Homework 3

As usual, submit a file named `hw3.py`.

1. Define a function called `splice` that takes two strings. It should return a string consisting of the first half of the first string and the second half of the second string. Tip: `int(x)` drops the fractional part of x and returns an integer. Example:

```
>>> splice('dogfish', 'fishfood')
'dogfood'
```

2. Define a function called `scrunch` that takes a string s and returns a length-two string consisting of the first and last letters of s. However, if the input string is less than two letters long, `scrunch` should just return the original string. Examples:

```
>>> scrunch('dog')
'dg'
>>> scrunch('a')
'a'
```

3. Write a function `add_s` that takes a word (i.e., a string) as input, and returns the word with s added to the end. However, if the word ends in *s, z, sh,* or *ch,* the ending should be *es.* Examples:

```
>>> add_s('dog')
'dogs'
>>> add_s('fish')
'fishes'
```

4. Write a function `is_plural` that takes a word as input, and returns `True` or `False` depending on whether the word is plural or not. Consider a word to be plural if it ends in *s,* unless the word ends in *ss* or *us,* in which case it is not plural. Return `False` for all single-letter words. Examples:

```
>>> is_plural('dogs')
True
>>> is_plural('campus')
False
>>> is_plural('s')
False
```

5. Write a function `make_np` that takes a word w and returns a two-element list consisting of `'the'` followed by w. Then write a function `make_sentence` that takes three words: subject, verb, and object. It should return a list in which the subject and object have been turned into noun phrases, and the verb form has been modified to match the subject. Specifically, `'the'`

should be inserted before the subject and before the object, and *s* should be added to the verb if the subject is singular, but not if the subject is plural. In the definition of `make_sentence`, you MUST call the functions `is_plural`, `add_s`, and `make_np`. Examples:

```
>>> make_sentence('dogs', 'chase', 'cat')
['the', 'dogs', 'chase', 'the', 'cat']
>>> make_sentence('cat', 'scratch', 'sofa')
['the', 'cat', 'scratches', 'the', 'sofa']
```

You may assume that the input verb is in the plural form (no ending).

For each of the questions 6–11, you must write a LIST COMPREHENSION. The answer will not count as correct unless you use a list comprehension to obtain it. Set:

```
mytext = 'This is a test ; it is only a test .  It is not a pipe .'.split()
```

6. Define a function called `vocab` that takes a text or list, eliminates duplicates, discards numbers and punctuation (anything that is not purely alphabetic), and maps uppercase to lowercase. The return value should be a set. Then do:

```
V = vocab(txt)
```

If you have done it right, the set `V` will contain eight elements.

7. Get the list of four-letter words from `V`, put them in alphabetic order, and store the result in `A7`.

8. Get the list of words from `V` that begin and end with the same letter, sorted alphabetically. Single-letter words are allowed. Store the result in `A8`.

9. Define a function called `acronym` that takes a string, splits it into words, and returns a string consisting of the initial letters of the words. For example:

```
>>> acronym('International Business Machines')
'IBM'
>>> acronym('Lord of the Rings')
'LotR'
```

10. This question has two steps. You only need to use a list comprehension in the second step.

a. There is a function called `reversed` that iterates over the the characters of a string or the elements of a list in reverse order. It returns a generator, which we can turn into a list:

```
>>> list(reversed('wolf'))
['f', 'l', 'o', 'w']
```

A **palindrome** is a word that is written the same forward and backward. That is, reversing it does not change it. Define a function called `is_palindrome` that takes a word w and returns `True` if w is a palindrome and `False` if not. For example:

```
>>> is_palindrome('wolf')
False
>>> is_palindrome('bob')
True
```

b. Set `M = vocab(text1)`. Get the list of palindromes that occur in `M`, sorted alphabetically, and store the result in `A10`.

11. Define a word w to be **reversible** if w written backwards is also a word. Set `A11` to the list of reversible words in `M`, sorted alphabetically.

We will often have occasion to print tables. For example, suppose we want to print a table of strings with their lengths.

```
>>> words = ['hi', 'there', 'cat']
>>> for w in words:
...     print(w, len(w))
...
hi 2
there 5
cat 3
```

To improve readability, we would like to have the first column be five characters wide all the way down. The method `format` will pad strings with spaces to make them be the right width:

```
>>> '{:5}--{:1}'.format('cat', 2)
'cat  --2'
```

In the format string, each pair of braces corresponds to one argument. Characters outside of the braces are preserved as they stand. Thus:

```
>>> for w in words:
...     print('{:5} {:1}'.format(w, len(w)))
...
hi    2
there 5
cat   3
```

You can get right alignment of strings by inserting `>` before the field width:

```
>>> for w in words:
...     print('{:>5} {}'.format(w, len(w)))
...
   hi 2
there 5
  cat 3
```

12. Write a function called `text_table` that takes a list of texts and prints out a table in which the first column contains the text name, the second column contains the number of tokens in the text, and the third column contains the number of types. The first column should have width 55, the second has width 6, and the third column has width 5. You should also print a header line containing the words `'TITLE'`, `'#TOK'`, and `'#TYP'`. Use the `format` method for the header line as well, and right-align the header strings for the last two columns. Here is an example of the output:

```
>>> text_table([text1, text2, text3])
TITLE                                                #TOK  #TYP
Moby Dick by Herman Melville 1851                  260819 19317
Sense and Sensibility by Jane Austen 1811          141576  6833
The Book of Genesis                                 44764  2789
```