

Data over the network

Chapter 22

This chapter covers

- Fetching files via FTP/SFTP, SSH/SCP, and HTTPS
- Getting data via APIs
- Structured data file formats: JSON and XML

Fetching files

- Before you can do anything with data files, you have to get them.
- Sometimes, this process is very easy, such as manually downloading a single zip archive, or maybe the files have been pushed to your machine from somewhere else.
- Quite often, however, the process is more involved.
- Maybe a large number of files needs to be retrieved from a remote server, files need to be retrieved regularly, or the retrieval process is sufficiently complex to be a pain to do manually.
- In any of those cases, you might well want to automate fetching the data files with Python.

Using Python to fetch files from an FTP server

```
>>> import ftplib
>>> ftp = ftplib.FTP('tgftp.nws.noaa.gov')
>>> ftp.login()
'230 Login successful.'

>>> ftp.cwd('data')
'250 Directory successfully changed.'
>>> ftp.nlst()

['climate', 'fnmoc', 'forecasts', 'hurricane_products', 'ls_SS_services',
 'marine', 'nsd_bbsss.txt', 'nsd_cccc.txt', 'observations', 'products',
 'public_statement', 'raw', 'records', 'summaries', 'tampa',
 'watches_warnings', 'zonecatalog.curr', 'zonecatalog.curr.tar']

>>> x = ftp.retrbinary('RETR observations/metar/decoded/KORD.TXT',
    open('KORD.TXT', 'wb').write)
'226 Transfer complete.'
```

Fetching files with SFTP

- If the data requires more security, such as in a corporate context in which business data is being transferred over the network, it's fairly common to use SFTP.
- SFTP is a full-featured protocol that allows file access, transfer, and management over a Secure Shell (SSH) connection.
- Python doesn't have an SFTP/SCP client module in its standard library, but a community-developed library called paramiko manages SFTP operations as well as SSH connections.

```
>>> import paramiko
>>> t = paramiko.Transport((hostname, port))
>>> t.connect(username, password)
>>> sftp = paramiko.SFTPClient.from_transport(t)
```

Retrieving files over HTTP/HTTPS

- The requests library is by far the easiest and most reliable way to access HTTP/HTTPS servers from Python code.
- Again, requests is easiest to install with `pip install requests`.
- The following example code fetches the monthly temperature data for Heathrow Airport since 1948 - a text file that's served via a web server.

```
>>> import requests
>>> response = requests.get("http://www.metoffice.gov.uk/pub/data/weather/uk/
    climate/stationdata/heathrowdata.txt")
```

```
>>> print(response.text)
Heathrow (London Airport)
Location 507800E 176700N, Lat 51.479 Lon -0.449, 25m amsl
```

Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month) is marked by ---.
Sunshine data taken from an automatic Kipp & Zonen sensor marked with a #,
otherwise sunshine data taken from a Campbell Stokes recorder.

yyyy	mm	tmax degC	tmin degC	af days	rain mm	sun hours
1948	1	8.9	3.3	---	85.0	---
1948	2	7.9	2.2	---	26.0	---
1948	3	14.2	3.8	---	14.0	---
1948	4	15.4	5.1	---	35.0	---
1948	5	18.1	6.9	---	57.0	---

Fetching data via an API

```
>>> import requests
>>> response = requests.get("http://marsweather.ingenology.com/v1/latest/
    ?format=json")
>>> response.text
'{"report": {"terrestrial_date": "2017-01-08", "sol": 1573, "ls": 295.0,
    "min_temp": -74.0, "min_temp_fahrenheit": -101.2, "max_temp": -2.0,
    "max_temp_fahrenheit": 28.4, "pressure": 872.0, "pressure_string":
    "Higher", "abs_humidity": null, "wind_speed": null, "wind_direction": "-
    -", "atmo_opacity": "Sunny", "season": "Month 10", "sunrise": "2017-01-
    08T12:29:00Z", "sunset": "2017-01-09T00:45:00Z"}}'
>>> response = requests.get("http://marsweather.ingenology.com/v1/archive/
    ?sol=155&format=json")
>>> response.text
'{"count": 1, "next": null, "previous": null, "results":
    [{"terrestrial_date": "2013-01-18", "sol": 155, "ls": 243.7, "min_temp":
    -64.45, "min_temp_fahrenheit": -84.01, "max_temp": 2.15,
    "max_temp_fahrenheit": 35.87, "pressure": 9.175, "pressure_string":
    "Higher", "abs_humidity": null, "wind_speed": 2.0, "wind_direction":
    null, "atmo_opacity": null, "season": "Month 9", "sunrise": null,
    "sunset": null}]}'
```


Structured data formats

- Although APIs sometimes serve plain text, it's much more common for data served from APIs to be served in a structured file format.
- The two most common file formats are JSON and XML.
- Both of these formats are built on plain text but structure their contents so that they're more flexible and able to store more complex information.

JSON data

```
>>> import json
>>> import requests
>>> response = requests.get("http://marsweather.ingenology.com/v1/latest/
    ?format=json")
>>> weather = json.loads(response.text)
>>> weather
{'report': {'terrestrial_date': '2017-01-10', 'sol': 1575, 'ls': 296.0,
    'min_temp': -58.0, 'min_temp_fahrenheit': -72.4, 'max_temp': 0.0,
    'max_temp_fahrenheit': None, 'pressure': 860.0, 'pressure_string':
    'Higher', 'abs_humidity': None, 'wind_speed': None, 'wind_direction': '-
    -', 'atmo_opacity': 'Sunny', 'season': 'Month 10', 'sunrise': '2017-01-
    10T12:30:00Z', 'sunset': '2017-01-11T00:46:00Z'}}
>>> weather['report']['sol']
1575
```

Pretty Printing

```
>>> from pprint import pprint as pp
>>> pp(weather)
{'report': {'abs_humidity': None,
            'atmo_opacity': 'Sunny',
            'ls': 296.0,
            'max_temp': 0.0,
            'max_temp_fahrenheit': None,
            'min_temp': -58.0,
            'min_temp_fahrenheit': -72.4,
            'pressure': 860.0,
            'pressure_string': 'Higher',
            'season': 'Month 10',
            'sol': 1575,
            'sunrise': '2017-01-10T12:30:00Z',
            'sunset': '2017-01-11T00:46:00Z',
            'terrestrial_date': '2017-01-10',
            'wind_direction': '--',
            'wind_speed': None}}
```

XML data

- XML (eXtensible Markup Language) has been around since the end of the 20th century.
- XML uses an angle-bracket tag notation similar to HTML, and elements are nested within other elements to form a tree structure.
- XML was intended to be readable by both machines and humans, but XML is often so verbose and complex that it's very difficult for people to understand.
- Nevertheless, because XML is an established standard, it's quite common to find data in XML format.
- And although XML is machine-readable, it's very likely that you'll want to translate it into something a bit easier to deal with.

```

<dwml xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:noNamespaceSchemaLocation="http://www.nws.noaa.gov/forecasts/xml/
DWMLgen/schema/DWML.xsd">
  <head>
    <product srsName="WGS 1984" concise-name="glance" operational-
mode="official">
      <title>
NOAA's National Weather Service Forecast at a Glance
      </title>
      <field>meteorological</field>
      <category>forecast</category>
      <creation-date refresh-frequency="PT1H">2017-01-08T02:52:41Z</creation-
date>
    </product>
    <source>
      <more-information>http://www.nws.noaa.gov/forecasts/xml/</more-
information>
      <production-center>
Meteorological Development Laboratory
      <sub-center>Product Generation Branch</sub-center>
      </production-center>
      <disclaimer>http://www.nws.noaa.gov/disclaimer.html</disclaimer>
      <credit>http://www.weather.gov/</credit>
      <credit-logo>http://www.weather.gov/images/xml_logo.gif</credit-logo>
      <feedback>http://www.weather.gov/feedback.php</feedback>
    </source>
  </head>
  <data>
    <location>
      <location-key>point1</location-key>
      <point latitude="41.78" longitude="-88.65"/>
    </location>
    ...
  </data>
</dwml>

```

How to read XML data?

- For simple data extraction, the handiest utility I've found is a library called `xmldict`, which parses your XML data and returns a dictionary that reflects the tree.
- In fact, behind the scenes it uses the standard library's `expat` XML parser, parses your XML document into a tree, and uses that tree to create the dictionary.

```
>>> import xmldict
>>> data = xmldict.parse(open("observations_01.xml").read())
```

Lab

Track Curiosity's Weather

Summary

- Using a Python script may not be the best choice for fetching files. Be sure to consider the options.
- Using the requests module is your best bet for fetching files by using HTTP/HTTPS and Python.
- Fetching files from an API is very similar to fetching static files.
- Parameters for API requests often need to be quoted and added as a query string to the request URL.
- JSON-formatted strings are quite common for data served from APIs, and XML is also used.
- Scraping sites that you don't control may not be legal or ethical and requires consideration not to overload the server.