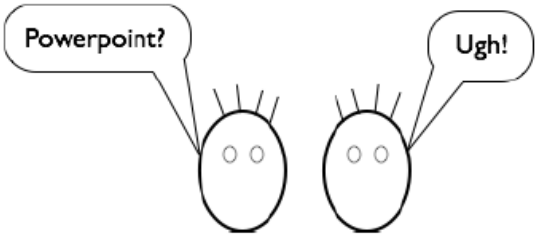
CPSC 110
Systematic Program Design

- Who, Why, What and How
- Start working on the first module
- Don't worry – this course uses almost no powerpoint after these intro slides!



While you wait

- https://cs110.students.cs.ubc.ca/setup.html
- Skip to installing DrRacket
  - but stop when you get to "setup test file"
  - that will let you type at DrRacket for class
  - go back after class and do skipped setup page steps
- Or, do all this after class and just use pen and paper during class!

Covid Safety (brief)

- You are required to wear a mask at all times in this class
  - Unless you have a mask exemption form from Centre for Accessibility
  - It must be tight fitting and cover your nose and mouth
  - No eating during class.
  - If you must take a drink keep mask on between sips
  - If you are not wearing a mask you will have to leave the class
- To reduce exposure please sit in the same seat or close to it every day

Welcome to CPSC 110 Online

- Our lecture will be very much like the on-campus lecture
- No talking in class, no chatting on Zoom
- Except when I ask you to work on a problem
  - then the chat will be turned on for you to ask questions
  - please do not answer other student's questions

Who?

Gregor Kiczales (he/him/his)
Professor of Computer Science

Research in programming languages and program design
Common Lisp Object System (CLOS)
Portable CommonLoops (PCL)
Metaobject Protocols (MOPs)
Art of the Metaobject Protocol
Aspect-Oriented Programming (AOP)
AspectJ
EdX Systematic Program Design (aka How to Code) courses

Who?

Joanna McGrenere (she/her/hers)
Professor of Computer Science

Research in human-computer intereaction
Personalized User Interfaces
Universal Usability
Interactive Technologies for Older Adults
Computer Supported Cooperative Work
Qualitative and Quantitative Methodologies

Who?

William J. Bowman

Research in programming language design and implementation, in particular, secure and verified compilers, program verification, metaprogramming, andlanguage interoperability.

Who?

Emily Fuchs
Course Coordinator

cpsc110-admin@cs.ubc.ca

<<< administrative question go here
Technical questions go to Piazza
only sensitive personal issues go to instructor email

+ 46 other TAs who work on labs, office hours, Piazza, and grading

Why?

- Programming is fun
- Software development is serious
  - well designed software can do wonderful things
  - but poorly designed software can
    - violate privacy, lose money, damage property, injure or kill people…
- We would like to have confidence in the software we write
  - that it does what is wanted, works properly, is reliable

writing software that just seems to work correctly is nowhere near good enough!

What?

- Foundations of software engineering
  - more than just programming

Margaret Hamilton accepting US Presidential Medal of Freedom

Key ideas (1/5)

quickly today
but revisit these a few
weeks into the course

- Representing information as data
  - information is out there in the world
  - programs operate on data inside the computer that represents that information
- Programs have structure
- Code that works properly is not good enough
- How we work determines what we produce
- Software is a team activity

Key ideas (1/5)

- Representing information as data
- Programs have structure
  - of different kinds
  - local and crosscutting
  - being able to work in terms of that structure is powerful
  - separates engineering from just typing code
- Code that works properly is not good enough
- How we work determines what we produce
- Software is a team activity

Key ideas (2/5)

- Representing information as data
- Programs have structure
- Code that works properly is not enough
  - Need to be able to explain why you are confident it works properly
  - Need to be able to reliably produce similar programs
- How we work determines what we produce
- Software is a team activity

Key ideas (3/5)

- Representing information as data
- Programs have structure
- Code that works properly is not enough
- How we work determines what we produce
  - we can't rely on jolts of brilliance
  - systematic program design makes solving hard problems easier
  - ACM Code of Ethics 2.1
    Strive to achieve high quality in both the processes and products of professional work.
- Software is a team activity

Key ideas (4/5)

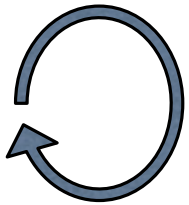
- Representing information as data
- Programs have structure
- Code that works properly is not enough
- How we work determines what we produce
- Software is a team activity
  - useful programs always have to be modified by other programmers later
  - being kind to other developers is essential to success
  - applies throughout our work – the code we produce, the questions we ask, the answers we give

kindness starts on Piazza!

Systematic Program Design (SPD)

- Systematic program design is a way of working that reliably produces well written, consistent, and well tested programs.
- Based on research and practice in programming languages and software engineering.
- Provides a foundation for professional software development
- Also relevant if you are NOT intending to be a software developer
  - helpful for programs of all sizes, including 2 page quick programs
  - underlying ideas help with all kinds of problem solving and design

Basic course structure



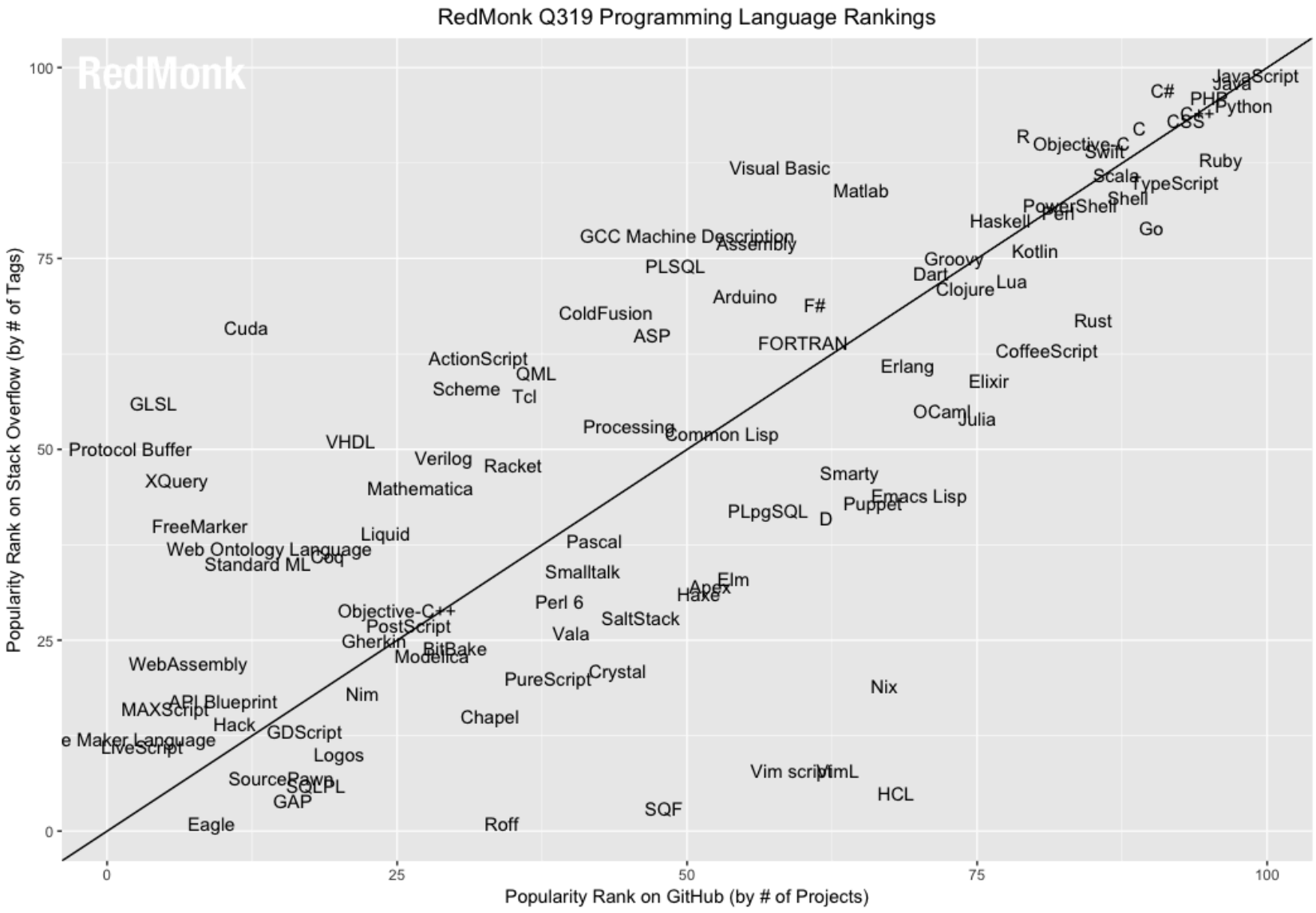
- First we use easy problems to learn SPD
  - then we use SPD to solve difficult problems

- That cycle happens every lecture, every week, across the term
  - lecture starts w/ easy problem, then harder
  - module starts w/ easy problems, then harder
  - course starts w/ easy problems, then harder
    - a program to produce twice the given number (Tuesday)
    - a program to schedule 110 lab TAs (near end of the course)

- The content and pace get more difficult as we go

Beginning Student Language (BSL)

- Programs are written in different languages
  - There are 10s of thousands of languages; thousands in active use. Hundreds are popular.
  - No one language is the most useful, best etc.
- BSL is the core of essentially all other languages
  - allows us to focus on learning systematic program design
  - prepares you for learning other languages quickly
  - never say a university course taught a language
- Puts all students on level playing field


RedMonk Q319 Programming Language Rankings
RedMonk
Popularity Rank on Stack Overflow (by # of Tags)
Popularity Rank on GitHub (by # of Projects)
0
25
50
75
100
JavaScript
Java
C#
PHP
Python
C++
CSS
R
Objective-C
C
Swift
Ruby
Scala
TypeScript
Shell
PowerShell
Perl
Haskell
Go
Visual Basic
Matlab
GCC Machine Description
Assembly
Kotlin
Groovy
PLSQL
Dart
Clojure
Lua
Arduino
F#
ColdFusion
Rust
Cuda
ASP
FORTRAN
CoffeeScript
ActionScript
QML
Erlang
Elixir
Scheme
Tcl
GLSL
OCaml
Julia
Processing
Common Lisp
VHDL
Protocol Buffer
Verilog
Racket
Smarty
XQuery
Mathematica
Emacs Lisp
Puppet
PLpgSQL
D
FreeMarker
Liquid
Pascal
Web Ontology Language
Coq
Standard ML
Smalltalk
Elm
Apex
Haxe
Perl 6
Objective-C++
SaltStack
PostScript
Vala
Gherkin
BitBake
Modelica
WebAssembly
Crystal
PureScript
Nix
Nim
API Blueprint
MAXScript
Chapel
Hack
GDScript
e Maker Language
LiveScript
Logos
Vim script
HCL
SourcePawn
SQLPL
GAP
SQF
Eagle
Roff

Software Design is

- The situated selection and use of:
  - Design techniques
  - Science and engineering expertise
  - Tool skills
  - Team skills

Situated means we need to be able to use the right skill, in the right way, at the right time.

Lecture and labs are based on situated learning. You work on solving a problem, we help you learn how to solve it.

The goal of the design problems

- In lecture/lab/problem-sets/homework you will be working through program design problems
  - The real goal is NOT to simply handin a working solution to the problems.
  - It is to learn to solve the problem on your own.
  - If we help you too much, if you look at the solution too soon, if you get help from a friend, then you won't learn how to solve them on your own.
  - You have to work through the difficulty on your own.
  - It will be difficult, you will get stuck. That's when the learning happens!
  - Watching your friend lifting weights doesn't make you stronger.

Course Components - Lecture

- Before lecture you will work through videos and problems on edge.edx.org
- Lecture will mix presentation of new material with you working on problems
  - "priming" enables situated learning of new topics
  - expect lecture to be difficult and tiring – experience doing real design
  - 15% of course grade is the work you hand in during lecture
  - more on this later, but must be DURING lecture
  - See me after class if you would prefer not to bring a laptop to class
- After lecture you will review material from lecture and work through additional videos and problems on edge.edx.org

Course Components - Labs

- Designing code to solve more challenging problems
- The problems will be relevant to what you recently learned in lecture
- Helping and supporting other students
- Answering design review questions
- Explaining your code
- TAs will also ask you questions about prior week problem set (for marks)

Course Components - Problem Sets

- Close out each module with a problem set that assesses your mastery of all the material to date
- THE PROBLEM SETS PREPARE YOU FOR THE EXAMS
- Collaboration policy is in the Welcome to 110 Section on edX

  >>> READ IT! The consequences of academic misconduct are quite bad. <<<
- Combined assessment:
  - automatic grading (autograder)
  - during lab a TA will ask you questions about how you designed the software

Course Components - Other

- Office hours - instructor and TA (See Piazza)
- Midterms and final
  - assessment of your mastery of systematic program design
  - on campus
- Average of Midterm 2 and Final must be passing to pass the course
- There is no textbook, everything you need is on edge.edx.org

Grading Scheme

Item	% of total course grade
Problem Sets	15%
Labs	10%
Lecture activity	15%
edX questions	0% - These are a good for your learning though, so do not skip them!
Midterm 1	15%
Midterm 2	20%
Final	25%

see https://cs110.students.cs.ubc.ca/syllabus.html for critical additional points

110 vs. 103+107

neither path is ultimately easier, but 103+107 is a more gradual pace

- additional useful 110 is all of Systematic Program Design, in 1 term, using teaching languages. Best and fastest foundation for being a major or taking CPSC 210 (Software Engineering in Java).
- 103 is about first 5-6 weeks of 110 in Python, with a few Python program templates.
- 107 is the last 7 weeks of 110, in the teaching languages, using 110 edX modules.
  - 107 takes 110 final exam.

What it takes to do well

- Don't need math, STEM, etc.
- Do need attention to detail, patience, humility
  - attention to detail because in a 100 line program one wrong character can make it wrong
  - patience because it takes time to solve hard design problems
  - humility because you need to be willing to spend time on the problems
- Many of you have attention to detail, patience, humility
  - athletes, musicians, gamers, artists, …

“Course Contract”

- Course staff will provide
  - state of the art content based on research and practice in programming and software engineering
  - delivered using state of the art pedagogy in active and online learning
  - supported by significant investment in materials and resources
  - 50 person team, extensive office hours, rapid response to questions on Piazza
- You will:
  - work hard (12+ hours/week) and stay up to date – not get behind by even a day
  - trust the design recipes to get you to a solution
  - follow course rules of decorum and academic honesty

After Class

- https://cs110.students.cs.ubc.ca/setup.html
- https://cs110.students.cs.ubc.ca/syllabus.html