

Processing data files

Chapter 21

This chapter covers

- Using ETL (extract-transform-load)
- Reading text data files (plain text and CSV)
- Reading spreadsheet files
- Normalizing, cleaning, and sorting data
- Writing data files

Welcome to ETL

- The need to get data out of files, parse it, turn it into a useful format, and then do something with it has been around for as long as there have been data files.
- In fact, there is a standard term for the process: extract-transform-load (ETL).
- The extraction refers to the process of reading a data source and parsing it, if necessary.
- The transformation can be cleaning and normalizing the data, as well as combining, breaking up, or reorganizing the records it contains.
- The loading refers to storing the transformed data in a new place, either a different file or a database.

Text encoding: ASCII, Unicode, and others

- The Unicode encoding called UTF-8 accepts the basic ASCII characters without any change but also allows an almost unlimited set of other characters and symbols according to the Unicode standard.
- Even with Unicode, there'll be occasions when your text contains values that can't be successfully encoded.

```
open('test.txt', 'wb').write(bytes([65, 66, 67, 255, 192,193]))
open('test.txt', errors='ignore').read()
open('test.txt', errors='replace').read()
open('test.txt', errors='surrogateescape').read()
open('test.txt', errors='backslashreplace').read()
```

Unstructured text

- Unstructured text files are the easiest sort of data to read but the hardest to extract information from.
- Processing unstructured text can vary enormously, depending on both the nature of the text and what you want to do with it, so any comprehensive discussion of text processing is beyond the scope of this course.

Call me Ishmael. Some years ago--never mind how long precisely--
having little or no money in my purse, and nothing particular
to interest me on shore, I thought I would sail about a little
and see the watery part of the world. It is a way I have
of driving off the spleen and regulating the circulation.
Whenever I find myself growing grim about the mouth;
whenever it is a damp, drizzly November in my soul; whenever I
find myself involuntarily pausing before coffin warehouses,
and bringing up the rear of every funeral I meet;
and especially whenever my hypos get such an upper hand of me,

Reads all of file as
a single string

Splits on two
newlines together

```
>>> moby_text = open("moby_01.txt").read()
>>> moby_paragraphs = moby_text.split("\n\n")
>>> print(moby_paragraphs[1])
```

There now is your insular city of the Manhattoes, belted round by wharves
as Indian isles by coral reefs--commerce surrounds it with her surf.
Right and left, the streets take you waterward. Its extreme downtown
is the battery, where that noble mole is washed by waves, and cooled
by breezes, which a few hours previous were out of sight of land.
Look at the crowds of water-gazers there.

```
>>> moby_text = open("moby_01.txt").read()
>>> moby_paragraphs = moby_text.split("\n\n")
>>> moby = moby_paragraphs[1].lower()
```

Reads all of the file
as a single string

Makes everything
lowercase

```
>>> moby = moby.replace(".", "")
>>> moby = moby.replace(",", "")
>>> moby_words = moby.split()
>>> print(moby_words)
```

Removes
commas

Removes
periods

```
['there', 'now', 'is', 'your', 'insular', 'city', 'of', 'the', 'manhattoes,',
 'belted', 'round', 'by', 'wharves', 'as', 'indian', 'isles', 'by',
 'coral', 'reefs--commerce', 'surrounds', 'it', 'with', 'her', 'surf',
 'right', 'and', 'left,', 'the', 'streets', 'take', 'you', 'waterward',
 'its', 'extreme', 'downtown', 'is', 'the', 'battery,', 'where', 'that',
 'noble', 'mole', 'is', 'washed', 'by', 'waves,', 'and', 'cooled', 'by',
 'breezes,', 'which', 'a', 'few', 'hours', 'previous', 'were', 'out',
 'of', 'sight', 'of', 'land', 'look', 'at', 'the', 'crowds', 'of',
 'water-gazers', 'there']
```


Delimited flat files

- This file is a simple example of temperature data in delimited format:

```
State|Month Day, Year Code|Avg Daily Max Air Temperature (F)|Record Count for  
Daily Max Air Temp (F)  
Illinois|1979/01/01|17.48|994  
Illinois|1979/01/02|4.64|994  
Illinois|1979/01/03|11.05|994  
Illinois|1979/01/04|9.51|994  
Illinois|1979/05/15|68.42|994  
Illinois|1979/05/16|70.29|994  
Illinois|1979/05/17|75.34|994  
Illinois|1979/05/18|79.13|994  
Illinois|1979/05/19|74.94|994
```

Example solution

- Whatever character is being used as the delimiter, if you know what character it is, you can write your own code in Python to break each line into its fields and return them as a list.
- In the previous case, you can use the string `split()` method to break a line into a list of values:

```
>>> line = "Illinois|1979/01/01|17.48|994"  
>>> print(line.split("|"))  
['Illinois', '1979/01/01', '17.48', '994']
```

The csv module

- The csv module is a perfect case of Python's "batteries included" philosophy.
- The csv module has been tested and optimized, and it has features that you probably wouldn't bother to write if you had to do it yourself, but that are truly handy and time-saving when available.

```
>>> results = []
>>> for line in open("temp_data_pipes_00a.txt"):
...     fields = line.strip().split("|")

...     results.append(fields)
...
>>> results
[['State', 'Month Day, Year Code', 'Avg Daily Max Air Temperature (F)',
  'Record Count for Daily Max Air Temp (F)'], ['Illinois', '1979/01/01',
  '17.48', '994'], ['Illinois', '1979/01/02', '4.64', '994'], ['Illinois',
  '1979/01/03', '11.05', '994'], ['Illinois', '1979/01/04', '9.51',
  '994'], ['Illinois', '1979/05/15', '68.42', '994'], ['Illinois', '1979/
05/16', '70.29', '994'], ['Illinois', '1979/05/17', '75.34', '994'],
  ['Illinois', '1979/05/18', '79.13', '994'], ['Illinois', '1979/05/19',
  '74.94', '994']]
```

```
>>> import csv
>>> results = [fields for fields in
    csv.reader(open("temp_data_pipes_00a.txt", newline=''), delimiter=",")]
>>> results
[['State', 'Month Day, Year Code', 'Avg Daily Max Air Temperature (F)',
  'Record Count for Daily Max Air Temp (F)'], ['Illinois', '1979/01/01',
  '17.48', '994'], ['Illinois', '1979/01/02', '4.64', '994'], ['Illinois',
  '1979/01/03', '11.05', '994'], ['Illinois', '1979/01/04', '9.51',
  '994'], ['Illinois', '1979/05/15', '68.42', '994'], ['Illinois', '1979/
  05/16', '70.29', '994'], ['Illinois', '1979/05/17', '75.34', '994'],
  ['Illinois', '1979/05/18', '79.13', '994'], ['Illinois', '1979/05/19',
  '74.94', '994']]
```

Reading a csv file as a list of dictionaries

- In the preceding examples, you got a row of data back as a list of fields.
- This result works fine in many cases, but sometimes it may be handy to get the rows back as dictionaries where the field name is the key.
- For this use case, the csv library has a DictReader, which can take a list of fields as a parameter or can read them from the first line of the data. If you want to open the data with a DictReader, the code would look like this:

```
>>> results = [fields for fields in csv.DictReader(open("temp_data_01.csv",
    newline=''))]
>>> results[0]
OrderedDict([('Notes', ''), ('State', 'Illinois'), ('State Code', '17'),
    ('Month Day, Year', 'Jan 01, 1979'), ('Month Day, Year Code', '1979/01/
    01'), ('Avg Daily Max Air Temperature (F)', '17.48'), ('Record Count for
    Daily Max Air Temp (F)', '994'), ('Min Temp for Daily Max Air Temp (F)',
    '6.00'), ('Max Temp for Daily Max Air Temp (F)', '30.50'), ('Avg Daily
    Min Air Temperature (F)', '2.89'), ('Record Count for Daily Min Air Temp
    (F)', '994'), ('Min Temp for Daily Min Air Temp (F)', '-13.60'), ('Max
    Temp for Daily Min Air Temp (F)', '15.80'), ('Avg Daily Max Heat Index
    (F)', 'Missing'), ('Record Count for Daily Max Heat Index (F)', '0'),
    ('Min for Daily Max Heat Index (F)', 'Missing'), ('Max for Daily Max
    Heat Index (F)', 'Missing'), ('Daily Max Heat Index (F) % Coverage',
    '0.00%')])
```

Excel files

- The other common file format that I discuss in this chapter is the Excel file, which is the format that Microsoft Excel uses to store spreadsheets.
- As it happens, Python's standard library doesn't have a module to read or write Excel files.
- To read that format, you need to install an external module.
- Fortunately, several modules are available to do the job.
- For this example, you use one called OpenPyXL, which is available from the Python package repository.
- You can install it with the following command from a command line

```
$pip install openpyxl
```



```

>>> from openpyxl import load_workbook
>>> wb = load_workbook('temp_data_01.xlsx')
>>> results = []
>>> ws = wb.worksheets[0]
>>> for row in ws.iter_rows():
...     results.append([cell.value for cell in row])
...
>>> print(results)
[['Notes', 'State', 'State Code', 'Month Day, Year', 'Month Day, Year Code',
  'Avg Daily Max Air Temperature (F)', 'Record Count for Daily Max Air
  Temp (F)', 'Min Temp for Daily Max Air Temp (F)', 'Max Temp for Daily
  Max Air Temp (F)', 'Avg Daily Max Heat Index (F)', 'Record Count for
  Daily Max Heat Index (F)', 'Min for Daily Max Heat Index (F)', 'Max for
  Daily Max Heat Index (F)', 'Daily Max Heat Index (F) % Coverage'],
 [None, 'Illinois', 17, 'Jan 01, 1979', '1979/01/01', 17.48, 994, 6,
  30.5, 'Missing', 0, 'Missing', 'Missing', '0.00%'], [None, 'Illinois',
  17, 'Jan 02, 1979', '1979/01/02', 4.64, 994, -6.4, 15.8, 'Missing', 0,
  'Missing', 'Missing', '0.00%'], [None, 'Illinois', 17, 'Jan 03, 1979',
  '1979/01/03', 11.05, 994, -0.7, 24.7, 'Missing', 0, 'Missing',
  'Missing', '0.00%'], [None, 'Illinois', 17, 'Jan 04, 1979', '1979/01/
  04', 9.51, 994, 0.2, 27.6, 'Missing', 0, 'Missing', 'Missing', '0.00%'],
 [None, 'Illinois', 17, 'May 15, 1979', '1979/05/15', 68.42, 994, 61,
  75.1, 'Missing', 0, 'Missing', 'Missing', '0.00%'], [None, 'Illinois',
  17, 'May 16, 1979', '1979/05/16', 70.29, 994, 63.4, 73.5, 'Missing', 0,
  'Missing', 'Missing', '0.00%'], [None, 'Illinois', 17, 'May 17, 1979',
  '1979/05/17', 75.34, 994, 64, 80.5, 82.6, 2, 82.4, 82.8, '0.20%'],
 [None, 'Illinois', 17, 'May 18, 1979', '1979/05/18', 79.13, 994, 75.5,
  82.1, 81.42, 349, 80.2, 83.4, '35.11%'], [None, 'Illinois', 17, 'May 19,
  1979', '1979/05/19', 74.94, 994, 66.9, 83.1, 82.87, 78, 81.6, 85.2,
  '7.85%']]

```

Data cleaning

- One common problem you'll encounter in processing text-based data files is dirty data.
- By dirty, it means that there are all sorts of surprises in the data, such as null values, values that aren't legal for your encoding, or extra whitespace.
- The data may also be unsorted or in an order that makes processing difficult.
- The process of dealing with situations like these is called data cleaning.

Data cleaning steps



Cleaning



Sorting

TRY THIS

How would you handle the fields with 'Missing' as possible values for math calculations? Can you write a snippet of code that averages one of those columns?

What would you do with the average column at the end so that you could also report the average coverage?

In your opinion, would the solution to this problem be at all linked to the way that the 'Missing' entries were handled?

Data cleaning issues and pitfalls



Beware of whitespace and null characters.



Beware punctuation.



Break down and debug the steps.

Writing data files

- These files may be used as input for other applications and analysis, either by people or by other applications.
- Usually, you have a particular file specification listing what fields of data should be included, what they should be named, what format and constraints there should be for each, and so on.

```
>>> temperature_data = [['State', 'Month Day, Year Code', 'Avg Daily Max Air  
Temperature (F)', 'Record Count for Daily Max Air Temp (F)'],  
['Illinois', '1979/01/01', '17.48', '994'], ['Illinois', '1979/01/02',  
'4.64', '994'], ['Illinois', '1979/01/03', '11.05', '994'], ['Illinois',  
'1979/01/04', '9.51', '994'], ['Illinois', '1979/05/15', '68.42',  
'994'], ['Illinois', '1979/05/16', '70.29', '994'], ['Illinois', '1979/  
05/17', '75.34', '994'], ['Illinois', '1979/05/18', '79.13', '994'],  
['Illinois', '1979/05/19', '74.94', '994']]  
>>> csv.writer(open("temp_data_03.csv", "w",  
newline='')).writerows(temperature_data)
```

```
>>> fields = ['State', 'Month Day, Year Code', 'Avg Daily Max Air Temperature  
            (F)', 'Record Count for Daily Max Air Temp (F)']  
>>> dict_writer = csv.DictWriter(open("temp_data_04.csv", "w"),  
                                fieldnames=fields)  
>>> dict_writer.writeheader()  
>>> dict_writer.writerow(data)  
>>> del dict_writer
```



```
>>> from openpyxl import Workbook
>>> data_rows = [fields for fields in csv.reader(open("temp_data_01.csv"))]
>>> wb = Workbook()
>>> ws = wb.active
>>> ws.title = "temperature data"
>>> for row in data_rows:
...     ws.append(row)
...
>>> wb.save("temp_data_02.xlsx")
```

Packaging data files

- If you have several related data files, or if your files are large, it may make sense to package them in a compressed archive.
- Although various archive formats are in use, the zip file remains popular and almost universally accessible to users on almost every platform.

Lab

Weather Observations

Summary

- ETL (extract-transform-load) is the process of getting data from one format, making sure that it's consistent, and then putting it in a format you can use. ETL is the basic step in most data processing.
- Encoding can be problematic with text files, but Python lets you deal with some encoding problems when you load files.
- Delimited or CSV files are common, and the best way to handle them is with the csv module.
- Spreadsheet files can be more complex than CSV files but can be handled much the same way.
- Currency symbols, punctuation, and null characters are among the most common data cleaning issues; be on the watch for them.
- Presorting your data file can make other processing steps faster.