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
### Announcements
- Welcome to the Fall 2022 edition of CS 100!

### Lectures
#### Aug 26
- Welcome: What is CS?
  - Lecture 01 Survey

### Assignments
#### Week 1 starting Aug 22 (due by the end of week 1 lab)
- Lab 00 - Panopticon Escape Room
  - Game Codes

#### Week 2 starting Aug 28 (due midnight, the day before your week 3 lab)
- Lab 01 - Picobot
  - Simulator page
  - Submission form

### Administrivia

#### Syllabus
- Course syllabus

#### Instructors
Lectures Friday 1:50-3:05pm in Stuart Building 104 (auditorium)
- Matthew Bauer
  - Office: [Google Meet](https://meet.google.com/)
  - Gmail login required
  - Hours: Wed 12:45-1:45pm, at other days/times [Discord cs100](https://discord.gg)
- Michael Lee
  - Office: [Zoom](https://zoom.com)
  - Hours: Wed/Fri 1-2PM

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

Matthew Bauer
Wed 12:45-1:45pm
(Discord / Google Meet)

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

CS 100-L01  Shambhawi Sharma
CS 100-L02  Andrew Cordell
CS 100-L03  Andrew Cordell
CS 100-L04  Justin Ray Religioso
CS 100-L05  Emma Diamon
CS 100-L06  Emma Diamon
CS 100-L07  Andrew Cordell
CS 100-L08  Andrew Cordell
CS 100-L09  Mark Lou
CS 100-L10  Gauri Kumari
CS 100-L11  Justin Ray Religioso
CS 100-L12  Gladys Toledo-Rodriguez
CS 100-L13  Gladys Toledo-Rodriguez
CS 100-L14  Gladys Toledo-Rodriguez
CS 100-L15  Gladys Toledo-Rodriguez
CS 100-L16  Mohammad Firas Sada
Grading

10%: Attendance
20%: Lecture Surveys/Quizzes
10%: Ethics Assignments/Debates
40%: P33 Project (and related assignments)
20%: Lab Assignments
Attendance

- Attendance for CS100 lecture is mandatory!
- Attendance for CS100 lab is mandatory!
- 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
Ethics Assignments/Debates

- Two ethics assignments
- Two debates, centered on current digital society topics
- Two teams of two or three 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></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>
P33 Project  https://p33chicago.com/

Build meaningful connections with tech companies from the very beginning of your college journey. In this subset of CS100, you’ll be engaging with local tech employers to solve real-life business problems. You will be working in small groups under the guidance of these professionals to understand current business practices, create high-quality solutions, and get a real sense of what working in tech is like. At the end of the course, you will have acquired the skills, experiences, and connections you need to successfully continue your undergraduate program of study and entry into the tech space.

From week 3 through 12, there will be weekly P33 team assignments, and a project deliverable due at the end of week 12
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!
§ 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?)
<table>
<thead>
<tr>
<th>Is:</th>
<th>Isn’t:</th>
</tr>
</thead>
<tbody>
<tr>
<td>software design</td>
<td>building computers</td>
</tr>
<tr>
<td>algorithms</td>
<td>hardware focused</td>
</tr>
<tr>
<td>theory of computing</td>
<td>a traditional “science”</td>
</tr>
<tr>
<td>mathematical proofs</td>
<td>information technology</td>
</tr>
</tbody>
</table>
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 |ˈsiːns|
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.
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**: tap/jump
- **stopped**: hit ground
- **dead**: tap/restart

States:
- running
- jumping
- stopped
- dead

Transitions:
- tap/jump: from running to jumping
- hit ground: from jumping to stopped
- tap/restart: from dead to running
- off screen: from stopped to dead
- run off ground/fall: from running to dead
Infinite Runner FSM

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

Start

obstacle: tap/restart
hit ground: tap/jump
run off ground/fall
miss ground/fall
off screen

no obstacle / move forward
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 clear / go north
- North blocked
- South clear / go south
- 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 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