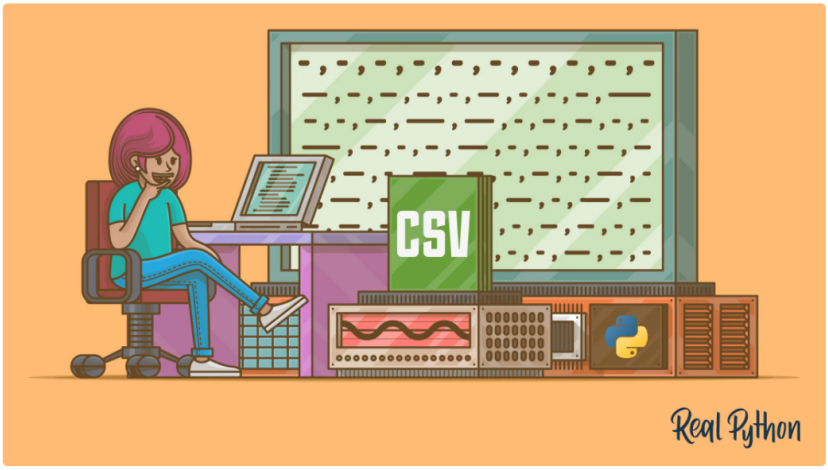


Python Practice Problems: Parsing CSV Files

by Jim Anderson Jun 14, 2021 15 Comments best-practices intermediate

Mark as Completed

Table of Contents

Are you a developer looking for some practice with **comma-separated values (CSV) files** before an upcoming interview? This tutorial will lead you through a series of Python CSV practice problems to help you get ready.

This tutorial is aimed at intermediate Python developers. It assumes a basic knowledge of Python and working with CSV files. Like other practice problem tutorials, each of the problems listed here shows the problem description. You'll see the problem statement first and then have a chance to develop your own solution.

In this tutorial, you'll explore:

- **Writing code** for working with CSV files
- Doing **test-driven development** with pytest
- **Discussing your solutions** and possible enhancements
- The trade-offs between the built-in CSV module and **pandas**

All Tutorial Topics
advanced api basics best-practices community databases data-science devops django docker flask front-end gamedev gui intermediate machine-learning projects python testing tools web-dev web-scraping

Table of Contents
- Python CSV Parsing: Football Scores
- Python CSV Parsing: Weather Data
- Python CSV Parsing: Refactoring
- Python CSV Parsing: pandas
- Conclusion

Mark as Completed
Tweet Share Email

You can get skeleton code with failing unit tests for each of the problems you'll see in this tutorial by clicking the link below:

> **Get the Source Code: Click here to get the source code you'll use** to practice parsing CSV files in this tutorial.

Python CSV Parsing: Football Scores

Your first problem deals with English Premier League team standings. You don't need any special football knowledge to solve this, just Python!

As you work through the problem, try to write more unit tests for each bit of functionality and *then* write the functionality to make the tests pass. This is known as test-driven development, and it can be a great way to show off not only your coding but also your testing chops!

Problem Description

For this round of the problem, stick to the standard library `csv` module. You'll get another shot at it using pandas later. Here's your first problem:

> **Find the Minimum Goal Differential**
>
> Write a program that takes a filename on the command line and processes the contents of a CSV file. The contents will be the end-of-season football standings for the English Premier League. Your program should determine which team had the smallest goal differential that season.
>
> The first line of the CSV file will be column headers, with each subsequent line showing the data for one team:
>
> CSV
> ```
> Team,Games,Wins,Losses,Draws,Goals For,Goals Against
> Arsenal,38,26,9,3,79,36
> ```
>
> The columns labeled `Goals For` and `Goals Against` contain the total number of goals scored for and against each team in that season. (So Arsenal scored 79 goals and had 36 goals scored against them.)
>
> Write a program to read the file, then print the name of the team with the smallest difference in `Goals For` and `Goals Against`. Create unit tests with pytest to test your program.

There is a single unit test supplied in the skeleton code that tests the problem statement that you'll see later. You can add more as you write your solution. There are also two pytest fixtures given:

Python
```python
# test_football_v1.py
import pytest
import football_v1 as fb

@pytest.fixture
def mock_csv_data():
    return [
        "Team,Games,Wins,Losses,Draws,Goals For,Goals Against",
        "Liverpool FC, 38, 32, 3, 3, 85, 33",
        "Norwich City FC, 38, 5, 27, 6, 26, 75",
    ]

@pytest.fixture
```

```
def mock_csv_file(tmp_path, mock_csv_data):
    datafile = tmp_path / "football.csv"
    datafile.write_text("\n".join(mock_csv_data))
    return str(datafile)
```

The first fixture supplies a list of strings that mocks real CSV data, and the second supplies a filename backed by that test data. Each string in the list of strings represents a line of the test file.

> **Note:** The solutions here will have a non-exhaustive set of tests for them that will prove out basic functionality only. For a real system, you would likely want a more complete test suite, possibly making use of parametrization.

Remember that the provided fixtures are only a start. Add unit tests that use them as you design each part of the solution!

Problem Solution

Here's a discussion of the solution that the *Real Python* team arrived at and how the team got there.

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

Solution for Minimum Goal Differential Show/Hide

Now that you've used the Python `csv` module to solve one problem, try again with a similar problem.

Python CSV Parsing: Weather Data

Your second problem will look fairly similar to your first. It might be a good idea to use a similar structure to solve it. Once you've walked through a solution for this problem, you'll read about some ideas for refactoring to reuse code, so keep that in mind as you work.

Problem Description

This problem involves parsing weather data in a CSV file:

> **Highest Average Temperature**
>
> Write a program that takes a filename on the command line and processes the contents of a CSV file. The contents will be a month of weather data, one day per line.
>
> Your program should determine which day had the highest average temperature, where the average temperature is the average of the day's high and low temperatures. This is not normally how average temperature is computed, but it'll work for this demonstration.
>
> The first line of the CSV file will be column headers:
>
> CSV
>
> ```
> Day,MaxT,MinT,AvDP,1HrP TPcn,PDir,AvSp,Dir,MxS,SkyC,MxR,Mn,R AvSLP
> 1,88,59,74,53.8,0,280,9.6,270,17,1.6,93,23,1004.5
> ```
>
> The day number, max temperature, and min temperature are the first three columns.
>
> Write unit tests with pytest to test your program.

As with the football scores problem, there are unit tests supplied in the skeleton code that test the problem statement:

Python

```
# test_weather_v1.py
import pytest
import weather_v1 as wthr

@pytest.fixture
def mock_csv_data():
    return [
        "Day,MxT,MnT,AvT,AvDP,1HrP TPcn,PDir,AvSp,Dir,MxS,SkyC,MxR,Mn,R AvSLP",
        "1,88,59,74,53.8,0,280,9.6,270,17,1.6,93,23,1004.5",
        "2,79,63,71,46.5,0,330,8.7,340,23,3.3,70,28,1004.5",
    ]

@pytest.fixture
def mock_csv_file(tmp_path, mock_csv_data):
    datafile = tmp_path / "weather.csv"
    datafile.write_text("\n".join(mock_csv_data))
    return str(datafile)
```

Again, note that there are two fixtures given. The first supplies a list of strings that mocks real CSV data, and the second supplies a filename backed by that test data. Each string in the list of strings represents a line of the test file.

Remember that the provided fixtures are only a start. Add tests as you design each part of the solution!

Problem Solution

Here's a discussion of what the *Real Python* team arrived at.

> **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 Highest Average Temperature Show/Hide

Python CSV Parsing: Refactoring

The two problems you've looked at so far are quite similar, and the programs to solve them have been quite similar as well. An interesting interview question might be to ask you to refactor these two solutions to find a way to share code and make them more maintainable.

Problem Description

This problem is a little different from the previous two. For this section, take your solutions from the previous problems and refactor them to reuse common code and structures. In a real-world situation, these solutions are small enough that the refactoring effort here is probably not worthwhile, but it does make for a good thought exercise.

Problem Solution

Here's the refactoring that the *Real Python* team arrived at.

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

Solution Refactoring Show/Hide

Python CSV Parsing: pandas

So far, you used the `csv.DictReader` class from the standard library for your solutions, and that has worked well for these relatively small problems.

For larger problems, the pandas package can provide great results with excellent speed. Your last challenge is to rewrite the football program above using pandas.

Problem Description

Here's your final problem for this tutorial. For this problem, you'll rewrite your solution to the football problem using pandas. The pandas solution will likely look different than the solution using just the standard library.

Problem Solution

Here's a discussion of the solution the team arrived at and how they got there.

Note: Remember, don't open the collapsed section below until you're ready to look at the answers for each of the Python practice problems!

Solution for Minimum Goal Differential Using pandas Show/Hide

Conclusion

That's the end of this set of Python CSV parsing practice problems! You've gotten some practice applying your Python skills to CSV files and also got to spend some time thinking about trade-offs you can discuss during an interview. You then looked at refactoring solutions, both in terms of a single problem and also refactoring common code out of two solutions.

In addition to solving these problems, you covered:

- **Writing code** with the `csv.DictReader()` class
- **Using pandas** to solve CSV problems
- **Discussing** your solutions during the interview
- Talking about **design decisions** and trade-offs

You're now ready to face a Python CSV parsing problem and discuss it in an interview! Feel free to reach out in the comments section below with any questions you have or suggestions for other Python practice problems you'd like to see. Good luck with the interview!

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

Get the Source Code: Click here to get the source code you'll use to practice parsing CSV files in this tutorial.

Mark as Completed

```
8
9 >>> z
10 {'c': 4, 'a': 1, 'b': 3}
```

Email Address

Send Me Python Tricks »

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

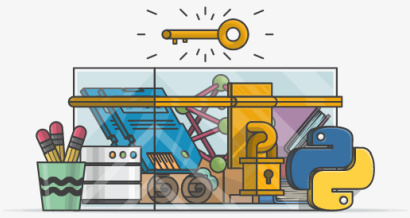
 Geir Arne

 Joanna

 Jacob

 Leodanis

Master Real-World Python Skills With Unlimited Access to Real Python



Join us and get access to hundreds of tutorials, hands-on video courses, and a community of expert Pythonistas:

Level Up Your Python Skills »

What Do You Think?

Tweet Share Email

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