Fundamentals of Programming

Spring 2023

rec00intro

6.101 Goals

Our goals involve helping you develop your programming skills, in multiple aspects:

- **Programming**: analyzing problems, developing plans
- **Coding**: translating plans into Python
- **Debugging**: developing test cases, verifying correctness, finding and fixing errors

6.101 Goals

So we will spend time discussing:

- high-level design strategies
- ways to manage complexity
- details and "goodies" of Python
- a mental model of Python's operation
- testing and debugging strategies

...but discussion only goes so far.

6.101 Pedagogy

Learning to program is a lot like learning a musical instrument or a sport.

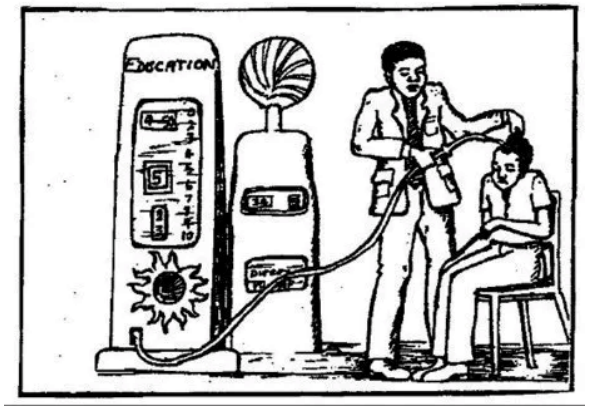
How does one learn those things?

Just like music/sports, practice is key!

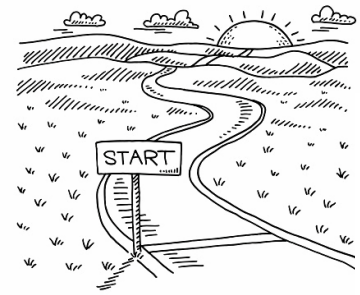
To improve as a programmer, you have to program.

And 6.101 asks you to program...a lot!

- Labs give opportunities to practice new techniques/skills to solve problems.
- Recitations equip you with tools useful for attacking those problems.
- Office hours give opportunities to receive expert feedback.



A week in the life of 6.101



reading n released → reading n due

lab n released → lab n prelim → lab n due → (makeup) → lab n cutoff

recitation: lab n midpoint

recitation: lab n wrap-up

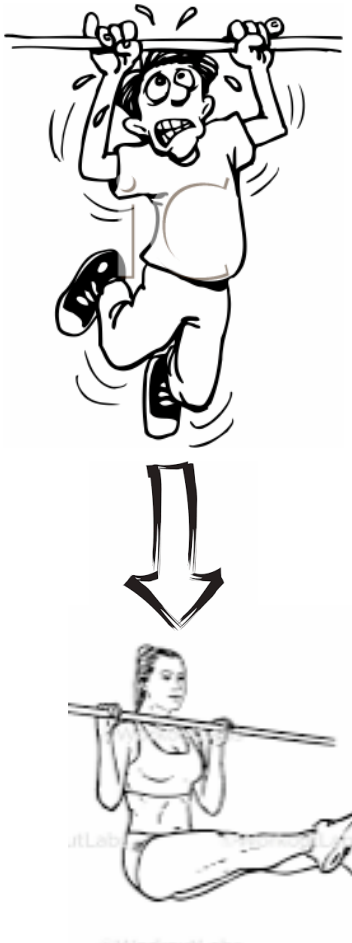
W R F Sa Su M T W R F Sa Su M

reading $n+1$ released → reading $n+1$ due

lab $n+1$ released → ...

Getting the most out of 6.101

- Recitations:
  - Step 1: Come to lecture/recitation, participate, and turn-in your answers!
  - Take notes *in your own words* and review them later
  - Ask questions! We want to have a conversation.
- Labs:
  - Start early (labs are week-long assignments)
  - Formulate a plan before writing code
    - Try to understand the problem thoroughly before writing code
    - When things go wrong, step away from the code and revisit the plan
  - Work through problems on your own
  - Ask for help when you need it!
    - Labs are intentionally challenging
    - Bugs are a natural part of life
    - Lots of opportunities for help (office hours / e-mail)



Academic Integrity

Our goal is that every student develops these skills throughout the course.

Learning a new skill is mostly a solitary process.

Collaborating too closely with others (or outsourcing pieces to other students or the web or ChatGPT) can rob you of an opportunity to develop those skills in yourself.

Please read our policies about academic integrity carefully.

https://py.mit.edu/spring23/info/academic_integrity

Using 6.101 Resources

We are going to push your limits in 6.101 because we have high expectations for you!

6.101 expects most students will ask for help, which is why we provide:

- Open lab hours – help and feedback with the lab (programming, coding, debugging, style), conceptual questions (readings, recitation, quiz material)
- 6.101-help@mit.edu – logistical questions, policy questions, website errors, also some technical help

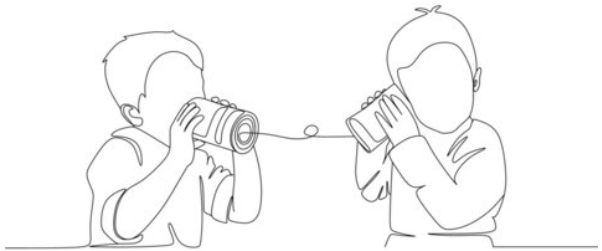
Help is not a shortcut!

- Recitations and readings are designed to prepare you for the labs and quizzes
  - If the queue gets busy, we may prioritize helping students who have come to recitation and completed readings first
- Try hard to do it yourself → seek help after making a reasonable attempt
- Help us help you: come prepared with a specific question and what you have tried

How to communicate with 6.101

Course website:

https://py.mit.edu



Email:

6.101-help@mit.edu

- for general technical or logistical help

6.101-personal@mit.edu

- for sensitive requests that involve personal or medical situations
- reaches just the instructors and course assistant

Why should your code should be readable?

- For yourself
  - You may need to remember what you did… days, weeks or years later!
  - You may need to debug it
  - You may need to modify, update or upgrade it
- For others
  - Most programs are a group effort… you want your colleagues to be friends!
  - If your code goes viral… others may want to understand, use, port, or update your code.
- For elegance
  - The process of making your code readable will lead to improved code
  - Code is your creative output… let it be cool

The Importance of Style

You want to communicate an idea.

Your reader has limited resources (time, concentration) they can spend.

Make that communication as efficient as possible.

write/Speak iN odd syntax, you could, with fonts unusual.

But that would mean your reader is wasting their effort parsing the message instead of understanding it.

Make it easy for your reader to understand your message.

Style, Clarity, and Good Coding Conventions

Code communicates ideas –

to the computer,

to other programmers,

… including ourselves!

Reading code is recognizing patterns

Code conventions create patterns

Use patterns everyone knows ⇒ code is easier to understand

Human Cognitive Limitations and 6.101

Working memory is limited, 7±2 digits (Miller, 1956).

n ideas in your head at once, and ***n*** is small.

Engineering tools create sets of ideas that work together correctly.

Then details put aside, working memory filled with other ideas.

If unlimited mental capacity, tools not needed. But we do have limits.

In 6.101 you will learn ways to overcome those limits so that we can write code to do great things.

How to design a good program

- Don't Repeat Yourself (DRY)
  - Avoid multiple fragments of code that describe redundant logic.
- Names Matter
  - Choose concise, descriptive names for functions, parameters, and other variables.
- Documentation Matters
  - Use docstrings and comments to describe assumptions and document non-obvious features of the code.
- Generality Wins
  - Define logic, functions, and programs as generally as possible.
- Plan for Change
  - Where possible, make programs that are (relatively) easy to change, should the need arise.