Chapter 9

Case study: word play

This chapter presents the second case study, which involves solving word puzzles by searching for words that have certain properties. For example, we'll find the longest palindromes in English and search for words whose letters appear in alphabetical order. And I will present another program development plan: reduction to a previously solved problem.

9.1 Reading word lists

For the exercises in this chapter we need a list of English words. There are lots of word lists available on the Web, but the one most suitable for our purpose is one of the word lists collected and contributed to the public domain by Grady Ward as part of the Moby lexicon project (see http://wikipedia.org/wiki/Moby_Project). It is a list of 113,809 official crosswords; that is, words that are considered valid in crossword puzzles and other word games. In the Moby collection, the filename is `113809of.fic` ; you can download a copy, with the simpler name `words.txt` , from http://thinkpython2.com/code/words.txt.

This file is in plain text, so you can open it with a text editor, but you can also read it from Python. The built-in function `open` takes the name of the file as a parameter and returns a `file object` you can use to read the file.

```
>>> fin = open('words.txt')
```

`fin` is a common name for a file object used for input. The file object provides several methods for reading, including `readline` , which reads characters from the file until it gets to a newline and returns the result as a string:

```
>>> fin.readline()
'aa\n'
```

The first word in this particular list is "aa", which is a kind of lava. The sequence "\n" represents the newline character that separates this word from the next.

The file object keeps track of where it is in the file, so if you call $readl \in e$ again, you get the next word:

```
>>> fin.readline()
'aah\n'
```

The next word is "aah"', which is a perfectly legitimate word, so stop looking at me like that. Or, if it's the newline character that's bothering you, we can get rid of it with the string method `strip` :

```
>>> line = fin.readline()
>>> word = line.strip()
>>> word
'aahed'
```

You can also use a file object as part of a `for` loop. This program reads `words.txt` and prints each word, one per line:

```
fin = open('words.txt')
for line in fin:
    word = line.strip()
    print(word)
```

In [1]:

```
pwd       # print working directory
```

Out[1]:

```
'e:\\Lectures\\Programming110B\\notebook\\chap09_new'
```

In [2]:

```
ls wor*.*
```

```
 磁碟區 E 中的磁碟是 DATA
 磁碟區序號:  30E7-0E19

 e:\Lectures\Programming110B\notebook\chap09_new 的目錄

2010/12/22  上午 11:54         1,130,523 words.txt
               1 個檔案       1,130,523 位元組
               0 個目錄  1,595,851,096,064 位元組可用
```

In [3]:

```
fin = open('words.txt')
print(fin)
```

```
<_io.TextIOWrapper name='words.txt' mode='r' encoding='cp950'>
```

In [4]:

```
fin.readline()
```

Out[4]:

```
'aa\n'
```

In [5]:

```
fin.readline()
```

Out[5]:

```
'aah\n'
```

In [6]:

```
# 注意：strip()會去掉'\n'

line = fin.readline()
word = line.strip()
print(line)
print(word)
```

```
aahed

aahed
```

In [7]:

```
# print only 10 words
# 注意：計數器的用法

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    count += 1
    print(count, ': ', word)
    if count == 10:
        break
```

```
1 :  aa
2 :  aah
3 :  aahed
4 :  aahing
5 :  aahs
6 :  aal
7 :  aalii
8 :  aaliis
```

```
9 :  aals
10 :  aardvark
```

In [5]:

```
# print 1 word every 10000 words
# 注意：計數器的用法

count = 1
fin = open('words.txt')
for line in fin:
    word = line.strip()

    if count % 1000 == 0:
        print(count, ': ', word)
    count += 1
```

```
1000 :  actuaries
2000 :  airbrushes
3000 :  amidin
4000 :  antipathies
5000 :  armoured
6000 :  auditoriums
7000 :  ballerinas
8000 :  becap
9000 :  beshrewing
10000 :  blabby
11000 :  bongo
12000 :  briards
13000 :  burgher
14000 :  camisados
15000 :  cascaras
16000 :  chalk
17000 :  choker
18000 :  cliental
19000 :  coistrels
20000 :  condylar
21000 :  coppiced
22000 :  cox
23000 :  cubits
24000 :  damned
25000 :  deficit
26000 :  desecrate
27000 :  diluvia
28000 :  disrupts
29000 :  dourines
30000 :  duplicated
31000 :  elegits
32000 :  englishing
33000 :  eryngos
34000 :  exiguous
35000 :  fanum
36000 :  fielder
37000 :  flichter
38000 :  forenoon
39000 :  frowsy
40000 :  gapy
41000 :  gingery
42000 :  gospelers
43000 :  guanacos
44000 :  handiness
45000 :  hegumeny
46000 :  hoe
47000 :  humans
48000 :  imbowered
49000 :  incudal
50000 :  inscribing
51000 :  invertors
52000 :  jessant
53000 :  kefir
54000 :  label
55000 :  lazes
56000 :  limeades
57000 :  loppering
58000 :  mahuangs
59000 :  massacres
60000 :  meriting
```

```
61000 :  minivacation
62000 :  mobile
63000 :  mottles
64000 :  naganas
65000 :  nilghai
66000 :  nuisance
67000 :  omasa
68000 :  outblazing
69000 :  overachievers
70000 :  ovular
71000 :  parfocal
72000 :  peke
73000 :  philanthropists
74000 :  pitiless
75000 :  polkaing
76000 :  prearrangements
77000 :  primos
78000 :  psaltries
79000 :  quantizes
80000 :  rape
81000 :  rechoose
82000 :  reformats
83000 :  remolds
84000 :  resorbed
85000 :  rhyton
86000 :  roughdried
87000 :  salutations
88000 :  schleps
89000 :  seems
90000 :  shark
91000 :  sidewise
92000 :  slat
93000 :  snore
94000 :  spadilles
95000 :  spur
96000 :  sticky
97000 :  stypsises
98000 :  sunwise
99000 :  sync
100000 :  tawdries
101000 :  thegnly
102000 :  tired
103000 :  traceries
104000 :  troiluses
105000 :  typecasting
106000 :  unevenest
107000 :  unsight
108000 :  utopisms
109000 :  vestry
110000 :  waddly
111000 :  weighters
112000 :  winnowers
113000 :  yah
```

In [7]:

```python
# print 1 word every 10000 words
# 注意：計數器的用法

count = 1
fin = open('words.txt')
for line in fin:
    word = line.strip()

    if count % 10000 == 0:
        print(count, ': ', word)
    count += 1

print('count = ', count)
```

```
10000 :  blabby
20000 :  condylar
30000 :  duplicated
40000 :  gapy
50000 :  inscribing
60000 :  meriting
70000 :  ovular
```

```
80000 :  rape
90000 :  shark
100000 :  tawdries
110000 :  waddly
count =  113810
```

9.2 Exercises

There are solutions to these exercises in the next section. You should at least attempt each one before you read the solutions.

Exercise 9.1.

Write a program that reads words.txt and prints only the words with more than 20 characters (not counting whitespace).

Exercise 9.2.

In 1939 Ernest Vincent Wright published a 50,000 word novel called ***Gadsby*** that does not contain the letter "e". Since "e" is the most common letter in English, that's not easy to do.

In fact, it is difficult to construct a solitary thought without using that most common symbol. It is slow going at first, but with caution and hours of training you can gradually gain facility.

All right, I'll stop now.

Write a function called "has_no_e" that returns `True` if the given word doesn't have the letter "e" in it.

Write a program that reads `words.txt` and prints only the words that have no "e". Compute the percentage of words in the list that have no "e".

Exercise 9.3.

Write a function named `avoids` that takes a word and a string of forbidden letters, and that returns `True` if the word doesn't use any of the forbidden letters.

Write a program that prompts the user to enter a string of forbidden letters and then prints the number of words that don't contain any of them. Can you find a combination of 5 forbidden letters that excludes the smallest number of words?

Exercise 9.4.

Write a function named `uses_only` that takes a word and a string of letters, and that returns `True` if the word contains only letters in the list. Can you make a sentence using only the letters `acefhlo` ? Other than "Hoe alfalfa"?

Exercise 9.5.

Write a function named `uses_all` that takes a word and a string of required letters, and that returns `True` if the word uses all the required letters at least once. How many words are there that use all the vowels `aeiou` ? How about `aeiouy` ?

Exercise 9.6.

Write a function called `is_abecedarian` that returns `True` if the letters in a word appear in alphabetical order (double letters are ok).
How many abecedarian words are there?

9.3 Search

All of the exercises in the previous section have something in common; they can be solved with the search pattern we saw in Section 8.6. The simplest example is:

```
def has_no_e(word):
    for letter in word:
        if letter == 'e':
            return False
    return True
```

The `for` loop traverses the characters in `word`. If we find the letter "e", we can immediately return `False`; otherwise we have to go to the next letter. If we exit the loop normally, that means we didn't find an "e", so we return `True`.

You could write this function more concisely using the `in` operator, but I started with this version because it demonstrates the logic of the search pattern.

`avoids` is a more general version of `has_no_e` but it has the same structure:

```
def avoids(word, forbidden):
    for letter in word:
        if letter in forbidden:
            return False
    return True
```

We can return `False` as soon as we find a forbidden letter; if we get to the end of the loop, we return `True`.

`uses_only` is similar except that the sense of the condition is reversed:

```
def uses_only(word, available):
    for letter in word:
        if letter not in available:
            return False
    return True
```

Instead of a list of forbidden letters, we have a list of available letters. If we find a letter in `word` that is not in `available`, we can return `False`.

`uses_all` is similar except that we reverse the role of the word and the string of letters:

```
def uses_all(word, required):
    for letter in required:
        if letter not in word:
            return False
    return True
```

Instead of traversing the letters in `word`, the loop traverses the required letters. If any of the required letters do not appear in the word, we can return `False`.

If you were really thinking like a computer scientist, you would have recognized that `uses_all` was an instance of a previously solved problem, and you would have written:

```
def uses_all(word, required):
    return uses_only(required, word)
```

This is an example of a program development plan called **reduction to a previously solved problem**, which means that you recognize the problem you are working on as an instance of a solved problem and apply an existing solution. (**簡化為先前解決的問題:** 將正在處理的問題視為已解決問題的實例，並應用現有解決方案)

9.4 Looping with indices

I wrote the functions in the previous section with `for` loops because I only needed the characters in the strings; I didn't have to do anything with the indices.

For `is_abecedarian` we have to compare adjacent letters, which is a little tricky with a `for` loop:

```
def is_abecedarian(word):
    previous = word[0]
    for c in word:
        if c < previous:
            return False
        previous = c
    return True
```

An alternative is to use recursion:

```
def is_abecedarian(word):
    if len(word) <= 1:
        return True
    if word[0] > word[1]:
        return False
    return is_abecedarian(word[1:])
```

Another option is to use a `while` loop:

```
def is_abecedarian(word):
    i = 0
    while i < len(word)-1:
        if word[i+1] < word[i]:
            return False
        i = i+1
    return True
```

The loop starts at `i=0` and ends when `i=len(word)-1`. Each time through the loop, it compares the ith character (which you can think of as the current character) to the $i+1$th character (which you can think of as the next).

If the next character is less than (alphabetically before) the current one, then we have discovered a break in the abecedarian trend, and we return `False`.

If we get to the end of the loop without finding a fault, then the word passes the test. To convince yourself that the loop ends correctly, consider an example like 'flossy'. The length of the word is 6, so the last time the loop runs is when `i` is 4, which is the index of the second-to-last character. On the last iteration, it compares the second-to-last character to the last, which is what we want.

Here is a version of `is_palindrome` (see Exercise 6.3) that uses two indices; one starts at the beginning and goes up; the other starts at the end and goes down.

```
def is_palindrome(word):
    i = 0
    j = len(word)-1

    while i<j:
        if word[i] != word[j]:
            return False
        i = i+1
        j = j-1

    return True
```

Or we could reduce to a previously solved problem and write:

```
def is_palindrome(word):
    return is_reverse(word, word)
```

Using `is_reverse` from Section 8.11.

9.5 Debugging

Testing programs is hard. The functions in this chapter are relatively easy to test because you can check the results by hand. Even so, it is somewhere between difficult and impossible to choose a set of words that test for all possible errors.

Taking `has_no_e` as an example, there are two obvious cases to check: words that have an 'e' should return `False`, and words that don't should return `True`. You should have no trouble coming up with one of each.

Within each case, there are some less obvious subcases. Among the words that have an "e", you should test words with an "e" at the beginning, the end, and somewhere in the middle. You should test long words, short words, and very short words, like the empty string. The empty string is an example of a **special case**, which is one of the non-obvious cases where errors often lurk.

In addition to the test cases you generate, you can also test your program with a word list like `words.txt`. By scanning the output, you might be able to catch errors, but be careful: you might catch one kind of error (words that should not be included, but are) and not another (words that should be included, but aren't).

In general, testing can help you find bugs, but it is not easy to generate a good set of test cases, and even if you do, you can't be sure your program is correct. According to a legendary computer scientist:

```
Program testing can be used to show the presence of bugs, but never to
show their absence!

--- Edsger W. Dijkstra
```

9.6 Glossary

- **file object:** A value that represents an open file.
- **reduction to a previously solved problem:** A way of solving a problem by expressing it as an instance of a previously solved problem.
- **special case:** A test case that is atypical or non-obvious (and less likely to be handled correctly).

練習1

完成 Exercise 9.7 至 9.9 的練習

9.7 Exercises

Exercise 9.7

This question is based on a Puzzler that was broadcast on the radio program *Car Talk* (http://www.cartalk.com/content/puzzlers):

```
Give me a word with three consecutive double letters. I'll give you a
couple of words that almost qualify, but don't. For example, the word
committee, c-o-m-m-i-t-t-e-e. It would be great except for the `i' that
sneaks in there. Or Mississippi: M-i-s-s-i-s-s-i-p-p-i. If you could
take out those i's it would work. But there is a word that has three
consecutive pairs of letters and to the best of my knowledge this may
be the only word. Of course there are probably 500 more but I can only
think of one. What is the word?
```

Write a program to find it. Solution: \url{http://thinkpython2.com/code/cartalk1.py}.

Exercise 9.8

Here's another *Car Talk* Puzzler (http://www.cartalk.com/content/puzzlers):

```
I was driving on the highway the other day and I happened to
notice my odometer. Like most odometers, it shows six digits,
in whole miles only. So, if my car had 300,000
miles, for example, I'd see 3-0-0-0-0-0.

Now, what I saw that day was very interesting. I noticed that the
last 4 digits were palindromic; that is, they read the same forward as
backward. For example, 5-4-4-5 is a palindrome, so my odometer
could have read 3-1-5-4-4-5.

One mile later, the last 5 numbers were palindromic. For example, it
could have read 3-6-5-4-5-6.  One mile after that, the middle 4 out of
6 numbers were palindromic.  And you ready for this? One mile later,
all 6 were palindromic!

"The question is, what was on the odometer when I first looked?"
```

Write a Python program that tests all the six-digit numbers and prints any numbers that satisfy these requirements.
Solution: http://thinkpython2.com/code/cartalk2.py.

Exercise 9.9

Here's another *Car Talk* Puzzler you can solve with a search (http://www.cartalk.com/content/puzzlers):

```
Recently I had a visit with my mom and we realized that
the two digits that make up my age when reversed resulted in her
age. For example, if she's 73, I'm 37. We wondered how often this has
happened over the years but we got sidetracked with other topics and
we never came up with an answer.

When I got home I figured out that the digits of our ages have been
reversible six times so far. I also figured out that if we're lucky it
would happen again in a few years, and if we're really lucky it would
happen one more time after that. In other words, it would have
happened 8 times over all. So the question is, how old am I now?
```

Write a Python program that searches for solutions to this Puzzler. Hint: you might find the string method `zfill` useful.

Solution: http://thinkpython2.com/code/cartalk3.py.

In [8]:

```
# 找出不含字母'e'的字

def has_no_e(word):
    for letter in word:
        if letter == 'e':
            return False
    return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if has_no_e(word):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  aa
2 :  aah
3 :  aahing
4 :  aahs
5 :  aal
6 :  aalii
7 :  aaliis
8 :  aals
9 :  aardvark
10 :  aardvarks
```

In [9]:

```
# 找出不含字母'e'的字

def has_no_e(word):
    for letter in word:
        if letter == 'e':
            return False
    return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if has_no_e(word):
        count += 1
        if count % 1000 == 0:
            print(count, ': ', word)
        #if count == 10:
        #    break

print("count = ", count)
```

```
1000 :  allots
2000 :  aristocratic
3000 :  bandana
4000 :  bolls
5000 :  buts
6000 :  cavalryman
7000 :  coagulum
8000 :  cordoba
9000 :  cyst
10000 :  dizzy
11000 :  fantast
12000 :  formants
13000 :  gird
14000 :  gymnast
15000 :  hoods
16000 :  indivisibility
17000 :  jostling
18000 :  lankily
19000 :  lustring
20000 :  mikvah
21000 :  moths
22000 :  nonpagans
23000 :  ostium
```

```
24000 :  parang
25000 :  playa
26000 :  protocol
27000 :  rapid
28000 :  sajou
29000 :  shamus
30000 :  smirking
31000 :  squinch
32000 :  summating
33000 :  thirams
34000 :  triazins
35000 :  uniform
36000 :  virtuosa
37000 :  wolfs
count =  37641
```

In [10]:

```
# 找出不含字母'e'的字

def has_no_e(word):
    if 'e' in word:
        return False
    else:
        return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if has_no_e(word):
        count += 1
        if count % 1000 == 0:
            print(count, ': ', word)
        #if count == 10:
        #    break

print("count = ", count)
```

```
1000 :  allots
2000 :  aristocratic
3000 :  bandana
4000 :  bolls
5000 :  buts
6000 :  cavalryman
7000 :  coagulum
8000 :  cordoba
9000 :  cyst
10000 :  dizzy
11000 :  fantast
12000 :  formants
13000 :  gird
14000 :  gymnast
15000 :  hoods
16000 :  indivisibility
17000 :  jostling
18000 :  lankily
19000 :  lustring
20000 :  mikvah
21000 :  moths
22000 :  nonpagans
23000 :  ostium
24000 :  parang
25000 :  playa
26000 :  protocol
27000 :  rapid
28000 :  sajou
29000 :  shamus
30000 :  smirking
31000 :  squinch
32000 :  summating
33000 :  thirams
34000 :  triazins
35000 :  uniform
36000 :  virtuosa
```

```
37000 :  wolfs
count =  37641
```

Exercise 9.3. Write a function named avoids that takes a word and a string of forbidden letters, and that returns True if the word doesn’t use any of the forbidden letters.

In [11]:

```python
# 避免使用某一個英文字母

def avoids(word, forbidden):
    for letter in word:
        if letter in forbidden:
            return False
    return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if avoids(word, 'e'):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  aa
2 :  aah
3 :  aahing
4 :  aahs
5 :  aal
6 :  aalii
7 :  aaliis
8 :  aals
9 :  aardvark
10 :  aardvarks
```

In [12]:

```python
def avoids(word, forbidden):
    for letter in word:
        if letter in forbidden:
            return False
    return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if avoids(word, 'aeiou'):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break

print("count = ", count)
```

```
1 :  by
2 :  byrl
3 :  byrls
4 :  bys
5 :  crwth
6 :  crwths
7 :  cry
8 :  crypt
9 :  crypts
10 :  cwm
count =  10
```

Exercise 9.4. Write a function named uses_only that takes a word and a string of letters, and that returns True if the word contains only letters in the list. Can you make a sentence using only the letters acefhlo? Other than "Hoe alfalfa?"

In [13]:

```python
# 限制只能使用某些字母
```

```
def uses_only(word, available):
    for letter in word:
        if letter not in available:
            return False
    return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if uses_only(word, 'abn'):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  aa
2 :  aba
3 :  an
4 :  ana
5 :  anna
6 :  ba
7 :  baa
8 :  baba
9 :  ban
10 :  banana
```

Exercise 9.5. Write a function named uses_all that takes a word and a string of required letters, and that returns True if the word uses all the required letters at least once. How many words are there that use all the vowels aeiou? How about aeiouy?

In [14]:

```
# 必要的字母須至少被使用一次

def uses_all(word, required):
    for letter in required:
        if letter not in word:
            return False
    return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if uses_all(word, 'abn'):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  abalone
2 :  abalones
3 :  abandon
4 :  abandoned
5 :  abandoning
6 :  abandonment
7 :  abandonments
8 :  abandons
9 :  abasement
10 :  abasements
```

In [15]:

```
# 必要的字母須至少被使用一次
# 利用已經有的 uses_only()
# 改寫 uses_all()

def uses_only(word, available):
    for letter in word:
        if letter not in available:
            return False
    return True
```

```
def uses_all(word, required):
    return uses_only(required, word)

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if uses_all(word, 'abn'):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  abalone
2 :  abalones
3 :  abandon
4 :  abandoned
5 :  abandoning
6 :  abandonment
7 :  abandonments
8 :  abandons
9 :  abasement
10 :  abasements
```

Exercise 9.6. Write a function called is_abecedarian that returns True if the letters in a word appear in alphabetical order (double letters are ok). How many abecedarian words are there?

In [16]:

```
# abecedarian: 照ABC次序的字
# 使用 for loop

def is_abecedarian(word):
    previous = word[0]
    for c in word:
        if c < previous:
            return False
        pevious = c
    return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if is_abecedarian(word):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  aa
2 :  aah
3 :  aahed
4 :  aahing
5 :  aahs
6 :  aal
7 :  aalii
8 :  aaliis
9 :  aals
10 :  aardvark
```

In [17]:

```
# abecedarian: 照ABC次序的字
# 使用 recursion

def is_abecedarian(word):
    if len(word) <= 1:
        return True
    if word[0] > word[1]:
        return False
    return is_abecedarian(word[1:])

count = 0
fin = open('words.txt')
```

```
for line in fin:
    word = line.strip()
    if is_abecedarian(word):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  aa
2 :  aah
3 :  aahs
4 :  aal
5 :  aals
6 :  aas
7 :  abbe
8 :  abbes
9 :  abbess
10 :  abbey
```

In [19]:

```
# abecedarian: 照ABC次序的字
# 使用 while loop, index

def is_abecedarian(word):
    i = 0
    while i < len(word)-1:
        if word[i+1] < word[i]:
            return False
        i = i+1
    return True

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if is_abecedarian(word):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  aa
2 :  aah
3 :  aahs
4 :  aal
5 :  aals
6 :  aas
7 :  abbe
8 :  abbes
9 :  abbess
10 :  abbey
```

Exercise 6.6. A palindrome is a word that is spelled the same backward and forward, like "noon" and "redivider". Recursively, a word is a palindrome if the first and last letters are the same and the middle is a palindrome.

In [18]:

```
# 順著寫、倒著寫拼法都一樣的字叫做 palindrome (回文)，例如：noon, redivider

def is_palindrome(word):
    i = 0
    j = len(word)-1

    while i<j:
        if word[i] != word[j]:
            return False

        i = i+1
        j = j-1

    return True

count = 0
```

```
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if is_palindrome(word):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  aa
2 :  aba
3 :  aga
4 :  aha
5 :  ala
6 :  alula
7 :  ama
8 :  ana
9 :  anna
10 :  ava
```

In [21]:

```
# 順著寫、倒著寫拼法都一樣的字叫做 palindrome (回文)，例如：noon, redivider

def is_reverse(word1, word2):
    if len(word1) != len(word2):
        return False

    i = 0
    j = len(word2) - 1

    while j > 0:
        if word1[i] != word2[j]:
            return False
        i = i+1
        j = j-1

    return True

def is_palindrome(word):
    return is_reverse(word, word)

count = 0
fin = open('words.txt')
for line in fin:
    word = line.strip()
    if is_palindrome(word):
        count += 1
        print(count, ': ', word)
        if count == 10:
            break
```

```
1 :  aa
2 :  aba
3 :  aga
4 :  aha
5 :  ala
6 :  alula
7 :  ama
8 :  ana
9 :  anna
10 :  ava
```

In [19]:

```
# 要確定程式的運作是否正確，可以用一些例子測試，然後用 print 列印一些關鍵的變數以檢查執行的過程

def is_abecedarian(word):
    previous = word[0]
    for c in word:
        print('previous, c = ', previous, ", ", c)
        if c < previous:
            return False
        previous = c
    return True
```

```
print(is_abecedarian('bob'))
print(is_abecedarian("banana"))
print(is_abecedarian("abbey"))
```

```
previous, c =  b ,  b
previous, c =  b ,  o
previous, c =  o ,  b
False
previous, c =  b ,  b
previous, c =  b ,  a
False
previous, c =  a ,  a
previous, c =  a ,  b
previous, c =  b ,  b
previous, c =  b ,  e
previous, c =  e ,  y
True
```

練習2

1. polyline (折線) 由連接連續 vertices (頂點) 的 segments (線段) 組成，若折線的起點和終點相同且線段沒有交錯，將構成一個封閉的多邊形，例如：polygon.dat。
2. 撰寫Python程式讀取 polygon.dat，計算多邊形的extent $(x_{min}, y_{min}, x_{max}, y_{max})$、重心(center)、周長與面積，重心的定義如下。
3. 程式需盡可能將會重複用到的部分寫成 function。
4. 程式只用到目前(Chapter 1~9)所學到的東西，不可使用 list。
5. 請自行設計一個看起來清楚的輸出資料格式。
6. 畫出程式的流程圖。

$$(X_{center}, Y_{center}) = (\frac{\Sigma_1^n x_i}{n}, \frac{\Sigma_1^n y_i}{n})$$

提示：(1) n 個頂點會構成 $n-1$ 個線段，利用迴圈讀取每一個點的 (x, y) 坐標資料，需記住前一個點的 (x_p, y_p) 坐標才能計算線段的長度；(2)計算多邊形面積可以先將多邊形切割成相鄰兩點與 X 軸的梯形，然後將每一個梯形的面積相加，方法請參考先前的 plot 筆記本。

大致上的處理流程如下：

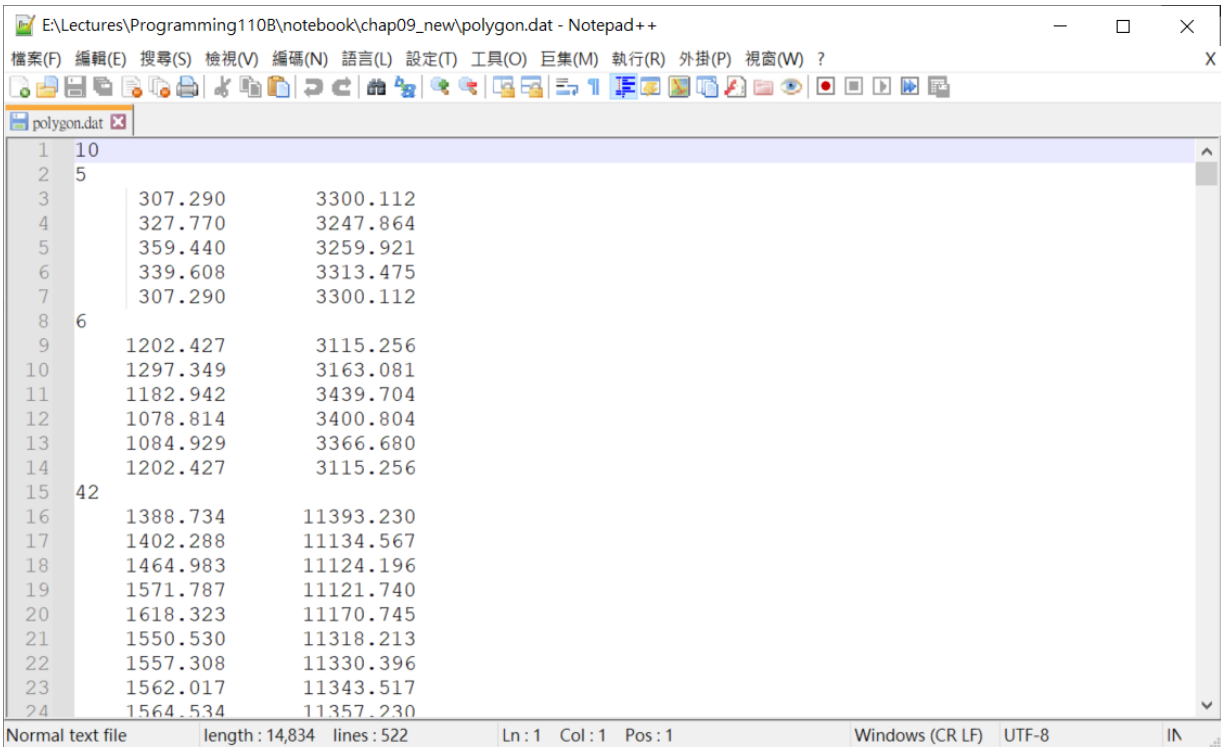
1. 讀進第1個點之 (x, y) 只有一個點還不能計算，先將其指派給 (x_p, y_p)
2. 繼續讀下一個點之 (x, y)，可以由 (x_p, y_p) 和 (x, y) 計算長度 L，和梯形面積 A
3. 將 (x, y) 指派給 (x_p, y_p)
4. 繼續步驟 2~3，過程中將 L 和 A 累加

按照上述流程就可以算出多邊形的周長和面積，3個點、300個點，300萬個點都沒有問題。

若使用list，需儲存所有的點資料，將用到較多的記憶體空間，計算的過程也比較花時間，用上述的做法完全不需使用 list。

polygon.dat 檔案內容：

1. 第一列為多邊形的個數 (number of polygons)
2. 之後為每一個多邊形的頂點個數(number of vertices)，緊接著是每個頂點(vertex)的(x, y)坐標



E:\Lectures\Programming110B\notebook\chap09_new\polygon.dat - Notepad++
檔案(F) 編輯(E) 搜尋(S) 檢視(V) 編碼(N) 語言(L) 設定(T) 工具(O) 巨集(M) 執行(R) 外掛(P) 視窗(W) ?
polygon.dat

```
10
5
     307.290       3300.112
     327.770       3247.864
     359.440       3259.921
     339.608       3313.475
     307.290       3300.112
6
    1202.427       3115.256
    1297.349       3163.081
    1182.942       3439.704
    1078.814       3400.804
    1084.929       3366.680
    1202.427       3115.256
42
    1388.734      11393.230
    1402.288      11134.567
    1464.983      11124.196
    1571.787      11121.740
    1618.323      11170.745
    1550.530      11318.213
    1557.308      11330.396
    1562.017      11343.517
    1564.534      11357.230
```

Normal text file length : 14,834 lines : 522 Ln : 1 Col : 1 Pos : 1 Windows (CR LF) UTF-8 IN

In [ ]: