Sets of this and sets of that

Python provides a primitive set type (set) that implements the concept of mathematical sets. Sets support `x in set`, `len(set)`, and iteration using `for x in set`. But elements in a set do not have a position or order. Therefore, sets do not support indexing, slicing, or other sequence-like behavior.

**Part (a):** Working with a partner, test your understanding of sets: on a scratch paper, write what you think the program below will print. (You cannot know the order that elements in a set will be printed, just the elements that are printed.) Consult the Sets Cheat Sheet for answers to questions you may have about the various set operations.

```python
s1 = set("hello")
s2 = {'w', 'o', 'r', 'l', 'd'}

print (s1)
print (s2)

print (s1 & s2)
print (s1 | s2)
print (s1 - s2)

s3 = set(range(0,10,2))
print (s3)
print (s3.add("hi"))
print (s3)

if 4 in s3:
    s3.remove(4)
else:
    s3.remove(5)

print (s3)
print( set() >= s1 )
print (s1 & s2 <= s2 <= s1 | s2)

total = 0
for i in s3:
    try:
        total += i
    except TypeError:
        pass

print (total)
```

Step through the **Visualization** at the link on the website to check your answers.
Q1: Like lists and dictionaries, sets are mutable. Three method calls in the above program perform operations that do not return a value, but are executed for their side effect on the set. Identify the three method calls that mutate the set.

Part (b):

```python
import string

def get_input_file():
    while True:
        try:
            file_name = input("Enter the input file name: ")
            open_file = open(file_name)
            break
        except FileNotFoundError:
            print("File {} could not be opened.").format(file_name)
    return open_file

def read_text( text_file ) :
    result = ()
    num = 0
    for line in text_file:
        num += 1
        line_lst = line.strip().split()
        word_lst = [w.strip(string.punctuation) for w in line_lst]
        word_lst = [w.lower() for w in word_lst]
        for w in word_lst:
            try:
                result[w] = result[w] | {num}
            except KeyError:
                result[w] = {num}
    return result

def print_nums(word, num_iterable):
    INDENT = 3*
    if num_iterable:
        num_lst = sorted(num_iterable)
        num_str_lst = [str(num_int) for num_int in num_lst]
        num_str = ', '.join(num_str_lst)
    else:
        num_str = 'Not found'
    print('{}:
{}').format(word, INDENT, num_str)

file = get_input_file()
index = read_text(file)
file.close()
word_lst = ['Fissure', 'Who', 'They']
for w in word_lst:
    w_lc = w.lower()
    if w_lc in index:
        print_nums(w, index[w_lc])
    else:
        print_nums(w, set())
```

Assuming the contents of the file named `fishing.txt` are as shown on the left.

Assume that this file is located in the same directory as the program below.

Assuming the user enters `fishing.txt` at the prompt, what will be displayed when the program is executed?
Download **partB.py** and **fishing.txt** to the same folder. Run the program to check your answer.

**Part (c):**
Download **lineLookup.py** into the same folder as **fishing.txt**. Replace the stub for the function **contains_all** so that the program behaves as described in the comments.

For example:

```
In [57]:
```

```
PLTL/Exercises/IntroToSets')
Enter the input file name: fishing.txt
Enter one or more words to search on:
Till a fish
Till a fish:
    3
In [57]:
```

```
PLTL/Exercises/IntroToSets')
Enter the input file name: fishing.txt
Enter one or more words to search on:
Named for
Named for:
    Not found
In [46]:
```