6.100
Intro to Programming and Computer Science

6.100 LECTURE 1

SPRING 2026

Course instructors

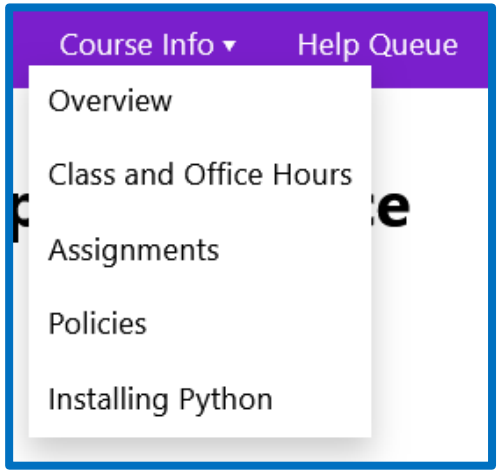
- **6.100**
  - Andrew Wang
- **6.100A**
  - Ana Bell
- **6.100B**
  - John Guttag
- **Contacts for 6.100**
  - 6.100-staff@mit.edu
  - 6.100-instructors@mit.edu

(Relatively) New courses!

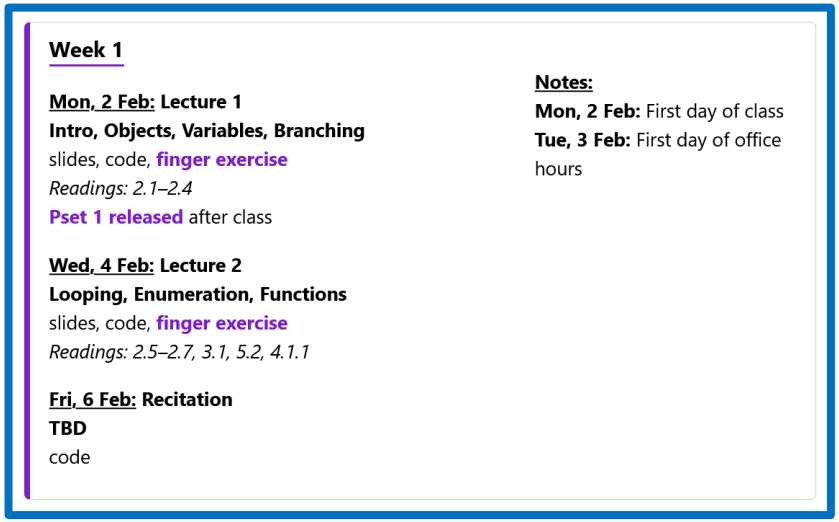
- **6.100**
  - 12-units full semester
  - Blended material from former half-semester 6.100A and 6.100B
- **6.100A**
  - 6-units full semester
  - Formerly called 6.100L
  - Former 6.100A material stretched
- **6.100B**
  - 6-units full semester
  - Former 6.100B material stretched
- See details
  -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

Course logistics

- All materials on website: https://introcomp.mit.edu/spring26
- Read **Course Info** pages



- **Assignments** are linked to from homepage calendar



Week 1

Mon, 2 Feb: Lecture 1
Intro, Objects, Variables, Branching
slides, code, **finger exercise**
Readings: 2.1–2.4
Pset 1 released after class

Wed, 4 Feb: Lecture 2
Looping, Enumeration, Functions
slides, code, **finger exercise**
Readings: 2.5–2.7, 3.1, 5.2, 4.1.1

Fri, 6 Feb: Recitation
TBD
code

Notes:
Mon, 2 Feb: First day of class
Tue, 3 Feb: First day of office hours

- Contacting us
  - Office hours – in person
  - Piazza – public questions only
  - Email – 6.100-staff@mit.edu

Course structure

- First third
  - **lists, functions, simulation, data analysis**
- Middle third
  - **dictionaries, graphs**
- Last third
  - **combinatorial optimization, classes**

- **Exams** are the largest factor determining your grade
- **Psets** and **checkoffs** are for learning and keeping up

Expectations

- Typical week
  - review **pre-lecture code** on Sunday/Tuesday
  - attend Monday/Wednesday **lecture**
  - complete **finger exercises** after each lecture
  - attend Friday **recitation**
- **Name cards**
  - optional to display during class
  - will help staff to learn names
- **Muddy cards**
  - anonymous survey of what isn't clear in lecture
  - will review at beginning of next lecture
- Collaboration and AI policy
  - **"Write your code yourself."**

Studying tips

- Two big themes
- **Be clear on the mechanics of how the code works**
  - How does the computer interpret the code?
- **Reflect on how code is designed**
  - How to break problems down?
  - Are there alternative ways of achieving the same goal?
- Expect some independence in 6.100
  - **Study lecturer/recitation on your own,** not enough time in class to go over all details
  - ***Ask for help soon if needed***
  - **Attend instructor office hours** or schedule a meeting

Let’s dive in!

Objects and types

- Computation is about **transforming data from input to output**
- What data can we represent?

- Data is represented in computer memory as **sequences of bits**
- Modern programming languages abstract away these sequences into **objects**
- Objects are classified into **types**

Numeric objects and operations

- Numeric types
  - **`int`** **`float`**
- Arithmetic
  - **`+`** **`-`** **`*`** **`/`** **`//`** **`%`** **`**`**
  - always produce new **`int`**s or **`float`**s
- Comparisons
  - **`==`** **`!=`** **`<`** **`<=`** **`>=`** **`>`**
  - always produce a new **`bool`** object
- Order of operations
  - https://docs.python.org/3/reference/expressions.html#operator-precedence
- Additional operations
  - **`round(), abs(), min(), max()`**
  - convert types with **`int(), float()`**

Variables

- Long expressions get unwieldy to write
  - also waste computation
  - would like to save and retrieve **intermediate results**
- **Variables** are simply names that reference (i.e., point to) objects
  - **variables only ever point to objects,** not other variables!
  - objects are stored in the **heap**
  - variables are stored on the **stack**
  - **syntax:** ***`variable = expression`***
- Functionally, variables save computation and make expressions easier to read
- **Example:** add deposit to bank account while accumulating interest
- Combined arithmetic and assignment
  - this statement: ***`variable += expression`***
  - is equivalent to: ***`variable = variable + expression`***
  - works with **`+=, -=, *=, /=,`** etc.

Branching

- Programs composed of only sequences of expressions can't react to input
- **Branching** allows us to specify alternate execution paths depending on Boolean conditions
  - ```
    if condition:
        statement
        ...
    else:
        statement
    ```

Indentation matters in Python!

- **Example:** account behavior when over-withdraw funds from bank account
  - pay overdraft fee
  - stop accumulating interest
- With variables and **`bool`**-type operations, can express complex yet redable conditions
  - operations on **`bool`** objects: **`and`** **`or`** **`not`**
- With nested conditionals, look for opportunities to reduce indentation, but be careful!
    ```

Strings

- Objects of type **`str`** represent sequences of characters
  - single character is a **`str`** of length 1
- Indexing and slicing
  - for **`str`** of length **`n`**, indices go from **`0`** to **`n-1`**
  - think of as half-open interval **`[0, n)`**
  - slicing syntax is **`[start : stop : step]`**
- **`+`** (concatenation) and **`*`** (repetition)
- Binary operations that yield **`bool`**s
  - substring testing **`in`**
  - comparison **`==`** **`!=`** **`<`** **`<=`** **`>=`** **`>`**
- Additional operations
  - **`len()`**
  - **`str`** methods: **`.find(), .replace(), .strip(), ...`**
  - https://docs.python.org/3/library/stdtypes.html#string-methods

So far

- Demonstrated Python's capabilities as a fancy calculator
- Added branching ability to encode multiple possible computations in one program
- Can't write all possible programs yet
- More features on Wednesday

Next steps

- Logistics
  - read course info pages
  - install Python
  - check recitation section
  - office hours start tomorrow
- Material
  - do Lecture 1 finger exercise
  - start reading Pset 1
  - review Lecture 1 code, links to Python docs
  - read pre-lecture code for Lecture 2, posted tomorrow morning