UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

Elektronske lekcije o nekim naprednim mogućnostima programskog jezika PHP

MASTER RAD

Student
Dušan Kovačević

Mentor
prof. dr Miroslav Marić

Beograd
Septembar, 2018

Sadržaj

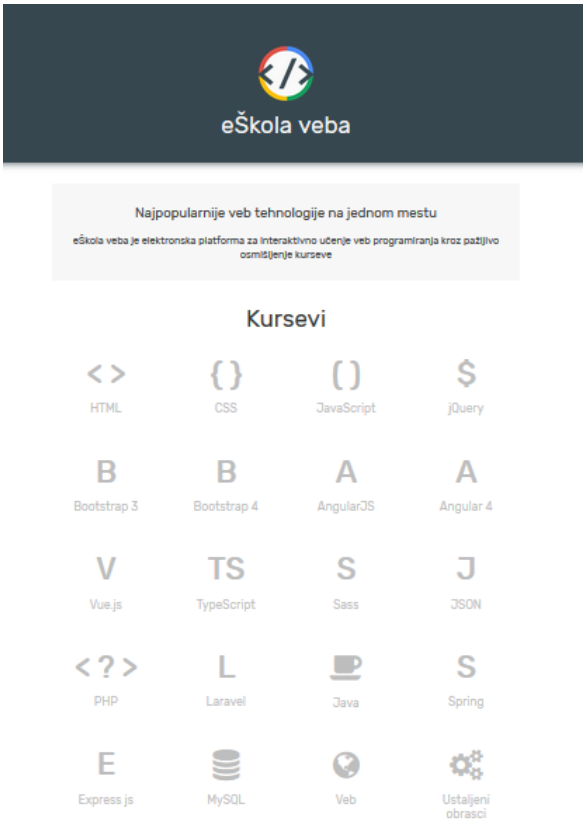
Uvod

Pojava interneta i opšta dostupnost računara doveli su do velikog napretka u oblasti informacionih tehnologija. Zbog opšteg razvoja interneta, savremene veb tehnologije su konstantno u ekspanziji. Elektronske platforme za učenje veb tehnologija su postali značajni i praktični izvori znanja. Jedna takva platforma, dostupna na srpskom jeziku jeste eŠkola veba [2]. Elektronska platforma *eŠkola veba* je elektronska škola koja omogućava interaktivno učenje veb tehnologija. Platforma predstavlja projekat koji se razvija na Matematičkom fakultetu u Beogradu, koji je javno dostupan na adresi `http://www.edusoft.matf.bg.ac.rs/eskola_veba/`. Na ovoj platformi se nalaze besplatni kursevi različitih veb tehnologija, koji su dostupni na srpskom jeziku. Svaki kurs se sastoji iz elektronskih lekcija. Elektronske lekcije o nekim naprednim mogućnostima programskog jezika PHP imaju za cilj obradu i sistematizaciju naprednih koncepata programskog jezika PHP.
Programski jezik PHP je jedan od najpoznatijih jezika namenjenih programiranju dinamičkih veb sadržaja. Početnu popularnost stekao je zbog jednostavne sintakse nasleđene od programskog jezika C. PHP je skriptni programski jezik koji se izvršava na serveru, što znači da programi napisani ovim jezikom ne zahtevaju prevođenje, nego se interpretiraju pri svakom izvršavanju. Zamišljen je kao proceduralni jezik, ali novije verzije podržavaju i objektno orjentisano programiranje. PHP poseduje veliki broj dodataka i biblioteka za rad sa slikama, datumima, povezivanju sa bazama podataka, kreiranje PDF dokumenata itd. Delovi nekih od najposećenijih sajtova današnjice poput Jutuba (engl. *Youtube*), Fejsbuka i Vikipedije isprogramirani su programskim jezikom PHP. Elektronske lekcije o naprednim mogućnostima u programskom jeziku PHP biće prikazane u okviru kursa PHP na platformi *eŠkola Veba*.

U ovom radu biće opisane elektronske lekcije o naprednim mogućnostima programskog jezika PHP. Lekcije su namenjene osobama koje poseduju elementarno znanje iz programskog jezika PHP, kako bi unapredili i proširili svoja znanja. Na početku rada biće obrađene funkcije za rad sa datumima i vremenom. U radu će biti prikazan detaljan rad sa datotekama: otvaranje, zatvaranje datoteke, učitavanje i upisivanje u datoteku, kao i slanje datoteka na server. Biće prikazano i uključivanje datoteke u PHP kôd. Zatim, biće obrađeni kolačići i sesije, kao neizostavan deo velikog broja sajtova. Biće prikazano slanje mejla pomoću programskog jezika PHP. Zatim, biće prikazana obrada grešaka, korišćenje filtera i regularnih izraza. Na kraju biće prikazani osnovni koncepti objektno orjentisanog programiranja i obrada XML dokumenata.

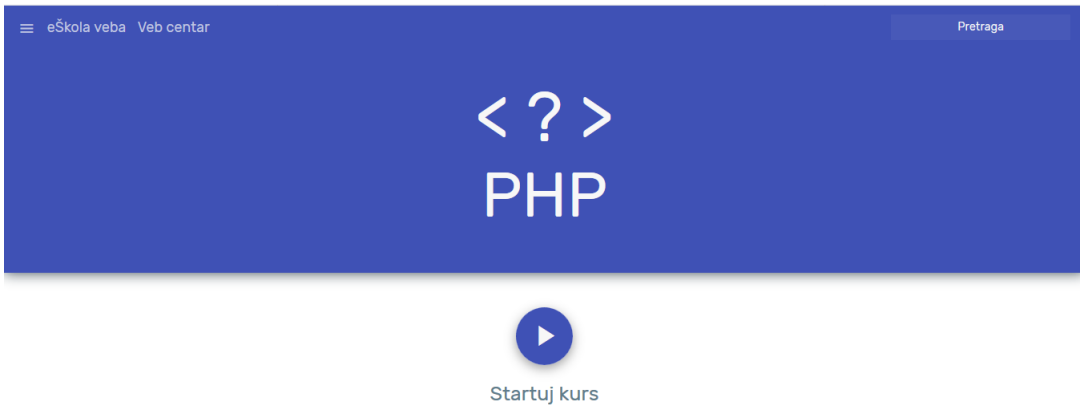
1 Elektronske lekcije

Elektronske lekcije o nekim naprednim mogućnostima programskog jezika PHP su dostupne na platformi eŠkola veba. Lekcije su dostupne na linku `http://edusoft.matf.bg.ac.rs/eskola_veba/#/course-details/php`, i namenjene su korisnicima koji poseduju elementarno znanje iz programskog jezika PHP. Sve lekcije su besplatne i dostupne na srpskom jeziku. Na slici 1 prikazan je izgled platforme.



Slika 1: eŠkola veba

Na platformi se nalaze krusevi različitih veb tehnologija. Jedan od kurseva je kurs o programskom jeziku PHP, koji je kreiran za potrebe ovog rada. Izborom ovog kursa otvara se početna strana kao na slici 2.



Slika 2: Naslovna strana kursa

Elektronske lekcije o nekim naprednim mogućnostima programskog jezika PHP su kreirane za potrebe lakšeg učenja i savladavanja naprednih koncepata programskog jezika PHP. Lekcije se sastoje iz 3 dela: prvi deo je sadržaj te lekcije, drugi deo je primer koji se može direktno menjati, treći deo je prikaz tog primera u veb pregledaču. Na slici 3 je prikazan izgled jedne lekcije u kursu PHP.

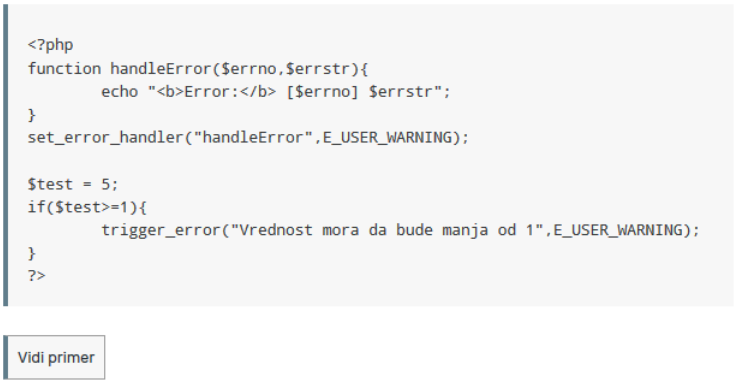
Slika 3: Izgled lekcije u kursu PHP

Za potrebe ovog rada kreirane su sledeće elektronske lekcije:

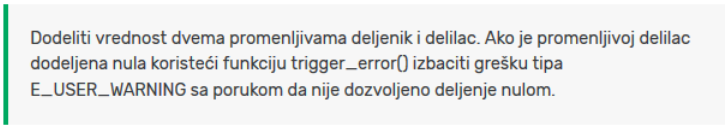
- Datum i vreme;
- Rad sa datotekama;
- Slanje poruke elektronske pošte;
- Korišćenje kolačića;

- Korišćenje sesija;
- Rad sa regularnim izrazima;
- Validacioni filteri;
- Rukovanje greškama;
- Rukovanje izuzecima;
- Uvod u objektno orjentisano programiranje;
- Obrada XML dokumenta u programskom jeziku PHP.

Sadržaj svake lekcije sadrži definicije i opis pojmova potrebnih za razumevanje konkretne teme. Svaku lekciju prati veliki broj kratkih primera. Primeri se mogu pokrenuti direktno sa platforme. Na kraju svake lekcije nalazi se zadatak koji testira naučeno znanje iz svake lekcije. Za svaki zadatak priloženo je i rešenje. Na slici 4 prikazan je jedan primer iz lekcije, dok je na slici 5 prikazan zadatak koji se nalazi na kraju lekcije.



Slika 4: Prikaz jednog primera u lekciji



Slika 5: Zadatak koji se nalazi na kraju lekcije

Elektronske lekcije su kreirane sistematično, redosledom koji prati osnovno razumevanje, a sa svakim novim pojmom korišćeni su i prethodni pojmovi. Cilj ovakvog pristupa je obnavljanje naučenog gradiva i otklanjanje nejasnoća iz prethodnog gradiva.

2 Rad sa datumima i vremenom

U programskom jeziku PHP se za rad sa datumima i vremenom koristi format poznat kao UNIX-ova vremenska oznaka. Radi se o broju sekundi proteklih od početka UNIX epohe, odnosno od ponoći 1. Januara 1970. godine.
UNIX format omogućuje da se podaci o datumu i vremenu čuvaju u sažetom obliku. Na UNIX format ne utiče problem „milenijumske bube“. UNIX vremenska oznaka se čuva kao 32-bitni broj, čime je ograničen opseg datuma: softver ne može da evidentira događaje pre 1902. godine ili posle 2038. godine.
PHP pruža moćne alate koji omogućuju lak rad sa datumima i vremenom. Funkcija *time* daje sve informacije koje su potrebne za trenutni datum i vreme. Ona nema argumente, a njen rezultat je ceo broj.
Ceo broj koji vraća je vreme u formatu UNIX-ove vremenske oznake. Taj broj je ljudima nečitljiv, ali PHP nudi alate da ovaj broj pretvori u čitljiv zapis.

Funkcija *date*

Najznačajnija funkcija za rad sa datumima je svakako funkcija *date*. Ona služi za pretvaranje UNIX vremenske oznake u željeni format. Argumenti koje prima su format i vremenska oznaka. Ukoliko se vremenska oznaka izostavi, uzima se trenutni datum i vreme. Funkcija vraća formatirani string koji predstavlja datum.
Ovo su neki karakteri koji se koriste pri formatiranju datuma:

- d - Predstavlja dan u mesecu (01 - 31);
- m - Predstavlja mesec (01 - 12);
- Y - Predstavlja godinu (zapisanu sa 4 cifre);
- l - Predstavlja dan u nedelji.

Karaktere kao što su „/”, „.”, „-”je dozvoljeno koristiti pri formatiranju datuma i vremena. U primeru 1 prikazano je nekoliko poziva funkcije *date*.

Primer 1: Pozivi funkcije *date*

```
<?php
  echo "Danas je " . date("Y/m/d") . "<br>";
  echo "Danas je " . date("Y.m.d") . "<br>";
  echo "Danas je " . date("Y-m-d") . "<br>";
  echo "Danas je " . date("l");
  ?>
```

Rezultat primera 1 je:

```
  Danas je 2018/7/30.
  Danas je 2018.7.30.
  Danas je 2018-7-30.
  Danas je Monday.
```

Ovo su neki karakteri koji se koriste pri formatiranju vremena:

- h - Predstavlja časove (01 - 12);
- i - Predstavlja minute (01 - 59);
- s - Predstavlja sekunde (01 - 59);
- a - Predstavlja vreme pre podne ili posle podne (am ili pm).

U primeru 2 je prikazano formatiranje vremena pomoću funkcije *date*.

Primer 2: Formatiranje vremena

```
<?php
echo "Trenutno vreme je " . date("h : i : s a") . "<br>";
?>
```

Rezultat primera 2 bi u zavisnosti od trenutka bio na primer:

```
Trenutno vreme je 1:06:54 am.
```

Funkcija *mktime*

Drugi argument funkcije *date* je UNIX vremenska oznaka, koja je u obliku koji nije čitljiv.

Primer 3: Drugi argument funkcije *date* je UNIX vremenska oznaka

```
<?php
echo date("Y-m-d","1359780799");
?>
```

```
2013-02-01
```

Funkcija *mktime* se koristi za pretvaranje podataka o datumu i vremenu u UNIX vremensku oznaku, koja se koristi kao drugi argument funkcije *date*.
Sintaksa funkcije *mktime*:

```
mktime(sati, minuti, sekunde, mesec, dan, godina);
```

Svi argumenti funkcije *mktime* su opcioni. Ukoliko se izostave, koriste se trenutni datum i trenutno vreme. Nije dopušteno izostavljanje parametara koji predstavlja vreme. Ako vreme nije bitno, mogu se zadati nule kao vrednosti paramerara sati, minuti, sekunde.

Primer 4: Poziv funkcije *mktime*

```
<?php
$datum = mktime(3,24,5,7,30,2018);
echo $datum;
?>
```

```
1532921045
```

U primeru 4 prikazano je kako se uz pomoć funkcije *mktime* ispisuje UNIX vremenska oznaka za datum 30.7.2018 i vreme 3:24:05.

Primer 5: Formatiranje određenog datuma i vremena

```
<?php
echo date("d/m/Y h:i:s a", mktime(3,24,5,7,30,2018));
?>
```

```
30/07/2018 03:24:05 am
```

U primeru 5 se kao drugi argument funkcije *date* korsiti funkcija *mktime* koja datum 30/7/2018 i vreme 3:24:05 pretvara u UNIX vremensku oznaku.

Funkcija *strtotime*

Funkcija *strtotime* se koristi da string zadat na engleskom jeziku pretvori u UNIX vremensku oznaku. U primeru 6 je prikazan poziv funkcije *strttotime*.

Primer 6: Poziv funkcije *strtotime*

```
<?php
$d = strtotime("7:50pm April 1 2018");
echo "Kreirani datum je ". date("Y-m-d h:i:sa",$d);
?>
```

```
Kreirani datum je 2018-04-01 7:50:00pm
```

U primeru 6 se uz pomoć funkcije *strtotime* formatira datum 1 April 2018 i vreme 7:50 posle podne.
PHP je veoma pametan u prevođenju stringa u datum, tako da funkcija *strtotime* kao argument može da prihvati dosta vrednosti.

Primer 7: Primeri poziva funkcije *strtotime*

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d)."<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d)."<br>";

$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d)."<br>";
?>
```

Napomena 1 *Funkcija* `strtotime` *nije perfektna. Ona vraća datum ako je korektan zapis stringa, a ako nije vraća false.*

Funkcija *getDate*

Funkcija *getDate* se koristi za dohvatanje pojedinog podatka iz vremenske oznake. Funkcija prima vremensku oznaku kao argument: `getDate(vremenska oznaka)`. Vremenska oznaka je opcioni argument. Predstavlja datum i vreme u UNIX formatu. Ako se ne navede, podrazumevana vrednost je trenutni datum i vreme. Rezultat funkcije *getDate* je asocijativni niz koji sadrži 10 elemenata, odnosno 10 podataka o datumu i vremenu.

Tabela 1: Elementi niza koji vraća funkcija *getDate*

Ključ	Vrednost
seconds	sekunde (0 - 59)
minutes	minuti $(0-59)$
hours	sati $(0-23)$
mday	dan u mesecu $(0-31)$
wday	dan u sedmice $(0-6)$
mon	mesec $(1-12)$
year	godina (4 cifre)
yday	dan u godini $(1-365)$
weekday	ime dana u sedmici
month	ime meseca u godini

U primeru 8 je prikazan poziv funkcije *getDate* i prolazak petljom kroz asocijativni niz koji ta funkcija vraća.

Primer 8: Poziv funkcije *getDate*

```
<?php
$vreme = mktime(3,24,5,8,13,2007);
$datum_niz = getdate($vreme);
foreach( $datum_niz as $kljuc => $vrednost){
    echo "$kljuc = $vrednost <br>";
    }
$datum =  "Datum je ";
$datum.=$datum_niz['mday']."/";
$datum.=$datum_niz['mon']."/";
$datum.=$datum_niz['year'];
echo $datum;
?>
```

```
seconds = 5
minutes = 24
hours = 3
mday = 13
wday = 1
mon = 8
year = 2007
yday = 224
weekday = Monday
month = August
Datum je 13/8/2007
```

Primer 8 pokazuje kako se uz pomoć funkcije *getDate* mogu dobiti komponente datuma 13/8/2007 3:24:5. Podaci o tom datumu se nalaze u nizu `$datum_niz`.

Funkcija *checkDate*

Pomoću funkcije *checkDate* moguće je utvrditi da li je datum ispravan (ovo je korisno kada korisnik unosi datum preko forme).
Sintaksa funkcije *checkDate*:

```
checkDate(int meseci, int dan, int godina);
```

Primeri:
`checkDate(4,15,2018)` vraća `true`.
`checkDate(4,31,2018)` vraća `false`, jer April ima samo 30 dana.
Funkcija *checkDate* pri ispitavanju da li je datum ispravan provarava: da li je godina ispravna celobrojna vrednost u opsegu od 0 do 32767, da li je mesec celobrojna vrednost u opsegu od 1 do 12 i da li zadati dan postoji u zadatom mesecu.
Više o datumima može se pročitati na [8], [9]

3 Rad sa datotekama

Postoje dva načina za smeštanje i čuvanje podataka. Prvi način je čuvanje u običnoj datoteci (engl. *flat file*), dok je drugi način čuvanje u bazi podataka (engl. *database*). Obična datoteka može imati mnoštvo oblika, ali se uglavnom pod tim terminom podrazumeva tekstualna datoteka. Ako se radi sa većom količinom podataka, pogodnija je baza podataka ali i obične datoteke imaju svoju primenu i postoje situacije kad je bolje njih koristiti.

3.1 Otvaranje i zatvaranje datoteke

Postoji potreba i mogućnost da program pristupi datoteci, odnosno da se povežu ime datoteke i programski iskaz kojim se čitaju podaci iz te datoteke. To se postiže otvaranjem datoteke funkcijom *fopen*.

Funkcija *fopen* prima dva argumentra. Prvi argument funkcije je putanja do datoteke. Putanja do datoteke može biti apsolutna (potpuna putanja do datoteke, npr. „D:/www/eSkolaVeba/datoteka.txt") ili relativna (putanja u odnosu na osnovni direktorijum dokumenta, npr. „eSkolaVeba/datoteka.txt"). Da bi program bez problema mogao da radi na različitim serverima, bolje je upotrebljavati relativne putanje. Ukoliko putanja nije navedena, datoteka će biti napravljena u istom direktorijumu gde je i kôd programa.
Drugi argument funkcije je način rada s datotekom. Radi se o tekstualnoj konstanti koja opisuje koje će se akcije izvršavati nad datotekom. Svi načini rada dati su u Tabeli 2.

Tabela 2: Način rada sa datotekama

Način rada	Objašnjenje
r	Otvara datoteku za čitanje.
r+	Otvara datoteku za čitanje i upisivanje.
w	Otvara datoteku za upisivanje. Ako datoteka postoji, briše postojeći sadržaj. Ako ne postoji, biće napravljena.
w+	Otvara datoteku za upisivanje i čitanje. Ako datoteka postoji, briše postojeći sadržaj. Ako ne postoji, biće napravljena.
a	Otvara datoteku za dodavanje (upisivanje). Postojeći sadržaj se ne briše. Ako datoteka ne postoji, biće napravljena.
a+	Otvara datoteku za dodavanje(upisivanje) i čitanje. Postojeći sadržaj se ne briše. Ako datoteka ne postoji, biće napravljena.
b	Rad sa binarnom datotekom. Koristi se u kombinaciji sa ostalim načinima (rb - za čitanje binarne datoteke, rw - za upisivanje u binarnu datoteku). Može se koristiti ako sistem razlikuje binarne i tekstualne datoteke. Windows pravi razliku, a Unix ne.

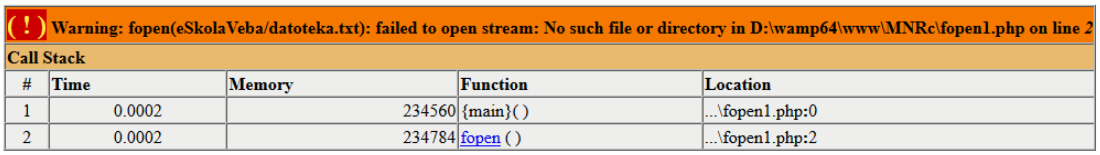
Preporučuje se navođenje najjednostavnijeg potrebnog načina rada (dakle, ako se planira samo čitanje iz datoteke, bolje je navesti „r" nego „r+"), jer se time štede resursi.
Ovako izgleda poziv funkcije *fopen*:

```
1 $datoteka = fopen("eSkolaVeba/datoteka.txt","r");
```

Ako funkcija *fopen* uspešno otvori datoteku, ona vraća pokazivač na tu datoteku. Taj pokazivač (u primeru $datoteka) je promenljiva tipa *resource*, i sav dalji rad sa datotekom se odvija uz pomoć njega.
Nakon poziva funkcije *fopen*, potrebno je proveriti da li otvaranje datoteke uspelo.
Provera da li je otvaranje datoteke uspelo, svodi se na proveru da li je vraćeni pokazivač prazan ili ne. Rad s datotekom treba nastaviti samo ako je datoteka uspešno otvorena.

```php
if($datoteka){
  //rad sa datotekom
}
```

Datoteka se neće uspešno otvoriti ukoliko nepostoji ili ako putanja do nje nije tačno navedena. U tom slučaju PHP će izbaciti poruku o grešci. Slika 6 pokazuje kako izgleda upozorenje da datoteka ne postoji.



Warning: fopen(eSkolaVeba/datoteka.txt): failed to open stream: No such file or directory in D:\wamp64\www\MNRc\fopen1.php on line 2

Call Stack

#	Time	Memory	Function	Location
1	0.0002	234560	{main}( )	...\fopen1.php:0
2	0.0002	234784	fopen ( )	...\fopen1.php:2

Slika 6: Upozorenje da datoteka ne postoji

Umesto poruka koje daje PHP, moguće je uneti svoju. Primer 9 pokazuje kako se definiše sopstvena greška.

Primer 9: Definisanje sopstvene poruke

```php
@$datoteka = fopen("eSkolaVeba/datoteka.txt","r");
if(!$datoteka){
  echo "Greška pri učitavanju datoteke. Pokušajte kasnije";
}
```

Simbol @ govori programu da ne prijavljuje greške nastale usled poziva funkcije.
Kad je rad s datotekom završen, potrebno ju je zatvoriti (opet radi štednje resursa). Zatvaranje datoteke obavlja se pomoću funkcije *fclose*. Ovoj se funkciji kao argument predaje pokazivač na datoteku. Ova funkcija vraća `true` ako je datoteka uspešno zatvorena, odnosno `false` ako nije. Prilikom zatvaranja datoteka ima mnogo manje problema nego pri otvaranju, pa rezultat ove funkcije nema potrebe proveravati.

```php
fclose($datoteka);
```

Koraci koje je potrebno obaviti prilikom rada s datotekom su:

1. otvaranje datoteke,
2. provera da li je datoteka uspešno otvorena,
3. rad sa datotekom,
4. zatvaranje datoteke.

U Primeru 10 su obavljeni svi potrebni koraci za rad sa datotekom.

Primer 10: Svi koraci potrebni za rad sa datotekom

```php
<?php
// otvaranje datoteke
```

```
3   $datoteka = fopen("Datoteka.txt", "w");
4  // provera da li je datoteka otvorena
5   if ($datoteka)
6   {
7   // rad s datotekom
8   // zatvaranje datoteke
9   fclose($datoteka);
10  }
11  ?>
```

3.2 Čitanje podataka iz datoteke

Za čitanje podataka iz datoteke otvorene za čitanje koristi se funkcija *fread*. Funkcija *fread* kao argumente prima pokazivač na datoteku ili naziv datoteke, i broj bajtova koji se učitava iz datoteke. Ukoliko je navedeni broj bajtova veći od veličine datoteke, učitava se ceo sadržaj datoteke.

Primer 11: Učitavanje datoteke „datoteka.txt"

```
1  <?php
2   @$datoteka = fopen("datoteka.txt", "r");
3   if($datoteka){
4     $text_datoteke = fread($datoteka, 20);
5   }
6  ?>
```

U primeru 11, učitava se pomoću funkcije *fread* prvih 20 bajtova iz datoteke „datoteka.txt".
Veličina datoteke u bajtovima može se utvrditi uz pomoć funkcije *filesize*. Pomoću naredbe `filesize("datoteka.txt")` dobija se veličina datoteke „datoteka.txt". Funkcija *filesize* se koristi zajedno sa funkcijom *fread* za učitavanje cele datoteke odjednom.

Primer 12: Učitavanje cele datoteke

```
1    <?php
2   @$datoteka = fopen("datoteka.txt", "r");
3   if($datoteka){
4     $text_datoteke = fread($datoteka, filesize(datoteka.txt));
5   }
6   fclose($datoteka);
7   echo $text_datoteke;
8     ?>
```

U primeru 12 učitava se ceo sadržaj datoteke, a zatim se sadržaj ispisuje. Funkcija *fgets* koristi se za učitavanje jednog reda iz datoteke. Funkcija *fgets* kao argument prima pokazivač na datoteku, a vraća pročitani red.
Funkcija *fgetc* koristi se za učitavanje jednog karaktera iz datoteke. Funkcija *fgetc* kao argument prima pokazivač na datoteku, a vraća pročitani karakter.
Pokazivač na datoteku pamti trenutnu poziciju u datoteci koja se čitanjem pomoću funkcija *fread*, *fgets* i *fgetc* pomiče za zadani broj bajtova, jedan red, odnosno jedan

karakter.
Provera da li je datoteka došla do kraja obavlja se pomoću funkcije *feof*. Ova funkcija prima pokazivač na datoteku a vraća vrednost `true` ako je pokazivač datoteke na kraju datoteke. Čitanje cele datoteke red po red prikazano je u primeru 13.

Primer 13: Čitanje datoteke red po red

```
<?php
@$datoteka = fopen("datoteka.txt", "r");
 if($datoteka){
   while(!feof($datoteka)){
     //Ispisivanje red po red
     echo fgets($datoteka)."<br>";
   }
   fclose($datoteka);
 }
?>
```

Čitanje cele datoteke karakter po karakter prikazano je u primeru 14.

Primer 14: Čitanje datoteke karakter po karakter

```
<?php
@$datoteka = fopen("datoteka.txt", "r");
 if($datoteka){
   while(!feof($datoteka)){
     //Ispisivanje karakter po karakter
     echo fgetc($datoteka);
   }
   fclose($datoteka);
 }
?>
```

3.3 Upisivanje podataka u datoteku

Za upisivanje podataka u datoteku otvorenu za upis, koristi se funkcija *fwrite*. Funkcija *fwrite* prima dva argumenta. Prvi argument je pokazivač na datoteku u koju se upisuje. Drugi argument je tekst koji se upisuje.
Poziv funkcije *fwrite* dat je u primeru 15.

Primer 15: Poziv funkcije *fwrite*

```
  $datoteka = fopen("datoteka.txt", "a");
  fwrite($datoteka, "dodatak");
```

Potrebno je datoteku otvoriti za upisivanje ili dodavanje. U primeru 16 datoteka je otvorena za dodavanje, zatim se funkcijom *fwrite* u datoteku dodaje tekst „dodatak".

3.4 Ostale korisne funkcije za rad sa datotekama

Sledeće funkcije mogu biti korisne pri radu sa datotekama:
Funkcija *file_exists* se koristi za proveru da li neka datoteka postoji, a da ne mora pret-

hodno da se otvori. U primeru 16 je prikazana upotreba funkcije *file_exists*.

Primer 16: Provara da li datoteka postoji

```
if(file_exists("datoteka.txt")){
echo "Datoteka postoji";
}
else{
echo "Datoteka ne postoji";
}
```

Za brisanje datoteke, kada više nije od koristi, koristi se funkcija *unlink* (ne postoji funkcija delete). Naredbom `unlink(„datoteka.txt")` se datoteka „datoteka.txt" briše. Funkcija vraća `false` ako ne može da obriše datoteku. To se obično dešava ako ne postoji datoteka.
Za kretanje kroz datoteku koriste se funkcije *rewind*, *fseek* i *ftell*. Pomoću navedenih funkcija moguće je otkriti položaj pokazivača u datoteci i pomerati ga.
Funkcija *rewind* pomera pokazivač na početak datoteke, dok funkcija *ftell* vraća trenutni položaj pokazivača u datoteci.
U primeru 17 su prikazani pozivi funkcija *ftell* i *rewind*. Sa `$fp` je označen pokazivač na datoteku.

Primer 17: Pozicije pokazivača

```
echo "Pokazivač se nalazi na poziciji ".ftell($fp);
echo "Nakon poziva funkcije rewind pokazivač se nalazi na poziciji ".rewind($fp);
```

Funkcija *fseek* pomera pokazivač za dati broj bajtova u odnosu na referentnu tačku. Njen prototip je:

```
int fseek(resource pokazivac, int pomeraj, int referentna tacka);
```

Referentna tacka je opcioni argument. Ako se ne navede, njegova podrazumevana vrednost je SEEK_SET, što je početak datoteke. Preostale dve moguće vrednosti su SEEK_CUR(trenutna pozicija pokazivača) i SEEK_END (kraj datoteke).

3.5 Slanje datoteke na server

Slanje datoteke na server je lako uz pomoć programskog jezika PHP i HTML forme. Međutim, sa lakoćom dolazi problem bezbednosti, zato treba biti dosta obazriv prilikom slanja datoteke na server.
Proces slanja datoteke na server sadrzi više koraka.

1. Korisnik otvara stranu koja sadrzi HTML formu, gde moze da izabere datoteku.
2. Korisnik klikne dugme za biranje i bira datoteku koji šalje sa svog računara na server.
3. Puna putanja do datoteke se pojavi u polju pored, zatim korsinik klikne na dugme za slanje.

4. Izabrana datoteka se šalje na server, na privremenu lokaciju.

5. PHP kôd stranice koja prihvata formu proverava da li je datoteka stigla i kopira ga u namenjeni direktorijum.

6. PHP kôd potvrdjuje korisniku da li je datoteka stigla.

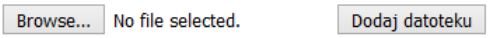
HTML forma koja omogućuje korisniku da izabere datoteku koju želi da doda na server prikazana je u primeru 18.

Primer 18: HTML forma

```
<!DOCTYPE html>
  <html>
  <head>

  </head>
  <body>
  <form action="upload.php" method="POST" enctype="multipart/form-data">
    <input type="file" name="fileToUpload">
    <input type="submit" name="submit" value="Dodaj datoteku">
  </form>
  </body>
  </html>
```

HTML forma iz primera 18 izgleda kao na slici 7.



Slika 7: Nije izabrana nijedna datoteka

U HTML formi zbrog bezbednosti, uvek se koristi metod POST. Formi je neophodan još jedan atribut: enctype=„multipart/form-data". On određuje koji tip sadržaja će se koristiti prilikom slanja forme. Primetiti još da se koristi atribut type = "file", koji polje za unos prikazuje kao polje za odabir datoteke.

Napomena 2 *Klikom na dugme „Browse" otvara se samo dijaloški okvir za odabir datoteke na klijentskom računaru. Nakon što se datoteka odabere i klikne "OK", datoteka još nije poslata. Slanje datoteke na server događa se tek kad se klikne na dugme forme za slanje podataka (dugme tipa submit).*

Forma šalje podatke datoteci koji se zove „upload.php". Njen izgled prikazan je u primeru 19.

Primer 19: Datoteka „upload.php"

```
<?php
$uploadOk = 1;
if(isset($_FILES['fileToUpload'])){
$target_file = "uploads/".basename($_FILES['fileToUpload']['name']);
```

```
5  $file_name = $_FILES['fileToUpload']['name'];
6  $file_size = $_FILES['fileToUpload']['size'];
7  $file_tmp = $_FILES['fileToUpload']['tmp_name'];
8  $file_type = $_FILES['fileToUpload']['type'];
9
10 if($file_size > 2097152){
11 echo "Datoteka mora da bude manja od 2MB";
12 $uploadOk = 0;
13 }
14
15 if(file_exists($target_file)){
16 echo "Datoteka već postoji.";
17 $uploadOk = 0;
18 }
19
20 if($uploadOk == 0){
21 echo "Datoteka nije otpremljena";
22 }
23 else{
24   move_uploaded_file($file_tmp,$target_file);
25   echo "Success";
26 }
27 }
28 ?>
```

U primeru 19 proverava se veličina datoteke i da li već postoji na serveru. Ako prođe proveru datoteka se šalje na željeno mesto na serveru.
Može iz primera 19 da se zaključi da je za dodavanje fajlova na server neophodno korišćenje globalne promenljive $_FILES. Ova promenljiva je asocijativni dvodimenzioni niz, koja čuva sve informacije vezane za datoteku koju šaljemo na server. Ako se za atribut name datoteke koja se dodaje na server kao u primeru 20 navede fileToUpload, PHP će kreirati 5 promenljivih:

- $_FILES['fileToUpload']['tmp_name'] - Lokacija datoteke na serveru.
- $_FILES['fileToUpload']['name'] - Naziv datoteke na korisnikovom računaru.
- $_FILES['fileToUpload']['size'] - veličina datoteke.
- $_FILES['fileToUpload']['type'] - tip datoteke.
- $_FILES['fileToUpload']['error'] - podaci o grešci

Poslata datoteka nalazi se na privremenoj lokaciji na serveru koja se može dobiti pomoću vrednosti $_FILES["fileToUpload"]["tmp_name"]. Da bi se datoteka smestila na željeno mesto (i pod željenim imenom), potrebno je koristiti funkciju *move_uploaded_file*. Funkcija je pozvana u primeru 19.
Prvi argument funkcije *move_uploaded_file* je trenutna, privremena putanja do datoteke, a drugi argument je željena putanja. U primeru 19 to je folder „uploads" koji se nalazi u istom direktorijumu kao „upload.php".

3.6 Uključivanje spoljnih datoteka u kôd

Moguće je u PHP datoteku uključiti sadržaj druge datoteke pre nego što se program izvrši. Postoje 2 funkcije koje mogu spoljnu datoteku da uključe u PHP datoteku:

- *include*,
- *require*.

Uključivanje datoteka je dosta korisno kad je potrebno isti PHP, HTML kôd ili običan tekst uključiti na više stranica sajta. To omogućuje programerima da uz minimalno napora promene izgled celog sajta. Najčešće se koristi pri uključivanju hedera i futera na više stranica sajta.

Funkcija *include*

Funkcija *include* ceo tekst iz odabrane datoteke kopira u datoteku u kojoj se poziva funkcija. Ako dođe do problema pri uključivanju datoteke, onda će funkcija *include* obezbediti upozorenje korisniku, ali će program nastaviti da se izvršava.
Datoteka „header.php", koja će biti korišćena u narednom sadržaju data je u primeru 20.

Primer 20: Datoteka „header.php"

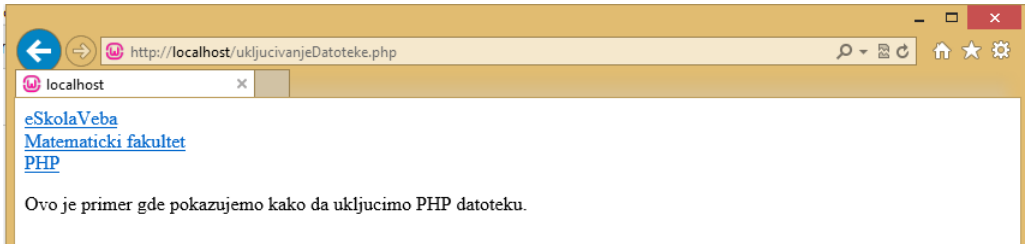
```
<a href="http://edusoft.matf.bg.ac.rs/eskola_veba/#/home">eSkolaVeba</a><br>
<a href="http://www.matf.bg.ac.rs/">Matematicki fakultet</a><br>
<a href="https://www.w3schools.com/php/">PHP</a>
```

U primeru 21 je prikazan poziv funkcije *include* za uključivanje datoteke header.php.

Primer 21: Poziv funkcije *include*

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<header>
<?php
include('header.php');
?>
<p>Ovo je primer gde pokazujemo kako da ukljucimo PHP datoteku.</p>
</header>
</body>
```

Rezultat primera 21 prikazan je na slici 8.



Slika 8: Uključivanje datoteke header.php

Funkcija *require*

Funkcija *require* takođe ceo tekst iz odabrane datoteke kopira u datoteku u kojoj se poziva funkcija. Ako dođe do problema pri uključivanju datoteke onda program izbacuje gresku i zaustavlja se izvršavanje programa.
U primeru 22 je prikazan poziv funkcije *require*, za učitavanje datoteke header.php.

Primer 22: Poziv funkcije *require*

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<header>
<?php
require('header.php');
?>
<p>Ovo je primer gde pokazujemo kako da ukljucimo PHP datoteku.</p>
</header>
</body>
```

Kad se datoteka uspešno učita, funkcije *include* i *require* imaju isti efekat. Jedina razlika između funkcija *include* i *require* je kada dođe do problema pri uključivanju datoteke. Pri korišćenju funkcije *include* program će izbaciti upozorenje ali ostatak programa će se izvršiti, dok kod funkcije *require* funkcije neće doći do nastavka izvršenja programa. Razlika između funkcija *include* i *require* data je u primerima 23 i 24.

Primer 23: Uključivanje nepostojeće datoteke pomoću funkcije *require*

```
<?php
require('nepostojeći_fajl.php');
?>
<p>Ovaj deo se neće izvršiti!</p>
```

Primer 24: Uključivanje nepostojeće datoteke pomoću funkcije *include*

```
<?php
include('nepostojeći_fajl.php');
?>
<p>Ovaj deo će se izvršiti!</p>
```

U primeru 23 pasus će se prikazati, dok se u primeru 24 neće prikazati.

Napomena 3
Treba koristiti `require` *kad je za program neophodno da se datoteka učita.*
Treba koristiti `include` *kad za program nije neophodno da se datoteka učita.*

Više o datotekama može se pročitati na [4], [5].

4 Slanje poruke elektronske pošte

Pomoću programskog jezika PHP moguće je na jednostavan način poslati poruku elektronske pošte. Poruka može biti poslata automatski (bez interakcije s korisnikom) ili može biti poslata kad korisnik upiše tekst poruke u HTML formu i klikne na dugme za slanje podataka.
Za slanje poruke elektronske pošte u programskom jeziku PHP koristi se funkcija *mail*. Sintaksa funkcije *mail* izgleda ovako:

```
mail(primalac, naslov poruke, poruka, zaglavlja, dodatni parametri);
```

Prvi argument funkcije *mail* je adresa primaoca. Ukoliko je primalaca više, adrese moraju biti odvojene zarezom. Adresa može biti u obliku: ime.prezime@domen.com ili u obliku: Ime Prezime<ime.prezime@domen.com>.
Drugi argument funkcije je naslov poruke. U naslovu poruke se ne sme nalaziti karakter za novi red.
Treći argument funkcije je tekst poruke. U tekstu poruke za novi red se koristi karakter „\n”, a linija može sadržati maksimalno 70 karaktera.
Četvrti argument funkcije su dodatna zaglavlja: pošaljilac poruke (*From*), dodatni primaoci (*Cc*, *Bcc*) i druga. Zaglavlja moraju biti odvojena kombinacijom znakova „\r\n”. To je opcioni argument.
Peti argument su dodatni parametri. Oni služe za prosleđivanje parametara programu na serveru koji šalje poruku.
Funkcija *mail* vraća `true` ako je poruka uspešno poslata, a `false` ako slanje nije uspelo. Slanje poruke prikazano je u primeru 25.

Primer 25: Slanje mejla korišćenjem funkcije *mail*

```
1 <?php
2 $primalac = 'xyz00@edusoft.matf.bg.ac.rs';
3 $naslov = 'Naslov poruke';
4 $tekst = 'Pozdrav svima!';
5 $zaglavlja = 'From: abc@edusoft.matf.bg.ac.rs'.
6 "\r\n" . 'Cc: efg@edusoft.matf.bg.ac.rs'
7 mail($primalac, $naslov, $tekst, $zaglavlja);
8 ?>
```

Podešavanje postavki za slanje poruke

Pre slanja poruke moraju se podesiti postavke za slanje poruke elektronske pošte. Pregled postavki dat je u tabeli 3.

Tabela 3: Postavke za slanje poruke elektronske pošte

Naziv postavke	Objašnjenje
SMTP	Naziv ili IP adresa servera elektronske pošte (ova postavka je dostupna samo pod operativnim sistemom Windows).
smtp_port	Port preko kojeg se šalje elektronska pošta (najčešće 25; ova postavka je dostupna samo pod operativnim sistemom Windows).
sendmail_from	Adresa sa koje se šalje poruka.
sendmail_path	Putanja programa za slanje poruka elektronske pošte na serveru.

Ove postavke nalaze se u datoteci php.ini. Za podešavanje ovih i drugih konfiguracijskih postavki iz koda može se koristiti funkcija *ini_set*. Argumenti koje ona prima su naziv postavke i vrednost postavke. Poziv funkcije *ini_set* prikazan je u primeru 26.

Primer 26: podešavanje postavki za slanje poruke elektronske pošte

```
<?php
ini_set("SMTP", "edusoft.matf.bf.ac.rs");
ini_set("sendmail_from", "abc@edusoft.matf.bg.ac.rs");
mail($primatelj, $naslov, $tekst);
?>
```

Više o slanju elektronske pošte može se videti na [1], [5].

5 Korišćenje kolačića i sesija

Veb pregledač i veb server međusobno komuniciraju pomoću HTTP protokola. Svaki HTTP zahtev koji veb pregledač šalje veb serveru nezavisan je od svih drugih zahteva, zbog čega se između dva zahteva ne čuva trenutno stanje. Kad stigne novi zahtev, server ne zna da je on stigao od istog korisnika kao i prethodni zahtev i da se nastavlja na prethodne korisnikove akcije. U aplikacijama u kojima je potrebna složenija intervencija korisnika, mora se obezbediti čuvanje stanja pri prelasku sa jedne stranice aplikacije na drugu. Za čuvanje tekućeg stanja PHP obezbeđuje kolačiće i sesije. Primer gde se pojavljuje čuvanje stanja jeste veb aplikacija prodavnice.
Korisnik dodaje stavke u korpu dok pretražuje ili pregleda neki katalog. Stanje korpe za kupovinu (stavke koje su sadržane u njoj) mora se negde čuvati. Kada korisnik zahteva stranicu za prikaz sadržaja korpe, mora se prikazati koje se stavke u njoj nalaze.
Stanje aplikacije (vrednosti pojedinih promenljivih) mora se negde skladištiti između dva HTTP zahteva. Promenljive koje opisuju stanje mogu se čuvati na dva mesta:

- u klijentskom veb pregledaču
- na veb serveru

5.1 Korišćenje kolačića

Kolačić je mala tekstualna datoteka koju server smešta u računar korisnika. Kolačići se često koriste za identifikaciju korisnika. Postoji 4 koraka u identifikaciji korisnika.

1. Pretraživač šalje zahtev serveru za nekom stranicom.
2. Server šalje odgovor pretraživaču i zajedno sa njim i kolačić.
3. Pretraživač skladišti kolačić (kao tekstualnu datoteku) u računar korisnika.
4. Pri sledećem zahtevu istom serveru, pretraživač šalje kolačić nazad na server, i server ga koristi za identifikaciju korisnika.

Kolačići se takođe koriste i za pamćenje raznih korisničkih podešavanja (npr. odabir jezika stranica koje će se pretraţivati), tako da se sledeći put kad korisnik poseti istu stranicu, primenjuju podešavanja koja su odabrana prošli put.
Za kreiranje kolačića u programskom jeziku PHP koristi se funkcija *setcookie*. Funkcija prima 6 argumenata, iako je samo prvi argument obavezan. Sintaksa funkcije *setcookie* izgleda ovako:

```
setcookie(naziv, vrednost, trajanje, putanja, domen, sigurnost);
```

Prvim argumentom funkcije se zadaje naziv kolačića. Koristi se za pristup kolačiću. Drugim argumentom se zadaje vrednost kolačića. Vrednost kolačića čiji je naziv my_cookie u kodu se dohvata sa $_COOKIE["my_cookie"]. Treći argument funkcije je vreme (u UNIX vremenskoj oznaci) kada kolačić prestaje da postoji, odnosno biće izbrisan sa korisničkog

računara. Najčešće korišćeni oblici su `time()`+broj sekundi ili `mktime()`. Četvrti argument predstavlja putanju na serveru na kojoj će kolačić biti dostupan. Vrednosti koje je moguće navesti su:

- „/"- ceo domen,
- „/www/admin/"- u direktorijumu www/admin i svim poddirektorijumima,
- default path - tekući direktorijum u kome se nalazi kôd koji kreira kolačić.

Petim argumentom funkcije se precizira domen u kome je kolačić dostupan. Vrednosti koje je moguće navesti su:

- „ "(prazan string) - domen kome pripada server na kome se nalazi kôd koji kreira kolačić,
- „www.examle.com"- u navedenom domenu,
- „.bg.ac.rs"- u svim poddomenima domena bg.ac.rs.

Podrazumevana vrednost šestog argumenta je 0 ili `false`, a ako se postavi na 1 ili `true`, tada se kolačić šalje samo preko sigurne HTTPS veze, a ne kroz sve (i sugurne i nesigurne) veze.

Napomena 4 *HTTPS veza je veza između servera i klijenta koja se odvija putem HTTPS protokola. Radi se o nadogradnji HTTP protokola (dodavanjem SSL sloja – Secure Sockets Layer) u kome se podaci ne šalju u jasnom (clear text), već u enkriptiranom obliku.*

Pozivi funkcije *setcookie* prikazani su u primeru 27.

Primer 27: Pozivi funkcije *setcookie*

```
<?php
setcookie("ime", "Marko Markovic", time()+3600, "/", "", 0);
setcookie("godine", "36", time()+3600);
?>
```

U primeru 28 kreirani su kolačić sa nazivom „ime" i vrednošću „Marko Marković", kao i kolačić sa nazivom „godine"i vrednošću „36". Oba kolačića su dostupna 3600 sekundi odnosno 1 sat. Pri slanju zahteva na server, klijent će poslati i sve kolačiće čiji domen odgovara domenu servera. Kolačićima koje klijent šalje, server može pristupiti preko globalne promenljive $_COOKIE. U primeru 28 je prikazano kako bi server pristupio kolačiću čiji je naziv „godine".

Primer 28: Pristup kolačiću

```
$starost = $_COOKIE['godine'];
echo $starost;
```

```
36
```

Kolačići moraju biti poslati pre bilo koji naredbe izlaza u okviru koda (ograničenje protokola). To znači da poziv funkcije *setcookie* prethodi bilo kom izlazu (npr. pozivu funkcije echo), uključujući i etikete <html>i <head>.
Korisno je koristiti funkciju *isset* za proveru da li je vrednost kolačića postavljena ili nije. U direktorijumu domaći (D:\Program Files\wamp\www\domaći) nalazi se datoteka kolačić.php, prikazana u primeru 29.

Primer 29: kolačić.php

```
<?php
$vrednost = "Neka vrednost kolačića!";
setcookie("testKolačić",$vrednost,time()+3600,"/domaci/");
//ističe za sat vremena
 ?>
```

U koren (*root*) direktorijumu (D:\Program Files\wamp\www) nalazi se datoteka kolačić2.php, prikazana u primeru 30.

Primer 30: kolačić2.php

```
  <?php
   if(isset($_COOKIE["testKolačić"])){
 echo $_COOKIE["testKolačić"];
   }
   ?>
```

U primeru 30 je prikazan je i poziv funkcije *isset*, kako bi se proverilo da li je postavljena vrednost kolačića. Kada se pokrene kolačić.php, a zatim kolačić2.php, na ekranu ništa nije ispisano, jer kolačić2.php nije u dozvoljenom direktorijumu (folderu).
Neka se kolačić2.php nalazi u direktorijumu \domaći, odnosno (D:\Program Files\wamp\www\domaći). Kada se pokrene kolačić.php, a zatim kolačić2.php, na ekranu se pojavi:

```
Neka vrednost kolačića!
```

Za brisanje kolačića koristi se takođe funkcija *setcookie*, samo sa negativnim danom isteka. Brisanje kolačića prikazano je u primeru 31.

Primer 31: Brisanje kolačića

```
  <?php
   $vrednost = "Neka vrednost kolačića!";
   setcookie("testKolačić", $vrednost, time()-3600, "/domaci/");
   // isteklo trajanje kolačića
   echo "Kolačić je obrisan";
   ?>
```

Kolačići se koriste u aplikacijama u kojima nije neophodno da se složeni podaci čuvaju između dva zahteva serveru.
Broj i veličina kolačića su ograničeni: veb pregledač može da čuva samo poslednjih 20 kolačića koji su mu bili poslati iz određenog domena, a veličina svakog kolačića je ograničena na 4KB (ovo se vremenom menja i zavisi od verzije pregledača).

Treba imati u vidu da, pošto se kolačići smeštaju na korisnikov računar, programer veb aplikacije ima veoma malo kontrole nad njima. Ukoliko ih korisnik iz nekog razloga obriše ili onemogući podršku za kolačiće u veb pretraživaču, programer tu ne može ništa, osim da izbegava pisanje koda koji mnogo zavisi od njih.

Primer 32: Provera posete tokom dana

```
<?php
if (!isset($_COOKIE['posečen'])) {
// ako kolačić ne postoji, kreira se
setcookie("posećen", "1", time()+86400, "/");
echo "Ovo je prva Vaša poseta u toku danasnjeg dana.";
}
else {
// ako kolačić već postoji
echo "Već ste bili danas ovde!";
}
?>
```

U primeru 32, prikazana je upotreba kolačića za proveru da li je korisnik tokom dana već posetio sajt. Prilikom posete kreira se kolačić sa imenom „posećen", koji traje 86400 sekundi, odnosno 1 dan.

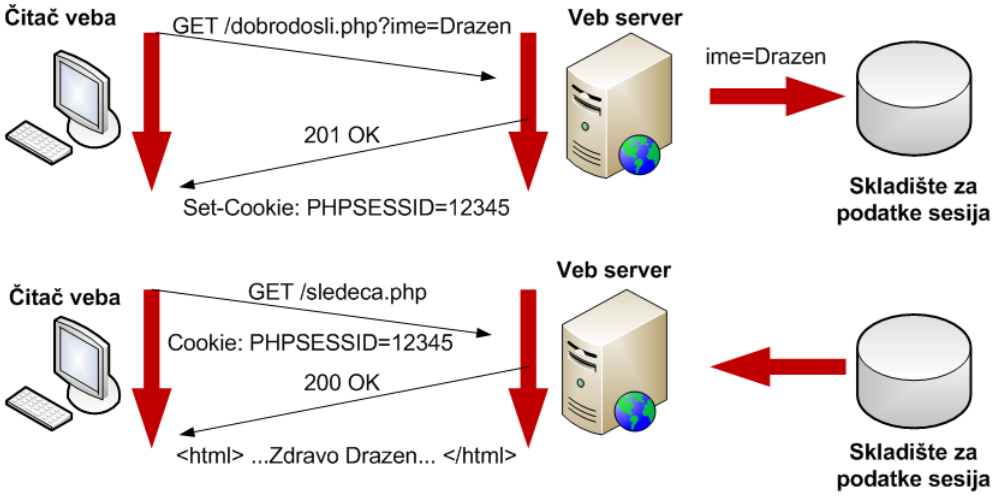
5.2 Korišćenje sesija

Drugi način čuvanja tekućeg stanja aplikacije jesu sesije (engl. *session*). Pomoću sesija se tokom posete sajtu čuvaju podaci specifični za datu konekciju. Podaci iz sesija se čuvaju tokom posete na serveru, a zatim se brišu (obično kada korisnik zatvori svoj veb pretraživač). Time što se podaci čuvaju na serveru rešava se problem čuvanja promenljivih stanja koje zauzimaju više prostora i/ili većeg broja promenljivih stanja.
Takođe, na taj način rešava se i problem zaštite podataka sadržanih u promenljivama stanja od nenamernih ili namernih izmena koje bi korisnik mogao načiniti.

Identifikator sesije

Svaka sesija ima jedinstveni identifikator koji se čuva na dva mesta: kod korisnika kao privremeni kolačić i na serveru gde se automatski generiše od strane programskog jezika PHP, tako da programer o tome ne mora da vodi računa.
Veb pregledač čuva i ugrađuje u svaki HTTP zahtev samo identifikator sesije, na osnovu koga server obrađuje prispeli zahtev u kontekstu tekućeg stanja.
Kada korisnik prvi put pokrene aplikaciju koja upravlja stanjem, tako što zahteva veb stranicu koja započinje sesiju, PHP prvo generiše identifikator (ID) sesije, a zatim formira datoteku za smeštaj vrednosti promenljivih koje se odnose na novu sesiju. U odgovor PHP ugrađuje kolačić čije je ime PHPSESSID koji sa drži ID sesije. ID sesije je broj od 32 nasumično generisane heksadecimalne cifre, kao npr. 242c489fb4a1bc79f5cf365988167e4d. Klijentski veb pregledač potom skladišti kolačić i umeće njegovu vrednost u sve naredne zahteve koje šalje serveru. Razmena podataka preko sesija prikazana je na slici 9.

Ako su kolačići onemogućeni ili nisu podržani u korisnikovom veb pregledaču, identifikator sesije će se proslediti preko URL-a. Primer takvog URL-a je http://www.matf.bg.ac.rs?PHPSESSID=242c489fb4a1bc79f5cf365988167e4d.
Datoteke sesija se smeštaju u direktorijum /tmp, a ime datoteke se sastoji od ID sesije i prefiksa sess_. Primer imena datoteke je sess_242c489fb4a1bc79f5cf365988167e4d.



Slika 9: Razmena podataka između pregledača i servera

Rad sa sesijama

Rad sa sesijama obuhvata započinjanje sesije, rad sa sesijskim promenljivama i na kraju okončavanje sesije. Za započinjanje nove sesije ili učitavanje postojeće sesije koristi se funkcija *session_start*.
Funkcija *session_start* kreira novu sesiju i time omogućava pristup superglobalnom nizu $_SESSION ili učitava postojeću sesiju koju identifikuje na osnovu ID sesije koji je prosleđen u HTTP zahtevu.
Ova funkcija ne prima argumente i uvek vraća vrednost `true`.

Napomena 5 *Poziv funkcije* `session_start` *se mora izvršiti na samom početku stranice, pre bilo kakvog izlaza u veb pretraživač klijenta.*

Rad sa sesijskim promenljivama

U sesijskim promenljivima se nalaze podaci koji opisuju trenutno stanje aplikacije. One su smeštene u globalni niz $_SESSION i njima se može pristupiti sa
`$_SESSION["naziv sesijske promenljive"]`.
Kreiranje nove sesijske promenljive je zapravo dodavanje novog člana u niz $_SESSION.
`$_SESSION['korisnik']=$imeKorisnika;`

Za proveru da li je neka sesijska promenljiva kreirana ili ne, može se koristiti funkcija *isset*. U primeru 33 je prikazan rad sa sesijskim promenljivama.

Primer 33: Rad sa sesijskim promenljivama

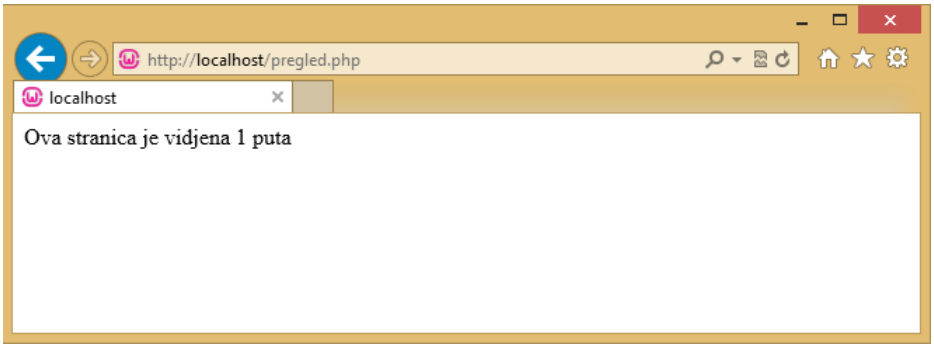
```
<?php
if(isset($_SESSION["korisnik"]))
{
echo $_SESSION["korisnik"];
}
?>
```

Primer korišćenja sesija je brojač poseta. Svaki put kada se ovako napravljena stranica osveži(engl. *refresh*), parametar brojač se poveća za 1. Sesija se zatvara isključivanjem veb pretraživača. Brojač sesija je prikazan u primeru 34.

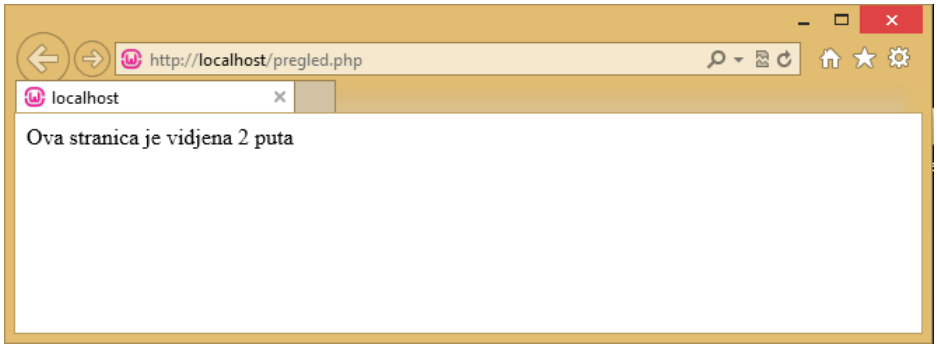
Primer 34: Brojač poseta

```
<?php
// inicijalizacija sesije
session_start();
if(isset($_SESSION['views']))
$_SESSION['views']=$_SESSION['views']+1;
else
$_SESSION['views']=1;
// stampaj vrednost
echo "Ova stranica je vidjena " . $_SESSION['views'] . " puta";
?>
```

Rezultat primera 34 prikazan je na slici 10 i slici 11.



Slika 10: Rezultat primera 34 ako se stranica prvi put učitava



Slika 11: Nakon osvežavanja (engl. *refresh*) stranice broj pregleda se pvećava za jedan

Brisanje postojeće sesijske promenljive se vrši pomoću funkcije *unset*. Jedini argument funkcije je naziv sesijske promenljive. U primeru 35 je prikazano brisanje sesijske promenljive korišćenjem funkcije *unset*.

Primer 35: Brisanje sesijske promenljive

```
<?php
session_start();
$_SESSION['korisničko_ime']="Jovan";
echo "Vaše korisničko ime je " .$_SESSION['korisničko_ime']."<br>";
unset($_SESSION['korisničko_ime']);
echo "Vaše korisničko_ime je sada: " .$_SESSION['korisničko_ime']. ".";
?>
```

Rezultat primera 35:

```
Vaše korisničko ime je Jovan.
Vaše korisničko ime je sada: .
```

Sesija bi trebalo da se okonča kada se korisnik odjavi iz aplikacije (izborom opcije „Kraj rada").
Okončavanje sesije moguće je izvršiti ranije: pozivom funkcije *session_destroy*. Funkcija *session_destroy* nema argumente i uvek vraća vrednost `true`. Nakon poziva funkcije sve vrednosti sesijskih promenljivih biće izbrisane.
Korisnici se često ne odjavljuju iz aplikacije na adekvatan način. U tom slučaju se korisnikova sesija neće završiti tj. njegove sesijske promenljive će se i dalje čuvati na serveru. Server nikada ne može da bude siguran da li se na drugoj strani veze još uvek nalazi korisnik (svaki HTTP zahtev je nezavisan od drugih zahteva). Zato server treba redovno da čisti stare, nezavršene (zamrznute) sesije u kojima se tokom određenog vremena nije ništa događalo. Zamrznute sesije troše resurse servera i smanjuju bezbednost podataka. Dužina intervala čekanja, pre nego što se neka zamrznuta sesija očisti, nije univerzalni parametar i zavisi od potreba aplikacije.
U mehanizam upravljanja sesijama ugrađen je mehanizam za sakupljanje smeća, koji obezbeđuje da se datoteke neaktivnih (zamrznutih) sesija posle izvesnog vremena brišu. Sakupljanje smeća sprečava se da se folder prepuni datotekama sesija, što slabi perfor-

manse sistema. Sakupljanjem smeća smanjuje se rizik da neko ko nasumično pogađa ID sesija ukrade neku staru sesiju koja se više ne koristi.
Postoje dva parametra koja mogu da se podese u podešavanjima na serveru (u datoteci php.ini) koja upravljaju sakupljanjem smeća. To su `session.gc_maxlifetime` i `session.gc_probability`. Tokom postupka sakupljanja smeća, ispituju se sve sesije i briše se svaka sesija kojoj niko nije pristupio tokom vremena određenog parametrom `gc_maxlifetime` (podrazumevana vrednost je 1440 sekundi).
Parametar `gc_probability` određuje procenat verovatnoće sakupljanja smeća (100% - svaki put kada se pozove funkcija session_start, 1% - sa verovatnoćom 0.01 svaki put kada se pozove funkcija *session_start*). Postupak sakupljanja smeća može poprilično da optereti server, naročito kod veb aplikacija sa velikim brojem korisnika (mora se ispitati datum poslednjeg ažuriranja svake sesije). Ako se parametar `gc_probability` podesi suviše visoko nepotrebno se opterećuje server, a ako se podesi suviše nisko javlja se problem zamrznutih sesija i problema koje one uzrokuju. Generalno, ne postoji pravilo za podešavanje parametara `gc_maxlifetime` i `gc_probability`. Vrednosti ovih parametara trebalo bi da budu odabrane tako da obezbeđuju ravnotežu između potreba aplikacije i performansi sistema.

5.3 Autentifikacija korisnika

Autentifikacija korisnika je provera da li je korisnik zaista onaj za koga se predstavlja. Najčešći način ostvarivanja autentifikacije je pomoću imena i lozinke. Korisnik se predstavlja svojim korisničkim imenom, dok se provera da li je li stvarno on izvršava pomoću lozinke.
Pri prvom pristupanju korisnika aplikaciji, od korisnika se traži unos korisničkog imena i lozinke. Nakon toga se proverava postoji li korisnik s tim korisničkim imenom i tom lozinkom u bazi podataka. Ako ne postoji, pristup ostatku aplikacije mu se onemogućuje dok ne upiše tačno ime i lozinku. (Dozvoljeni broj pokušaja upisivanja imena i lozinke se često ograničava).
Ako je korisnik upisao odgovarajuće ime i lozinku, omogući će mu se pristup ostatku aplikacije. Na ovom mestu će se najčešće, korišćenjem sesijske promenljive, zapisati da je autentikacija korisnika izvršena. Zahvaljujući tome, korisnik neće morati ponovno upisivati lozinku prilikom svakog novog zahteva.
Kad sesija istekne, korisnik će morati ponovno da se autentifikuje upisivanjem korisničkog imena i lozinke.
Postavlja se pitanje je li moguće da korisnik podmetne lažni identifikator sesije (koji može promeniti izmenom kolačića ili URL-a) i na taj način se predstavi kao već autentifikovan korisnik. Ta je mogućnost matematički vrlo malo verovatna, zato što svaka PHP sesija dobija nasumični identifikator, koji se zatim enkriptira.
Jedna od mogućnosti za neovlašćeni pristup aplikaciji je tzv. *session hijacking*. Ako napadač presretne klijentov HTTP zahtev, iz njega može preuzeti identifikator sesije i predstaviti se kao taj klijent. Ista opasnost postoji i ako napadač presretne HTTP zahtev koji sadrži lozinku upisanu u obrazac. Zbog toga se u slučajevima kad je potrebna veća sigurnost koristi HTTPS protokol.

I lozinke se u bazi podataka često drže u kriptovanom obliku, da neko ko ima pristup bazi ne bi mogao preuzeti lozinke drugih korisnika.
Za enkripciju lozinki, (kao i identifikatora sesije), koristi se posebna metoda enkripcije – izračunavanje sažetka poruke (*message digest, hash*) koja je jednosmerna. To znači da se iz zapisa u bazi ne može dobiti originalna lozinka, ali se iz lozinke može lako dobiti njen enkriptirani oblik i tako proveri i da li je upisana lozinka tačna.
Više o kolačićima i sesijama može se pročitati na [1], [4] [7].

6 Rad sa regularnim izrazima

Pomoću regularnih izraza (engl. *regular expresions*) može se utvrditi da li neki podaci tipa string zadovoljavaju određene složene šablone (datum, adresa e-pošte,...). Prilično su nečitki, ali veoma korisni u radu sa znakovnim podacima.
Funkcije za rad sa regularnim izrazima omogućavaju čak i izdvajanje i zamenu složenijih podvrednosti unutar zadatog stringa, kao i proveru pojavljivanja uzorka u delu teksta.
Funkcije za rad sa regularnim izrazima treba izbegavati kad god je to moguće zato što troše dosta resursa (prostornih i vremenskih).
PHP podržava regularne izraze preko „*preg_*" familije funkcija:

- *preg_match*,
- *preg_match_all*,
- *preg_replace*,
- *preg_split*,
- *preg_grep*,
- *preq_quote*.

Funkcija *preg_match* prihvata 2 argumenta: String uzorak i String izvor. Funkcija proverava da li se unutar stringa izvor, nalazi izraz koji odgovara regularnom izrazu uzorak. Ako se nalazi, funkcija vraća `true`, u suprotnom vraća `false`.
Funkcija *preg_match_all* prihvata 3 argumenta: String uzorak, String izvor i Array niz. Funkcija pronalazi sva pojavljivanja izraza koji odgovara regularnom izrazu uzorak unutar stringa izvor i upisuje ih u niz, i vraća broj tih pojavljivanja.

Napomena 6 *Funkcija* `preg_match` *zaustavlja pretraživanje nakon prvog pojavljivanja, dok funkcija* `preg_match_all` *nastavlja pretragu do kraja stringa i pronalazi sva pojavljivanja.*

Funkcija *preg_replace* prihvata 3 argumenta: String uzorak, String zamena i String izvor. Sva pojavljivanja izraza koja odgovara regularnom izrazu uzorak unutar stringa izvor, zamenjuje sa stringom zamena. Na kraju, funkcija vraća izmenjeni string izvor.
Funkcija *preg_split* prihvata 2 argumenta: String uzorak i String izvor. Funkcija razdvaja

string izvor u različite elemente, granice svakog elementa zasnovane su na pojavi uzorka u stringu.
Funkcija *preg_grep* prihvata 2 argumenta: String uzorak i Array izvor. Funkcija vraća niz koji se sastoji od elemenata ulaznog niza izvor, koji odgovaraju datom obrascu.
Funkcija *preg_quote* prima samo argument String izvor. Fnkcija dodaje karakter „\" ispred svakog karaktera u stringu izvor koji ima posebno značenje u regularnim izrazima.

Sintaksa regularnih izraza

Sintaksa regularnih izraza se sastoji od raznih slovnih ili drugih karaktera. Svaki karakter definiše neku akciju nad stringovima. Takođe ti karakteri mogu međusobno maltene neograničeno da se kombinuju i pri tome rezultiraju razne radnje nad stringovima. Karakteri koji u regulartnom izrazu imaju specijalno značenje nazivaju se metakarakterima. To su:
`. * ? + [ ] ( ) { } ^ $ | \`
Ostali karakteri unutar regularnog izraza predstavljaju sami sebe. Ukoliko se metakarakteri koriste kao obični literali, potrebno je ispred njih dodati "\" (*backslash*). Na primer, da bi se tačka koristila kao literal zapisuje se kao „\.", ili da bi se karakter „\" koristio kao literal zapisuje se kao „\\".

Karakterske klase

Karakterske klase omogućuju raspolaganje s jednim od definisanih karaktera unutar skupa. Za definisanje karakterske klase koriste se uglaste zagrade (`[ ]`) i unutar njih sve znakove koji se mogu pojaviti. Na primer, definisani regularni izraz „ka[fs]a" se podudariti sa reči „kafa", ali i sa reči „kasa".

- `[.]` - Bilo koji karakter osim karaktera za novi red „\n".
- `[abc]` - Samo slova a, b, c.
- `[a-z]` - Sva mala slova od a do z.
- `[A-Z]` - Sva velika slova od A do Z.
- `[a-zA-Z]` - Sva slova od a do z ili od A do Z.
- `[0-9]` - Sve cifre.

Unutar uglastih zagrada karakter „`-`" ima specijalno značenje, označava opseg.

Logički opepratori	
`^`	operator negacije
`|`	operator ili
`&&`	operator i

- `[^0-9]` - string ne sme da sadrži cifre.
- `[a-z&&[^aeiou]]` - suglasnik, jer obuhvata sva slova od a do z osim samoglasnika.

Primer 36 pokazuje kako pronaći da li izraz koji odgovara uzorku postoji u stringu ili ne, koristeći funkciju *preg_match*.

Primer 36: Provera da li izraz odgovara regularnom izrazu

```
<?php
$uzorak = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
if(preg_match($uzorak, $text)){
    echo "Pronađen!";
}
else{
    echo "Nije pronađen.";
}
?>
```

```
Pronađen!
```

Primer 37 pokazuje kako se koristeći funkciju *preg_match_all* mogu pronaći izrazi koji odgovaraju uzorku.

Primer 37: Svi izrazi koji odgovaraju regularnom izrazu

```
<?php
$uzorak = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
$br_pojavljivanja = preg_match_all($uzorak, $text, $izrazi);
echo $br_pojavljivanja . " pojavljivanja su pronadjena"."<br>";
for($i=0; $i<$br_pojavljivanja; $i++){
echo $izrazi[0][$i]."<br>";
}
?>
```

Rezultat primera 37 prikazan je na slici 12.

Slika 12: Rezultat funkcije *preg_match_all*

Predefinisane karakterske klase

Neke karaterske klase se toliko često koriste da postoje skraćenice za njih.

- \d - Svaka cifra. Isto kao `[0-9]`.
- \D - Bilo koji karakter koji nije cifra. Isto kao `[^0-9]`.
- \w - Slovo, podvlaka, ili cifra. Isto kao `[a-zA-Z_0-9]`.
- \W - Bilo koji karakter koji nije ni slovo ni podvlaka ni cifra `[^w]`.
- \s - Belina (\t, \n, \r)

Mogu se koristiti i sledeće skraćenice:

- `[[:alnum:]]` - Bilo koji alfanumerički karakter (isto što i `[a-zA-Z0-9]`).
- `[[:alpha:]]` - Bilo koje slovo (isto što i `[a-zA-Z]`).
- `[[:digit:]]` - Bilo koji broj (isto što i `[0-9]`).
- `[[:upper:]]` - Bilo koje veliko slovo (isto što i `[A-Z]`).
- `[[:lower:]]` - Bilo koje malo slovo (isto što i `[a-z]`).

Primer 38 pokazuje kako pronaći belinu i zameniti je sa karakterom „-” pomoću funkcije *preg_replace*.

Primer 38: Poziv funkcije preg_replace

```
<?php
$uzorak = "/\s/";
$zamena = "-";
$text = "Zemlja se okreće\noko\tSunca";
echo preg_replace($uzorak, $zamena, $text);
?>
```

Rezultat primera 38:

```
Zemlja-se-okreće-oko-Sunca
```

Kvantifikatori

Kvantifikatori su brojevni mehanizmi s kojima se metakarakteri ili skupovi karaktera mogu ponavljati. Simbole kvantifikatora i njihov opis, sadrži sledeća lista:

- p+ - jedno ili više pojavljivanja p,
- p* - nula, jedno ili više pojavljivanja p,
- p? - nula ili jedno pojavljivanje p,

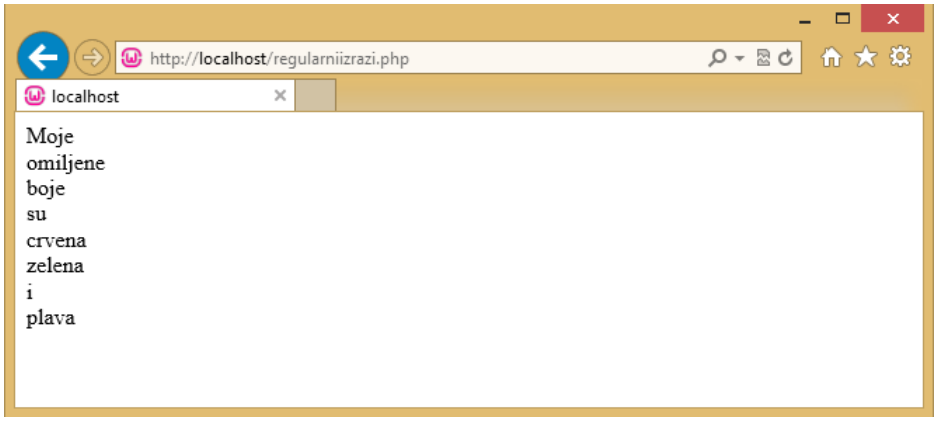
- p{2} - tačno 2 pojavljivanja p,
- p{2,3} - najmanje 2 a najviše 3 pojavljivanja p,
- p{2,} - najmanje 2 pojavljivanja p,
- p{,3} - najviše 3 pojavljivanja p,

Karakteri `? , * , + , { }` se mogu pojavljivati iza karaktera, liste karaktera ili podizraza regularnog izraza (razdvojenog zagradama) i u tom slučaju označava određeni broj ponavljanja karaktera, liste znakova, podizraza regularnog izraza .
Primer 39 pokazuje kako pomoću funkcije *preg_split* podeliti string u zarezima, razmacima i njihovim kombinacijama.

Primer 39: Primena funkcije preg_split

```
<?php
$uzorak = "/[\s,]+/";
$text = "Moje omiljene boje su crvena, zelena i plava";
$delovi = preg_split($uzorak, $text);
foreach($delovi as $deo){
    echo $deo . "<br>";
}
?>
```

Rezultat primera prikazan je na slici 13.



Slika 13: Primena kvantifikatora `+` na `[\s,]`

Primer regularnog izraza je definisanje izraza koji će se podudarati sa telefonskim brojem koji može biti zapisan u bilo kojem od 3 formata: $123-456-789$, 123 654 879, ili 987654321.
Regularni izraz napisan pomoću kvantifikatora bi izgledao ovako:

```
"\d{1,3}[-\s]?\d{3}[-\s]?\d{3}"
```

Primeniti regularni izraz s kvantifikatorom kao što je ovaj „'.+'" (uočiti jednostruke navodnike kao deo izraza) na sledeći tekst: „srpski 'čelik', engleski 'steel'". Moglo bi se očekivati da će kao rezultat biti pronađene reči 'čelik' i 'steel', ali ono što će se zapravo dobiti biće deo rečenice koji glasi „'čelik', engleski 'steel'". Takvo ponašanje naziva se *pohlepno ponašanje kvantifikatora*.
Pohlepno ponašanje se pretpostavlja u svim kvantifikatorima. Pohlepan kvantifikator će pokušati da se uskladi s koliko god je više moguće znakova, kako bi imao najveći mogući rezultat podudaranja.
Pravilan regularni izraz za gornji primer glasio bi: „'.+?'", koji bi dao rezultat: 'čelik', 'steel'.

Pozicija sidra

Pomoću sidra u regularnom izrazu može se zadati da uzorak treba da se nalazi na početku ili na kraju vrednosti koja se ispisuje.

Tabela 4: Pozicija sidra

^	sidri uzorak na početak znakovne vrednosti
$	sidri uzorak na kraj znakovne vrednosti

Obe vrste sidra se koriste u regularnom izrazu u kome se zahteva potpuno poklapanje. U primeru 40 biće prikazana samo ona imena iz niza koja počinju slovom 'J'.

Primer 40: Sidro na početku vrednosti

```
<?php
$uzorak = "/^J/";
$niz = array("Jhon Carter", "Clark Kent", "John Rambo");
$imena = preg_grep($uzorak, $niz);
foreach($imena as $ime){
    echo $ime . "<br>";
}
?>
```

```
Jhon Carter
John Rambo
```

Više o regularnim izrazima može se pročitati na [9],[1], [4].

7 Validacioni filteri

Filteri se koriste za proveru ispravnosti spoljnog unosa. Podaci u neispravnom obliku mogu dovesti do pucanja veb stranice. Korišćenje filtera garantuje da će podatak biti unet u traženom obliku. Postoje 2 tipa filtera: filteri koji proveravaju ispravnost oblika unosa i filteri koji vrše korekciju unosa.
PHP ima ugrađene filtere koji se ostvaruju pomoću funkcije *filter_var*. Ona prima 3 argumenta, prva dva argumenta su obavezna, dok je treći argument je opcion.
Prvi argument je promenljiva koja se proverava. Drugi argument je naziv filtera koji se koristi. Treći argument je niz opcija koje se odnose na filter. Rezultat funkcije je ili `false` ako je greska, ili filtriran podatak.

Ispravljanje stringa

U primeru 41, funkcija *filter_var* se koristi za brisanje svih HTML etiketa iz stringa, uz pomoć filtera FILTER_SANITIZE_STRING.

Primer 41: Brisanje HTML etiketa

```
<?php
$str = "<h1>Hello world</h1>";
$str = filter_var($str,FILTER_SANITIZE_STRING);
echo $str;
?>
```

Rezultat:

```
Hello world!
```

Proveravanje da li je unet ceo broj

U primeru 42, funkcija *filter_var* se koristi za proveru da li je vrednost promenljive ceo broj, uz pomoć filtera FILTER_VALIDATE_INT. Ako jeste, ispisuje se poruka: „Jeste ceo broj", a ako nije, ispisuje se poruka: „Nije ceo broj".

Primer 42: Provera da li je unet ceo broj

```
<?php
$int = 150;
if(!filter_var($int,FILTER_VALIDATE_INT)===false){
echo "Jeste ceo broj";
}
else{
echo "Nije ceo broj";
}
?>
```

Rezultat primera je:

```
Jeste ceo broj
```

U primeru 42, ako se za promenljivu `$int` postavi vrednost na nula, (`$int = 0`) funkcija bi vratila rezultat: Nije ceo broj. Ovaj problem se rešava na sledeći način:

Primer 43: Provera da li je nula ceo broj

```
<?php
$int = 0;
if(filter_var($int,FILTER_VALIDATE_INT)===0 || !filter_var($int,FILTER_VALIDATE_INT)===
  ↪ false){
echo "Jeste ceo broj";
}
else{
echo "Nije ceo broj";
}
?>
```

Sada će rezultat biti tačan.

```
Jeste ceo broj
```

Ispravan e-mail

U primeru 44 funkcija *filter_var* se koristi prvo za brisanje nedozvoljenih karaktera iz promenljive `$email`, a zatim za proveru da li je e-mail u ispravnom obliku. Za brisanje nedozvoljenih karaktera kao filter se koristi FILTER_SANITIZE_EMAIL, dok se za proveru ispravnosti oblika kao filter koristi FILTER_VALIDATE_EMAIL.

Primer 44: Korekcija i provera da li je e-mail ispravan

```
<?php
$email = "marko.markovic@@primer.com";
$email = filter_var($email,FILTER_SANITIZE_EMAIL);
if(!filter_var($email,FILTER_VALIDATE_EMAIL)===false){
echo "$email je ispravna e−mail adresa";
}
else{
echo "$email nije ispravna e−mail adresa";
}
?>
```

Rezultat:

```
marko.markovic@@primer.com nije ispravna e−mail adresa
```

marko.markovic@@primer.com nije ispravna adresa jer u svom zapisu sadrži 2 karaktera „@”, što nije dozvoljeno.

Ispravan URL

U primeru 45 funkcija *filter_var* se koristi prvo za brisanje svih nedozvoljenih karaktera iz promenljive `$url`, a zatim za proveru da li je `$url` u ispravan URL. Za brisanje nedozvoljenih karaktera URL-a se kao filter koristi FILTER_SANITIZE_URL, dok se za proveru ispravnosti oblika URL-a koristi FILTER_VALIDATE_URL.

Primer 45: Provera ispravnosti URL-a

```
<?php
$url = "https://edusoft.matf.bg.ac.rs/eskola_veba/#/home";
$url = filter_var($url,FILTER_SANITIZE_URL);
if(!filter_var($url,FILTER_VALIDATE_URL)===false){
echo "$url je ispravan URL";
}
else{
echo "$url nije ispravan URL";
}
?>
```

Rezultat:

```
https://edusoft.matf.bg.ac.rs/eskola_veba/#/home je ispravan URL
```

Provera da li je ceo broj u određenom opsegu

U primeru 46, funkcija *filter_var* se koristi za proveru da li je u pitanju ceo broj i da li je između 1 i 200, uz pomoć filtera FILTER_VALIDATE_INT. Kao treći argument funkcije *filter_var* se prosleđuje niz koji sadrži informacije o opsegu.

Primer 46: Provera da li je broj između 1 i 200

```
<?php
$int = 122;
$min = 1; $max = 200;
if(!filter_var($int, FILTER_VALIDATE_INT, array("options"=> array("min_range"=>$min,"
  ↪ max_range"=>$max)))===false){
echo "Promenljiva je u traženom opsegu";
}
else{
echo "Promenljiva nije u traženom opsegu";
}
?>
```

Rezultat:

```
Promenljiva je u traženom opsegu.
```

Brisanje karaktera čiji je ASCII kôd veći od 127

U primeru 47, funkcija *filter_var* se koristi za brisanje svih HTML etiketa iz stringa, kao i svih karaktera čiji je ASCII kôd veći od 127. Za brisanje svih karaktera čiji je ASCII kôd veći od 127 se kao treći argument funkcije *filter_var* prosleđuje FILTER_FLAG_STRIP_HIGH.

Primer 47: Brisanje karaktera čiji je ASCII kôd veći od 127

```
<?php
$str = "<h1>Slova srpske ćirilice biće izbrisana!</h1>";
```

```
3   $newstr = filter_var($str, FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_HIGH);
4   echo $newstr;
5   ?>
```

```
1   Slova srpske irilice bie izbrisana!
```

Više o filterima na [8].

8 Rad sa greškama

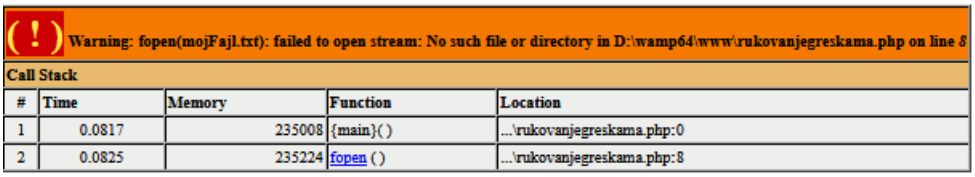
Rad sa greškama predstavlja proces hvatanja grešaka koje pokreće program i zatim preuzimanje odgovarajućih mera. Ukoliko se greške ne obrade, može doći do nepredviđenih posledica.

8.1 Rukovanje greškama

Rukovanje greškama veoma je bitan element u svakom programskom jeziku, na praktično svim platformama. Nikada nije potpuno sigurno šta će i kada korisnik uraditi, kao ni kada će sistem i njegovi resursi, iz nekog razloga, omesti program u pravilnom izvršavanju.
Upravo iz tog razloga, obaveza programera je da predvidi koje tačke u programu predstavljaju potencijalnu opasnost po njegovo izvršavanje i da te pozicije obezbedi dodatnim zaštitnim sistemima.
Najbolnije tačke jedne aplikacije su trenuci kada se ta aplikacija obraća resursima. A ti resursi, odnosno njihovo poreklo, najčešće je baza podataka ili fajl sistem. Pored njih, kritični trenuci događaju se i prilikom svakog korisničkog ulaza u aplikaciju (unos podataka), komunikacije sa uređajima...
Postoje 2 tipa grešaka. Prvi tip su eksterne greške. One nastaju kada nije moguće uspostaviti konekciju sa serverom. Uvek mogu da se jave i uvek o njima treba voditi računa. Drugi tip grešaka su greške u kodu. One nastaju kada program ne radi na način na koji bi trebalo da radi. Obično se ne zna gde mogu da se jave i krivci su sami programeri.

Ručna obrada grešaka

Pre nego što se pristupi nekom resursu, najbolje je ispitati da li taj resurs postoji i na osnovu toga nastaviti delovanje programa. Pri pokušaju otvaranja datoteke mojFajl.txt, `$file=fopen("mojFajl.txt","r");`, ukoliko se datoteka ne nalazi u folderu aplikacije, ovaj kôd će izazvati grešku, i program će izbaciti upozorenje kao na slici 14.



Warning: fopen(mojFajl.txt): failed to open stream: No such file or directory in D:\wamp64\www\rukovanjegreskama.php on line 8

Call Stack

#	Time	Memory	Function	Location
1	0.0817	235008	{main}( )	...\rukovanjegreskama.php:0
2	0.0825	235224	fopen ( )	...\rukovanjegreskama.php:8

Slika 14: Upozorenje da datoteka ne postoji

Zato bi prethodno trebalo proveriti da li datoteka postoji i u odnosu na to, obezbediti ga, kao što je urađeno u primeru 48.

Primer 48: Ručna provera da li datoteka postoji

```
<?php
if(!file_exists("mojFajl.txt "))
{
die("Nepostojeci fajl");
 }
else
 {
$file=fopen("mojFajl.txt ","r");
 }
?>
```

Sada će rezultat biti:

```
Nepostojeći fajl
```

Funkcija *die* prekida izvršenje programa uz emitovanje definisane poruke. Ukoliko dalje izvršenje programa zavisi od resursa koje treba da se iščita (na primer od datoteke mojFajl.txt u prethodnim primerima), onda svakako treba prekinuti dalje izvršavanje programa, ukoliko tog resursa nema.

Emitovanje sopstvenih grešaka

U primeru 48, izvršavanje programa je prekinuto funkcijom *die*. Kada program prijavi grešku, zaustavljanje programa nekad nije najbolje rešenje. Moguće je napisati svoju funkciju koja upravlja greškama. Moguće je odabrati svoj naziv za funkciju. Funkcija može da primi 5 argumenata od kojih su prva 2 obavezna. Njena sintaksa izgleda ovako:

```
handle_function(err_level, err_message, err_file, err_line, err_context);
```

Argumenti funkcije su prikazani u tabeli 5.

Tabela 5: Argumenti funkcije handle_function

Argument	Objašnjenje
err_level	Predstavlja nivo greške. Uvek se za vrednost uzima broj. U tabeli 6 pogledati šta koji broj označava.
err_message	Predstavlja poruku koja se šalje korisniku.
err_file	Predstavlja ime fajla u kome se javila greška.
error_line	Predstavlja liniju u kojoj je pronađena greška.
err_context	Predstavlja niz koji sadrži sve promenljive i njihove vrednosti koje su aktivne u trenutku kada se javila greška.

Tabela 6: Nivo greške

Broj greške	Nivo greške	Objašnjenje
1	E_ERROR	Greška od koje se program ne ume oporaviti i koja uslovljava prekidanje izvršavanja programa.
2	E_WARNING	Ne prekida izvršenje programa, ali ukazuje da se nešto loše dogodilo.
8	E_NOTICE	Program je pronašao potencijalnu grešku.

Sada je moguće napisati sopstvenu funkciju koja upravlja greškama, kao što je urađeno u primeru 49.

Primer 49: Sopstvena funkcija za upravljanje greškama

```
1   function mojHendler($brojGreške, $porukaGreške)
2 {
3  echo "Greška: [$brojGreške] $porukaGreške";
4 }
```

Nakon što se napravi funkcija kojom bi se rukuvalo greškama, potrebno je da se ona inicijalizuje kao podrazumevani rukovalac greškama (engl. *error handler*). Inicijalizovanje se vrši pomoću funkcije *set_error_handler* čiji je jedini argument naziv funkcije koja se inicijalizuje kao rukovalac greškama. Inicijalizovanje funkcije `mojHandler` iz primera 49 se vrši naredbom:

```
set_error_handler("mojHendler");
```

Nakon toga, sve greške će biti, umesto u ugrađeni rukovalac greškama, prosleđivane u

ručno kreiranu funkciju `mojHendler`.

Primer 50: Inicijalizovanje sopstvene funkcije

```
<?php
function mojHendler($brojGreške,$porukaGreške){
echo "Error: [$brojGreške] $porukaGreške";
}
set_error_handler("mojHendler");
echo($test);  //promenljiva koja nije definisana prouzrokovaće grešku E−NOTICE.
?>
```

U primeru 50 se inicijalizuje funkcija `mojHendler` kao rukovalac greškama i definiše sopstvena poruka o grešci.
Rezultat primera 50:

```
Error: [8] Undefined variable: test
```

Greške iz tabele 6, moguće je priključiti svom rukovaocu greškama kroz argument funkcije *set_error_handler*:

```
set_error_handler("mojHendler",E_NOTICE);
```

U ovom slučaju, kroz rukovalac (engl. *handler*), proći će samo greške tipa E_NOTICE, dok će ostale greške biti obrađivane kroz ugrađeni rukovalac.

Funkcija *trigger_error*

Programeru je ostavljena mogućnost da razmotri i uvede neke svoje kategorizacije grešaka. Brojevi i nivoi tih grešaka dati su u tabeli 7.

Tabela 7: Nivo greške

Broj greške	Nivo greške
256	E_USER_ERROR
512	E_USER_WARNING
1024	E_USER_NOTICE

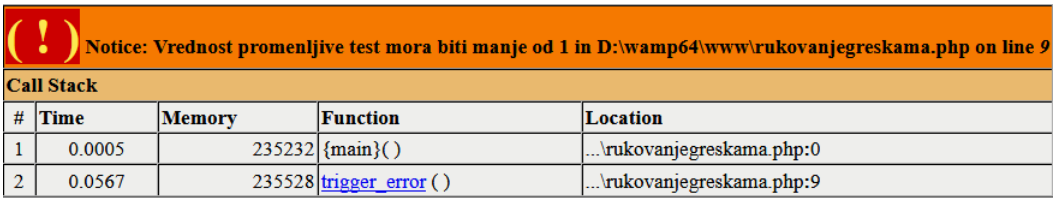
One su analogne greškama E_ERROR, E_WARNING, E_NOTICE, i mogu se prijaviti korišćenjem funkcije *trigger_error*. Koriste se u programu gde korisnik unosi podatke, gde je korisno aktivirati grešku, kada dođe do neželjenog unosa. Poziv funkcije *trigger_error* je prikazan u primeru 51.

Primer 51: Poziv funkcije trigger_error

```
<?php
$test = 2; //umesto unosa odmah se promenljivoj dodaje vrednost
```

```
3  if($test>=1){
4  trigger_error("Vrednost promenljive test mora biti manje od 1");
5  }
6  ?>
```

Prikaz greške sa porukom dat je na slici 15.



Notice: Vrednost promenljive test mora biti manje od 1 in D:\wamp64\www\rukovanjegreskama.php on line 9

Call Stack

#	Time	Memory	Function	Location
1	0.0005	235232	{main}( )	...\rukovanjegreskama.php:0
2	0.0567	235528	trigger_error ( )	...\rukovanjegreskama.php:9

Slika 15: Prikazivanje greške sa porukom koja je razumljiva korisniku

U razvojnoj fazi aplikacije prikazivanje greški je korisno, ali u aktivnim aplikacijama nikako. Naime, greške zbunjuju korisnika i ostavljaju loš utisak. Takođe greške mogu odati važne informacije i ugroziti bezbednost aplikacije. Zato je potrebno napraviti funkciju koja rukuje greškama, i inicijalizovati je kao podrazumevani rukovalac greškama. Definisanje sopstvenog rukovaoca greškama i sopstvene poruke o grešci prikazano je u primeru 52.

Primer 52: Sopstveni rukovalac greškama i sopstvena poruka

```
1  <?php
2  function handleError($brojGreške,$porukaGreške){
3  echo "Greška: [$brojGreške] $porukaGreške";
4  }
5  set_error_handler("handleError");
6  $test = 2; //umesto unosa odmah se promenljivoj dodaje vrednost
7  if($test>=1){
8  trigger_error("Vrednost promenljive test mora biti manje od 1");
9  }
10 ?>
```

Sada će rezultat biti:

```
1  Greška: [1024] Vrednost promenljive test mora biti manje od 1
```

Tip na ovaj način emitovane greške, biće 1024 (Notice), što možda nije nivo ozbiljnosti koji bi predočili korisniku. Da se ovo promeni, može se kao drugi argument uneti jedan od tri tipa grešaka koje ova funkcija prihvata i čije konvencije bi trebalo poštovati pri korišćenju.
Primer:

```
1  trigger_error("Vrednost promenljive test mora biti manje od 1", E_USER_WARNING);
```

Dodavanjem drugog argumenta funkciji *trigger_error* u prethodnom primeru dobija se sledeći rezultat:

```
1  Greška: [512] Vrednost promenljive test mora biti manje od 1
```

Više o rukovanju greškama može se pročitati na [8],[3].

8.2 Izuzeci

Izuzetak obično signalizira grešku ili neki neobičan događaj u programu koji zaslužuje posebnu pažnju. Ukoliko se usred koda koji rukuje normalnim operacijama programa, istovremeno rukuje mnoštvom greškama, struktura programa postaje kompliovana i teška za razumevanje.
Glavna korist od izuzetaka jeste što oni razdvajaju kôd koji obrađuje greške od koda koji se izvršava kada stvari idu glatko. Ne treba sve greške u programima signalizirati izuzecima, samo neuobičajene ili katastrofalne. Primer kada nema potrebe koristiti izuzetke je neispravan ulazni podatak. Za to ne treba koristi izuzetke, jer se radi o uobičajenom događaju.
Razlog za takvu odluku je što rukovanje izuzecima uključuje mnogo dodatnog procesiranja, pa će program koji rukuje izuzecima većinu vremena, biti sporiji nego što bi mogao da bude.
PHP pruža moćan mehanizam za rukovanje izuzecima. Za razliku od tradicionalnog pristupa rukovanja greškama, rukovanje izuzecima je objektno orjentisan metod za rukovanje greškama. Rukovanje izuzecima je podržano u programskom jeziku PHP od verzije PHP 5.
Za izuzetak se kaže da je „izbačen" (engl. *thrown*). Za kôd koji prima izuzetak kao parametar, se kaže da ga „hvata" (engl. *catch*). Kada se izbaci izuzetak, kôd koji sledi se neće izvršiti, a PHP će pokušati da pronađe odgovarajući blok „catch". Ako izuzetak nije uhvaćen, fatalna greška će biti izdata sa porukom „Uncaught Ekception".
Primer 53 je primer gde je izuzetak izbačen, ali nije uhvaćen.

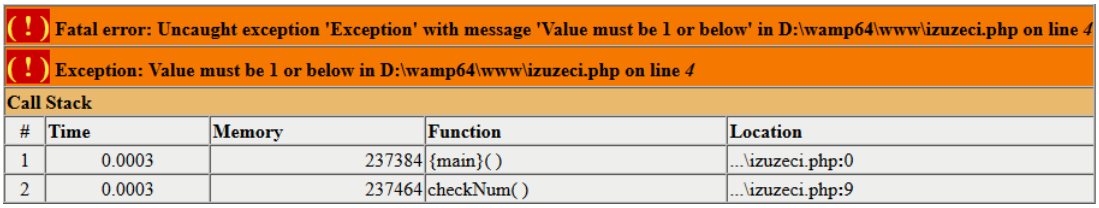
Primer 53: Izuzetak je izbačen, a da nije uhvaćen

```
1  <?php
2  function checkNum($number) {
3  if($number>1) {
4  throw new Exception("Value must be 1 or below");
5  }
6  return true;
7  }
8
9  checkNum(2);
10 ?>
```

Kôd će prouzrokovati fatalnu gresku, greška je prikazana na slici 16.

Rukovanje izuzecima

Da bi izbegli grešku iz primera 53, potrebno je napisati odgovarajući kôd koji će da hvata izbačeni izuzetak. Za rukovanje izuzecima tamo gde se oni dese, potrebno je uključiti 2 bloka koda i jednu komandu.



Fatal error: Uncaught exception 'Exception' with message 'Value must be 1 or below' in D:\wamp64\www\izuzeci.php on line 4

Exception: Value must be 1 or below in D:\wamp64\www\izuzeci.php on line 4

Call Stack

#	Time	Memory	Function	Location
1	0.0003	237384	{main}( )	...\izuzeci.php:0
2	0.0003	237464	checkNum( )	...\izuzeci.php:9

Slika 16: Izuzetak je izbačen, a da nije uhvaćen

- `try` blok- Obuhvata kôd gde se može javiti jedan ili više izuzetaka. Naredbe koje mogu da izbace izuzetak koji treba uhvatiti, moraju biti unutar `try` bloka.
- `throw` - komanda kojom se izbacuje izuzetak. Svakoj komandi `throw` odgovara bar jedna komanda `catch`.
- `catch` blok - Obuhvata kôd koji hvata izuzetak i pravi objekat koji sadrži informacije o izuzetku.

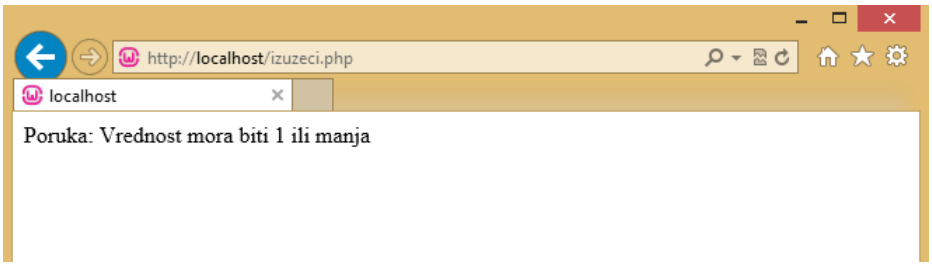
Kada dođe do ispaljivanja izuzetka, sve naredbe koje dolaze iza naredbe koja je proizvela izuzetak se preskaču, a program nastavlja izvršenje u `catch` bloku koji upravlja tim izuzetkom. Tada se izvršava složena naredba `catch` bloka koja je izabrana a zatim se izvršsavanje programa nastavlja u naredbi koja sledi iza poslednjeg `catch` bloka u listi. Ako ne postoji `catch` blok, program će izbaciti grešku kao u primeru 53. Primer 54 pokazuje korišćenje `try`, `catch` sintakse, za obradu grešaka.

Primer 54: Hvatanje izuzetka

```
<?php
//kreiranje funkcije sa izuzetkom
function checkNum($number) {
if($number>1) {
throw new Exception("Vrednost mora biti 1 ili manja");
}
return true;
}
// kod koji može da ispali izuzetak nalazi se u try bloku
try {
checkNum(2);
//Ako je izbačen izuzetak, naredni tekst neće biti ispisan
echo 'Ako je vidljivo ovo, broj je 1 ili manji';
}
//hvatanje izuzetka
catch(Exception $e) {
echo 'Poruka: ' .$e->getMessage();
}
?>
```

U primeru 54 je izbačen i uhvaćen izuzetak, jer je kao argument funkcije `checkNum` prosleđen broj veći od 1. `Catch` blok hvata izuzetak i kreira objekat ($e) koji čuva informacije o izuzetku. Poruka o grešci se poziva pomoću `$e->getMessage()`. Ona je

prikazana na slici 17. Nijedna komanda u `try` bloku navedena posle izbačenog izuzetka neće biti izvršena.



Slika 17: Uhvaćen je izuzetak i ispisana je poruka o grešci

Za pozivanje poruke greške iz izuzetka korišćena je funkcija *getMessage*. Od značaja mogu biti i sledeće funkcije:

- *getFile* - vraća punu putanju do datoteke u kojoj je izbačen izuzetak.
- *getLine* - vraća broj linije u kodu, gde je izbačen izuzetak.
- *getTraceAsString* - vraća vraća niz s podacima o stablu pozivanja koji omogućavaju utvrđivanje mesta na kome je izbačen izuzetak, formatiran kao string.

9 Uvod u objektno orjentisano programiranje

Savremeni programski jezici najčešće podržavaju ili čak zahtevaju objektno orijentisani pristup razvoju softvera. Sada objektno orijentisana implementacija programskog jezika PHP ima sve karakteristike koje se očekuju od potpuno objektno orijentisanog jezika.
Objektno orjentisan način progrmiranja ima nešto drugačiju logiku od proceduralnog, prvobitnog načina u programskom jeziku PHP. U objektnom pristupu mnogi složeni i komplikovani zadaci se znatno lakše rešavaju nego u standardnom obliku i mnogo se lakše održavaju i menjaju.
Ključni koncepti objektno orjentisanog programskog jezika su baziranost na objektima i klasama, podržavanje nasleđivanja i podržavanje polimorfizma.

9.1 Objekti i klase

Objektno orjentisan softver se sastoji od više samostalnih objekata koji su međusobno povezani atributima i operacijama. Atributi su svojstva ili promenljjive koje se odnose na objekat. Operacije su metode, radnje ili funkcije koje objekat može da izvršava.
Objekat u programskom jeziku PHP može biti bilo šta što je vezano za softver, na primer neki fajl, polje za unos teksta, ili neki objekti iz stvarnog sveta koje treba predstaviti u softveru, na primer neki proizvodi, kupci itd.
Objekti se mogu grupisati u klase. Svaka klasa predstavlja skup određenih objekata

koji imaju izvestan broj zajedničkih karakteristika. Jedna klasa predstavlja sve objekte koji mogu da obave iste operacije, koji se identično ponašaju i imaju iste atribute koji predstavljaju iste stvari, mada se vrednosti tih atributa mogu razlikovati od jednog objekta do drugog.

Struktura klase

Deklaracija klase se sastoji od ključne reči `class` i imena klase. Najjednostavnija definicija klase prikazana je u primeru 55.

Primer 55: Prazna klasa

```
1  class ime{
2
3  }
```

Klase bez atributa i operacija su beskorisne. Kroz atribute objekta se definišu podaci koji opisuju stanje, odnosno osobine objekta. Atributi klase se definišu tako što se unutar klase deklariše promenljiva ispred koje stoji rezervisana reč `var`. Primer 56 definiše klasu „ime" s dva atributa, `$atribut1` i `$atribut2`.

Primer 56: Definisanje atributa klase

```
1  class ime{
2  var $atribut1;
3  var $atribut2;
4  }
```

Operacije se deklarišu kao funkcije unutar definicije klase. Funkcija se kreira pomoću rezervisane reči `function`. Funkcija mora da ima svoje ime, a može imati proizvoljan broj argumenata. Vrednosti izvršavanja funkcije mogu biti vraćene opcionom naredbom `return`.
U primeru 57 definiše se klasa koja ima dve operacije. Operacija `operacija1` nema parametre, dok `operacija2` ima dva parametra:

Primer 57: Definisanje operacija klase

```
1  class ime{
2  function operacija1() {}
3  function operacija2($arg1, $arg2) {}
4  }
```

Funkcije imaju posebnu ulogu u klasama. Funkcije, sastavni delovi klasa, nazivaju se *metodi* ili *metode*.
Specifične akcije koje će objekat moći da uradi se definišu u klasi u njenim metodama. Cilj metoda je da menjaju vrednosti atributa objekta i tako menjaju stanje, odnosno osobine objekta.

Konstruktori

Funkcija (ili funkcije) koja učestvuje u kreiranju klase naziva se konstruktor. Ona se automatski poziva pri kreiranju objekta klase.
Bilo koji kôd koji treba da se izvrši prilikom prve pojave objekta treba da se smesti u konstruktor. Konstruktor se najčešće koristi za inicijalizaciju vrednosti, to jest za dodeljivanje početnih vrednosti atributima.
Konstruktor se deklariše kao i ostale operacije, ali ima specijalno ime *construct*. U ranijim verzijama se ime konstruktora poklapalo sa imenom klase, ali od verzije 7.0 ova sintaksa je zastarela (engl. *deprecated*), i u budućnosti će biti potpuno izbačena iz upotrebe. PHP prvo traži konstruktor nazvan *construct*, a ako ga ne nađe traži metod koji ima isto ime kao klasa, da bi i stare klase mogle da se izvršavaju. Konstruktor kao i druge metode može ručno da se pozove ali njegova glavna svrha je da se automatski poziva prilikom pravljenja objekta. U primeru 58 prikazano je kreiranje konstruktora klase ime.

Primer 58: Kreiranje konstruktora

```
1  class ime{
2    function __construct($arg1){
3    echo "Pozvan konstruktor sa argumentom $arg1<br>"
4    }
5  }
```

9.2 Instanciranje klase

Programiranje objekta – člana klase naziva se pravljenje instance ili instanciranje klase. Objekat se pravi upotrebom rezervisane reči `new`. Potrebno je navesti klasu i zadati argumente konstruktora.
U primeru 59 definiše klasa ime sa konstruktorom i zatim se prave tri objekta te klase.

Primer 59: Pravljenje 3 instance klase ime

```
1  class ime{
2    function __construct($arg1){
3    echo "Pozvan konstruktor sa argumentom $arg1<br>";
4    }
5  }
6    $a = new ime("Prvi");
7    $b = new ime("Drugi");
8    $c = new ime();
```

Pošto se konstruktor poziva svaki put kada se pravi objekat, rezultat primera 60 izgleda ovako:

```
1  Pozvan konstruktor sa argumentom Prvi
2  Pozvan konstruktor sa argumentom Drugi
3  Pozvan konstruktor sa argumentom
```

Upotreba atributa klase

Unutar klase postoji promenljiva `this` koja se odnosi na samu klasu. Ova promenljiva ne može biti korišćena izvan klase. Dizajnirana je da se koristi u naredbama unutar klase kako bi omogućila pristup promenljivima same klase. Obično se koristi u formatu `$this->atr`, gde je `$atr` jedan atribut klase. Na primer:

Primer 60: Korišćenje promenljive this

```
class ime {
    var $atribut;
    function operacija($parametar){
    //atributu se dodaje vrednost parametra metode
    $this -> atribut=$parametar;
    echo $this -> atribut;
    }
}
```

Napomena 7 *Obratiti pažnju da se znak $ koristi ispred promenljive this, ali ne i ispred promenljive atribut.*

Pošto se u primeru 61 ni na koji način ne ograničava pristup atributima klase, može im se pristupiti van klase, na sledeći način:

Primer 61: Pristup atributima klase, van klase

```
<?php
class ime {
    var $atribut;
}
    $a = new ime();
    $a -> atribut="vrednost";
    echo $a -> atribut;
?>
```

Međutim ovakav direktan pristup atributima izvan klase nije preporučljiv. Jedna od prednosti objektno orjentisanog razvoja je ta što podstiče enkapsuliranje (skrivanje podataka). Umesto direktnog pristupa atributima klase moguće je definisati pristupne funkcije, pomoću kojih bi se pristupalo atributima. U primeru 62 pristupna funkcija `get_nadimak` vraća atribut `$nadimak`. Na taj način je moguće pristupiti van klase atributu klase. Funkcija `set_nadimak` čiji je argument `$nova_vrednost` koristi se da se atributu klase pristupi van klase i dodeli vrednost `$nova_vrednost`.

Primer 62: Pristupne funkcije

```
<?php
class ime {
    var $nadimak = "Pera";
    function get_nadimak(){
    return $this -> nadimak;
    }
```

```
7     function set_nadimak($nova_vrednost){
8         $this -> nadimak=$nova_vrednost;
9         }
10    }
11    ?>
```

Primer 63: Pristup funkcijama iz primera 62

```
1     $a = new ime();
2     echo $a -> get_nadimak()."<br>";
3     $a -> set_nadimak("Neki nadimak");
4     echo $a -> get_nadimak();
```

Rezultat:

```
1     Pera
2     Neki nadimak
```

Pozivanje operacija klase

Operacija klase se poziva na isti način kao i atribut klase. U primeru 64 prikazana je klasa ime.

Primer 64: Klasa ime

```
1     class ime{
2         function operacija1(){
3
4         }
5         function operacija2($parametar1, $parametar2){
6         }
7     }
```

Neka je pomoću naredbe `$a = new ime();` kreiran objekat klase ime.
Operacije se pozivaju navođenjem imena i potrebnih parametara u zagradama. S obzirom da te operacije pripadaju objektima a ne nekoj običnoj funkciji, potrebno je naglasiti kom objektu pripada. Ime objekta se koristi isto kao i prilikom pristupanja atributu tog objekta. U primeru 65 se pozivaju operacije klase ime.

Primer 65: Pozivi operacija iz klase ime

```
1     $x = $a -> operacija1();
2     $y = $a -> operacija2(11,'proba');
```

Funkcije sa opcionim arugmentima

Opcioni argument funkcije se mora nalaziti na kraju liste argumenata i imati podrazumevanu vrednost koja je najčešće `null` ili prazan string. U primeru 66 se u klasi Osoba definiše funkcija čiji je drugi argument opcion, kao i pozivi te funkcije sa opcionim i bez opcionog argumenta.

Primer 66: Funkcija sa opcionim argumentom

```php
<?php
    class Osoba{
        var $ime = "Pera";
        function pozdrav($ime, $prezime = null){
            echo "Zdravo. Ja sam Pera ".$prezime."! <br>";
        }
    }
    $objekat = new Osoba;
    $objekat->pozdrav("Pera", "Peric");
    $objekat->pozdrav("Pera");
    $objekat->pozdrav(); // greska -$ime je obavezno
?>
```

Prva dva poziva funkcije su korektna jer drugi argument može a i ne mora da se navede. Treći poziv funkcije nije korektan jer je prvi argument obavezan, pa nastaje greška.

Zdravo. Ja sam Pera Peric!
Zdravo. Ja sam Pera !

Warning: Missing argument 1 for Osoba::pozdrav(), called in D:\wamp64\www\oop.php on line 11 and defined in D:\wamp64\www\oop.php on line 4

Call Stack

#	Time	Memory	Function	Location
1	0.0811	239496	{main}( )	...\oop.php:0
2	0.0907	239816	Osoba->pozdrav( )	...\oop.php:11

Slika 18: Nije navden prvi argument funkcije `pozdrav`, koji je obavezan

9.3 Modifikatori pristupa

Modifikatori pristupa određuju pravo pristupa članovima klase. Članovi klase (atributi i operacije) mogu imati sledeća prava pristupa:

- public (javni) - članovima klase koji se deklarišu kao javni može se pristupati i unutar i izvan klase;
- protected (zaštićen) - članovima klase koji se deklarišu kao zaštićen može se pristupiti samo unutar te klase i u klasama koje nasleđuju klasu;
- private (privatni) - članovima klase koji se deklarišu kao privatni može se pristupati samo unutar te klase; ne mogu da se nasleđuju. Funkcije označene kao privatne su uglavnom pomoćne funkcije koje se koriste unutar klase.

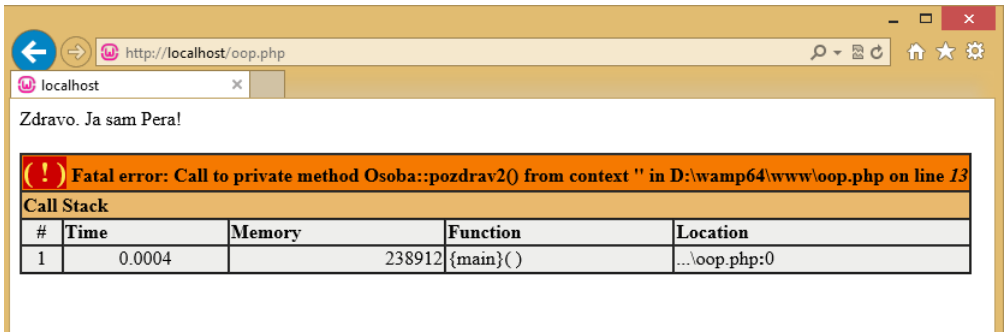
Primer 67 ilustruje privatne i javne metode klase.

Primer 67: privatne i javne metode klase

```php
<?php
    class Osoba{
        var $ime = "Pera";
        public function pozdrav(){ //javni metod
            echo "Zdravo. Ja sam ".$this->ime."! <br>";
```

```
6           }
7           private function pozdrav2(){ //privatni metod
8               echo "Zdravo. Ja sam ".$this->ime."! <br>";
9           }
10      }
11      $objekat = new Osoba;
12      $objekat->pozdrav();
13      $objekat->pozdrav2(); //poziv privatnog metoda van klase proizrokovaće grešku.
14  ?>
```

Rezultat primera 67 prikazan je na slici 19. Kako je metod `pozdrav` javni (public), njemu je moguće pristupiti izvan klase. Kako je metod `pozdrav2` privatni (private), njemu nije moguće direktno pristupiti van klase. Zbog toga će program izbaciti grešku.



Slika 19: Poziv privatnog metoda van klase prouzrokovaće grešku

9.4 Nasleđivanje

Nasleđivanje je bitan deo objektno orjentisanog programiranja. Nasleđivanje predstalja kreiranje klase bazirane na nekoj već definisanoj klasi. Bazna klasa (natklasa) je klasa iz koje se vrši izvođenje. Izvedena klasa naziva se još i direktna potklasa bazne klase. Moguće je izvoditi klasu iz izvedene klase.
Nasleđivanje se obavlja pomoću rezervisane reči `extends`. U klasi koja nasleđuje, dostupna su sva svojstva i metode iz natklase koje nisu privatne. Pored toga, u potklasi mogu biti definisani i novi članovi. Primer nasleđivanja dat je u primeru 68.

Primer 68: klasa Zaposleni nasleđuje klasu Osoba

```
1   <?php
2       class Osoba{
3           public $ime;
4           public function __construct(){
5               echo "Osoba je kreirana<br>";
6           }
7       }
8
9       class Zaposleni extends Osoba{
10          public $plata;
```

```
11          public function opis($ime,$vrednost){
12              $this->ime = $ime; // atribut nasleđen iz natklase
13              $this->plata = $vrednost;
14              echo $this->ime ."-" . $this->plata;
15          }
16      }
17
18      $zaposleni = new Zaposleni();
19      $zaposleni->opis("Pera",5000);
20  ?>
```

Rezultat primera 68 je:

```
1   Osoba je kreirana
2   Pera-5000
```

Kako klasa Zaposleni nasleđuje klasu Osoba, pored toga što nasledjuje atribut `$ime`, nasleđuje i konstruktor klase Osoba. Pored toga, u klasi Zaposleni definisani su i novi atribtut `$plata` i metod `opis`.

Redefinisanje (engl. *Overriding*)

Ponekad je korisno deklarisati iste atribute i operacije kao u nadklasi. To se može uraditi da bi se nekom atributu u potklasi dala drugačija vrednost u odnosu na isti atribut u natklasi ili da bi se operaciji u potklasi promenila funkcionalnost u odnosu na istu operaciju u natklasi. Promena funkcionalnosti u potklasi je redefinisanje. Neka su date klasa Osoba i klasa Zaposleni koja je nasleđuje kao u primerima 69 i 70.

Primer 69: Klasa Osoba

```
1   class Osoba {
2       var $plata = 0;
3       function opis(){
4           echo "Osoba <br>";
5           echo "plata je $this->plata";
6       }
7   }
```

Primer 70: Klasa Zaposleni

```
1   class Zaposleni extends Osoba {
2       var $plata=5000;
3       function opis(){
4           echo "Zaposleni<br>";
5           echo "plata je $this->plata";
6       }
7   }
```

U klasi Zaposleni je promenjena podrazumevana vrednost atributa `$plata`, i funkcionalnost operacije `opis`. Definisanje klase Zaposleni ne utiče na prvobitnu definiciju klase Osoba, atribut `$plata` i metod `opis` u klasi Osoba ostaju nepromenjeni.
Primeri poziva metoda klase Osoba i klase Zaposleni dati su u primerima 71 i 72.

Primer 71: Metoda klase Osoba

```
$osoba = new Osoba();
$osoba->operacija();
```

Rezultat:

```
Osoba
plata je 0
```

Primer 72: Metoda klase Zaposleni

```
$zaposleni = new Zaposleni();
$zaposleni->opis();
```

Rezultat:

```
Zaposleni
plata je 5000
```

Moguće je pristupiti metodama bazne klase uz pomoć operatora :: i rezervisane reči `parent`. Prsitupa se naredbom `parent::metoda`.
Moguće je zabraniti redefinisanje (*overriding*) pojedinih metoda sa ključnom reči `final`. U primeru 73 se redefiniše metod `opis` bazne klase. U primeru se takođe pomoću reči `final` onemogućeno redefinisanje metoda `zabranjeno_redefinisanje`.

Primer 73: Redefinisanje metoda bazne klase

```
<?php
class Osoba{
    var $ime;
    function opis(){
        return "Osoba ";
    }
    final function zabranjeno_redefinisanje(){
        return "Ova metoda ne može da se redefinise";
    }
}
    class Zaposleni extends Osoba{
        function opis(){
            return  parent::opis()."je zaposlena";
        }
    }
    $osoba = new Osoba;
    echo $osoba->opis()."<br>"; //poziv metode iz klase Osoba
    $zaposleni = new Zaposleni();
    echo $zaposleni->opis()."<br>";//poziv redefinisane metode iz klase Zaposleni
    echo $zaposleni->zabranjeno_redefinisanje();
?>
```

Rezultat primera 73:

```
Osoba
Osoba je zaposlena
Ova metoda ne može da se redefinise
```

Ukoliko se kao u primeru 74, pokuša da se redefiniše metod `zabranjeno_redefinisanje` u klasi Zaposleni, program izbacuje grešku, kao na slici 20.

Primer 74: Pokušaj redefinisanja final metoda

```
1  class Zaposleni extends Osoba{
2          function opis(){
3            return  parent::opis().”zaposlena”;
4          }
5          function zabranjeno_redefinisanje(){
6           return ”Možda ipak može da se redefiniše”; //pozivanje metode iz bazne klase kojoj je
       ↪ zabranjeno redefinisanje
7          }
8     }
```

Slika 20: Poziv `final` metoda iz bazne klase prouzrokovaće grešku

9.5 Statička polja klase

Ponekad klase poseduju i elemente koji se mogu „pozvati” i bez instanciranja same klase. Ovakvi elementi nazivaju se statički elementi i mogu biti istog tipa kao i svi ostali elementi jedne klase (metode i atributi). Definicija ovakvih elemenata vrši se ključnom rečju `static`. U primeru 75 je definisana statička promenljiva `$x` i statička metoda `f`.

Primer 75: Primer statičke promenljive i statičke metode

```
1  class mojaKlasa
2  {
3      public static $x=20;
4      public static function f(){
5          echo ”ovo je staticki metod”;
6          }
7  }
```

Statičke promenljive i statičke metode su zajedničke za sve objekte neke klase. Prilikom pozivanja statičkih elemenata, instanca klase nije neophodna, već se elementi pozivaju direknim unosom klase koja ih sadrži, pri čemu, umesto znaka `->` koristi se znak :: (*scope resolution operator).*
Za pozivanje statičkih svojstava i metoda unutar klase koristi se `self::svojstvo`, odnosno `self::metoda`. Za pristup statičkoj metodi ili promenljivoj u okviru iste klase koristi se ključna reč `self`, self je klasama što je $this objektima. Za pozivanje statičkih svojstava i metoda izvan klase koristi se `ImeKlase::svojstvo`, odnosno `ImeKlase::metoda`.

U primeru 76 je prikazano pozivanje statičke promenljive i metode unutar klase.

Primer 76: Pozivanje statičke promenljive i statičke metode

```
<?php
    class Krug{
        public static $pi = 3.14;
        public static function vrati(){
            return self::$pi;
        }
        public static function vrati1(){
            return self::vrati();
            }
    }
    echo Krug::$pi;
    echo Krug::vrati();
?>
```

```
3.14
3.14
```

Statički elementi često mogu biti veoma korisni, ako se radi na velikom projektu, s mnogo klasa. Da bi se pristupilo elementima jedne klase iz druge klase, moraju se ti elementi ručno provlačiti kroz klasu, što može biti veoma bolno i može se dobiti zbrkan kôd što nikako nije poželjno, statički elementi rešavaju taj problem.
Više o objektno orjentisanom programiranju može se pročitati na [1], [6], [7].

10 Obrada XML dokumenata

XML (eXtensible Markup Language) je način za serijalizaciju podataka, odnosno način na koji se jednostavno i brzo mogu zapamtiti podaci (a da pri tom to nije baza podataka) i proslediti ih nekome ko će ih takođe razumeti, jer se poštuju iste konvencije. Svaki XML dokument bi trebalo da poseduje deklaraciju:

```
<?xml version="1.0"? encoding="UTF-8" ?>
```

Deklaracija sadrži osnovne podatke o dokumentu: koja verzija XML-a je u pitanju, koji kodni raspored je korišćen u dokumentu. Opciona je i deklaracija tipa dokumenta (DTD) koja označava skup pravila koja će biti poštovana tokom parsiranja dokumenta. Ova deklaracija može biti napravljena direktno u dokumentu ili učitana iz eksternog izvora.

```
<!DOCTYPE film SYSTEM "test.dtd">
```

Nakon deklaracije sledi koreni (document) element dokumenta, i konačno, serijalizovani podaci. Konačno, cela XML struktura izgleda ovako:

1. XML Deklaracija (XML će funkcionisati i bez deklaracije, ali je poželjno da ona postoji),

2. DTD Deklaracija (za kompleksnije serijalizacije),

3. koreni element (dokument),

4. elementi (lista elemenata),

5. element – atributi i vrednosti (stavke iz liste),

6. podaci (Podaci po stavci).

XML se sastoji od tagova, odnosno, od elemenata i atributa. Tag je selekcija u nekom dokumentu markirana određenim oznakama i nazivom. Ove oznake su < i >, a naziv taga može biti bilo koji tekst.
Na primer: `<mojTag>`
Ovo je pravilno napisan tag, ali nedovoljan da zaokruži jednu celinu u XML dokumentu. Da bi neka celina bila zaokružena u XML-u, potrebna su bar dva taga: jedan otvarajući i jedan zatvarajući, ili, eventualno, jedan „samozatvarajući" tag.

```
<mojTag></mojTag>  ili  <mojTag/>
```

Primer 77 je primer jednog XML dokumenta.

Primer 77: XML dokument

```
<?xml version="1.0" encoding="UTF-8" ?>
<koren>
<država oznakaDržave="sr">
    <naziv>Srbija</naziv>
    <glavniGrad>Beograd</glavniGrad>
    <opis>Opis Srbije.....</opis>
</država>
<država oznakaDržave ="fr">
    <naziv>Francuska</naziv>
    <glavniGrad>Pariz</glavniGrad>
    <opis>Opis Francuske.....</opis>
</država>
</koren>
```

U primeru 77, koren je koreni element dokumenta, država predstavlja jedan element. oznakaDržave je atribut tog elementa, a naziv, glavniGrad i opis su ugnježdeni elementi (podelementi) elementa država.

10.1 XML parseri

Iako XML podaci imaju lako čitljivu strukturu i pri tom jednostavnu za ručno parsiranje, kao i ostali jezici, i u programskom jeziku PHP je sve više dobrih mehanizama za rukovanje XML-om.
XML parseri se koriste za čitanje, ažuriranje, kreiranje i manipulisanje XML dokumentom. U programskom jeziku PHP postoje se 2 glavna tipa parsera:

- Parseri zasnovani na drvetu,

- Parseri zasnovani na događajima.

Parseri zasnovani na drvetu čuvaju ceo dokument u memoriji i transformišu XML dokument u strukturu drveta. Analiziraju ceo dokument i omogućavaju pristup elementima drveta (DOM). Ova vrsta parsera je bolja opcija za manje XML dokumente, ali ne i za veliki XML dokument jer izaziva velike probleme sa performansama.
Najznačajniji primeri parsera zasnovanog na drvetu su SimpleXML i DOM.
Parseri zasnovani na događajima ne drže ceo dokument u memoriji, umesto toga čitaju ih u jednom čvoru istovremeno i omogu cavaju interakciju sa njim u realnom vremenu. Kada se pređe na sledeći čvor, stari se baci. Ova vrsta parsera je pogodna za velike XML dokumente. Brže parsira i troši manje memorije.
Primeri parsera zasnovanih na događajima su XMLReader i XML Expat Parser.

10.2 SimpleXML parser

SimpleXML je parser zasnovan na drvetu. SimpleXML nudi jednostavan način dobijanja imena elementa, atributa i tekstualnog sadržaja ako znate strukturu ili izgled XML dokumenta.
SimpleXML pretvara XML dokument u strukturu podataka kroz koju se može iterirati kao kroz kolekciju nizova i objekata.

SimpleXML - Iščitavanje iz stringa

Funkcija *simple_load_string* se koristi za iščitavanje XML podatka iz stringa. Neka se XML dokument čuva u stringu, kao u primeru 78.

Primer 78: XML dokument u stringu

```
$XMLdokument =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Marko Markovic</to>
<from>Janko Jankovic</from>
<heading>Podsetnik</heading>
<body>Nemoj zaboraviti da me zoves za vikend</body>
</note>"
```

Primer 79 pokazuje kako uz pomoć funkcije *simple_load_string* mogu ti podaci da se iščitaju. Funkcija kao argument prima string u kome je sačuvan XML dokument i vraća SimpleXML objekat.

Primer 79: Iščitavanje podataka

```
<?php
$XMLdokument =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Marko Markovic</to>
<from>Janko Jankovic</from>
<heading>Podsetnik</heading>
```

```
<body>Nemoj zaboraviti da me zoves za vikend</body>
</note>";

$xml=simplexml_load_string($XMLdokument) or die("Greska");
print_r($xml);
?>
```

Rezultat:

```
SimpleXMLElement Object ( [to] => Marko Markovic [from] => Janko Jankovic [heading] =>
  ↪ Podsetnik [body] => Nemoj zaboraviti da me zoves za vikend )
```

Ukoliko je dokument sačuvan u datoteci moguće ga je na sličan način iščitati. To se postiže uz pomoć funkcije *simple_load_file*, čiji je jedini argument naziv datoteke u kojoj je XML dokument sačuvan, i vraća SimpleXML objekat.
Neka je XML dokument sačuvan u datoteci Note.xml. U primeru 80 prikazano je iščitavanje dokumenta iz datoteke.

Primer 80: Iščitavanje iz datoteke Note.xml

```
<?php
$xml=simplexml_load_file(Note.xml) or die("Greska");
print_r($xml);
?>
```

Rukovanje greškama: Korisno je koristiti funkcionalnost **libxml** da bi se preuzele sve XML greške prilikom učitavanja dokumenta i potom isčitale sve te greške. U primeru 82 prikazan je pokušaj da se učita nepravilan XML string.

Primer 81: Nepravilan XML dokument

```
<?php
libxml_use_internal_errors(true);
$XMLdokument =
"<?xml version='1.0' encoding='UTF-8'?>
<document>
<user>Marko Marković</wronguser>
<email>marko@primer.com</wrongemail>
</document>";

$xml = simplexml_load_string($XMLdokument);
if ($xml === false) {
echo "Greška pri učitavanju XML: ";
foreach(libxml_get_errors() as $error) {
    echo "<br>", $error->message;
    }
}
else {
print_r($xml);
}
?>
```

```
Greška pri učitavanju XML:
```

```
    Opening and ending tag mismatch: user line 3 and wronguser
    Opening and ending tag mismatch: email line 4 and wrongemail
```

U nizu `libxml_get_errors` se nalaze sve greške nastale pri učitavanju XML dokumenta. Greške su nastale zbog pogrešnih zatvarajućih etiketa </wronguser>i </wrongemail>. Moguće je pristupiti vrednostima čvorova iz datoteke Note.xml, kao u primeru 82.

Primer 82: Pristup čvorovima datoteke note.xml

```
<?php
$xml=simplexml_load_file("note.xml") or die("Greška");
echo $xml->to . "<br>";
echo $xml->from . "<br>";
echo $xml->heading . "<br>";
echo $xml->body;
```

Rezultat primera:

```
Marko Markovic
Janko Jankovic
Podsetnik
Nemoj zaboraviti da me zoves za vikend
```

Neka je dat drugi xml dokument Kursevi.xml, prikazan u primeru 83.

Primer 83: Kursevi.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<kursevi>
  <kurs kategorija = "JAVA">
    <naslov>Java</naslov>
    <predavač>Marko Marković</predavač>
    <trajanje>10</trajanje>
    <cena>30$</cena>
  </kurs>

  <kurs kategorija = "PHP">
    <naslov>php</naslov>.
    <predavač>Janko Janković</predavač>
    <trajanje>8</trajanje>
    <cena>50$</cena>
  </kurs>

  <kurs kategorija = "HTML">
    <naslov>html</naslov>
    <predavač>Nenad Nenadović</predavač>
    <trajanje>5</trajanje>
    <cena>20$</cena>
  </kurs>

  <kurs kategorija = "WEB">
    <naslov>Web Technologije</naslov>
    <predavač>Jovan Jovanović</predavač>
    <trajanje>10</trajanje>
    <cena>60$</cena>
```

```
29    </kurs>
30
31 </kursevi>
```

Moguće je pristupiti vrednostima podelemenata <naslov> svakog elementa <kurs>. Primer 84 pokazuje na koji način se može pristupiti vrednosti elementa <naslov> u prvom i drugom <kurs> elementu u dateteci kursevi.xml.

Primer 84: Pristup pod elementima

```
1  <?php
2  $xml=simplexml_load_file("kursevi.xml") or die("Greška");
3  echo $xml->kurs[0]->naslov . "<br>";
4  echo $xml->kurs[1]->naslov;
5  ?>
```

Rezultat:

```
1  Java
2  php
```

Ako se u promenljivoj `$xml` čuva SimpleXML objekat koji iščitava datoteku kursevi.xml. Niz koji se dobija komandom `$xml->children()` sadrži sve elemente u datoteci kursevi.xml. Primer 85 pokazuje kako se pomoću petlje prolazi kroz sve <kurs> elemente u datoteci kursevi.xml i dobijaju vrednosti elemenata <naslov>, <predavač>, <trajanje>, <cena>.

Primer 85: Svi elementi u datoteci kursevi.xml

```
1  <?php
2  $xml=simplexml_load_file("kursevi.xml") or die("Greška");
3  foreach($xml->children() as $kurs) {
4  echo $kurs->naslov . ", ";
5  echo $kurs->predavač . ", ";
6  echo $kurs->trajanje . ", ";
7  echo $kurs->cena . "<br>";
8  }
9  ?>
```

Rezultat:

```
1  Java, Marko Markovic, 10, 30$
2  php, Janko Janković, 8, 50$
3  html, Nenad Nenadović, 5, 20$
4  Web Technologije, Jovan Jovanović, 10, 60$
```

Primer 86 pokazuje kako pomoću petlje mogu dobiti sve vrednosti atributa kategorija svih <kurs>elemenata.

Primer 86: Sve vrednosti atributa kategorija

```
<?php
$xml=simplexml_load_file("kursevi.xml") or die("Greška");
foreach($xml->children() as $kurs) {
echo $kurs['kategorija'];
echo "<br>";
}
?>
```

Rezultat:

```
JAVA
PHP
HTML
WEB
```

10.3 Expat parser

Expat parser je XML parser zasnovan na događajima. Neka je dat XML element `<from>Janko Janković</from>`. Expat parser kao parser zasnovan na događajima ovaj element gleda kao niz 3 događaja: početni element <from>, vrednost Janko Janković, krajnji element </from>.
Neka je dat XML dokument Note.xml, prikazan u primeru 87.

Primer 87: Note.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Marko Marković</to>
<from>Janko Janković</from>
<heading>Podsetnik</heading>
<body>Nemoj zaboraviti da me zoveš za vikend</body>
</note>
```

Prvo bi trebalo inicijalizovati XML parser, zatim definisati rukovaoce (engl. *handlers*) za različite XML događaje i nakon taga je moguće parsirati XML dokument. XML parser se inicijalizuje funkcijom *xml_praser_create*, dok se rukovaoci definišu pomoću funkcija *xml_set_element_handler* i *xml_set_character_data_handler*.

Primer 88: Inicijalizovanje parsera i definisanje ruovalaca za događaje

```
<?php
//Inicijalizovanje XML parsera
$parser = xml_parser_create();
//funkcija koja se koristi na početku elementa
function start($parser,$element_name,$element_attrs){
  switch ($element_name) {
    case 'NOTE':
      echo "-- Note -- <br>";
      break;
    case 'TO':
      echo "To: ";
```

```
12          break;
13        case 'FROM':
14          echo "From: ";
15          break;
16        case 'HEADING':
17          echo "Heading: ";
18          break;
19        case 'BODY':
20          echo "Message: ";
21          break;
22      }
23    }
24    //funkcija koja se koristi na kraju elementa
25    function stop($parser,$element_name){
26      echo "<br>";
27    }
28    function char($parser,$data){
29      echo $data;
30    }
31
32    xml_set_element_handler($parser,"start","stop");
33    xml_set_character_data_handler($parser,"char");
34    $fp = fopen("Note.xml","r");
35    while ($data=fread($fp, 4096)) {
36      xml_parse($parser, $data,feof($fp)) or
37    die (sprintf("XML Error: %s at line %d",
38    xml_error_string(xml_get_error_code($parser)),
39    xml_get_current_line_number($parser)));
40
41    }
42    xml_parser_free($parser);
43    ?>
```

U primeru 88, na početku je inicijalizocan XML parser korišćenjem funkcije *xml_parser_create*. Pomoću funkcije *xml_set_element_handler* određuje se koja će se funkcija izvršiti kada parser naiđe na otvarajuće i zatvarajuće etikete. Kreiranjem funkcije *xml_set_character_data_handler* određuje koja će se funkcija izvršiti kada parser naiđe na tekstualni podatak(*character data*).Parsira se dokument Note.xml, i ako se naiđe na grešku, funkcija *xml_error_string* će je obraditi. Na kraju se poziva funkcija *xml_parser_free* koja oslovađa memoriju koju je parser zauzimao.
Rezultat primera 88:

```
-- Note --
To: Marko Marković
From: Janko Janković
Heading: Podsetnik
Message: Nemoj zaboraviti da me zoveš za vikend
```

10.4 DOM parser

DOM parser je XML parser zasnovan na drvetu. Neka je dat XML dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<from>Janko Janković</from>
```

Kako je zasnovan na drvetu, DOM parser ga vidi kao drvoidnu strukturu:

- XML Dokument,
- koreni element <from>,
- tekstualni element: Janko Janković.

Naredbom `$xmlDoc = new DOMDocument` se inicijalizuje DOM parser, dok naredbama `$xmlDoc->load()` i `$xmlDoc->saveXML` isčitava xml sadržaj iz datoteke. U primeru 89 je DOM parser primenjen na dokument Note.xml.

Primer 89: DOM parser primenjen na Note.xml

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("Note.xml");
print $xmlDoc->saveXML();
?>
```

Rezultat:

```
Marko Marković Janko Janković Podsetnik Nemoj zaboraviti da me zoveš za vikend
```

Ako se u Veb pregledaču izabere opcija „View source"prikazaće se HTML kôd indentičan kodu iz XML dokumenta.

```
<note>
<to>Marko Markovic</to>
<from>Janko Jankovic</from>
<heading>Podsetnik</heading>
Nemoj zaboraviti da me zoves za vikend
</note>
```

U primeru, kreira se DOMDocument objekat i učitava XML dokument. Funkcija *saveXML* pretvara XML kôd u string koji se zatim ispisuje.
U primeru 90 kreira se XML parser, učitava XML dokument i pomoću petlje prolazi se kroz sve elemente <note> elementa (Pogledati primer, svi elementi su unutar <note>elementa)

Primer 90: Prolazak kroz sve elemente xml dokumenta

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");
$x = $xmlDoc->documentElement; // <note> element
```

```
5   foreach ($x->childNodes AS $item) {
6       print $item->nodeName . " = " . $item->nodeValue . "<br>";
7   }
8   ?>
```

U primeru 90, promenljivoj `$x` se dodeljuje element <note>. `childNodes` je atrubut u kome se čuvaju svi njegovi potomci u drvoidnoj strukturi, i svaki od njih ima atribute `nodeName` i `nodeValue`.
Rezultat primera dat je na slici 21.

Slika 21: Svi elementi datoteke note.xml

Može se primetiti da između svaka dva elementa postoji prazan tekstualni čvor. Kada se XML generiše često postoji razmak između 2 čvora. DOM parser ih tretira kao zaseban element XML dokumenta, i ako programer nije svestan da oni postoje, mogu prouzrokovati grešku.
Više o radu sa XML dokumentima može se pročitati na [3], [8], [7]

Zaključak

Elektronske lekcije o nekim naprednim mogućnostima u programskom jeziku PHP predstavljaju obradu i sistematizaciju naprednih koncepata programskog jezika PHP.

Sadržaj elektronskih lekcija pokriva neke napredne mogućnosti programskog jezika PHP. Lekcije su pisane metodički sa ciljem da budu prilagođene korisnicima koji poseduju elementarno znanje iz programskog jezika PHP, i da predstavljaju osnovu za njihov dalji razvoj. Svaka elektronska lekcija sadrži odgovarajuće primere koji bi trebalo da korisniku lekcija pomognu pri učenju.

PHP je impresivan programski jezik, prilagođen za kreiranje veb aplikacija i dinamičku izmenu HTML sadržaja. Iako je PHP specijalizovan programski jezik, koncepti koji se mogu naučiti njegovom upotrebom primenljivi su u velikom broj drugih programskih okruženja.

Literatura za učenje programskog jezika PHP je obimna i dostupna na internetu, ali je učenje sporije, jer zahteva dobro razumevanje strane literature. Zbog toga lekcije imaju pogodnost što su dostupne na srpskom jeziku. Sve lekcije o nekim naprednim mogućnostima u programskom jeziku PHP su dostupne na platformi *eŠkola veba* i namenjene su svima koji žele da unaprede svoje znanje u oblasti veb tehnologija i imaju cilj da osposobe čitaoca za kreiranje veb stranica i jednostavnih aplikacija. Direktno pokretanje koda iz svake lekcije i zadaci na kraju svake lekcije omogućavaju korisnicima da provere naučeno, pri čemu je za svaki zadatak priložen i pomoćni kôd. Sve elektronske lekcije su dostupne na linku `http://edusoft.matf.bg.ac.rs/eskola_veba/#/course-details/php`.

Literatura

[1] Zvanični sajt programskog jezika PHP http://php.net/

[2] Jurić Nemanja, Marić Miroslav, *eŠkola Veba, VII simpozijum Matematika i primene*, Matematički fakutet, Beograd, 5. novembar 2016.

[3] Vladimir Filipović, *Predavanja iz predmeta Uvod u veb i internet tehnologije* http://poincare.math.rs/~vladaf/Courses/Matf UVIT/Predavanja/

[4] Luke Velling, Laura Thompson, PHP and MySQL Web Development (5th Edition)

[5] *Uvod u PHP i mySql* - Tečajevi Srce, Univerzitet u Zagrebu https://www.srce.unizg.hr/files/srce/docs/edu/osnovni-tecajevi/d350_polaznik.pdf

[6] Anđelka Zečević, *Predavanje iz predmeta Programiranje za Veb* http://poincare.matf.bg.ac.rs/~andjelkaz/pzv/veb1718.html

[7] Elektronski kurs programskog jezika PHP: http://www.link-elearning.com/kurs-PHP-programiranje_289_

[8] Sajt platforme za interaktivno učenje veb tehnologija https://www.w3schools.com/

[9] Sajt platforme za interaktivno učenje veb tehnologija https://www.tutorialspoint.com/