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
CS 100: Introduction to the Profession

Announcements
- Welcome to the Fall 2021 edition of CS 100!

Lectures
Aug 27
- Welcome: What is CS?
  - Lecture 01 Survey

Assignments
Week 1 starting Aug 23
- Lab 00 - Panopticon Escape Room
  - Game Codes
  - Leaderboard

Adminstrivia

Syllabus
- Course syllabus

Instructors
Lectures Friday 2:3:15pm in Stuart Building 104 (auditorium)
- Matthew Bauer
  - Office: Google Meet IIT
  - Gmail login required
  - Hours: Tues and even numbered Thurs 12:45-1:45pm
- Michael Lee
  - Office: Zoom
  - Hours: Wed/Fri 1-2PM

Website: moss.cs.iit.edu/cs100
Instructors

Matthew Bauer
Tues, even numbered Thurs 12:45-1:45pm
(Google Meet)

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

Sec 01 M 2-3:15: Christopher Sherman - Hours: F 3:30-5:30pm
   Google Meet

Sec 02 M 3:35-4:50: Grace Arnold - Hours:

Sec 03 M 5:10-6:25: Gladys Toledo-Rodriguez - Hours:
   Tues/Thurs 2-3pm (108SB)

Sec 04 T 2-3:15: George Schaefer - Hours:

Sec 05 T 3:35-4:50: George Stock - Hours:

Sec 06 T 5:10-6:25: George Stock - Hours:

Sec 07 W 2-3:15: Christopher Sherman - Hours: F 3:30-5:30pm Google Meet

Sec 08 W 3:35-4:50: Grace Arnold - Hours:

Sec 09 W 5:10-6:25: Christopher Dharma - Hours: Thurs 2-3pm (108SB)

Sec 10 R 2-3:15: George Schaefer - Hours:

Sec 11 R 3:35-4:50: Gladys Toledo-Rodriguez - Hours: Tues/Thurs 2-3pm (108SB)

Sec 12 R 5:10-6:25: Samuel Golden - Hours:
Grading

5%: CS100 Attendance
5%: First Year Experience Attendance
30%: Lecture Surveys
10%: Debates
20%: Team Assignment
30%: Lab Assignments
Attendance

- Live or Synchronous-online attendance for CS100 lecture is mandatory! Live rotation: Last names A-L odd-numbered weeks, M-Z even numbered weeks
- Live attendance for lab is mandatory!
- Two absences are automatically excused. Each following absence reduces attendance score by 10%

- First Year Experience attendance is required every week. 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, due at the end of lecture)
- Each week on the course website (login to IIT gmail before accessing)
- Today’s Password: FSM
Debates

- Two debates, centered on current digital society topics, will be held during lab in weeks 6 and 12.
- Two teams of three or four will be told the topic and side (supporting or opposing) one week in advance, so they can prepare.
- Each student will be assigned to a debate team twice over the course of the semester.
- Non-debaters will complete surveys during the 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 posted in 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:

- Lab 03 - Team Project - Pitch
- Lecture week 5 - Team Project - Executive Summary
- Lab 06 - Team Project - Storyboard & Draft Script
- Lab 08 - Team Project - Intro Video
- Lecture week 11 - Team Project - Work on Full Video
- Lecture week 15 - Team Project - Submission
§ 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 \(\rightarrow\) 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
- **Jumping**: hit ground
- **Stopped**: miss ground / fall
- **Dead**: off screen

- **Start**
- **Obstacle**: tap / jump
- **Hit ground**: tap / jump
- **Run off ground / fall**: tap / restart
Infinite Runner FSM

0: no obstacle / move forward
1: hit ground
2: tap/jump
3: off screen

- start
- obstacle
- tap/jump
- miss ground/fall
- 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

north-bound

north clear
/ go north

north blocked

south-bound

south blocked

south clear
/ go south
Straight-line Robovac

Diagram:

- State 0: North clear / go north
- State 1: South clear / go south
- North blocked
- 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