6.1000
Intro to Programming and Computer Science

6.1000 LECTURE 1

New courses!

- **6.1000**
  - 12-units full semester
  - Blended material from former half-semester 6.100A and 6.100B
- **6.100A**
  - 6-units full semester
  - Formerly called 6.100L
  - Former 6.100A material stretched
- **6.100B**
  - 6-units full semester
  - Former 6.100B material stretched
- See details
  - https://www.eecs.mit.edu/changes-to-6-100a-b-l/

Planning ahead

- **6.1010** next spring will require full 12 units pf 6.100
  - 6.1000 or (6.100A + 6.100B)
- **6-3 (CS)** and **6-4 (AI+D)** majors require full 12 units
- Several other majors require full 12 units, too
  - 9, 12, 16, 20, 22, 1-12, etc.
- Reasons for taking 6.100A now
  - **Inaccurate**
    - 6.100A is enough for 6.101 next spring
    - 6.100A is enough for 6-3 and 6-4 majors
    - take 6.100A now and 6.100B in second half of term
  - **Valid**
    - No prior programming experience
    - Planning to major in 6-5 (EE) or other degree that requires only 6.100A
    - Can only fit 6 units into credit limit

Course logistics

- All materials on website https://introcomp.mit.edu/fall25
- Canvas only for announcements
  - Don't message us there
- Recitations are optional, go to any section
  - No 10 am section in 37-212
  - Ignore registrar's assignments
- Contacting us
  - Office hours – in person
  - Piazza – public questions only
  - Email – 6.1000-staff@mit.edu

Class structure

- Mix of slides, blackboard, code, exercises
- Take notes on paper, tablet, or keyboard
- Bring laptop for interactive coding
- Screens must be focused on class material

Assignments

- Low weight on finger exercises and problem sets
  - Use them to learn!
- **Ask for help early!**
  - Office hours
    - 20+ teaching assistants
    - 40+ lab assistants
    - Instructor office hours
  - **Student Support Services S^3**
- Pset extensions
  - Email 6.1000-staff@mit.edu on due date to get 24-hour extension

Collaboration and AI

- Goal is to develop your coding independence and judgment
  - How does the code work?
  - Does it actually work?
  - Could I have written it myself?
  - How well-written is the code?
- Human collaboration policy
  - Free to discuss problem interpretation and solution strategy
  - **Write your own code**
- AI usage policy
  - **Treat it like a human partner**

Let’s dive in!

What is computation? [BOARD]

- Square roots example
  - math vs computation
- Algorithm
  - input
  - output
  - operations
  - rules for when to apply operations

Numeric objects and operations [BOARD]

- Labeled groups of bits
- Arithmetic
  - `+  -  *  /  //  %  **`
- Comparisons
  - `==  !=  <  <=  >=  >`
- Object types
  - `int  float  bool`
- Order of operations
  - https://docs.python.org/3/reference/expressions.html#operator-precedence
- Demonstration
  - directly evaluate in Python's Read-Eval-Print Loop (REPL)
  - run Python on code.py, `print()` results

Variables [BOARD + CODE]

- Compound interest example
- Variables are names/labels that point to object
  - make structure of expressions more readable
  - maintain references to intermediate results
- Operations on `bool` types
  - `and  or  not`

Strings [BOARD + CODE]

- `str` type
  - double vs single quotes
- Binary operations
  - concatenation, repetition, substring, comparison
- Indexing and slicing
  - 0-based indexing
  - negative indices
- Method operations
  - https://docs.python.org/3/library/stdtypes.html#string-methods
- f-strings
  - “drop-in” syntax for composing strings
  - reference: https://fstring.help/cheat/

So far

- Demonstrated Python's capabilities as a fancy calculator
- Can't program our square root-finding algorithm yet
- More programming features next week
- Pset 1 won't be released until after Lecture 2

What to do now

- Work on Lecture 1 finger exercises soon
- Read course info pages
- Install Python
- Experiment with mechanics of basic types and variables

- Office hours start tomorrow
- Recitation on Friday
  - explain some mechanics of coding environments
  - files and folders, terminals