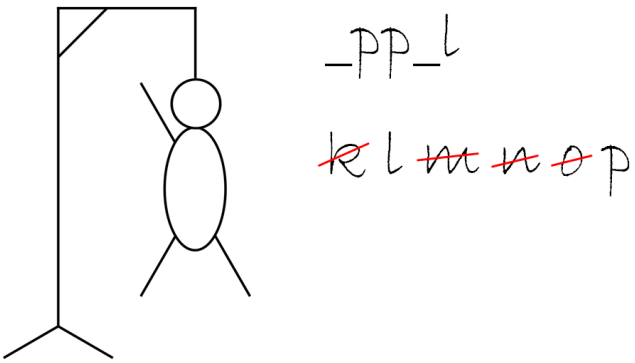
Cheating hangman

Hangman is a paper and pencil guessing game for two players. One player thinks of a word and the other tries to guess it by suggesting letters. The word to guess is represented by a pattern that initially contains a sequence of underscores. Herewith, each underscore (_) represents a letter of the word. The number of underscores in the initial pattern thus corresponds to the number of letters in the word to guess.



If the guessing player suggests a letter which occurs in the word, the other player fills up the pattern at all open positions (indicated by underscores) where the letter occurs in the word. If the suggested letter does not occur in the word, the other player draws one element of a hanged man stick figure as a tally mark. The guessing player wins the game if he completes all letters of the word before all elements of the hanged man stick figure have been drawn. Otherwise the other player wins the game.

Assignment

The aim of this assignment is to make an implementation of the player of the hangman game that thinks of a word that the other player must guess. Only words containing lowercase letters are considered. The player, however, make use of a cheating strategy that makes it as a hard as possible for the other player to win the game. The idea behind this strategy is simple. Instead of choosing fixed word of a given length n at the start of the game, the player initially considers all words of length n as possible candidates. Each time the other player suggests a new letter, the player fills the pattern in such a way that the number of remaining candidates that are consistent with the pattern and all suggested letters is maximized. By following this strategy, most of the time another body part will be added to the gibbet. To make it concrete, you should implement at least the following functions:

- Write a function candidate that takes three string arguments. The first argument represents a word (containing lowercase letters only), the second a pattern that can be partially filled (contains underscores and lowercase letters only), and the third contains all (lowercase) letters that have been suggested previously. The function must return a Boolean value that indicates whether or nit the word is consistent with the pattern and all previously suggested letters. This is the case if the word has the same length as the pattern, and each position of the pattern either contains the same letter at the corresponding position of the word that is also contained in the previously suggested letters, or contains an underscore while the letter at the corresponding position of the word is not contained in the previously suggested letters.

- Write a function fill that takes three string arguments. The first argument represents a word (containing lowercase letters only), the second a pattern that can be partially filled (contains underscores and lowercase letters only), and the third contains a single lowercase letter. The function may assume that the given word is a candidate for the given pattern. The function must return a new pattern, that consists of the given pattern that was filled with all occurrences of the new letter in the given word.
- Use the functions candidate and fill to write a function select that takes three string arguments. The first argument represents a pattern that can be partially filled (contains underscores and lowercase letters only), the second contains all (lowercase) letters that have been suggested previously (where the function may assume that all letters in the pattern occur in this sequence of letters), and the third contains a single lowercase letter that does not occur in the previous argument. The function may also assume that the current directory contains a text file words.txt that contains a list of words (lowercase letters only), each on a separate line. First, the function must determine a new pattern for each remaining candidate (the words of the file words.txt that are consistent with the given pattern and the previously suggested letters (second argument)) that consists of the given pattern that is filled with the newly suggested letter (third argument). Then, the function must return a tuple containing two strings. The first string of the tuple contains the new pattern that was found most often during the previous step. The second string of the tuple is a randomly selected word from the file words.txt that is consistent with the selected new pattern (first string of tuple) and all suggested letters (from both the second and third argument).

Example

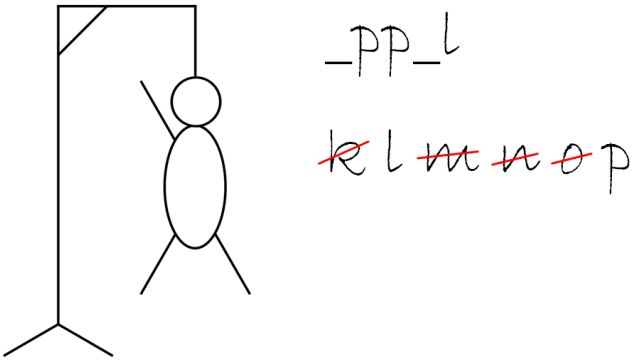
```
>>> candidate('apple', '_ppl_', 'klmnop')
True
>>> candidate('apples', '_ppl_', 'klmnop')
False
>>> candidate('apple', '_ppl_', 'p')
False
>>> candidate('apple', '_ppl_', 'apl')
False
>>> candidate('apple', '__pl_', 'klmnop')
False
>>> candidate('apple', '_kpl_', 'klmnop')
False
>>> candidate('angel', '_ppl_', 'klmnop')
False

>>> fill('apple', '_pp__', 'p')
'_pp__'
>>> fill('apple', '_pp__', 'l')
'_ppl_'
>>> fill('apple', '_ppl_', 'x')
'_ppl_'

>>> select('_ppl_', 'klmnop', 'a')
('appl_', 'apple')
>>> select('a__l_', 'al', 'p')
('a__l_', 'angle')
>>> select('a__l_', 'abcdefghijklmno', 'p')
('a__l_', 'artly')
>>> select('_ppl_', 'klmnop', 'x')
('_ppl_', 'apple')
```

```
>>> select('_____', '', 'x')
('_____', 'angle')
```

Galgje is een spel voor twee spelers dat met potlood en papier kan gespeeld worden. Eén van de spelers denkt aan een bepaald woord, dat de andere speler moet door telkens een nieuwe letter te suggereren. Om het woord te kunnen raden krijgt die speler een patroon te zien dat initieel bestaat uit een reeks underscores. Hierbij stelt elke underscore (_) een letter van het woord voor. Het aantal underscores in het patroon komt dus overeen met de lengte van het te raden woord.



Als de speler die het woord moet raden een letter suggereert die in het woord voorkomt, dan moet de andere speler de letter in het patroon invullen op alle open posities (aangegeven door underscores) waar de letter in het woord voorkomt. Als de gesuggereerde letter niet in het woord voorkomt, dan moet de andere speler een volgende stok tekenen van een stokfiguur die een galg met daaraan een hangend mannetje voorstelt. De speler die het woord moest raden wint indien hij alle letters van het woord heeft gevonden voordat de andere speler alle stokken van de stokfiguur getekend heeft. Anders wint de speler die het woord in gedachten hield.

Opgave

Het doel van deze opgave is om een implementatie te maken van de speler van een spelletje galgje die een woord in gedachten moet houden. Hierbij beschouwen we enkel woorden die bestaan uit kleine letters. De speler gebruikt echter een bedriegelijke strategie die het de andere speler zo moeilijk mogelijk maakt om het spel te winnen. Het idee achter deze strategie is zeer eenvoudig. In plaats van bij aanvang van het spelletje een vast woord van een bepaalde lengte n te kiezen, beschouwt de speler initieel alle woorden van lengte n als mogelijke kandidaten. Telkens wanneer de andere speler een nieuwe letter suggereert, werkt hij het patroon zo bij dat er zo veel mogelijk kandidaten overblijven die consistent zijn met het vorige patroon en alle gesuggereerde letters. Concreet ga je bij de implementatie als volgt te werk:

- Schrijf een functie `kandidaat` waaraan drie stringargumenten moeten doorgegeven worden. Het eerste argument stelt een woord voor (bevat enkel kleine letters), het tweede een patroon dat reeds deels kan ingevuld zijn (bevat enkel underscores en kleine letters), en het derde bevat alle (kleine) letters die reeds gesuggereerd werden. De functie moet een Booleaanse waarde teruggeven, die aangeeft of het woord consistent is met het patroon en alle letters die reeds gesuggereerd werden. Dat is het geval als het woord dezelfde lengte heeft als het patroon, en elke positie van het patroon ofwel een letter bevat die gelijk is aan de overeenkomstige letter van het woord en voorkomt in de reeds gesuggereerde letters, ofwel een underscore bevat en de overeenkomstige letter van het woord niet voorkomt in de reeds gesuggereerde letters.

- Schrijf een functie aanvullen waaraan drie stringargumenten moeten doorgegeven worden. Het eerste argument stelt een woord voor (bevat enkel kleine letters), het tweede een patroon dat reeds deels kan ingevuld zijn (bevat enkel underscores en kleine letters), en het derde één enkele kleine letter. Hierbij mag de functie ervan uitgaan dat het gegeven woord een kandidaat is voor het gegeven patroon. De functie moet een nieuw patroon teruggeven, dat bestaat uit het gegeven patroon dat verder werd aangevuld met alle voorkomens van de gegeven letter in het gegeven woord.
- Gebruik de functies kandidaat en aanvullen om een functie kiezen te schrijven waaraan drie stringargumenten moeten doorgegeven worden. Het eerste argument stelt een patroon voor dat reeds deels kan ingevuld zijn (bevat enkel underscores en kleine letters), het tweede bevat de (kleine) letters die reeds gesuggereerd werden (waarbij je er mag van uitgaan dat alle letters in het patroon in deze reeks letters voorkomen), en het derde bevat een nieuw gesuggereerde (kleine) letter die niet voorkomt in de letters uit het vorige argument. De functie mag er ook van uitgaan dat de huidige directory een tekstbestand woorden.txt bevat, dat bestaat uit een lijst van woorden (enkel kleine letters) die elk op een afzonderlijke regel staan. De functie moet eerst voor alle overblijvende kandidaten (de woorden uit het bestand woorden.txt die consistent zijn met het gegeven patroon en de reeds gesuggereerde letters (tweede argument)) een nieuw patroon bepalen dat bestaat uit het gegeven patroon aangevuld met de nieuw gesuggereerde letter (derde argument). Daarna moet de functie een tuple met twee strings teruggeven. De eerste string van het tuple bevat het nieuwe patroon dat in de vorige stap het vaakst bepaald werd. De tweede string van het tuple is een willekeurig gekozen woord uit het bestand woorden.txt dat consistent is met het geselecteerde nieuwe patroon (eerste string van het tuple) en alle gesuggereerde letters (zowel de letters uit het tweede argument als de letter uit het derde argument).

Voorbeeld

```
>>> kandidaat('appel', '_pp_l', 'klmnop')
True
>>> kandidaat('appels', '_pp_l', 'klmnop')
False
>>> kandidaat('appel', '_pp_l', 'p')
False
>>> kandidaat('appel', '_pp_l', 'apl')
False
>>> kandidaat('appel', '__p_l', 'klmnop')
False
>>> kandidaat('appel', '_kp_l', 'klmnop')
False
>>> kandidaat('angel', '_pp_l', 'klmnop')
False

>>> aanvullen('appel', '_pp__', 'p')
'_pp__'
>>> aanvullen('appel', '_pp__', 'l')
'_pp_l'
>>> aanvullen('appel', '_pp_l', 'x')
'_pp_l'

>>> kiezen('_pp_l', 'klmnop', 'a')
('app_l', 'appel')
>>> kiezen('a___l', 'al', 'p')
('a___l', 'angel')
```

```
>>> kiezen('a___l', 'abcdefghijkl', 'p')
('a_p_l', 'ampul')
>>> kiezen('_pp_l', 'klmnop', 'x')
('_pp_l', 'appel')
>>> kiezen('_____', '', 'x')
('_____', 'angel')
```