SWE 621

FALL 2022

PROGRAMMING STYLES

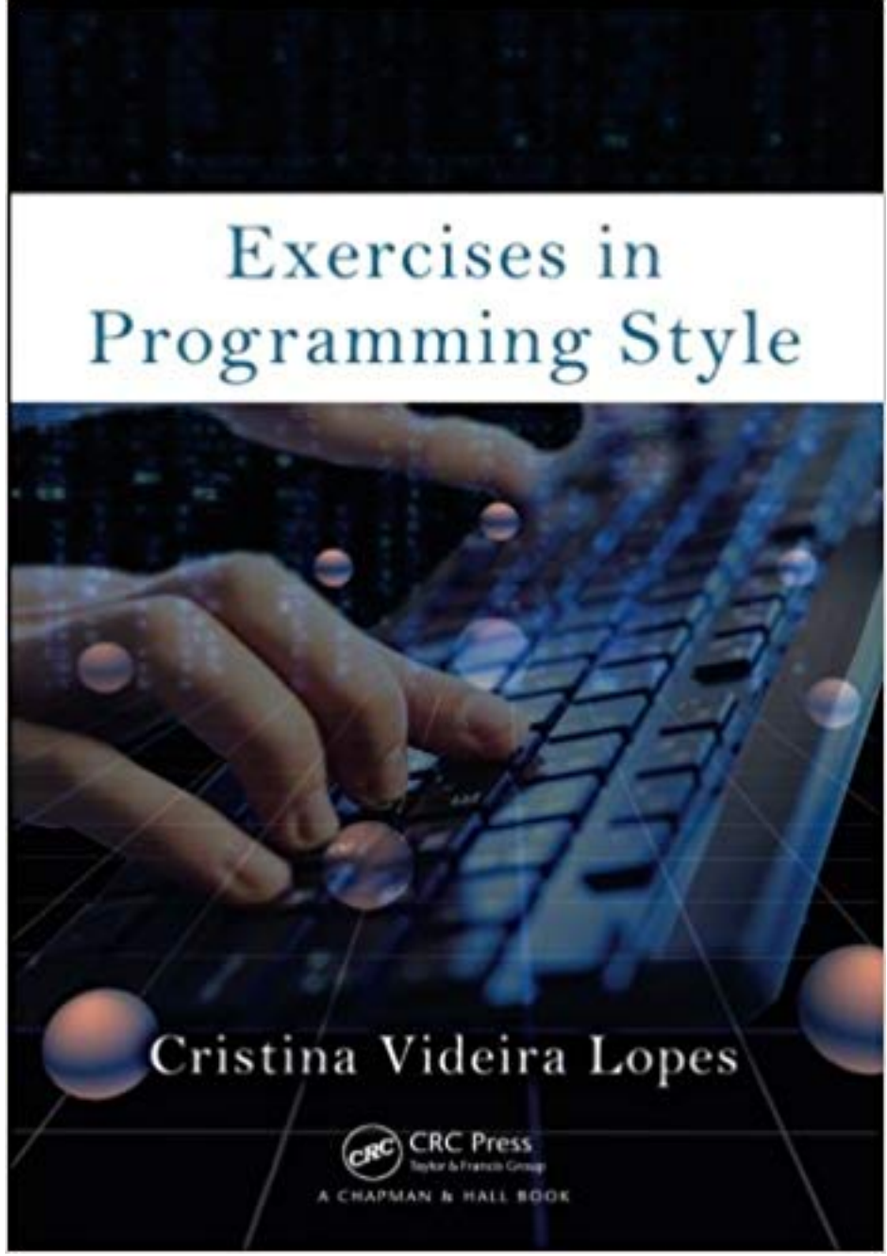
LOGISTICS

- HW4 due today
- HW5 due on 11/21

REVIEW OF LAST WEEK'S ACTIVITY

PROGRAMMING STYLE

- A set of constraints on how code is written which help achieve specific requirements or quality attributes
- Describe alternative ways in which code might be written
  - make it object-oriented
  - make it functional
  - lazily load data from input source
  - give each element a separate thread
- Like architectural styles and design patterns, has consequences that adopting programming style help achieve
  - But not always as well-defined and enumerated



EXERCISES IN PROGRAMMING STYLE

- Presentation is centered around an example problem
- Each program offers the same baseline behavior (sometimes adding an additional feature)
- Can directly compare and contrast how the same problem is solved each style
- Directly illustrates the diversity of ways of programming
  - Many different ways to solve the same problem
- Some are related to programming language features (e.g., OO, functional, reflection)
  - But many modern languages support a range of language features that support a diversity of styles
  - Can write something in a procedural style (i.e., ignoring OO features) even in Java
- Examples written in Python

EXAMPLE PROBLEM: TERM FREQUENCY

- Given a text file, print the 25 most frequent words and corresponding frequencies
- Sort from most frequent to least frequent
- Normalize for capitalization and ignore "stop" words (e.g., the, for, ...)

Input

Tigers live mostly in India

Wild lions live mostly in Africa

Output

live - 2
mostly - 2
africa - 1
india - 1
lions - 1
tigers - 1
wild - 1

SOME TYPES OF PROGRAMMING STYLES

- Basic styles
- Functional styles
- Reflection styles
- Data-centric styles
- Concurrency styles

EXAMPLES OF PROGRAMMING STYLES

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

- 5-cookbook/procedural https://github.com/crista/exercises-in-programming-style/tree/master/05-cookbook
- 6-pipeline https://github.com/crista/exercises-in-programming-style/tree/master/06-pipeline
- 7-code golf https://github.com/crista/exercises-in-programming-style/blob/master/07-code-golf/tf-07.py
- 8-infinite mirror / recursive https://github.com/crista/exercises-in-programming-style/tree/master/08-infinite-mirror
- 10-things/OO https://github.com/crista/exercises-in-programming-style/tree/master/11-things
- 15-hollywood/inversion of control https://github.com/crista/exercises-in-programming-style/tree/master/15-hollywoo
- 16-b board /publish subscribe https://github.com/crista/exercises-in-programming-style/tree/master/16-bulletin-boar
- 19-aspects https://github.com/crista/exercises-in-programming-style/tree/master/19-aspects
- 20-plugins https://github.com/crista/exercises-in-programming-style/tree/master/20-plugins
- 26-persistent tables/relational https://github.com/crista/exercises-in-programming-style/tree/master/26-persistent-tables
- 28-lazy rivers/streams https://github.com/crista/exercises-in-programming-style/blob/master/28-lazy-rivers/tf-28.py
- 31-map reduce https://github.com/crista/exercises-in-programming-style/tree/master/31-map-reduce

COOKBOOK / PROCEDURAL

- Complexity tamed by dividing problem into procedures
- Procedures take input, but don't necessarily produce output relevant to problem (e.g., output status codes)
- Procedures instead often share state through global variables
- Problem is solved by repeatedly applying procedures to update shared state
- Consequences
  - Not idempotent - repeatedly calling procedure generates new output
  - Global variables can be hard to debug and reason about

PIPELINE

- Problem decomposed into functions, which take input and produce output
- No shared state between functions
- Problem solved by composing functions (f(g(x)))
- Consequences
  - Easy to test, easy to parallelize (e..g, MapReduce)

CODE GOLF

- As few lines as possible
- Consequences
- Sometimes: hard to understand, bugs
- But also sometimes: easy to understand, elegant
- Helpful when used appropriately

INFINITE MIRROR / RECURSIVE

- Problem is solved using induction, specifying a base case (n0) and inductive step (n + 1)
- Consequences
  - Can lead to stack overflow for languages that don't support tail recursion optimization

THINGS / OO

- Problem decomposed into things that make sense for problem domain
- Thing exposes operations and has state
- State is hidden and accessed only through operations

HOLLYWOOD / INVERSION OF CONTROL

- Elements are never called on directly
- Provide interfaces to register for callbacks (i.e., use Observer)
- Consequences
  - Inverts dependency relationship
  - Promotes extensibility

B BOARD / PUBLISH SUBSCRIBE

- Elements never called directly
- Central infrastructure for publishing and subscribing to events (bulletin board)

ASPECTS

- Aspects are added to functions / procedures without any edits to code
- External binding mechanism binds abstractions to aspects
- Consequences
  - Can reify scattered concerns in many methods into one place (e.g., tracing, logging, security)
  - Can inject dependencies

PLUGINS

- Main program and plugins separately compiled
- Plugins loaded dynamically by main program, using external config
- Main program uses plugins without knowing implementation
- Consequences
  - Enables adding 3rd party behavior to a program

PERSISTENT TABLES

- Data exists before and after execution of program and shared between programs
- Data is stored in way that makes it easier and faster to explore
- Problem is solved through queries against data
-

LAZY RIVERS / PIPES & FILTERS

- Data is available on streams
- Functions are filters / transformers from one kind of data stream to another

MAP / REDUCE

- Input data divided into blocks
- Map function applies a given worker function to each block of data, potentially in parallel
- Reduce function takes the results of many workers functions and recombines them into coherent output

SUMMARY

- Many choices about how to implement a solution
- Programming styles offer a vocabulary for talking about alternative implementations
- Makes explicit the constraints which lead to a specific style of programming
  - Can consider explicitly the consequences of following these constraints

IN CLASS ACTIVITY

SKETCH IMPLEMENTATION IN LAZY-RIVER STYLE

- Work in groups of 2 or 3, pick an OO language (e.g., Java, Python, C#)
- Sketch an implementation of the following
  - Given a text file, output all words alphabetically, along with the page numbers on which they occur. Ignore all words that occur more than 100 times. Assume a page is a sequence of 45 lines.
    - abatement - 89
      abhorrence - 101, 145, 152, 241, 274, 281
      abhorrent - 253
      abide - 158, 292
- Does not need to compile and run, just looking for a sketch that illustrates following the programming style for this problem