
# CS10: The Beauty and Joy of Computing - Fall 2015

Lecture 1: Introduction, Abstraction

- Introduction
- Course Overview
- Abstraction

# We Live in the Future

"When wireless is perfectly applied the whole earth will be converted into a huge brain, which in fact it is, all things being particles of a real and rhythmic whole. We shall be able to communicate with one another instantly, irrespective of distance. Not only this, but through television and telephony we shall see and hear one another as perfectly as though we were face to face, despite intervening distances of thousands of miles; and the instruments through which we shall be able to do this will be amazingly simple compared with our present telephone. A man will be able to carry one in his vest pocket.

We shall be able to witness and hear events--the inauguration of a President, the playing of a world series game, the havoc of an earthquake or the terror of a battle--just as though we were present."

- Nikola Tesla, "When Woman Is Boss", Colliers Magazine, 1926

# Why Study CS?

- Massive impact on our lives and society as a whole.
- Increasingly useful for all fields of study and areas of employment.
- Beautiful mathematics and ideas.

# Course Overview

# Course Staff

Instructor: Josh Hug (me) hug@cs.berkeley.edu 779 Soda

GSIs:

| | |
|---|---|
| Carlos Flores | Joseph Cawthorne |
| Arany Uthayakumar | Andy Schmitt |
| Adam Kuphaldt | Rachel Huang |
| Steven Traversi | Victoria Shi |
| Janna Golden | Alex McKinney |
| Amruta Yelamanchili | Claire Watanabe |

Erik Dahlquist

(Or just Carjorandy Adarastevijannalex Amrutaclairek for short)

# Who Are You Guys?

# The Snap! and Python Programming Languages

Snap!: Super easy to learn. Exactly as powerful as Python.

Python: Harder to learn. Faster to use for experts. Wildly popular in real world.


```
def max(x, y):
    if (x > y):
        return x
    else:
        return y
```

# BJC Topics

Big Ideas of Programming:

- Abstraction
- Algorithms (2)
- Recursion (2)
- Functions-as-data (2)
- Programming paradigms
- Cloud computing

Big Ideas of Computing:

- How the internet works
- Research summaries
- The power of data (big, small)
- Social implications of computing
- Saving the world with computing
- Cloud computing
- Limits of computing
- Future of computing
- Robots

Beauty and Joy

- “CS Unplugged” activities
- Lab work in pairs
- “Creation” project (of your own choice)
- “Explore” project (of your own choice)

# Course Websites

Course Websites:

Not quite done yet.
Will be later today.

- Main site, used for everything not listed below: http://cs10.org
- Discussion forums and announcements: https://piazza.com
  - You should be enrolled already. Click here if not.
- bCourses, used only for submitting homework: https://bcourses.berkeley.edu/courses/1371647
- Exam grades and regrade requests: www.gradescope.com

Syllabus including all course logistics can be found on our main website at the Course Policies link.

- Next 5 slides only summarize most important policies.
- Make sure to read these policies carefully! We expect you to know them.

# Weekly Workload

- Do weekly readings over weekend (or at least before lab).
  - No required textbook. All readings posted online.
- Lectures on Monday and Wednesday (2 hours per week)
  - Attendance not required, but clicker questions earn EPA (see slide 12).
- Lab (4 hours per week)
  - Attendance required for reading quizzes and lab check offs.
- Discussion (1 hour per week) on Friday
  - Attendance not required, but can earn EPA (see slide 12).
- CS10 is a demanding course, but very fun!

| Weekly Readings | Lecture 1 (Monday) | Lab 1, incl. reading quiz | Lecture 2 (Wednesday) | Lab 2 | Discussion (Friday) | Homework (F/Sa/Su) |
|---|---|---|---|---|---|---|
| | 1 hour | 2 hours | 1 hour | 2 hours | 1 hour | |

# Goals of Each Part of the Course

Lecture: Big picture ideas. Less emphasis on programming language details.

Lab: Nuts and bolts of programming. This is where you build fundamental programming skills.

HW and Projects: Refinement of programming skills and deeper exploration of topics that most interest you.

Discussion: Discussion of big ideas and tricky concepts.

Exams: Assess big picture understanding (for both you and your transcript).

# Lab Logistics

Reading quiz: Reflections on the readings (starting next week).

*Pair Programming* exercises: Will work with a partner on labs.

- One person is the “driver”: Does all the typing and mousing.
- One person is the “navigator”: Reviews as the other person types.
- Used to great effect in both education and industry.

Lab checkoffs (starting next week):

- Check-off questions: Required, covering only some of each lab.
- Extra questions: To self-check your own understanding.
  - Experience has shown that students who complete all of the labs do better in CS10.

# Grades and Grading

Total score is out of 500.

- In-lab reading quizzes: 20
- Lab checkoffs: 30
- Homeworks: 10+20+30 = 60
- Projects: 75+75 = 150
- Exams: 25+75+100 = 200
- Writing Assignment: 40

3 slip days. See course policies.

More details later.

No curve. Will consider EPA for borderline grades.

- E: Effort, e.g. office hours, reading Piazza
- P: Participation, e.g. lecture, discussion
- A: Altruism, e.g. helping others in lab or on Piazza

| Points | Grade |
|---|---|
| 485-500 | A+ |
| 460-484 | A |
| 450-459 | A- |
| 440-449 | B+ |
| 420-439 | B |
| 400-419 | B- |
| 375-399 | C+ |
| 360-374 | C |
| 350-359 | C- |
| 300-349 | D |
| < 299 | F |

# Waitlist

We will not be adding additional sections, but anticipate being able to accommodate everyone.

- Drop rate for CS10 is fairly high at the start of the semester.
- If you are enrolled but need a different section, don’t mess with Telebears, you’ll just end up waitlisting yourself.
- Feel free to attend a different discussion or lab than the one for which you are officially registered, or if not registered, go to any lab or discussion.
    - **Caveat: If we run out of seats in a particular discussion or lab, priority will go to those officially enrolled (even if they show up late)!**

# Logistics Questions?

# Abstraction

# Complexity and Abstraction

Programming is easy, so long as your programs are small.

- Complexity is our enemy.
- Abstraction is the key to conquering complexity.

Learning to reason using the most appropriate abstraction is a key goal of CS10, CS61A, CS61B, CS61C, and beyond.

# Abstraction Allows Us to Hide Unnecessary Details

Abstraction:

- **Detail Removal**: Hide unnecessary details from users.

Detail Removal Example:

- Modern user interface: Right pedal is “accelerate”, left is “decelerate”
- Even as underlying technology has changed, this abstraction has not!
  - Computer controlled fuel injection.
  - Anti-lock brakes (ABS).

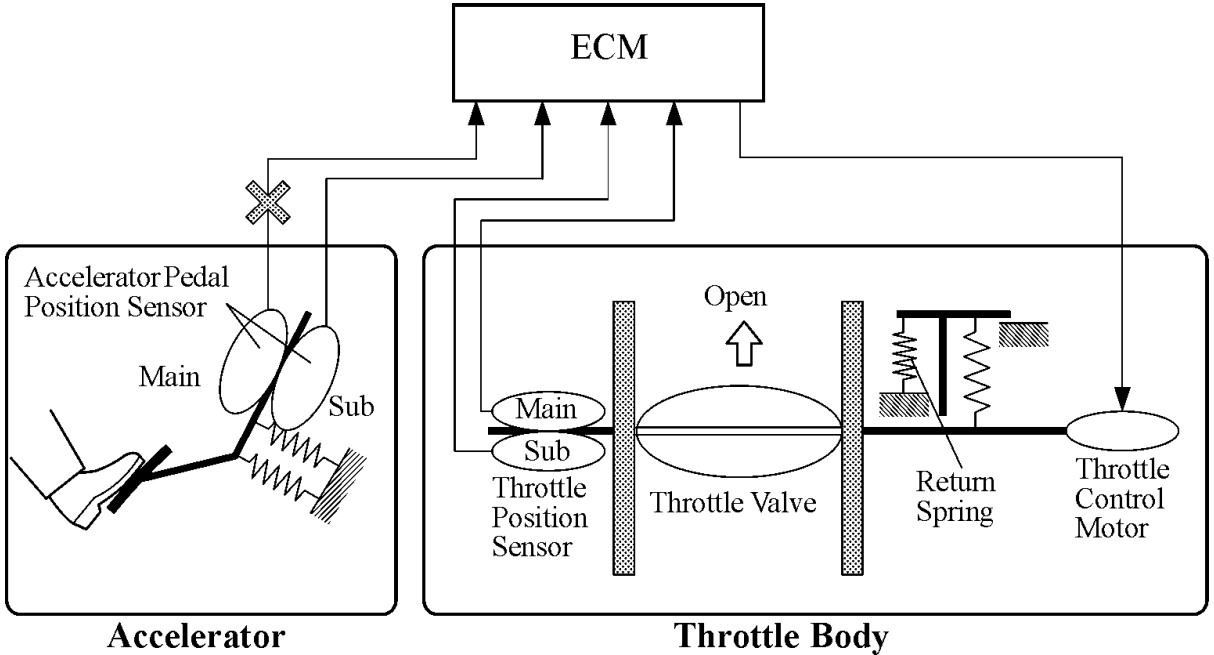
Would have been bad design to add more controls!

- e.g. nobody wants an ABS control knob!

# Abstraction: Allows Us to Hide Unnecessary Details

Abstraction:

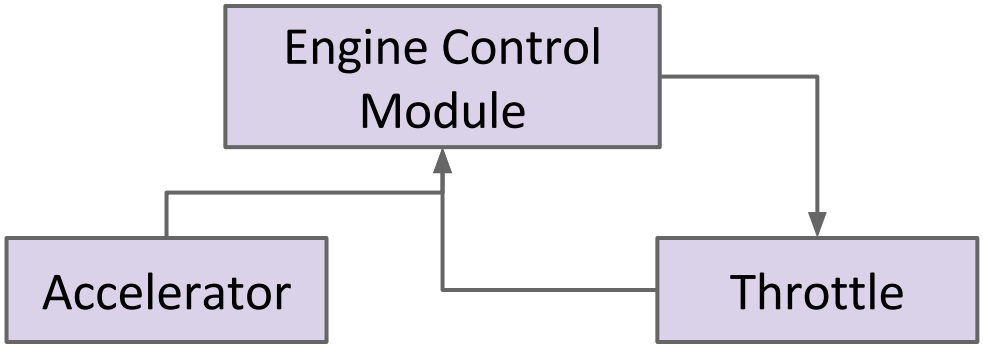
- **Detail Removal**: Hide unnecessary details from **other designers**.
  - e.g. Engine Control Module designer doesn't care about the return spring inside the Throttle.



# Abstraction: Allows Us to Hide Unnecessary Details

Abstraction:

- **Detail Removal**: Hide unnecessary details from **other designers**.
  - e.g. Engine Control Module designer doesn't care about the return spring inside the Throttle.
- Nice to be able to think of a system as a hierarchy of well defined "chunks" with precise functionality. In CS, we say that we have a **separation of concerns**.



# Abstraction: Allows Us to Build General Purpose Artifacts

Abstraction:

- **Detail Removal**: Hide unnecessary details from users and designers.
- **Generalization**: Avoid unnecessary repetitive work.

Generalization Example:

- Extensible shower rods: Can be used in nearly any shower.

Let’s see how these principles apply when we’re writing programs.

- There will be MANY unfamiliar concepts during this demo. Don’t panic!

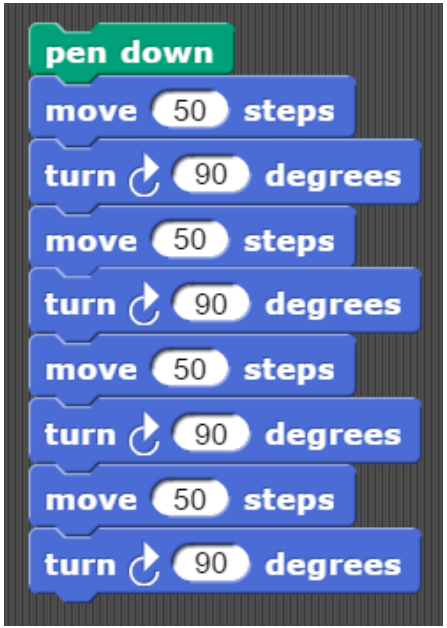
# Our First Snap! Program: Drawing a Square

(See webcast for live demo!)

# Abstraction and Snap! Sneak Preview: Drawing a Square

The following programs both draw the figure on the far right.

- The program on the right makes better use of abstraction. We observe that drawing a square is just moving/turning 4 times. This saves us work.
  - Are any details hidden?

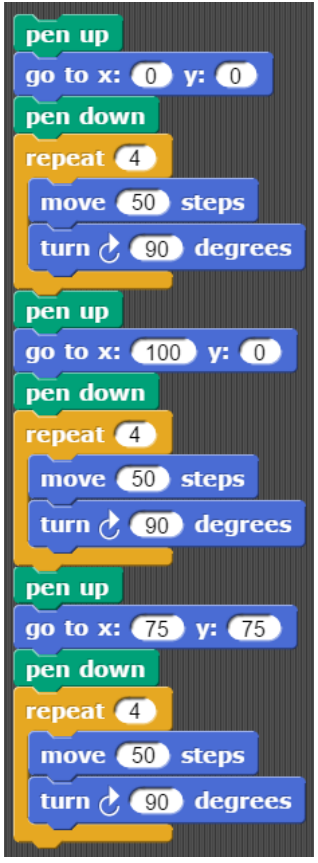


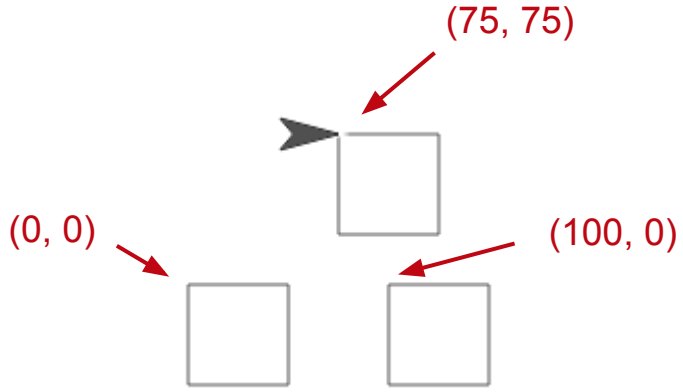


Note: If you're looking at these slides online and don't understand the programs above, this is OK. Snap will be covered in this weeks lab!

# Abstraction and Snap! Sneak Preview: Drawing Three Squares

The following program draws a square at positions (0, 0), (100, 0), (75, 75).

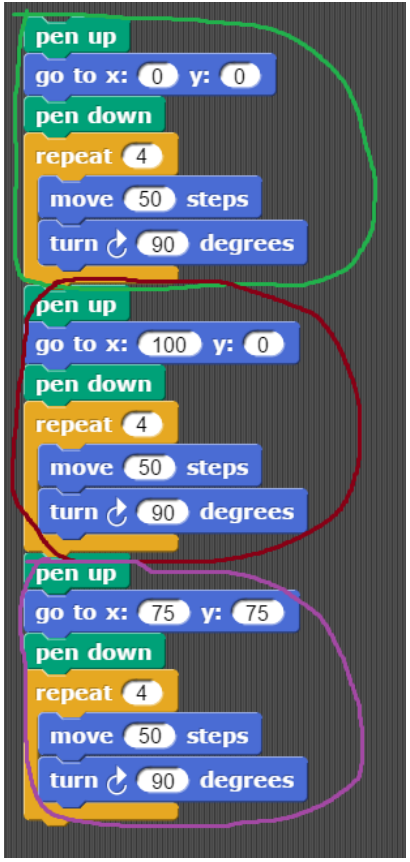



What is beautiful about this program? What is not?

# Abstraction and Snap! Sneak Preview: Drawing Three Squares

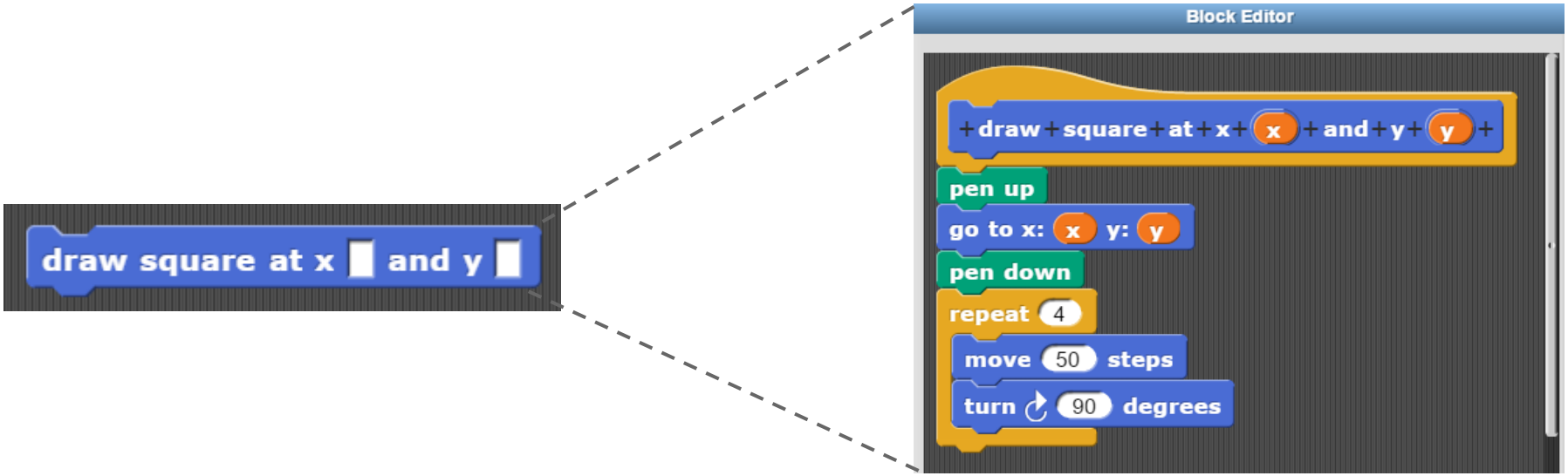
If you were designing Snap!, how would you let your programmers generalize?



# Sneak Prevew of Next Week's Lab: Building-your-own-blocks

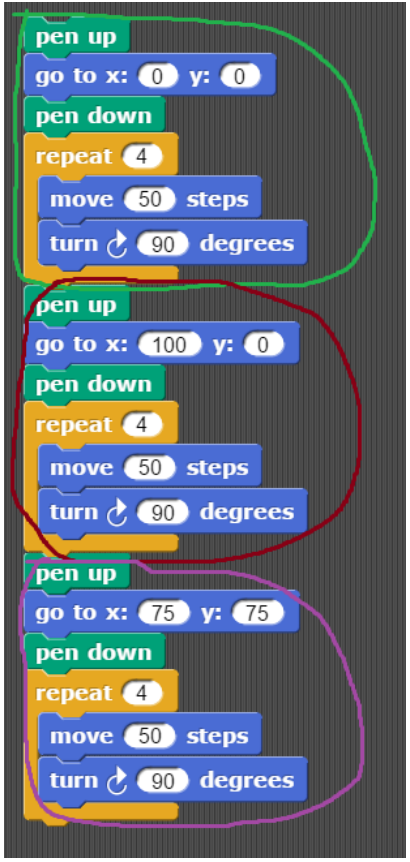
In Snap!, we build our own blocks.

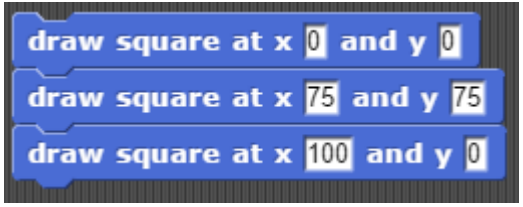
- We added a new block called "draw square at x and y".
- Details of how it works are hidden from user.
- Acts as a general square drawing tool. User specifies the x and y coordinate.



# Sneak Prevew of Next Week's Lab: Building-your-own-blocks

In Snap!, we build our own blocks. Left program is ugly. Right program is nice!



# Parting Thoughts

One of the most important challenges in computer science:

- Identifying the blocks you need to build to solve your problem.
- Equivalent statement: Identifying the right abstraction that you need to solve your problem.

Personal observation: Biggest difference between weak and strong programmers in 61B is the ability to do this!

# Lab This Week

Make sure to go to your SECOND scheduled lab of the week this week.

- Example: If your labs are M/W, go to your Wednesday lab.
- Example 2: If you labs are W/F, go to your Friday lab.

Lab will cover many important ideas not discussed in lecture! Lecture will focus more on the big picture, not Snap! minutiae.

- Snap! language details are easier to learn through your own experimentation rather than by watching me.

Reminder: Lecture today was not to teach you how to program, but to introduce you to some big ideas. You'll gain programming familiarity during your first lab.

# Citations

Gas and brake “padal” image from:

https://driversed.com/images/v2008coursecontent/brakepadal.jpg

Detailed car internals block diagram image from: http://cheapautopartsdiscount.pridebike.ru/toyota-4runner-2003/srm3_files/srm3-1.png

Self driving car image from The Guardian: http://www.theguardian.com/technology/2014/may/28/google-self-driving-car-how-does-it-work

Lecture structure adapted from Dan Garcia, Brian Harvey, and Gerald Friedland.