Question 2 – Mirror Words

- Implement the procedure 'find_mirrors' found on the next page.
  - o in_file contains a list of words, one word per line.
  - o a sample list of words is available at: http://www.cs.duke.edu/~ola/ap/linuxwords
- You will write to out_file a list of in_file words where the first word is a mirror image (letter reversed) copy of the second word, and both words exist in in_file. There should be one pair of words per line.
- For example, out_file might contain:
  - o Bard/draB, Bud/duB, Are/erA, Bag/gaB, Brag/garB, etc.
- Requirements:
  - o Describe how your algorithm works
  - o Describe why you chose to implement it the way you did.
  - o Eliminate Palindrome words like eye, civic and deed
  - o Use a case sensitive compare: Aa would match aA but not aa
  - o Include a copy of out_file in your response
  - o Just implement find_mirrors, we're not looking for other improvements

Solution

Pseudocode/Algorithm

Steps:

1. Implement function find_mirrors(in_file, out_file)

```
def find_mirrors(in_file, out_file):
```

2. Open the input file in_file in 'read' mode

```
with open('linuxwords.txt', 'r') as in_file:
```

3. Open the output file out_file in 'write' mode

```
with open('output.txt', 'w') as out_file:
```

4. Read the in_file contents line by line

5. Pre-process the in_file data, Check if it has any of the special chars, white spaces, \t,\r etc chars and ignore them using .strip()

6. Initialize count for non-palindrome word = 0

7. Implement function is_palindrom(word) for checking whether words in in_file list lines[] one by one, Is palindrome or not/case-sensitive compare

```
def is_palindrom(word):
    word = ''.join(c.lower() for c in word if c not in string.punctuation)
# For making a case sensitive compare
# --   word != re.search('Ababa', 'ababA',re.IGNORECASE) ---
    if word == word[::-1]:
      return True

  return False
```

8. Run the function in step 7 for all the lines in in_file

9. If is_palindrome(word) != True, Append/Move non-palindrome words in in_file to to_write[] list and

to_write.append(word[::-1])

else

- Increment count++

- And print count of palindromes in in_file

print('there are ' + str(cnt) + ' palindromes!')

10. Iterate through the list of non-palindrome words in to_write[] list

for word in to_write[:-1]:

11. Write all the non-palindromes and reverse of words into out_file as follows

writing non-palindrome(original) word followed by '/' followed by reversed word
#till the end of the 'to_write' list to out_file

```
        out_file.write(word[::-1] + '/' + word + '\n')
    out_file.write(to_write[-1])
```

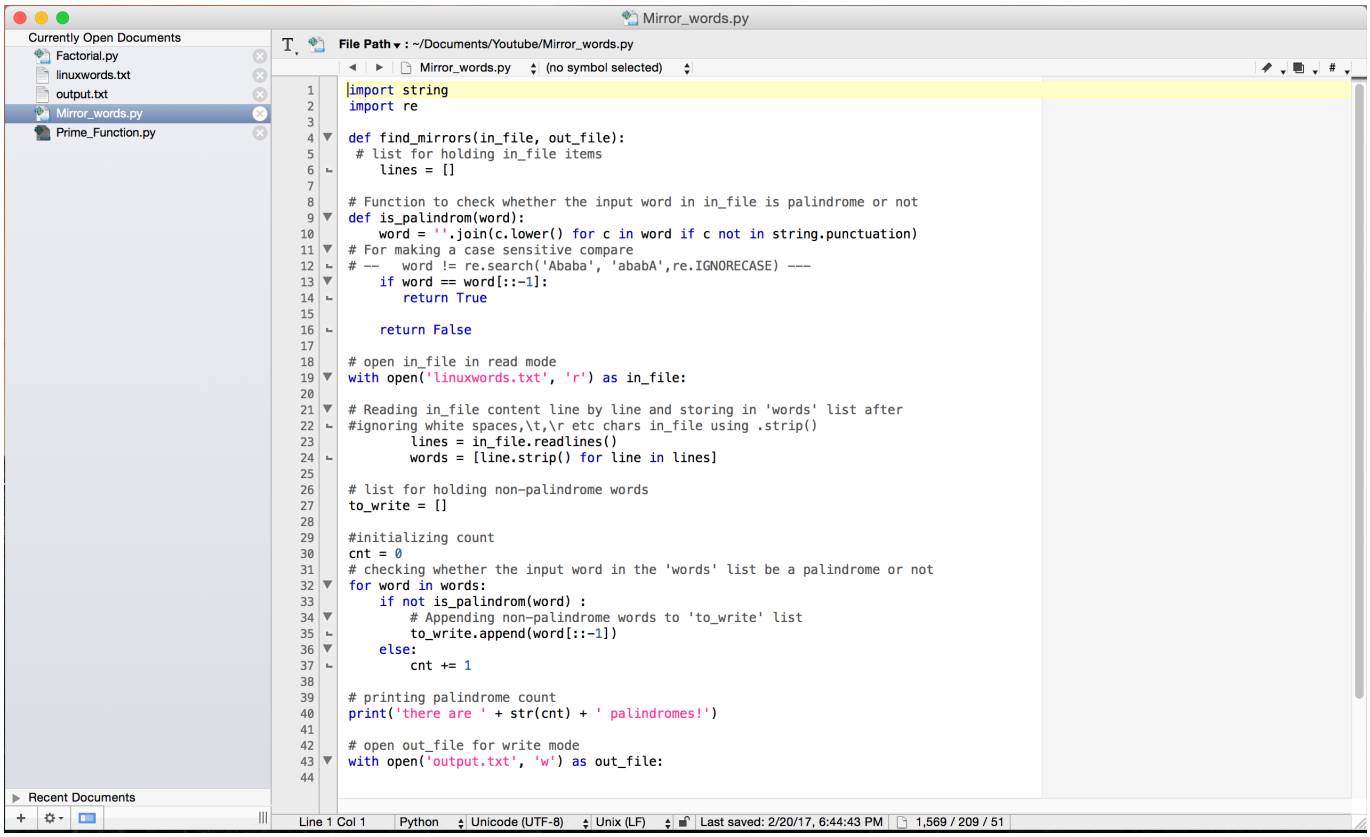
12. Close out_file out_file.close()

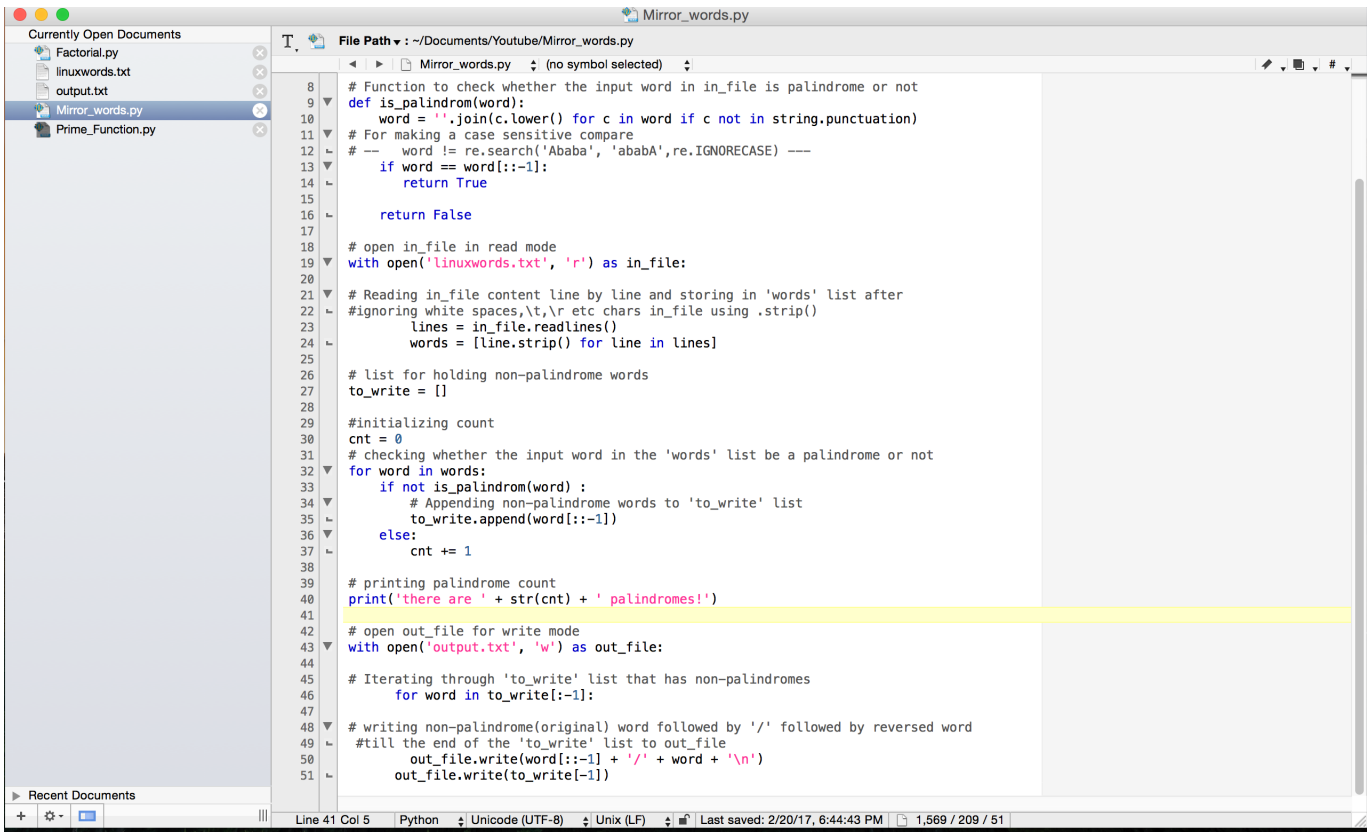
Program

```
import string
import re

def find_mirrors(in_file, out_file):
 # list for holding in_file items
   lines = []

# Function to check whether the input word in in_file is palindrome or not
def is_palindrom(word):
   word = ''.join(c.lower() for c in word if c not in string.punctuation)
# For making a case sensitive compare
# --   word != re.search('Ababa', 'ababA',re.IGNORECASE) ---
   if word == word[::-1]:
      return True

   return False

# open in_file in read mode
with open('in_file.txt', 'r') as in_file:
```

```
# Reading in_file content line by line and storing in 'words' list after
#ignoring white spaces,\t,\r etc chars in_file using .strip()
        lines = in_file.readlines()
        words = [line.strip() for line in lines]

# list for holding non-palindrome words
to_write = []

#initializing count
cnt = 0
# checking whether the input word in the 'words' list be a palindrome or not
for word in words:
            if not is_palindrom(word) :
                        # Appending non-palindrome words to 'to_write' list
                        to_write.append(word[::-1])
            else:
                        cnt += 1

# printing palindrome count
print('there are ' + str(cnt) + ' palindromes!')

# open out_file for write mode
with open('out_file.txt', 'w') as out_file:

# Iterating through 'to_write' list that has non-palindromes
      for word in to_write[:-1]:

# writing non-palindrome(original) word followed by '/' followed by reversed word
 #till the end of the 'to_write' list to out_file
            out_file.write(word[::-1] + '/' + word + '\n')
      out_file.write(to_write[-1])
```

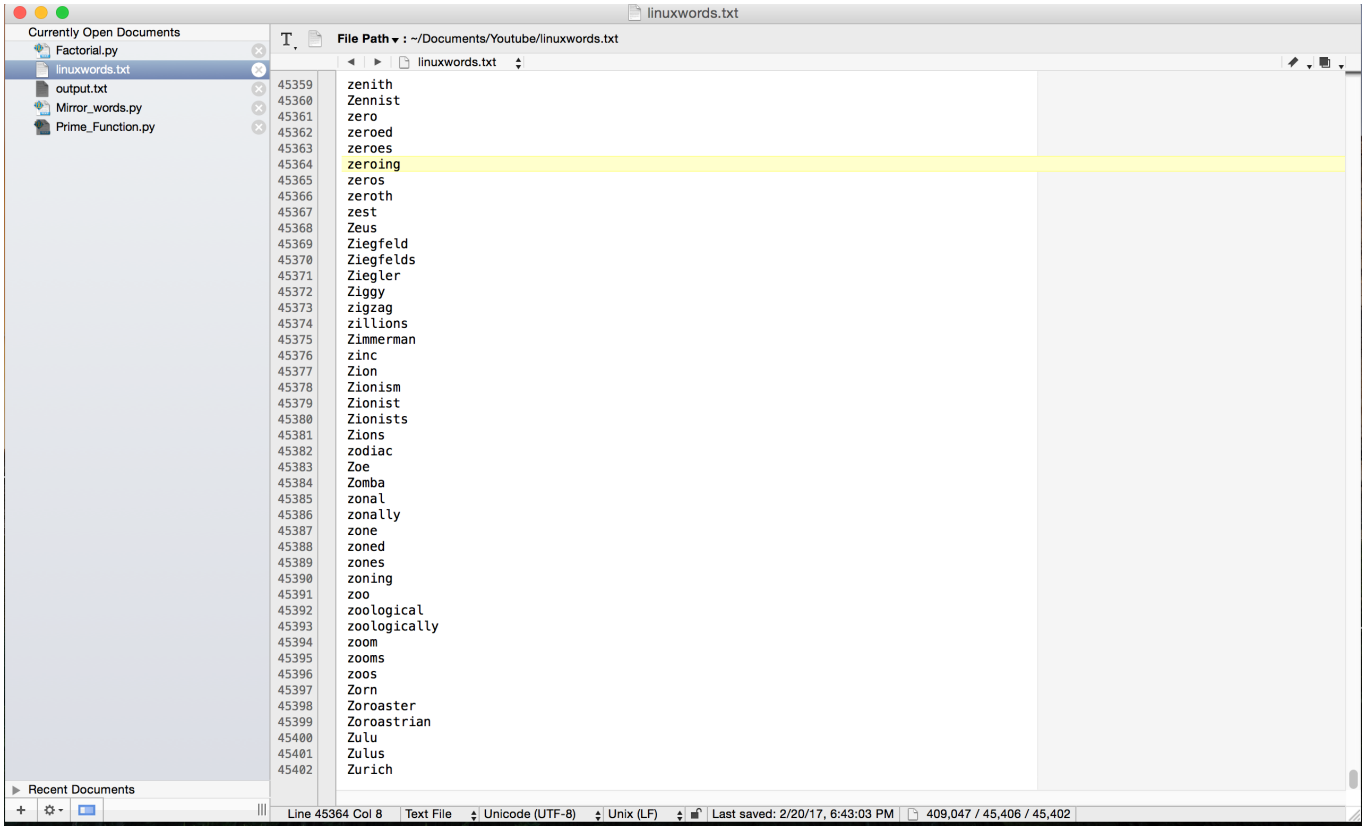
Screen shot:


Mirror_words.py
Currently Open Documents
Factorial.py
linuxwords.txt
output.txt
Mirror_words.py
Prime_Function.py
File Path : ~/Documents/Youtube/Mirror_words.py
Mirror_words.py
(no symbol selected)
import string
import re
def find_mirrors(in_file, out_file):
# list for holding in_file items
lines = []
# Function to check whether the input word in in_file is palindrome or not
def is_palindrom(word):
word = ''.join(c.lower() for c in word if c not in string.punctuation)
# For making a case sensitive compare
# -- word != re.search('Ababa', 'ababA',re.IGNORECASE) ---
if word == word[::-1]:
return True
return False
open in_file in read mode
with open('linuxwords.txt', 'r') as in_file:
Reading in_file content line by line and storing in 'words' list after
#ignoring white spaces,\t,\r etc chars in_file using .strip()
lines = in_file.readlines()
words = [line.strip() for line in lines]
# list for holding non-palindrome words
to_write = []
#initializing count
cnt = 0
# checking whether the input word in the 'words' list be a palindrome or not
for word in words:
if not is_palindrom(word) :
# Appending non-palindrome words to 'to_write' list
to_write.append(word[::-1])
else:
cnt += 1
# printing palindrome count
print('there are ' + str(cnt) + ' palindromes!')
# open out_file for write mode
with open('output.txt', 'w') as out_file:
Recent Documents
Line 1 Col 1
Python
Unicode (UTF-8)
Unix (LF)
Last saved: 2/20/17, 6:44:43 PM
1,569 / 209 / 51


Mirror_words.py
Currently Open Documents
Factorial.py
linuxwords.txt
output.txt
Mirror_words.py
Prime_Function.py
File Path : ~/Documents/Youtube/Mirror_words.py
Mirror_words.py
(no symbol selected)
# Function to check whether the input word in in_file is palindrome or not
def is_palindrom(word):
word = ''.join(c.lower() for c in word if c not in string.punctuation)
# For making a case sensitive compare
# -- word != re.search('Ababa', 'ababA',re.IGNORECASE) ---
if word == word[::-1]:
return True
return False
open in_file in read mode
with open('linuxwords.txt', 'r') as in_file:
Reading in_file content line by line and storing in 'words' list after
#ignoring white spaces,\t,\r etc chars in_file using .strip()
lines = in_file.readlines()
words = [line.strip() for line in lines]
# list for holding non-palindrome words
to_write = []
#initializing count
cnt = 0
# checking whether the input word in the 'words' list be a palindrome or not
for word in words:
if not is_palindrom(word) :
# Appending non-palindrome words to 'to_write' list
to_write.append(word[::-1])
else:
cnt += 1
# printing palindrome count
print('there are ' + str(cnt) + ' palindromes!')
# open out_file for write mode
with open('output.txt', 'w') as out_file:
# Iterating through 'to_write' list that has non-palindromes
for word in to_write[:-1]:
# writing non-palindrome(original) word followed by '/' followed by reversed word
#till the end of the 'to_write' list to out_file
out_file.write(word[::-1] + '/' + word + '\n')
out_file.write(to_write[-1])
Recent Documents
Line 41 Col 5
Python
Unicode (UTF-8)
Unix (LF)
Last saved: 2/20/17, 6:44:43 PM
1,569 / 209 / 51

Input file Path:

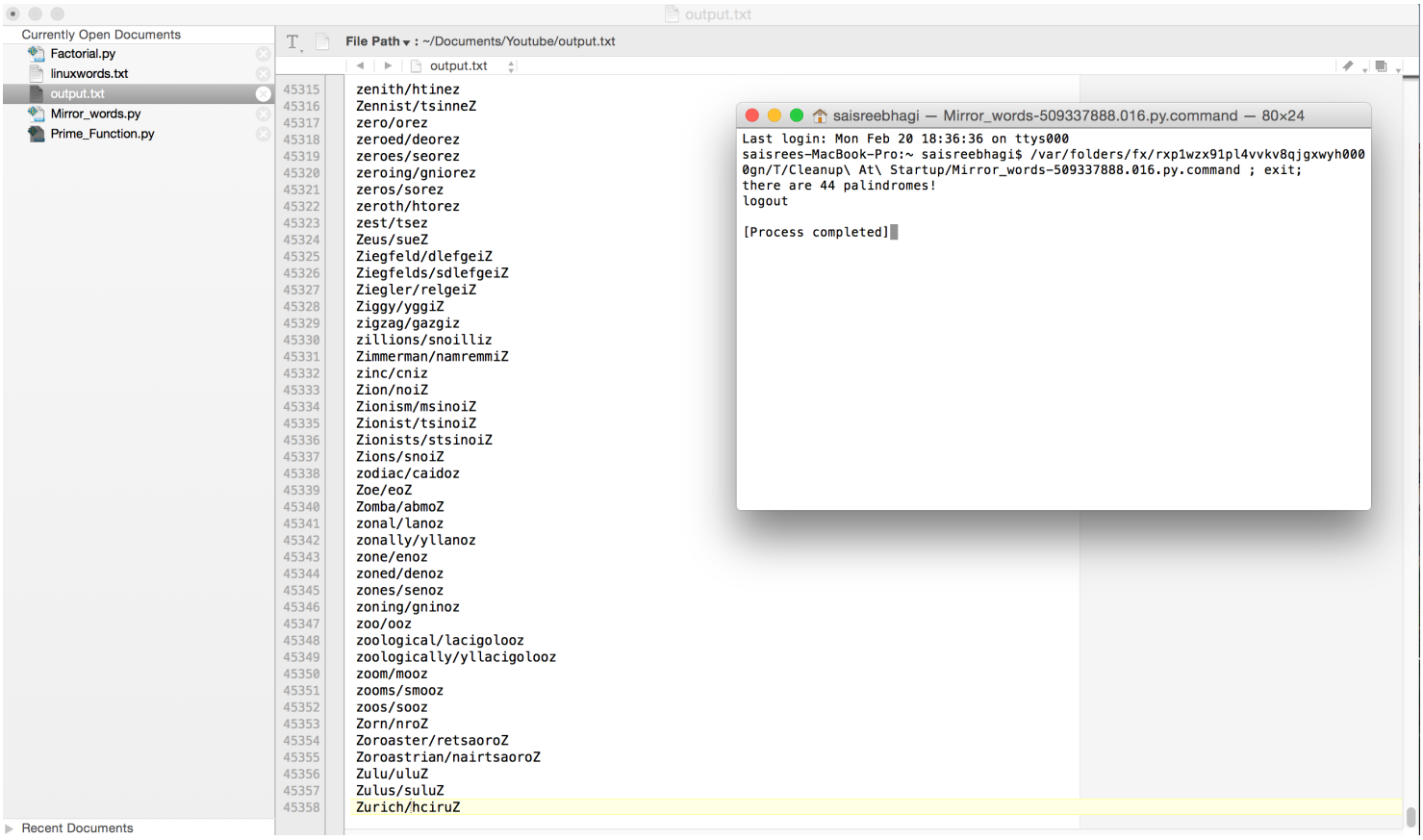
Input file screen:



Output file Path:

Output Screen Shots:

Case when running above find_mirrors(in_file,out_file) function on input file “linuxwords.txt” be shown the output at “output.txt” file in below screens



Future Scope/Alternative:

To improve the performance of above problem in case of No.of reads/writes in files or to reduce no.of loop iterations and for increasing running efficiency of the above algorithm, Guess we can achieve in another way by using ‘Hadoop Map/Reduce’ way