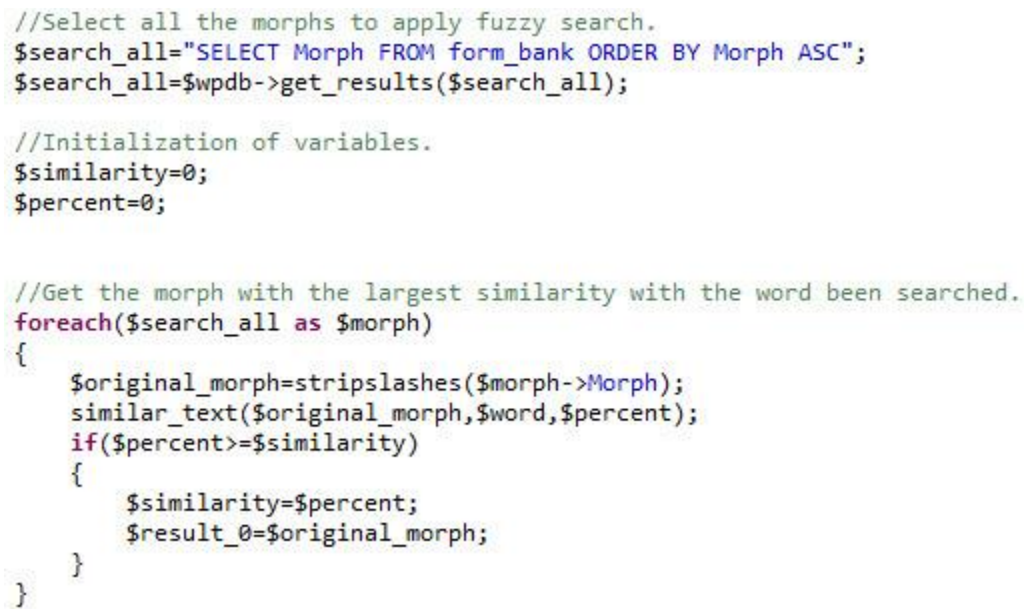
# INTRODUCTION TO SOME USEFUL FUNCTIONS IN PHP

We all hope that our plugin can be powerful and user-friendly, thus sometimes we may want to achieve special goals in our plugin through coding in PHP. I believe that some of us are experts in coding and have no trouble write these functions themselves. However, PHP is a powerful language and it offers many functions that significantly reduce our workload. Using these functions properly and we are able to build a more powerful and robust project with improved efficiency. I want to introduce some functions I used in our project, and I hope these functions will help you.

1. Implementation of Fuzzy Search

If you are to develop a plugin involves in searching in the database, you ought to take the possible mistakes people may make during typing into consideration. This is what I learned from Mr. Koo and I think you should all pay some attention to this issue. This kind of search is called fuzzy search, meaning that even if the user incorrectly type the words, we are also obliged to offer the information he wants. Typically, this kind of work can be done through the comparison of similarity between the word been typed and the tuples in the database. PHP offers such a comparison function called *similar_text*, which can compare two strings and assign the percentage of similarity to a certain variable. Here is an example of my use of this function:



```
//Select all the morphs to apply fuzzy search.
$search_all="SELECT Morph FROM form_bank ORDER BY Morph ASC";
$search_all=$wpdb->get_results($search_all);

//Initialization of variables.
$similarity=0;
$percent=0;


//Get the morph with the largest similarity with the word been searched.
foreach($search_all as $morph)
{
    $original_morph=stripslashes($morph->Morph);
    similar_text($original_morph,$word,$percent);
    if($percent>=$similarity)
    {
        $similarity=$percent;
        $result_0=$original_morph;
    }
}
```

First all the words in the table form_bank are stored in a variable. Then they are compared with the word typed by the user, return the similarity to the variable $percent, and one with the maximum similarity will be returned as the searching result. It is not difficult to figure out that the similarity of two identical words is 100.

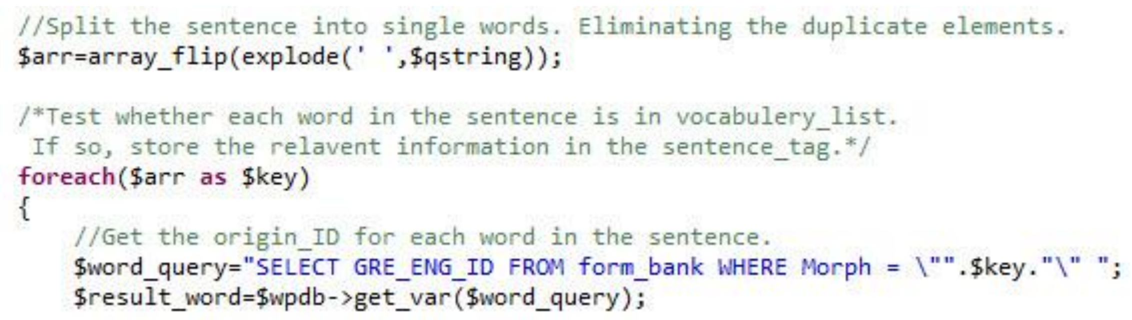
However, this method remains some problems that we should pay special attention to. First of all, if the user correctly type a word but it does not exist in the database, this fuzzy search will also return an irrelevant result, which is not desired by the user. One possible solution is to let the user add new tuples to the database if the searching result is not satisfying. Another problem with this fuzzy search is related to the algorithm of the function *similar_text*. The algorithm contains iteration, and the complexity is $O(N$^3), where N refers to the length of the longer string. Therefore, when the strings to be compared are too long, such comparison is slow. One solution to this problem is to compare part of the two strings instead of the whole strings. Maybe there are other better solutions with the problems I come up with, hope to receive your feedback.

2. Avoid Repetitive Insertions to the Database

Sometimes it may incur us that we are to extract data from a string and insert it into the database. However, if the string contains redundant data, i.e. two tuples to be inserted are identical, severe problem will happen.

To deal with this problem, a possible solution is to go through the data to be inserted and eliminate the repetitive ones. This requires additional coding work, and our solution may be slow and inefficient. PHP offers a function called *array_flip*, which returns a reversed array, which means that the original keys are now values, and the original values are now keys. From the property of keys we know that two keys with an identical name will be rejected, and it is how this function works: if a key appears more than once, only the last one will be remained, while others were eliminated. Properly using this function we can eliminate the duplicate in the insertion.

Someone may notice that this function returns a reversed array. Fortunately, it doesn't matter since most of the future use does not involve in whether the information is a key or a value. Even if it matters, apply the same function and we obtain a duplicate-eliminated array. The following example shows my use of this function in the project.



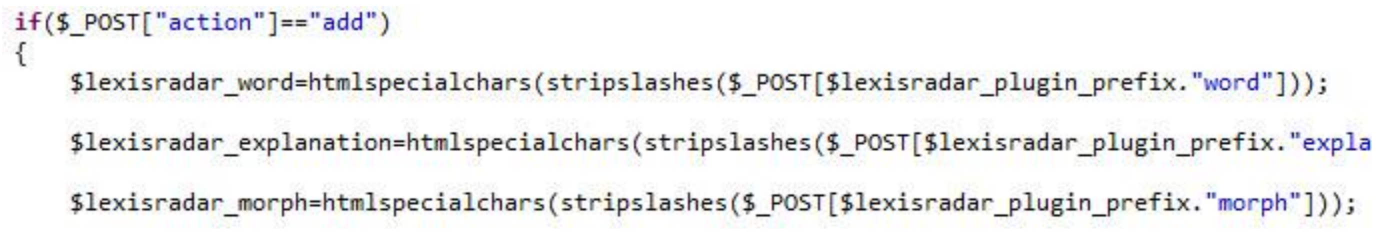
```
//Split the sentence into single words. Eliminating the duplicate elements.
$arr=array_flip(explode(' ',$qstring));

/*Test whether each word in the sentence is in vocabulery_list.
 If so, store the relavent information in the sentence_tag.*/
foreach($arr as $key)
{
    //Get the origin_ID for each word in the sentence.
    $word_query="SELECT GRE_ENG_ID FROM form_bank WHERE Morph = \"".$key."\" ";
    $result_word=$wpdb->get_var($word_query);
```

## 3. Eliminate Undesired Symbols and Tabs

Sometimes the data to be inserted into the database comes from the computer-generated text, which may contain some undesired symbols or tabs which is important to computer language but disastrous to database maintenance. PHP offers functions dealing with this problem, and they can be nested to deal with different problems. The functions I frequently use include *trim*, *stripslashes*, and *htmlspecialchars*.

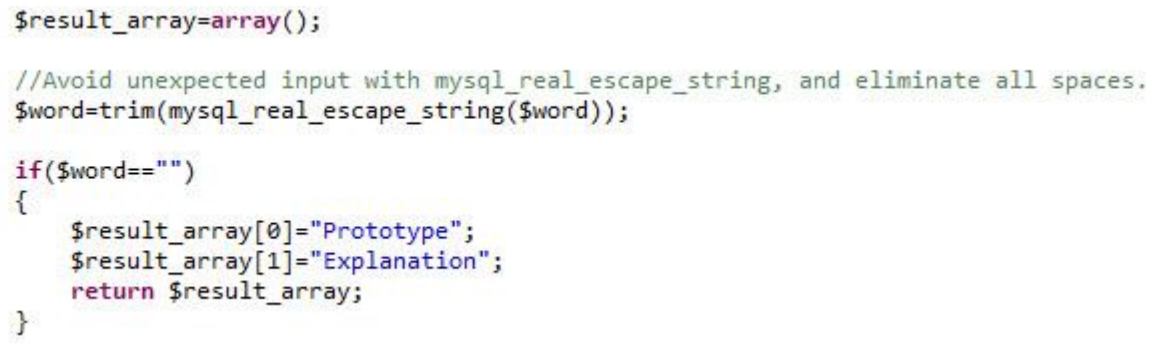
The function *trim* eliminate the blank characters in the array and returns an array without such tabs. The characters to be eliminated include spaces, tabs, newlines, etc. Consequently, we will get the pure text from the string.

Function *stripslashes* and *htmlspecialchars* are widely used especially when sending message from the webpage. They are used to deal with the special characteristics in HTML and ensure that the content been returned is pure text. Using this method, we can effectively avoid the error caused by unpredictable input context. Here are examples of my use of these functions.



```
if($_POST["action"]=="add")
{
    $lexisradar_word=htmlspecialchars(stripslashes($_POST[$lexisradar_plugin_prefix."word"]));

    $lexisradar_explanation=htmlspecialchars(stripslashes($_POST[$lexisradar_plugin_prefix."expla

    $lexisradar_morph=htmlspecialchars(stripslashes($_POST[$lexisradar_plugin_prefix."morph"]));
```

Another example:



```
$result_array=array();

//Avoid unexpected input with mysql_real_escape_string, and eliminate all spaces.
$word=trim(mysql_real_escape_string($word));

if($word=="")
{
    $result_array[0]="Prototype";
    $result_array[1]="Explanation";
    return $result_array;
}
```

Those above only cover a tiny part of the powerful PHP functions, and most of the functions we can think about have already been solved. Therefore, I think it a good habit to check in the API to see whether there is a function that meets your requirement.

Source: http://toyhouse.cc/profiles/blogs/personal-report-of-week-9