[Skip to content](https://github.com/camilledewavrin/Apriori/blob/master/Apriori.php" \l "start-of-content)

* Why GitHub?
* [Team](https://github.com/team)
* [Enterprise](https://github.com/enterprise)
* Explore
* [Marketplace](https://github.com/marketplace)
* Pricing

Top of Form



Bottom of Form

[Sign in](https://github.com/login?return_to=%2Fcamilledewavrin%2FApriori%2Fblob%2Fmaster%2FApriori.php)[Sign up](https://github.com/join?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F%3Cuser-name%3E%2F%3Crepo-name%3E%2Fblob%2Fshow&source=header-repo&source_repo=camilledewavrin%2FApriori)

[**camilledewavrin**](https://github.com/camilledewavrin)**/[Apriori](https://github.com/camilledewavrin/Apriori)**

* [Watch](https://github.com/login?return_to=%2Fcamilledewavrin%2FApriori)[**1**](https://github.com/camilledewavrin/Apriori/watchers)
* [Star](https://github.com/login?return_to=%2Fcamilledewavrin%2FApriori)[**1**](https://github.com/camilledewavrin/Apriori/stargazers)
* [Fork](https://github.com/login?return_to=%2Fcamilledewavrin%2FApriori)[**3**](https://github.com/camilledewavrin/Apriori/network/members)
* [**Code**](https://github.com/camilledewavrin/Apriori)
* [Issues](https://github.com/camilledewavrin/Apriori/issues)
* [Pull requests](https://github.com/camilledewavrin/Apriori/pulls)
* [Actions](https://github.com/camilledewavrin/Apriori/actions)
* [Projects](https://github.com/camilledewavrin/Apriori/projects)
* [Security](https://github.com/camilledewavrin/Apriori/security)
* [Insights](https://github.com/camilledewavrin/Apriori/pulse)

Dismiss

**Join GitHub today**

GitHub is home to over 50 million developers working together to host and review code, manage projects, and build software together.

[Sign up](https://github.com/join?source=prompt-blob-show&source_repo=camilledewavrin%2FApriori)

 master

[**Apriori**](https://github.com/camilledewavrin/Apriori)**/Apriori.php / Jump to**

[Go to file](https://github.com/camilledewavrin/Apriori/find/master)

[@camilledewavrin](https://github.com/camilledewavrin)

[camilledewavrin](https://github.com/camilledewavrin) [Added final version](https://github.com/camilledewavrin/Apriori/commit/2be50017039bd7da808ffafc06f4d2f9ed800ecf)

Latest commit [2be5001](https://github.com/camilledewavrin/Apriori/commit/2be50017039bd7da808ffafc06f4d2f9ed800ecf) on Apr 24, 2016[**History**](https://github.com/camilledewavrin/Apriori/commits/master/Apriori.php)

**1** contributor

644 lines (537 sloc)  18.2 KB

[Raw](https://github.com/camilledewavrin/Apriori/raw/master/Apriori.php)[Blame](https://github.com/camilledewavrin/Apriori/blame/master/Apriori.php)

|  |  |
| --- | --- |
|  | <pre> |
|  | <?php |
|  |  |
|  | class Apriori { |
|  | private $delimiter = ','; |
|  | private $minSup = 2; |
|  | private $minConf = 75; |
|  |  |
|  | private $rules = array(); |
|  | private $table = array(); |
|  | private $allthings = array(); |
|  | private $allsups = array(); |
|  | private $keys = array(); |
|  | private $freqItmsts = array(); |
|  | private $phase = 1; |
|  |  |
|  | //maxPhase>=2 |
|  | private $maxPhase = 20; |
|  |  |
|  | private $fiTime = 0; |
|  | private $arTime = 0; |
|  |  |
|  | public function setDelimiter($char) |
|  | { |
|  | $this->delimiter = $char; |
|  | } |
|  |  |
|  | public function setMinSup($int) |
|  | { |
|  | $this->minSup = $int; |
|  | } |
|  |  |
|  | public function setMinConf($int) |
|  | { |
|  | $this->minConf = $int; |
|  | } |
|  |  |
|  | public function setMaxScan($int) |
|  | { |
|  | $this->maxPhase = $int; |
|  | } |
|  |  |
|  | public function getDelimiter() |
|  | { |
|  | return $this->delimiter; |
|  | } |
|  |  |
|  | public function getMinSup() |
|  | { |
|  | return $this->minSup; |
|  | } |
|  |  |
|  | public function getMinConf() |
|  | { |
|  | return $this->minConf; |
|  | } |
|  |  |
|  | public function getMaxScan() |
|  | { |
|  | return $this->maxPhase; |
|  | } |
|  |  |
|  |  |
|  | // Recupere les datas de notre bdd et les inseres dans un tableau, qui sera return |
|  | private function getDatabaseData(){ |
|  | $bdd = new PDO('mysql:host=localhost;dbname=apriori;charset=utf8', 'root', ''); |
|  |  |
|  | //Il faut recuperer le nombre total de produit |
|  |  |
|  | $nbpaniersrep = $bdd->query('SELECT count(\*) FROM paniers'); |
|  | $nbpan = $nbpaniersrep->fetch()[0]; |
|  |  |
|  | // $reqSupport = $bdd->query('SELECT count(\*) FROM comporte WHERE id\_produit='.$i.';'); |
|  | for($i=1; $i<$nbpan+1;$i++) { |
|  | $req\_nom\_prod = $bdd->query('SELECT \* FROM comporte INNER JOIN produits WHERE id\_panier='.$i.' AND comporte.id\_produit=produits.id\_produit;'); |
|  | while ($donnees = $req\_nom\_prod->fetch()) { |
|  | $tabSupport[$i][] = $donnees['nom\_produit']; |
|  |  |
|  | } |
|  | } |
|  | $req\_nom\_prod->closeCursor(); // Termine le traitement de la requ�te |
|  |  |
|  | return $tabSupport; |
|  |  |
|  |  |
|  | } |
|  |  |
|  | //Insere les donnees du tableau passe en parametre dans un fichier fichier.txt, selon la forme attendu |
|  | private function setDataToFile($tabData) |
|  | { |
|  | $file = fopen( "fichier.txt", "w" ); |
|  |  |
|  | foreach($tabData as $key=>$value){ |
|  | foreach($value as $keyProduct=>$nameProduct){ |
|  |  |
|  | fwrite($file,$nameProduct.', '); |
|  | } |
|  | fwrite($file, "\n"); |
|  | } |
|  | fclose($file); |
|  | } |
|  |  |
|  | //Créer un tableau depuis le paramètre passé |
|  | private function makeTable($db) |
|  | { $table = array(); |
|  | $array = array(); |
|  | $counter = 1; |
|  |  |
|  | if(!is\_array($db)) |
|  | { |
|  | $db = file($db); |
|  | } |
|  |  |
|  | $num = count($db); |
|  |  |
|  | for($i=0; $i<$num; $i++) |
|  | { |
|  | $tmp = explode($this->delimiter, $db[$i]); |
|  |  |
|  |  |
|  | $num1 = count($tmp); |
|  | $x = array(); |
|  | for($j=0; $j<$num1; $j++) |
|  | { |
|  | //nom\_produit |
|  | $x = trim($tmp[$j]); |
|  |  |
|  | if($x==='') |
|  | { |
|  | continue; |
|  | } |
|  | //s'il existe un element |
|  | if(!isset($this->keys['v->k'][$x])) |
|  | { |
|  | //keys = tab de deux sous tableaux : |
|  | // [v->k] : [element] => id et |
|  | // [k->v]: [id]=> element |
|  | $this->keys['v->k'][$x] = $counter; |
|  | $this->keys['k->v'][$counter] = $x; |
|  | $counter++; |
|  | } |
|  | //Si le couple array[contenu de[v->k]] (=si array[cle element]) ne contient pas une valeur |
|  | if(!isset($array[$this->keys['v->k'][$x]])) |
|  | { |
|  | //On l'initialise a 1 |
|  | $array[$this->keys['v->k'][$x]] = 1; |
|  | $this->allsups[$this->keys['v->k'][$x]] = 1; |
|  | } |
|  | else |
|  | { |
|  | //Sinon on incremente sa valeur (le nombre d'occurence de l'element) |
|  | $array[$this->keys['v->k'][$x]]++; |
|  | //allsups contiendra egalement le nombre d'occurence de chaque [cle element] (= id\_element) |
|  | $this->allsups[$this->keys['v->k'][$x]]++; |
|  | } |
|  | //contient, pour chaque element de cahque panier (chaque ligne) la valeur 1 |
|  | $table[$i][$this->keys['v->k'][$x]] = 1; |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  | $tmp = array(); |
|  | //On ne garde que les id\_�l�ments de ceux dont le support est > minSupport |
|  | foreach($array as $item => $sup) |
|  | { |
|  |  |
|  | if($sup>=$this->minSup) |
|  | { |
|  | $tmp[] = array($item); |
|  | } |
|  |  |
|  | } |
|  | //Tableau contenant en value l'id\_element de ceux dont le support > minSupport |
|  | $this->allthings[$this->phase] = $tmp; |
|  | //contient, pour chaque element de cahque panier (chaque ligne) la valeur 1 |
|  | $this->table = $table; |
|  |  |
|  | } |
|  |  |
|  | //retourne les supports de TOUS les elements ET combinaisons d'elements de ^$arr |
|  | private function scan($arr, $implodeArr = '') |
|  | { |
|  | $cr = 0; |
|  | if($implodeArr) |
|  | { |
|  | if(isset($this->allsups[$implodeArr])) |
|  | { |
|  | return $this->allsups[$implodeArr]; |
|  | } |
|  | } |
|  | else |
|  | { |
|  | sort($arr); |
|  | $implodeArr = implode($this->delimiter, $arr); |
|  | if(isset($this->allsups[$implodeArr])) |
|  | { |
|  | return $this->allsups[$implodeArr]; |
|  | } |
|  | } |
|  | $num = count($this->table); |
|  | $num1 = count($arr); |
|  | for($i=0; $i<$num; $i++) |
|  | { |
|  | $bool = true; |
|  | for($j=0; $j<$num1; $j++) |
|  | { |
|  | if(!isset($this->table[$i][$arr[$j]])) |
|  | { |
|  | $bool = false; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | if($bool) |
|  | { |
|  | $cr++; |
|  | } |
|  | } |
|  |  |
|  | $this->allsups[$implodeArr] = $cr; |
|  | return $cr; |
|  |  |
|  | } |
|  |  |
|  | /// Ressort toutes les combinaisons possibles entre $arr et $arr2 (avec support >minSupport, car ca aura ete trie avant) |
|  | private function combine($arr1, $arr2) |
|  | { |
|  | $result = array(); |
|  |  |
|  | $num = count($arr1); |
|  | $num1 = count($arr2); |
|  | for($i=0; $i<$num; $i++) |
|  | { |
|  | if(!isset($result['k'][$arr1[$i]])) |
|  | { |
|  | $result['v'][] = $arr1[$i]; |
|  | $result['k'][$arr1[$i]] = 1; |
|  | } |
|  | } |
|  |  |
|  | for($i=0; $i<$num1; $i++) |
|  | { |
|  | if(!isset($result['k'][$arr2[$i]])) |
|  | { |
|  | // donne le tab [key]=>id\_element |
|  | $result['v'][] = $arr2[$i]; |
|  | //donne le tab [id\_elementt]=>1 |
|  | $result['k'][$arr2[$i]] = 1; |
|  | } |
|  | } |
|  | return $result['v']; |
|  | } |
|  |  |
|  |  |
|  | //Donne le nom veritable de chaque champ du tableau, plutot que son id |
|  | private function realName($arr) |
|  | { |
|  | $result = ''; |
|  |  |
|  | $num = count($arr); |
|  | for($j=0; $j<$num; $j++) |
|  | { |
|  | if($j) |
|  | { |
|  | $result .= $this->delimiter; |
|  | } |
|  |  |
|  | $result .= $this->keys['k->v'][$arr[$j]]; |
|  | } |
|  |  |
|  | return $result; |
|  | } |
|  |  |
|  |  |
|  | //Verifie simplement si la regle est valable, pour qu'on sache s'il fuat la tester/verifier ou pas : |
|  | //1-2=>2-3 : false |
|  | //1-2=>5-6 : true |
|  | private function checkRule($a, $b) |
|  | { |
|  | $a\_num = count($a); |
|  | $b\_num = count($b); |
|  | for($i=0; $i<$a\_num; $i++) |
|  | { |
|  | for($j=0; $j<$b\_num; $j++) |
|  | { |
|  | if($a[$i]==$b[$j]) |
|  | { |
|  | return false; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | return true; |
|  | } |
|  |  |
|  | //Calcule la confiance de l'element grace aux supports passe en parametre |
|  | //Applique la formule "Conf(X=>Y) = ||X,Y||/||X|| |
|  | private function confidence($sup\_a, $sup\_ab) |
|  | { |
|  | return round(($sup\_ab / $sup\_a) \* 100, 2); |
|  | } |
|  |  |
|  | //Retourne les sous ensemble de chaque sous ensemble |
|  | private function subsets($items) |
|  | { |
|  | $result = array(); |
|  | $num = count($items); |
|  | $members = pow(2, $num); |
|  | for($i=0; $i<$members; $i++) |
|  | { |
|  | $b = sprintf("%0".$num."b", $i); |
|  | $tmp = array(); |
|  | for($j=0; $j<$num; $j++) |
|  | { |
|  | if($b[$j]=='1') |
|  | { |
|  | $tmp[] = $items[$j]; |
|  | } |
|  | } |
|  |  |
|  | if($tmp) |
|  | { |
|  | sort($tmp); |
|  | $result[] = $tmp; |
|  | } |
|  | } |
|  | return $result; |
|  | } |
|  |  |
|  |  |
|  | //Si le numMax passe en parametre >= 3, c'est qu'on va �tre dans un ensemble >=3-elements |
|  | //On supprimera donc les support de 2-elements restant qui n'auront pas �t� unset avant |
|  | private function deleteDeprecatedItems($numMax){ |
|  | if ($numMax >= 3) { |
|  | foreach($this->freqItmsts as $k => $v){ |
|  | // On r�cup�re un tab de sous ensemble |
|  | $arr = explode($this->delimiter, $k); |
|  | //On compte le nombre d'�l�ment du sous ensemble |
|  | $num = count($arr); |
|  | if($num<3){ |
|  | // SI ce sous ensemble est compos� de moins de 3 �l�m�ents on l'unset |
|  | unset($this->freqItmsts[implode($this->delimiter, $arr)]); |
|  | } |
|  |  |
|  | } |
|  | } |
|  | } |
|  |  |
|  | //Unset les items qui ne sont pas conforme a ce qui est attendu (decrit en com dans la fonction) |
|  | private function freqItemsets($db) |
|  | { |
|  | $this->fiTime = $this->startTimer(); |
|  |  |
|  | $this->makeTable($db); |
|  | while(1) |
|  | { |
|  | if($this->phase>=$this->maxPhase) |
|  | { |
|  | break; |
|  | } |
|  | $num = count($this->allthings[$this->phase]); |
|  | $cr = 0; |
|  | for($i=0; $i<$num; $i++) |
|  | { |
|  | for($j=$i; $j<$num; $j++) |
|  | { |
|  | if($i==$j) |
|  | { |
|  | continue; |
|  | } |
|  | //Sors toutes les combinaisons des elements dont support > minSUpport (car allthings contient que |
|  | // ces elements) |
|  | $item = $this->combine($this->allthings[$this->phase][$i], $this->allthings[$this->phase][$j]); |
|  | sort($item); |
|  |  |
|  | $implodeArr = implode($this->delimiter, $item); |
|  | if(!isset($this->freqItmsts[$implodeArr])) |
|  | { |
|  | $sup = $this->scan($item, $implodeArr); |
|  | if($sup>=$this->minSup) |
|  | { |
|  | $this->allthings[$this->phase+1][] = $item; |
|  | $this->freqItmsts[$implodeArr] = 1; |
|  | $cr++; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | if($cr<=1) |
|  | { |
|  | break; |
|  | } |
|  | $this->phase++; |
|  | } |
|  |  |
|  | //Pour chaque freqItmsts, on supprime les "sous sous elements"(sauf celui representant le sous element lui-meme) |
|  | //de chaque sous elements, |
|  | // si le sous element est compose de 3 elements ou plus |
|  | foreach($this->freqItmsts as $k => $v) |
|  | { |
|  | //on cree $arr = un sous-ensemble(ex : L1, L2, L5 =3) -> [0]=>L1, [1]=>L2, [2]=>L3 |
|  | $arr = explode($this->delimiter, $k); |
|  | $maxNum=0; |
|  | //Nombre d'element dans le sous ensemble |
|  | $num = count($arr); |
|  | //Si le nombre de sous-elements > max ce nombre devient nouveau max |
|  | if($num > $maxNum){ |
|  | $maxNum = $num; |
|  | } |
|  |  |
|  | //Si l'ensemble traite est >= 3 elements (ex : {L1, L2, L5} =3) |
|  | if($num>=3) |
|  | { |
|  | //Affiche les sous-sousensemble possible du tableau passe en parametre |
|  | //ex : {L1, L2, L5} = {L1}, {L1,L2}... |
|  | $subsets = $this->subsets($arr); |
|  | $num1 = count($subsets); |
|  | //Pour chaque sous\_ensemble de l'ensemble $arr |
|  | for($i=0; $i<$num1; $i++) |
|  | { |
|  | // nombre d'elements dans le sosu-ensemble $i |
|  | //S'il est inferieur a num |
|  | if(count($subsets[$i])<$num) |
|  | { |
|  | //on enleve le sous element (ex pour L1, L2, L3 on ne gardera que ce |
|  | // sous sous ensemble, et non pas L1; L1,L2 ; L1,L3... |
|  | unset($this->freqItmsts[implode($this->delimiter, $subsets[$i])]); |
|  | // print\_r($this->freqItmsts); |
|  | // var\_dump('fin'); |
|  |  |
|  |  |
|  | } |
|  | else |
|  | { |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  | $this->deleteDeprecatedItems($maxNum); |
|  | $this->fiTime = $this->stopTimer($this->fiTime); |
|  | } |
|  |  |
|  | //Fonction principale |
|  | public function process($db) |
|  | { |
|  |  |
|  | $tabData=$this->getDatabaseData(); |
|  | $checked = array(); |
|  | $result = array(); |
|  | $this->setDataToFile($tabData); |
|  |  |
|  | $this->freqItemsets($db); |
|  |  |
|  | //Partie sur la confiance et les regles d'association, en partant sur le même principe |
|  | $this->arTime = $this->startTimer(); |
|  |  |
|  | foreach($this->freqItmsts as $k => $v) |
|  | { |
|  |  |
|  |  |
|  | $arr = explode($this->delimiter, $k); |
|  | $subsets = $this->subsets($arr); |
|  | $num = count($subsets); |
|  |  |
|  | for($i=0; $i<$num; $i++) |
|  | { |
|  | for($j=0; $j<$num; $j++) |
|  | { |
|  |  |
|  | if($this->checkRule($subsets[$i], $subsets[$j])) |
|  | { |
|  | $n1 = $this->realName($subsets[$i]); |
|  | $n2 = $this->realName($subsets[$j]); |
|  |  |
|  | $scan = $this->scan($this->combine($subsets[$i], $subsets[$j])); |
|  |  |
|  | $c1 = $this->confidence($this->scan($subsets[$i]), $scan); |
|  | $c2 = $this->confidence($this->scan($subsets[$j]), $scan); |
|  |  |
|  | if($c1>=$this->minConf) |
|  | { |
|  | $result[$n1][$n2] = $c1; |
|  | } |
|  |  |
|  | if($c2>=$this->minConf) |
|  | { |
|  | $result[$n2][$n1] = $c2; |
|  | } |
|  |  |
|  | $checked[$n1.$this->delimiter.$n2] = 1; |
|  | $checked[$n2.$this->delimiter.$n1] = 1; |
|  |  |
|  | } |
|  | } |
|  | } |
|  | } |
|  | $this->arTime = $this->stopTimer($this->arTime); |
|  |  |
|  | return $this->rules = $result; |
|  | } |
|  |  |
|  | //Affichage des Items fréquents et de leur support |
|  | public function printFreqItemsets() |
|  | { |
|  | echo 'Time: '.$this->fiTime.' second(s)<br />===============================================================================<br />'; |
|  |  |
|  | foreach($this->freqItmsts as $k => $v) |
|  | { |
|  | $tmp = ''; |
|  | $tmp1 = ''; |
|  | $k = explode($this->delimiter, $k); |
|  | $num = count($k); |
|  | for($i=0; $i<$num; $i++) |
|  | { |
|  | if($i) |
|  | { |
|  | $tmp .= $this->delimiter.$this->realName($k[$i]); |
|  | $tmp1 .= $this->delimiter.$k[$i]; |
|  | } |
|  | else |
|  | { |
|  | $tmp = $this->realName($k[$i]); |
|  | $tmp1 = $k[$i]; |
|  | } |
|  | } |
|  |  |
|  | echo '{'.$tmp.'} = '.$this->allsups[$tmp1].'<br />'; |
|  | } |
|  | } |
|  |  |
|  | //Sauvegarde dans le fichier passe en parametre |
|  | public function saveFreqItemsets($filename) |
|  | { |
|  | $content = ''; |
|  |  |
|  | foreach($this->freqItmsts as $k => $v) |
|  | { |
|  | $tmp = ''; |
|  | $tmp1 = ''; |
|  | $k = explode($this->delimiter, $k); |
|  | $num = count($k); |
|  | for($i=0; $i<$num; $i++) |
|  | { |
|  | if($i) |
|  | { |
|  | $tmp .= $this->delimiter.$this->realName($k[$i]); |
|  | $tmp1 .= $this->delimiter.$k[$i]; |
|  | } |
|  | else |
|  | { |
|  | $tmp = $this->realName($k[$i]); |
|  | $tmp1 = $k[$i]; |
|  | } |
|  | } |
|  |  |
|  | $content .= '{'.$tmp.'} = '.$this->allsups[$tmp1]."\n"; |
|  | } |
|  |  |
|  | file\_put\_contents($filename, $content); |
|  | } |
|  |  |
|  | //Affiche le support et les éléments de chaque sous ensemble final |
|  | public function getFreqItemsets() |
|  | { |
|  | $result = array(); |
|  |  |
|  | foreach($this->freqItmsts as $k => $v) |
|  | { |
|  | $tmp = array(); |
|  | $tmp['sup'] = $this->allsups[$k]; |
|  | $k = explode($this->delimiter, $k); |
|  | $num = count($k); |
|  | for($i=0; $i<$num; $i++) |
|  | { |
|  | $tmp[] = $this->realName($k[$i]); |
|  | } |
|  |  |
|  | $result[] = $tmp; |
|  | } |
|  |  |
|  | return $result; |
|  | } |
|  |  |
|  | //Affiche les regles d'associations et leurs confiances |
|  | public function printAssociationRules() |
|  | { |
|  | echo 'Time: '.$this->arTime.' second(s)<br />===============================================================================<br />'; |
|  |  |
|  | foreach($this->rules as $a => $arr) |
|  | { |
|  | foreach($arr as $b => $conf) |
|  | { |
|  | echo "$a => $b = $conf%<br />"; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | //Savegarde les règles d'associations dans un fichier passé en paramètre |
|  | public function saveAssociationRules($filename) |
|  | { |
|  | $content = ''; |
|  |  |
|  | foreach($this->rules as $a => $arr) |
|  | { |
|  | foreach($arr as $b => $conf) |
|  | { |
|  | $content .= "$a => $b = $conf%\n"; |
|  | } |
|  | } |
|  |  |
|  | file\_put\_contents($filename, $content); |
|  | } |
|  |  |
|  | //Affiches les règles d'association et leur confiance sous forme de tableau |
|  | public function getAssociationRules() |
|  | { |
|  | return $this->rules; |
|  | } |
|  |  |
|  | private function startTimer() |
|  | { |
|  | list($usec, $sec) = explode(" ", microtime()); |
|  | return ((float)$usec + (float)$sec); |
|  | } |
|  |  |
|  | private function stopTimer($start, $round=2) |
|  | { |
|  | $endtime = $this->startTimer()-$start; |
|  | $round = pow(10, $round); |
|  | return round($endtime\*$round)/$round; |
|  | } |
|  |  |
|  | } |
|  | $Apriori = new Apriori(); |
|  |  |
|  |  |
|  | ?> |
|  | </pre> |

* © 2020 GitHub, Inc.
* [Terms](https://github.com/site/terms)
* [Privacy](https://github.com/site/privacy)
* [Security](https://github.com/security)
* [Status](https://githubstatus.com/)
* [Help](https://docs.github.com/)
* [Contact GitHub](https://github.com/contact)
* [Pricing](https://github.com/pricing)
* [API](https://docs.github.com/)
* [Training](https://services.github.com/)
* [Blog](https://github.blog/)
* [About](https://github.com/about)