Welcome to CS 100!

CS 100: Introduction to the Profession
Matthew Bauer & Michael Saelee
Agenda

- Syllabus & Administrivia
- ITP: Course overview
- What is CS? (What is it not?)
- Teaching computers
§ Syllabus & Administrivia
Website: moss.cs.iit.edu/cs100
Instructors

Matthew Bauer
SB 206D
Mon/Tue 8:30AM-11AM

Michael Saelee
SB 226C
Wed/Fri 12:45PM-3/1:45PM
Teaching Assistants

**Section 01**: Samuel Golden  
Hours: Mon 5PM-6PM

**Section 02**: Grace Arnold  
Hours: Fri 11AM-12PM

**Section 03**: Przemyslaw Warias  
Hours: Tue 10AM-11AM

**Section 04**: Avery Peck  
Hours: Thu 3:30PM-4:30PM

**Section 05**: Felix Campbell  
Hours: Thu 3PM-4PM

**Section 06**: Safa Slote  
Hours: Thu 11:30AM-12:30PM

**Section 07**: Noah Dela Rosa  
Hours: Thu 3:30PM-4:30PM

**Section 08**: Gladys Toledo-Rodriguez  
Hours: Fri 10AM-11AM
Grading

10% Attendance (lecture)
20% Lecture surveys
30% Debates
10% Extracurricular assignment
30% Lab assignments
Attendance

- Bring your ID! Sign in with card reader each lecture.
- Two absences are automatically excused
  - Each following absence reduces attendance score by 10%
Lecture surveys

- Online surveys/quizzes administered during lecture (must be present to take them)
Debates

- Held during lab — resolutions typically related to prior lecture/guest speaker topic

- Topics announced in advance, but debaters are not!

- 3 for, 3 against; everyone must come prepared to argue either side — non-debaters vote for winners
## Debate Format

<table>
<thead>
<tr>
<th>Role</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affirmative: Opening argument</td>
<td>2 mins</td>
</tr>
<tr>
<td>Negative: Opening argument</td>
<td>2 mins</td>
</tr>
<tr>
<td>Prep time (Both teams)</td>
<td>4 mins</td>
</tr>
<tr>
<td>Affirmative: Rebuttal</td>
<td>2 mins</td>
</tr>
<tr>
<td>Negative: Rebuttal</td>
<td>2 mins</td>
</tr>
<tr>
<td>Grand crossfire (All debaters)</td>
<td>4 mins</td>
</tr>
<tr>
<td>Affirmative: Closing argument</td>
<td>2 mins</td>
</tr>
<tr>
<td>Negative: Closing argument</td>
<td>2 mins</td>
</tr>
</tbody>
</table>
Extracurricular assignment

- You must attend some number of extracurricular activities (e.g., hackathons, talks, conferences) and submit a writeup
- Details of activities & requirements will be decided soon!
Lab assignments

- Team paper/activity/machine problem/etc. based on lecture topic assigned in lab and (typically) submitted online
- Each graded by TAs on 4 point scale
  - 0 (did not attempt) - 4 (well executed & meets all reqs.)
- Scores on Blackboard 1-2 weeks after lab
- Missed lab = 0 for lab!
§ ITP: Course overview
“Introduction to the Profession” — i.e., what’s CS all about?
Survey of (curated) subfields of computer science

- Concurrent programming
- Machine learning & AI
- Data science
- Algorithms
- Data encryption
- High performance computing
Also: what does it mean to be a CS practitioner today?

- Ethical and social concerns
- Research / Industry career paths
- Teamwork and collaboration
Lots of lecture demonstrations, guest speakers, extracurricular opportunities, and lab activities!
§ What is CS?
(What is it not?)
Is:
- software design
- algorithms
- theory of computing
- mathematical proofs

Isn’t:
- building computers
- hardware focused
- a traditional “science”
- information technology
Computer science is no more about computers than astronomy is about telescopes.

Anonymous
Not about computers?

- Sure: we use computers as tools
- But so do folks in nearly every other data/computation intensive fields!
- Physics, Chemistry, Economics, Sociology, Music Production, etc.
Science?

science |ˈsɪəns|
noun
the intellectual and practical activity encompassing the **systematic study** of the structure and behavior of the **physical and natural world** through **observation** and **experiment**

*New Oxford American Dictionary*
Science?

- i.e., the scientific method
  - observe, hypothesize, experiment, analyze → refute/validate hypothesis
- Yeah. We don’t really do that.
Computer science is often defined as “the systematic study of algorithmic processes, their theory, design, analysis, implementation and application.” An algorithm is a precise method usable by a computer for the solution of a problem.

Encyclopedia.com
Ultimate Problem Solvers

- After a computer scientist comes up with the solution to a problem — *an algorithm* — a monkey can apply it!

- A monkey with boundless patience, a perfect memory, and who can follow instructions to the letter

- I.e., a computer
Programs

- We codify solutions into *programs* which effectively teach computers how to solve our problems for us.

- And, ideally, reuse our code to build every grander programs!
Programs have *billions* of moving pieces!

The Great Wall of China has nothing on an operating system kernel’s codebase.

Nor does any ingenuous *mechanical device*. 
Programming is certainly *not all we do*, but in order to efficiently carry out the solutions we invent, it’s often a critical step!
§ Teaching computers
Question: what are some different ways in which we can program (teach) a computer to solve problems for us?
- Pre-existing software (typically application specific)
- Step-by-step instructions (*imperative* programming)
- Describing *what* we want done, but not *how* to do it (*declarative* programming)
- Building a system to *learn* how to solve the problem on its own (machine learning)

... and many more!
Types of Programming Languages

- Imperative: *here’s how to do it*
- Declarative: *here’s what to do*
  - Logic: *deduce what I want*
  - Functional: *compute what I want*
- Domain-specific: tailored to the application
But wait!

Before we start programming, we must decide how to represent data related to the problem on a computer, and determine what computational resources are actually available to help us solve it!
Data Representation

- Most complex problems have lots of associated data!
- E.g., pieces in a chess game, airports in an airline routing system, sensors in a self-driving car
- How to store/organize/access all this data?
- Data structures, databases, file systems, etc.
There are only two hard things in Computer Science: cache invalidation and naming things.

*Phil Karlton*
Resource Constraints

- Generally: \textit{time} and \textit{space}
  
  - \textit{Computing time} — i.e., how long can we / are we willing to wait for an answer from the CPU
  
  - \textit{Memory/Disk space} — i.e., how much do we have / are we permitted / can we afford?
E.g., Robotic Vacuum (Roomba)

- How to program a robot to vacuum a room thoroughly?
- Goal: maximize manufacturer profit (i.e., minimize cost of production), but still make a good robotic
- One solution: complex AI, fast CPU, lots of memory, full-room mapping — necessary?
- What’s the alternative?
Finite-State Machine

- *Computational model* for describing programmable logic
- Consists of *states*, *transitions* between states based on *inputs*, and possible *actions* (aka *outputs*) that occur on transitions
- We often use a *state-transition diagram* to describe a FSM
Infinite Runner FSM

- **running**
  - obstacle → tap/jump
  - no obstacle → move forward

- **jumping**
  - hit ground
  - tap/jump

- **stopped**
  - miss ground/fall
  - tap/restart

- **dead**
  - run off ground/fall
  - tap/restart
Infinite Runner FSM

0: no obstacle / move forward
1: obstacle
2: miss ground / fall
3: tap/restart

Transition:
- start
- tap/jump
- hit ground
- run off ground / fall
What inputs/actions might be needed for a robotic vacuum?
- inputs: collision sensors
- actions: move in direction; suck (given)
Straight-line Robovac

- North-bound state
  - North clear / go north
  - North blocked

- South-bound state
  - South clear / go south
  - South blocked
Straight-line Robovac

- North clear / go north
- North blocked
- South clear / go south
- South blocked
Domain Specific Language

- **Syntax:** `STATE SURROUNDINGS -> ACTION NEXT_STATE`
- `STATE / NEXT_STATE = 0, 1, 2, ...`
- `SURROUNDINGS = 4 letters for matching NEWS sensor inputs` — 'X' for clear, * to ignore, direction letter for blocked
- `ACTION = N, E, W, S` for movement in direction, X for no move
Straight-line Robovac

0 x*** -> N 0  # head N if N is clear
0 N*** -> X 1  # N is blocked, switch state
1 ***x -> S 1  # head S if S is clear
1 ***S -> X 0  # S is blocked, switch state
Next Monday’s Lab: Picobot

- Write program(s) to make a simulated robovac navigate rooms with different kinds of obstacles
- Interesting question: is an FSM-based bot capable of fully covering any kind of room? (Arbitrary layout/obstacles)
- CS meta-problem: computability