog niza
Array ( [0] => Testiran [1] => je funkce [2] => ije str_ [3] => split() [4] => u php-u. )
```

*Funkcija "**print_r**" služi za prikazivanje nekog niza. Ona dati niz formatira u nekom obliku koji je čitljiv za ljude, odakle možemo zaključiti indeks elementa nekog niza kao i vrednost nekog elementa.

Pored navedenih funkcija postoje mnoge druge koje se mogu iskoristiti i koje mogu biti od koristi za rešavanje nekog konkretnog problema, za više detalja pogledati na <http://php.net/manual/en/ref.strings.php>.

5 Nizovi

Kao što je poznato, u nizovima se čuvaju vrednosti elemenata koji su istog tipa i koji su slični na neki način. Ti elementi se zovu elementi niza i svaki element ima svoj jedinstven ID (*Identifier*) i može mu se lako pristupiti. Postoje tri tipa nizova:

- Numerički niz, gde elementi imaju numeričke ID ključeve
- Asocijativni niz (*associative array*), gde je svaki ID ključ asociran sa vrednošću, tj. postoji par ključ=>vrednost (*key=>value*)
- Multidimenzioni nizovi (*multidimensional array*), niz koji se sastoji od jednog ili više nizova.

Element niza može pripadati i bilo kom tipu podataka, bez obzira na ostale elemente.

Nizovi zapravo predstavljaju kolekciju tipova, oni mogu sadržati bilo koji tip, čak mogu sadržati i druge nizove. Kod nizova svaki element niza sadrži određeni ključ (key) preko kojeg ga možemo identifikovati ili pronaći u datom nizu. Za nizove je karakteristična ta relacija ključ => vrednost (key => value).

Pogledajmo kako se definišu nizovi u php-u:

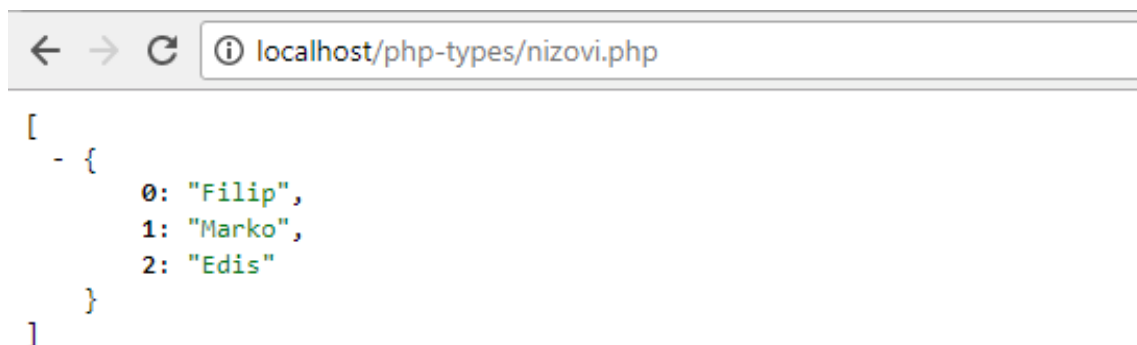
```
<?php
    //Prvi nacin
    $imena = array('Filip', 'Marko', 'Edis');

    print_r($imena);
?>

<?php
    //Drugi nacin
    $prezimana = ['Filipovic', 'Stanimirovic', 'Sarda'];

    print_r($prezimana);
?>
```

Rezultati:



```
[
  - {
    0: "Filip",
    1: "Marko",
    2: "Edis"
  }
]
```



```
localhost/php-types/nizovi.php

[
  - {
    0: "Filipovic",
    1: "Stanimirovic",
    2: "Sarda"
  }
]
```

Pored svake vrednosti koja je uneta stoji i odgovarajući indeks (koji zapravo predstavlja ključ elementa koji smo malopre pominjali), sa obzirom da nisu unešeni indeksi već je to php uradio za nas, onda se ovi nizovi nazivaju *indeksirani*.

Primer:

```
<?php
//Mesoviti niz
$mixedArray = [1, 0.5, 'Neki string', true, false];

print_r($mixedArray);
?>
```

Rezultat:

```
localhost/php-types/nizovi.php

[
  - {
    0: "1",
    1: "0.5",
    2: "Neki string",
    3: "1",
    4: ""
  }
]
```

Kod indeksiranih nizova, kod kojih programer ne menja indekse (key) već ih php sam generiše. Međutim ukoliko se promeni vrednost nekog ključa(key), taj niz ne nazivamo indeksiranim, već asocijativnim nizom.

Kod asocijativnih nizova pored vrednosti za elemente unose se i ključevi (par key => value gorepomenuti pominjali). Primer:

```
<?php
//Asocijativni niz
$associativeArray = array(
    "ime" => "Petar",
    "prezime" => "Petrovic",

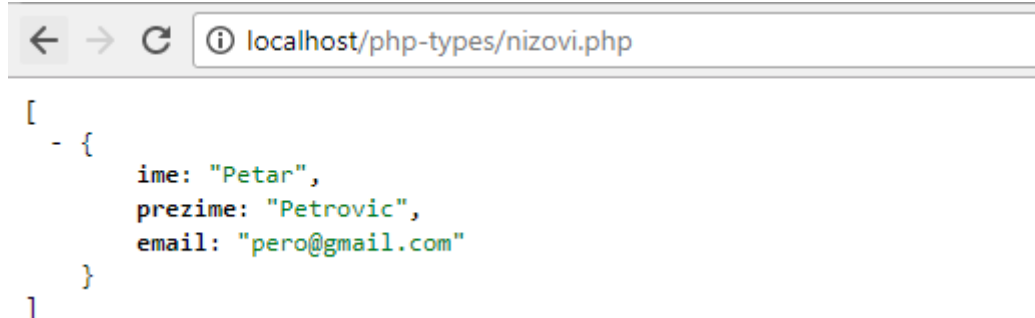
```

```

        "email" => "pero@gmail.com"
    );
    print_r($associativeArray);
?>

```

Rezultat:



```

[
  - {
    ime: "Petar",
    prezime: "Petrovic",
    email: "pero@gmail.com"
  }
]

```

Uglavnom za ključ (key) kod asocijativnih nizova se stavlja neki string ali takođe za ključ se može postaviti broj, međutim, to ga neće činiti indeksiranim nizom. Još jedna zanimljiva karakteristika asocijativnih nizova jeste to da se nekom elementu može dodeliti vrednost bez ključa a da će php tom elementu sam da dodeli indeks i to neće krenuti od nule ('0') već će tom elementu za ključ dodeliti za jedan veći indeks od poslednjeg indeksa koji se nalazi u nizu. Primer:

```

<?php
    //Asocijativni niz
    $associativeArray = array(
        8 => "Pero",
        "pero@gmail.com",
        "ime" => "Petar",
        "prezime" => "Petrovic"
    );
    print_r($associativeArray);
?>

```

Rezultat:



```

[
  - {
    8: "Pero",
    9: "pero@gmail.com",
    ime: "Petar",
    prezime: "Petrovic"
  }
]

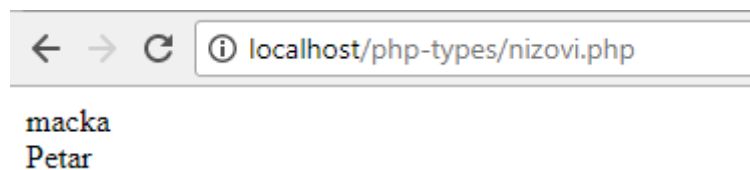
```

5.1 array_key_exists()

Da bi se pristupilo nekom elementu niza potreban je njegov ključ, koji ga jedinstveno identifikuje. Primer:

```
<?php
//Pristup elementu indeksiranog niza
$indexedArray = ['pas', 'macka', 'ptica', 'slon'];
echo $indexedArray[1];
echo "<br/>"
//Pristup elementu asociativnog niza
$associativeArray = array(
    8 => "Pero",
    "pero@gmail.com",
    "ime" => "Petar",
    "prezime" => "Petrovic"
);
echo $associativeArray['ime'];
?>
```

Rezultat:

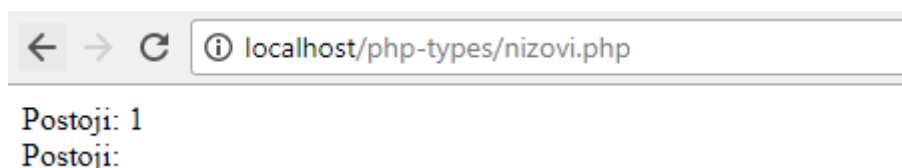


```
← → ↻ ⓘ localhost/php-types/nizovi.php
macka
Petar
```

Međutim problem nastaje kada nije poznato da li u nizu postoji ključ koji je prosleđeno, ukoliko bi u prethodnom primeru umesto "1" prosledili niz broj "5" php bi javio grešku iz razloga što u nizu ne postoji element sa ključem "5". Prvo treba proveriti da li ključ koji je prosleđen postoji u nizu. U tom slučaju funkcija "array_key_exists()" kao parametre prima ključ koji se traži i niz u kojem treba da proveriti da li se neki kjuč nalazi. Ukoliko se ključ nalazi vratiće vrednost "1". Ukoliko se ne nalazi, tada će kao i sve ostale funkcije u php-u vratiti prazan string sto u php-u simbolizuje "false". Primer:

```
<?php
$indexedArray = ['pas', 'macka', 'ptica', 'slon'];
echo 'Postoji: ' . array_key_exists(1, $indexedArray);
echo "<br/>";
echo 'Postoji: ' . array_key_exists(5, $indexedArray);
echo "<br/>";
?>
```

Rezultat:



```
← → ↻ ⓘ localhost/php-types/nizovi.php
Postoji: 1
Postoji:
```

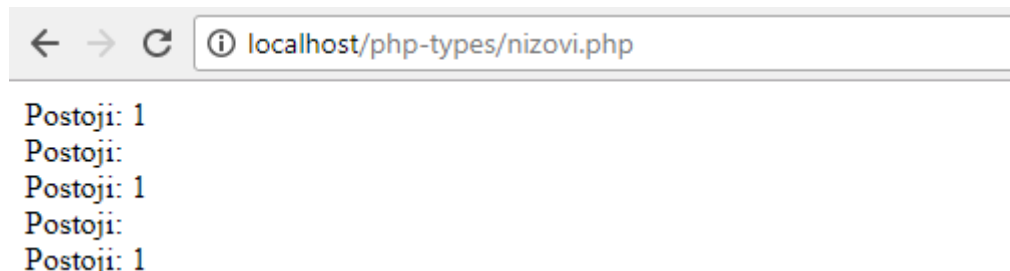
5.2 in_array()

Funkcija koja proverava da li postoji neka vrednost u okviru niza je *in_array* funkciju. Ova funkcija prima dva obavezna parametra i jedan opcioni. Prvi parametar jeste vrednost koju se traži u okviru niza, drugi parametar jeste niz u okviru kojeg se traži neka vrednost i treći opcioni parametar koji je tipa *boolean* govori da li da se pored toga što se vrednosti poklapaju da se i poklapaju njihovi tipovi. Primer:

```
<?php
$array = array(
    "Pero",
    "pero@gmail.com",
    "Petar",
    "Petrovic",
    12
);

echo 'Postoji: ' . in_array("Pero", $array) . '<br/>';
echo 'Postoji: ' . in_array("Milan", $array) . '<br/>';
echo 'Postoji: ' . in_array(12, $array, true) . '<br/>';
echo 'Postoji: ' . in_array("12", $array, true) . '<br/>';
echo 'Postoji: ' . in_array("12", $array) . '<br/>';
?>
```

Rezultat:



```
Postoji: 1
Postoji:
Postoji: 1
Postoji:
Postoji: 1
```

5.3 array_push()

Jedna od osnovnih operacija kada su nizovi u pitanju jeste dodavanje novog elementa u taj niz. Za dodavanje elemenata u niz postoje dva načina, prvi način jeste da se koristi gore navedena funkcija koja prima dva parametra. Prvi parametar jeste niz u koji se ubacuje neki element, a drugi parametar jeste element koji se ubacuje u niz. Primer:

```
<?php
$array = array('Filip', 'Marko', 'Edis');

array_push($array, 'Milos');

print_r($array);
```

```
?>
```

Rezultat:

```
[
  - {
    0: "Filip",
    1: "Marko",
    2: "Edis",
    3: "Milos"
  }
]
```

Drugi način za ubacivanje novog elementa u niz jeste da se koriste uglaste zagrade, primer:

```
<?php
$array = array('Filip', 'Marko', 'Edis');

$array[] = 'Milos';
print_r($array);
?>
```

Rezultat:

```
[
  - {
    0: "Filip",
    1: "Marko",
    2: "Edis",
    3: "Milos"
  }
]
```

Ukoliko bi uporedili ova dva načina za ubacivanje elemenata u niz zaključili bi da je drugi način svakako bolji, to jest bolje je koristiti uglaste zagrade umesto gore navedene funkcije. Ukoliko se koristi funkcija treba voditi računa da je niz inicijalizovan u suprotnom došlo bi do greške, za razliku od pristupa kada se koriste uglaste zagrade, gde će php, ukoliko niz nije kreiran, sam će ga kreirati i ubaciti element. Takođe još jedna velika razlika je ta da pomoću funkcije ukoliko se želi ubaciti element u asocijativni niz, on bi za ključ tog elementa odabrao prvi slobodan indeks ne dajući slobodu da se samostalno definiše ključ za taj element, za razliku od slučaja, kada se koriste uglaste zagrade gde je to moguće. Primer:

```
<?php
$associativeArray = array(
    "ime" => "Petar",
);

array_push($associativeArray, "Petrovic");
print_r($associativeArray);
?>
```

Rezultat:

```
[
  - {
    0: "Petrovic",
    ime: "Petar"
  }
]
```

```
<?php
    $associativeArray = array(
        "ime" => "Petar",
    );

    $associativeArray["prezime"] = "Petrovic";
    print_r($associativeArray);
?>
```

Rezultat:

```
[
  - {
    ime: "Petar",
    prezime: "Petrovic"
  }
]
```

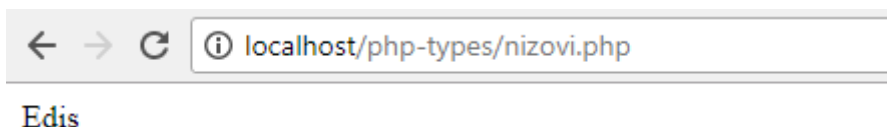
5.4 array_pop()

Postoji i funkcija za izbacivanje nekog elementa iz niza. Ukoliko se posmatra funkcija “array_pop()” njoj se prosleđuje samo jedan parametar, a to je niz. Onda funkcija izbacuje poslednji element iz tog niza i kao rezultat vraća element koji je izbacila. Primer:

```
<?php
    $array = array('Filip', 'Marko', 'Edis');

    $lastElement = array_pop($array);
    echo $lastElement;
?>
```

Rezultat:



Edis

Ukoliko bi se sada ispisao niz:

```
print_r($array);
```

Rezultat bi bio:

```
[
  - {
    0: "Filip",
    1: "Marko"
  }
]
```

5.5 unset()

Funkciju “unset()” funkcija oslobađa resurse. Kada se pozove ova funkciju nad nekom promenljivom koja je nekog određenog tipa ona tu promenljivu briše iz memorije, kao da nikada nije ni postojala. Primeri:

```
<?php
$array = array('Filip', 'Marko', 'Edis');
unset($array[1]);
print_r($array);
?>
```

Rezultat:

```
[
  - {
    0: "Filip",
    2: "Edis"
  }
]
```

```
<?php
$array = array('Filip', 'Marko', 'Edis');
unset($array[1], $array[2]);
print_r($array);
?>
```

Rezultat:

```
[
  - {
    0: "Filip"
  }
]
```

```
<?php
$associativeArray = array(
    "ime" => "Petar",
    "prezime" => "Petrovic"
);
unset($associativeArray["ime"]);
```

```
print_r($associativeArray);  
?>
```

Rezultat:

```
[  
  - {  
      prezime: "Petrovic"  
  }  
]
```

```
<?php  
$array = array('Filip', 'Marko', 'Edis');  
  
unset($array);  
  
$associativeArray = array(  
    "ime" => "Petar",  
    "prezime" => "Petrovic"  
);  
  
unset($associativeArray["ime"], $associativeArray["prezime"]);  
print_r($associativeArray);  
print_r($array);  
?>
```

Rezultat:

```
[  
  { },  
  "<br>",  
  "<b>Notice</b>: Undefined variable: array in <b>C:\xampp\htdocs\php-types\nizovi.php</b> on line <b>13</b><br>"  
]
```

Funkcija *unset* funkcioniše i za asociativne nizove i pomoću nje se može ukloniti više promenljivih koristeći zarez (",") kada navodimo parametre funkcije, to jest možemo proslediti koliko god želimo promenljivih. U poslednjem primeru je pomoću *unset* ispraznjen ceo asociativni niz. Dok je u drugom slučaju došlo do greške, jer je pokušano da se pozove niz (promenljiva: *\$array*), koji je pomoću funkcije *unset* prethodno oslobođen.

5.6 Sortiranje nizova

Sortiranje predstavlja operaciju kada se elementi unutar niza dovode u neki uređen redosled po nekom kriterijumu. Postoji nekoliko funkcija koje php nudi za sortiranje nizova.

5.6.1 sort()

Za sortiranje nizova možemo iskoristiti funkciju `sort()` koja sortira niz po abecednom redosledu, veoma je pogodna za indeksirane nizove, dok za asociativne nizove i nije toliko primenljiva jer ukoliko se primeni nad nekim asociativnim nizom ona će umesto ključeva (key) unutar tog niza da postavi indekse, primer:

```
<?php
    $indexedArray = array('Filip', 'Stefan', 'Ana');

    sort($indexedArray);

    $associativeArray = array(
        "ime" => "Petar",
        "prezime" => "Aleksic",
        "srednje_ime" => "Beli",
    );

    sort($associativeArray);
    print_r($associativeArray);
    print_r($indexedArray);
?>
```

Rezultat:

```
[
  - {
    0: "Aleksic",
    1: "Beli",
    2: "Petar"
  },
  - {
    0: "Ana",
    1: "Filip",
    2: "Stefan"
  }
]
```

5.6.2 asort()

Za razliku od prethodne funkcije koja i nije toliko pogodna za rad sa asocijativnim nizovima “`asort()`” funkcija u potpunosti odgovara za asociativne nizove, dok se veoma retko ili čak nikada neće naći njena primena za indeksirane nizove. Ključna prednost “`asort()`” funkcije je u tome što ona prilikom sortiranja čuva integritet ključ => vrednost para (key => value) tako da kada se dobije sortiran niz tada će svaki element imati isti onaj ključ koji je imao i pre

nego što je započeto sortiranje. Upravo iz tog razloga on nalazi slabu primenu kod indeksiranih nizova jer prilikom sortiranja se izgubi integritet indeksa. Primer:

```
<?php
    $associativeArray = array(
        "ime" => "Petar",
        "prezime" => "Aleksic",
        "srednje_ime" => "Beli",
    );

    asort($associativeArray);
    print_r($associativeArray);
?>
```

Rezultat:

```
[
    - {
        prezime: "Aleksic",
        srednje_ime: "Beli",
        ime: "Petar"
    }
]
```

5.6.3 ksort()

Ukoliko se javi potreba da se sortira neki niz prema vrednostima ključeva za elemente onda je funkcija *ksort()* najbolje rešenje za. Ova funkcija posmatra samo ključeve i vrši sortiranje prema njima, ali kao i funkcija *asort()* takođe vodi računa o tome da zadrži integritet ključ => vrednost (key => value), to jest da prilikom sortiranja za svaki ključ ostane isti element koji je bio i pre sortiranja. Primer:

```
<?php
    $associativeArray = array(
        "ime" => "Petar",
        "srednje_ime" => "Beli",
        "prezime" => "Aleksic",
    );

    ksort($associativeArray);
    print_r($associativeArray);
?>
```

Rezultat:

```
[  
  - {  
    ime: "Petar",  
    prezime: "Aleksic",  
    srednje_ime: "Beli"  
  }  
]
```

5.7 count()

Još jedna veoma korisna funkcija jeste upravo funkcija “*count()*” koja omogućava da se odredi broj elemenata unutar nekog niza. Dakle funkcija kao parametar prima niz za koji se traži broj elemenata i kao rezultat vraća integer, to jest broj elemenata unutar tog niza. Primer:

```
<?php  
$students = ['Filip', 'Marko', 'Edis', 'Petar'];  
  
echo count($students);  
?>
```

Rezultat:

4

Primer sa višedimenzionalnim nizom:

```
<?php  
$persons =  
[  
  "students" => [  
    'Filip',  
    'Marko',  
    'Edis'  
  ],  
  "teachers" => [  
    'Marko',  
    'Dejan',  
    'Milos'  
  ]  
];  
echo count($persons);  
?>
```

Rezultat:

2

Ovaj put funkcija “count()” je kao rezultat vratila broj dva iz razloga što se u nizu “\$persons” nalaze dva elementa koji su takođe nizovi, ali bez obzira na to u nizu “\$persons” se nalaze samo dva elementa. Da bi se izračunao broj svih elemenata svih nizova u okviru nekog niza potrebno je koristiti drugi opcioni parametar funkcije “count()”, što je konstanta koja je ugrađena u php-u “COUNT_RECURSIVE”. S obzirom da je konstanta ugrađena u php nema potrebe da se ta konstanta deklarira (Napomena: umesto konstante može se koristiti i broj ‘1’). Pogledati to na prethodnom primeru gde sesamo izmeni poziv funkcije “count()” u:

```
// Ukoliko koristimo konstantu
echo count($persons, COUNT_RECURSIVE);
// Ukoliko ne koristimo konstantu
echo count($persons, 1);
```

Rezultat:

8

5.8 Višedimenzionalni nizovi

Višedimenzionalni nizovi predstavljaju slučaj kada unutar nekog niza imamo druge nizove. U nastavku je dat primer višedimenzionalnog niza:

```
<?php
$persons =
[
    "students" => [
        "first_year" => [
            [
                "ime" => "Filip",
                "prezime" => "Furtula"
            ],
            [
                "ime" => "Marko",
                "prezime" => "Stanimirovic"
            ]
        ],
        "second_year" => [
            [
                "ime" => "Edis",
                "prezime" => "Sarda"
            ]
        ]
    ],
    "teachers" => [
        "profesors" => [
            [
                "ime" => "Marko",
```

```

        "prezime" => "Perovic"
    ]
  ],
  "assistants" => [
    [
      "ime" => "Milos",
      "prezime" => "Zlatic"
    ]
  ]
];
print_r($persons);

```

Rezultat bi bio sledeci:

```

- {
  - students: {
    - first_year: {
      - 0: {
        ime: "Filip",
        prezime: "Furtula"
      },
      - 1: {
        ime: "Marko",
        prezime: "Stanimirovic"
      }
    },
    - second_year: {
      - 0: {
        ime: "Edis",
        prezime: "Sarda"
      }
    }
  },
  - teachers: {
    - profesors: {
      - 0: {
        ime: "Marko",
        prezime: "Perovic"
      }
    },
    - assistants: {
      - 0: {
        ime: "Milos",
        prezime: "Zlatic"
      }
    }
  }
},

```

Jedna od bitnih stvari kada su višedimenzionalni nizovi u pitanju jeste svakako način kako se može pristupiti nekom elementu tog niza. Za pristup elementima niza koriste se uglaste zagrade. Primer:

```
print_r($persons["students"]);
```

Rezultat izvršavanja ovog koda je sledeći:

```
[
  - {
    - first_year: {
      - 0: {
        ime: "Filip",
        prezime: "Furtula"
      },
      - 1: {
        ime: "Marko",
        prezime: "Stanimirovic"
      }
    },
    - second_year: {
      - 0: {
        ime: "Edis",
        prezime: "Sarda"
      }
    }
  }
]
```

Na ovaj način je pristupljeno samo elementu "students" u okviru niza "persons", to jest prikazani su samo njegovi elementi.

Ukoliko bi se nastavilo sa traženjem nekog elementa, na primer:

```
print_r($persons["students"]["first_year"]);
```

Rezultat bi bio sledeći:

```
[
  - {
    - 0: {
      ime: "Filip",
      prezime: "Furtula"
    },
    - 1: {
      ime: "Marko",
      prezime: "Stanimirovic"
    }
  }
]
```

Kao što se mkože primetiti na ovaj način je prisupljeno elementima unutar niza "first_year" koji se nalazi u okviru niza "students", a niz "students" se nalazi u okviru čitavog niza "persons". Ukoliko bi se dalje nastavilo u strukturi:

```
print_r($persons["students"]["first_year"][1]);
```

Rezultat:

```
[
  - {
    ime: "Marko",
    prezime: "Stanimirovic"
  }
]
```

Pošto se sada nizu *"first_year"* pristupa preko indeksa, tada se za ključ prilikom pristupa u nizu koriste indeksi. Ukoliko bi se išlo dalje do konkretne vrednosti za neki atribut koji se traži, odnosno ukoliko je potrebno naći ime prvog studenta koji se nalazi u nizu studenata trebalo bi koristiti sledeći kod:

```
print_r($persons["students"]["first_year"][1]["ime"]);
```

Rezultat ovog koda je sledeći:

```
Marko
```

Naravno ovo je bio pristup kada se zna kom elementu se pristup i kakva je struktura u kojoj se on nalazi i takođe se pristupa samo jednom elementu. Međutim ukoliko bi se ovaj pristup koristio za sve elemente, to bi bilo krajnje neefikasno. Iz tog razloga kada se radi sa nizovima, jedna od nezaobilaznih tema jesu petlje. Petlje omogućavaju da se na jednostavan način prođe kroz čitavu strukturu niza i da se posete svi elementi ili samo neke od njih. Jedna od najčešće korišćenih petlji u php-u jeste **foreach** petlja. Pogledati u nastavku kako se može koristiti **foreach** petlja.

```
<?php
    $students = ['Filip', 'Marko', 'Edis', 'Petar'];
    foreach($students as $student)
    {
        echo $student . '<br/>';
    }
?>
```

Rezultat:

```
Filip
Marko
Edis
Petar
```

Sintaksa je veoma jednostavna i intuitivna, ukoliko bi se pročitala foreach petlju ona bi glasila: "Za svaki element iz niza *\$students* obrati mu se kao *\$student* i nakon toga ispiši taj element". Sledi primer kako pomoću foreach petlje ispisali sva imena i prezimena studenata prve godine iz niza *"persons"* koji smo koristili u prethodnim primerima:

```
foreach($persons["students"]["first_year"] as $student)
{
    echo $student["ime"] . ' ' . $student["prezime"] . '<br/>';
}
```

```
}
```

Rezultat izvršavanja ovog koda bi bio sledeći:

```
Filip Furtula  
Marko Stanimirovic
```

Naravno foreach petlja omogućava i da se pristupi našim ključevima (key) za elemente i da se iskoriste u kodu. Pogledati na primeru:

```
foreach($persons["students"]["first_year"][1] as $key => $value)  
{  
    echo $key . ': ' . $value . '<br/>';  
}
```

Rezultat je sledeći:

```
ime: Marko  
prezime: Stanimirovic
```

To je bio slučaj kada se za jednog studenta hteli da ispisuje ime i prezime. Ukoliko postoji potreba za sve studente sa prve godine moras se koristiti foreach petlju unutar druge foreach petlje. Primer:

```
foreach($persons["students"]["first_year"] as $student)  
{  
    foreach($student as $key => $value)  
    {  
        echo $key . ': ' . $value . '<br/>';  
    }  
}
```

Rezultat:

```
ime: Filip  
prezime: Furtula  
ime: Marko  
prezime: Stanimirovic
```


6 Izdvojeni koncepti i mogućnosti PHP

6.1 Date

PHP ugrađena funkcija `date()` je korisna i često korišćena funkcija. Ova funkcija formatira *timestamp* (broj sekundi koji je protekao od 1. Januara 1970. godine) u razumljiv oblik prikazivanja datuma i vremena. Sintaksa:

`date(format, timestamp)`

Prvi parametar date funkcije *format* određuje format prikazivanja datuma i vremena. Evo nekih slova koja se mogu koristiti uz *format* parametar:

Format	Značenje	Primer
a	am ili pm	Am
A	AM or PM	AM
d	Dan u mesecu	01
D	Dan u nedelji	Sun
F	Ime meseca	January
h	Sat u 12 - očasovnom formatu	04
H	Sat u 24 - očasovnom formatu	16
g	Sat u 12 - časovnom formatu bez nule ispred	4
G	Sat u 24 - časovnom formatu bez nule ispred	16
i	Minuti	35
j	Dan u mesecu	2
l	Dan u nedelji (ime)	Sunday
L	Prestupna godina (1 da, 0 ne)	1
m	Mesec u godini – tri slova	Jul
M	Mesec u godini, potpun naziv	July
N	Mesec u godini (redni broj)	7
Y	Godina u standardnom formatu	1999, 2008
y	Godina zapisana pomoću dva simbola	99, 08
s	Sekund	58
S	Sufiks za dan	th, nd, st, rd
R	Stardizovan format datuma	Thu, 15 Dec 2005 16:49:39-0600
U	Timestamp	1134512479

```

<html>
<head> <title> Primer za date() funkciju </title> </head>
<body>
<?php
echo date("Y/M/D");
echo "<br>";
echo date("D/M/Y");
echo "<br>";
echo date("D.M.Y");
?>
</body>
</html>

```

Primer 38: Maketime

```

<?php
echo("Validating: 5/31/2005<br>");
if (checkdate(5,31,2005)) {
    echo('Date accepted: ');
    $new_date=mktime(18,05,35,5,31,2005);
    echo date("r",$new_date);
}
else {
    echo ('Invalid date. ');
}
echo("<br>");
?>

```

6.2 Include i Require

U PHP fajl može da se unese sadržaj nekog drugog fajla, pre nego što PHP fajl server izvrši uz pomoć funkcija `include()` ili `require()`. Obe navedene funkcije su skoro iste, jedino se razlikuje način na koji tretiraju greške u kodu (*code errors*). Funkcija `include()` upozorava, ali skripta nastavlja da se izvršava, dok funkcija `require()` generiše ključnu grešku (*fatal error*) i izvršavanje skripte se prekida. Ova tehnologija je poznata ka SSI (*Server Side Includes*).

Dve navedene funkcije se koriste u kreiranju funkcija, hedera (*header*), futera (*footer*) ili drugih elemenata koji se mogu koristiti više puta, na različitim Veb stranim.

Funkcija `include()` uzima ceo sadržaj željenog fajla i kopira ga u fajl u kome se poziva `include()` funkcija. Kao primer, uzeta su dva fajla, jedan se zove *include.php*, a drugi fajl, koji će biti uključen u prvi, zove se *linkovi.html*.

Primer Kod fajla *linkovi.html*

```

<html>
<head> <title> Linkovi </title> </head>
<body>
<a href="http://www.google.com"> Kliknite ovde ako hocete na sajlt Googl -
a </a>
<br>
<a href="http://www.fon.bg.ac.yu"> Kliknite ovde ako hocete na sajt Fon - a
</a>
<br>

```

```
<a href="http://www.yahoo.com"> Kliknite ovde ako hocete na sajt Yahoo - a
</a>
</body>
</html>
```

Primer Kod fajla *include.php*:

```
<html>
<head> <title> Primer za include() </title> </head>
<body>
<?php
include("linkovi.html");
echo "<br>";
echo "Vidite linkove?";
// moze se koristiti i require() funkcija umesto include(), isti je efekat,
samo je error report drugaciji
?>
</body>
</html>
```

6.3 PHP napredne funkcije za rukovanje fajlovima

Da bi se sadržaj nekog fajla pročitao/modifikovao iz PHP skripte, koristi se `fopen()` funkcija. `Fopen()` funkcija prima dva parametra. Prvi parametar je ime fajla nad kojim se operiše, a drugi parametar je način na koji se manipuliše željenim fajlom (*mod*).

Sintaksa:

```
fopen("ime_fajla", "mod");
```

Sledeća tabela prikazuje sve modove koje može sadržati `fopen()` funkcija.

Modovi	Opis
r	Čitanje. Počinje od početka fajla, tj. prvog reda (line).
r+	Čitanje/Pisanje. Počinje od početka fajla, tj. prvog reda (line).
w	Pisanje. Otvara fajl i briše ceo sadržaj fajla, a ako fajl ne postoji, kreira fajl.
w+	Čitanje/Pisanje. Otvara fajl i briše ceo sadržaj fajla, a ako fajl ne postoji, kreira fajl.
a	Dodaje. Otvara fajl i dodaje nov sadržaj na kraj fajla.
a+	Čitanje/Dodaje. Otvara fajl i dodaje nov sadržaj na kraj fajla.
x	Pisanje. Kreira nov fajl, vraća error, ako fajl već postoji.
x+	Čitanje/Pisanje. Kreira nov fajl, vraća error, ako fajl već postoji.

Funkcija `fclose()` nije obavezna, ali kao dobra programerska praksa važi da se otvoreni fajl zatvori.

Funkcija `feof` (*naziv fajla ili promenljive kojoj je fajl dodeljen*) proverava da li je dostignut kraj fajla koji se čita, tj. da li je pokazivač došao do poslednjeg reda fajla.

Funkcija `fgets` (*naziv fajla ili promenljive kojoj je fajl dodeljen*) služi za čitanje fajla i to red po red.

Funkcija `fgetc` (*naziv fajla ili promenljive kojoj je fajl dodeljen*) služi za čitanje fajla i to karakter po karakter.

Za potrebe sledećeg primera, kreiran je fajl sa proizvoljnim tekstom koji je nazvan *proba.txt*. Nakon toga, novokreirani fajl će biti otvoren iz PHP skripte i njegov sadržaj će biti iščitao i prikazan na ekranu i to red po red. Kod izgleda ovako:

```
<html>
<head> <title> Primer za fopen() </title> </head>
<body>
<?php
$fajl=fopen("proba.txt", "r") or exit ("Ne mogu da otvorim fajl");
// otvara se fajl proba.txt u modu r(read)
while (!feof($fajl)){
// dok se ne dodje do kraja fajla, izvršava se kod u viticastim zagradama
echo fgets($fajl) . "<br>";
}
fclose($fajl);
?>
</body>
</html>
```

6.4 Regularni izrazi

Regularni izraz praktično predstavlja poseban skup znakova (string) u kome se odgovarajućom sintaksom (eng. pattern) taj niz upoređuje sa nekim drugim skupom znakova. Može se koristiti za pretragu unutar nekog teksta, izvlačenje određenog podstringa, validaciju (e-maila na primer) i sl. PHP podržava takozvane [POSIX](#) kao i [Perl Kompatibilne](#) regularne izraze. Iako među njima postoje izvesne razlike, osnovna sintaksa je u suštini ista.

Dva osnovna specijalna karaktera jesu '^' i '\$'. Oni označavaju početak, odnosno kraj stringa.

Tako na primer:

- "^foo" - proverava da li string počinje sa "foo"
- "foo\$" - proverava da li se string završava sa "foo"

Pravila rada sa regularnim izrazima su data u nastavku. Simboli '?', '+', '*' i '{' označavaju broj pojavljivanja nekog karaktera u stringu:

? - Karakter koji prethodi znaku '?' može se pojaviti jednom ili nijednom

(za pattern "ab?" odgovaralo bi "a", "ab")

* - Karakter koji prethodi znaku '*' može se pojaviti **nijednom** ili više puta (za pattern "ab*" odgovaralo bi "a", "ab", "abb", "abbb", ...)

+ - Karakter koji prethodi znaku '+' može se pojaviti **jedanput** ili više puta (za pattern "ab+" odgovaralo bi "ab", "abb", "abbb", ...)

{n} - Karakter koji prethodi znaku '{n}' može se pojaviti tačno n puta (za pattern "ab{3}" odgovaralo bi "abbb")

{n, } - Karakter koji prethodi znaku {n, } može se pojaviti najmanje n puta
(za pattern "ab{3,}" odgovaralo bi "abbb", "abbbb", "abbbb", ...)

{n,m} - Karakter koji prethodi znaku {n,m} može se pojaviti n do m puta.
(za pattern "ab{2,4}" odgovaralo bi "abb", "abbb", "abbbb")

Pored broja pojavljivanja, može se definisati i tačan skup znakova koje string sme da sadrži.
Na primer:

- '.' - Bilo koji karakter
- [abc] - Samo slova a, b i c
- [a-z] - Sva mala slova od a do z
- [A-Z] - Sva velika slova od A do Z
- [a-zA-z] - Sva slova, mala ili velika
- [0-9] - Svi brojevi od 0 - 9 [a-zA-Z0-9] Svi alfanumericki karakteri

Unutar zagrada [] simbol '^' se koristi kao negaciju, tako na primer, ako string ne sadrži brojeve treba koristiti bi nešto poput:
[[^]0-9].

Kako regularne izraze upotrebiti u praksi, za validaciju forme:

```
<?php

// validacija korisnickog imena
// dozvoljavamo samo korisnicko ime koje sadrzi alfanum i donju crtu
// minimum 6, max 20 karaktera
$found = preg_match("/^[a-zA-Z0-9_]{6,20}$/", $username);

if(!$found)
{
    echo "Korisnicko ime nije validno";
}

// validacija telefona
// prihvatamo samo brojeve i karaktere iz skupa [+/( )]
//obratite paznju na koriscenje escape karaktera "\" za '.', '/', '(' i ')'
$found = preg_match("/^[0-9+\\-\\.\\/\\(\\) ]{6,30}$/", $phonenum);

if(!$found)
{
    echo "Telefon nije validan";
}

//validacija datuma u mysql formatu (YYYY-MM-DD)
```

```

$found = preg_match("/^[0-9]{4}-[0-9]{2}-[0-9]{2}$/", $datum);

if(!$found)
{
    echo "Datum nije validan";
}

// validacija usa zip koda
// USA zipocode je u formatu xxxxx ili xxxxx-xxxx
// gde je x bilo koji ceo broj
// na primer 12345 ili 12345-1234

$found = preg_match("/^([0-9]{5}|([0-9]{5}-[0-9]{4}))$/", $datum);

if(!$found)
{
    echo "Zipcode nije validan";
}

// jednostavna validacija emaila
$found = preg_match("/^[a-zA-Z0-9\.\-]+\@[a-zA-Z0-9\.\-]+\$/", $datum);

if(!$found)
{
    echo "e-mail nije validan";
}
?>

```

6.5 Superglobals

Sve promenljive koje spolja ulaze u PHP kod, dolaze u jednom od nekoliko specifičnih nizova pod nazivom *superglobals*. Superglobalne promenljive¹ su dostupne u bilo kom delu skripta, čak i unutar objekata, funkcija, nizova. Drugim rečima, one se ne definišu, one su već prisutne. Predstavljaju specifične nizove varijabli, koje dolaze spolja u PHP fajl. Obuhvataju podatke poslate sa HTML formi, podatke o kukijima, informacije o sesiji, upload-ovanim fajlovima serveru i čine ih raspoloživim na svakom mestu. Predstavljaju način da PHP otkrije odakle određena vrednost dolazi. Na ovaj način se ne dozvoljava da bilo koja promenljiva submit – ovana od strane korisnika, automatski uđe u PHP kod. Varijable unutar PHP skripta ostaju izolovane od potencijalnih napada.

Ako na primer korisnik popuni formu, server neće dodeliti te vrednosti globalnim promenljivama \$name, \$password. Umesto toga, ti podaci će biti podeljeni u niz: \$_POST['name'], \$_POST['password'] i može im se pristupiti samo preko globalne promenljive \$_POST

¹ Nisu bile raspoložive sve do verzije PHP 4.1, ali su postojale alternativne metode

\$ GLOBALS

Asocijativni niz koji sadrži reference ka svim globalnim (i superglobalnim) varijablama dostupnim u okviru lokalne skripte. Imena varijabli predstavljaju ključeve ovog niza.

\$ SERVER

Sadrži informacije o okruženju veb servera. Predstavlja niz varijabli podešenih od strane servera ili povezanih sa izvršnim okruženjem trenutne skripte. Pre nego što se uspostavi kontrola nad skriptom, PHP postavlja niz vrednosti u vezi sa serverom, okruženjem, i zahtevima od strane klijenta. Sve one su smeštene u `$_SERVER` superglobalnoj varijabli.

U sledećoj tabeli je prikazano samo nekoliko karakterističnih varijabli iz niza smeštenog u **\$ SERVER**

Ime	Vrednost
HTTP_REFERER	Ako je korisnik kliknuo na link da bi došao do trenutne stranice, ovo će sadržati URL prethodne strane na kojoj je bio ili će biti prazno ako je direktno uneo adresu
HTTP_USER_AGENT	Ime korisnikovog browser -a
PATH_INFO	Sve podatke unete u URL – u nakon imena skripte
PHP_SELF	Ime trenutne skripte, koja se izvršava
.....	

\$ GET i \$ POST

Opisane u delu o formama

\$ COOKIE

Sadrži informacije iz HTTP kukija, detaljnije obrađeno u delu o kukijima

\$ FILES

Asocijativni niz varijabli poslatih u PHP fajl putem HTTP POST metode.

\$ ENV

Sadrži informacije o okruženju. Ove varijable su importovane u PHP globalni namespace iz okruženja u kojem radi PHP paser.

\$ SESSION

Asocijativni niz varijabli sa informacijama registrovanim u okviru sesije. Detaljnije obrađeno u delu o sesijama

\$ REQUEST

Sve informacije dobijene od strane korisnika. Sadrži sve varijable prikupljene preko GET, PUT, COOKIE metoda.

6.6 PHP napredne funkcije za rukovanje cookies

Cookie se obično koristi za identifikaciju korisnika, zbog prirode HTTP protokola koji ne čuva podatke o interakciji klijent/server. Cookie je mali fajl koji server "prikači" za računar korisnika. Cookie se šalje u okviru HTTP zahteva browser-a. Cookie se može kreirati pomoću JavaScript-a, ali često se cookie-ima u potpunosti upravlja preko servera. Ovo znači da se cookie šalje u Set-Cookie header polju kao deo HTTP odgovora:

```
HTTP/1.0 200
Content-Length: 1276
Content-Type: text/html
Date: Tue, 06 Nov 2001 04:12:49 GMT
Expires: Tue, 06 Nov 2001 04:12:59 GMT
Server: simvebs/3.1.6
Set-Cookie: animal=egg-laying-mammal

<html>...</html>
```

Kada veb server primi odgovor, u sledećem HTTP zahtevu ukjučuje cookie u header polju Cookie:

```
GET /duck/bill.php HTTP/1.0
Connection: Keep-Alive
Cookie: animal=egg-laying-mammal
Host: www.vebdatabasebook.com
Referer: http://www.vebdatabasebook.com/
```

U PHP - u, mogu se i kreirati i čitati vrednosti cookie fajla. Za potrebe kreiranja cookie fajlova, PHP koristi ugrađenu (*built – in*) funkciju `setcookie()`. Sintaksa:

`setcookie(ime, vrednost, datum_isteka (u sekundama), putanja, domen)`

Prva dva parametra `setcookie()` funkcije su obavezna, dok su ostala 3 opciona.

Parametar	Opis	Primer
ime	Ime koje koristi cookie za skladištenje ili izvlačenje informacija	username
vrednost	Vrednost smeštena u cookie.	Internet

Datum_isteka	Unix timestamp kad ističe cookie, ako nije podešeno ističe čim se zatvori browser	Time()+60*60*24*7 - ističe za nedelju dana
putanja	URL putanja na sajtu koja može da pristupi cookie – u. Default znači da svaki direktorijum može pristupiti cookie - u	/testing
domen	Slično putanji, samo što pristup može biti ograničen na poddomen sajta,	Da bi se ograničio pristup samo na www na sajtu example.com koristi se www.example.com . Za pristup svim domenima sajta koristi se .example.com.
sigurnost	Ako je podešeno na 0, cookie se šalju samo preko HTTPS connection. HTTPS konekcija koristi enkripciju između klijenta i servera	0 za sigurnu konekciju i 1 za nesigurnu, što je i default vrednost

Za potrebe čitanja vrednosti cookie fajlova, PHP koristi globalnu promenljivu `$_COOKIE[]`, koja kao parametar uzima ime (*name*) parametar funkcije `setcookie()`.

Funkcija `isset()` proverava da li je vrednost cookie – ja podešena.

Takođe treba napomenuti da je neophodno PHP kod koji sadrži `setcookie()` funkciju staviti pre HTML koda (pošto se cookie šalje, odnosno razmenjuje u okviru header-a), da bi se vrednosti mogle upisivati u cookie fajl i čitati iz njega.

U sledećem primeru dat je prikaz `setcookie()` funkcije.. Datum isteka postavljen je na 1 čas (3600 sekundi).

```
<?php
setcookie("ime", "imekukija", time()+3600);
// postavljen je cookie cije je ime - ime, vrednost - imekukija i datum
isteka 1 sat
?>
<?php
if (isset($_COOKIE['ime'])) {
// proverava da li je za ime - ime podesena neka vrednost u cookie - ju
echo "Dobrodosla" . " " . $_COOKIE['ime'];
}
else {
echo "Dobrodosao posetioce";
}
?>
<html>
<head> <title> Primer za Cookie </title> </head>
<body>
</body>
</html>
```

Ako postoji potreba da se izbriše cookie pre isteka vremena, ponovo se koristi funkcija `setcookie()` i kao argument `time()` navodi se negativno vreme, npr. `setcookie("ime", "Internet", time()-3600)` ili se pozove funkcija `setcookie("ime")`, sa samo jednim parametrom.

U sledećem primeru se preko kukija čita vreme poslednje posete sajtu.

```
<?php
$Month = 2592000 + time();
//this adds 30 days to the current time
setcookie(AboutVisit, date("F jS - g:i a"), $Month);
?>

<?php
if(isset($_COOKIE['AboutVisit']))
{
    $last = $_COOKIE['AboutVisit'];
    echo "Welcome back! <br> You last visited on ". $last;
}
else
{
    echo "Welcome to our site!";
}
```

6.6.1 Slanje header-a

HTTP je protokol za prenos podataka između servera i klijenta (*browser*). Pored podataka koje vidimo u *browser*-u kao što su tekst i slike ovim protokolom se istovremeno šalju i podaci koji se ne vid. Ovi podaci se nazivaju *header*-i i oni sadrže informacije o serveru, *browser*-u, jeziku, tipu podataka koje podržavaju i sl. Primer hedera koje šalje *browser*:

```
Host: www.bigcompany.com
User-Agent: Mozilla 4.0 (Compatible; MSIE; 5.5)
Accept: text/*, text/html, text/html;level=1, */*
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

Header-e možemo grubo podeliti u dve grupe: *Request header*-e koje *browser* šalje kada se zahteva neki fajl sa servera i *Response header*-e koje šalje server kada isporučuje taj fajl. Na osnovu tih podataka možemo zaključiti koji *browser* korisnik koristi, koje servise podržava uređaj korisnika i sl.

U narednom kodu se vidi kako se u PHP-u čitaju podatke iz header-a:

```
$accept = $_SERVER["HTTP_ACCEPT"];
$user_agent = $_SERVER["HTTP_USER_AGENT"];
$accept_charset = $_SERVER["HTTP_ACCEPT_CHARSET"];
$accept_language = $_SERVER["HTTP_ACCEPT_LANGUAGE"];
$x_wap_profile = $_SERVER["HTTP_X_WAP_PROFILE"];
$profile = $_SERVER["HTTP_PROFILE"];
```

Funkcija `header()` omogućava slanje stringova preko HTTP header-a

```
int header(string string);
```

Format je:

```
"header_name: header_value"
```

Na primer za redirekciju *browser*-a na drugu stranicu, koristi se sledeći zapis:

```
header("Location: http://www.mylab.net");
```

Ili da bi se sprečilo keširanje stranica

```
header("Cache-Control: no-cache, must-revalidate");
```

Treba napomenuti da sve što se nalazi u header-u, mora biti poslato browser-u pre bilo kakvog ispisa u HTML

6.7 PHP napredne funkcije za sesijama

Koncept HTTP protokola kao protokola “bez stanja” veoma često može da stvori probleme kompleksnim aplikacijama koje podrazumevaju određen vid interakcije sa korisnikom. Takvi sajtovi se ne mogu implementirati kao niz nepovezanih stranica, odnosno niz stranica koje ne čuvaju stanje. Običan HTML sajt ne prenosi podatke sa jedne stranice na drugu. Drugim rečima, sve informacije se zaboravljaju čim se učita nova stranica. Ovo može da bude veliki problem, posebno u slučajevima kao što je “potrošačka korpa”, kada je neophodno imati sve podatke (npr. koje proizvode je kupac uzeo), bez obzira da li su kupljeni na prvoj ili poslednjoj stranici sajta. Svi selektovani proizvodi moraju biti zapamćeni na neki način, pre dodavanja sledećeg u nizu.

Sesija predstavlja apstraktni koncept, koji čini niz HTTP zahteva i odgovora između određenog veb servera i veb browser-a. Sesija je veoma korisna u veb aplikacijama kada je neophodno deliti i prenositi informacije između stranica. Pošto HTTP kao protokol ne podržava ovaj koncept, svi server-side jezici su razvili tehnologije za upravljanje sesijama.

Postoje tri ključne stvari koje treba posmatrati kada su u pitanju sesije na vebu.

- Informacije o stanju moraju biti negde zapamćene.
- Svaki HTTP zahtev mora sadržati nekakav identifikator koji omogućava serveru da obradi zahtev sa svim skladištenim (zapamćenim stanjima). Na primer, kada korisnik završi kupovinu, podaci o svim proizvodima u korpi, kao i podacima o korisniku moraju biti na raspolaganju serveru.
- Sesije moraju imati period trajanja (timeout)

PHP rešava ovo pitanje, omogućavajući skladištenje informacija o korisniku na serveru (npr. ime, username, selektovani proizvodi i sl.). Uglavnom, informacije iz sesija se ne čuvaju predugo, ali u potpunosti je izvodljivo čuvati ih u nekoj bazi.

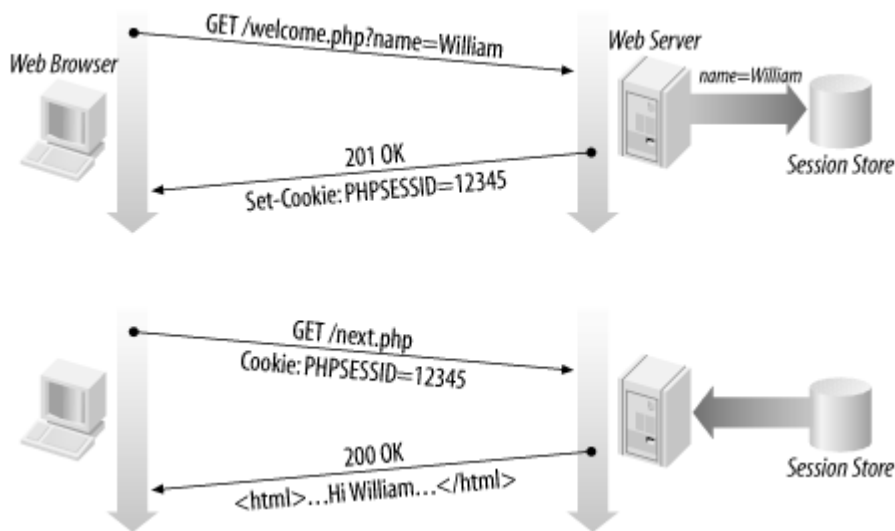
Ključno pitanje je na koji način identifikovati sesiju i održati tzv. Session ID (identifikator, id sesije).

PHP sesije rade tako što kreiraju jedinstveni id tzv. UID (Unique ID) za svakog posetioca veb sajta i čuvaju korisničke promenljive koje su zasnovane na UID. UID se čuva ili u cookie fajlu ili u URL – u (Uniform Resource Locator).

Na sledećoj slici je prikazano kako PHP upravlja sesijama:

- 1) Kada korisnik zahteva određenu stranicu u koju je ugrađen concept sesije, PHP generiše id sesije i kreira fajl koji čuva sve varijable vezane za sesiju.
- 2) U okviru odgovora server postavlja cookie sa id sesije
- 3) Browser pamti cookie i šalje ga u sledećem zahtevu

U primeru welcome.php pamti varijable sesije na odgovarajućoj lokaciji i one postaju dostupne i u svakom sledećem zahtevu (next.php)



Da bi se koristila u PHP – u, sesija mora da se aktivira. Sesija se otvara pomoću session_start() funkcije. Ova funkcija u PHP – fajlu se mora navesti pre HTML taga, odnosno pre <html>. Istovremeno, funkcija session_start() može da služi i za nastavak postojeće sesije na osnovu id sesije poslatog u zahtevu za stranicom u kojoj je ova funkcija. Njenim pozivom se automatski aktiviraju sve varijable sesije (ako je upitanju nastavak postojeće)

Kada se prvi put pozove funkcija session start(), osim kreiranja id sesije, automatski se u odgovoru generiše i polje Set-Cookie. Ubacuje se cookie bez potrebe za eksplicitnim pozivanjem funkcija setcookie() ili header().

Identifikator sesije je dugačak string koji se sastoji iz 32 heksadecimalne cifre (fcc17f071bca9bf7f85ca281094390b4) i smešten je u sistemskoj PHP varijabli pod imenom PHPSESSID.

Direktorijum u kojem se čuvaju fajlovi sesija se definiše u session.save_path u php.ini fajlu. Po default-u, fajlovi se čuvaju u /tmp direktorijumu, koji će u ovom slučaju izgledati /tmp/sess_fcc17f071bca9bf7f85ca281094390b4.

Kada se u PHP stranici potraži session_start(), a pri tom PHPSESSID se nalazio u HTTP zahtevu, PHP pokušava da pronađe fajl sesije u koji su skladištene sve varijable sesije. Ako ne postoji fajl pridružen ovoj sesiji, kreira se novi.

Nakon startovanja sesije i prelaska na bilo koju drugu stranicu, PHP prenosi id sesije na svaku stranu ili preko kukija ili preko PHPSESSID kao GET varijable.

Postoje dva načina za prenos i održavanje identifikatora sesije sa stranice na stranicu:

- Cookies
- URL parametar

Cookies su optimalno rešenje, ali se može dogoditi da nisu omogućeni. Drugi način podrazumeva ubacivanje id sesije direktno u URL.

Moguće je koristiti i konstantu SID, koja se automatski definiše kada se sesija startuje. Ova konstanta ima oblik `session_name=session_id`.

Sve varijable vezane za jednu određenu sesiju se mogu pročitati preko globalne promenljive `$_SESSION`. U narednom primeru će biti prikazano kako se uz pomoć `$_SESSION[]` promenljive može videti koliko puta je određena veb strana pregledana od strane korisnika. Sesija započinje pomoću funkcije `session_start()` i na taj način se inicijalizuje niz varijabli u okviru superglobalne `$_SESSION[]`. Nakon toga je moguće kreirati varijable i dodeljivati im vrednosti. Sesija mora biti startovana pre bilo kakvog ispisa na ekranu, usled ograničenja HTTP protokola, koji zahteva slanje sesija u okviru header-a.

Brojanje pregleda stranica

```
<?php
session_start();
?>
<html>
<head> <title> Primer za $_SESSION() promenljivu </title> </head>
<body>
<?php
if (isset($_SESSION['view'])) {
    $_SESSION['view'] = $_SESSION['view'] + 1;
}
else {
    $_SESSION['view']=1;
}
echo "Stranica je pregledana" . " " . $_SESSION['view'] . " " . "puta";
?>
</body>
</html>
```

U sledećem primeru se vidi suština koncepta sesije. Prelaskom sa jedne na drugu stranicu ne gubi se vrednost promenljive *ukupaniznos*.

Sabiranje potrošenog novca

prva.php

```
<?php
// pocetna stranica
session_start();
$_SESSION["ukupaniznos"] = 1;
print "<a href='druga.php'> Prelaskom na sledecu stranicu povecavate iznos
za 5 dinara</a>";
?>
```

druga.php

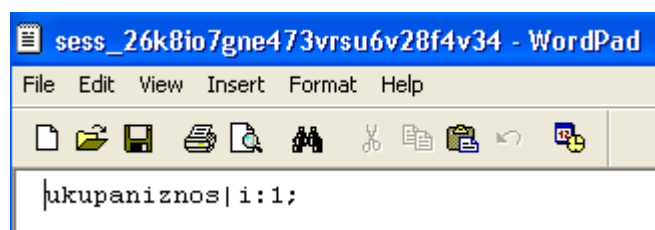
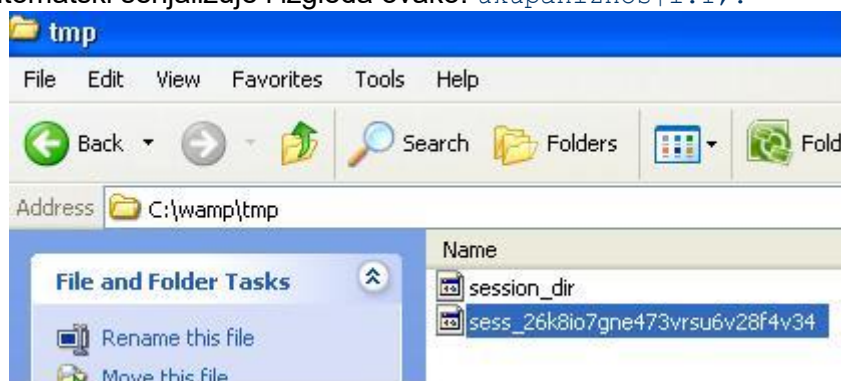
```
<?php
```

```
// druga.php
session_start();
$_SESSION["ukupaniznos"]=$_SESSION["ukupaniznos"] + 5;
print $_SESSION["ukupaniznos"];
?>
```

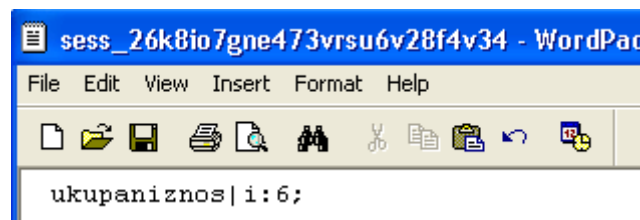
Treba primetiti da se vrednost varijable ukupan iznos povećala prilikom prelaska na stranicu druga.php. Sesija se vezuje za skup svih stranica koje se posećuju u okviru nje, a ne samo na pojedinačne stranice. Prilikom poziva funkcije session_start(), ne inicijalizuje se nova sesija, nego se samo ponovo aktivira postojeća sa svim svojim varijablama. Na slici se vidi cookie, koji sa sobom nosi id sesije.



U ovom primeru se nakon izvršavanja prve varijable njena vrednost čuva u fajlu sesije tako što se ona automatski serijalizuje i izgleda ovako: `ukupaniznos|i:1;.`



Nakon poziva druge stranice ona u serijalizovanom obliku izgleda ovako



Da bi varijable sesije ostale zapamćene, neophdno ih je ili postaviti preko globalne varijable `$_SESSION` ili registrovati pomoću funkcije `session_register()`. Na taj način, one se pamte u definisanom fajlu:

```
// Registruj varijablu "ime"
session_register("ime");
$ime = "internet tehnologije";
```

Kada je potrebno obrisati određenu varijablu iz sesije koristi se funkcija `unset()`

```
unset($_SESSION["session_name"]),
```

ako treba obrisati sve podatke vezane za sesiju, upotrebljava se funkcija `session_destroy()`;

Ukoliko bi se u prethodnom primeru koristio prenos id sesije preko URL deo koda:

```
print "<a href='druga.php'> Prelaskom na sledecu stranicu povecavate iznos  
za 5 dinara</a>";
```

bi se promenio u:

```
print "<a href='druga.php?' . SID . '> Prelaskom na sledecu stranicu  
povecavate iznos za 5 dinara</a>";
```

Kako proveriti da li sesija koristi cookie ili query string

Kada se sesija startuje, po default-u se id sesije skaldišti na klijentskoj mašini. Međutim,ukoliko računar korisnika ne podržava cookie, PHP šalje id sesije preko query stringa. URL će na primer izgledati ovako:

`http://www.example.com/file.php?SID=43b4a19d1962304012a7531fb2bc50dd`

Ovaj drugi način pamćenja i prenošenja id sesije nije preporučiv usled sigurnosnih ograničenja i pojedinih default podešavanja u konfiguracionim fajlovima PHP-a. Da bi se proverilo koji način sesija koristi za prenos id sesije, može se koristiti sledeći script:

```
<?
session_start();
```

```

if(isset($_COOKIE['PHPSESSID']))
{
    echo 'The session ID has been store in a cookie';
}
if (defined(SID))
{
    echo 'The session ID has been stored in the query string';
}
?>

```

Sve operacije vezane za sesije se moraju obaviti pre nego što se bilo kakav sadržaj pošalje korisniku. Ovo se može osigurati na više načina. Prvi je da se te operacije stave na sam početak skripte. U tom slučaju pre tih operacija ne sme biti nikakav kod, HTML ili prazan red. Ovo je u većini situacija vrlo teško postići. Drugi način za onemogućavanje slanja bilo kakvog sadržaja korisniku je korišćenjem output buffering funkcija. Output buffering je kontrolisanje slanja generisanog HTML koda korisniku. Znači da se na samom početku skripte može isključiti slanje outputa PHP koda (ili drugim rečima uključiti output buffering). Ovim se osigurava da se korisniku neće ništa poslati pre nego što treba. Ovom metodom se operacije sesije mogu obavljati bilo gde na stranici.

```
<? ob_start() ;// output buffering
```

Ovim je uključen output buffering i korisniku se neće slati nikakv sadržaj pre pojavljivanja ob_end_flush();

Upotreba sesija za autentikaciju

PHP sesije su idealne za veb sajtove koji zahtevaju logovanje pomoću korisničkog imena i šifre. Ovakvi sajtovi se veoma često sastoje iz većeg broja stranica i potrebno je informaciju u logovanju preneti i na druge stranice (ne očekuje se da se korisnik ponov loguje na svakoj stranici). Postupak u PHP bi mogao da izgleda ovako:

- 1) Pročitaju se podaci koje je korisnik uneo prilikom pokušaja logovanja na sistem
- 2) Vrš se proveru u bazi.
- 3) Ukoliko je korisnik uneo korektne podatke započinje se sesija i postavlja određena promenljiva sesije:
\$_SESSION['login']='go';
- 4) Kada god korisnik pređe na neku drugu stranicu, proverava se preko promenljive login da li je ulogovan
- 5) Ako je ulogovan prikazuje se sadržaj stranice, a ako nije korisnik se vraća na stranicu za logovanje

Da bi se izvršila provera da li je korisnik ulogovan ili ne, korsiti se sledeći kod:

```

<?php
session_start();
if ($_SESSION['login'] != "go" ) {
header("Location: loginPage.php");
exit();
}
else
if ($_SESSION['login'] = "go" )
{
//Prikaži sadržaj stranice

```



```
}  
?>
```

Naravno, neopohodno je u okviru sesije voditi računa i o tome da li je korisnik preduzeo akciju logout i u skladu sa tim podesiti i odgovarajuću promenljivu u sesiji.

6.8 PHP napredne funkcije za upravljanje greškama

Upravljanje greškama i izuzecima predstavlja važan aspekt programiranja. PHP sadrži veliki broj funkcija za ove potrebe.

Primer:

```
<?php  
    $a = 5;  
    $b = 0;  
    $c;  
    podeli($a, $b, $c);  
    function podeli($a, $b, &$amp;$c) {  
        if (!$b) {  
            throw new Exception('Deljenje sa nulom.');        }  
        $c = $a / $b;  
    }  
    echo $c;  
?>
```

Kao što se može videti, potrebno je podeliti dva broja, međutim, imenilac je nula, i u tom slučaju deljenje nije moguće. U ovom slučaju je potrebno da ukoliko je imenilac program “baci” grešku i onemogući dalje izvršavanje našeg koda.

Takođe ukoliko se neka greška “baci” ona može i da se “uhvati” i “obradi”. To je omogućeno preko “try / catch” bloka gde se pogramu kaže prvo se proba da se neki kod izvrši, a ukoliko se desi da on “baci” grešku, ta greška će biti “uhvaćena” u catch delu gde će se ona i obraditi. Pogledajmo to na prethodnom primeru:

```
<?php  
    $a = 5;  
    $b = 0;  
    $c;  
  
    try{  
        podeli($a, $b, $c);  
    } catch(Exception $ex) {  
        echo $ex->getMessage();  
    }  
}
```

```

function podeli($a, $b, &$c) {
    if (!$b) {
        throw new Exception('Deljenje sa nulom.');
```

Pored “try/catch” postoji i blok “finally” u kojem se stavljaju neke delovi koda, koji treba da se u svakom slučaju izvrše bez obzira da li je došlo do greške ili ne. Naročito se može videti korist pri konekciji sa bazom ili sa nekim drugim servisima, gde je potrebno osigurati bez obzira na to da li je došlo do greške ili ne da se konekcija ka bazi ili servisima oslobodi.

Kao što se može videti izuzeci nisu ništa drugo nego obična klasa koja ima razne predefinisane metode u sebi. Postoji čitava hijerarhija izuzetaka u php-u koja je dosta slična izuzecima u ostalim programskim jezicima. Više detalja se može videti na sledećoj adresi: <http://php.net/manual/en/language.exceptions.php>.

Klasa Exception je nadklasa svih ostalih klasa koje služe za obradu grešaka i sve te klase moraju da naslijeđe klasu Exception. Na taj način klasa Exception može da “uhvati” bilo koju grešku koja je nastala u mom kodu.

Naravno postoji i mogućnost da se kreira sopstvena klasa za obradu grešaka, koja će u sebi imati sve metode potrebne da bi se neka greška obradila.

```

<?php
class MojIzuzetak extends Exception { }

class Proba {
    public function testiranje() {
        try {
            try {
                throw new MojIzuzetak('poruka1!');
```

Rezultat izvršavanja ovog koda je sledeći:

string(8) "poruka1!"

Važan segment PHP koda je upravljanje greškama (tzv. *error handler*). Default *error handler* u PHP – u je jako jednostavan. Poruka o grešci se šalje do browser - a sa nazivom php fajla u kome se greška nalazi, broju linije koda gde je greška i poruci koja opisuje prirodu greške.

Kada se prave skripte i veb aplikacije, rukovanje greškama je nezaobilazan postupak. Ako programskom kodu nedostaje provera grešaka, aplikacija izgleda neprofesionalno i može biti izložena rizicima sigurnosti.

U nastavku teksta, biće prikazana tri metode za upravljanje greškama:

- Jednostavna `die()` naredba
- Personalizovane funkcije za upravljanje greškama i okidači grešaka
- Izvestavanje o greškama

Najjednostavniji način za upravljanje greškama (error handling) je funkcija `die()`, koja je ilustrovana sledećim primerom.

```
<html>
<head> <title> Primer za error handling - funkcija die() </title> </head>
<body>
<?php
if (!file_exists("dobrodosli.txt")) {
    die ("Fajl dobrodosli.txt ne postoji na Vasem file sistemu");
}
else {
    $fajl=fopen("dobrodosli.txt", "r");
}
?>
</body>
</html>
```

U prethodnom primeru se vidi da ako fajl *dobrodosli.txt* ne postoji, u browser – u, se ispisuje poruka da fajl ne postoji, a ako fajl postoji, fajl se otvara.

Drugi način za upravljanje greškama (*error handling*) je uz pomoć personalizovanih funkcija za upravljanje greškama i okidačima grešaka (*error triggers*). Kreiranje personalizovanih funkcija za upravljanje greškama je jednostavan postupak. Ta funkcija se poziva svaki put kada se naiđe na grešku u PHP – u. Personalizovana funkcija za upravljanje greškama prima najmanje dva parametra - error level (nivo greške) i error message (poruka koja se ispisuje kada se greška javi), a, najviše pet parametara. Poslednja tri parametara su opcioni. Sledeća tabela prikazuje listu parametara koja može primiti personalizovana funkcija za upravljanje greškama:

Parametar	6.8.1.1.1 Opis
<code>error_level</code>	Nephodan parametar. Određuje tip izveštaja o greški za korisničko – definisanu grešku. Ovaj parametar mora biti ceo broj.
<code>error_message</code>	Nephodan parametar. Određuje poruku kada se greška javi za korisničko

	– definisanu grešku.
error_file	Opcioni parametar. Ispisuje ime fajla u kome se greška javila.
error_line	Opcioni parametar. Ispisuje broj linije koda gde se greška javila.
error_context	Opcioni parametar. Ispisuje imena svih promenljivih i njihovih vrednosti koje su bile aktivne kada je došlo do greške

Sledeća tabela prikazuje tip izveštaja o grešci, tj. broj koji se definiše u parametru error_message.

Vrednost	Konstanta	Opis
2	E_WARNING	Nije fatalna run – time greška. Izvršenje skripte se nastavlja.
8	E_NOTICE	Run – time primedba. Primećeno je nešto u toku izvršavanja koda što može biti , ali i ne mora biti greška.
256	E_USER_ERROR	Fatalna korisnički – definisana greška.
512	E_USER_WARNING	Nije fatalna korisnički – definisana greška.
1024	E_USER_NOTICE	Korisnički – generisana primedba.
4096	E_RECOVERABLE_ERROR	Uhvatljiva (catchable) fatalna greška.
8191	E_ALL	Sve greške i upozorenja.

U narednom primeru prikazana je personalizovana funkcija za upravljanje greškama (*error handling*). Prvo će biti kreirana takva funkcija koja će prikazivati nivo greške (error level) i poruku o grešci:

```
<html>
<head> <title> Primer za error handler - personalizovana greska </title>
</body>
<body>
<?php
function personalizovanaGreska($errno, $errstr) {
    echo "<b> Greska: <b> [$errno] $errstr ";
}
// sada ce se postaviti error_handler
set_error_handler("personalizovanaGreska");
// okidanje, tj. nailazenje na gresku,prikazace se vrednost promenljive
koja ne postoji
echo ($test);
?>
</body>
</html>
```

U nastavku, biće prikazan još jedan primer u kome ako je dati broj veći od 2, izvršavanje skripte se prekida pomoću trigger_error() funkcije:

E_USER_WARNING će se aktivirati, ako je test promenljiva veća od 1. Ako se aktivira E_USER_WARNING, personalizovana funkcija za upravljanje greškama biće aktivirana da bi se izvršenje skripte prekinulo:

```
<html>
<head> <title> Primer za personalizovani error handler(3) </title> <head>
<body>
<?php
function personalizovanaGreska($errno,$errstr)
```

```
{
    echo "<b> Greska </b> [$errno] $errstr";
    echo "Kraj skripte";
    die();
}
set_error_handler("personalizovanaGreska",E_USER_WARNING);
// okidanje greske
$test=2;
if($test>1){
    trigger_error("vrednost mora biti veca od 1", E_USER_WARNING);
}
?>
</body>
</html>
```

7 Uvod u forme

Jedna od ključnih stavki u kreiranju aplikacija je prikupljanje podataka od korisnika. Ovo se najčešće rešava pomoću HTML formi. Korisnik unosi različite vrednosti u formu i nakon selektovanja opcije za submit forme, browser formatira podatke i šalje zahtev ka serveru. Između ostalog u zahtevu se nalazi i naziv fajla na serveru koji bi trebalo da obradi zahtev.

Zbog prirode PHP-a kao serverskog skriptnog jezika za dinamičku manipulaciju web sadržajem koja u mnogim situacijama zavisi od korisničkog unosa ili korisničkih akcija (klikatanje na linkove i slične akcije), rad sa formama predstavlja jedno od najbitnijih oblasti.

Forme se koriste u velikom broju web aplikacija, nezavisno u kom je jeziku aplikacija napisana. Zbog ovoga je potrebno imati znanje samih HTML formi (tipove polja, koja su za koju namenu najbolja, načine grupisanja elemenata ...).

Uz poznavanje HTML dela formi, potrebno je znati kako pomoću PHP-a prihvatiti i obraditi vrednosti koje su unesene u formu.

7.1 O formama uopšteno

Sa tehničke strane gledano, forma je HTML element pomoću kojeg se grupiše više elemenata za unos podataka (tipa text box, select izbornik, opcije...). U elemente forme korisnik može uneti (ili izabrati) informacije koje se kasnije koriste za bilo koju svrhu (unos informacija u bazu podataka, slanje e-maila sa unesenim informacijama ...). Razlog grupisanja više elemenata za unos podataka u jednu formu je više – manje očigledan.

Uglavnom, kada se vrši nekakav unos na web stranici, obično se radi o više povezanih informacija koje čine celinu (primer : login forma sa username - om i password - om). Pošto su elementi forme grupisani šalju se na obradu zajedno kada se forma submit - uje.

Primer 1 : Umetanje forme u HTML dokument

```
<html>
<head>
<title>Primer 1 : Umetanje forme u HTML dokument</title>
</head>
<body>
<h1>Primer 1 : Umetanje forme </h1>
<hr>
<p>Ispunite formu : </p>
<form method="post" action="prihvat.php" name="formal">
<input name="submitdugme" type="submit" value="Posalji">
</form>
</body>
</html>
```

Forma se predstavlja preko jednog od osnovnih HTML tagova koji ima svoj početak i kraj, i određene attribute.

Prvi atribut je method="post". Definiše metodu kojom će se podaci iz forme proslediti dokumentu (skripti) za obradu forme.

Drugi atribut je `action="prihvat.php"`. Preko action atributa se definiše fajl na serveru, koji će obraditi zahtev

Treći atribut je `name="forma1"`. Ovaj argument nije bitan za prihvatanje i obradu podataka forme pomoću PHP-a. Važnost ovog argumenta se više ogleda u izradi JavaScript klijentskih kodova, i služi za identifikaciju ove forme.

Preko input taga se kreiraju različiti tipovi elemenat koji omogućavaju unos od strane korisnika. Ukoliko je input elemenat tipa submit, na ekranu je prikazan kao dugme I što je najvažnije klik na to dugme generiše događaj onsubmit, odnosno dovodi do submit-ovanja forme.

7.2 Metode slanja i prihvatanja podataka

Dve osnovne metode prosleđivanja podataka forme nekom dokumentu (skripti) su POST i GET. Odabir metode prosleđivanja podataka forme se vrši navođenjem method argumenta `<form>` taga. Moguće vrednosti metod argumenta su "post" i "get".

Odabirom metode se utiče na koji način će se sami podaci forme proslediti stranici, i to je nezavisno od jezika u kojem će se ti podaci prihvatiti i obraditi. U PHP-u odabirom metode se bira i način na koji će biti prihvaćeni podaci.

Klasična dilema u web aplikacijama je da li koristiti GET ili POST metodu. Pravi odgovor je teško definisati.

Prva razlika između ovih metoda je najdrastičnija i odmah uočljiva. Kada se odabere metoda GET podaci forme se šalju kroz komandnu liniju (query string, tj. iza znaka ? u adress baru browsera). Podaci se prenose u obliku ključ=vrednost. Na primer:

[http://www.myelab.net/php/userform.php?username=internet,](http://www.myelab.net/php/userform.php?username=internet)

znači da će se izvršiti fajl userform.php u php direktorijumu na serveru myelab, a promenljiva koja se šalje je username sa vrednošću Internet.

Odabirom metode POST podaci nisu vidljivi u komandnoj liniji već se šalju transparentno kroz **telo** (body) HTTP zahteva (HTTP request). POST-om se podaci forme šalju kroz request body i time se na njih ne može uticati izmenom linka u adress baru. Takođe, postoji i ograničenje na količinu podataka koji se mogu poslati putem GET metode. Ograničenje zavisi od postavki samog servera na kom se nalazi dokument na kom se procesira forma i varira od 256 byte-ova do čak 32 Kb. Zato, ukoliko se radi o formama koje u sebi sadrže polja za unos velikih tekstova ili sličnih podataka GET i nije adekvatna metoda za prosleđivanje informacija.

Metoda koja se koristi za prosleđivanje informacija zavisi od situacije do situacije, mada se za obradu formi češće koristi POST metoda.

U GET metodi informacije stavljaju u URL, što ovu metodu čini idealnom u slučaju kada želite omogućiti posetiocima sitea da ubace stranicu koju gledaju u svoje favorites, jer će staviti URL zajedno sa prikazanim query stringom.

GET je prilično nesigurna metoda jer posetilac vrlo lako može izmeniti zahtev jednostavnom promenom URL-a u adress baru svog browsera, tako da nije preporučljivo koristiti ovu

metodu za prosleđivanje recimo usernamea i passworda u login formama i sličnih osetljivih informacija. Jedino pravilo koje se mora poštovati je da se obavezno mora koristiti POST metoda kada se preko forme upload-uje određen fajl.

Kada korisnik popuni formu na nekom sajtu i klikne na Submit dugme, browser šalje serveru GET ili POST zahtev, u zavisnosti od vrednosti atributa method pri deklarisanju forme.

Na primer, HTML kod forme na slici je sledeći :

Ime:

Prezime:

```
<form method="post" action="obrada.php">  
  Ime:<br/>  
  <input type="text" name="ime" /><br/>  
  Prezime:<br/>  
  <input type="text" name="prezime" /><br/>  
  <input type="submit" /><br/>  
</form>
```

Nakon klika na taster submit, šalje se http zahtev, koji se razlikuje u zavisnosti da li se koristi GET ili POST metoda.

Primer POST zahteva prosleđenog sa forme (nakon klika na submit):

```
POST /obrada.php HTTP/1.1  
Host: www.mojserver.com  
Content-Type: application/x-www-form-urlencoded  
Content-Length: [veličina tela zahteva]
```

```
ime=Pera&prezime=Peric
```

Primer GET zahteva prosleđenog sa forme (nakon klika na submit):

```
GET /obrada.php?ime=Pera&prezime=Peric HTTP/1.1  
Host: www.mojserver.com  
Content-Length: [veličina tela zahteva]
```

Sledeći primer se sastoji iz dva dela. Radi se o istoj formi, samo što se u slučaju a) koristi GET metoda, a u slučaju b) POST.

Primer se sastoji od dva dokumenta za oba slučaja. Prvi će biti nazvan forma.php, a drugi prihvat.php. Iz imena se zaključuje da će se u forma.php nalaziti sama forma, a u prihvat.php će se nalaziti PHP skripta koja će prihvatiti informacije i ispisati ih na ekran. Ovi primeri bi trebalo da ilustruju osnovnu metodu za prihvatanje podataka iz forme.

Primer a) Jednostavna forma pomoću GET metode

Dokument : forma.php


```

<html>
<head>
<title> Prosledivanje informacije GET metodom</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<form name="primer-a" method="get" action="prihvat.php">
Unesi svoje ime :
<input name="ime" type="text" >
<br>
<input name="submit" type="submit" value="Posalji">
</form>
</body>
</html>

```

Dokument : prihvat.php

```

<html>
<head>
<title> Prihvatanje podataka GET metode</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?php
echo "Pozdrav " . $_GET["ime"];
?>
</body>
</html>

```

Potrebno je uočiti samo da je u metod argumentu nameštena vrednost get. U ovom primeru je u formu ubačeno dva elementa. Radi se o <input> tagu koji ima više različitih tipova.

Prvi put je korišćen u svom text tipu :

```
<input name="ime" type="text" >
```

a drugi put se radi o tasteru :

```
<input name="submit" type="submit" value="Posalji">
```

U oba elementa je potrebno primeniti korišćenje name argumenta. Pomoću njega se daje polju ime i to ime se koristi u PHP-u za prihvatanje podataka koji su uneseni u to polje. Ukoliko se ne imenuje polje, podaci koji se unesu u njega nisu vidljivi PHP skripti koja je namenjena za prihvatanje podataka iz forme.

Prihvatanje podataka je sam po sebi vrlo jednostavan. Unutar PHP-u postoje superglobalne varijable. To su varijable (nizovnog tipa) koje definiše sam PHP pri pokretanju svake skripte.

\$_GET

Sadrži sve varijable poslate preko HTTP GET zahteva. Ovo se odnosi i na one poslate direktno preko URL - a (http://www.example.com/page.php?id=2) i preko submit – ovanja forme korišćenjem GET metoda.

```
echo "Pozdrav " . $_GET["ime"];
```

Gornja linija ispisuje poruku koja se sastoji od konstantnog dela Pozdrav i dela u kojem se ispisuje ono što je uneseno u text box polje u forma.php dokumentu koje je nazvano ime. Iz ovog jednostavnog primera je vidljivo da se podacima iz forme pristupa pomoću odgovarajuće superglobalne varijable. Vrednost nekog elementa forme se nalazi unutar superglobalne varijable na indeksu koji odgovara imenu tog elementa forme. U kojoj se superglobalnoj varijabli nalaze podaci zavisi o izabranoj metodi prosleđivanja podataka. U ovom primeru je metoda bila GET, tako da je korišćena superglobalna varijabla \$_GET.

Pre nego što se krene sa drugim delom primera, treba primetiti da su se podaci koji su ispunjeni u forma.php dokumentu zalepili na link u prihvati.php dokumentu koji ispisuje podatke.

```
http://localhost/phpbook/forme/primer2/prihvati.php?ime=MojeIme&submit=Posalji
```

Sve iza znaka ? u gornjem linku se naziva QUERY STRING. Radi se o seriji parova imena i vrednosti međusobno odvojenim znakom &.

Bilo bi moguće ovim načinom preneti podatke iz jedne stranice na drugu čisto putem linkova. Naime, PHP-u je svejedno da li se podaci u query stringu popunjavaju putem neke forme, ili se radilo o <a> tagu u kojem je u href argumentu napisan url skupa sa query stringom.

Znači, ovo bi bio ekvivalent formi iz primera, kada se u text box unese Marko

```
<a href='prihvati.php?ime=Marko'>Moje ime je Marko</a>
```

Čak je i izostavljena druga varijabla u query stringu pošto ona za sada nije potrebna u prihvati.php dokumentu.

Primer b) : Jednostavna forma pomoću POST metode

Dokument : forma.php

```
<html>
<head>
<title> Prosleđivanje informacije POST metodom</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<h2> Prosleđivanje informacije POST metodom </h2>
<hr>
<form name="primer-b" method="post" action="prihvati.php">
Unesi svoje ime :
<input name="ime" type="text" >
<br>
<input name="submit" type="submit" value="Posalji">
</form>
</body>
</html>
```

Dokument : prihvati.php

```
<html>
<head>
<title> Prihvatanje podataka POST metode</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
```

```

<?php
if(isset($_POST["ime"])){
echo "Pozdrav " . $_POST["ime"];
} else {
echo "Niste prosledili parametar";
}
?>
</body>
</html>

```

\$_POST

Sadrži sve varijable poslate preko HTTP PUT zahteva.

Princip prihvata podataka poslatih POST metodom je isti kao i kod prihvata podataka GET metode, sa manjim razlikama.

Prva razlika je u forma.php dokumentu, gde je promenjena metoda u POST.

```
<form name="primer2-b" method="post" action="prihvat.php">
```

Druga razlika je u već spomenutoj superglobalnoj varijabli koja je ovog puta \$_POST.

```
echo "Pozdrav " . $_POST["ime"];
```

Dodatna i očigledna razlika je da sada više nema query stringa i nije moguće direktno uticati na podatke nakon što je forma ispunjena i submitovana.

7.3 Provera metode pristupa dokumentu / skripti

U nekim situacijama je potrebno znati kojom metodom je neki dokument / skripta otvorena, pa na osnovu toga se obavljaju potrebne operacije. Ovo je moguće proverom vrednosti \$_SERVER["REQUEST_METHOD"] ili skraćeno \$REQUEST_METHOD varijable.

Njene moguće vrednosti su POST i GET, zavisno od metode otvaranja stranice. Njena vrednost će uvek biti GET osim u slučaju da se dokumentu pristupilo nakon submitovanja forme sa POST metodom.

Iz tog razloga ovo nije najbolja metoda za proveru da li je neko stvarno ispunio formu i pristupio dokumentu za prihvatanje i obradu podataka.

Kod search formi je dodatna, već spomenuta, prednost ta što se query string čuva zajedno sa URL-om u favorites posetioca, tako da je moguće opet se vratiti na isto pretraživanje nakon više dana jednostavnim odabirom bookmarka.

7.4 Forme i srpski znakovi

Problem neprikazivanja, tj. lošeg prikazivanja srpskih znakova leži u nepravilnom i nedoslednom korišćenju charseta unutar svih dokumenata skripte. Naime, pre izrade skripti dobra praksa je odlučiti se za jedan charset pa ga dosledno koristiti na svim dokumentima unutar sitea i time će biti izbegnuti svi problemi sa njima. Ovo pravilo vredi i za rad sa bazom podataka.

Prilikom deklarisanja HTML dokumenta, trebalo bi uvek koristiti utf-8 kodni raspored.

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

7.5 Prikaz i obrada unutar jednog dokumenta

U srednjim i većim aplikacijama je vrlo nepraktično imati odvojene dokumente za prikaz i obradu forme iz očiglednog razloga što bi se broj dokumenata nekog sajta popeo do tako velikog broja da bi postao problem praćenja šta se gde odvija unutar strukture dokumenata sajta.

Iz tog razloga je poželjno stvari koje su usko povezane, poput prikaza i obrada forme držati unutar istog dokumenta u najmanju ruku.

U primeru će biti korišćena ista forma i obrada iz b slučaja prethodnog primera.

Primer: Prikaz i obrada forme unutar jednog dokumenta

```
<html>
<head>
<title> Prikaz i obrada forme unutar jednog dokumenta</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<h2> Prikaz i obrada forme unutar jednog dokumenta</h2>
<hr>
<form name="primer3" method="post" action="<?=$_SERVER['PHP_SELF']?>">
Unesi svoje ime :
<input name="ime" type="text" >
<br>
<input name="submituj" type="submit" value="Posalji">
</form>
<?php
if ( !isset($_POST["submituj"]))
{
}
else {
echo "Pozdrav " . $_POST["ime"];
}
?>
</body>
</html>
```

Zapravo je eliminisana potrebu za dva odvojena dokumenta i sve se obavlja unutar istog dokumenta. Prva stvar koja se odmah uočava je

```
if ( !isset($_POST["submituj"]))
```

U ovom primeru se proverava postojanje vrednosti unutar superglobalne \$_POST varijable, a ukoliko se u njoj nalazi neka vrednost podrazumeva i da je \$REQUEST_METHOD == POST.

Ono što treba primetiti u primeru je da se proverava postojanje vrednosti za submit taster unutar forme. To se radi zato što je ta vrednost obavezno prisutna ukoliko se forma ispunila i submit - ovala.

Logika primera je poprilično jednostavna. Ukoliko ne postoji vrednost u `$_POST["submituj"]` znači da forma nije još ispunjena, a ukoliko ta vrednost postoji znači da je forma ispunjena i submito - vana, pa se ide na obradu podataka.

```
<form name="primer3" method="post" action="<?=$_SERVER['PHP_SELF']?>">
```

`$PHP_SELF` je varijabla koju definiše sam PHP i u njoj se nalazi ime trenutno aktivnog dokumenta. Pošto se želi da se forma prikaže i obradi unutar istog dokumenta, to je tačno ono što je potrebno. Iako se tu moglo ručno upisati ime dokumenta, korišćenje `$PHP_SELF` je preporučljivo, zato što se ovako može promeniti ime dokumenta, a u njemu će sve i dalje raditi kako je zamišljeno, zato što PHP sam namešta njenu vrednost.

U slučaju da se želi imati u jednoj formi više submit tastera, u zavisnosti od toga koji je kliknut treba u obradi forme obaviti različite operacije. Na primer, kada se radi nekim administrativnim alatima, gde je potrebno napraviti više različitih operacija nad istim podacima.

Primer : Forme sa više tastera

```
<html>
<head>
<title> Forma sa vise submit tastera</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<h2> Forma sa vise submit tastera</h2>
<hr>
<form name="primer4" method="post" action="<?=$_SERVER['PHP_SELF']?>">
<input type="submit" name="taster1" value="Taster 1">
<br>
<input type="submit" name="taster2" value="Taster 2">
</form>
<?
if (isset($_POST["taster1"]))
echo "Pritisnuli ste taster 1";
if (isset($_POST["taster2"]))
echo "Pritisnuli ste taster 2";
?>
</body>
</html>
```

Ovog puta je kao uslov prikaza ili obrade forme korišćen `$REQUEST_METHOD`. On je korišćen zato što ovog puta postoje dva submit tastera, i pritiskom bilo kojeg od njih želi se umesto prikaza forme pokrenuti obradu iste.

Suština je u tome (a to je i cela poenta ovog primera) da će se nakon submit - ovanja forme pomoću jednog od tih tastera popuniti vrednost unutar superglobalne `$_POST` varijable samo onog tastera koji je pritisnut, što dokazuje deo koda koji obrađuje formu.

```
if (isset($_POST["taster1"]))
echo "Pritisnuli ste taster 1";
if (isset($_POST["taster2"]))
echo "Pritisnuli ste taster 2";
```

7.6 Elementi za unos podataka

Postoji očigledna potreba za više različitih načina prikupljanja podataka od korisnika. Do sada je u primerima korišćen samo text box element, radi jednostavnosti primera. Elementi HTML formi² :

- Text box
- Text area
- Sakriveno polje / hidden field
- Checkbox
- Radio taster
- Lista / meni – select izbornik
- Polje za upload datoteka

7.7 Text box polje

```
<input type="text" name="tekst">
```

Radi se o polju za unos jedne linije teksta. Ime pomoću kojeg mu se pristupa u PHP-u mu se navodi u name atributu. Ukoliko se želi da pri učitavanju stranice u ovom elementu nalazi neka vrednost može se navesti pomoću value atributa.

```
<input type="text" name="tekst" value="Ovo je defaultna vrednost polja i  
može se  
promeniti">
```

7.8 Text area

```
<textarea name="velikiTekst"></textarea>
```

Ovo je polje za unos teksta sa više redova. Korisnik može prebaciti unos teksta u novi red pa ga kao takvog poslati na obradu. Pravila oko imenovanja su ista kao i za sve ostale elemente. Za razliku od text boxa, text area nema value atribut, već se defaultna vrednost zadaje na sledeći način:

```
<textarea name="velikiTekst">Ovo je default vrednost elementa i može imati  
više redova </textarea>.
```

Kod davanja defaultne vrednosti je bitno paziti da će se 2 space znaka u samom elementu prikazati kao dva prazna prostora, pa da prelazak u novi red u HTML-u uzrokuje prebacivanje unosa podataka u elementu u novi red.

Korišćenje text area elementa

```
<html>  
<head>  
<title> Korišćenje text area elementa</title>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
</head>  
<body>  
<h2> Korišćenje text area elementa</h2>
```

```

<hr>
<?
if ( !isset($_POST["submituj"])){
?>
<form name="primer3" method="post" action="<?=$_SERVER['PHP_SELF']?>">
Unesi tekst od vise redova :<br>
<textarea name="tekst"></textarea>
<br>
<input name="submituj" type="submit" value="Posalji">
</form>
<?
} else {
echo "<b>Uneli ste : </b><br>" . $_POST["tekst"];
echo "<hr>";
// ispis sa obracanjem paznje na prelazak u nov red.
echo "<b>Tekst u izvornom obliku : </b><br>";
echo nl2br($_POST["tekst"]);
}
?>
</body>
</html>

```

Unešena vrednost u textarea elementu se nalazi unutar superglobalne varijable `$_POST` ili `$_GET`, zavisno od metode prosleđivanja podataka forme. Pošto je u ovom primeru metoda POST vrednost se nalazi unutar `$_POST` varijable.:

```
echo "<b>Uneli ste : </b><br>" . $_POST["tekst"];
```

Radi se o tome da HTML u svom source - u zanemaruje sve praznine, tj. više uzastopnih praznina interpretira poput jednog praznog prostora. Pod isto pravilo spada i prelazak u novi red u HTML sourceu. Ugrađena funkcija PHP - a je korišćena u drugom ispisu u primeru, a ispis unesene vrednosti prati isti format teksta koji je upisan u text area polje.

Korišćena funkcija je :

```
echo nl2br($_POST["tekst"]);
(nl2br – citaj new line to break)
```

Ime funkcije govori što ona radi. U stringu koji joj se da kao argument menja sve nove redove (`\n`) sa HTML `
` tagom koji prebacuje ispis u novi red pri gledanju dokumenta u browseru.

7.9 Hidden polje

```
<input type="hidden" name="nevidljivoPolje" value="Vrednost polja">
```

Kao što i samo ime elementa / polja kaže, radi se o polju koje nije vidljivo pri gledanju dokumenta u browseru, pa korisnik ne može direktno uticati na vrednost koja je sačuvana u njemu. Vrednost koje se nalazi u u takvom polju se mora namestiti direktno u source - u dokumenta ili unosom u HTML source ručno ili pomoću PHP-a. Drugi način promene vrednosti je dinamički, za vreme gledanja stranice u browseru putem JavaScripta. Kao i kod text boxa, vrednost se unosi u value atribut. Ovo polje obično služi za pamćenje nekih vrednosti koje ne zavise od korisnika (posetioca), pa se na ovaj način štite od korisničkog unosa i izmene.

7.10 Checkbox

```
<input type="checkbox" name="jezik" value="srpski">
```

Checkbox je element koji se uglavnom koristi u slučaju kada se korisniku želi omogućiti da izabere jednu, više ili ni jednu vrednost iz nekog logički povezanog skupa vrednosti. Recimo, u gornjem primeru bi značilo da korisnik priča srpski ukoliko je označio to polje, ili ga ne priča ukoliko ga je ostavio neoznačenim. Normalno, pošto se radi o grupi povezanih informacija trebalo bi biti ponuđeno više jezika pomoću više checkboxova. Iako gornja tvrdnja ne mora biti uvek tačna, recimo može se raditi o samo jednoj vrednosti, poput označavanja saglasnosti sa nečim ili sličnim da / ne situacijama.

Ovaj primer se može koristiti u situaciji u kojoj je potrebna potvrda posetioca da je punoletan i da je saglasan sa pravilima sajta

Primer a) : Prihvatanje checkbox elementa

```
<html>
<head>
<title> Prihvatanje checkbox elementa</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<h2> Prihvatanje checkbox elementa</h2>
<hr>
<?
if (! $_POST["ime"] ){
?>
<form name="form1" method="post" action="">
<p>
<input type="checkbox" name="punoletan" value="1">
Punoletan sam i saglasan sam sa pravilima sitea</p>
<p>
<input name="ime" type="submit" value="Ulaz">
</p>
</form>
<?
} else {
// proverava da li je označen checkbox
if ($_POST["punoletan"]) {
echo "Posto si punoletan mozes dalje koristiti ove stranice";
} else {
echo "Posto nisi punoletan ne mozes pristupiti sadrzaju ovih stranica";
die();
}
}
?>
</body>
</html>
```

Način prihvata informacije iz checkboxa je identičan onom iz prethodnih primera.

Primer b) : Prihvatanje vrednosti više povezanih checkbox elemenata

```
<html>
```



```

<head>
<title> Prihvatanje vrednosti vise povezanih checkbox elemenata</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<h2> Prihvatanje vrednosti vise povezanih checkbox elemenata</h2>
<hr>
<?
if ((!isset($_POST["ime"])) || !$_POST["ime"]){
?>
<form name="form1" method="post" action="">
<p>Koje od ponudjenih jezika pricate tecno?</p>
<p>
<input type="checkbox" name="jezik[]" value="engleski">
Engleski<br>
<input type="checkbox" name="jezik[]" value="nemacki">
Nemacki<br>
<input type="checkbox" name="jezik[]" value="francuski">
Francuski<br>
<input type="checkbox" name="jezik[]" value="ruski">
Ruski </p>
<p>
<input name="ime" type="submit" id="ime" value="Posalji">
</p>
</form>
<?
} else {
if ((isset($_POST["jezik"])) and (is_array($_POST["jezik"]) and count
($_POST["jezik"]) > 0 ))
{
echo "Jezici koje pricate su :";
foreach ($_POST["jezik"] as $pricam){
echo $pricam . ",";
}
} else {
echo "Jeste sigurni da ne pricate ni
jedan od navedenih jezika?";
}
}
?>
</body>
</html>

```

7.11 Upload fajla

PHP upravlja upload-om fajlova preko superglobalne varijable `$_FILES`, koja sadrži niz varijabli, poslatih preko POST metode.

`$_FILES['userfile']['name']` - originalno ime fajla na klijentskoj mašini

`$_FILES['userfile']['type']` - mime type fajla (npr. "image/gif").

`$_FILES['userfile']['size']` - veličina fajla u bajtovima

`$_FILES['userfile']['tmp_name']` – privremeni folder u kojem je smešten upload-ovani fajl

`$_FILES['userfile']['error']` – error code vezan za eventualnu grešku pri uploa-u

Da bi se omogućio upload fajla neophodno je podesiti atribut ENCTYPE attribute na "multipart/form-data" i action atribut ka PHP stranici koja će upravljati upload-om

```
<form enctype="multipart/form-data" action="upload.php" method="POST">
Odaberite fajl za upload
<input name="uploaded" type="file" /><br />
<input type="submit" value="Upload" />
</form>
```

Fajl se čuva u temporary direktorijumu. Fajl će biti uklonjen, odmah nakon zahteva, ako se ne sačuva, t.j ne kopira na alternativnu lokaciju. Fajlu se pristupa preko imena isto kao i bilo kom drugom input elementu. Funkcija koja se koristi za čuvanje je copy (). U novijim verzijama PHP, koristi se funkcija move_uploaded_file (string \$filename, string \$destination)

```
<?php
$target = "upload/";
$target = $target . basename( $_FILES['uploaded']['name']) //basename daje
ime fajla koji je uploadovan ;
if(move_uploaded_file($_FILES['uploaded']['tmp_name'], $target))
{
echo "Fajl je uspešno uploadovan";
}
else {
echo "Upload nije uspeo";
}
?>
```

8 Dinamički linkovi

GET metoda u PHP-u se osim za obrade podatke u formi može koristiti i za generisanje dinamičkih linkova koji sadrže različite parametre. Ovaj pristup korišćenja GET metode se u praksi češće sreće nego upotreba GET metode kod obrade forme. POST metodu nije moguće koristiti za generisanje linkova, pošto se njeni parametri ne prenose putem URL adrese. Internet pretraživači, kao što su Mozilla Firefox, Google Chrome i drugi, za slanje zahteva podrazumevano koriste GET metodu.

Generisanje dinamičkih linkova je karakteristično za dinamičke veb sajtove, tj. veb aplikacije. Oni se koriste kada je potrebno napraviti linkove ka vestima koje se skladište u bazi, kao i u slučaju definisanja više različitih HTML strana u jednom PHP dokumentu, što omogućava minimalne izmene u kodu u slučaju promene izgleda sajta.

Primer 8 : Dinamički linkovi u PHP-u

```
<html>
<head>
</head>
<body>
<h1>Moj sajt</h1>
<ul>
<li><a href="get-dinamicki-link.php?strana=pocetna">Početna</a></li>
<li><a href="get-dinamicki-link.php?strana=o-nama">O nama</a></li>
<li><a href="get-dinamicki-link.php?strana=kontakt">Kontakt</a></li>
</ul>
<?php
if (isset($_GET["strana"])){
    $strana = $_GET["strana"];
} else {
    $strana = "pocetna";
}

switch($strana){
    case "pocetna":
        echo "Dobrodošli na naš sajt!";
        break;
    case "o-nama":
        echo "Laboratorija za elektronsko poslovanje kao osnovni
cilj poslovanja ima unapređenje poslovnog, naučno-istraživačkog i
obrazovnog rada u oblasti primene informaciono komunikacionih tehnologija u
poslovanju, kao i projektovanja informacionih sistema u Internet okruženju.
Rad laboratorije zasniva se na primeni savremenih tehnologija i na
permanantnim inovacijama.";
        break;
    case "kontakt":
        echo "ELAB se nalazi na Fakultetu organizacionih nauka u
Beogradu. Adresa je Jove Ilića 154, kabinet 304.";
        break;
    default:
        echo "Greška 404: Stranica nije pronađena!";
        break;
}

?>
</body>
</html>
```

