Welcome to 6.101!

You are currently signed up as a listener in 6.101. While listeners have full access to the assignments, etc, they are not assigned recitation sections and are not allowed to make use of open lab hours or take quizzes.

Table of Contents

1) Subject Overview

Welcome to 6.101! We're really looking forward to working with all of you, and we hope that you're excited as well. Together, we're about to embark on the next leg of the journey of learning computer programming, one of the most interesting, fun, and practically-useful skills that anyone can possess in the modern world. Every semester, we have a diverse set of students in 6.101, not only in terms of background but in terms of goals for the future as well. Some of you are planning for futures as software engineers. Some of you are not interested in that kind of path but recognize the utility in being able to make computers work for you on problems in your areas of interest. Some of you find programming fun and interesting in and of itself. You may fall into one or more of those categories, or maybe you have other reasons for taking 6.101 this semester. Regardless, if your future goals and plans involve computation in any way, we hope that 6.101 will be a fun and valuable experience for you!

6.101 is the second programming subject in a sequence (6.100, 6.101, 6.102). Our focus is on adjusting our programming practices as we move from relatively small programs, run in an artificial environment, to more complex programs designed to be run on your own computer, working with real data, and producing meaningful results. As our programs grow in terms of scope and scale, we'll encounter new concerns that weren't as important when working with smaller programs (or maybe didn't even exist at that scale), and so we'll need to adjust our approach. And we'll focus on three separate but related parts of the programming process:

- **programming:** analyzing and decomposing problems, developing plans
- **coding:** translating those plans into correct, efficient, idiomatic, understandable Python code
- **debugging:** developing test cases, verifying correctness, and finding and fixing the errors that inevitably happen

In order to facilitate growth in these areas, we'll spend time on a number of different topics in 6.101, including:

- high-level design strategies,
- ways of managing complexity,
- intermediate and advanced features of Python,
- a mental model of Python's operation,
- testing and debugging strategies,
- and more!

Of course, just talking about these things is not enough. Deliberate practice is key to developing as a programmer, just like with any new skill (think of learning a musical instrument, a sport, or a second language). To improve as a programmer, it helps to:

- watch how experienced programmers approach problems,
- write programs of your own, and
- receive feedback on your work from more experienced programmers.

And the good news is that, by design, 6.101 will provide you with a *lot* of opportunities to do all of these things.

It's worth mentioning, too, that while learning a new skill is generally very rewarding, not every part of the process will be particularly enjoyable. Learning a new skill involves a lot of time, dedication, patience, hard work, and struggle. And so it is with programming. It is natural to encounter difficulties along the way; and sometimes those difficulties can cause us to feel unsure, vulnerable, intimidated, insecure, and not very bright. Those feelings are perfectly natural (even for very experienced programmers), and they do not mean that you cannot ultimately accomplish your goals as they relate to programming (whatever they may be); they just mean that there is more work to be done. So along the way, if and when those feelings creep in, take a deep breath and remember that those kinds of obstacles are normal, that all experienced programmers have been there, and that there are lots of people here to support you along the way.

We're excited to work alongside you as you take the next steps of this journey, and we hope that you're excited to get underway as well!

Learning Objectives

At some level, 6.101 is about creating computer programs and using computation to solve interesting problems. And throughout 6.101, you will improve as a programmer. But we also hope to explore deeper conceptual questions, so that you'll not only be able to make the computer work for you, but you will be able to reason about how to structure your programs so that they are correct, efficient, and understandable, particularly as the programs you are writing grow in terms of scale and complexity.

Specifically, we have several goals for your learning in 6.101, all of which we think are broadly useful:

- **Programming Objectives**

  After taking 6.101, you should be able to...

  - ...design and implement small- and medium-sized Python programs in idiomatic style, including using some advanced features of the Python language.
  - ...implement, test, and debug Python programs in your preferred programming environment.
  - ...use programming as a tool to solve problems in other domains.
  - ...use the command line to interact with your computer.

- **Conceptual Objectives**

  After taking 6.101, you should be able to...

  - ...deconstruct a large problem into smaller pieces that can be planned out, implemented, and debugged independently.
  - ...reason about the behavior of small- and medium-sized Python programs without using a computer, through the use of environment diagrams and other tools.

- ...understand the tradeoffs associated with various algorithms and data structures, in terms of efficiency and correctness.
- ...predict some edge cases and failure modes when designing a program, before writing any code.

2) General Course Information

Please read these three pages now:

- Basic Structure and Schedule
- Grading Policies
- Getting Help

Familiarizing yourself with these policies is essential, because 6.101 is different from other courses you've taken, and it changes over time. Structures or policies that you or your peers remember from previous semesters may no longer hold true.

3) Academic Integrity and Cheating

Please read this page now:

- Academic Integrity

There are many ways to get help in 6.101, but the work you submit must be your own. This page explains what this means with respect to this course. **Make sure you read and understand this policy**, because your intuitions about or prior experience with collaboration may not be aligned with this course's expectations, and violations of academic integrity can have serious consequences.

4) Lab Assignments

Each week in 6.101 centers around one large programming assignment, which we refer to as a "lab." The labs are substantial assignments, and we expect that, in the average week, the average student will spend about 10 hours on the 6.101 lab. They are also intended to be a challenge, to push you a little bit. Some of them will require a bit of struggle. There will be places where you'll need to stop, check your work, reevaluate your assumptions and the work you've done, and ask questions. We set things up this way not out of a sadistic desire to watch students suffer, but because we believe that striving for and working toward a difficult task is one of the best ways to learn.

Our goal in 6.101 is not about perfection, but rather about steady growth. There may be times when progress seems slow, but our hope is that, when you have time for reflection, you will see that the hard work you're putting in is paying off, and that you've grown a lot through the experience!

4.1) This Week

A typical lab releases on a Friday and comes due the following Friday. Our first week of classes is condensed, but we do have a lab assignment for the first few days of class: lab 0, which is intended to help you get used to the flow of 6.101 labs and with the infrastructure you'll use to work on them. Given the condensed timeframe, lab 0 is somewhat shorter than a typical lab.

5) Setting Yourself Up For Success in 6.101 and Beyond

A common question near the start of any new semester is: what can I do to make sure that I get the most out of the subjects I'm taking this semester? In this section, we'll do our best to answer that question as it pertains to 6.101; and, given that 6.101 has a number of moving parts, we'll try to offer specific advice for maximizing the utility of each of those parts.

We believe that following this advice will not only help you do well in 6.101, but it will also help you maximize your learning over the semester and your ability to make use of the 6.101 content in the future.

Readings

- **Reserve Time to Read.** Find time on your schedule to do the reading before it's due. Mark it on your calendar the same way you would a lecture time.
- **Read the Words**. Focus and read the words. For tricky parts that aren't clear at first, it may help to reread the paragraph, or to read it aloud. If reading on a screen is distracting, print the reading and read it on paper.
- **Engage with the Questions**. Every reading has some questions embedded in it (including this one!). Think carefully about each question and try to answer it yourself first. You can try these reading questions any number of times, but, if you find yourself guessing randomly or trying all possible answers, then pause, step back, and ask for help. You do not need to get all the questions correct, but random guessing is unlikely to count for reading credit.
- **Try the Code**. Most readings (not this one) will include example Python code. Try it out in your own editing environment! The best way to try out a piece of code is to read it carefully first, understand and internalize how it works, and then recreate it yourself in your code editor, out of your own head, without looking at the reading. Look back at the reading only if you can't get it to work, and pay particular attention to the bits you missed. This gives you practice similar to creating the code yourself, in a way that direct copying does not. Benjamin Franklin used this technique to improve his writing, and it works in other disciplines too.

Recitations

- **Come to Recitation**. Since 6.101 has no lectures, recitation sections are the classroom experience for this course. It's hard to succeed without going to class, so put the section times on your calendar and make sure to come.
- **Laptops Closed**. Your laptop is a great tool for programming but unfortunately also a huge opportunity for multitasking and distraction. The price of that distraction is paid not only by you but by all those around you who can see your screen. In one study, multitasking led to an 11% drop in score on a subsequent test. But, for people *without laptops*, merely having a laptop multitasker in their field of view led to a 17% drop in test score. That's 1.5 letter grades worse! Laptop distraction is like secondhand smoking. It hurts other people, not just you. So please keep your laptop closed and put away during recitation.
- **Take Notes**. Bring paper or a notebook, and take notes! Taking notes is a great way to engage your brain, focus your attention, and help you remember afterward what you learned. When taking notes, use your own words, and try to emphasize the big ideas; then, return later to review and fill in the gaps.

- **Ask Questions**. When you are confused by something, or, if you think you see a bug in the instructor's code, please raise your hand and ask. Chances are that other people have the same confusion that you do. Please help make the class better by asking questions!

Labs

- **Start Early**. Read the lab handout on the day it is released, and start thinking about how you might approach each part. Reading the entire handout before starting is very important, because it gives you the big picture and goals that the different parts of the lab are moving toward. You may even find that, even if you don't start *actively* working on a lab early in the week, reading through it right away will allow your subconscious brain to be thinking about it as you relax or work on other things during the week. Starting early will also give you more time to *iterate* on the lab if necessary, because sometimes your first approach won't be the right one and you'll need to change it, and you may need time to improve its performance and readability too.

- **Plan Before Coding**. Before opening your code editor, plan out how your function might work on paper or a whiteboard. Write your algorithm first as pseudocode (English that is close to code). Look for subparts of the problem that might be pulled out as helper functions and give them good names.

- **Work On Your Own**. 6.101 is not about group software engineering; it is intended instead to develop your individual self-efficacy at programming. In order to maximize your own development, try to tackle the labs yourself and only ask for help on a component of a lab after you have spent some time working on it individually.

- **Ask for Help When You Need It**. Of course, it's very likely that there will be times when you will encounter problems that you aren't able to tackle on your own. Spinning your wheels for too long while being stuck is not a very efficient way to learn, either. When this happens, it's a good idea to ask for help. For debugging questions, in-person help is best.

6) Policy Questions

In addition to prose and examples, each reading will also contain several questions that you can use to make sure you're following along. For this week, answer the following questions to make sure you understand 6.101's course policies. And if you have any questions, please feel free to reach out via open lab hours or the mailing lists!

The lateness penalty will be automatically forgiven for how many assignments over the semester?

3

Submit

You have submitted this assignment 2 times.

Solution: 3

Explanation:

Review Grading Policies if you are not sure about the answer to this question.

Will additional lateness accommodations / extensions be given for students who add the class late?

No

Submit

You have submitted this assignment 1 time.

Solution: No

Explanation:

Review Grading Policies if you are not sure about the answer to this question.

To what e-mail address should you send questions about *technical content*?

6.101-help@mit.edu

Submit

You have submitted this assignment 1 time.

Solution: 6.101-help@mit.edu

Explanation:

Review Getting Help if you are not sure about the answer to this question.

To what e-mail address should you send questions about *personal or medical issues*?

6.101-personal@mit.edu

Submit

You have submitted this assignment 1 time.

Solution: 6.101-personal@mit.edu

Explanation:

Review Getting Help if you are not sure about the answer to this question.

Is it okay to get help from a friend who took this class in a previous semester, as long as the help is at a high level and doesn't include actual code? not okay

Submit

You have submitted this assignment 2 times.

Solution: not okay

Explanation:

Review Academic Integrity if you are not sure about the answer to this question.

Is it okay to work side-by-side with other students in the class, talking about the code you're writing, as long as nobody speaks or texts any code longer than one line? not okay

Submit

You have submitted this assignment 1 time.

Solution: not okay

Explanation:

Review Academic Integrity if you are not sure about the answer to this question.

Your code has a bug that you can't figure out. What's the best way to get help with debugging it?

go to open lab hours and talk to a staff member

Submit

You have submitted this assignment 1 time.

Solution: go to open lab hours and talk to a staff member

Explanation:

Review Getting Help if you are not sure about the answer to this question.

You solved all the problems on lab 0, you're passing all the test cases, and you submitted your code as explained in the lab handout. Are you done with lab 0? No

Submit

You have submitted this assignment 2 times.

Solution: No

Explanation:

Lab 0 is one of the labs that requires a checkoff conversation with a staff member. Review Grading Policies for more about checkoffs, and don't forget the checkoff step at the end of these labs.