1. Podizanje HyperV

https://ubuntu.com/download/desktopVirtual switch manager – definirti switch

Prikazivanje IP adrese

**ip addr show**

ip a

**Promjena root PWD:**

sudo passwd root

**Promjena vlasništva na direktoriju**

sudo chown -R $USER:$USER /var/www/html/ostechnix1.lan/public\_html

**Install apache on Linux**

Sudo apt update

Sudo apt install apache2

**Konfiguracija Appache**

[**https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-22-04**](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-22-04)

$ sudo nano /etc/apache2/sites-enabled/000-default.conf

Restart Appache:

sudo systemctl restart apache2

**Mjesta gdje možeš pronaći .htaccess na Apache2:**

1. **Root direktorij web stranice:**
   * .htaccess datoteka se najčešće nalazi u root direktoriju vaše web stranice. Ovisno o konfiguraciji vašeg servera, to će biti jedan od sljedećih direktorija:
     + /var/www/html/ (Ako koristite osnovnu instalaciju)
     + /var/www/yourdomain.com/ (Ako imate nekoliko web stranica)
     + /home/username/public\_html/ (Ako koristite shared hosting)

Na primjer, ako vaša stranica koristi direktorij /var/www/html/, .htaccess bi se trebala nalaziti tamo.

1. **Konfiguracija MySql**
   * sudo apt install mysql-server
2. sudo mysql
3. ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'password';
4. Mysql exit;
   * sudo mysql\_secure\_installation (promjena pwd za root)
   * mysql -u root -p (pristup bazi s root korisnikom)
5. Instalacija PHP
   * sudo apt install php libapache2-mod-php php-mysql
   * php -v
   * PHP ini:
   * /etc/php/7.4/apache2/php.ini
6. **Composer:**
   * **Korak 1: Preuzmi Composer installer**

[**https://www.digitalocean.com/community/tutorials/how-to-install-and-use-composer-on-ubuntu-18-04**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-composer-on-ubuntu-18-04)

cd ~

php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

* + **Korak 2: Verificiraj hash**

cd ~

curl -sS [https://getcomposer.org/installer -o /tmp/composer-setup.php](https://getcomposer.org/installer%20-o%20/tmp/composer-setup.php)

HASH=`curl -sS [https://composer.github.io/installer.sig`](https://composer.github.io/installer.sig%60)

echo $HASH

php -r "if (hash\_file('SHA384', '/tmp/composer-setup.php') === '$HASH') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP\_EOL;"

* + **Instalacja composer-a**

sudo php /tmp/composer-setup.php --install-dir=/usr/local/bin --filename=composer

PRIMJER:

1. **Kreiranje mape za projekat**

Sada, kreiraj traženu mapu **ComposerExercise** u direktoriju /var/www/html/BackendDeveloper.

**Korak 1**:Promjeni vlasnika na BackendDeveloper

sudo chown -R $USER:$USER /var/www/html/BackendDeveloper

**Korak 2: Kreiraj mapu**

sudo mkdir -p /var/www/html/BackendDeveloper/ComposerExercise

**Korak 2: Prebaci se u novokreiranu mapu**

cd /var/www/html/BackendDeveloper/ComposerExercise

**3. Instaliraj paket sa Packagist-a**

Za primjer, instalirat ćemo paket guzzlehttp/guzzle (popularni HTTP client paket).

**Korak 1: Pokreni Composer inicijalizaciju u mapi**

composer init

Ovdje ćeš unijeti osnovne informacije o projektu (ime, opis itd.).

**Korak 2: Instaliraj željeni paket**

composer require guzzlehttp/guzzle

**4. Provjeri instalaciju**

Nakon instalacije paketa, sve zavisne datoteke bit će preuzete u mapu vendor/, a composer.json i composer.lock datoteke će biti kreirane. Provjeri sadržaj mape:

**ls -l**

* strukturu projekta s composer.json, composer.lock i vendor/ mapa.

1. **Git:**

* kreirati direktorij
* pokrenuti git init – definira se glavna grana
* preimenovati master u main – **git branch -M main**
* dodati neku datoteku u master granu
* pokrenuti git add .
* pokrenuti git commit -m „App init“
* git branch – pokazuje grane

Na main branchu se drži kod koji je u produkciji, na njemu se ne radi

Git branch naredba pokazuje u kojem smo branchu – pokazano sa **\***

Prebacivanje na drugu granu

Git checkout „ime grane“

Stvaranje nove grane i automatski prebacivanje

Git checkout -b „naziv grane“

Nazivi za grane: <https://gist.github.com/qoomon/5dfcdf8eec66a051ecd85625518cfd13>

**Brisanje brancheva**

* Pozicioniranje u branch koji nećemo brisati
* Moramo se pozicionirati u main - **git checkout main**
* Brisanje - git branch -d „ime brancha“
* Ako ima neprenesenih promjena force delete **git branch -D „ime brancha“**

**Spajanje promjena n jednom branchu u drugi**

* Pozicionirani u branch u koji ćemo mergati
* **Git merge featureA**

**-prebaci i commit**

**Ako radimo na istoj datoteci u dvije različite grane, a želimo promjene comitati u treću**

1. **Pokrenemo git add .**
2. **Pokrenemo git commit -m „opis priomjene“ na svakoj grani**
3. **Prebacimo se u granu u kojoj želimo spojiti promjene:**
4. **Pokrenemo git merge „naziv grane 1“**
5. **Pokrenemo git merge „naziv grane 2“**

**U VS studio odaberemo što želimo napraviti s promjenama i nakon toga pokrenemo**

**Git add . i git commit – m „Fix ….“**

**Git log - popis svih commiteva**

**git checkout <commit-hash> - vraćanje na neki od commiteva**

**Udaljeni repozitorij:**

Github /dashboard

Prijenos https protokol – uvijek pita za lozinku kad želimo nešto prenijeti na repozitoriji, traži svaki puta prijavu

Ssh protokol – uspostavlja sigurni vezu i mehanizam s kojim osiguravamo da se naše računalo predstavi git hubu bez unošenja lozinke

**Ssh ključevi**

* Prenese javni ključ u git hub servis – predstavljamo se s tim ključem

[**https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent**](https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent)

**Pokrenemo:**

**ssh-keygen -t ed25519 -C "your\_email@example.com"**

**odaberemo naziv ključa**

**generira dvije datoteke : javnu i privatnu (javni i privatni ključ)ž**

**javni ključ se daje servisima i na taj način komuniciramo, privatni ne ide nigdje**

**kopiramo public ključ u notepadu i odemo na git hub**

**Slika na kojoj se prikazuje tekst, softver, snimka zaslona, broj

Opis je automatski generiran**

**Kreirati udaljeni repozitorij**

Gumb novi repozitorij and

Prebaciti s https u ssh

git remote add origin git@github.com:vesnav111/git-branches.git

git remote -v – provjera statusa

git push -u origin „naziv grane“

git branch -a - pokazuje sve grane i online

Povezivanje s dev grane na main granu online

Zadnji korak odabrati „Megre pull request“ - odabrati „rebase and merge“ – da se branchevi svedu na istu razinu

Pull request – smisao ima jedino kad se radi u timu, šalje se na odobrenje, team lead prihvaća je li kod u redu

**Semantički sustav verzioniziranja:**

Semantički sustav verzioniranja (eng. Semantic Versioning, često skraćeno kao SemVer) je standard za označavanje verzija softvera koji pomaže programerima i korisnicima razumjeti promjene u softveru na temelju brojeva verzija. Koristi format MAJOR.MINOR.PATCH (npr. 1.4.2), gdje svaki broj ima jasno definirano značenje:

Komponente verzije:

MAJOR (glavni broj):

Mijenja se kada se uvedu promjene koje nisu kompatibilne s prethodnim verzijama (tzv. "breaking changes").

Npr. prelazak s 1.x.x na 2.0.0 označava da su određene funkcionalnosti promijenjene ili uklonjene te možda neće raditi s kodom koji koristi stariju verziju.

MINOR (manji broj):

Mijenja se kada se dodaju nove funkcionalnosti koje su kompatibilne s postojećom verzijom.

Npr. prelazak s 1.3.0 na 1.4.0 označava dodavanje novih značajki bez utjecaja na postojeći kod.

PATCH (zakrpa):

Mijenja se kada se isprave greške ili uvedu manje izmjene koje ne mijenjaju funkcionalnost (kompatibilne su).

Npr. prelazak s 1.3.2 na 1.3.3 označava ispravak buga bez utjecaja na ponašanje sustava.

Pravila semantičkog verzioniranja:

Verzija započinje s 0.1.0 tijekom razvoja (prije stabilnog izdanja). Verzija 0.x.y se smatra nestabilnom i može se često mijenjati.

Kada softver postane stabilan, prelazi na 1.0.0.

Brojevi se povećavaju na temelju vrsta promjena:

"Breaking changes" -> povećava MAJOR.

Dodavanje kompatibilnih značajki -> povećava MINOR.

Ispravci grešaka -> povećava PATCH.

**KLONIRANJE REPOZITORIJA**

* **Kreira se repozitorij**
* **Nakon toga odabere se suradnik**
* **Suradnik dobije poveznicu**
* **Kopira kod – lokalno računalo + ssh**
* **Lokalno se s git bash pozicioniramo gdje želimo i napravimo clone**
* **Git clone + kopiramo kod**
* **Kreiramo svoju granu**
* **Kad smo gotovi napravimo**
* **Git push -u origin „naziv grane“**

**Dolazi na odobrenje team leadu**

**### Knjiga ###**

**https://tkrajina.github.io/uvod-u-git/git.pdf**

**### Dokumentacija ###**

**https://git-scm.com/docs**

**### Kako generairati SSH ključ ###**

[**https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent**](https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent)

**Osnove PHP**

[**https://www.php.net/**](https://www.php.net/)

Php direktive

Bitno je gdje se napravi include, na kojem mjestu u datoteci

**Ako include ne pronađe datoteku dobije se samo upozorenje**

**Include** - Naredba uključi uključuje i procjenjuje navedenu datoteku.

**Include once –** Naredba include\_once uključuje i procjenjuje navedenu datoteku tijekom izvođenja skripte. Ovo je ponašanje slično naredbi uključivanja, s jedinom razlikom što ako je kod iz datoteke već uključen, neće biti ponovo uključen. Kao što naziv sugerira, bit će uključen samo jednom.

include\_once se može koristiti u slučajevima kada ista datoteka može biti uključena i procijenjena više puta tijekom određenog izvođenja skripte, tako da u ovom slučaju može pomoći u izbjegavanju problema kao što su redefinicije funkcija, preraspodjele vrijednosti varijable itd.

uključi\_jednomrequire

**reqiuire** - require je identičan uključivanju osim što će u slučaju neuspjeha proizvesti fatalnu pogrešku razine E\_COMPILE\_ERROR. Drugim riječima, zaustavit će skriptu, dok uključi samo emitira upozorenje (E\_WARNING) koje omogućuje nastavak skripte.

**require\_once** - Naredba require\_once je identična kao require osim što će PHP provjeriti je li datoteka već uključena, i ako je, neće je ponovno uključiti (zahtijevati).

**error\_reporting (0)** – ugasi greške u php i worning, ma developmentu mora biti upaljeno

Varijable

**Varijabla** – u nju se sprema podatak kojeg kasnije koristimo i ona je varijabilna, vrijednosti u varijabli se mijenjanju

**Konstanta** - konstanta sadrži vrijednost isto kao ivarijabla, ali za razliku od varijable,kojoj se vrijednost može mijenjati tijekom izvođenja programa, vrijednost konstante se ne mijenja



**String varijanle** definiraju se pomoću dvostrukih ili jednostrukih navodnika

Prednost dvostrukih navodnika je da unutar vrijednosti može koristiti specijalne znakove (npr. \za novi red) i za konaktenaciju ne mora koristiti točku nego samo navede ime varijable

**Reference** – služe kako bi jednu varijablu referencirali na drugu. Promjenom varijable na koju je referencirano mijenja se i vrijednost referencirane varijable. Kod referenciranja stavlja se **&**ispred reference

* Varijable nije potrebno unaprijed deklarirati

Varijabla bez vrijednosti – deklaracija, kad joj se pridruži vrijednost – inicijalizacija

**Echo** je funkcija koja može ispisati jedan ili više stringova i ne vraća ništa

Naredba **print** ispisuje samo jedan string i vraća nešto

Ako pomoću funkcije Echo želimo ispisati nešto što nije string npr. boolean događa se **Type juggling** i ne ispisuje se true, nego je taj true pretvoren u **1, false ne ispisuje (u svijetu strinfa je to prazan string) -implicitna konverzija**

**Type casting –** mi forsiramo promjenu vrste podatka u varijabli **echo (int) $varijabla – explicitna konverzija**

Kad se zapisuje u datoteku novi red **\n**

U browseru **<br>** (tag)

**\** - escape znak

**Konkatenacija** = $ime . „ „ .$prezime

**Interpolacija** = „$ime $ prezime“ (ne radi unutar jednostrukih navodnih znakova)

Null – nema vijednost, ispravnije je odmah inicijalizirati vrijednost (ne samo $var)

isSet – vraža je li varijabla postvljena

**Operetori**

$++ (postfix operator)

$b=20

$c=$b++ ($c=20, $b=21)

++$ (prefix operator)

$b=20

$c=++$b ($c=21, $b=21) - ++ ima veći prioritet od = (sučelila su se dva operatora i gleda se prioritet koji je jači)

**Refernca**

Kad jednoj varijabli dodijelimo vrijednost druge varijable **(&)** – ispred varijable na desnoj strani, svaki puta kada se u jednoj varijbli promijeni vrijednost promijeni se vrijednost i u drugoj varijabli

**(vrijedi i u suprotnom smjeru)**

**Smisao reference je da se mijenjaju vrijednosti globalnih varijabli unutar funkcija**

**Stringovi**

Problem s encodingom, funkcije koje počinju sa str rade jedino sa ASCII znkovima (ne rade dobro s našim znakovima npr.

$ime=“SAŠA“

**strtoupper** ($ime) vrati SAšA (svali znak je jedan byte, a znak Š ima 2 znaka), moramo koristiti funkciju multibyte **mb\_strtoupper,** ima parametar encoding (utf-8)

**crosssidescripting** - sigurnosni problem – podvali se skripta, iz PHP se pošalje u JS cod, browser izvrši taj kod kad dobije tagove <script></script>

kad se gleda input od strane korisnika nikada se ne upisuju drrektno, postoje funkcije koje čiste taj string, **htmlspecialchar – mjenja problematične znakove u specijalne znakove,** skripta se ne izvršava nego ispisuje

**echo htmlspecialchars("<script>alert(\" DGSDGSDGSDG \")</script>");**

**NIZOVI (Array)**

**Liste –** poredani skup elementa

**Mapa –** sastpji se od ključa i vrijednosti, nebitan je redoslijed

Niz se inicijalizira sa **[]** ili **Array** funkcijom

**Višedimenzionalni nizovi** – moguće – ugniježđeno

Kad u jednostavni niz želimo **umetnuti** element **na kraj** pišemo:

$imeniza []='Vrijednost', treba umetati preporuka uvijek na kraj

Mapa

$imeključa['naziv svojstva'] = „vrijednost“

Funkcije za provjeru Niza:

Empty – provjera je li niz prazan

Isset – provjerava se je li neki ključ uopće postavljen

**Unset** – ukloniti elemente iz niza

Unset je ok kod mapa, ako uklanjamo element iz lista kad maknemo vrijednost, unset reinedksira listu

Traženje elemenata – ugrađene funkcije (in\_array, array\_serch)

Array\_search – vraća indeks pod kojim eje zapisana ta vrijednost, ako ne nađe javlja false

In\_array – vraća true ili false

Sortiranje Array-a – ugrađene funkcije – **mergesort** – postoje algoritmi

Potrebno je optimizirati sortiranje

Prazan Array =false;

**Dodavanje elementa:**

**Array\_push** – dodaje element na kraj arrey-a

**Micanje prvog elementa**

**Unset ($fruits[0]); -** indexsi se zadrže, samo se makne element

**Array\_shift –** makne prvi element i preindexsira

**Array\_merge** – spaja dva niza ili

$spojeni niz=[…$prvi, …$drugi] – (… - operator dekompozicije – rastavi niz na članove

**array\_filter(**[**array**](https://www.php.net/manual/en/language.types.array.php)**$array,**[**?**](https://www.php.net/manual/en/language.types.null.php)[**callable**](https://www.php.net/manual/en/language.types.callable.php)**$callback =**[**null**](https://www.php.net/manual/en/reserved.constants.php#constant.null)**,**[**int**](https://www.php.net/manual/en/language.types.integer.php)**$mode = 0): [array](https://www.php.net/manual/en/language.types.array.php) – iede element po element i šalje callback funkciji parametar, odgovor iz Callback funkcijie mora biti true ili false (npr vrati samo brojeve koji su veći od 5) – piše u novi array, funkcija je anonimna i ne može se pozvati, živi samo kad se izvršava array\_filter, novi array ostaje indexsacija**

**PHP kontrolne strukture**

* **Usmjeravaju tijek izvršenja koda**

**Petlje i Uvjetovane**

**Što su petlje?**

Petlja izvršava slijed naredbi mnogo puta dok navedeni uvjet ne postane lažan. A [petlja](https://www.guru99.com/hr/c-loop-statement.html) sastoji se od dva dijela, tijela petlje i kontrolne naredbe. Kontrolna izjava je kombinacija nekih uvjeta koji usmjeravaju tijelo petlje da se izvrši sve dok navedeni uvjet ne postane lažan. Svrha petlje je ponavljanje istog koda više puta.

**Što je while petlja?**

To je ulazno kontrolirana petlja. U while petlji, uvjet se procjenjuje prije obrade tijela petlje. Ako je uvjet istinit, tada i samo tada se izvršava tijelo petlje.

Nakon što se tijelo petlje izvrši, kontrola se ponovno vraća na početak, a uvjet se provjerava. Ako je istinit, isti se proces izvršava sve dok uvjet ne postane lažan. Jednom kada uvjet postane lažan, kontrola izlazi iz petlje.

U while petlji, ako uvjet nije istinit, tada se tijelo petlje neće izvršiti, niti jednom.

**Što je Do-While petlja?**

Do-while petlja slična je while petlji osim što se uvjet uvijek izvršava nakon tijela petlje. Također se naziva petlja kontrolirana izlazom.

U do-while petlji, tijelo petlje uvijek se izvršava barem jednom. Nakon što je tijelo izvršeno, onda provjerava stanje. Ako je uvjet istinit, tada će ponovno izvršiti tijelo petlje. U suprotnom, kontrola se prenosi izvan petlje.

**Uvjetovane**

**If -** neovisni if

**If/ else –** else se izvrši ako if uvije nije zadovoljen

**If / else if –** međusobno su vezani, ne pokreću se sve dok prethodni uvijet nije zadovoljen

**Switch case –** obavezan break; nakon što je uvijet zadovoljen

4 vrste petlji

**While ili do while** se koristi – kad se ne zna broj ponavljanja – dok s ne ispuni uvijet

**For petlja** koristi se kad znamo broj ponavljanja i sastoji se od:

-iterator

-uvijet koji uključuje iterator

-promjena iteratora

**Foreach** – koristi se u radu s nizovima

* + Radi sa elementima koji imaju implementirano sučelje Countable
  + Iterator unutar petlje
  + Array ima svojstvo prebrojivosti
  + Ne trebaju se definirati iteratori ni uvjeti, prebrojiv element uvjetuje do kada petlja radi

**For petlja:**

**Break** – zaustavlja petlju

**Continue** - preskače korak prekida trenutnu iteraciju i nastavlja izvođenje petlje

**Return –** ponaša se kao bake, vraća rezultat koji smo dobili izvršvanjem neke funkcije

# FUNKCIJE

Overload – svojstvo funkcije da imamo funkciju s različitim brojem parametra kod proceduralnog programiranja

Camel case naziv funkcije – prva riječ počinje malim slovom ostale sve velikim

Funkcija može ili ne mora vratiti vrijednost, ako ne vraća vrijednost to se naglašava sa **:void** kod definiranja funkcije

Plimorfizam – jedna metoda u različitim objektima rdi različite stvari

Stroga tipizacija – određuje koji tip podatka funkcija mora vratiti, direktiva koja uklučuje strogu tipizaciju

***Declare(strict\_types=1); - koji tip podatka može ući u funkciju i što funkcija može vratiti***

***Kod definicije funkcije :string***

**Parametar je varijabla koja se nalazi u zagradi funkcije**

Argument je nekakav izraz koji šaljemo u parametar funkcije

Postoji mogućnost da se definiraju opcionalni parametri to znači da već ti parametri imaju već definiranu vrijednost

**Imenovani argumenti** – neovisno o redoslijedu

**Pozicijski argumenti** prvi argument na prvu poziciju, drugi argument na drugu …

Varijble – globalni i lokalni scope

Sve globalne varijable pohranjuje u jednu superglobalnu varijablu

-imenski prostori

$GLOBALS – bilo gdje u kodu može se doći do varijable putem $Globals i naziva varijable – roditeljski array i sve varijable koje se kreiraju idu u taj array – ako već moramo raditi sa superglobalnim varijablama

Izbjegava se to i definiraju se **lokalne varijable** i varijabla živi samo unutar te funkcije – o tome brinu garbbage collectori, ako se želi unutar funkcije doći do superglobalne varijable piše se

$GLOBALS['imevarijble']

Primjer Callble funkcije:

function **array\_f**(array $a, **callable** $callback): array{

    $newArray=[];

    foreach ($a as $value){

        if ($callback($value)){

            $newArray[]=$value;

        }

    }

return $newArray;

}

$a=array\_f([5,10,15], **function($value**){

    return $value >5;

});

print\_r ($a);

**Parametri:**

1. **array $a**: Ovo je ulazni niz (array) koji funkcija obrađuje.
2. **callable $callback**: Funkcija koja se prosljeđuje kao argument (callback funkcija). Ova funkcija definira uvjet koji svaki element niza treba ispuniti kako bi bio uključen u rezultat.

**Kako radi:**

1. **$newArray = []**: Stvara se novi, prazan niz u koji će se dodavati elementi koji zadovoljavaju uvjet.
2. **foreach ($a as $value)**: Petlja prolazi kroz svaki element ulaznog niza $a.
3. **if ($callback($value))**:
   * Poziva se funkcija proslijeđena kao *callback* s trenutnom vrijednošću $value.
   * Ako funkcija vrati true, trenutna vrijednost se dodaje u $newArray.
4. **return $newArray**: Na kraju, funkcija vraća novi niz koji sadrži samo elemente koji su prošli uvjet.

Callback funkcija je u stvari funkcija u funkciji, proširenje funkcionalnosti glavne funkcije

**Reference –** parametar u funkciji je referenca &$imeparanetra na neku drugu varijablu, šalje se argument koji je refernca na neku varijablu u programu

**Rekurzija** – funkcija koja poziva samu sebe

U memoriji se markiraju koraci, kako bi se po tim koracima znali vratiti (faktorijeli), koristi se u matemtici, fibobonacijevi niz, sort algoritmi (merge sort)

* base case
* korak rekurzije
* uvijet kada rekurzije prestaje, rekurzija radi na stack memoriji, dok petlje rade direktno na cpu, samo odrđeni djelovi petlje rade na stack-u

stack memorija omogučava da markira stvari i onda se vraća

Nedostatak rekurzije – kad potroši memoriju u problemu smo

Pelja – obrađuje svi elementi iteracije, rekurzija vrti tako da na stek zapisuje sve dok ne dođe do baznog slučaja, kad dođe do baznog slučaja koristi markacije da izvuče rezultat

**Static varijble –** definira varijabliu kao memorijski prostor koji ostaje zapamćen između poziva funkcija, ne događa se ponovna inicijalizacija na nulu

Static ključna riječ ima mogućnost pamćenja između poziva funkcija

Funkcija kao varijabla – ime funkcije pridruži se varijabli i onda se varijabla poziva kao funkcija

**Generatori –** funkcija kojom možemo generirati različite stvari tko da ga zadržimo do iduće iteracije, generator omogućuje da funkciju pauziramo do iduće iteracije, funkcija im je uradu s datotekama, omogućuje da obrađujemo liniju po liniju, ada ne učitavamo cijelu datoteku.

Koristi yeald – znači vrati vrijednost, ali pauziraj funkciju do idućeg pokretanja, funkcija generator vraća objekt, sadrži pokazivače na kojem se property-u nalazimo

# RAD S DATOTAKAMA

Tip podataka - resource

**Bitna je putanja do datoteke**

* **Prvo kreiramo resorce – fopen vraća resorce, trebamo dobiti pointer na datoteku kako bi mogli čitati iz datoteke, omogućuje da čitamo znak po znak, fopen ima različite modove**

[**https://www.php.net/manual/en/function.fopen.php**](https://www.php.net/manual/en/function.fopen.php)

**filesize, zna kad file završava**

**fread – čita datoteku, ako je parametar fsize učita cijelu datoteku**

**bakon što smo završili s datotekom moramo završiti s datotekom i oslonbditi memoriju fclose**

**Ako ne može pročitati datoteku kod fopen dodati or die(„Ne mogu otvoriti datoteku $file name“)**

**Feof funkcija ispituje je li kraj dtoteke**

Fgets – daje red iz dtoteke i prebaccuje pointer na novi red

**File\_get\_contents()** – vrati sadržaj datoteke kao string za razliku od file(), može vratiti da nije pronašao datoteku false

**Json\_decode** - očekuje string, a može dobiti i false, drugi element je true (vrati asocijativni array)

Stroga tipizacija donekle popravi problem

Spremanje u json datoteku:

File\_put\_contents

Const BOOKS\_FILE = „books.json“ – definira konstantu isto kao i define('konstanta', 3.14)

* **Json – java script object notation, nastao 1999**

Ako u parametru upišemo **?string $title** - parametar je opcionalno string ili null, ali smo i dalje dužni poslati value i postavimo defult value na **null**

Zašto

**Prazan string** možemo tretirati kao da nema vrijednosti, ali prazan string je i dalje po tipu podatka string,

Foreach ($books as $book) – staje kad nađe na prvi element koji zadovoljava uvijet

Array\_map – ne staje kod prvog uvijeta nego ide dalje do kraja arrey-a