

Exercises in Programming Style

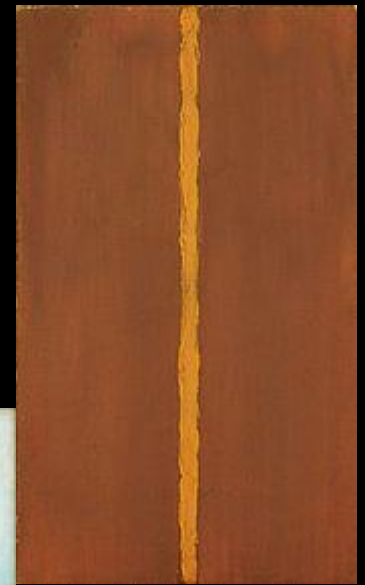
Crista Lopes



modernism



impressionism



abstract expressionism



realism



surrealism



cubism



photorealism

Art History, Simplified



DaVinci



El Greco



Rembrandt



Hals



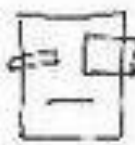
Van Gogh



Seurat



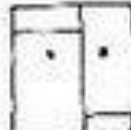
Munch



Braque



Picasso



Mondrian



Malevich



Gericault



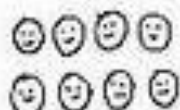
Wood



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
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
```

```

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
10 for line in open(sys.argv[1]):
11
12     for c in line:
13
14         if start_char == None:
15             if c.isalnum():
16                 # We found the start of a word
17                 start_char = i
18
19         else:
20             if not c.isalnum():
21                 # We found the end of a word. Process it
22                 found = False
23                 word = line[start_char:i].lower()
24                 # Ignore stop words
25                 if word not in stop_words:
26                     pair_index = 0
27                     # Let's see if it already exists
28                     for pair in word_freqs:
29                         if word == pair[0]:
30                             pair[1] += 1
31                             found = True
32                             found_at = pair_index
33                             break
34                     pair_index += 1
35                 if not found:
36                     word_freqs.append([word, 1])
37                 elif len(word_freqs) > 1:
38                     # We may need to reorder
39                     for n in reversed(range(pair_index)):
40                         if word_freqs[pair_index][1] >
41                             word_freqs[n][1]:
42                             # swap
43                             word_freqs[n], word_freqs[
44                                 pair_index] = word_freqs[
45                                 pair_index], word_freqs[n]
46                             pair_index = n
47                 # Let's reset
48                 start_char = None
49
50     i += 1
51
52 for tf in word_freqs[0:25]:
53     print tf[0], ' - ', tf[1]

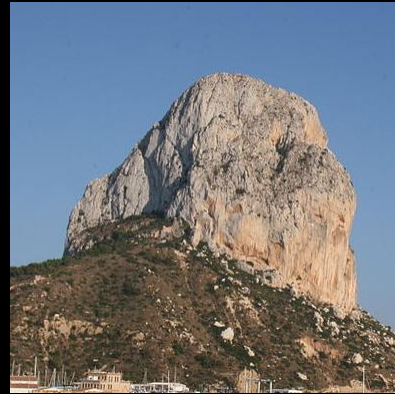
```


Style #1 Main Characteristics

- ▷ No abstractions
- ▷ No use of libraries

Style #1 Main Characteristics

- ▷ No abstractions
- ▷ No use of libraries



Monolith

@cristalopes #style1 *name*

Style #1 Main Characteristics

- ▷ No abstractions
- ▷ No use of libraries



Brain-dump Style

@cristalopes #style1 *name*

@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

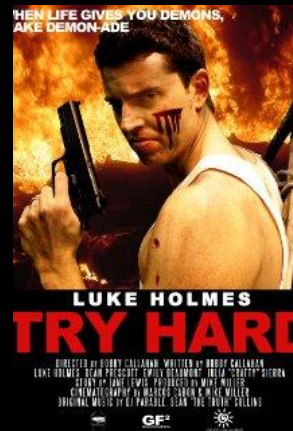
- ▷ No [named] abstractions
- ▷ Very few [long] lines of code
- ▷ Advanced libraries / constructs



Code Golf Style

Style #2 Main Characteristics

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



Try Hard Style

@cristalopes #style2 name

@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

```

```

55 def frequencies():
56     """
57     Creates a list of pairs associating
58     words with frequencies
59     """
60     global words
61     global word_freqs
62     for w in words:
63         keys = [wd[0] for wd in word_freqs]
64         if w in keys:
65             word_freqs[keys.index(w)][1] += 1
66         else:
67             word_freqs.append([w, 1])
68
69 def sort():
70     """
71     Sorts word_freqs by frequency
72     """
73     global word_freqs
74     word_freqs.sort(lambda x, y: cmp(y[1], x[1]))
75
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]

```

```
1 data=[]
2
3
4 words=[]
5
6
7 freqs=[]
8
```

```
1 def read_file(path):
```

```
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())
```

```
1 def filter_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()
```

```
1 def scan():
```

```
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()
```

```
1 def rem_stop_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
```

```
51 def frequencies():
```

```
58 words with frequencies
59 """
60 global words
61 global word_freqs
62 for w in words:
63     keys = [wd[0] for wd in word_freqs]
64     if w in keys:
65         word_freqs[keys.index(w)][1] += 1
66     else:
67         word_freqs.append([w, 1])
```

```
1 def sort():
```

```
71 Sorts word_freqs by frequency
72 """
73 global word_freqs
74 word_freqs.sort(lambda x, y: cmp(y[1], x[1]))
75
76
```

```
#
```

```
# 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 ... re('[\W_]+')
22    str_data)
23
24
25 def normalize(str_data):
26     """
27     Returns a copy with all chars in lower case
28    """
29    return ... er()
30
31
32 def scan(str_data):
33     """
34     Takes a string and scans for words, returning
35    """
36    return ... str_data.split()
37
38
39 def rem_stop_words(wordl):
40     """
41     Takes a list of words and returns a copy with all stop
42     words removed
43    """
44    f = open('../stop_words.txt')
45    stop_words = f.read().split(',')
46    f.close()
47
48    return ... words
49    list(string.ascii_lowercase)
50    word_list if not w in stop_words]
51
52
53 def frequencies(wordl):
54     """
55     Takes a list of words and returns a dictionary associating
56     words with frequencies of occurrence
57    """
58    word_freqs = {}
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
```

```
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 dictionary of word frequencies
65     and returns a list of pairs where the entries are
66     sorted by frequency
67    """
68    return ... req.iteritems(), key=operator.itemgetter
69    (1))
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
```

```
#
# Main
#
wfreqs=st(fq(r(sc(n(fc(rf(sys.argv[1]))))))))

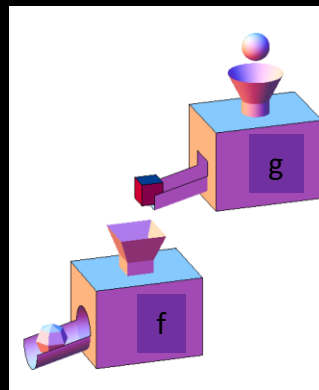
for tf in wfreqs[0:25]:
    print tf[0], ' - ', tf[1]
```

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

@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    """ Takes a list of words and returns a copy with all stop
41    words removed """
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 func([w for w in word_list if not w in stop_words],
48               sort)
49
50 def frequencies(word_list, func):
51    """
52    Takes a list of words and returns a dictionary associating
53    words with frequencies of occurrence
54    """

```

```

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 # The main function
73 #
74 word_freqs = read_file(sys.argv[1], filter_chars)
75
76 for tf in word_freqs[0:25]:
77     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     return func(..., normalize)
```

```
17 def filter_chars(data, func):
18     ...
19     return func(..., scan)
```

```
23 def normalize(data, func):
24     ...
25     return func(..., remove_stops)
```

```
29 def scan(data, func):
30     ...
31     return func(..., frequencies)
```

```
37 def remove_stops(data, func):
38     ...
39     return func(..., sort)
```

```
46 def frequencies(
47     """
48     Takes a list
49     words with f
50     """
```

Etc.

```
dictionary associating
e
"""
```

```
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
```

```
# Main
w_freqs=read_file(sys.argv[1],
                  filter_chars)

for tf in w_freqs[0:25]:
    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

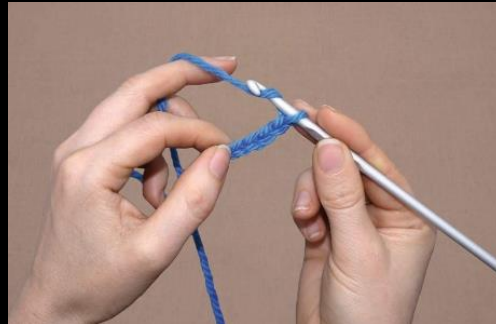


Kick teammates

@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        def __init__(self, path_to_file):
18            f = open(path_to_file)
19            self._data = f.read()
20            f.close()
21            self.__filter_chars()
22            self.__normalize()
23
24        def __filter_chars(self):
25            """
26            Takes a string and returns a copy with all nonalphanumeric
27            chars
28            replaced by white space
29            """
30            pattern = re.compile('[\W_]+')
31            self._data = pattern.sub(' ', self._data)
32
33        def __normalize(self):
34            """
35            Takes a string and returns a copy with all characters in
36            lower case
37            """
38            self._data = self._data.lower()
39
40        def words(self):
41            """
42            Returns the list words in storage
43            """
44            data_str = ''.join(self._data)
45            return data_str.split()
46
47        def info(self):
48            return self.__class__.__name__ + ": My major data
49               structure is a " + self._data.__class__.__name__
50
51    class StopWordManager(TFExercise):
52        """ Models the stop word filter """
53        _stop_words = []
54        def __init__(self):

```

```

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

```

```

1 import sys, re, operator, string
2 from abc import ABCMeta
3
4 #
5 # The classes

```

```
class TFExercise():
```

```

9
10 def info(self):
11     """
12         structure is a " + self.__class__.__name__

```

```
class DataStorageManager(TFExercise):
```

```

15     _data = ''
16     def __init__(self, path_to_file):
17         f = open(path_to_file)
18         self._data = f.read()
19         f.close()
20         self.__filter_chars()
21         self.__normalize()

```

```

22
23     def __filter_chars(self):
24         """
25         Takes a string and returns a copy with all nonalphanumeric
26         chars
27         replaced by white space
28         """
29         pattern = re.compile('[\W_]+')
30         self._data = pattern.sub(' ', self._data)

```

```

31     def __normalize(self):
32         """
33         Takes a string and returns a copy with all characters in
34         lower case
35         """
36         self._data = self._data.lower()

```

```
def words(self):
```

```

37     """
38     Returns the list words in storage
39     """
40     data_str = ''.join(self._data)

```

```
def info(self):
```

```

41     return self.__class__.__name__ + ": My major data

```

```
class StopWordManager(TFExercise):
```

```

48     """ Models the stop word filter """
49     _stop_words = []
50     def __init__(self):

```

```

51     f = open('../stop_words.txt')
52     self._stop_words = f.read().split(',')
53     f.close()
54     # add single-letter words

```

```
def is_stop_word(self, word):
```

```

56     return word in self._stop_words
57
58 def info(self):
59     """
60     structure is a " + self._stop_words.__class__.__name__

```

```
class WordFreqManager(TFExercise):
```

```

65     word_freqs = {}
66
67     def inc_count(self, word):
68         """
69         self.word_freqs[word] += 1
70     else:
71         self.word_freqs[word] = 1

```

```
def sorted(self):
```

```

72     return sorted(self.word_freqs.items(), key=operator.
73         itemgetter(1), reverse=True)
74
75 def info(self):
76     """
77     structure is a " + self.word_freqs.__class__.__name__

```

```
class WordFreqController(TFExercise):
```

```

81     def __init__(self, path_to_file):
82         self._storage_manager = DataStorageManager(path_to_file)
83         self._stop_word_manager = StopWordManager()
84         self._word_freq_manager = WordFrequencyManager()

```

```
def run(self):
```

```

85
86     for w in self._storage_manager.words():
87         if not self._stop_word_manager.is_stop_word(w):
88             self._word_freq_manager.increment_count(w)
89
90     word_freqs = self._word_freq_manager.sorted()
91     for tf in word_freqs[0:25]:
92         print tf[0], ' - ', tf[1]
93
94

```

```

# Main
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 #
4 # Functions for map reduce
5 #
6 def partition(data_str, nlines):
7     """
8     Generator function that partitions the input data_str (a big
9     string)
10    into chunks of nlines.
11    """
12    lines = data_str.split('\n')
13    for i in xrange(0, len(lines), nlines):
14        yield '\n'.join(lines[i:i+nlines])
15
16 def split_words(data_str):
17     """
18     Takes a string, filters non alphanumeric characters,
19     normalizes to
20     lower case, scans for words, and filters the stop words.
21     It returns a list of pairs (word, 1), one for each word in the
22     input, so
23     [(w1, 1), (w2, 1), ..., (wn, 1)]
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]

```

```

55    # The actual work of splitting the input into words
56    result = []
57    words = _remove_stop_words(_scan(_normalize(_filter_chars(
58        data_str))))
59    for w in words:
60        result.append((w, 1))
61
62    return result
63
64 def count_words(pairs_list_1, pairs_list_2):
65     """
66     Takes a two lists of pairs of the form
67     [(w1, 1), ...]
68     and returns a list of pairs [(w1, frequency), ...],
69     where frequency is the sum of all the reported occurrences
70     """
71    mapping = dict((k, v) for k, v in pairs_list_1)
72    for p in pairs_list_2:
73        if p[0] in mapping:
74            mapping[p[0]] += p[1]
75        else:
76            mapping[p[0]] = 1
77
78    return mapping.items()
79
80 #
81 # Auxiliary functions
82 #
83 def read_file(path_to_file):
84     """
85     Takes a path to a file and returns the entire
86     contents of the file as a string
87     """
88    f = open(path_to_file)
89    data = f.read()
90    f.close()
91    return data
92
93 def sort(word_freq):
94     """
95     Takes a collection of words and their frequencies
96     and returns a collection of pairs where the entries are
97     sorted by frequency
98     """
99    return sorted(word_freq, key=operator.itemgetter(1), reverse=
100        True)
101
102 #
103 # The main function
104 #
105 splits = map(split_words, partition(read_file(sys.argv[1]), 200))
106 splits.insert(0, []) # Normalize input to reduce
107 word_freqs = sort(reduce(count_words, splits))

```

```

1 import sys, re, operator, string
2
3 #
4 # Functions for map reduce
5 #
6 def partition(data_str, nlines):
7     """
8     Generator function that partitions the input data_str (a big
9     string)
10    into chunks of nlines.
11    """
12    lines = data_str.split('\n')
13    for i in xrange(0, len(lines), nlines):
14        yield '\n'.join(lines[i:i+nlines])
15
16 def split_words(data_str):
17     """
18     Takes a string, filters non alphanumeric characters,
19     normalizes to
20     lower case, scans for words, and filters the stop words.
21     It returns a list of pairs (word, 1), one for each word in the
22     input, so
23     [(w1, 1), (w2, 1), ..., (wn, 1)]
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
44     a list of words.
45     """
46     return str_data.split()
47
48 def _remove_stop_words(words):
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('a-z'))
54     return [w for w in words if w not in stop_words]

```

```

50 # The actual work of splitting the input into words
51 result = []
52 words = _remove_stop_words(_scan(_normalize(_filter_chars(
53     data_str))))
54 for w in words:
55     result.append((w, 1))
56
57 return result
58
59 def count_words(pairs_list_1, pairs_list_2):
60     """
61     Takes a two lists of pairs of the form
62     [(w1, 1), ...]
63     and returns a list of pairs [(w1, frequency), ...],
64     where frequency is the sum of all the reported occurrences
65     """
66     mapping = dict((k, v) for k, v in pairs_list_1)
67     for p in pairs_list_2:
68         if p[0] in mapping:
69             mapping[p[0]] += p[1]
70         else:
71             mapping[p[0]] = 1
72
73     return mapping.items()
74
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()

```

```

# Main
splits = map(split_words,
              partition(read_file(sys.argv[1]), 200))
splits.insert(0, [])
word_freqs = sort(reduce(count_words, splits))

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

```

```

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

```

```

50 # The actual work of splitting the input into words
51 result = []
52 words = _remove_stop_words(_scan(_normalize(_filter_chars(
53     data_str))))
54 for w in words:
55     result.append((w, 1))
56
57 return result
58
59 def count_words(pairs_list_1, pairs_list_2):
60     """
61     Takes a two lists of pairs of the form
62     [(w1, 1), ...]
63     and returns a list of pairs [(w1, frequency), ...]

```

```

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 # 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]

```

```

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))

```

```

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

```

```

50 # The actual work of splitting the input into words
51 result = []
52 words = _remove_stop_words(_scan(_normalize(_filter_chars(
53     data_str))))
54 for w in words:
55     result.append((w, 1))
56
57 return result
58

```

```
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()
```

```

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

```

```

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))

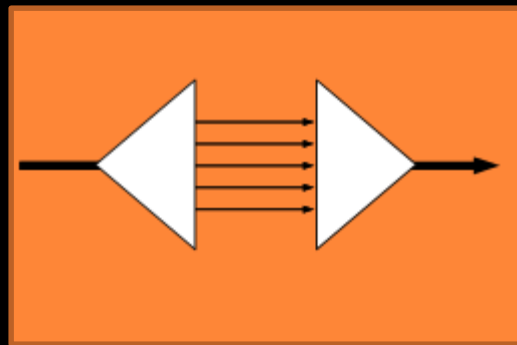
```

Style #7 Main Characteristics

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

Style #7 Main Characteristics

- ▷ Two key abstractions:
map(f, chunks) and
reduce(g, 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 def load_file_into_database(path_to_file, connection):
16     """ Takes the path to a file and loads the contents into the
17         database """
18     def _read_file(path_to_file):
19         """
20         Takes a path to a file and returns the entire contents of
21         the
22         file as a string
23         """
24         f = open(path_to_file)
25         data = f.read()
26         f.close()
27         return data
28     def _filter_chars_and_normalize(str_data):
29         """
30         Takes a string and returns a copy with all nonalphanumeric
31         chars
32         replaced by white space, and all characters lower-cased
33         """
34         pattern = re.compile('[\W_]+')
35         return pattern.sub(' ', str_data).lower()
36     def _scan(str_data):
37         """ Takes a string and scans for words, returning a list
38             of words. """
39         return str_data.split()
40     def _remove_stop_words(word_list):
41         f = open('../stop_words.txt')
42         stop_words = f.read().split(',')
43         f.close()
44         # add single-letter words
45         stop_words.extend(list(string.ascii_lowercase))
46         return [w for w in word_list if not w in stop_words]
47
48 # The actual work of splitting the input into words
49 words = _remove_stop_words(_scan(_filter_chars_and_normalize(
50     _read_file(path_to_file))))

```

```

49 # Now let's add data to the database
50 # Add the document itself to the database
51 c = connection.cursor()
52 c.execute("INSERT INTO documents (name) VALUES (?)", (
53     path_to_file,))
54 c.execute("SELECT id from documents WHERE name=?", (
55     path_to_file,))
56 doc_id = c.fetchone()[0]
57
58 # Add the words to the database
59 c.execute("SELECT MAX(id) FROM words")
60 row = c.fetchone()
61 word_id = row[0]
62 if word_id == None:
63     word_id = 0
64 for w in words:
65     c.execute("INSERT INTO words VALUES (?, ?, ?)", (word_id,
66     doc_id, w))
67 # Add the characters to the database
68 char_id = 0
69 for char in w:
70     c.execute("INSERT INTO characters VALUES (?, ?, ?)", (
71     char_id, word_id, char))
72     char_id += 1
73     word_id += 1
74 connection.commit()
75 c.close()
76
77 #
78 # The main function
79 #
80 connection = sqlite3.connect(':memory:')
81 create_db_schema(connection)
82 load_file_into_database(sys.argv[1], connection)
83
84 # Now, let's query
85 c = connection.cursor()
86 c.execute("SELECT value, COUNT(*) as C FROM words GROUP BY value
87           ORDER BY C DESC")
88 for i in range(25):
89     row = c.fetchone()
90     if row != None:
91         print row[0] + ' - ' + str(row[1])
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 def load_file_into_database(path_to_file, connection):
16     """ Takes the path to a file and loads the contents into the
17         database """
18     def _read_file(path_to_file):
19         """
20         Takes a path to a file and returns the entire contents of
21         the
22         file as a string

```

```

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))

```

```

# 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()

```

```
1 import sys, re, string, sqlite3
```

```
def create_db_schema(connection):  
    c = connection.cursor()  
    c.execute('''CREATE TABLE documents(id PRIMARY KEY AUTOINCREMENT, name)''')  
    c.execute('''CREATE TABLE words(id, doc_id, value)''')  
    c.execute('''CREATE TABLE characters(id, word_id, value)''')  
    connection.commit()  
    c.close()
```

```
17 def _read_file(path_to_file):  
18     """  
19     Takes a path to a file and returns the entire contents of  
20     the  
21     file as a string  
22     """  
23     f = open(path_to_file)  
24     data = f.read()  
25     f.close()  
26     return data  
27  
28 def _filter_chars_and_normalize(str_data):  
29     """  
30     Takes a string and returns a copy with all nonalphanumeric  
31     chars  
32     replaced by white space, and all characters lower-cased  
33     """  
34     pattern = re.compile('[\W_]+')  
35     return pattern.sub(' ', str_data).lower()  
36  
37 def _scan(str_data):  
38     """ Takes a string and scans for words, returning a list  
39     of words. """  
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 # The actual work of splitting the input into words  
51 words = _remove_stop_words(_scan(_filter_chars_and_normalize(  
52     _read_file(path_to_file))))
```

```
49  
50 # Now let's add data to the database  
51 # Add the document itself to the database
```

```
65 # Add the characters to the database  
66 char_id = 0  
67 for char in w:  
68     c.execute("INSERT INTO characters VALUES (?, ?, ?)", (  
69         char_id, word_id, char))  
70     char_id += 1  
71     word_id += 1  
72 connection.commit()  
73 c.close()  
74  
75 #  
76 # The main function  
77 #  
78 connection = sqlite3.connect(':memory:')  
79 create_db_schema(connection)  
80 load_file_into_database(sys.argv[1], connection)  
81  
82 # Now, let's query  
83 c = connection.cursor()  
84 c.execute("SELECT value, COUNT(*) as C FROM words GROUP BY value  
85     ORDER BY C DESC")  
86 for i in range(25):  
87     row = c.fetchone()  
88     if row != None:  
89         print row[0] + ' - ' + str(row[1])  
90  
91 connection.close()
```

```
1 import sys, re, string, sqlite3
```

```
49
```

```
# Now let's add data to the database
```

```
50
```

```
# Add the document itself to the database
```

```
51
```

```
# 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_file,))
c.execute("SELECT id from documents WHERE name=?", (path_to_file,))
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, w, doc_id))
    # Add the characters to the database
    char_id = 0
    for char in w:
        c.execute("INSERT INTO characters VALUES (?, ?, ?)", (word_id, char, doc_id))
        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

*in bookstores Spring 14