

Exercises in Programming Style

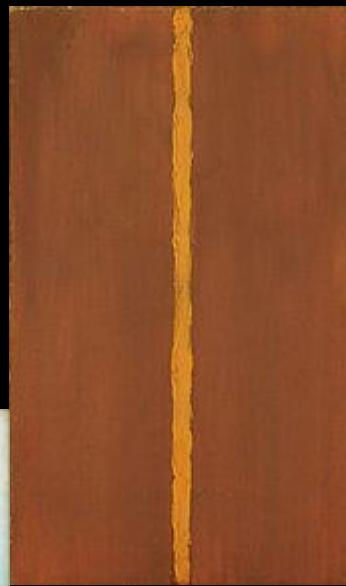
Crista Lopes



modernism



impressionism



abstract expressionism



realism



surrealism



cubism

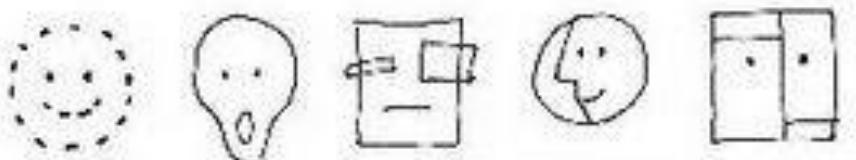


photorealism

Art History, Simplified



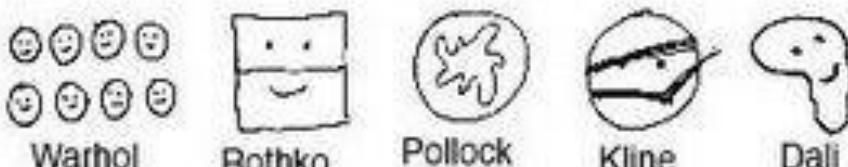
DaVinci El Greco Rembrandt Hals Van Gogh



Seurat Munch Braque Picasso Mondrian



Malevich Gericault Wood Miro Kahlo



Warhol Rothko Pollock Kline Dali



Johns Close Keane Kinkade Mingo

Rules and constraints in software construction

PROGRAMMING STYLES

Programming Styles

- ▷ Ways of expressing tasks
- ▷ Exist at all scales
- ▷ Recur in multiple scales
- ▷ Codified in PLs

Why Are Styles Important?

- ▷ Many
- ▷ Common vocabularies
- ▷ Basic frames of reference
- ▷ Some better than others
 - Depending on many things!

Programming Styles

How do you teach this?

Raymond Queneau



Queneau's Exercises in Style

- ▷ Metaphor
- ▷ Surprises
- ▷ Dream
- ▷ Prognostication
- ▷ Hesitation
- ▷ Precision
- ▷ Negativities
- ▷ Asides
- ▷ Anagrams
- ▷ Logical analysis
- ▷ Past
- ▷ Present
- ▷ ...
- ▷ (99)

Exercises in Programming Style

The story:

Term Frequency

given a text file,
output a list of the 25
most frequently-occurring
non stop, words, ordered by
decreasing frequency

Exercises in Programming Style

The story:

Pride and Prejudice

TF

Term Frequency

given a text file,
output a list of the 25
most frequently-occurring
words, ordered by decreasing
frequency

mr - 786
elizabeth - 635
very - 488
darcy - 418
such - 395
mrs - 343
much - 329
more - 327
bennet - 323
bingley - 306
jane - 295
miss - 283
one - 275
know - 239
before - 229
herself - 227
though - 226
well - 224
never - 220
...

[http://github.com/crista/
exercises-in-programming-style](http://github.com/crista/exercises-in-programming-style)

@cristalopes #style1 *name*

STYLE #1

```
1 import sys, string
2 # the global list of [word, frequency] pairs
3 word_freqs = []
4 # the list of stop words
5 with open('../stop_words.txt') as f:
6     stop_words = f.read().split(',')
7 stop_words.extend(list(string.ascii_lowercase))
8
9 # iterate through the file one line at a time
10 for line in open(sys.argv[1]):
11     start_char = None
12     i = 0
13     for c in line:
14         if start_char == None:
15             if c.isalnum():
16                 # We found the start of a word
17                 start_char = i
18             else:
19                 if not c.isalnum():
20                     # We found the end of a word. Process it
21                     found = False
22                     word = line[start_char:i].lower()
23                     # Ignore stop words
24                     if word not in stop_words:
25                         pair_index = 0
26                         # Let's see if it already exists
27                         for pair in word_freqs:
28                             if word == pair[0]:
29                                 pair[1] += 1
30                                 found = True
31                                 found_at = pair_index
32                                 break
33                         pair_index += 1
34                     if not found:
35                         word_freqs.append([word, 1])
36                     elif len(word_freqs) > 1:
37                         # We may need to reorder
38                         for n in reversed(range(pair_index)):
39                             if word_freqs[pair_index][1] >
40                                 word_freqs[n][1]:
41                                 # swap
42                                 word_freqs[n], word_freqs[
43                                     pair_index] = word_freqs[
44                                         pair_index], word_freqs[n]
45                                 pair_index = n
46                         # Let's reset
47                         start_char = None
48                         i += 1
49
50 for tf in word_freqs[0:25]:
51     print tf[0], ' - ', tf[1]
```

```
# the global list of [word, frequency] pairs
word_freqs = []
# the list of stop words
with open('../stop_words.txt') as f:
    stop_words = f.read().split(',')
stop_words.extend(list(string.ascii_lowercase))
```

```

8 # iterate through the file one line at a time
9 for line in open(sys.argv[1]):
10     start_char = None
11     i = 0
12     for c in line:
13         if start_char == None:
14             if c.isalnum():
15                 # We found the start of a word
16                 start_char = i
17             else:
18                 if not c.isalnum():
19                     # We found the end of a word. Process it
20                     found = False
21                     word = line[start_char:i].lower()
22                     # Ignore stop words
23                     if word not in stop_words:
24                         pair_index = 0
25                         # Let's see if it already exists
26                         for pair in word_freqs:
27                             if word == pair[0]:
28                                 pair[1] += 1
29                                 found = True
30                                 found_at = pair_index
31                                 break
32                         pair_index += 1
33                     if not found:
34                         word_freqs.append([word, 1])
35                 elif len(word_freqs) > 1:
36                     # We may need to reorder
37                     for n in reversed(range(pair_index)):
38                         if word_freqs[pair_index][1] >
39                             word_freqs[n][1]:
40                             # swap
41                             word_freqs[n], word_freqs[
42                                 pair_index] = word_freqs[
43                                     pair_index], word_freqs[n]
44                             pair_index = n
45             # Let's reset

```

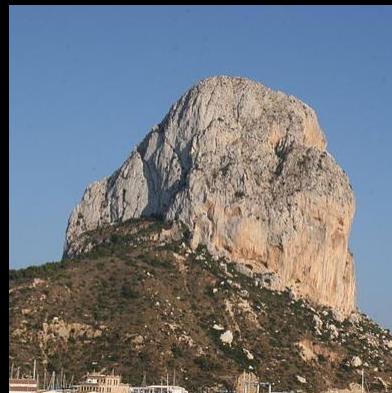
```
1 import sys, string
2 # the global list of [word, frequency] pairs
3 word_freqs = []
4 # the list of stop words
5 with open('../stop_words.txt') as f:
6     stop_words = f.read().split(',')
7 stop_words.extend(list(string.ascii_lowercase))
8
9 for line in open(sys.argv[1]):
10
11     for c in line:
12
13         if start_char == None:
14             if c.isalnum():
15                 # We found the start of a word
16                 start_char = i
17             else:
18                 if not c.isalnum():
19                     # We found the end of a word. Process it
20                     found = False
21                     word = line[start_char:i].lower()
22                     # Ignore stop words
23                     if word not in stop_words:
24                         pair_index = 0
25                         # Let's see if it already exists
26                         for pair in word_freqs:
27                             if word == pair[0]:
28                                 pair[1] += 1
29                                 found = True
30                                 found_at = pair_index
31                                 break
32                         pair_index += 1
33                     if not found:
34                         word_freqs.append([word, 1])
35                     elif len(word_freqs) > 1:
36                         # We may need to reorder
37                         for n in reversed(range(pair_index)):
38                             if word_freqs[pair_index][1] >
39                                 word_freqs[n][1]:
40                                 # swap
41                                 word_freqs[n], word_freqs[
42                                     pair_index] = word_freqs[
43                                         pair_index], word_freqs[n]
44                                 pair_index = n
45                         # Let's reset
46                         start_char = None
47                         i += 1
48
49 for tf in word_freqs[0:25]:
50     print tf[0], ' - ', tf[1]
```

Style #1 Main Characteristics

- ▷ No abstractions
- ▷ No use of libraries

Style #1 Main Characteristics

- ▷ No abstractions
- ▷ No use of libraries



Monolith

Style #1 Main Characteristics

- ▷ No abstractions
- ▷ No use of libraries



Brain-dump Style

@cristalopes #style2 *name*

STYLE #2

```
import re, string, sys

stops = set(open("../stop_words.txt").read().split(",") + list(string.ascii_lowercase))
words = [x.lower() for x in re.split("[^a-zA-Z]+", open(sys.argv[1]).read()) if len(x) > 0 and x.lower() not in stops]
unique_words = list(set(words))
unique_words.sort(lambda x, y: cmp(words.count(y), words.count(x)))
print "\n".join(["%s - %s" % (x, words.count(x)) for x in unique_words[:25]])
```

Credit: *Laurie Tratt*, Kings College London

```
import re, string, sys

stops = set(open("../stop_words.txt").read().split(",") +
            list(string.ascii_lowercase))
words = [x.lower() for x in re.split("[^a-zA-Z]+",
                                         open(sys.argv[1]).read())
         if len(x) > 0 and x.lower() not in stops]
unique_words = list(set(words))
unique_words.sort(lambda x, y: cmp(words.count(y),
                                     words.count(x)))
print "\n".join(["%s - %s" % (x, words.count(x))
                for x in unique_words[:25]])
```

```
import re, string, sys

stops = set(open("../stop_words.txt").read().split(",") +
            list(string.ascii_lowercase))
words = [x.lower() for x in re.split("[^a-zA-Z]+",
                                      open(sys.argv[1]).read())
                     if len(x) > 0 and x.lower() not in stops]
unique_words = list(set(words))
unique_words.sort(lambda x,y:cmp(words.count(y),
                         words.count(x)))

print "\n".join(["%s - %s" % (x, words.count(x))
                  for x in unique_words[:25]])
```

Style #2 Main Characteristics

- ▷ No [named] abstractions
- ▷ Very few [long] lines of code
- ▷ Advanced libraries / constructs

Style #2 Main Characteristics

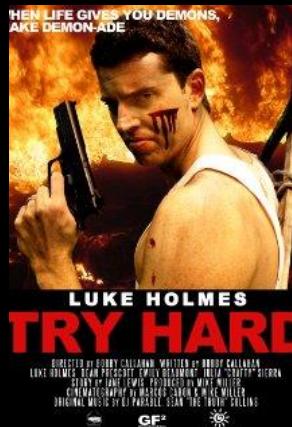
- ▷ No [named] abstractions
- ▷ Very few [long] lines of code
- ▷ Advanced libraries / constructs



Code Golf Style

Style #2 Main Characteristics

- ▷ No [named] abstractions
- ▷ Very few [long] lines of code
- ▷ Advanced libraries / constructs



Try Hard Style

@cristalopes #style3 *name*

STYLE #3

```

1 import sys, string
2
3 # The shared mutable data
4 data = []
5 words = []
6 word_freqs = []
7
8 #
9 # The functions
10 #
11 def read_file(path_to_file):
12     """
13     Takes a path to a file and assigns the entire
14     contents of the file to the global variable data
15     """
16     global data
17     f = open(path_to_file)
18     data = data + list(f.read())
19     f.close()
20
21 def filter_chars_and_normalize():
22     """
23     Replaces all nonalphanumeric chars in data with white space
24     """
25     global data
26     for i in range(len(data)):
27         if not data[i].isalnum():
28             data[i] = ' '
29         else:
30             data[i] = data[i].lower()
31
32 def scan():
33     """
34     Scans data for words, filling the global variable words
35     """
36     global data
37     global words
38     data_str = ''.join(data)
39     words = words + data_str.split()
40
41 def remove_stop_words():
42     global words
43     f = open('../stop_words.txt')
44     stop_words = f.read().split(',')
45     f.close()
46     # add single-letter words
47     stop_words.extend(list(string.ascii_lowercase))
48     indeces = []
49     for i in range(len(words)):
50         if words[i] in stop_words:
51             indeces.append(i)
52     for i in reversed(indeces):
53         words.pop(i)
54

```

```

56 def frequencies():
57     """
58     Creates a list of pairs associating
59     words with frequencies
60     """
61     global words
62     global word_freqs
63     for w in words:
64         keys = [wd[0] for wd in word_freqs]
65         if w in keys:
66             word_freqs[keys.index(w)][1] += 1
67         else:
68             word_freqs.append([w, 1])
69
70 def sort():
71     """
72     Sorts word_freqs by frequency
73     """
74     global word_freqs
75     word_freqs.sort(lambda x, y: cmp(y[1], x[1]))
76
77 #
78 # The main function
79 #
80 read_file(sys.argv[1])
81 filter_chars_and_normalize()
82 scan()
83 remove_stop_words()
84 frequencies()
85 sort()
86
87 for tf in word_freqs[0:25]:
88     print tf[0], ' - ', tf[1]

```

```
data=[]
words=[]
freqs=[]

def read_file(path):
    """
    Takes a path to a file and assigns the entire
    contents of the file to the global variable data
    """
    global data
    f = open(path_to_file)
    data = data + list(f.read())

def filter_normalize():
    """
    Replaces all nonalphanumeric chars in data with white space
    """
    global data
    for i in range(len(data)):
        if not data[i].isalnum():
            data[i] = ' '
        else:
            data[i] = data[i].lower()

def scan():
    """
    Scans data for words, filling the global variable words
    """
    global data
    global words
    data_str = ''.join(data)
    words = words + data_str.split()

def rem_stop_words():
    f = open('../stop_words.txt')
    stop_words = f.read().split(',')
    f.close()
    # add single-letter words
    stop_words.extend(list(string.ascii_lowercase))
    indeces = []
    for i in range(len(words)):
        if words[i] in stop_words:
            indeces.append(i)
    for i in reversed(indeces):
        words.pop(i)
```

```
def frequencies():
    """
    words with frequencies
    """
    global words
    global word_freqs
    for w in words:
        keys = [wd[0] for wd in word_freqs]
        if w in keys:
            word_freqs[keys.index(w)][1] += 1
        else:
            word_freqs.append([w, 1])

def sort():
    """
    Sorts word_freqs by frequency
    """
    global word_freqs
    word_freqs.sort(lambda x, y: cmp(y[1], x[1]))

# Main
# read_file(sys.argv[1])
# filter_normalize()
# scan()
# rem_stop_words()
# frequencies()
# sort()

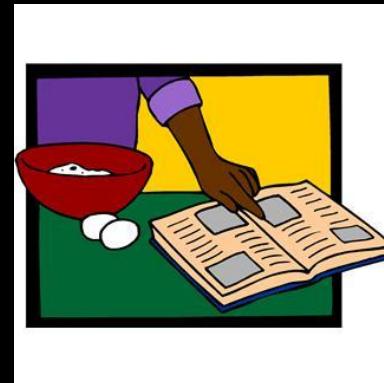
for tf in word_freqs[0:25]:
    print tf[0], ' - ', tf[1]
```

Style #3 Main Characteristics

- ▷ Procedural abstractions
 - maybe input, no output
- ▷ Shared state
- ▷ Commands

Style #3 Main Characteristics

- ▷ Procedural abstractions
 - maybe input, no output
- ▷ Shared state
- ▷ Commands



Cook Book Style

@cristalopes #style4 *name*

STYLE #4

```
1 import sys, re, operator, string
2
3 #
4 # The functions
5 #
6 def read_file(path_to_file):
7     """
8         Takes a path to a file and returns the entire
9         contents of the file as a string
10    """
11    f = open(path_to_file)
12    data = f.read()
13    f.close()
14    return data
15
16 def filter_chars(str_data):
17     """
18         Takes a string and returns a copy with all nonalphanumeric
19         chars replaced by white space
20    """
21    pattern = re.compile('[\W_]+')
22    return pattern.sub(' ', str_data)
23
24 def normalize(str_data):
25     """
26         Takes a string and returns a copy with all chars in lower case
27    """
28    return str_data.lower()
29
30 def scan(str_data):
31     """
32         Takes a string and scans for words, returning
33         a list of words.
34    """
35    return str_data.split()
36
37 def remove_stop_words(word_list):
38     """
39         Takes a list of words and returns a copy with all stop
40         words removed
41    """
42    f = open('../stop_words.txt')
43    stop_words = f.read().split(',')
44    f.close()
45    # add single-letter words
46    stop_words.extend(list(string.ascii_lowercase))
47    return [w for w in word_list if not w in stop_words]
48
49 def frequencies(word_list):
50     """
51         Takes a list of words and returns a dictionary associating
52         words with frequencies of occurrence
53    """
54    word_freqs = {}
```

```
55        for w in word_list:
56            if w in word_freqs:
57                word_freqs[w] += 1
58            else:
59                word_freqs[w] = 1
60        return word_freqs
61
62 def sort(word_freq):
63     """
64         Takes a dictionary of words and their frequencies
65         and returns a list of pairs where the entries are
66         sorted by frequency
67    """
68    return sorted(word_freq.iteritems(), key=operator.itemgetter
69                  (1), reverse=True)
70
71 #
72 # The main function
73 #
74 word_freqs = sort(frequencies(remove_stop_words(scan(normalize(
75             filter_chars(read_file(sys.argv[1]))))))))
76
77 for tf in word_freqs[0:25]:
78     print tf[0], ' - ', tf[1]
```

```
1 import sys, re, operator, string
2
3 #
4 # The functions
5
6 def read_file(path):
7
8     Takes a path to a file and returns the entire
9     contents of the file as a string
10    """
11    f = open(path_to_file)
12
13    return ...
14
15
16 def filter(str_data):
17
18    Takes a string and returns a copy with all nonalphanumeric
19    chars replaced by white space
20    """
21    return ... .e('[\W_]+') + ' ', str_data)
22
23
24 def normalize(str_data):
25
26    Returns a copy with all chars in lower case
27    """
28    return ... er()
29
30
31 def scan(str_data):
32
33    Takes a string and scans for words, returning
34    """
35    return str_data.split()
36
37
38 def rem_stop_words(wordl):
39
40    Takes a list of words and returns a copy with all stop
41    words removed
42    """
43    f = open('../stop_words.txt')
44    stop_words = f.read().split(',')
45    f.close()
46
47    return ... words
48    ist(string.ascii_lowercase)
49    word_list if not w in stop_words]
50
51
52 def frequencies(wordl):
53
54    Takes a list of words and returns a dictionary associating
55    words with frequencies of occurrence
56    """
57    word_freqs = {}
```

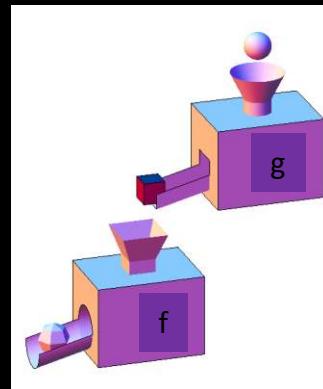
```
55        for w in word_list:
56            if w in word_freqs:
57                word_freqs[w] += 1
58
59        return ... - 1
60
61
62 def sort(word_freqs):
63
64    Takes a list of pairs where the entries are
65    sorted by frequency
66    """
67    return ... eq.iteritems(), key=operator.itemgetter(1))
68
69
70
71 #
72 # Main
73 #
74 wfreqs=st(fq(r(sc(n(fc(rf(sys.argv[1]))))))))
75
76
77 for tf in wfreqs[0:25]:
78     print tf[0], ' - ', tf[1]
```

Style #4 Main Characteristics

- ▷ Function abstractions
 - $f: \text{Input} \rightarrow \text{Output}$
- ▷ No shared state
- ▷ Function composition $f \circ g$

Style #4 Main Characteristics

- ▷ Function abstractions
 - $f: \text{Input} \rightarrow \text{Output}$
- ▷ No shared state
- ▷ Function composition $f \circ g$



Candy Factory Style

@cristalopes #style4 name

Image credit: Nykamp DQ, From *Math Insight*. http://mathinsight.org/image/function_machines_composed

@cristalopes #style5 *name*

STYLE #5

```
1 import sys, re, operator, string
2
3 #
4 # The functions
5 #
6 def read_file(path_to_file, func):
7     """
8         Takes a path to a file and returns the entire
9         contents of the file as a string
10    """
11    f = open(path_to_file)
12    data = f.read()
13    f.close()
14    return func(data, normalize)
15
16 def filter_chars(str_data, func):
17     """
18         Takes a string and returns a copy with all nonalphanumeric
19             chars
20             replaced by white space
21     """
22     pattern = re.compile('[\W_]+')
23     return func(pattern.sub(' ', str_data), scan)
24
25 def normalize(str_data, func):
26     """
27         Takes a string and returns a copy with all characters in lower
28             case
29     """
30     return func(str_data.lower(), remove_stop_words)
31
32 def scan(str_data, func):
33     """
34         Takes a string and scans for words, returning
35             a list of words.
36     """
37     return func(str_data.split(), frequencies)
38
39 def remove_stop_words(word_list, func):
40     """
41         Takes a list of words and returns a copy with all stop
42             words removed """
43     f = open('../stop_words.txt')
44     stop_words = f.read().split(',')
45     f.close()
46     # add single-letter words
47     stop_words.extend(list(string.ascii_lowercase))
48     return func([w for w in word_list if not w in stop_words],
49                 sort)
50
51 def frequencies(word_list, func):
52     """
53         Takes a list of words and returns a dictionary associating
54             words with frequencies of occurrence
55     """
56
```

```
56     word_freqs = {}
57     for w in word_list:
58         if w in word_freqs:
59             word_freqs[w] += 1
60         else:
61             word_freqs[w] = 1
62     return func(word_freqs, no_op)
63
64 def sort(word_freq, func):
65     """
66         Takes a dictionary of words and their frequencies
67         and returns a list of pairs where the entries are
68             sorted by frequency
69     """
70     return func(sorted(word_freq.iteritems(), key=operator.
71                         itemgetter(1), reverse=True), None)
72
73 def no_op(a, func):
74     return a
75
76 # The main function
77 #
78 word_freqs = read_file(sys.argv[1], filter_chars)
79
80 for tf in word_freqs[0:25]:
81     print tf[0], ' - ', tf[1]
```

```

1 import sys, re, operator, string
2
3 #
4 # The functions
5 #
6
7 def read_file(path, func):
8
9     ...
10
11     return func(..., normalize)
12
13
14
15
16 def filter_chars(data, func):
17
18     ...
19
20     return func(..., scan)
21
22
23 def normalize(data, func):
24
25     ...
26
27     return func(..., remove_stops)
28
29
30 def scan(data, func):
31
32     ...
33
34     return func(..., frequencies)
35
36
37 def remove_stops(data, func):
38
39     ...
40
41     return func(..., sort)
42
43
44
45
46 def frequencies(
47     """
48     Takes a list
49     words with f
50     """
51
52     dictionary associating
53     e

```

Etc.

```

51 word_freqs = {}
52 for w in word_list:
53     if w in word_freqs:
54         word_freqs[w] += 1
55     else:
56         word_freqs[w] = 1
57 return func(word_freqs, no_op)
58
59 def sort(word_freq, func):
60     """
61     Takes a dictionary of words and their frequencies
62     and returns a list of pairs where the entries are
63     sorted by frequency
64     """
65     return func(sorted(word_freq.iteritems(), key=operator.
66                     itemgetter(1), reverse=True), None)
67
68 def no_op(a, func):
69     return a
70
71
72 # Main
73 w_freqs=read_file(sys.argv[1],
74                   filter_chars)
75
76
77 for tf in w_freqs[0:25]:
78     print tf[0], ' - ', tf[1]

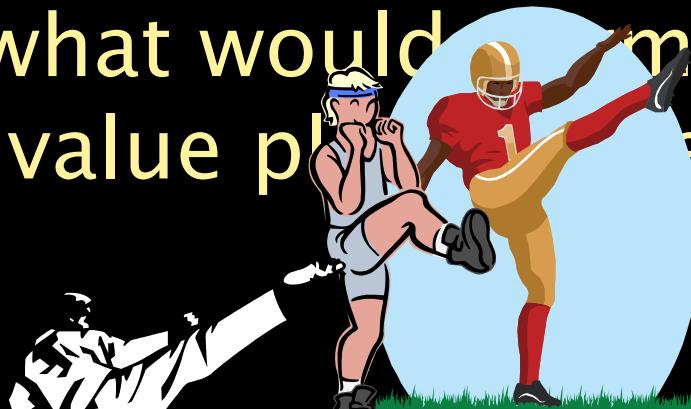
```

Style #5 Main Characteristics

- ▷ Functions take one additional parameter, f
 - called at the end
 - given what would normally be the return value plus the next function

Style #5 Main Characteristics

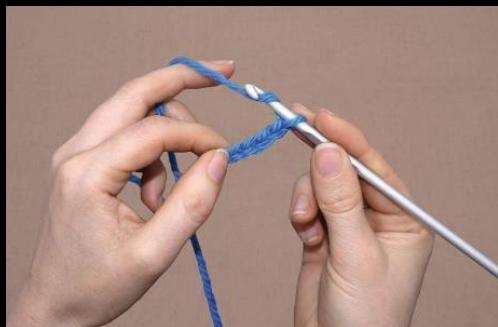
- ▷ Functions take one additional parameter, `f`
 - called at the end
 - given what would normally be the return value plus next function



Kick teammates
@cristalopes #style5 name

Style #5 Main Characteristics

- ▷ Functions take one additional parameter, f
 - called at the end
 - given what would normally be the return value plus the next function



Crochet Style
@cristalopes #style5 name

@cristalopes #style6 *name*

STYLE #6

```

1 import sys, re, operator, string
2 from abc import ABCMeta
3
4 #
5 # The classes
6 #
7 class TFExercise(object):
8     __metaclass__ = ABCMeta
9
10    def info(self):
11        return self.__class__.__name__ + ": No major data
12            structure"
13
14 class DataStorageManager(TFExercise):
15     """ Models the contents of the file """
16     _data = ''
17
18     def __init__(self, path_to_file):
19         f = open(path_to_file)
20         self._data = f.read()
21         f.close()
22         self.__filter_chars()
23         self.__normalize()
24
25     def __filter_chars(self):
26         """
27             Takes a string and returns a copy with all nonalphanumeric
28                 chars
29                 replaced by white space
30             """
31         pattern = re.compile('[\W_]+')
32         self._data = pattern.sub(' ', self._data)
33
34     def __normalize(self):
35         """
36             Takes a string and returns a copy with all characters in
37                 lower case
38             """
39         self._data = self._data.lower()
40
41     def words(self):
42         """
43             Returns the list words in storage
44             """
45         data_str = ''.join(self._data)
46         return data_str.split()
47
48     def info(self):
49         return self.__class__.__name__ + ": My major data
50             structure is a " + self._data.__class__.__name__
51
52 class StopWordManager(TFExercise):
53     """ Models the stop word filter """
54     _stop_words = []
55
56     def __init__(self):

```

```

51         f = open('../stop_words.txt')
52         self._stop_words = f.read().split(',')
53         f.close()
54
55         # add single-letter words
56         self._stop_words.extend(list(string.ascii_lowercase))
57
58     def is_stop_word(self, word):
59         return word in self._stop_words
60
61     def info(self):
62         return self.__class__.__name__ + ": My major data
63             structure is a " + self._stop_words.__class__.__name__
64
65     class WordFrequencyManager(TFExercise):
66         """ Keeps the word frequency data """
67         _word_freqs = {}
68
69         def increment_count(self, word):
70             if word in self._word_freqs:
71                 self._word_freqs[word] += 1
72             else:
73                 self._word_freqs[word] = 1
74
75         def sorted(self):
76             return sorted(self._word_freqs.iteritems(), key=operator.
77                           itemgetter(1), reverse=True)
78
79         def info(self):
80             return self.__class__.__name__ + ": My major data
81                 structure is a " + self._word_freqs.__class__.__name__
82
83     class WordFrequencyController(TFExercise):
84         def __init__(self, path_to_file):
85             self._storage_manager = DataStorageManager(path_to_file)
86             self._stop_word_manager = StopWordManager()
87             self._word_freq_manager = WordFrequencyManager()
88
89         def run(self):
90             for w in self._storage_manager.words():
91                 if not self._stop_word_manager.is_stop_word(w):
92                     self._word_freq_manager.increment_count(w)
93
94             word_freqs = self._word_freq_manager.sorted()
95             for tf in word_freqs[0:25]:
96                 print tf[0], ' - ', tf[1]
97
98     #
99     # The main function
100    #
101    WordFrequencyController(sys.argv[1]).run()

```

```

1 import sys, re, operator, string
2 from abc import ABCMeta
3
4 #
5 # The classes
6
7 class TFExercise():
8
9     def info(self):
10         return self.__class__.__name__ + ": No major data"
11         structure"
12
13 class DataStorageManager(TFExercise):
14
15     _data = ''
16
17     def __init__(self, path_to_file):
18         f = open(path_to_file)
19         self._data = f.read()
20         f.close()
21
22         self.__filter_chars()
23         self.__normalize()
24
25         def __filter_chars(self):
26             """
27                 Takes a string and returns a copy with all nonalphanumeric
28                 chars
29                 replaced by white space
30             """
31
32             pattern = re.compile('[\W_]+')
33             self._data = pattern.sub(' ', self._data)
34
35         def __normalize(self):
36             """
37                 Takes a string and returns a copy with all characters in
38                 lower case
39             """
40
41             self._data = self._data.lower()
42
43         def words(self):
44             """
45                 Returns the list words in storage
46             """
47
48             data_str = ''.join(self._data)
49
50         def info(self):
51             return self.__class__.__name__ + ": My major data"
52             structure is a " + self._stop_words.class_.name_
53
54 class StopWordManager(TFExercise):
55
56     """ Models the stop word filter """
57     _stop_words = []
58
59     def __init__(self):
60
61         f = open('../stop_words.txt')
62         self._stop_words = f.read().split(',')
63         f.close()
64
65         # add single-letter words
66
67         def is_stop_word(self, word):
68             return word in self._stop_words
69
70         def info(self):
71             return self.__class__.__name__ + ": My major data"
72             structure is a " + self._stop_words.class_.name_
73
74 class WordFreqManager(TFExercise):
75
76     word_freqs = {}
77
78         def inc_count(self, word):
79             if word in self._word_freqs:
80                 self._word_freqs[word] += 1
81             else:
82                 self._word_freqs[word] = 1
83
84         def sorted(self):
85             return sorted(self._word_freqs.items(), key=operator.
86                         itemgetter(1), reverse=True)
87
88         def info(self):
89             return self.__class__.__name__ + ": My major data"
90             structure is a " + self._word_freqs.class_.name_
91
92 class WordFreqController(TFExercise):
93
94     def __init__(self, path_to_file):
95         self._storage_manager = DataStorageManager(path_to_file)
96         self._stop_word_manager = StopWordManager()
97         self._word_freq_manager = WordFrequencyManager()
98
99
100        def run(self):
101            words()
102            if not self._stop_word_manager.is_stop_word(w):
103                self._word_freq_manager.increment_count(w)
104
105            word_freqs = self._word_freq_manager.sorted()
106            for tf in word_freqs[0:25]:
107                print tf[0], ' - ', tf[1]
108
109
110 # Main
111 WordFreqController(sys.argv[1]).run()

```

Style #6 Main Characteristics

- ▷ Things, things and more things!
- ▷ Capsules of data and procedures
- ▷ Data is never accessed directly
- ▷ Capsules say “I do the same things as that one, and more!”

Style #6 Main Characteristics

- ▷ Things, things and more things!
- ▷ Capsules of data and procedures
- ▷ Data is never accessed directly
- ▷ Capsules say “I do the same things as that one and more!”



Kingdom of Nouns Style

@cristalopes #style6 *name*

@cristalopes #style7 *name*

STYLE #7

```

1 import sys, re, operator, string
2
3 # Functions for map reduce
4 #
5 def partition(data_str, nlines):
6     """
7         Generator function that partitions the input data_str (a big
8             string)
9             into chunks of nlines.
10            """
11    lines = data_str.split('\n')
12    for i in xrange(0, len(lines), nlines):
13        yield '\n'.join(lines[i:i+nlines])
14
15 def split_words(data_str):
16     """
17         Takes a string, filters non alphanumeric characters,
18             normalizes to
19             lower case, scans for words, and filters the stop words.
20             It returns a list of pairs (word, 1), one for each word in the
21                 input, so
22                 [(w1, 1), (w2, 1), ..., (wn, 1)]
23            """
24
25 def _filter_chars(str_data):
26     """
27         Takes a string and returns a copy with all nonalphanumeric
28             chars
29             replaced by white space
30            """
31    pattern = re.compile('[\W_]+')
32    return pattern.sub(' ', str_data)
33
34 def _normalize(str_data):
35     """
36         Takes a string and returns a copy with all characters in
37             lower case
38            """
39    return str_data.lower()
40
41 def _scan(str_data):
42     """
43         Takes a string and scans for words, returning
44             a list of words.
45            """
46    return str_data.split()
47
48 def _remove_stop_words(word_list):
49    f = open('../stop_words.txt')
50    stop_words = f.read().split(',')
51    f.close()
52    # add single-letter words
53    stop_words.extend(list(string.ascii_lowercase))
54    return [w for w in word_list if not w in stop_words]

```

```

50
51     # The actual work of splitting the input into words
52     result = []
53     words = _remove_stop_words(_scan(_normalize(_filter_chars(
54         data_str))))
55     for w in words:
56         result.append((w, 1))
57
58     return result
59
60 def count_words(pairs_list_1, pairs_list_2):
61     """
62         Takes a two lists of pairs of the form
63             [(w1, 1), ...]
64             and returns a list of pairs [(w1, frequency), ...],
65             where frequency is the sum of all the reported occurrences
66            """
67    mapping = dict((k, v) for k, v in pairs_list_1)
68    for p in pairs_list_2:
69        if p[0] in mapping:
70            mapping[p[0]] += p[1]
71        else:
72            mapping[p[0]] = 1
73
74    return mapping.items()
75
76 # Auxiliary functions
77 #
78
79 def read_file(path_to_file):
80     """
81         Takes a path to a file and returns the entire
82             contents of the file as a string
83            """
84    f = open(path_to_file)
85    data = f.read()
86    f.close()
87    return data
88
89 def sort(word_freq):
90     """
91         Takes a collection of words and their frequencies
92             and returns a collection of pairs where the entries are
93                 sorted by frequency
94            """
95    return sorted(word_freq, key=operator.itemgetter(1), reverse=
96                  True)
97
98 #
99 # The main function
100 #
101 splits = map(split_words, partition(read_file(sys.argv[1]), 200))
102 splits.insert(0, []) # Normalize input to reduce
103 word_freqs = sort(reduce(count_words, splits))

```

```

1 import sys, re, operator, string
2
3 # Functions for map reduce
4 #
5 def partition(data_str, nlines):
6     """
7         Generator function that partitions the input data_str (a big
8             string)
9             into chunks of nlines.
10    """
11    lines = data_str.split('\n')
12    for i in xrange(0, len(lines), nlines):
13        yield '\n'.join(lines[i:i+nlines])
14
15 def split_words(data_str):
16     """
17         Takes a string, filters non alphanumeric characters,
18             normalizes to
19             lower case, scans for words, and filters the stop words.
20             It returns a list of pairs (word, 1), one for each word in the
21                 input, so
22                 [(w1, 1), (w2, 1), ..., (wn, 1)]
23     """
24
25 def _filter_chars(str_data):
26     """
27         Takes a string and returns a copy with all nonalphanumeric
28             chars
29             replaced by white space
30     """
31     pattern = re.compile('[\W_]+')
32     return pattern.sub(' ', str_data)
33
34 def _normalize(str_data):
35     """
36         Takes a string and returns a copy with all characters in
37             lower case
38     """
39     return str_data.lower()
40
41 def _scan(str_data):
42     """
43         Takes a string and scan
44             a list of words.
45     """
46     return str_data.split()
47
48 def _remove_stop_words(word
49 f = open('../stop_words'
50 stop_words = f.read()
51 f.close()
52 # add single-letter wor
53 stop_words.extend(list
54 return [w for w in word
55
56
57 # The actual work of splitting the input into words
58 result = []
59 words = _remove_stop_words(_scan(_normalize(_filter_chars(
60     data_str))))
61 for w in words:
62     result.append((w, 1))
63
64
65 return result
66
67 def count_words(pairs_list_1, pairs_list_2):
68     """
69         Takes a two lists of pairs of the form
70             [(w1, 1), ...]
71         and returns a list of pairs [(w1, frequency), ...],
72             where frequency is the sum of all the reported occurrences
73     """
74     mapping = dict((k, v) for k, v in pairs_list_1)
75     for p in pairs_list_2:
76         if p[0] in mapping:
77             mapping[p[0]] += p[1]
78         else:
79             mapping[p[0]] = 1
80
81     return mapping.items()
82
83
84 # Auxiliary functions
85 #
86
87 def read_file(path_to_file):
88     """
89         Takes a path to a file and returns the entire
90             contents of the file as a string
91     """
92     f = open(path_to_file)
93     data = f.read()
94     f.close()
95
96
97 # Main
98 splits = map(split_words,
99                 partition(read_file(sys.argv[1]), 200))
100 splits.insert(0, [])
101 word_freqs = sort(reduce(count_words, splits))
102
103
104 for tf in word_freqs[0:25]:
105     print tf[0], ' - ', tf[1]

```

```

1 import sys, re, operator, string
2
3 # Functions for map reduce
4 #
5 def partition(data_str, nlines):
6     """
7         Generator function that partitions the input data_str (a big
8             string)
9             into chunks of nlines.
10            """
11    lines = data_str.split('\n')
12    for i in xrange(0, len(lines), nlines):
13        yield '\n'.join(lines[i:i+nlines])

```

def split_words(data_str)

"""

Takes a string (many lines), filters, normalizes to lower case, scans for words, and filters the stop words. Returns a list of pairs (word, 1), so
[(w1, 1), (w2, 1), ..., (wn, 1)]

"""

...

result = []

words = _rem_stop_words(_scan(_normalize(_filter(data_str))))

for w in words:

result.append((w, 1))

return result

```

39 a list of words.
40 """
41     return str_data.split()
42
43 def _remove_stop_words(word_list):
44     f = open('../stop_words.txt')
45     stop_words = f.read().split(',')
46     f.close()
47     # add single-letter words
48     stop_words.extend(list(string.ascii_lowercase))
49     return [w for w in word_list if not w in stop_words]

```

```

50
51     # The actual work of splitting the input into words
52     result = []
53     words = _remove_stop_words(_scan(_normalize(_filter_chars(
54         data_str))))
55     for w in words:
56         result.append((w, 1))
57
58     return result
59
60 def count_words(pairs_list_1, pairs_list_2):
61     """
62         Takes a two lists of pairs of the form
63         [(w1, 1), ...]
64
65         ...
66
67         sorted by frequency
68 """
69     return sorted(word_freq, key=operator.itemgetter(1), reverse=
70                   True)
71
72
73     #
74     # The main function
75     #
76     splits = map(split_words, partition(read_file(sys.argv[1]), 200))
77     splits.insert(0, []) # Normalize input to reduce
78     word_freqs = sort(reduce(count_words, splits))
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

```

```

1 import sys, re, operator, string
2
3 # Functions for map reduce
4 #
5 def partition(data_str, nlines):
6     """
7     Generator function that partitions the input data_str (a big
8     string)
9     into chunks of size nlines.
10    """
11    lines = data_str.splitlines()
12    for i in xrange(0, len(lines), nlines):
13        yield '\r\n'.join(lines[i:i+nlines])
14
15 def split_words(data_str):
16     """
17     Takes a string and splits it into words.
18     Normalizes the words to lower case, strips punctuation and
19     removes stop words.
20     It returns a list of pairs [(w1, 1), (w2, 1), ...].
21     """
22    def _filter_chars(data):
23        """
24        Takes a string and replaces all non-alphanumeric characters
25        by underscores.
26        """
27        pattern = re.compile('[\W_]+')
28        return pattern.sub('_', data)
29
30    def _normalize(data):
31        """
32        Takes a string and converts it to lower case.
33        """
34        return data.lower()
35
36    def _scan(str_data):
37        """
38        Takes a string and returns a list of words.
39        """
40        return str_data.split()
41
42    def _remove_stop_words(word_list):
43        f = open('../stop_words.txt')
44        stop_words = f.read().split(',')
45        f.close()
46        # add single-letter words
47        stop_words.extend(list(string.ascii_lowercase))
48        return [w for w in word_list if not w in stop_words]
49
50
51    # The actual work of splitting the input into words
52    result = []
53    words = _remove_stop_words(_scan(_normalize(_filter_chars(
54        data_str))))
55    for w in words:
56        result.append((w, 1))
57
58    return result
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

```

def count_words(pairs_list_1, pairs_list_2)

"""

Takes two lists of pairs of the form
`[(w1, 1), ...]`

and returns a list of pairs `[(w1, frequency), ...]`,
where frequency is the sum of all occurrences

"""

```

mapping = dict((k, v) for k, v in pairs_list_1)
for p in pairs_list_2:
    if p[0] in mapping:
        mapping[p[0]] += p[1]
    else:
        mapping[p[0]] = 1

return mapping.items()


```

sorted by frequency

"""

```

return sorted(word_freq, key=operator.itemgetter(1), reverse=True)

#
#
# The main function
#
splits = map(split_words, partition(read_file(sys.argv[1]), 200))
splits.insert(0, []) # Normalize input to reduce
word_freqs = sort(reduce(count_words, splits))

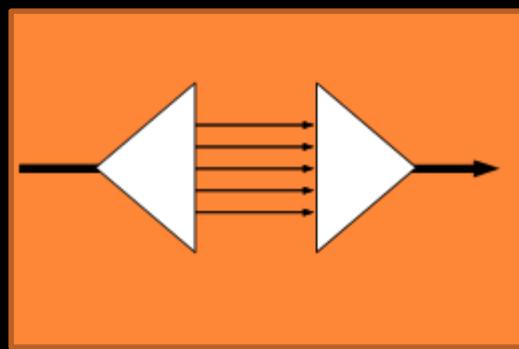

```

Style #7 Main Characteristics

- ▷ Two key abstractions:
`map(f, chunks)` and
`reduce(g, results)`

Style #7 Main Characteristics

- ▷ Two key abstractions:
 $\text{map}(f, \text{chunks})$ and
 $\text{reduce}(g, \text{results})$



iMux Style

@cristalopes #style7 *name*

@cristalopes #style8 *name*

STYLE #8

```

1 import sys, re, string, sqlite3
2
3 #
4 # The relational database of this problem consists of 3 tables:
5 # documents, words, characters
6 #
7 def create_db_schema(connection):
8     c = connection.cursor()
9     c.execute(''CREATE TABLE documents (id INTEGER PRIMARY KEY
10           AUTOINCREMENT, name)''')
11    c.execute(''CREATE TABLE words (id, doc_id, value)'')
12    c.execute(''CREATE TABLE characters (id, word_id, value)'')
13    connection.commit()
14    c.close()
15
16 def load_file_into_database(path_to_file, connection):
17     """ Takes the path to a file and loads the contents into the
18         database """
19     def _read_file(path_to_file):
20         """
21             Takes a path to a file and returns the entire contents of
22                 the
23                 file as a string
24         """
25         f = open(path_to_file)
26         data = f.read()
27         f.close()
28         return data
29
30     def _filter_chars_and_normalize(str_data):
31         """
32             Takes a string and returns a copy with all nonalphanumeric
33                 chars
34                 replaced by white space, and all characters lower-cased
35         """
36         pattern = re.compile('[\W_]+')
37         return pattern.sub(' ', str_data).lower()
38
39     def _scan(str_data):
40         """
41             Takes a string and scans for words, returning a list
42                 of words.
43         """
44         return str_data.split()
45
46     def _remove_stop_words(word_list):
47         f = open('../stop_words.txt')
48         stop_words = f.read().split(',')
49         f.close()
50         # add single-letter words
51         stop_words.extend(list(string.ascii_lowercase))
52         return [w for w in word_list if not w in stop_words]
53
54     # The actual work of splitting the input into words
55     words = _remove_stop_words(_scan(_filter_chars_and_normalize(
56         _read_file(path_to_file))))

```

```

49
50     # Now let's add data to the database
51     # Add the document itself to the database
52     c = connection.cursor()
53     c.execute("INSERT INTO documents (name) VALUES (?)", (
54         path_to_file,))
55     c.execute("SELECT id from documents WHERE name=?", (
56         path_to_file,))
57     doc_id = c.fetchone()[0]
58
59     # Add the words to the database
60     c.execute("SELECT MAX(id) FROM words")
61     row = c.fetchone()
62     word_id = row[0]
63     if word_id == None:
64         word_id = 0
65     for w in words:
66         c.execute("INSERT INTO words VALUES (?, ?, ?)", (word_id,
67             doc_id, w))
68     # Add the characters to the database
69     char_id = 0
70     for char in w:
71         c.execute("INSERT INTO characters VALUES (?, ?, ?)", (
72             char_id, word_id, char))
73         char_id += 1
74     word_id += 1
75     connection.commit()
76     c.close()
77
78     # The main function
79     connection = sqlite3.connect(':memory:')
80     create_db_schema(connection)
81     load_file_into_database(sys.argv[1], connection)
82
83     # Now, let's query
84     c = connection.cursor()
85     c.execute("SELECT value, COUNT(*) as C FROM words GROUP BY value
86           ORDER BY C DESC")
87     for i in range(25):
88         row = c.fetchone()
89         if row != None:
90             print row[0] + ' - ' + str(row[1])
91
92     connection.close()

```

```
1 import sys, re, string, sqlite3
2
3 #
4 # The relational database of this problem consists of 3 tables:
5 # documents, words, characters
6 #
7 def create_db_schema(connection):
8     c = connection.cursor()
9     c.execute(''CREATE TABLE documents (id INTEGER PRIMARY KEY
10           AUTOINCREMENT, name)''')
11    c.execute(''CREATE TABLE words (id, doc_id, value)'')
12    c.execute(''CREATE TABLE characters (id, word_id, value)'')
13    connection.commit()
14    c.close()
15
16 def load_file_into_database(path_to_file, connection):
17     """ Takes the path to a file and loads the contents into the
18         database """
19     def _read_file(path_to_file):
20         """
21             Takes a path to a file and returns the entire contents of
22             the
23             file as a string
24
# Main
connection = sqlite3.connect(':memory:')
create_db_schema(connection)
load_file_into_database(sys.argv[1], connection)

# Now, let's query
c = connection.cursor()
c.execute("SELECT value, COUNT(*) as C FROM words GROUP BY value ORDER BY C DESC")
for i in range(25):
    row = c.fetchone()
    if row != None:
        print row[0] + ' - ' + str(row[1])

connection.close()

49
50     # Now let's add data to the database
51     # Add the document itself to the database
52     c = connection.cursor()
53     c.execute("INSERT INTO documents (name) VALUES (?)", (
54         path_to_file,))
55     c.execute("SELECT id from documents WHERE name=?", (
56         path_to_file,))
57     doc_id = c.fetchone()[0]
58
59     # Add the words to the database
60     c.execute("SELECT MAX(id) FROM words")
61     row = c.fetchone()
62     word_id = row[0]
63     if word_id == None:
64         word_id = 0
65     for w in words:
66         c.execute("INSERT INTO words VALUES (?, ?, ?)", (word_id,
67             doc_id, w))
68     # Add the characters to the database
69     char_id = 0
70     for char in w:
71         c.execute("INSERT INTO characters VALUES (?, ?, ?)", (
72             char_id, word_id, char))
```

```

1 import sys, re, string, sqlite3
2
3 def create_db_schema(connection):
4     c = connection.cursor()
5     c.execute(''CREATE TABLE documents(id PRIMARY KEY AUTOINCREMENT, name)')
6     c.execute(''CREATE TABLE words(id, doc_id, value)')
7     c.execute(''CREATE TABLE characters(id, word_id, value)')
8     connection.commit()
9     c.close()
10
11     def _read_file(path_to_file):
12         """
13             Takes a path to a file and returns the entire contents of
14                 the
15                     file as a string
16         """
17         f = open(path_to_file)
18         data = f.read()
19         f.close()
20         return data
21
22     def _filter_chars_and_normalize(str_data):
23         """
24             Takes a string and returns a copy with all nonalphanumeric
25                 chars
26                     replaced by white space, and all characters lower-cased
27         """
28         pattern = re.compile('[\W_]+')
29         return pattern.sub(' ', str_data).lower()
30
31     def _scan(str_data):
32         """
33             Takes a string and scans for words, returning a list
34                 of words.
35         """
36         return str_data.split()
37
38     def _remove_stop_words(word_list):
39         f = open('../stop_words.txt')
40         stop_words = f.read().split(',')
41         f.close()
42         # add single-letter words
43         stop_words.extend(list(string.ascii_lowercase))
44         return [w for w in word_list if not w in stop_words]
45
46     # The actual work of splitting the input into words
47     words = _remove_stop_words(_scan(_filter_chars_and_normalize(
48         _read_file(path_to_file))))
49
50     # Now let's add data to the database
51     # Add the document itself to the database
52
53     # Add the characters to the database
54     char_id = 0
55     for char in w:
56         c.execute("INSERT INTO characters VALUES (?, ?, ?)", (
57             char_id, word_id, char))
58         char_id += 1
59         word_id += 1
60     connection.commit()
61     c.close()
62
63     #
64     # The main function
65     #
66     connection = sqlite3.connect(':memory:')
67     create_db_schema(connection)
68     load_file_into_database(sys.argv[1], connection)
69
70     # Now, let's query
71     c = connection.cursor()
72     c.execute("SELECT value, COUNT(*) as C FROM words GROUP BY value
73                 ORDER BY C DESC")
74     for i in range(25):
75         row = c.fetchone()
76         if row != None:
77             print row[0] + ' - ' + str(row[1])
78
79     connection.close()

```

```
1 import sys, re, string, sqlite3
2
# Now let's add data to the database
# Add the document itself to the database
c = connection.cursor()
c.execute("INSERT INTO documents (name) VALUES (?)", (path_to_
c.execute("SELECT id from documents WHERE name=?", (path_to_fi
doc_id = c.fetchone()[0]
#
# Add the words to the database
c.execute("SELECT MAX(id) FROM words")
row = c.fetchone()
word_id = row[0]
if word_id == None:
    word_id = 0
for w in words:
    c.execute("INSERT INTO words VALUES (?, ?, ?)", (word_id,
        # Add the characters to the database
        char_id = 0
        for char in w:
            c.execute("INSERT INTO characters VALUES (?, ?, ?)", (
                char_id += 1
                word_id += 1
connection.commit()
c.close()
```

Style #8 Main Characteristics

- ▷ Entities and relations between them
- ▷ Query engine
- ▷ Declarative queries

Style #8 Main Characteristics

- ▷ Entities and relations between them
- ▷ Query engine
 - Declarative queries

Z	Model	1 Gyr	4 Gyr	8 Gyr	12 Gyr	17 Gyr
0.008	V96	6.24	6.63	6.79	6.88	6.97
0.008	grid II	5.78	7.21	7.31	7.43	7.48
0.02	V96	8.32	8.44	8.25	8.22	8.09
0.02	grid II	6.84	8.57	8.57	8.63	8.57
0.05	V96	8.50	8.90	8.34	8.08	7.92
0.05	grid II	7.22	9.92	9.62	9.65	9.63

Tabular Style

@cristalopes #style8 *name*

Exercises in Programming Style*



@cristalopes

github.com/crista/exercises-in-programming-style