Automatisk annonsgranskning

Programvara som granskar användargenererat innehåll

Nässjö 2016-04-19

Handledare: Sven Terlegård

Författare: Hammar Gustav  
Sandberg Måns  
Swahn Pontus

Sammanfattning

I detta arbete undersöks möjligheten att låta ett datorsystem granska användargenererat innehåll före publicering på internet. Utgångspunkten för arbetet var att effektivisera annonsgranskning på annonssidor som till exempel Blocket, som idag enbart granskar sina annonser manuellt. Anledningen till det är att Blocket har valt bort automatiska granskningssystem för att de som tidigare har testats inte uppfyller deras höga krav. Målet med arbetet var att utveckla ett webbaserat system som automatiskt granskar text och bild som en användare lägger upp. Syftet var att göra granskningsprocessen mer effektiv.

Vi använde oss av tekniker från olika befintliga system för att utveckla en applikation i skriptspråket PHP. Vi började med att skapa en hemsida för att testa granskningssystemet Banbuilder, för att sedan ta ett stycke kod innehållande reguljära uttryck för att identifiera så kallat ”leetspeek” och lägga in i den egenutvecklade applikationen. Sedan skapades en ordlista innehållande fula ord och en ordlista som innehåller de flesta svenska ord, en svensk ordlista. Den färdiga applikationen använder sig av ordlistorna för att hitta olämplig text och visade sig fungera ganska bra förutom vissa brister som till exempel att den inte kan filtrera bort text innehållande felstavade fula ord. Det kan behövas en manuell granskare som komplement till applikationen.

Bildgranskningen började vi utveckla genom att testa Banbuilders inbyggda bildgranskning men den visade ha många brister, därför utvecklade vi vår egen bildgranskning men även den hade brister och visade sig vara alltför missvisande för att användas i praktiken. Att granska bilder automatiskt är komplext. Det är enkelt att lura systemet att acceptera olämpliga bilder genom att ändra färgskalan i bilden och den gav allt för många falsklarm.

Vi kom fram till att det till viss del är möjligt att filtrera bort olämplig text, men att det är svårt att filtrera bort olämpliga bilder.

Abstract

In this work, the outcome of letting a computer system review user-generated content before publishing on the Internet is examined. The starting point for this work was to streamline the advertisement review of advertising pages such as Blocket, which currently only reviews their ads manually. The reason for this is that the Blocket has opted out of automatic inspection systems because those that previously have been tested do not meet their high standards. The aim of this work was to develop a web-based system that automatically examines user-created text and images. The aim is to make the review process more efficient.

We used techniques from various existing systems to develop an application in the scripting language called PHP. We began by creating a website to test Banbuilder, a different review system, then we took a piece of code containing regular expressions which identify "leetspeek" and put it in the application. We created a dictionary containing foul language and a dictionary that contains most of the Swedish words, a Swedish dictionary. The finished application uses dictionaries to find inappropriate text and proved to work quite well apart from some flaws such as it can not filter out text containing misspelled ugly words. It may require a manual reviewer to supplement the application.

We started developing the picture reviewing part of the application by testing the embedded image review in Banbuilder but it had too many deficiencies. We developed our own image review code but it also had deficiencies and was found to be too inaccurate to be used in practice. It is easy to trick the system to accept inappropriate images by changing the color scheme of the images.

Our conclusion is that it is somewhat possible to filter inappropriate text, but it is much harder to filter inappropriate images.

Innehåll

­­

[1 Inledning 1](#_Toc450643927)

[1.1 Bakgrund/presentation 1](#_Toc450643928)

[1.2 Problemdiskussion 1](#_Toc450643929)

[1.3 Syfte 1](#_Toc450643930)

[1.4 Frågeställningar 1](#_Toc450643931)

[1.5 Avgränsningar 1](#_Toc450643932)

[2 Befintlig teori 2](#_Toc450643933)

[3 Metod/arbetssätt 3](#_Toc450643934)

[4 Genomförande 4](#_Toc450643935)

[4.1 Resultat 6](#_Toc450643936)

[4.1.1 Textfilter 6](#_Toc450643937)

[4.1.2 Bildgranskning 7](#_Toc450643938)

[4.2 Resultatdiskussion 8](#_Toc450643939)

[4.3 Metoddiskussion 8](#_Toc450643940)

[5 Slutsatser 9](#_Toc450643941)

[5.1 Egna reflektioner 9](#_Toc450643942)

[Litteraturförteckning 10](#_Toc450643943)

[6 Bilagor 11](#_Toc450643944)

# Inledning

I det här arbetet undersöker vi om det går att filtrera bort oönskat innehåll på annonssajter, forum och andra webbsidor, till exempel Blocket och Facebook, samt om det går att automatisera den filtreringen med hjälp av mjukvara.

## Bakgrund/presentation

Internet gör det möjligt för gemene man att publicera innehåll som når ut till hela världen. Det fria ordet går precis som i verkligheten att missbruka. På annonssajter som till exempel Blocket förekommer det ibland annonser som inte är relevanta och är stötande. För närvarande måste alla annonser granskas manuellt för att förhindra olämpligt innehåll. Vi tycker att det saknas ett fungerande filter som sorterar bort olämpligt innehåll automatiskt.

## Problemdiskussion

På Blocket uppstår problem när annonsören ska lägga in en ny annons. Varje annons måste granskas manuellt och det tar ofta lång tid för annonsen att godkännas. Det är även dyrt med granskningspersonal.

## Syfte

Syftet med det här arbetet är att utveckla en webbaserad applikation som förenklar och snabbar upp granskningen av användargenererat webbinnehåll i samband med både annonser och foruminlägg.

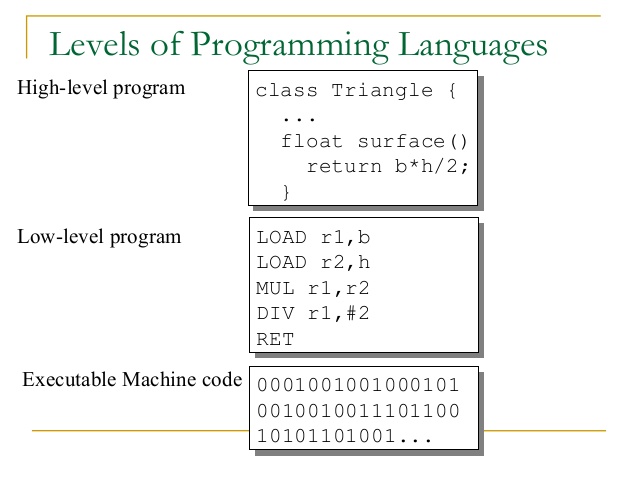
## Frågeställningar

Är det möjligt att automatiskt filtrera bort opassande innehåll med till exempel svordomar och porrbilder från forum och annonssajter?

## Avgränsningar

Vi avgränsar oss till att enbart filtrera olämplig text och olämpliga bilder. Med olämplig text menas text som innehåller fula ord eller innehåller ord som med avsikt ska likna fula ord men även text som inte innehåller fula ord och samtidigt är stötande. Med olämpliga bilder menas bilder med avsikt att vara stötande. Blocket har särskilda regler för vad som får ligga i vilken kategori och andra annonsregler men vi bortser från dem och inriktar oss mot endast olämpligt användargenererat innehåll.

# Befintlig teori

Blocket har tidigare undersökt möjligheterna med automatisk granskning av användargenererade annonser. (Kundservice, 2016) De problem som man har upplevt att systemen inte klarar av den höga nivån av komplexa regler som Blocket har på sina annonser. Det finns dessutom många undantag. Vidare har man konstaterat att automatisk granskning av bilder är svårt och att det förekommer bedrägerier. Antalet bedrägerier är få, men ett system som automatisk varnar för bedrägerier riskerar att flagga för fler falsklarm än faktiska bedrägerier. (Kundservice, 2016)

Figur 1 Förklarande bild som visar skillnaden mellan hög- och lågnivåspråk

PHP är ett skriptspråk och högnivåspråk (se Figur 1) som körs på serversidan på en webbserver (Wikipedia, 2016) och det är också där som all filtrering körs på ett kommersiellt system. Vi väljer det för att det är just det språket som används tillsammans med webbplatser som till exempel Blocket och Facebook. Språket har många fördelar i och med att det körs tillsammans med webbplatsen och det blir på så vis smidigare att utveckla i det. Tjänsten Webpurify erbjuder svordomsfilter, bild- och videomoderering. Den som vill använda tjänsten kan välja mellan PHP och ASP.NET, beroende på vilken plattform som webbadministratören har valt att bygga sin webbsajt på. Webpurify erbjuder filtrering på 15 olika språk. En svart respektive vit lista gör det möjligt för administratören att själv välja vilka ord som ska filtreras bort och vilka som ska behållas. (WebPurify, 2016)

# Metod/arbetssätt

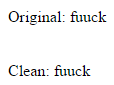
Genom att studera tidigare testade granskningssystem kommer vi att dra slutsatser om för- och nackdelar med de olika systemen. Det gör att vi kan på ett effektivt och noggrant sätt skapa ett system där vi har tagit de bästa bitarna och slagit ihop till ett lämpligt system. Vi ska använda oss av öppen källkod, det vill säga programkod som är fri att använda av vem som helst. Det finns trådar i forum som till exempel Stack Overflow där användare har diskuterat olika typer av filter. På dessa sidor ligger exempelkod som går att använda som grund i egna projekt. Vi kommer att utgå från PHP-kod, men vi har också kunskaper inom andra programmeringsspråk. Dessa kunskaper kan vi använda för att läsa kod skriven i till exempel ASP.NET och finna lösningar på problem som inte nödvändigtvis behöver vara skriva i PHP, men som vi sedan kan skriva om till PHP-kod.

Vi har valt att skapa en applikation därför att vi får bättre insikt i hur den kommer fungera praktiskt och för att vi kommer att upptäcka problem som kan uppstå vid automatisk granskning som man inte upptäcker genom att endast genomföra arbetet teoretiskt. Vi har valt att använda oss av öppen källkod så att vi kan modifiera den och slå ihop olika källkoder utan några problem.

I projektets slutskede kommer vi att utvärdera vår applikation för att se om den uppfyller sitt syfte.

# \\hvfs03.intern.hoglandet.se\SkolaHem01$\emansan\My Pictures\Banbuilder.PNGGenomförande

Figur 2 Visar resultatet av filtrering med hjälp av Banbuilder

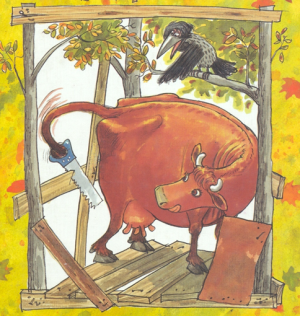
Arbetet började med att vi skapade en webbsida som innehåller en textruta för inmatning av text, som sedan ska granskas. I arbetets tidiga stadium byggde vi in Banbuilder för granskning av den inmatade texten. Vi upptäckte att Banbuilder delvis hanterade bokstäver som, på grund av grafisk likhet, har bytts ut mot siffror eller andra tecken.

Figur 3 Resultatet av en Banbuilderfiltrering av olämpligt ord med dubbel vokal

Banbuilders filtrering använder sig av reguljära uttryck för att hitta ord med utbytta bokstäver. Detta gör det möjligt att i viss utsträckning hantera Leetspeek (där man bland annat byter ut en del bokstäver mot siffror eller andra tecken som har grafiska likheter med det utbytta tecknet). Med hjälp av reguljära utryck är det möjligt att hitta uttryck som upprepas 0 eller flera gånger. Denna funktion finns inte i Banbuilder.

Efter närmare undersökning av systemet konstaterade vi att det är extra omständigt att ändra om Banbuilder för att få det att passa vårt arbete. Det finns flera anledningar till det. Det är svårare att komma in som utomstående i ett programmeringsprojekt och på en gång förstå hur koden fungerar och därmed också svårare att fortsätta att utveckla. Dessutom är man tvungen att, på varje dator som används som webbserver för denna sida, installera extra funktioner. Exakt hur detta skulle ske, och implementeras i koden, var så pass komplicerat att vi valde bort Banbuilder och istället skapade ett eget system.

Parallellt med arbetet med att ta fram programkoden för granskningssystemet har listor på svenska ord tagits fram. Det är i grunden två ordlistor. Den ena är en vanlig ordlista, med så många ord som möjligt och den andra är en lista med kraftuttryck. Listan med kraftuttryck byggs in i systemet så att systemet hittar dessa ord i de texter som skickas in i systemet.

Utöver textgranskningen har vi testat bildgranskning med utgångspunkt i Banbuilders bildgranskning. Även denna granskning byggdes in på en webbsida. Denna granskning gör genom att jämföra färgerna i bilden med spektrum för hudfärger. Vi ändrade programmet så att vi såg hur många procent som var hudfärgat och så gav den svaren ” This image is most likely porn.” och ” This image is most likely not porn.". vid testning av Mamma Mu bild så gav den över 50 % var hud färgad se figur 4.  
Vi testade olika bilder

Figur 4: Mamma mu sågar

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tabell över bilderna i bildtestet med resultatet av hudigenkänning i procent | | | | | |
| Namn | Bild | Procent | Namn | Bild | Procent |
| Mamma Mu |  | 50 % | Amy |  | 46 % |
| Måns |  | 0 % | Close-  window |  | 69 % |
| Gustav |  | 22 % |  |  |  |
| Pontus |  | 7 % |  |  |  |
| Svamp |  | 33 % |  |  |  |
| Mörkhyad man |  | 19 % |  |  |  |
| Bröst- bild |  | 81 % |  |  |  |
| Pingviner |  | 1 % |  |  |  |

## Resultat

### Textfilter

Resultatet blev en applikation som är baserad på både egna idéer och idéer från Banbuilder. Koden är helt egenutvecklad med undantaget att vi tog ett stycke programkod från Banbuilder, den koden byter ut ”leetspeak”-tecken mot motsvarigheten i det svenska alfabetet. Den färdiga applikationen har följande flödesschema:

C:\fågelmat\Untitled Diagram.png

Flödesschemat beskriver hur filtreringen fungerar för ett enskilt ord men programmet körs för alla ord i inmatningen. Detta för att flödet ska bli mer tydligt.

Status 0 betyder att texten inte innehåller några fula ord på något vis och då är det mycket säkert att texten inte är olämplig eftersom programmet ger status 2 eller status 1 väldigt ofta.

Status 1 betyder att det är otydligt om texten är olämplig eller inte, vilket betyder att texten måste

granskas av en person före publicering på till exempel Blocket.

I det här sammanhanget betyder status 2 att texten innehåller ett ord som absolut inte är passande i sammanhanget och att det måste filtreras bort.

Här nedan följer exempel på inmatade texter i fulfiltret samt utmatade fulheter.

|  |  |  |
| --- | --- | --- |
| Exempel | Text | Fulhet |
| 1 | ”Två personer som inte känner varandra så väl står och pratar på en fest och kommer in på yrken och sysselsättningar. ”Jaså, jobbar du med undertextning?” utbrister den ena. ”Vet du vad jag såg en gång? Att man hade översatt ’we have a lot of marines to feed’ med ’vi har många mariner att mata’. Är det inte pinsamt när det blir så fel?” Jo. Det man kan göra är att se till att förutsättningarna för dem som översätter och textar är så pass bra att risken för att fel ska uppstå är så liten som möjligt, samt ta ansvar för produktionen från början till slut. Det gör vi. Svenskt Medietext startades 2006 med ett klart och tydligt syfte: att se undertextare som en kvalificerad yrkesgrupp och översättning och textning som ett hantverk. Det innebär att vi sedan starten har prioriterat bra villkor och avtalsenliga löner för våra översättare och undertextare och tycker att kollektivavtal är en självklarhet. Vi låter alla översättningar granskas av en redaktör som tillsammans med översättaren sedan gör en slutversion. Och vi tummar aldrig på vår huvudregel: vi tar textning på allvar. ” | 0 |
| 2 | ”Du din idiot, dig ska jag döda. Din brevlåda ska jag spränga. Din mamma ska jag döda. ” | 0 |
| 3 | ”Jag har en lam lama som är fantastiskt fin. Den äter gräs hela dagarna till skillnad från dig. Du röker gräs hela dagarna. ” | 0 |
| 4 | ”Göken Göran gökade en gök. Göken gillade att göka Görans gök Gösta med Göran. Gökade göken Göran?” | 2 |
| 5 | ”För i helvete, vilket jävla fanskap kom på detta?” | 2 |
| 6 | ”Du är en jävelskapande gubbe.” | 1 |
| 7 | ”Jääävlar vad trevligt vi hade det. ” | 0 |

Det som syns i resultaten ovan är att vanlig text utan olämpligheter gick igenom filtret ordentligt, även exemplet om laman. De felaktigheter som uppstod var att hotet som finns i exempel 2 inte upptäcktes. Det är för att filtret enbart upptäcker enstaka fula ord och bedömer texten utifrån dem. I hotet finns ordet "döda" som egentligen skulle kunna läggas till i listan med olämpliga ord beroende på hur filtret ska användas. Även exempel 7 skulle inte ha gått igenom men det beror på att filtret inte klarar av ord där enstaka bokstäver i ordet har tagits bort eller lagts till.

De exempel som inte gick igenom var exempel 4, 5 och 6, vilket stämmer. Exempel 6 innehåller ett sammansatt olämpligt ord; "jävelskapande", ger fulheten 0.

Den fullständiga koden som vi har skrivit för textgranskning återfinns under bilagor.

### Bildgranskning

Vi kom fram till att bildgranskning är svårare än text granskning, för att på bildgranskningen så använde vi oss av en hudfärgskala som gav utslag på saker som den inte skulle, som till exempel ”Mamma Mu” se figur 4. Det finns även en chans att den inte ger utslag för bilder som den ska get utslag för, t.ex. om bilden är i gråskala eller är utzoomad.

## Resultatdiskussion

Det program som vi har utvecklat klarar av att identifiera ord i användargenererade texter. Det kräver dock att man på förhand har listat de ord som den ska söka efter och att även ordlistan med ”alla” svenska ord är uppdaterad med nytillkomna ord. Den klarar i dagsläget inte heller av engelska ord, eftersom ordlistorna enbart innehåller svenska ord.

Porgrammet saknar möjligheten att tolka ord där bokstäver upprepas fler gånger i ett ord för att lura systemet. Den klarar inte heller av felstavningar. Däremot är det möjligt för den att tolka ”leetspeek”, i en viss utsträckning. Det förutsätter en inte för komplex ”leetspeek”-text och att de utbytta tecknen följer de fördefinierade reguljära uttryck-strängarna.

Bildgranskningen är inte pålitlig utan ger mycket missvisande eftersom att den går på färg så kan till exempel en svartvit bild lura systemet och därför godkänner bilder som den inte borde. Detta är alltså en väldig opålitlig granskare.

## Metoddiskussion

Det slutgiltiga systemet är inte perfekt. Det saknar en del funktioner. Skulle vi ha genomfört detta arbete i full skala hade vi utgått från autentiska exempel på hatkommentarer och olämpligt beteende på internet. Vi hade undersökt möjligheten att låta en testgrupp testa systemet och låta deras språk och ordval bidra till systemets granskning.

# Slutsatser

Det är till viss del möjligt att filtrera bort ord, som till exempel svordomar, men det kan bli komplicerat då det kräver att man på förhand har byggt systemet så att det hanterar språkliga variationer och felstavningar. När det kommer till filtrering av bilder är det nästan omöjligt att göra ett system med ett felsäkert resultat.

## Egna reflektioner

Arbetet har varit roligt och lärorikt. Det har fungerat bra att arbeta tillsammans. Vi tycker att det har fungerat bra att ha avstämningspunkter under arbetets gång, med tydliga delmål i arbetet för varje avstämningspunkt.

# Litteraturförteckning

"Snipe", 2016-04-06. *Github.* [Online]   
Available at: https://github.com/snipe/banbuilder

Kundservice, B., 2016. *mailkonversation med Blockets kundtjänst* [Intervju] (27 01 2016).

Kundservice, B., 2016. *Mailkonversation med Blockets kundtjänst* [Intervju] (20 01 2016).

WebPurify, 2016. *WebPurify.* [Online]   
Available at: https://www.webpurify.com/  
[Använd 02 02 2016].

Wikipedia, 2016. *PHP.* [Online]   
Available at: https://en.wikipedia.org/wiki/PHP  
[Använd 02 02 2016].

# Bilagor

<?php

**class** Censor {

**private** $censorChar = "\*";

**private** $badwords = **array**();

**private** $wordlist = **array**();

**public** **function** initialize() {

**header**('Content-Type: text/html; charset=UTF-8');

**mb\_internal\_encoding**('UTF-8');

**mb\_http\_output**('UTF-8');

**mb\_http\_input**('UTF-8');

**mb\_regex\_encoding**('UTF-8');

$this->badwords = **file**("Svenska.txt", **FILE\_IGNORE\_NEW\_LINES**);

$this->wordlist = **file**("ordlista.txt", **FILE\_IGNORE\_NEW\_LINES**);

$leet\_replace = **array**();

$leet\_replace['a'] = '(a|a\.|a\-|4|@|Á|á|À|Â|à|Â|â|Ã|ã|α|Δ|Λ|λ)';

$leet\_replace['b'] = '(b|b\.|b\-|8|\|3|ß|Β|β)';

$leet\_replace['c'] = '(c|c\.|c\-|Ç|ç|¢|€|<|\(|{|©)';

$leet\_replace['d'] = '(d|d\.|d\-|&part;|\|\)|Þ|þ|Ð|ð)';

$leet\_replace['e'] = '(e|e\.|e\-|3|€|È|è|É|é|Ê|ê|∑)';

$leet\_replace['f'] = '(f|f\.|f\-|ƒ)';

$leet\_replace['g'] = '(g|g\.|g\-|6|9)';

$leet\_replace['h'] = '(h|h\.|h\-|Η)';

$leet\_replace['i'] = '(i|i\.|i\-|!|\||\]\[|]|1|∫|Ì|Í|Î|Ï|ì|í|î|ï)';

$leet\_replace['j'] = '(j|j\.|j\-)';

$leet\_replace['k'] = '(k|k\.|k\-|Κ|κ)';

$leet\_replace['l'] = '(l|1\.|l\-|!|\||\]\[|]|£|∫|Ì|Í|Î|Ï)';

$leet\_replace['m'] = '(m|m\.|m\-)';

$leet\_replace['n'] = '(n|n\.|n\-|η|Ν|Π)';

$leet\_replace['o'] = '(o|o\.|o\-|0|Ο|ο|Φ|¤|°|ø)';

$leet\_replace['p'] = '(p|p\.|p\-|ρ|Ρ|¶|þ)';

$leet\_replace['q'] = '(q|q\.|q\-)';

$leet\_replace['r'] = '(r|r\.|r\-|®)';

$leet\_replace['s'] = '(s|s\.|s\-|5|\$|§)';

$leet\_replace['t'] = '(t|t\.|t\-|Τ|τ)';

$leet\_replace['u'] = '(u|u\.|u\-|υ|µ)';

$leet\_replace['v'] = '(v|v\.|v\-|υ|ν)';

$leet\_replace['w'] = '(w|w\.|w\-|ω|ψ|Ψ)';

$leet\_replace['x'] = '(x|x\.|x\-|Χ|χ)';

$leet\_replace['y'] = '(y|y\.|y\-|¥|γ|ÿ|ý|Ÿ|Ý)';

$leet\_replace['z'] = '(z|z\.|z\-|Ζ)';

$leet\_replace['å'] = '(å|å\.|å\-|Å)';

$leet\_replace['ä'] = '(ä|ä\.|ä\-|Ä)';

$leet\_replace['ö'] = '(ö|ö\.|ö\-|Ö)';

$this->leet\_replace = $leet\_replace;

}

**public** **function** run($text, $censor = **false**) {

$output = **array**();

$words = **explode**(" ", $text);

**foreach**($words **as** $word) {

$badness = 0;

$outputWord = $word;

$lowerWord = $word;

**foreach**($this->leet\_replace **as** $key => $value)

$lowerWord = **preg\_replace**('/' . $value . '/i', $key, $lowerWord);

**if**(**in\_array**($lowerWord, $this->badwords)) {

$badness = 2;

$outputWord = $this->censorString($word);

} **else** **if**(!**in\_array**($lowerWord, $this->wordlist)) {

$current = $lowerWord;

**foreach**($this->badwords **as** $badword) {

**while**(**mb\_strpos**($current, $badword) !== **false**) {

$current = $this->censorStringInString($badword, $current);

$badness = 1;

}

}

**if**(**strcmp**($current, $lowerWord) != 0)

$outputWord = $current;

}

**if**(!$censor) {

**if**(!**array\_key\_exists**($word, $output))

$output[$word] = **array**();

**array\_push**($output[$word], $badness);

} **else**

**array\_push**($output, $outputWord);

}

**if**(!$censor)

**return** $output;

**else** {

$outputString = "";

**foreach**($output **as** $outputWord) {

$outputString .= ($outputWord . " ");

}

**return** $outputString;

}

}

**private** **function** censorString($string) {

**return** $this->censorStringInString($string, $string);

}

**private** **function** censorStringInString($string, $mother) {

$output = $mother;

$pos = **mb\_strpos**($mother, $string);

**if**(**is\_numeric**($pos)) {

**for**($i = $pos; $i < **mb\_strlen**($string) + $pos; ++$i) {

$output = $this->mb\_substr\_replace($output, $this->censorChar, $i, $i + 1);

}

} **else**

**echo**("There was an issue with the word \"" . $mother . "\" ");

**return** $output;

}

**private** **function** mb\_substr\_replace($output, $replace, $posOpen, $posClose) {

**return** **mb\_substr**($output, 0, $posOpen) . $replace . **mb\_substr**($output, $posClose);

}

}

?>