# Counting Words & Pattern Matching

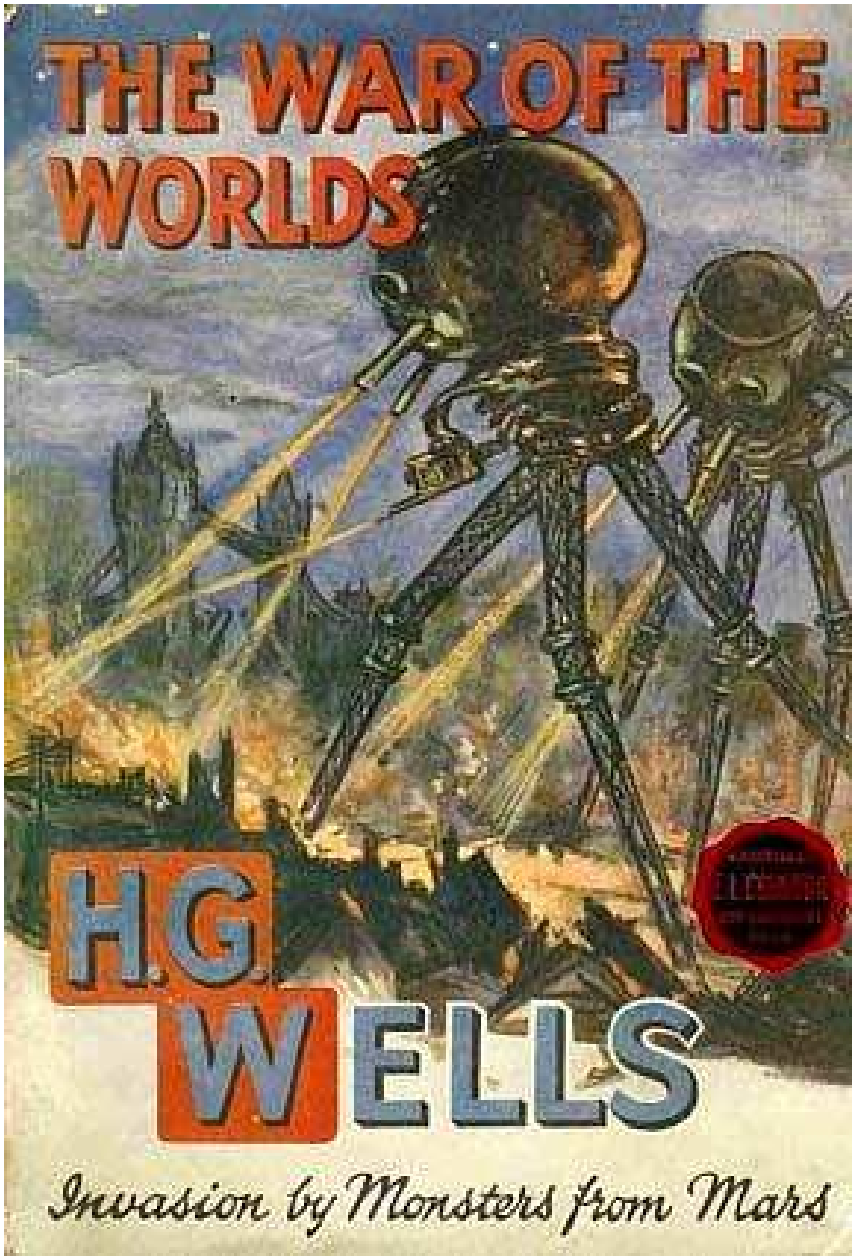
1. Dictionaries
   - free books as .txt files
   - dictionaries in Python
   - sorting dictionary items
2. Pattern Matching
   - using the `re` module
   - matching strings with match()
3. Regular Expressions
   - common regular expression symbols
   - groups of regular expressions

MCS 275 Lecture 7
Programming Tools and File Management
Jan Verschelde, 25 January 2017

# Counting Words
# Pattern Matching

1. Dictionaries
   - free books as .txt files
   - dictionaries in Python
   - sorting dictionary items
2. Pattern Matching
   - using the `re` module
   - matching strings with match()
3. Regular Expressions
   - common regular expression symbols
   - groups of regular expressions

# War of the Worlds




1898 novel

text in public domain

`www.gutenberg.org`

downloaded .txt file

for this course:

word frequencies

"war of the words"

# download the book line by line

```
URL = 'http://www.gutenberg.org/files/36/36.txt'

from urllib.request import urlopen

INFILE = urlopen(URL)
while True:
    LINE = INFILE.readline()
    if LINE == b'':
        print('End of file reached.')
        break
    print(LINE)
    ans = input('Get another line ? ')
    if ans != 'y':
        break
```

# War of the Words

## frequencies of words

Processing a text file:

1. How many different words?
2. We compute the frequency for each word.
3. For each frequency, we list of the words.
4. We list all words used 100 times or more.

To us, anything separated by spaces is a word.

# results of the analysis

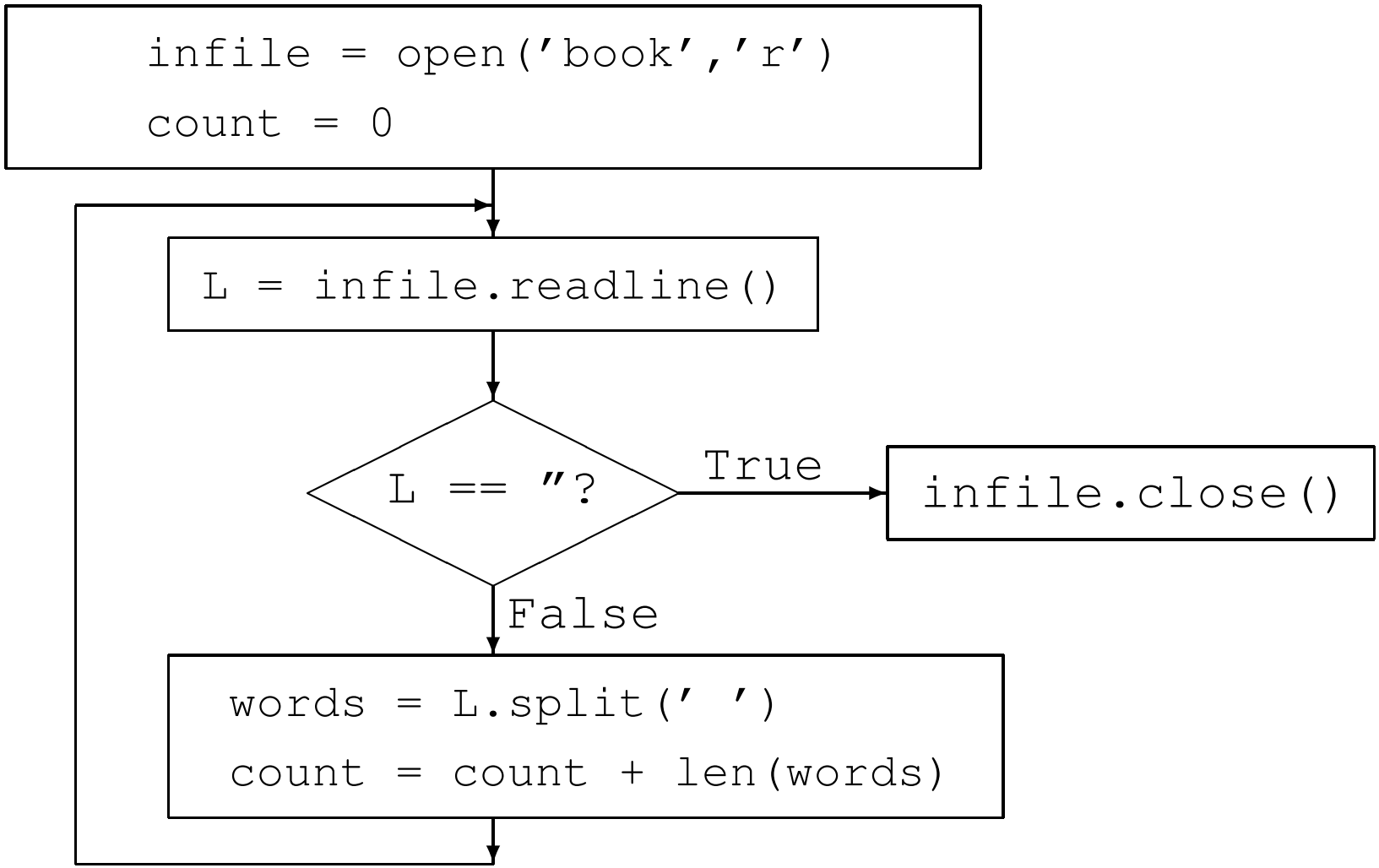
Total words counted: 66,491.

Number of different words: 13,003.

Words used more than 100 times:

```
(4076, ['the'])
...
(119, ['there'])
(116, ['people', 'And'])
(114, ['an'])
(113, ['Martians'])
(111, ['saw', 'through'])
...
```

# the algorithm to count words on file



# scanning files

```
BOOK = "war_of_the_worlds.txt"

def word_count(name):
    """
    Opens the file with name and counts the
    number of words.  Anything that is separated
    by spaces is considered a word.
    """
    file = open(name, 'r')
    count = 0
    while True:
        line = file.readline()
        if line == '':
            break
        words = line.split(' ')
        count = count + len(words)
    file.close()
    return count
```

# Counting Words
# Pattern Matching

1. Dictionaries
   - free books as .txt files
   - dictionaries in Python
   - sorting dictionary items
2. Pattern Matching
   - using the `re` module
   - matching strings with match()
3. Regular Expressions
   - common regular expression symbols
   - groups of regular expressions

# dictionaries in Python

A dictionary in Python is ...

- a set of `key:value` pairs
  any type goes for `value`
  but `key` must belong to an ordered type
- a hash table where order of elements
  allows for fast access.

Our application: frequency table

- type of key: `str`,
- type of value: `int`.

An example of a `key:value` pair:

'the': 4076

# Using Dictionaries

```
>>> D = {}
>>> D['s'] = 1
>>> 's' in D
True
>>> 't' in D
False
>>> D['t'] = 2

>>> D.items()
dict_items([('s', 1), ('t', 2)])
>>> D.values()
dict_values([1, 2])
>>> D.keys()
dict_keys(['s', 't'])
```

# Useful Constructions

| Python | what it means |
|---|---|
| `D = { }` | initialization |
| `D[<key>] = <value>` | add a `key:value` pair |
| `D[<key>]` | selection of value, given key |
| `<key> in D` | returns True or False depending on whether D[<key>] exists |
| `D.items()` | `dict_items` of tuples (key, value) |
| `D.keys()` | returns `dict_keys` of all keys |
| `D.values()` | returns `dict_values` of all values |

# a dictionary of frequencies

```
def word_frequencies(name):
    """
    Returns a dictionary with the frequencies
    of the words occurring on file with name.
    """
    file = open(name, 'r')
    result = {}
    while True:
        line = file.readline()
        if line == '':
            break
        words = line.split(' ')
        for word in words:
            if word in result:
                result[word] += 1
            else:
                result[word] = 1
    file.close()
    return result
```

# composite values

With `freq = word_frequencies('book')`
we see how many times each word occurs,
for example: `freq['the']` returns 4076.

But we want to know the most frequent words,
that is: we want to query `freq` on the values.

We revert the dictionary:

1. the keys are the frequency counts,
2. because words occur more than once,
   the values are lists.

For example: `invfreq[295]` will be `['for', 'from']`,
if `invfreq` is the reverted `freq`.

# frequencies as keys

```
def frequencies_of_words(freq):
    """
    Reverts the keys and values of the
    given dictionary freq.  Because several
    words may occur with the same frequency,
    the values are lists of words.
    """
    result = {}
    for key in freq:
        if freq[key] in result:
            result[freq[key]].append(key)
        else:
            result[freq[key]] = [key]
    return result
```

# Counting Words
# Pattern Matching

1. Dictionaries
   - free books as .txt files
   - dictionaries in Python
   - sorting dictionary items
2. Pattern Matching
   - using the `re` module
   - matching strings with match()
3. Regular Expressions
   - common regular expression symbols
   - groups of regular expressions

# Sorting Dictionary Items

Recall our goal: words used more than 100 times.

The `items()` method on any dictionary returns a list of tuples:

```
>>> L = list(D.items())
>>> L
[('s', 1), ('t', 2)]
```

To sort on a key, from high to low:

```
>>> L.sort(key=lambda i: i[1],reverse=True)
>>> L
[('t', 2), ('s', 1)]
```

# the main program

```
def main():
    """
    Analysis of words in a book.
    """
    cnt = word_count(BOOK)
    print('words counted :', cnt)
    freq = word_frequencies(BOOK)
    print('number of different words :', len(freq))
    invfreq = frequencies_of_words(freq)
    lstfreq = list(invfreq.items())
    lstfreq.sort(key=lambda e: e[0], reverse=True)
    print("words used more than 100 times :")
    for item in lstfreq:
        if item[0] < 100:
            break
        print(item)
```

# Counting Words
# Pattern Matching

1. Dictionaries
   - free books as .txt files
   - dictionaries in Python
   - sorting dictionary items
2. Pattern Matching
   - using the `re` module
   - matching strings with match()
3. Regular Expressions
   - common regular expression symbols
   - groups of regular expressions

# Regular Expressions – using `re`

Manipulating text and strings is an important task:

- parse to ensure entered data is correct,
- search through confidential data: use program.

Suppose `answer` contains the answers
to a yes or no question.

Acceptable *yes* answers:

1. `y` or `yes`
2. `Y` or `Yes`

Testing all these cases is tedious.

Support for **r**egular **e**xpressions:

```
>>> import re
```

`re` is a standard library module.

# Matching short and long Answers

```
>>> import re
>>> (short, long) = ('y', 'Yes')
>>> re.match('y',short) != None
True
>>> re.match('y',long) != None
False
>>> re.match('y|Y',long) != None
True
>>> re.match('y|Y', long)
<_sre.SRE_Match object; span=(0, 1), match='Y'>
>>> re.match('y|Y',long).group()
'Y'
>>> re.match('y|Y',short).group()
'y'
```

# Counting Words
# Pattern Matching

1. Dictionaries
   - free books as .txt files
   - dictionaries in Python
   - sorting dictionary items
2. Pattern Matching
   - using the `re` module
   - matching strings with match()
3. Regular Expressions
   - common regular expression symbols
   - groups of regular expressions

# Matching Strings with `match()`

The function `match()` in the `re` module:

```
>>> re.match( < pattern > , < string > )
```

If the `string` does not match the `pattern`,
then `None` is returned.

If the `string` matches the `pattern`,
then a match object is returned.

```
>>> re.match('he','hello')
<_sre.SRE_Match object at 0x5cb10>
>>> re.match('hi','hello') == None
True
```

# the `group()` method

What can we do with the match object?

```
>>> re.match('he','hello')
<_sre.SRE_Match object at 0x5cb10>
>>> _.group()
'he'
```

After a successful match, `group()` returns that part of the pattern that matches the string.

# searching versus matching

The match only works from the start:

```
>>> re.match('ell','hello') == None
True
```

Looking for the first occurrence of the pattern in the string, with `search()`:

```
>>> re.search('ell','hello')
<_sre.SRE_Match object at 0x5cb10>
>>> _.group()
'ell'
```

# Counting Words
# Pattern Matching

1. Dictionaries
   - free books as .txt files
   - dictionaries in Python
   - sorting dictionary items
2. Pattern Matching
   - using the `re` module
   - matching strings with match()
3. Regular Expressions
   - common regular expression symbols
   - groups of regular expressions

# Regular Expressions

| pattern | strings matched |
|---|---|
| `literal` | strings starting with `literal` |
| `re1\|re2` | strings starting with `re1` or `re2` |

```
>>> from time import ctime
>>> now = ctime()
>>> now
'Wed Jan 27 09:43:50 2016'
>>> p = ...
'\w{3}\s\w{3}\s\d{2}\s\d{2}:\d{2}:\d{2}\s\d{4}'
>>> re.match(p,now) != None
True
```

| pattern | strings matched |
|---|---|
| `\w` | any alphanumeric character, same as `[A-Za-z]` |
| `\d` | any decimal digit, same as `[0-9]` |
| `\s` | any whitespace character |
| `re{`*n*`}` | *n* occurrences of `re` |

# Matching 0 or 1 Occurrences

Allow `Ms.`, `Mr.`, `Mrs.`, with or without the `.` (dot):

```
>>> title = 'Mr?s?\.? '
```

- `?` matches 0 or 1 occurrences
- `.` matches any character
- `\.` matches the dot `.`

```
>>> re.match(title,'Ms ') != None
True
>>> re.match(title,'Ms. ') != None
True
>>> re.match(title,'Miss ') != None
False
>>> re.match(title,'Mr') != None
False
>>> re.match(title,'Mr ') != None
True
>>> re.match(title,'M ') != None
True
```

# character classes

Match with specific characters.

A name has to start with upper case:

```
>>> name = '[A-Z][a-z]*'
>>> G = 'Guido van Rossum'
>>> re.match(name,G)
>>> _.group()
'Guido'

>>> g = 'guido'
>>> re.match(name,g) == None
True
```

# Counting Words
# Pattern Matching

1. Dictionaries
   - free books as .txt files
   - dictionaries in Python
   - sorting dictionary items
2. Pattern Matching
   - using the `re` module
   - matching strings with match()
3. Regular Expressions
   - common regular expression symbols
   - groups of regular expressions

# the `groups()` method

Groups of regular expressions are designated with parenthesis, between `(` and `)`.

Syntax:

```
< pattern > = ( < group1 > ) ( < group2 > )
m = re.match( < pattern > , < string > )
if m != None: m.groups()
```

After a successful match, groups() returns a tuple of those parts of the string that matched the pattern.

# Extracting hours, seconds, minutes

Using the `groups()` method:

```
>>> import re
>>> from time import ctime
>>> now = ctime()
>>> now
'Wed Jan 27 09:51:28 2016'
>>> t = now.split(' ')[3]
>>> t
'09:51:28'
>>> format = '(\d\d):(\d\d):(\d\d)'
>>> m = re.match(format, t)
>>> m.groups()
('09', '51', '28')
>>> (hours, minutes, seconds) = _
```

# Summary and Exercises

Read Chapter 8 of *Python Programming in Context*.

1. Words may contain `\n` or other special symbols. Modify the code to first strip the word of special symbols and to convert to lower case before updating the dictionary.
2. Modify the script `wordswardict.py` to count letters instead of words.
3. Download of 2 different authors 2 different texts from `www.gutenberg.org`. Do the word frequencies tell which texts are written by the same author?
4. Write a regular expression to match all words that start with `a` and end with `t`.
5. Modify `wordswardict.py` so that it prompts the user for a regular expression and then builds a frequency table for those words that match the regular expression.