

# Computer Programming with Scala

## Week 4: Sorting Bits

Martin Quinson

December 2015



# Introduction

A comprehensive step-by-step guide

Programming in

# Scala

Second Edition



Updated for Scala 2.8

artima

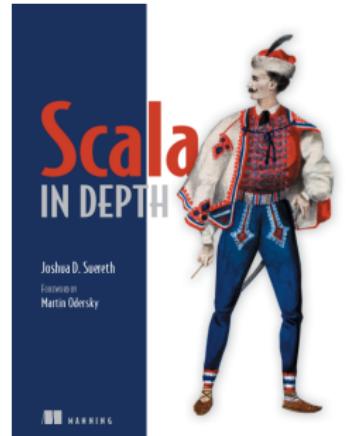
Martin Odersky  
Lex Spoon  
Bill Venners

## Coming next

- ▶ How can you become a mature journalist? (poet)
- ▶ Train your programming skill
  - ▶ Experience comes from exercising
  - ▶ You don't learn swimming or skiing in the books
- ▶ Expertise can be (inefficiently) taught
  - ▶ Best practices can be presented, not sure if you get it
  - ▶ At least, get (exposed to) the vocabulary

## What we saw so far

- ▶ We covered the base book (or almost)
- ▶ You received the language syntax and content
- ▶ Basics, OOP and FP syntax, some stylistic advices
- ▶ You know your letters, some know to read/write



*Experience is what enables you to recognize a mistake when you make it again.*

# Design Patterns

## What

- ▶ Classical patterns that you can find in large programs
  - ▶ Seminal work: common patterns from at least 3 large well organised projects
- ▶ Good programmer: someone that can detect patterns in someone else code
  - ▶ Mean of communication

## Why

- ▶ Best Practice: Reuse the expertise of the ones before us
- ▶ Mean of communication: Put simple words on common concepts

I have this object with some important information and these objects over here need to know when its information changes. These other objects come and go. I'm thinking I should separate out the notification and client registration functionality from the functionality of the object and just let it focus on storing and manipulating its information. Do you agree?

I'm thinking of using the Observer pattern.  
Do you agree?

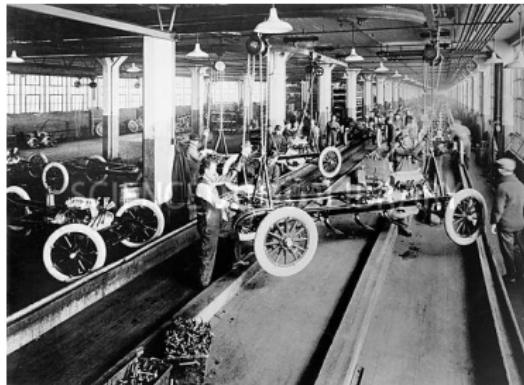
## Why Not

- ▶ Sometimes boilerplates to circumvent language limitations. Bad idea.
- ▶ Lecturing on the 50 Design Patterns is too soporific, let's pick some.

# Factory Pattern

## What

- ▶ Provides an interface for creating an object, encapsulating class instantiation
- ▶ Lets subclasses decide which class to instantiate.



```
trait Animal
private class Dog extends Animal
private class Cat extends Animal
object Animal {
    def apply(kind: String): kind match {
        case "dog" => new Dog()
        case "cat" => new Cat()
    }
}
val d = Animal("dog")
```

## Why? When?

- ▶ Extract complex object creation code
- ▶ Select which class to instantiate
- ▶ Cache objects

## Discussion

- 😊 Resembles constructor invocation
- 😢 Static factory

# Singleton Pattern

## What:

- ▶ Restrict the instantiation to one object
- ▶ Provide a global access point to this object



*The Solitary Tree, Bobbi Jones Jones*

```
object myApp extends App {  
    println("Hello World")  
}
```

## Why? When?

- ▶ Have exactly one object of a class

## Discussion

- ☺ Integrated into the language
- ☺ Java/C++  $\sim$  boilerplate code

# Adapter Pattern

What: Converts an interface into expected interface



Multiple jacks / adaptors, Mihai Andoni

```
trait Log {  
    def warning(message: String)  
    def error(message: String)  
}  
  
final class Logger {  
    def log(lvl: Level, msg: String) { /* ... */ }  
}  
  
implicit class Adapter(logger: Logger) extends Log {  
    def warning(msg: String) { logger.log(WARNING, msg) }  
    def error(msg: String) { logger.log(ERROR, msg) }  
}  
  
val log: Log = new Logger()
```

Why? When?

- ▶ Integrate incompatible classes
- ▶ Adapt existing components

Discussion

- 😊 Clear intent, Concise syntax
- 😢 Implicit behavior easily overseen

# Decorator Pattern

What: Extends functionality of an object (alternative to subclassing)



Colour Pencils, George Hodan

```
trait OutputStream {
    def write(b: Byte)
    def write(b: Array[Byte])
}

class FileOutputStream(path: String) extends OutputStream { }

trait Buffering extends OutputStream {
    abstract override def write(b: Byte) {
        // ...
        super.write(buffer)
    }
}

new FileOutputStream("foo.txt") with Buffering
```

## Why? When?

- ▶ Extend final classes
- ▶ Arbitrarily graph of combination
- ▶ Multiple inheritance

## Discussion

- 😊 Clear intent, Concise syntax
- 😊 Separation of concern
- 😢 Static binding
- 😢 No constructor parameter

# Chain of Responsibility

What: Decouples the sender of a request from its receiver



Things in motion: Colored Domino,  
Sigurd Decroos

```
case class Event(source: String)
type EventHandler = PartialFunction[Event, Unit]
val defaultHandler: EventHandler = PartialFunction(_ =>
  ())
val keyboardHandler: EventHandler = {
  case Event("keyboard") => /* ... */
}
val mouseHandler(delay: Int) : EventHandler = {
  case Event("Mouse") => /* ... */
}
keyboardHandlerorElse mouseHandler(100)orElse defaultH
```

## Why? When?

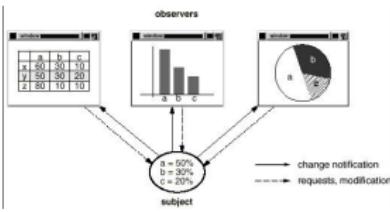
- ▶ Request proposed to the chain until handled
- ▶ Gives more than one object a chance

## Discussion

- 😊 Concise syntax, Built-in logic
- 😢 General-purpose type

# Observer Pattern

**What:** Broadcast object's state changes notifications to dependents



```
trait Observer {  
    def notify(changed: Observable)  
}  
  
trait Observable {  
    private var observers: List[Observer] = Nil  
    def addObserver(o: Observer) = observers ::= o  
    def fireChange = for (o <- observers) o.notify(this)  
}
```

**Why? When?**

- ▶ Set of loosely coupled objects
- ▶ Decouple data Model from View

**Discussion**

- 😊 (rather) concise syntax
- 😢 Control flow complex to follow

# The Pattern Bestiary (1/3)

## Creational Patterns

- ▶ **Abstract Factory:** Factory for building related objects
- ▶ **Builder:** Factory for building complex objects incrementally
- ▶ **Factory Method:** Method in a derived class creates associates
- ▶ **Prototype:** Factory for cloning new instances from a prototype
- ▶ **Singleton:** Factory for a singular (sole) instance

## Structural Patterns

- ▶ **Adapter:** Translator adapts a server interface for a client
- ▶ **Bridge:** Abstraction for binding one of many implementations
- ▶ **Composite:** Structure for building recursive aggregations
- ▶ **Decorator:** Extends an object transparently
- ▶ **Facade:** Simplifies the interface for a subsystem
- ▶ **Flyweight:** Many fine-grained objects shared efficiently.
- ▶ **Proxy:** One object approximates another

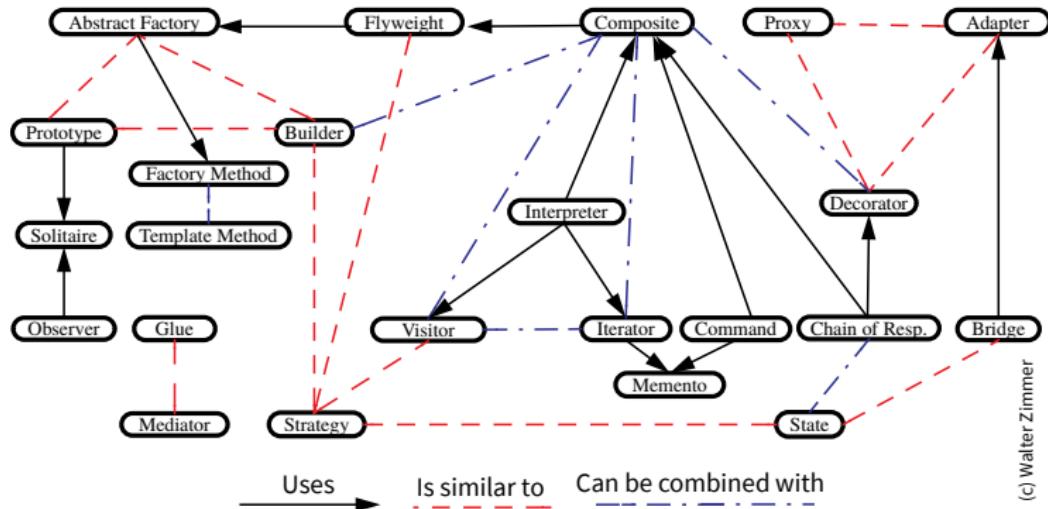
# The Pattern Bestiary (2/3)

## Behavioral Patterns

- ▶ **Chain of Responsibility:** Request delegated to the responsible service provider
- ▶ **Command:** Request or Action is first-class object, hence storable
- ▶ **Iterator:** Aggregate and access elements sequentially
- ▶ **Interpreter:** Language interpreter for a small grammar
- ▶ **Mediator:** Coordinates interactions between its associates
- ▶ **Memento:** Snapshot captures and restores object states privately
- ▶ **Observer:** Dependents update automatically when subject changes
- ▶ **State:** Object whose behavior depends on its state
- ▶ **Strategy:** Abstraction for selecting one of many algorithms
- ▶ **Template Method:** Algorithm with some steps supplied by a derived class
- ▶ **Visitor:** Operations applied to elements of a heterogeneous object structure

## The Pattern Bestiary (3/3)

## Patterns Interactions



## Conclusion on Design Patterns

- ▶ These are only the 23 presented in seminal work. Maaaany more can be found
  - ▶ Learning them is boring, Knowing them helps building on elder's experience  
<https://speakerdeck.com/pavelfatin/design-patterns-in-scala>

<https://speakerdeck.com/pavelfatin/design-patterns-in-scala>

# Anti-Patterns

- ▶ Set of well known bad ideas to common problems
- ▶ **Bicycle shed:** Giving disproportionate weight to trivial issues
- ▶ **Bleeding edge:** Operating with cutting-edge technologies that are still unstable
- ▶ **Overengineering:** Spending time making a project more complex than needed
- ▶ **Big ball of mud:** A system with no recognizable structure
- ▶ **Interface bloat:** Making an interface so powerful that it is difficult to implement
- ▶ **God object:** Concentrating too many functions in a class
- ▶ **Object orgy:** Failed encapsulation permitting unrestricted access to internals
- ▶ **Sequential coupling:** Methods to be called in a particular order
- ▶ **Accidental complexity:** Unneeded complexity due to approach ( $\neq$  inherent)
- ▶ **Hard code:** Spread the constant values everywhere in the code
- ▶ **Repeating yourself:** copy paste considered harmful (DRY SPOT)
- ▶ **Action at a distance, Circular dependency, Premature optimization, Programming by permutation, Reinventing the square wheel, Shotgun surgery**

*Experience is what enables you to recognize a mistake when you make it again*

# Code Smell

## What

- ▶ When you have a bad feeling from reading the code
- ▶ Often surface symptoms of deeper issues
- ▶ Some problems can be automatically detected

## Examples

- ▶ Duplicated code: identical or very similar code exists in more than one location
- ▶ Contrived complexity: forced usage of overly complicated design patterns
- ▶ God class: a class that has grown too large; Lazy class: does too little
- ▶ Cyclomatic complexity: too many branches or loops
- ▶ Spread literals: should be named constants
- ▶ Refused bequest: An overrided method that changes the contract ( $\neq$  Liskov)
- ▶ Excessively short/long identifiers
- ▶ Too many parameters: often denotes an ill-conceived function

[https://en.wikipedia.org/wiki/Code\\_smell](https://en.wikipedia.org/wiki/Code_smell)

<https://en.wikipedia.org/wiki/Anti-pattern>

# Functional Patterns

## OO Pattern

- ▶ Factory Pattern
- ▶ Strategy Pattern
- ▶ Decorator Pattern
- ▶ Visitor Pattern
- ▶ Interface Segregation Principle

## FP Principle

- ▶ Functions
- ▶ Functions
- ▶ Functions
- ▶ Functions
- ▶ Yes, functions

# Functional Patterns

## OO Pattern

- ▶ Factory Pattern
- ▶ Strategy Pattern
- ▶ Decorator Pattern
- ▶ Visitor Pattern
- ▶ Interface Segregation Principle

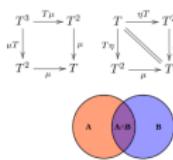
## FP Principle

- ▶ Functions
- ▶ Functions
- ▶ Functions
- ▶ Functions
- ▶ Yes, functions

Design Patterns are OO boilerplate, FP is different

- ▶ (more on that troll: <http://c2.com/cgi/wiki?AreDesignPatternsMissingLanguageFeatures>)
- ▶ Now: quick tour of some FP recurring concepts

- ▶ Steal from Mathematics
- ▶ Types are not classes



- ▶ Functions are things 
- ▶ Composition everywhere 

# FP Core Principle #1: Steal from Mathematics

---

## In Mathematics we Trust

- ▶ Mathematical assertions: unusually precise, general (large class of instances)
- ▶ Mathematics: discipline of reasoning ↠ unusually high confidence level

## Some quotes from E. W. Dijkstra

- ▶ *Programming is one of the most difficult branch of applied mathematics.*
- ▶ *OOP is an exceptionally bad idea which could only have originated in California.*
- ▶ *CS is not more related to Computers than Astronomy to Telescopes.*

## Why are mathematical functions so cool?

# FP Core Principle #1: Steal from Mathematics

---

## In Mathematics we Trust

- ▶ Mathematical assertions: unusually precise, general (large class of instances)
- ▶ Mathematics: discipline of reasoning ↠ unusually high confidence level

## Some quotes from E. W. Dijkstra

- ▶ *Programming is one of the most difficult branch of applied mathematics.*
- ▶ *OOP is an exceptionally bad idea which could only have originated in California.*
- ▶ *CS is not more related to Computers than Astronomy to Telescopes.*

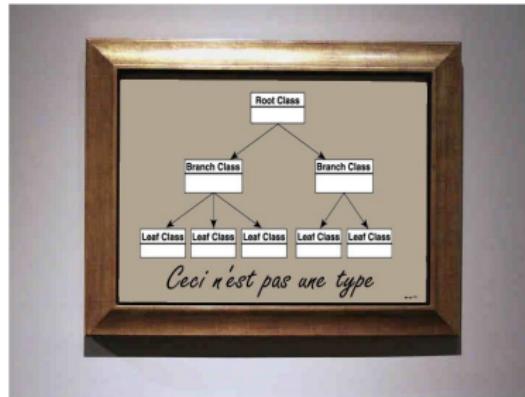
## Why are mathematical functions so cool?

- ▶ They only map a value to another
- ▶ No calculation, just an immutable mapping (no side effect)

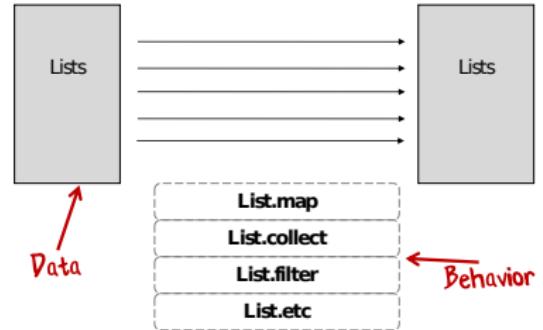
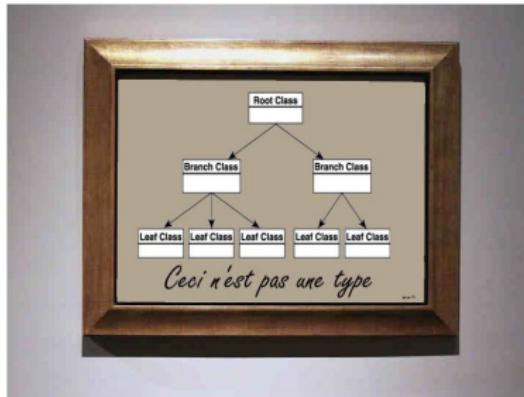
## You should strive for Pure Functions

- ▶ Easy to understand, to reason about
- ▶ Lazily cachable results, Evaluation order does not matter

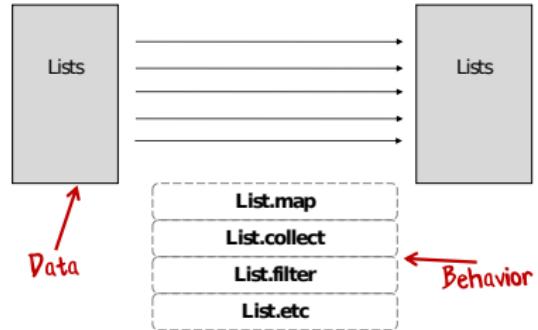
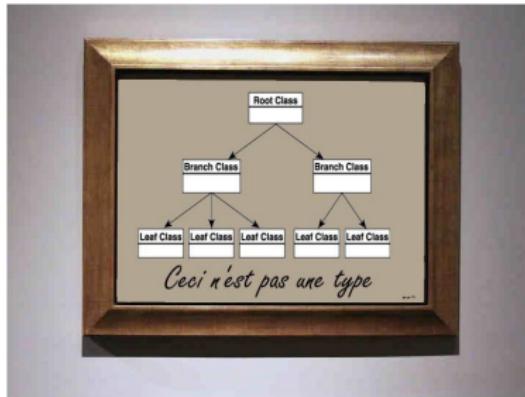
# FP Core Principle #2: Types are not classes



# FP Core Principle #2: Types are not classes



## FP Core Principle #2: Types are not classes



Types are just Data (set of values)  $\neq$  Classes = Data + Behavior

Behavior is within the (pure) functions

# FP Core Principle #3: Functions as things

A function is a standalone thing, not attached to a class

- ▶ You can manipulate functions as any other values
- ▶ Advice: Do as many pure functions as possible; **Avoid mutable objects**
- ▶ Advice: Parametrize all the things (Don't repeat yourself)

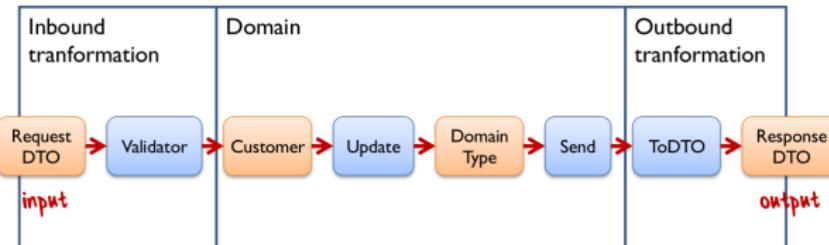
```
public static int Product(int n) {  
    int product = 1;  
    for (int i = 1; i <= n; i++) {  
        product *= i;  
    }  
    return product;  
}  
  
public static int Sum(int n) {  
    int sum = 0;  
    for (int i = 1; i <= n; i++) {  
        sum += i;  
    }  
    return sum;  
}
```

```
def product(n:Int) =  
  (1 to n).foldLeft(1)(_ * _)  
  
def product(n:Int) =  
  (1 /: (1 to n)) (_ * _)  
  
def product(n:Int) =  
  (1 to n).foldLeft(0)(_ + _)  
  
def sum(n:Int)      =  
  (0 /: (1 to n)) (_ + _)
```

# FP Core Principle #4: Composition everywhere

Functions can naturally be composed

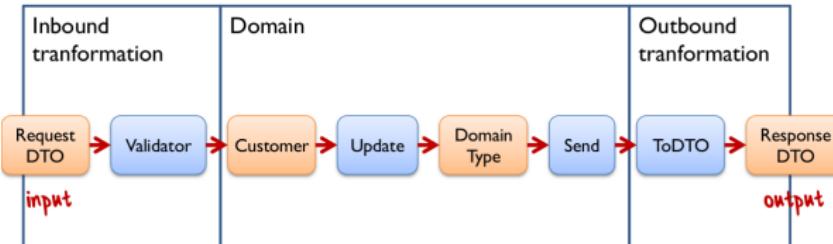
The FP flow is linear



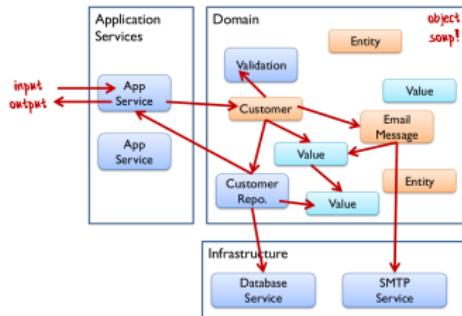
# FP Core Principle #4: Composition everywhere

Functions can naturally be composed

The FP flow is linear



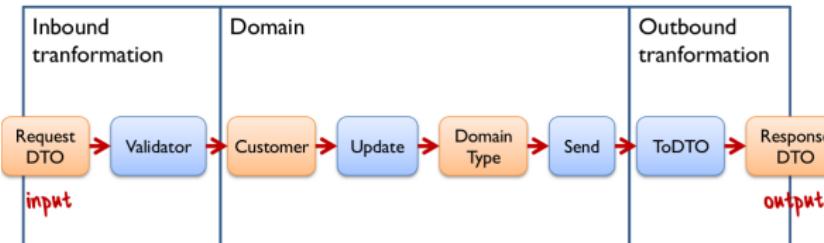
The OO flow soup



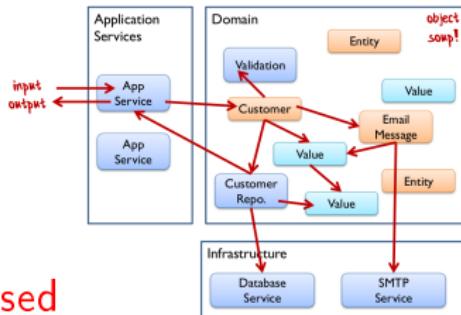
# FP Core Principle #4: Composition everywhere

Functions can naturally be composed

The FP flow is linear



The OO flow soup



Types can also be composed

Product Type

- ▶ Tuple and type alias

```
val p = (1,2) // new Tuple2(1,2)
type Point = (Int, Int)
val p:Point = (1,2)
```

- ▶ Case class

```
case class Point(x: Int, y:Int)
val p = Point(1,2)
```

Sum Type

```
sealed trait Suit
case object Club    extends Suit
case object Diamond extends Suit
case object Spade    extends Suit
case object Heart    extends Suit
```

sealed: no inheritance from other files

FP style guide: <http://fr.slideshare.net/ScottWlaschin/fp-patterns-buildstuffit>

# Painting Style



modernism



impressionism



abstract expressionism



realism



surrealism



cubism



photorealism

From: [http://gotocon.com/dl/goto-aar-2013/slides/CristinaVideiraLopes\\_ExercisesInStyle.pdf](http://gotocon.com/dl/goto-aar-2013/slides/CristinaVideiraLopes_ExercisesInStyle.pdf)

# Writing Style



## EXERCISES IN STYLE



RAYMOND QUENEAU

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

What could be something like a programming style?

# Exercises in Programming Style\*



@cristalopes

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

# 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 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
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
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
48             # Let's reset
49             start_char = None
50
51             i += 1
52
53     for tf in word_freqs[0:25]:
54         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

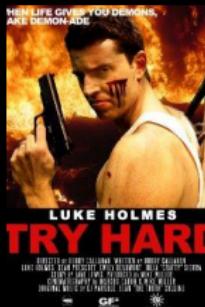
- ▷ No [named] abstractions
- ▷ Very few [long] lines of code
- ▷ Advanced libraries / constructs



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

```

```

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 # The main function
78 #
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 words=[]
4
5 freqs=[]
6
7
8
9
10
11
12
13     Takes a path to a file and assigns the entire
14     contents of the file to the global variable data
15     """
16
17     global data
18     f = open(path_to_file)
19     data = data + list(f.read())
20
21
22
23     Replaces all nonalphanumeric chars in data with white space
24     """
25
26     global data
27     for i in range(len(data)):
28         if not data[i].isalnum():
29             data[i] = ' '
30         else:
31             data[i] = data[i].lower()
32
33
34     Scans data for words, filling the global variable words
35     """
36
37     global data
38     global words
39     data_str = ''.join(data)
40     words = words + data_str.split()
41
42
43
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     indeces = []
50     for i in range(len(words)):
51         if words[i] in stop_words:
52             indeces.append(i)
53     for i in reversed(indeces):
54         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     """Takes a path to a file and returns the entire
8     contents of the file as a string
9     """
10    f = open(path_to_file)
11
12    return ...
13
14
15 def filter(str_data):
16     """Takes a string and returns a copy with all nonalphanumeric
17     chars replaced by white space
18     """
19    return ... .replace('[\W_]+', ' ', str_data)
20
21
22 def normalize(str_data):
23     """Returns a copy with all chars in lower case
24     """
25    return ... .lower()
26
27
28 def scan(str_data):
29     """Takes a string and scans for words, returning
30     """
31    return ...
32    .lower().split()
33
34
35 def rem_stop_words(wordl):
36     """Takes a list of words and returns a copy with all stop
37     words removed
38     """
39    f = open('../stop_words.txt')
40    stop_words = f.read().split(',')
41    f.close()
42
43    return ... [word for word
44                if not word in stop_words]
45
46
47 def frequencies(wordl):
48     """Takes a list of words and returns a dictionary associating
49     words with frequencies of occurrence
50     """
51    word_freqs = {}
52
```

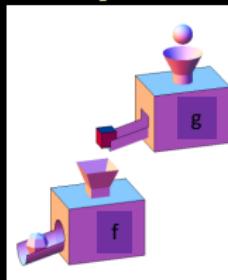
```
55    for w in word_list:
56        if w in word_freqs:
57            word_freqs[w] += 1
58
59    return ... - 1
60
61
62 def sort(word_freqs):
63     """Sorts the word frequency dictionary by frequency
64     and returns a list of pairs where the entries are
65     sorted by frequency
66     """
67
68    return ... .sort(key=operator.itemgetter(1))
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](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     return func(..., normalize)
10
11
12 def filter_chars(data, func):
13     ...
14     return func(..., scan)
15
16 def normalize(data, func):
17     ...
18     return func(..., remove_stops)
19
20 def scan(data, func):
21     ...
22     return func(..., frequencies)
23
24
25 def remove_stops(data, func):
26     ...
27     return func(..., sort)
28
29
30 def frequencies():
31     """
32     Takes a list
33     words with f
34     """
35
36
37 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

# 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

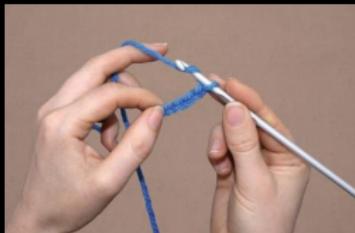
- ▶ Functions take one additional parameter, f
  - called at the end
  - given what would normally be the return value plus next function



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 TFEExercise(object):
8     __metaclass__ = ABCMeta
9
10    def info(self):
11        return self.__class__.__name__ + ": No major data
12            structure"
13
14 class DataStorageManager(TFEExercise):
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(TFEExercise):
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         # 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(TFEExercise):
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(TFEExercise):
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
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         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
46     def info(self):
47         return self.__class__.__name__ + ": My major data"
48
49     """ Models the stop word filter """
50     _stop_words = []
51     def __init__(self):
52
53         f = open('../stop_words.txt')
54         self._stop_words = f.read().split(',')
55         f.close()
56         # add single-letter words
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 WordFreqManager(TFExercise):
66
67     word_freqs = {}
68
69     def inc_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
77         return sorted(self._word_freqs.items(), key=operator.
78                         itemgetter(1), reverse=True)
79
80     def info(self):
81         return self.__class__.__name__ + ": My major data"
82         + "structure is a " + self._word_freqs.__class__.__name__
83
84 class WordFreqController(TFExercise):
85
86     def __init__(self, path_to_file):
87         self._storage_manager = DataStorageManager(path_to_file)
88         self._stop_word_manager = StopWordManager()
89         self._word_freq_manager = WordFrequencyManager()
90
91     def run(self):
92
93         for word in self._storage_manager.words():
94             if not self._stop_word_manager.is_stop_word(word):
95                 self._word_freq_manager.increment_count(word)
96
97         word_freqs = self._word_freq_manager.sorted()
98         for tf in word_freqs[0:25]:
99             print tf[0], ' - ', tf[1]
100
101
102 # Main
103 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
56 # The actual work of splitting the input into words
57 result = []
58 words = _remove_stop_words(_scan(_normalize(_filter_chars(
59     data_str))))
60 for w in words:
61     result.append((w, 1))
62
63 return result
64
65 def count_words(pairs_list_1, pairs_list_2):
66     """
67         Takes a two lists of pairs of the form
68             [(w1, 1), ...]
69         and returns a list of pairs [(w1, frequency), ...],
70             where frequency is the sum of all the reported occurrences
71
72 mapping = dict((k, v) for k, v in pairs_list_1)
73 for p in pairs_list_2:
74     if p[0] in mapping:
75         mapping[p[0]] += p[1]
76     else:
77         mapping[p[0]] = 1
78
79 return mapping.items()
80
81 #
82 # Auxiliary functions
83 #
84
85 def read_file(path_to_file):
86     """
87         Takes a path to a file and returns the entire
88             contents of the file as a string
89
90 f = open(path_to_file)
91 data = f.read()
92 f.close()
93
94 return data
95
96 def sort(word_freq):
97     """
98         Takes a collection of words and their frequencies
99         and returns a collection of pairs where the entries are
100             sorted by frequency
101
102         return sorted(word_freq, key=operator.itemgetter(1), reverse=
103             True)
104
105 #
106 # The main function
107 #
108 splits = map(split_words, partition(read_file(sys.argv[1]), 200))
109 splits.insert(0, []) # Normalize input to reduce
110 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 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().split()
51     f.close()
52     # add single-letter words
53     stop_words.extend(list('aeiou'))
54     return [w for w in word if w not 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 #
77 # Auxiliary functions
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]

```

```

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])

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

    A LIST OF WORDS.
    """
    return str_data.split()

def _remove_stop_words(word_list):
    f = open('../stop_words.txt')
    stop_words = f.read().split(',')
    f.close()
    # add single-letter words
    stop_words.extend(list(string.ascii_lowercase))
    return [w for w in word_list if not w in stop_words]

# The actual work of splitting the input into words
result = []
words = _remove_stop_words(_scan(_normalize(_filter_chars(
    data_str))))
for w in words:
    result.append((w, 1))

return result

def count_words(pairs_list_1, pairs_list_2):
    """
    Takes a two lists of pairs of the form
    [(w1, 1), ...]
    """
    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))

```

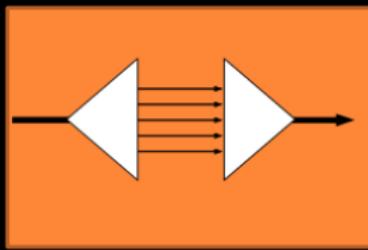
```
1 import sys, re, operator, string
2
3 #
4 # Functions for map reduce
5 #
6 def partition(data_str, nlines):
7     """
8     Generator function that partitions the input data_str (a big
9     string)
10    into chunks of size nlines
11    """
12    lines = data_str
13    for i in xrange(nlines):
14        yield '\r\n'.join(lines[i:i+nlines])
15
16 def split_words(data):
17     """
18     Takes a string and splits it into words, normalizing them to
19     lower case, and removing punctuation and numbers.
20     It returns a list of pairs [(w1, 1), (w2, 1), ...]
21     """
22    def _filter_chars(char):
23        """
24        Takes a character and replaces it with a space if it is not
25        a letter or digit, otherwise returns the character.
26        """
27        pattern = '[^a-zA-Z0-9]'
28        replaced = pattern.replace(char, ' ')
29        return replaced
30
31    def _normalize(string):
32        """
33        Takes a string and converts it to lower case.
34        """
35        lower = string.lower()
36        return lower
37
38    def _scan(string):
39        """
40        Takes a string and returns a list of words.
41        """
42        return string.split()
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]
51
52    # The actual work of splitting the input into words
53    result = []
54    words = _remove_stop_words(_scan(_normalize(_filter_chars(
55        data_str))))
56    for w in words:
57        result.append((w, 1))
58
59    return result
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
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
```

# 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*

# Programming Style

There is always more than one way to express an idea

- ▶ Kiss! Getting things straight forward is very difficult  
*Simplicity does not precede complexity, but follows it*
- ▶ One programmer's elegant solution is another one unpleasant beast

Improve your style now

- ▶ There is a huge literature on good programming style/methodology/buzzwords
- ▶ Reading code should be the way to learn, or at least write a lot of code

But don't search for your own style

- ▶ You are a journalist, not a poet: express ideas clearly with no stylistic trick
- ▶ Programming is a team game ↵ stick to some programming standards
- ▶ Programming tricks and Golf style should remain a game

# Bad Style Coding as a Game

## The International Obfuscated C Code Contest ([www.ioccc.org](http://www.ioccc.org))

- ▶ Yearly contest of intentionally obfuscated codes (in C; exist for other languages)

Example:

(arachnid, 2004 entry)

```
#include <curses.h>/*************  
    int          m[256] [ 256 ], a  
, b ;;; ;;; WINDOW*w; char*l="" "\176qx1" "q" "q" "k" "w\  
xm" "x" "t" "j" "v" "u" "n" ,Q[  
]= "Z" "pt!ftd" "qdc!eu" "dq!$c!nnwf"/* *** */"t\040\t";c(  
int u , int v){ v?m [u] [v-  
1] |=2,m[u][v-1] & 48?W] [v-1] & 15]):0:0;u?m[u -1] [v] |=1 ,m[  
u- 1] [ v]& 48? W-1 ] [v ] &  
15] ):0:0;v< 255 ?m [ u] [v+1] |=8,m [u] [v+1] & 48? W] [ v+1]&15]  
) :0 :0; u < 255 ?m [ u+1 ] [v ] |=  
4,m[u+1] [ v]&48?W+1] [v ] &15]):0:0;W] [ v]& 15]);}cu(char*q){ return  
*q ?cu (q+ 1)& 17q [0] ++:  
q[0] -- :1; )d( int u , int/**/v, int/**/x, int y){ int  
Y=y -v, X=x -u; int S,s ;Y< 0?Y =-Y ,s,  
s=- i:( s=1);X<0?X=X ,S=-1 :(S= 1); Y<= 1;X<<=i; if(X>Y){  
int f=Y -(X >>1 ); while(u!= x){  
f>= 0?v+=s,f-=X:0;u +=S ;f+= Y;m[u] [v] |=32;mvwaddch(w,v ,u, m[u  
][ v]&16){c(u,v); ;;; return;}} lelse{int f=X -(Y>>1 ); while  
(v !=y ){f >=0 ?u +=S, f-= Y:0  
;v +=s ;f+=X;m[u] [v] |= 32;mvwaddch(w,v ,u, m[u] [v] &64?60:46);if(m[u  
][ v]& 16){c( u,v );  
; return;}}}}Z( int/**/a, int b){ le( int/**/y,int/**/ x){  
int i ; for (i= a;i <=a  
+S;i++)d(y,x,i,b),d(y,x,i,b+L);for(i=b;i<=b+L;i++)d(y,x,a,i),d(y,x,a+ S,i  
); ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;  
mvwaddch(w,x,y,64); ; ; ; ; ; ; ; ; ; ; ; ; ; ; ; ;  
) } main( int V , char *C[  
] ) {FILE*f= fopen(V==1?"arachnid.c":/* */ :C[ 1],"r");int/**/x,y,c,
```

# Bad Style Coding as a Game

The International Obfuscated C Code Contest ([www.ioccc.org](http://www.ioccc.org))

- ▶ Yearly contest of intentionally obfuscated codes (in C; exist for other languages)

Example: Full (interactive) Maze Escape Game (arachnid, 2004 entry)

```
#include <ncurses.h>/*************  
    int          m[256] [ 256 ],a  
,b ;;; ;;; WINDOW*w; char*l="" "\176qx1" "q" "q" "k" "w\  
xm" "x" "t" "j" "v" "u" "n" ,Q[  
]= "Z" "pt!ftd" "qdc!eu" "dq!$c:nnwf"/* *** */"t\040\t";c(  
int u ,      int v){           v?m [u] [v-  
1] |=2,m[u][v-1] & 48?W[v-1] & 15]):0:0;u?m[u -1][v] |=1 ,m[  
u- 1][v]& 48? W-1 ] [v ] &  
15] ):0:0;v< 255 ?m [ u ][v+1] |=8,m [ u ][v+1] & 48? W[ [ v+1]&15]  
) :0 :0; u < 255 ?m [ u+1 ][v ] |=  
4,m[u+1] [ v]&48?W+1 [ v ] &15]):0:0;W[ [ v ] & 15 ]);}cu(char*q){ return  
*q ?cu (q+ 1)& 17q [0] ++:  
q[0] -- :1; )d( int u , int/**/v, int/**/x, int y){ int  
Y=y -v, X=x -u; int S,s ;Y< 0?Y =-Y ,s,  
s=- i:( s=1);X<0?X=X ,S=-1 :(S= 1); Y<= 1;X<<=i; if(X>Y){  
int f=Y -(X >>1 ); while(u!= x){  
f>= 0?v+=s,f-=X:0;u +=S ;f+= Y;m[u][v] |=32;mvwaddch(w,v ,u, m[u  
][ v]&16){c(u,v); ;;; return;}} lelse{int f=X -(Y>>1 ); while  
(v !=y ){f >=0 ?u +=S, f-= Y:0  
;v +=s ;f+=X;m[u][v] |= 32;mvwaddch(w,v ,u,m[u][v]&64?60:46);if(m[u  
][ v]& 16){c( u,v );  
; return;}}}}Z( int/**/a, int b){ le( int/**/y,int/**/ x){  
int i ; for (i= a;i <=a  
+S;i++)d(y,x,i,b),d(y,x,i,b+L);for(i=b;i<=b+L;i++)d(y,x,a,i),d(y,x,a+  
S,i  
); ;;; ;;; ;;; ;;; ;;; ;;; ;  
mvwaddch(w,x,y,64); ;;; ;;; ;;; refresh( w,b,a,0,0 ,L- 1,S-1  
);} main( int V , char *C[  
] ) {FILE*f= fopen(V==1?"arachnid.c"/**/ :C[ 1],"r");int/**/x,y,c,
```

# Bad Style Coding as a Game

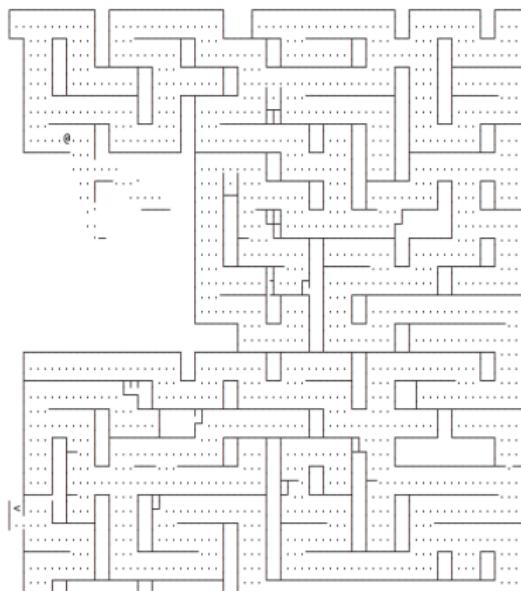
The International Obfuscated C Code Contest ([www.ioccc.org](http://www.ioccc.org))

- Yearly contest of intentionally obfuscated codes (in C; exist for other languages)

Example: Full (interactive) Maze Escape Game (arachnid, 2004 entry)  
Screenshot

```
#include <ncurses.h>/*************  
    int          m[256]           ] [      256 ],a  
,b ;;; ;;; WINDOW*w;   char*l=""  "\176qx1" "q" "q" "k" "w\"  
xm" "x" "t" "j" "v" "u" "n" ,Q[  
]= "Z" "pt!ftd" "qdc!eu" "dq!$c!nnwf"/* *** */"t\040\t";c(  
int u ,      int v){           v?m [u] [v-  
] i=2,m[u][v-1] & 48?W[v-1] & 15]):0:0;u?m[u-1][v] |=1 ,m[  
u-1] [v]& 48? W-1 ] [v ] &  
15] ):0:0;v< 255 ?m [u][v+1] |=8,m[u][v+1] & 48? W [v+1]&15]  
4,m[u+1] [v]&48?W+1] [v]&15]):0:0;W [v]&15 ]);}cu(char*q){ return  
*q ?cu (q+ 1)& 17q [0] ++:  
q[0] -- :1; }d( int u , int/**/v, int/**/x, int y){ int  
Y=y -v, X=x -u; int S,s ;Y< 0?Y =-Y ,s,  
s=- i:( s=1);X<0?X=-X,S =-i :(S= 1); Y<= 1;X<<=i; if(X>Y){  

```



# Recreational Obfuscation: Phillips entry of IOCCC'88

## Program code

```
#include <stdio.h>
main(t,_,a){return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n":9:16:t<0?t<-72?main(_,_t,
"@n'++,#'/*{}w+/w#cdnr/+,{ }r/*de}++/*{**+,/w{ %+,/w#q#n+/,#{l ,+/,n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/' r : 'd* '3,}{w+K w' K: '+}e#';dq#`l \
q#'+d'K#!/+k#;q#`r}eKK#}w'r)eKK{nl}'/#;#q#n'){})#}w'){}){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l ,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iWk{KK{nl]!/w{ %'1##w#' i; :{nl]}'/*{q#`ld;r'}{nlwb!/*de}'c \
;:{nl}'-{ }rw]'/+,}##'*}#nc, ,#nw]'/+kd'+e}+;#'rdq#w! nr' / ') }+}{rl#'{n' ' )# \
}'+}##(!!/")
:t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
:0<t?main(2,2,"%s")*:a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#1,{ }: \nuwloca-0;m .vpbks,fxntdCeghiry"),a+1);}
```

## Output

On the first day of Christmas my true love gave to me  
a partridge in a pear tree.

On the second day of Christmas my true love gave to me  
two turtle doves  
and a partridge in a pear tree.

On the third day of Christmas my true love gave to me  
three french hens, two turtle doves  
and a partridge in a pear tree.

On the fourth day of Christmas my true love gave to me  
four calling birds, three french hens, two turtle doves  
and a partridge in a pear tree.

## Output (cont)

On the eighth day of Christmas my true love gave to me  
eight maids a-milking, seven swans a-swimming,  
six geese a-laying, five gold rings;  
four calling birds, three french hens, two turtle doves  
and a partridge in a pear tree.

On the ninth day of Christmas my true love gave to me  
nine ladies dancing, eight maids a-milking, seven swans a-swimming  
six geese a-laying, five gold rings;  
four calling birds, three french hens, two turtle doves  
and a partridge in a pear tree.

On the tenth day of Christmas my true love gave to me  
ten lords a-leaping,  
nine ladies dancing, eight maids a-milking, seven swans a-swimming

# Bad Coding Style as an Art

## Another example

```
#include <stdio.h>
int l;int main(int o,char **o,
int I){char c,*D=o[1];if(o>0){
for(l=0;D[l]           ];D[l
++]-=10){D[l+=1]-=120;D[l]-=
110;while  (!main(0,0,l))D[l]
+= 20;  putchar((D[l]+1032)
/20    );putchar(10);}else{
c=o+   (D[I]+82)%10-(I>l/2)*
(D[I-l+I]+72)/10-9;D[I]+=I<0?0
:! (o==main(c/10,0,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

# Bad Coding Style as an Art

## Another example: Computing Integer Square Roots

```
#include <stdio.h>
```

```
int l;int main(int o,char **o,
int I){char c,*D=0[1];if(o>0){
for(l=0;D[l] ] ;D[l
++]-=10){D [l++]-=120;D[l] ==
110;while (!main(0,0,l))D[l]
+= 20; putchar((D[l]+1032)
/20 ) ;}putchar(10);}else{
c=o+ (D[I]+82)%10-(I>1/2)*
(D[I-1+I]+72)/10-9;D[I]+=I<0?
:!(o==main(c/10,0,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

It actually works

```
$ ./cheong 1234
```

```
35
```

( $35 \times 35 = 1225$ ;  $35 \times 36 = 1296$ )

```
$ ./cheong 112233445566
```

```
335012
```

$335012 \times 335012 = 112233040144$

$335013 \times 335013 =$

112233710169

# Bad Coding Style as an Art

## Another example: Computing Integer Square Roots

```
#include <stdio.h>
int l;int main(int o,char **o,
int I){char c,*D=0[1];if(o>0){
for(l=0;D[l] ] ;D[l
++]-=10){D [l++]-=120;D[l] ==
110;while (!main(0,0,l))D[l]
+= 20; putchar((D[l]+1032)
/20 ) ;putchar(10);}else{
c=o+ (D[I]+82)%10-(I>1/2)*
(D[I-1+I]+72)/10-9;D[I]+=I<0?0
:! (o==main(c/10,0,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

It actually works

```
$ ./cheong 1234
35
(35 × 35 = 1225; 35 × 36 = 1296)
$ ./cheong 112233445566
335012
335012 × 335012 = 112233040144
335013 × 335013 = 112233710169
```

Author claim: code self-documented...

```
#include <stdio.h>
int l;int main(int o,char **o,
int I){char c,*D=0[1];if(o>0){
for(l=0;D[l] ] ;D[l
++]-=10){D [l++]-=120;D[l] ==
110;while (!main(0,0,l))D[l]
+= 20; putchar((D[l]+1032)
/20 ) ;putchar(10);}else{
c=o+
(D[I]+82)%10-(I>1/2)*
(D[I-1+I]+72)/10-9;D[I]+=I<0?0
:! (o==main(c/10,0,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

# Bad Coding Style as an Art

## Another example: Computing Integer Square Roots

```
#include <stdio.h>
int l;int main(int o,char **o,
int I){char c,*D=0[1];if(o>0){
for(l=0;D[l] ] ;D[l
++]-=10){D [l++]-=120;D[l] ==
110;while (!main(0,0,1))D[l]
+= 20; putchar((D[l]+1032)
/20 ) ;}putchar(10);}else{
c=o+ (D[I]+82)%10-(I>1/2)*
(D[I-1+I]+72)/10-9;D[I]+=I<0?0
:! (o==main(c/10,0,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

It actually works

```
$ ./cheong 1234
35
(35 × 35 = 1225; 35 × 36 = 1296)
$ ./cheong 112233445566
335012
335012 × 335012 = 112233040144
335013 × 335013 =
112233710169
```

Author claim: code self-documented...

```
#include <stdio.h>
int l;int main(int o,char **o,
int I){char c,*D=0[1];if(o>0){
for(l=0;D[l] ] ;D[l
++]-=10){D [l++]-=120;D[l] ==
110;while (!main(0,0,1))D[l]
+= 20; putchar((D[l]+1032)
/20 ) ;}putchar(10);}else{
c=o+ (D[I]+82)%10-(I>1/2)*
(D[I-1+I]+72)/10-9;D[I]+=I<0?0
:! (o==main(c/10,0,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

*It is an old observation that the best writers sometimes disregard the rules of rhetoric. When they do so, however, the reader will usually find in the sentence some compensating merit, attained at the cost of the violation. Unless he is certain of doing as well, he will probably do best to follow the rules.*

– William Strunk, Jr. (1918)

# Last one, just for fun: dhyang IOCCC'00

Saitou Hajime image

## Source code

```
#define /*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*-----*/
```

## Last one, just for fun: dhyang IOCCC'00

# Saitou Hajime image that prints a prog

## Source code

# Output 1

## Last one, just for fun: dhyang IOCCC'00

Saitou Hajime image that prints a prog that prints a prog

## Source code

## Output 1

## Output 2

Last one, just for fun: dhyang IOCCC'00

Saitou Hajime image that prints a prog that prints a prog that prints a prog

## Source code

## Output 1

## Output 2

## Output 3

## Last one, just for fun: dhyang IOCCC'00

Saitou Hajime image that prints a prog that prints a prog that prints a prog ...  
Repeating endlessly "aku soku zan", Hajime's motto meaning *slay evil immediatly*.

## Source code

## Output 1

### Output 3

## Output 2

## Output 4 (=1)

This is a Quine: [https://en.wikipedia.org/wiki/Quine\\_\(computing\)](https://en.wikipedia.org/wiki/Quine_(computing))

# Conclusion

## Computer Science is the Science of Abstraction

- ▶ Computer Scientists are engineers terraforming ideas and concepts
- ▶ Write code to communicate with humans, and accidentally to execute it
- ▶ Programming is not about technology for its own sake. It's about being able to express your ideas precisely and efficiently. <http://prog21.dadgum.com/>

## FP vs. OOP: how you prefer to state your ideas?

- ▶ OOP is all about nouns, FP is all about verbs
- ▶ Please keep the troll level low: It's the mutable state that is evil, not the object

## Don't hope to get it right on the first time

- ▶ To a great extent the act of coding is one of organization. Refactoring. Simplifying. Figuring out how to remove extraneous manipulations here and there.
- ▶ Write it. Rewrite it correct. Rewrite it efficient. Rewrite it modifiable / elegant.

## This course is now over

- ▶ I really hope that you will like your long journey on the programmer path