LECTURE 9: CASE STUDY – WORD PLAY

Hung-Yu Wei

Case Study

- Word list
  - *Solve word puzzle*
  - *Search words that have certain properties*

File processing

- Read from a file
  - *Open a file object*
  - *Syntax*
    - **open** ('file_name')

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

- Read a line
  - *File_object.**readline()***

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

- Strip a line (get rid of the newline character \n)
  - *Line_object.**strip()***

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

Review: Searching in a string

- Search
  - *Find a character in a string*
  - *Return the index of the first matched element*

Section 8.6

```
1  def find(word, letter):
2      index = 0
3      while index < len(word):
4          if word[index] == letter:
5              return index
6          index = index + 1
7      return -1
8
9  fruit_string="banana"
10 print("Find a: ",find(fruit_string, 'a'))
11 print("Find n: ",find(fruit_string, 'n'))
12 print("Find z: ",find(fruit_string, 'z'))
```

```
Find a:  1
Find n:  2
Find z:  -1
```

Summary of examples

1. has_no_e (**word**)
   - *Include "**e**" in word?*
2. avoids (**word**, ***forbidden***)
   - *Avoid "**forbidden**" letters in the word?*
3. use_only (**word**, ***available***)
   - *Only include "**available**" letters in the word?*
4. uses_all (**word**, ***required***)
   - *Use all the "**required**" letters in the word?*
   - *2 versions (**reduction to a previously solved problem**)*
5. is_abecedarian (**word**)
   - *If the letters in the word appear in **alphabetical** order?*
   - *3 versions (**for, recursion, while**)*

照a,b,c 順序

6. is_palindrome (**word**)
   - *A palindrome is a word that is spelled the **same backward and forward** (e.g. "noon", "redivider")*
   - *2 versions (**reduction to a previously solved problem**)*

迴文

Example (1): does not include a specific letter

- Has no “e”

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

Example(2): does not include a set of letters

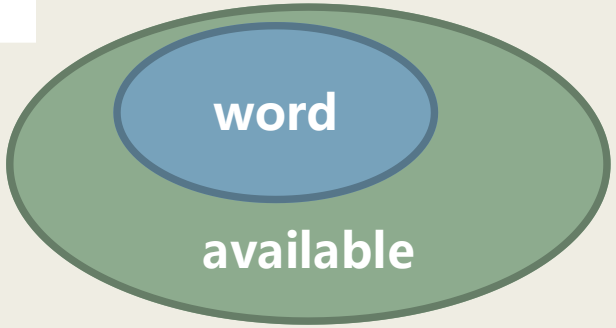
- avoids letters in “forbidden”

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

Example(3): use only a set of letters

- Use only “available”

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



Example(4a): use all letters in a set

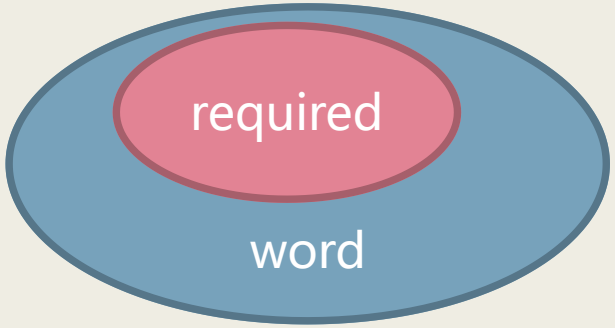
- Use all letters in “required”

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

Example (4b): use all letters in a set

- **Reduction to a previously solved problem**
- Use all letters in “required”
- Re-think Example (3)

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



Example(5a): letters are in alphabetical order

- abecedarian
  - *letters in the word appear in alphabetical order*
- ***for*** loop

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

Example(5b): letters are in alphabetical order

- abecedarian
  - *letters in the word appear in alphabetical order*
- ***recursion***

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

Example(5c): letters are in alphabetical order

- abecedarian
  - *letters in the word appear in alphabetical order*
- ***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
```

Example (6a): palindrome

- A **palindrome** is a word that is spelled the same backward and forward
  - *e.g. "noon", "redivider"*

```
1  def is_palindrome(word):
2      i = 0
3      j = len(word)-1
4
5      while i<j:
6          if word[i] != word[j]:
7              return False
8          i = i+1
9          j = j-1
10
11     return True
12
13 is_palindrome("abba")
```

[Review]Section 8.11: is_reverse

- Is a string the reverse version of the other string?
  - *Compare element-by-element*

```
1  def is_reverse(word1, word2):
2      if len(word1) != len(word2):
3          return False
4      i = 0
5      j = len(word2)-1
6
7      while j >= 0:
8          print(i,j)
9          if word1[i] != word2[j]:
10             return False
11         i = i+1
12         j = j-1
13
14     return True
15
16 answer=is_reverse("pots", "stop")
17 print(answer)
```

```
0 3
1 2
2 1
3 0
True
```

Example (6b): palindrome

- **Reduction to a previously solved problem**

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

Debugging

- Testing
  - *Use examples to test*
  - *Special case (e.g. empty string)*

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

  ~ *Edsger Dijkstra*

Reading

- Chapter 9 in textbook “Think Python”

- Remember upper case and lower case characters are different
  - *A, a*