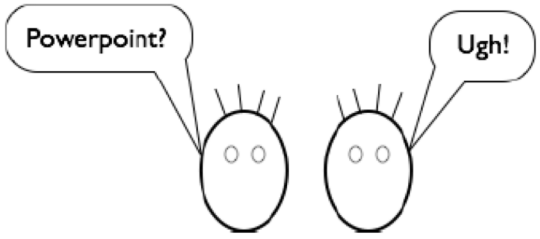
CPSC 110
Systematic Program Design

Plan for today:

- Who, Why, What and How of 110
- Start working on the first module

- Don't worry – this course uses almost no powerpoint after these intro slides!



While you wait

- Google cpsc 110 setup
  - Create an edge.edx.org account
  - Join the CPSC 110 2020S course
  - Go to the Course tab, Syllabus, Setup, scroll to Installing Dr. Racket, and follow those instructions
- Or, do all this right after class.
- Also, get ready for Socrative
  last name [A - L] https://api.socrative.com/rc/qy74dk
  last name [M - Z] https://api.socrative.com/rc/ELfukf

Who? - Staff

Gregor Kiczales
Professor of Computer Science

Pronouns he/him/his.

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? – Staff

Ming Gong

Patrick Lee

Sassan Shokoohi

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

What?

- Foundations of software engineering
  - more than just programming

Margaret Hamilton accepting US Presidential Medal of Freedom

Key ideas

- 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

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

Key ideas

- 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

- 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

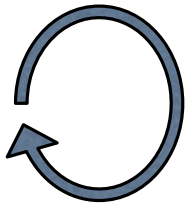
Systematic Program Design

- 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 ides help with all kinds of problem solving and design

Systematic Program Design (SPD)

- What information does the program process?
- What should the program do?
- Are there any edge or special cases?
- What are the sufficient tests?
- What key properties of the program should we write down?
- What will make it easiest for others to change later?
- Can we use prior work to shorten the program?

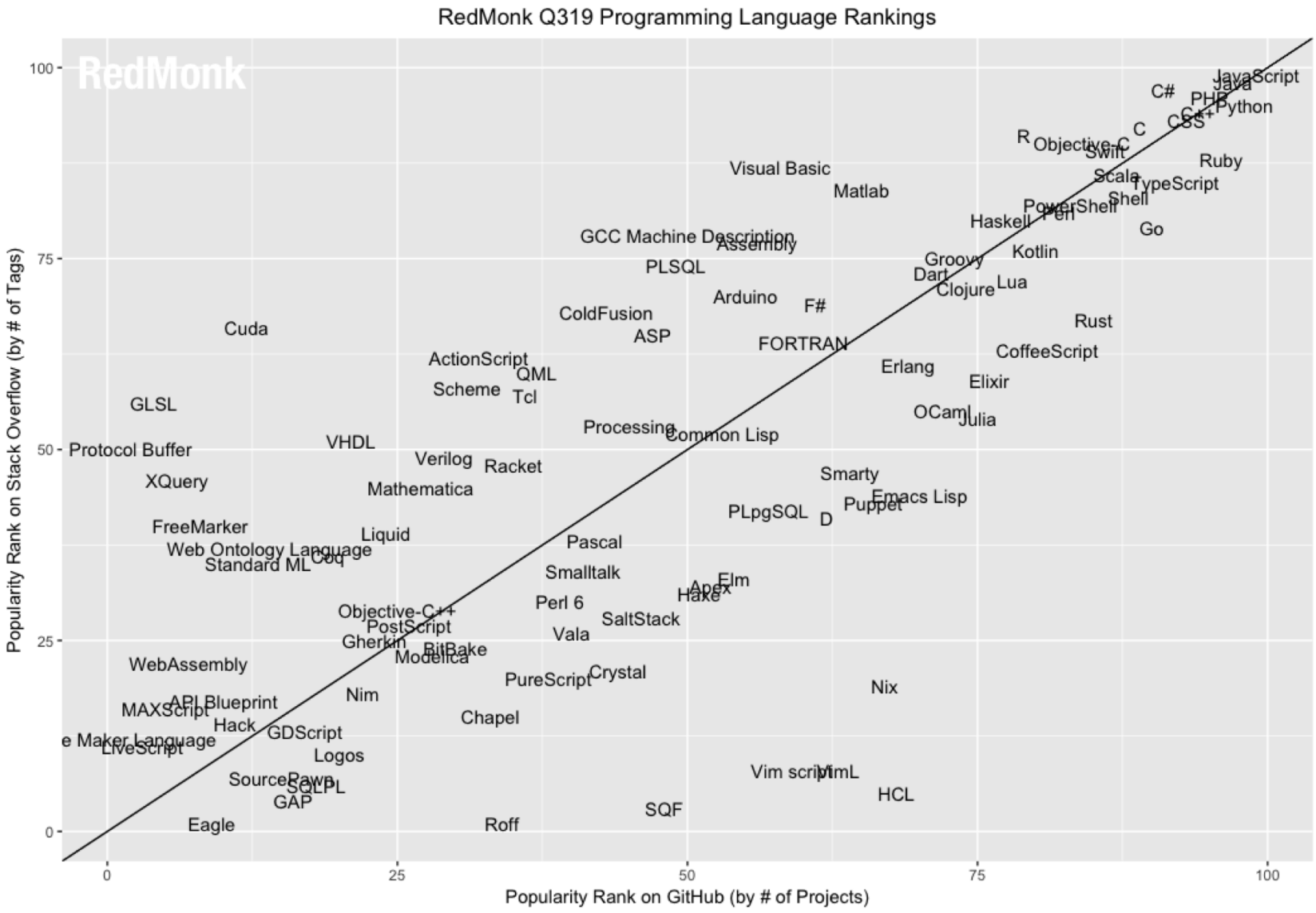
Basic course structure



- 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
    - 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


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
PLpgSQL
D
Puppet
FreeMarker
Liquid
Pascal
Web Ontology Language
Coq
Standard ML
Smalltalk
Elm
Apex
Haxe
Perl 6
Objective-C++
SaltStack
PostScript
Vala
Gherkin
BitBake
Modelica
WebAssembly
PureScript
Crystal
Nix
Nim
API Blueprint
MAXScript
Chapel
Hack
GDScript
e Maker Language
LiveScript
Logos
Vim script
VimL
SourceMod
SQLPL
HCL
GAP
SQF
Eagle
Roff

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 design
  - prepares you for learning other languages quickly
  - never say a university course taught a language

Software Design is

- The situated use 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 problems

- In lecture/lab/problem-sets/practice-problems you will be working program design problems
  - The goal is NOT to handin a solution to the problems!
  - It is to learn to solve the problem.
  - If we help you too much, if you look at the solution too soon, 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!

Course Components

- Lecture
  - Before lecture you will work through videos and problems on edge.edx.org
  - Lecture will be a mix of lecturing and working on problems
    - "priming" enables situated learning of new topics
    - expect lecture to be difficult and tiring – experience doing real design
  - After lecture you will review material from lecture and work through additional videos and problems on edge.edx.org
- Labs
  - Chance to work through problems in depth with TA assistance (average score is high)

Course Components

- Problem sets
  - Close out each "module" with a problem set that assesses your mastery of all the material to date
  - Collaboration policy is in the Welcome to 110 Section on edX READ IT!
  - Combined assessment:
    - autograder
    - sitting down with TA together with in lab. <<<< NEW

Course Components

- Office hours - instructor and TA (See Welcome to 110 section on edX)
- Midterms and final
  - Paper based assessment of your mastery of systematic program design
  - Average of Midterm 2 and Final must be passing to pass the course <<<<< NEW
- There is no textbook, everything you need is on edge.edx.org

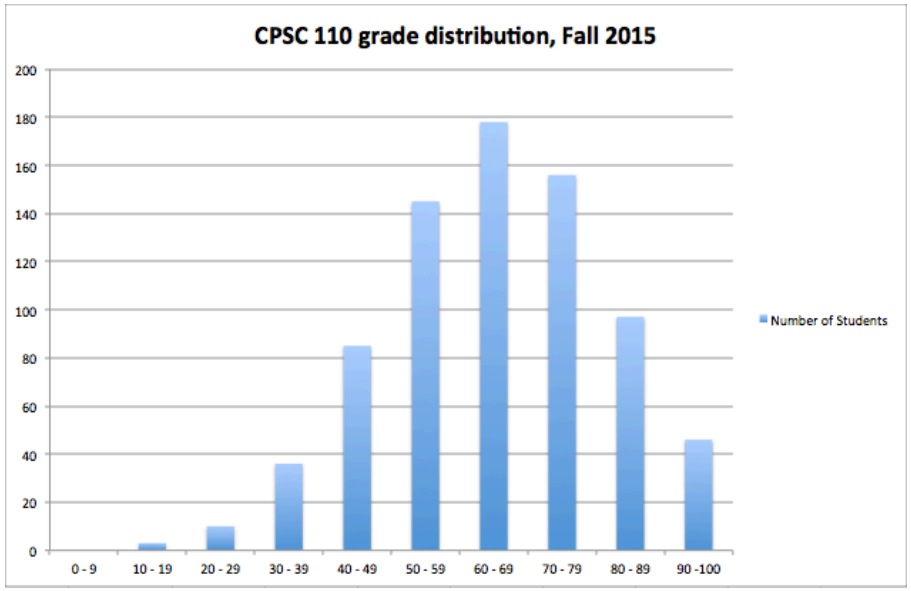
Socrative (alternative to clickers)

- We will have online questions during lecture
- Almost always right at the beginning
- The edX online questions are great prep for these
- They are graded on correctness
- last name [A - L] https://api.socrative.com/rc/qy74dk
  last name [M - Z] https://api.socrative.com/rc/ELfukf

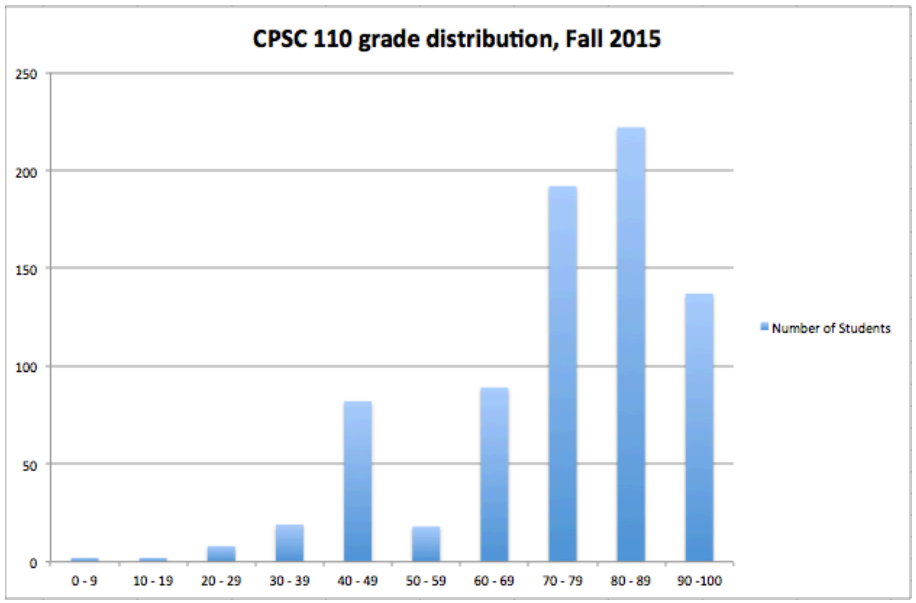
Which is the correct…

You've heard that 110 is hard… which is the correct grade distribution?

A



B



110 vs. 103+107

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

- 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 additional useful Python program templates.
- 107 is the last 6-7 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, …

Decorum

- Why?
  - Course and lecture require attention and hard work
  - Research shows that doing other stuff in class distracts others
  - ~ $470k in tuition ~ $500k more in government support for domestic students
- What?
  - Do not interfere with anyone else's learning
  - No Facebook, Twitter, YouTube etc.
  - No talking, texting etc.
  - We are here for active learning, we are here for hard work

300	80%	240	$ 176.45	$ 705.80		$ 169,392	$ 508,176
	20%	60	$ 1,256.33	$ 5,025.32		$ 301,519	
						$ 470,911	

After Class

- Google search "UBC CPSC 110", go to Google Sites listing
- Follow instructions to create edX EDGE account.
- Go to the edX EDGE course
- Look for START HERE