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
Mon/Wed 12:45-1:45PM
(Google Meet)

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

Section 01: George Stock
Hours: Fri 10AM-12PM

Section 02: Prince Hodonou
Hours: TBA

Section 03: Gladys Toledo-Rodriguez
Hours: Thu 4PM-7PM

Section 04: George Stock
Hours: Fri 10AM-12PM

Section 05: Christopher Dharma
Hours: TBA

Section 06: Gladys Toledo-Rodriguez
Hours: Thu 4PM-7PM

Section 07: TBA
Hours: TBA

Section 08: Prince Hodonou
Hours: TBA

Section 09: Gladys Toledo-Rodriguez
Hours: Thu 4PM-7PM

Section 10: Neveen Elmajdoub
Hours: TBA

Section 11: Prince Hodonou
Hours: TBA
Grading

10% Attendance
30% Lecture surveys
20% Debates
30% Lab assignments
10% Team assignment
Attendance

- Live or Synchronous-online attendance for both lab/lecture is mandatory!
- Live 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%
Lecture surveys

- Online surveys/quizzes administered during lecture (must be present to take them)
- Today’s lecture survey on the course website!
  - Password: FSM
Debates

- Each student will participate in one debate on CS & society
- Held on one of five lectures (two debates per debate day)
- Debaters and topics announced 1 week in advance
- 6 for, 6 against per topic
- All debates online; non-debaters vote via survey
## Debate Format

<table>
<thead>
<tr>
<th>Round</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>
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

- Six separate weeks: 4, 6, 8 & 10, 12 & 14

- Separate deliverables: (1) pitch, (2) executive summary, (3) storyboard & draft script, (4) intro video, (5) full video
§ 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**
  - no obstacle / move forward
  - tap/jump

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

- **stopped**
  - miss ground/fall

- **dead**
  - tap/restart
  - off screen
  - run off ground/fall

Start state.

Obstacle condition.

Tap/jump transition.
Infinite Runner FSM

- **Start**: 0
- **States**: 0, 1, 2, 3
- **Transitions**:
  - From 0:
    - Tap/jump to 1
    - Hit ground to 2
    - Run off ground/fall to 3
  - From 1:
    - Tap/jump to 0
    - Hit ground to 2
    - Off screen to 3
  - From 2:
    - Tap/jump to 0
    - Miss ground/fall to 3
  - From 3:
    - Tap/restart to 0

**Actions**:
- Tap/jump
- Hit ground
- Run off ground/fall
- Miss ground/fall
- Tap/restart

**Conditions**:
- No obstacle/move forward
- Obstacle
- Tap/restart
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

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

- South-bound
  - South blocked
  - South clear / go south
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 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 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*