Welcome to CS 100!

CS 100: Introduction to the Profession
Matthew Bauer & Michael Lee
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
Tue, even numbered Thu 12:45-1:45PM
(Google Meet)

Michael Lee
Wed/Fri 1PM-2PM
(Discord / Zoom)
Teaching Assistants

Section 01: Christopher Sherman
Section 02: Grace Arnold
Section 03: Gladys Toledo-Rodriguez
Section 04: George Schaefer
Section 05: George Stock
Section 06: George Stock
Section 07: Christopher Sherman
Section 08: Grace Arnold
Section 09: Christopher Dharma
Section 10: George Schaefer
Section 11: Gladys Toledo-Rodriguez
Section 12: Samuel Golden

(office hours on website)
Grading

5%  CS 100 Attendance
5%  First Year Experience Attendance
30% Lecture surveys
10% Debates
30% Lab assignments
20% Team assignment
Attendance

- Live or Synchronous-online attendance for CS 100 lab and lecture is mandatory!

- Live lecture rotation: Last names A-L odd-numbered weeks, M-Z even numbered weeks

- Two absences are automatically excused

- Each following absence reduces attendance score by 10%
Attendance

- First Year Experience attendance is required every week
- Each following absence reduces attendance score by 10%
Lecture surveys

- Online surveys/quizzes administered during lecture (must be present to take them, due at the end of lecture)

- Today’s lecture survey on the course website!
  - Password: FSM
Debates

- Two debates, centered on current digital society topics, will be held during lab in weeks 6 and 12
- Two teams of 3-4 will be given their topics and side (for/against) a week in advance
- Each student will be assigned to a debate team twice over the course of the semester
- Non-debaters will vote via survey during debates
# Debate Format

<table>
<thead>
<tr>
<th>Role</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affirmative: Opening argument</td>
<td>3 mins</td>
</tr>
<tr>
<td>Negative: Opening argument</td>
<td>3 mins</td>
</tr>
<tr>
<td>Prep time (Both teams)</td>
<td>6 mins</td>
</tr>
<tr>
<td>Affirmative: Rebuttal</td>
<td>3 mins</td>
</tr>
<tr>
<td>Negative: Rebuttal</td>
<td>3 mins</td>
</tr>
<tr>
<td>Grand crossfire (All debaters)</td>
<td>6 mins</td>
</tr>
<tr>
<td>Affirmative: Closing argument</td>
<td>3 mins</td>
</tr>
<tr>
<td>Negative: Closing argument</td>
<td>3 mins</td>
</tr>
</tbody>
</table>
Lab assignments

- Activity/coding 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, equally weighted

- Missed lab = 0 for lab!
Team Assignment

- “Kurzgesagt” (in-a-nutshell) video on a current computing or technology topic — dual focus on how it works & effects on society

<table>
<thead>
<tr>
<th>Lab 3</th>
<th>Lect 5</th>
<th>Lab 6</th>
<th>Lab 8</th>
<th>Lect 11</th>
<th>Lect 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pitch</td>
<td>Executive Summary</td>
<td>Storyboard &amp; Draft Script</td>
<td>Intro Video</td>
<td>Work on full video</td>
<td>Final submission</td>
</tr>
</tbody>
</table>
§ 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, 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 |ˈsīəns|
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
Two Central Issues

- Data representation: how do we describe the problem?
- Resource constraints: how much / what sort of computing power do we have available?
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 vacuum

- One solution: fast CPU, lots of memory, complex AI, full-room mapping — is this really necessary?

- What’s the alternative?
Computational Models

- We tend to reach for the most familiar — at this point, probably a general purpose CPU that can execute a “regular” computer program
- A “Turing Machine”
- But other, possibly more efficient computing models exist
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 can use a state-transition diagram to describe a FSM
Infinite Runner FSM

- **running**
  - tap/jump
  - no obstacle
  - run off ground/fall
  - hit ground

- **jumping**
  - tap/jump
  - miss ground/fall

- **stopped**
  - tap/restart
  - off screen

- **dead**
  - start
Infinite Runner FSM

- **Start State (0)**: Move forward without an obstacle.
- **State 1**: Tap/jump to switch to State 2.
- **State 2**: Tap/jump to switch to State 3 or hit the ground to switch back to State 0.
- **State 3**: Tap/restart to switch back to State 0.

Transitions:
- From State 0 to State 1 on tap/jump.
- From State 1 to State 2 on tap/jump.
- From State 2 to State 3 on tap/restart.
- From State 3 to State 0 on tap/restart.
- From State 0 to State 0 on no obstacle/move forward.
- From State 1 to State 0 on hit ground.
- From State 2 to State 0 on miss ground/fall.
- From State 3 to State 0 on off screen/run off ground/fall.
What inputs/actions might be needed for a robotic vacuum?

- inputs: collision sensors

- actions: move in direction; suck (perpetually — won’t specify)
Straight-line Robovac

![Diagram showing the state transitions for a straight-line Robovac. The states are 'north-bound' and 'south-bound.' The transitions are 'north clear / go north,' 'north blocked,' 'south clear / go south,' and 'south blocked.']
Straight-line Robovac
Domain Specific Language

- **Syntax:** `STATE SURROUNDINGS -> ACTION NEXT_STATE`
- `STATE / NEXT_STATE = 0, 1, 2, ...`
- `SURROUNDINGS = 4 letters for matching N, E, W, S 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 Week’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