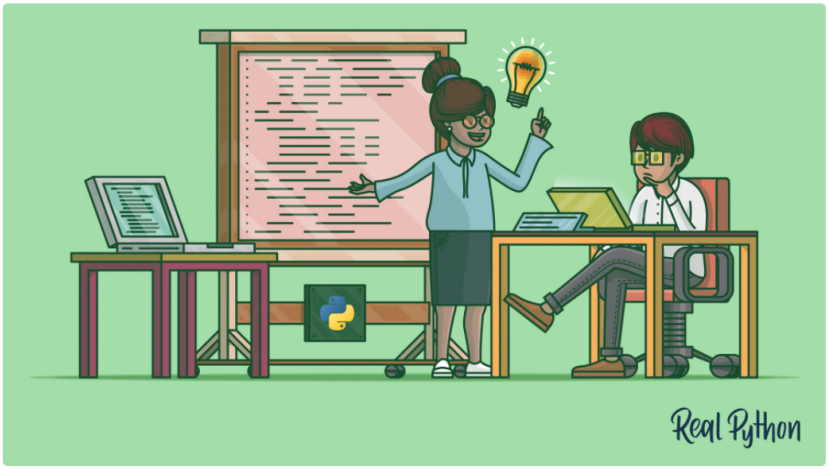

Python Practice Problems: Get Ready for Your Next Interview

by Jim Anderson Sep 21, 2020 26 Comments best-practices intermediate

Table of Contents

Are you a Python developer brushing up on your skills before an interview? If so, then this tutorial will usher you through a series of **Python practice problems** meant to simulate common coding test scenarios. After you develop your own solutions, you'll walk through the *Real Python* team's answers so you can optimize your code, impress your interviewer, and land your dream job!

In this tutorial, you'll learn how to:

- **Write code** for interview-style problems
- **Discuss your solutions** during the interview
- Work through **frequently overlooked details**
- Talk about **design decisions** and trade-offs

Table of Contents

- Python Practice Problem 1: Sum of a Range of Integers
- Python Practice Problem 2: Caesar Cipher
- Python Practice Problem 3: Caesar Cipher Redux
- Python Practice Problem 4: Log Parser
- Python Practice Problem 5: Sudoku Solver
- Conclusion

This tutorial is aimed at intermediate Python developers. It assumes a basic knowledge of Python and an ability to solve problems in Python. You can get skeleton code with failing unit tests for each of the problems you'll see in this tutorial by clicking on the link below:

Download the sample code: Click here to get the code you'll use to work through the Python practice problems in this tutorial.

Each of the problems below shows the file header from this skeleton code describing the problem requirements. So download the code, fire up your favorite editor, and let's dive into some Python practice problems!

Python Practice Problem 1: Sum of a Range of Integers

Let's start with a warm-up question. In the first practice problem, you'll write code to sum a list of integers. Each practice problem includes a problem description. This description is pulled directly from the skeleton files in the repo to make it easier to remember while you're working on your solution.

You'll see a solution section for each problem as well. Most of the discussion will be in a collapsed section below that. Clone that repo if you haven't already, work out a solution to the following problem, then expand the solution box to review your work.

Problem Description

Here's your first problem:

> **Sum of Integers Up To n** (`integersums.py`)
>
> Write a function, `add_it_up()`, that takes a single integer as input and returns the sum of the integers from zero to the input parameter.
>
> The function should return 0 if a non-integer is passed in.

Remember to run the unit tests until you get them passing!

Problem Solution

Here's some discussion of a couple of possible solutions.

Note: Remember, don't open the collapsed section below until you're ready to look at the answer for this Python practice problem!

Solution for Sum of a Range of Integers Show/Hide

Python Practice Problem 2: Caesar Cipher

The next question is a two-parter. You'll code up a function to compute a Caesar cipher on text input. For this problem, you're free to use any part of the Python standard library to do the transform.

Hint: There's a function in the `str` class that will make this task much easier!

Problem Description

The problem statement is at the top of the skeleton source file:

> **Caesar Cipher** (`caesar.py`)
>
> A Caesar cipher is a simple substitution cipher in which each letter of the plain text is substituted with a letter found by moving `n` places down the alphabet. For example, assume the input plain text is the following:
>
> ```
> abcd xyz
> ```
>
> If the shift value, `n`, is 4, then the encrypted text would be the following:
>
> ```
> efgh bcd
> ```
>
> You are to write a function that accepts two arguments, a plain-text message and a number of letters to shift in the cipher. The function will return an encrypted string with all letters transformed and all punctuation and whitespace remaining unchanged.
>
> **Note:** You can assume the plain text is all lowercase ASCII except for whitespace and punctuation.

Remember, this part of the question is really about how well you can get around in the standard library. If you find yourself figuring out how to do the transform without the library, then save that thought! You'll need it later!

Problem Solution

Here's a solution to the Caesar cipher problem described above.

> **Note:** Remember, don't open the collapsed section below until you're ready to look at the answers for this Python practice problem!

Solution for Caesar Cipher Show/Hide

Now that you've seen a solution using the Python standard library, let's try the same problem again, but without that help!

Python Practice Problem 3: Caesar Cipher Redux

For the third practice problem, you'll solve the Caesar cipher again, but this time you'll do it without using `.translate()`.

Problem Description

The description of this problem is the same as the previous problem. Before you dive into the solution, you might be wondering why you're repeating the same exercise, just without the help of `.translate()`.

That's a great question. In normal life, when your goal is to get a working, maintainable program, rewriting parts of the standard library is a poor choice. The Python standard library is *packed* with working, well-tested, and fast solutions for problems large and small. Taking full advantage of it is a mark of a good programmer.

That said, this is not a work project or a program you're building to satisfy a need. This is a learning exercise, and it's the type of question that might be asked during an interview. The goal for both is to see how you can solve the problem and what interesting design trade-offs

you make while doing it.

So, in the spirit of learning, let's try to resolve the Caesar cipher without `.translate()`.

Problem Solution

For this problem, you'll have two different solutions to look at when you're ready to expand the section below.

> **Note:** Remember, don't open the collapsed section below until you're ready to look at the answers for this Python practice problem!

Solutions for Caesar Cipher Redux Show/Hide

Now that you've written the Caesar cipher three different ways, let's move on to a new problem.

Python Practice Problem 4: Log Parser

The log parser problem is one that occurs frequently in software development. Many systems produce log files during normal operation, and sometimes you'll need to parse these files to find anomalies or general information about the running system.

Problem Description

For this problem, you'll need to parse a log file with a specified format and generate a report:

> **Log Parser** (`logparse.py`)
>
> Accepts a filename on the command line. The file is a Linux-like log file from a system you are debugging. Mixed in among the various statements are messages indicating the state of the device. They look like this:
>
> ```
> Jul 11 16:11:51:490 [139681125603136] dut: Device State: ON
> ```
>
> The device state message has many possible values, but this program cares about only three: `ON`, `OFF`, and `ERR`.
>
> Your program will parse the given log file and print out a report giving how long the device was `ON` and the timestamp of any `ERR` conditions.

Note that the provided skeleton code doesn't include unit tests. This was omitted since the exact format of the report is up to you. Think about and write your own during the process.

A `test.log` file is included, which provides you with an example. The solution you'll examine produces the following output:

Shell

```
$ ./logparse.py test.log
Device was on for 7 seconds
Timestamps of error events:
   Jul 11 16:11:54:661
   Jul 11 16:11:56:067
```

While that format is generated by the Real Python solution, you're free to design your own format for the output. The sample input file should generate equivalent information.

Problem Solution

In the collapsed section below, you'll find a possible solution to the log parser problem. When you're ready, expand the box and compare it with what you came up with!

> **Note:** Remember, don't open the collapsed section below until you're ready to look at the answers for this Python practice problem!

Solution for Log Parser Problem Show/Hide

That's it for the log-parsing solution. Let's move on to the final challenge: sudoku!

Python Practice Problem 5: Sudoku Solver

Your final Python practice problem is to solve a sudoku puzzle!

Finding a fast and memory-efficient solution to this problem can be quite a challenge. The solution you'll examine has been selected for readability rather than speed, but you're free to optimize your solution as much as you want.

Problem Description

The description for the sudoku solver is a little more involved than the previous problems:

> **Sudoku Solver** (`sudokusolve.py`)
>
> Given a string in SDM format, described below, write a program to find and return the solution for the sudoku puzzle in the string. The solution should be returned in the same SDM format as the input.
>
> Some puzzles will not be solvable. In that case, return the string "Unsolvable".
>
> The general SDM format is described here.
>
> For our purposes, each SDM string will be a sequence of 81 digits, one for each position on the sudoku puzzle. Known numbers will be given, and unknown positions will have a zero value.
>
> For example, assume you're given this string of digits:
>
> ```
> 004006079000000602056092300078061030509000406020540890007410920105000000840600l0
> ```
>
> The string represents this starting sudoku puzzle:
>
> ```
> 0 0 4   0 0 6   0 7 9
> 0 0 0   0 0 0   6 0 2
> 0 5 6   0 9 2   3 0 0
>
> 0 7 8   0 6 1   0 3 0
> 5 0 9   0 0 0   4 0 6
> 0 2 0   5 4 0   8 9 0
>
> 0 0 7   4 1 0   9 2 0
> 1 0 5   0 0 0   0 0 0
> 8 4 0   6 0 0   1 0 0
> ```
>
> The provided unit tests may take a while to run, so be patient.
>
> **Note:** A description of the sudoku puzzle can be found on Wikipedia.

You can see that you'll need to deal with reading and writing to a particular format as well as generating a solution.

Problem Solution

When you're ready, you can find a detailed explanation of a solution to the sudoku problem in the box below. A skeleton file with unit tests is provided in the repo.

> **Note:** Remember, don't open the collapsed section below until you're ready to look at the answers for this Python practice problem!

Solution for Sudoku Solver Show/Hide

That's the end of your Python practice problems adventure!

Conclusion

Congratulations on working through this set of Python practice problems! You've gotten some practice applying your Python skills and also spent some time thinking about how you can respond in different interviewing situations.

In this tutorial, you learned how to:

- **Write code** for interview-style problems
- **Discuss your solutions** during the interview
- Work through **frequently overlooked details**
- Talk about **design decisions** and trade-offs

Remember, you can download the skeleton code for these problems by clicking on the link below:

> **Download the sample code: Click here to get the code you'll use** to work through the Python practice problems in this tutorial.

Feel free to reach out in the comments section with any questions you have or suggestions for other Python practice problems you'd like to see! Also check out our "Ace Your Python Coding Interview" Learning Path to get more resources and for boosting your Python interview skills.

Good luck with the interview!

Mark as Completed

About **Jim Anderson**

Jim has been programming for a long time in a variety of languages. He has worked on embedded systems, built distributed build systems, done off-shore vendor management,

and sat in many, many meetings.

» More about Jim

Each tutorial at Real Python is created by a team of developers so that it meets our high quality standards. The team members who worked on this tutorial are:

 Aldren

 Geir Arne

 Jon

 Joanna

 Jacob

What Do You Think?

Real Python Comment Policy: The most useful comments are those written with the goal of learning from or helping out other readers—after reading the whole article and all the earlier comments. Complaints and insults generally won't make the cut here.

What's your #1 takeaway or favorite thing you learned? How are you going to put your newfound skills to use? Leave a comment below and let us know.

Keep Learning

Related Tutorial Categories: best-practices intermediate

Keep Learning

Related Tutorial Categories: best-practices intermediate