Extra Chapters for Norfolk Southern Python Intro

John Strickler

Version 1.0, January 2022

Table of Contents

Chapter 1: Effective Scripting

Objectives

- Launch external programs
- Check permissions on files
- Get system configuration information
- Store data offline
- Create Unix-style filters
- Parse command line options
- Configure application logging

Using glob

- Expands wildcards
- Windows and non-windows
- Useful with **subprocess** module

When executing external programs, sometimes you want to specify a list of files using a wildcard. The **glob** function in the **glob** module will do this. Pass one string containing a wildcard (such as `*.txt`) to glob(), and it returns a sorted list of the matching files. If no files match, it returns an empty list.

Example

glob_example.py

```
#!/usr/bin/env python

from glob import glob

files = glob('../DATA/*.txt') ①
print(files, '\n')

no_files = glob('../JUNK/*.avi')
print(no_files, '\n')
```

① expand file name wildcard into sorted list of matching names

glob_example.py

```
['../DATA/presidents_plus_biden.txt', '../DATA/columns_of_numbers.txt',
'../DATA/poe_sonnet.txt', '../DATA/computer_people.txt', '../DATA/owl.txt',
'../DATA/eggs.txt', '../DATA/world_airport_codes.txt', '../DATA/stateinfo.txt',
'../DATA/fruit2.txt', '../DATA/us_airport_codes.txt', '../DATA/parrot.txt',
'../DATA/http_status_codes.txt', '../DATA/fruit1.txt', '../DATA/alice.txt',
'../DATA/littlewomen.txt', '../DATA/spam.txt', '../DATA/world_median_ages.txt',
'../DATA/phone_numbers.txt', '../DATA/sales_by_month.txt', '../DATA/engineers.txt',
'../DATA/underrated.txt', '../DATA/tolkien.txt', '../DATA/tyger.txt',
'../DATA/example_data.txt', '../DATA/states.txt', '../DATA/kjv.txt', '../DATA/fruit.txt',
'../DATA/areacodes.txt', '../DATA/float_values.txt', '../DATA/unabom.txt',
'../DATA/chaos.txt', '../DATA/noisewords.txt', '../DATA/presidents.txt',
'../DATA/bible.txt', '../DATA/breakfast.txt', '../DATA/Pride_and_Prejudice.txt',
'../DATA/nsfw_words.txt', '../DATA/mary.txt',
'../DATA/2017FullMembersMontanaLegislators.txt', '../DATA/badger.txt',
'../DATA/README.txt', '../DATA/words.txt', '../DATA/ncvoter32.txt',
'../DATA/primeministers.txt', '../DATA/nc_counties_avg_wage.txt', '../DATA/grail.txt',
'../DATA/alt.txt', '../DATA/knights.txt', '../DATA/world_airports_codes_raw.txt',
'../DATA/correspondence.txt']

[]
```

Using shlex.split()

- Splits string
- Preserves white space

If you have an external command you want to execute, you should split it into individual words. If your command has quoted whitespace, the normal **split()** method of a string won't work.

For this you can use **shlex.split()**, which preserves quoted whitespace within a string.

Example

shlex_split.py

```
#!/usr/bin/env python
#
import shlex

cmd = 'herp derp "fuzzy bear" "wanga tanga" pop'  ①

print(cmd.split())  ②
print()

print(shlex.split(cmd))  ③
```

① Command line with quoted whitespace

② Normal split does the wrong thing

③ shlex.split() does the right thing

shlex_split.py

```
['herp', 'derp', '"fuzzy', 'bear"', '"wanga', 'tanga"', 'pop']

['herp', 'derp', 'fuzzy bear', 'wanga tanga', 'pop']
```

The subprocess module

- Spawns new processes
- works on Windows and non-Windows systems
- Convenience methods
  - **run()**
  - **call()**, **check_call()**

The **subprocess** module spawns and manages new processes. You can use this to run local non-Python programs, to log into remote systems, and generally to execute command lines.

subprocess implements a low-level class named Popen; However, the convenience methods **run()**, **check_call()**, and check_output()**, which are built on top of Popen(), are commonly used, as they have a simpler interface. You can capture *stdout** and **stderr**, separately. If you don't capture them, they will go to the console.

In all cases, you pass in an iterable containing the command split into individual words, including any file names. This is why this chapter starts with glob.glob() and shlex.split().

Table 1. CalledProcessError attributes

Attribute	Description
args	The arguments used to launch the process. This may be a list or a string.
returncode	Exit status of the child process. Typically, an exit status of 0 indicates that it ran successfully. A negative value -N indicates that the child was terminated by signal N (POSIX only).
stdout	Captured stdout from the child process. A bytes sequence, or a string if run() was called with an encoding or errors. None if stdout was not captured. If you ran the process with stderr=subprocess.STDOUT, stdout and stderr will be combined in this attribute, and stderr will be None. stderr

subprocess convenience functions

- run(), check_call() , check_output()
- Simpler to use than Popen

subprocess defines convenience functions, **call()**, **check_call()**, and **check_output()**.

```
proc subprocess.run(cmd, ...)
```

Run command with arguments. Wait for command to complete, then return a **CompletedProcess** instance.

```
subprocess.check_call(cmd, ...)
```

Run command with arguments. Wait for command to complete. If the exit code was zero then return, otherwise raise CalledProcessError. The CalledProcessError object will have the return code in the returncode attribute.

```
check_output(cmd, ...)
```

Run command with arguments and return its output as a byte string. If the exit code was non-zero it raises a CalledProcessError. The CalledProcessError object will have the return code in the returncode attribute and output in the output attribute.

NOTE run() is only implemented in Python 3.5 and later.

Example

subprocess_conv.py

```
#!/usr/bin/env python

import sys
from subprocess import check_call, check_output, CalledProcessError
from glob import glob
import shlex

if sys.platform == 'win32':
    CMD = 'cmd /c dir'
    FILES = r'..\DATA\t*'
else:
    CMD = 'ls -ld'
    FILES = '../DATA/t*'

cmd_words = shlex.split(CMD)
cmd_files = glob(FILES)

full_cmd = cmd_words + cmd_files

try:
    check_call(full_cmd)
except CalledProcessError as err:
    print("Command failed with return code", err.returncode)

print('-' * 60)

try:
    output = check_output(full_cmd)
    print("Output:", output.decode(), sep='\n')
except CalledProcessError as e:
    print("Process failed with return code", e.returncode)

print('-' * 50)
```

subprocess_conv.py

```
-rw-r--r--  1 jstrick  staff  3178541 Nov  2  2020 ../DATA/tate_data.zip
-rwxr-xr-x  1 jstrick  staff      297 Nov 17  2016 ../DATA/testscores.dat
-rwxr-xr-x  1 jstrick  staff     2198 Feb 14  2016 ../DATA/textfiles.zip
-rw-r--r--  1 jstrick  staff   106960 Jul 26  2017 ../DATA/titanic3.csv
-rw-r--r--@ 1 jstrick  staff   284160 Jul 26  2017 ../DATA/titanic3.xls
-rwxr-xr-x  1 jstrick  staff    73808 Feb 14  2016 ../DATA/tolkien.txt
-rwxr-xr-x  1 jstrick  staff      834 Feb 14  2016 ../DATA/tyger.txt
------------------------------------------------------------
Output:
-rw-r--r--  1 jstrick  staff  3178541 Nov  2  2020 ../DATA/tate_data.zip
-rwxr-xr-x  1 jstrick  staff      297 Nov 17  2016 ../DATA/testscores.dat
-rwxr-xr-x  1 jstrick  staff     2198 Feb 14  2016 ../DATA/textfiles.zip
-rw-r--r--  1 jstrick  staff   106960 Jul 26  2017 ../DATA/titanic3.csv
-rw-r--r--@ 1 jstrick  staff   284160 Jul 26  2017 ../DATA/titanic3.xls
-rwxr-xr-x  1 jstrick  staff    73808 Feb 14  2016 ../DATA/tolkien.txt
-rwxr-xr-x  1 jstrick  staff      834 Feb 14  2016 ../DATA/tyger.txt

------------------------------------------------
```

NOTE showing Unix/Linux/Mac output – Windows will be similar

TIP (Windows only) The following commands are *internal* to CMD.EXE, and must be preceded by `cmd /c` or they will not work: ASSOC, BREAK, CALL ,CD/CHDIR, CLS, COLOR, COPY, DATE, DEL, DIR, DPATH, ECHO, ENDLOCAL, ERASE, EXIT, FOR, FTYPE, GOTO, IF, KEYS, MD/MKDIR, MKLINK (vista and above), MOVE, PATH, PAUSE, POPD, PROMPT, PUSHD, REM, REN/RENAME, RD/RMDIR, SET, SETLOCAL, SHIFT, START, TIME, TITLE, TYPE, VER, VERIFY, VOL

Capturing stdout and stderr

- Add stdout, stderr args
- Assign subprocess.PIPE

To capture stdout and stderr with the subprocess module, import **PIPE** from subprocess and assign it to the stdout and stderr parameters to run(), check_call(), or check_output(), as needed.

For check_output(), the return value is the standard output; for run(), you can access the **stdout** and **stderr** attributes of the CompletedProcess instance returned by run().

NOTE	output is returned as a bytes object; call decode() to turn it into a normal Python string.

Example

subprocess_capture.py

```
#!/usr/bin/env python

import sys
from subprocess import check_output, Popen, CalledProcessError, STDOUT, PIPE ①
from glob import glob
import shlex

if sys.platform == 'win32':
    CMD = 'cmd /c dir'
    FILES = r'..\DATA\t*'
else:
    CMD = 'ls -ld'
    FILES = '../DATA/t*'

cmd_words = shlex.split(CMD)
cmd_files = glob(FILES)

full_cmd = cmd_words + cmd_files

②
try:
    output = check_output(full_cmd) ③
    print("Output:", output.decode(), sep='\n') ④
except CalledProcessError as e:
    print("Process failed with return code", e.returncode)

print('-' * 50)
```

```
⑤
try:
    cmd = cmd_words + cmd_files + ['spam.txt']
    proc = Popen(cmd, stdout=PIPE, stderr=STDOUT) ⑥
    stdout, stderr = proc.communicate() ⑦
    print("Output:", stdout.decode()) ⑧
except CalledProcessError as e:
    print("Process failed with return code", e.returncode)

print('-' * 50)

try:
    cmd = cmd_words + cmd_files + ['spam.txt']
    proc = Popen(cmd, stdout=PIPE, stderr=PIPE) ⑨
    stdout, stderr = proc.communicate() ⑩
    print("Output:", stdout.decode()) ⑪
    print("Error:", stderr.decode()) ⑪
except CalledProcessError as e:
    print("Process failed with return code", e.returncode)

print('-' * 50)
```

① need to import PIPE and STDOUT

② capture only stdout

③ check_output() returns stdout

④ stdout is returned as bytes (decode to str)

⑤ capture stdout and stderr together

⑥ assign PIPE to stdout, so it is captured; assign STDOUT to stderr, so both are captured together

⑦ call communicate to get the input streams of the process; it returns two bytes objects representing stdout and stderr

⑧ decode the stdout object to a string

⑨ assign PIPE to stdout and PIPE to stderr, so both are captured individually

⑩ now stdout and stderr each have data

⑪ decode from bytes and output

subprocess_capture.py

```
Output:
-rw-r--r--  1 jstrick  staff  3178541 Nov  2  2020 ../DATA/tate_data.zip
-rwxr-xr-x  1 jstrick  staff      297 Nov 17  2016 ../DATA/testscores.dat
-rwxr-xr-x  1 jstrick  staff     2198 Feb 14  2016 ../DATA/textfiles.zip
-rw-r--r--  1 jstrick  staff   106960 Jul 26  2017 ../DATA/titanic3.csv
-rw-r--r--@ 1 jstrick  staff   284160 Jul 26  2017 ../DATA/titanic3.xls
-rwxr-xr-x  1 jstrick  staff    73808 Feb 14  2016 ../DATA/tolkien.txt
-rwxr-xr-x  1 jstrick  staff      834 Feb 14  2016 ../DATA/tyger.txt

--------------------------------------------------
Output: -rw-r--r--  1 jstrick  staff     3178541 Nov  2  2020 ../DATA/tate_data.zip
-rwxr-xr-x  1 jstrick  staff         297 Nov 17  2016 ../DATA/testscores.dat
-rwxr-xr-x  1 jstrick  staff        2198 Feb 14  2016 ../DATA/textfiles.zip
-rw-r--r--  1 jstrick  staff      106960 Jul 26  2017 ../DATA/titanic3.csv
-rw-r--r--@ 1 jstrick  staff      284160 Jul 26  2017 ../DATA/titanic3.xls
-rwxr-xr-x  1 jstrick  staff       73808 Feb 14  2016 ../DATA/tolkien.txt
-rwxr-xr-x  1 jstrick  staff         834 Feb 14  2016 ../DATA/tyger.txt
-rw-r--r--  1 jstrick  students       22 Dec  9 16:30 spam.txt

--------------------------------------------------
Output: -rw-r--r--  1 jstrick  staff     3178541 Nov  2  2020 ../DATA/tate_data.zip
-rwxr-xr-x  1 jstrick  staff         297 Nov 17  2016 ../DATA/testscores.dat
-rwxr-xr-x  1 jstrick  staff        2198 Feb 14  2016 ../DATA/textfiles.zip
-rw-r--r--  1 jstrick  staff      106960 Jul 26  2017 ../DATA/titanic3.csv
-rw-r--r--@ 1 jstrick  staff      284160 Jul 26  2017 ../DATA/titanic3.xls
-rwxr-xr-x  1 jstrick  staff       73808 Feb 14  2016 ../DATA/tolkien.txt
-rwxr-xr-x  1 jstrick  staff         834 Feb 14  2016 ../DATA/tyger.txt
-rw-r--r--  1 jstrick  students       22 Dec  9 16:30 spam.txt

Error:
--------------------------------------------------
```

Permissions

- Simplest is os.access()
- Get mode from os.lstat()
- Use binary AND with permission constants

Each entry in a Unix filesystem has a inode. The inode contains low-level information for the file, directory, or other filesystem entity. Permissions are stored in the 'mode', which is a 16-bit unsigned integer. The first 4 bits indicate what kind of entry it is, and the last 12 bits are the permissions.

To see if a file or directory is readable, writable, or executable use os.access(). To test for specific permissions, use the os.lstat() method to return a tuple of inode data, and use the S_IMODE () method to get the mode information as a number. Then use predefined constants such as stat.S_IRUSR, stat.S_IWGRP, etc. to test for permissions.

Example

file_access.py

```
#!/usr/bin/env python

import sys
import os

if len(sys.argv) < 2:
    start_dir = "."
else:
    start_dir = sys.argv[1]

for base_name in os.listdir(start_dir):  ①
    file_name = os.path.join(start_dir, base_name)
    if os.access(file_name, os.W_OK):  ②
        print(file_name, "is writable")
```

① os.listdir() lists the contents of a directory

② os.access() returns True if file has specified permissions (can be os.W_OK, os.R_OK, or os.X_OK, combined with | (OR))

file_access.py ../DATA

```
../DATA/hyper.xlsx is writable
../DATA/presidents.csv is writable
../DATA/Bicycle_Counts.csv is writable
../DATA/wetprf is writable
../DATA/uri-schemes-1.csv is writable
../DATA/presidents.html is writable
../DATA/presidents.xlsx is writable
../DATA/pokemon_data.csv is writable
../DATA/presidents_plus_biden.txt is writable
../DATA/baby_names is writable
```

...

Using shutil

- Portable ways to copy, move, and delete files
- Create archives
- Misc utilities

The **shutil** module provides portable functions for copying, moving, renaming, and deleting files. There are several variations of each command, depending on whether you need to copy all the attributes of a file, for instance.

The module also provides an easy way to create a zip file or compressed **tar** archive of a folder.

In addition, there are some miscellaneous convenience routines.

Example

shutil_ex.py

```
#!/usr/bin/env python
#
import shutil
import os

shutil.copy('../DATA/alice.txt', 'betsy.txt') ①

print("betsy.txt exists:", os.path.exists('betsy.txt'))

shutil.move('betsy.txt', 'fred.txt') ②
print("betsy.txt exists:", os.path.exists('betsy.txt'))
print("fred.txt exists:", os.path.exists('fred.txt'))

new_folder = 'remove_me'

os.mkdir(new_folder) ③
shutil.move('fred.txt', new_folder)

shutil.make_archive(new_folder, 'zip', new_folder) ④

print("{}.zip exists:".format(new_folder), os.path.exists(new_folder + '.zip'))

print("{} exists:".format(new_folder), os.path.exists(new_folder))

shutil.rmtree(new_folder) ⑤

print("{} exists:".format(new_folder), os.path.exists(new_folder))
```

① copy file

② rename file

③ create new folder

④ make a zip archive of new folder

⑤ recursively remove folder

shutil_ex.py

```
betsy.txt exists: True
betsy.txt exists: False
fred.txt exists: True
remove_me.zip exists: True
remove_me exists: True
remove_me exists: False
```

Creating a useful command line script

- More than just some lines of code
- Input + Business Logic + Output
- Process files for input, or STDIN
- Allow options for customizing execution
- Log results

A good system administration script is more than just some lines of code hacked together. It needs to gather data, apply the appropriate business logic, and, if necessary, output the results of the business logic to the desired destination.

Python has two tools in the standard library to help create professional command line scripts. One of these is the **argparse** module, for parsing options and parameters on the script's command line. The other is fileinput, which simplifies processing a list of files specified on the command line.

We will also look at the logging module, which can be used in any application to output to a variety of log destinations, including a plain file, syslog on Unix-like systems or the NTLog service on Windows, or even email.

Creating filters

- Filter reads files or STDIN and writes to STDOUT

> Common on Unix systems Well-known filters: awk, sed, grep, head, tail, cat Reads command line arguments as files, otherwise STDIN use fileinput.input()

A common kind of script iterates over all lines in all files specified on the command line. The algorithm is

```
for filename in sys.argv[1:]:
    with open(filename) as F:
       for line in F:
            # process line
```

Many Unix utilities are written to work this way – sed, grep, awk, head, tail, sort, and many more. They are called filters, because they filter their input in some way and output the modified text. Such filters read STDIN if no files are specified, so that they can be piped into.

The fileinput.input() class provides a shortcut for this kind of file processing. It implicitly loops through sys.argv[1:], opening and closing each file as needed, and then loops through the lines of each file. If sys.argv[1:] is empty, it reads sys.stdin. If a filename in the list is '-', it also reads sys.stdin.

fileinput works on Windows as well as Unix and Unix-like platforms.

To loop through a different list of files, pass an iterable object as the argument to fileinput.input().

There are several methods that you can call from fileinput to get the name of the current file, e.g.

Table 2. fileinput Methods

Method	Description
filename()	Name of current file being readable
lineno()	Cumulative line number from all files read so far
filelineno()	Line number of current file
isfirstline()	True if current line is first line of a file
isstdin()	True if current file is sys.stdin
close()	Close fileinput

Example

file_input.py

```
#!/usr/bin/env python

import fileinput

for line in fileinput.input():  ①
    if 'bird' in line:
        print('{}: {}'.format(fileinput.filename(), line), end=' ')  ②
```

① fileinput.input() is a generator of all lines in all files in sys.argv[1:]

② fileinput.filename() has the name of the current file

file_input.py ../DATA/parrot.txt ../DATA/alice.txt

```
../DATA/parrot.txt: At that point, the guy is so mad that he throws the bird into the
../DATA/parrot.txt: For the first few seconds there is a terrible din. The bird kicks
../DATA/parrot.txt: bird may be hurt. After a couple of minutes of silence, he's so
../DATA/parrot.txt: The bird calmly climbs onto the man's out-stretched arm and says,
../DATA/alice.txt: with the birds and animals that had fallen into it:  there were a
../DATA/alice.txt: bank--the birds with draggled feathers, the animals with their
../DATA/alice.txt: some of the other birds tittered audibly.
../DATA/alice.txt: and confusion, as the large birds complained that they could not
```

Parsing the command line

- Parse and analyze **sys.argv**
- Use **argparse**
  - Parses entire command line
  - Flexible
  - Validates options and arguments

Many command line scripts need to accept options and arguments. In general, options control the behavior of the script, while arguments provide input. Arguments are frequently file names, but can be anything. All arguments are available in Python via sys.argv

There are at least three modules in the standard library to parse command line options. The oldest module is **getopt** (earlier than v1.3), then **optparse** (introduced 2.3, now deprecated), and now, **argparse** is the latest and greatest. (Note: **argparse** is only available in 2.7 and 3.0+).

To get started with **argparse**, create an ArgumentParser object. Then, for each option or argument, call the parser's add_argument() method.

The add_argument() method accepts the name of the option (e.g. '-count') or the argument (e.g. 'filename'), plus named parameters to configure the option.

Once all arguments have been described, call the parser's parse_args() method. (By default, it will process sys.argv, but you can pass in any list or tuple instead.) parse_args() returns an object containing the arguments. You can access the arguments using either the name of the argument or the name specified with dest.

One useful feature of **argparse** is that it will convert command line arguments for you to the type specified by the type parameter. You can write your own function to do the conversion, as well.

Another feature is that **argparse** will automatically create a help option, -h, for your application, using the help strings provided with each option or parameter.

argparse parses the entire command line, not just arguments

Table 3. add_argument() named parameters

parameter	description
dest	Name of attribute (defaults to argument name)
nargs	Number of arguments Default: one argument, returns string '*': 0 or more arguments, returns list '+': 1 or more arguments, returns list '?': 0 or 1 arguments, returns list N: exactly N arguments, returns list
const	Value for options that do not take a user-specified value
default	Value if option not specified
type	type which the command-line arguments should be converted ; one of 'string', 'int', 'float', 'complex' or a function that accepts a single string argument and returns the desired object. (Default: 'string')
choices	A list of valid choices for the option
required	Set to true for required options
metavar	A name to use in the help string (default: same as dest)
help	Help text for option or argument

Example

parsing_args.py

```
#!/usr/bin/env python
import re
import fileinput
import argparse
from glob import glob  ①
from itertools import chain  ②

arg_parser = argparse.ArgumentParser(description="Emulate grep with python")  ③

arg_parser.add_argument(
    '-i',
    dest='ignore_case', action='store_true',
    help='ignore case'
)  ④

arg_parser.add_argument(
    'pattern', help='Pattern to find (required)'
)  ⑤

arg_parser.add_argument(
    'filenames', nargs='*',
    help='filename(s) (if no files specified, read STDIN)'
)  ⑥

args = arg_parser.parse_args()  ⑦

print('-' * 40)
print(args)
print('-' * 40)

regex = re.compile(args.pattern, re.I if args.ignore_case else 0)  ⑧

filename_gen = (glob(f) for f in args.filenames)  ⑨
filenames = chain.from_iterable(filename_gen)  ⑩

for line in fileinput.input(filenames):  ⑪
    if regex.search(line):
        print(line.rstrip())
```

① needed on Windows to parse filename wildcards

② needed on Windows to flatten list of filename lists

③ create argument parser

④ add option to the parser; dest is name of option attribute

⑤ add required argument to the parser

⑥ add optional arguments to the parser

⑦ actually parse the arguments

⑧ compile the pattern for searching; set re.IGNORECASE if -i option

⑨ for each filename argument, expand any wildcards; this returns list of lists

⑩ flatten list of lists into a single list of files to process (note: both filename_gen and filenames are generators; these two lines are only needed on Windows — non-Windows systems automatically expand wildcards)

⑪ loop over list of file names and read them one line at a time

parsing_args.py

```
usage: parsing_args.py [-h] [-i] pattern [filenames [filenames ...]]
parsing_args.py: error: the following arguments are required: pattern, filenames
```

parsing_args.py -i '\bbil' ../DATA/alice.txt ../DATA/presidents.txt

```
----------------------------------------
Namespace(filenames=['../DATA/alice.txt', '../DATA/presidents.txt'], ignore_case=True,
pattern='\\bbil')
----------------------------------------
                The Rabbit Sends in a Little Bill
Bill's got the other--Bill! fetch it here, lad!--Here, put 'em up
Here, Bill! catch hold of this rope--Will the roof bear?--Mind
crash)--`Now, who did that?--It was Bill, I fancy--Who's to go
then!--Bill's to go down--Here, Bill! the master says you're to
  `Oh! So Bill's got to come down the chimney, has he?' said
Alice to herself.  `Shy, they seem to put everything upon Bill!
I wouldn't be in Bill's place for a good deal:  this fireplace is
above her:  then, saying to herself `This is Bill,' she gave one
Bill!' then the Rabbit's voice along--`Catch him, you by the
  Last came a little feeble, squeaking voice, (`That's Bill,'
The poor little Lizard, Bill, was in the middle, being held up by
end of the bill, "French, music, AND WASHING--extra."'
Bill, the Lizard) could not make out at all what had become of
Lizard as she spoke.  (The unfortunate little Bill had left off
42:Clinton:William Jefferson 'Bill':1946-08-19:NONE:Hope:Arkansas:1993-01-20:2001-01-
20:Democratic
```

parsing_args.py -h

```
usage: parsing_args.py [-h] [-i] pattern [filenames [filenames ...]]

Emulate grep with python

positional arguments:
  pattern     Pattern to find (required)
  filenames   filename(s) (if no files specified, read STDIN)

optional arguments:
  -h, --help  show this help message and exit
  -i          ignore case
```

Simple Logging

- Specify file name
- Configure the minimum logging level
- Messages added at different levels
- Call methods on logging

For simple logging, just configure the log file name and minimum logging level with the basicConfig() method. Then call one of the per-level methods, such as logging.debug or logging.error, to output a log message for that level. If the message is at or above the minimal level, it will be added to the log file.

The file will continue to grow, and must be manually removed or truncated. If the file does not exist, it will be created.

The logger module provides 5 levels of logging messages, from DEBUG to CRITICAL. When you set up a logger, you specify the minimum level of messages to be logged. If you set up the logger with the minimum level set to ERROR, then only messages at ERROR and CRITICAL levels will be logged. Setting the minimum level to DEBUG allows all messages to be logged.

Table 4. Logging Levels

Level	Value
CRITICAL FATAL	50
ERROR	40
WARN WARNING	30
INFO	20
DEBUG	10
UNSET	0

Example

logging_simple.py

```
#!/usr/bin/env python

import logging

logging.basicConfig(
    filename='../TEMP/simple.log',
    level=logging.WARNING,
) ①

logging.warning('This is a warning') ②
logging.debug('This message is for debugging') ③
logging.error('This is an ERROR') ④
logging.critical('This is ***CRITICAL***') ⑤
logging.info('The capital of North Dakota is Bismark') ⑥
```

① setup logging; minimal level is WARN

② message will be output

③ message will NOT be output

④ message will be output

⑤ message will be output

⑥ message will not be output

simple.log

```
WARNING:root:This is a warning
ERROR:root:This is an ERROR
CRITICAL:root:This is ***CRITICAL***
```

Formatting log entries

- Add format=format to basicConfig() parameters
- Format is a string containing directives and (optionally) other text
- Use directives in the form of %(item)type
- Other text is left as-is

To format log entries, provide a format parameter to the basicConfig() method. This format will be a string contain special directives (i.e. Placeholders) and, optionally, other text. The directives are replaced with logging information; other data is left as-is.

Directives are in the form %(item)type, where item is the data field, and type is the data type.

Example

logging_formatted.py

```
#!/usr/bin/env python

import logging

logging.basicConfig(
    format='%(name)s %(asctime)s %(levelname)s %(message)s', ①
    filename='../TEMP/formatted.log',
    level=logging.INFO,
)

logging.info("this is information")
logging.warning("this is a warning")
logging.info("this is information")
logging.critical("this is critical")
```

① set the format for log entries

formatted.log

```
root 2022-01-12 08:02:24,600 INFO this is information
root 2022-01-12 08:02:24,600 WARNING this is a warning
root 2022-01-12 08:02:24,600 INFO this is information
root 2022-01-12 08:02:24,600 CRITICAL this is critical
```

Table 5. Log entry formatting directives

Directive	Description
%(name)s	Name of the logger (logging channel)
%(levelno)s	Numeric logging level for the message (DEBUG, INFO, WARNING, ERROR, CRITICAL)
%(levelname)s	Text logging level for the message ("DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL")
%(pathname)s	Full pathname of the source file where the logging call was issued (if available)
%(filename)s	Filename portion of pathname
%(module)s	Module (name portion of filename)
%(lineno)d	Source line number where the logging call was issued (if available)
%(funcName)s	Function name
%(created)f	Time when the LogRecord was created (time.time() return value)
%(asctime)s	Textual time when the LogRecord was created
%(msecs)d	Millisecond portion of the creation time
%(relativeCreated)d	Time in milliseconds when the LogRecord was created, relative to the time the logging module was loaded (typically at application startup time)
%(thread)d	Thread ID (if available)
%(threadName)s	Thread name (if available)
%(process)d	Process ID (if available)
%(message)s	The result of record.getMessage(), computed just as the record is emitted

Logging exception information

- Use logging.exception()
- Adds exception info to message
- Only in **except** blocks

The logging.exception() function will add exception information to the log message. It should only be called in an **except** block.

Example

logging_exception.py

```
#!/usr/bin/env python

import logging

logging.basicConfig( ①
    filename='../TEMP/exception.log',
    level=logging.WARNING,  ②
)

for i in range(3):
    try:
        result = i/0
    except ZeroDivisionError:
        logging.exception('Logging with exception info') ③
```

① configure logging

② minimum level

③ add exception info to the log

exception.log

```
ERROR:root:Logging with exception info
Traceback (most recent call last):
  File "logging_exception.py", line 12, in <module>
    result = i/0
ZeroDivisionError: division by zero
ERROR:root:Logging with exception info
Traceback (most recent call last):
  File "logging_exception.py", line 12, in <module>
    result = i/0
ZeroDivisionError: division by zero
ERROR:root:Logging with exception info
Traceback (most recent call last):
  File "logging_exception.py", line 12, in <module>
    result = i/0
ZeroDivisionError: division by zero
```

Logging to other destinations

- Use specialized handlers to write to other destinations
- Multiple handlers can be added to one logger
  - NTEventLogHandler for Windows event log
  - SysLogHandler for syslog
  - SMTPHandler for logging via email

The logging module provides some preconfigured log handlers to send log messages to destinations other than a file.

Each handler has custom configuration appropriate to the destination. Multiple handlers can be added to the same logger, so a log message will go to a file and to email, for instance, and each handler can have its own minimum level. Thus, all messages could go to the message file, but only CRITICAL messages would go to email.

Be sure to read the documentation for the particular log handler you want to use

NOTE

On Windows, you must run the example script (logging.altdest.py) as administrator. You can find `Command Prompt (admin)` on the main Windows 8/10 menu. You can also right-click on `Command Prompt` from the Windows 7 menu and choose "Run as administrator".

Example

logging_altdest.py

```
#!/usr/bin/env python
import sys
import logging
import logging.handlers

logger = logging.getLogger('ThisApplication')  ①
logger.setLevel(logging.DEBUG)  ②

if sys.platform == 'win32':
    eventlog_handler = logging.handlers.NTEventLogHandler("Python Log Test")  ③
    logger.addHandler(eventlog_handler)  ④
else:
    syslog_handler = logging.handlers.SysLogHandler()  ⑤
    logger.addHandler(syslog_handler)  ⑥

# note -- use your own SMTP server...
email_handler = logging.handlers.SMTPHandler(
    ('smtpcorp.com', 8025),
    'LOGGER@pythonclass.com',
    ['jstrick@mindspring.com'],
    'ThisApplication Log Entry',
    ('jstrickpython', 'python(monty)'),
)  ⑦

logger.addHandler(email_handler)  ⑧

logger.debug('this is debug')  ⑨
logger.critical('this is critical')  ⑨
logger.warning('this is a warning')  ⑨
```

① get logger for application

② minimum log level

③ create NT event log handler

④ install NT event handler

⑤ create syslog handler

⑥ install syslog handler

⑦ create email handler

⑧ install email handler

⑨ goes to all handlers

Chapter 1 Exercises

Exercise 1-1 (copy_files.py)

Write a script to find all text files (only the files that end in ".txt") in the DATA folder of the student files and copy them to C:\TEMP (Windows) or /tmp (non-windows). On Windows, create the C:\TEMP folder if it does not already exist.

Add logging to the script, and log each filename at level INFO.

TIP	use shutil.copy() to copy the files.

Chapter 2: Working with the File System

Objectives

- Walking through a directory tree
- Working with path names
- Checking for existence of files
- Performing file operations

Paths, directories and filenames

- os module and os.path module
- Many functions for working with files and paths

The os and os.path modules provide many functions for working with paths. They also contain some os-specific attributes.

Some of the more common methods: * os.path.exists() return True if file or directory exists * os.path.getsize() return size of file in bytes * os.path.getmtime() return time of last modification * os.path.getctime() return file creation time (only on Windows) * os.path.split() split a path into "head" and "tail"

Useful attributes: * os.sep directory component separator (typically '/' or '\') * os.pathsep separator for PATH variable (typically ':' or ';') * os.extsep separator between filename and extension (typically '.')

```
NOTE: To get file creation time on Mac, use `os.stat()` and get the st_birthtime value.
There is no convenient way to get file creation time on Linux.
```

Example

paths.py

```
#!/usr/bin/env python
import sys
import os.path

unix_p1 = "bin/spam.txt"  ①
unix_p2 = "/usr/local/bin/ham"  ②

win_p1 = r"spam\ham.doc"  ③
win_p2 = r"\\spam\ham\eggs\toast\jam.doc"  ④

if sys.platform == 'win32':  ⑤
    print("win_p1:", win_p1)
    print("win_p2:", win_p2)
    print("dirname(win_p1):", os.path.dirname(win_p1))  ⑥
    print("dirname(win_p2):", os.path.dirname(win_p2))
    print("basename(win_p1):", os.path.basename(win_p1))  ⑦
    print("basename(win_p2):", os.path.basename(win_p2))
    print("isabs(win_p1):", os.path.isabs(win_p1))  ⑧
    print("isabs(win_p2):", os.path.isabs(win_p2))
else:
    print("unix_p1:", unix_p1)
    print("unix_p2:", unix_p2)
    print("dirname(unix_p1):", os.path.dirname(unix_p1))  ⑥
    print("dirname(unix_p2):", os.path.dirname(unix_p2))
    print("basename(unix_p1):", os.path.basename(unix_p1))  ⑦
    print("basename(unix_p2):", os.path.basename(unix_p2))
    print("isabs(unix_p1):", os.path.isabs(unix_p1))  ⑧
    print("isabs(unix_p2):", os.path.isabs(unix_p2))
    print(
        'format("cp spam.txt {}".format(os.path.expanduser("~"))):',  ⑨
        format("cp spam.txt {}".format(os.path.expanduser("~"))),
    )
    print(
        'format("cd {}".format(os.path.expanduser("~root"))):',  ⑩
        format("cd {}".format(os.path.expanduser("~root"))),
    )
```

① Unix relative path

② Unix absolute path

③ Windows relative path

④ Windows UNC path

⑤ What platform are we on?

⑥ Just the folder name

⑦ Just the file (or folder) name

⑧ Is it an absolute path?

⑨ ~ is current user's home

⑩ ~NAME is NAME's home

paths.py

```
unix_p1: bin/spam.txt
unix_p2: /usr/local/bin/ham
dirname(unix_p1): bin
dirname(unix_p2): /usr/local/bin
basename(unix_p1): spam.txt
basename(unix_p2): ham
isabs(unix_p1): False
isabs(unix_p2): True
format("cp spam.txt {}".format(os.path.expanduser("~"))): cp spam.txt /Users/jstrick
format("cd {}".format(os.path.expanduser("~root"))): cd /var/root
```

Table 6. os.path methods

Method	Description
abspath(path)	Return normalized absolutized version of the pathname path.
basename(path)	Return the base name of pathname path.
commonprefix(list)	Return the longest path prefix (taken character-by-character) that is a prefix of all paths in list. If list is empty, return the empty string ('').
dirname(path)	Return the directory name of pathname path.
exists(path)	Return True if path refers to an existing path. Returns False for broken symbolic links. May be subject to permissions
lexists(path)	Return True if path refers to an existing path. Returns True for broken symbolic links.
expanduser(path)	On Unix, return the argument with an initial component of "~" or "~user" replaced by that user's home directory. Only "~" is supported on Windows.
expandvars(path)	Return the argument with environment variables expanded. Substrings of the form "$name" or "${name}" are replaced by the value of environment variable name. Malformed variable names and references to non-existing variables are left unchanged.
getatime(path)	Return the time of last access of path. (seconds since epoch).
getmtime(path)	Return the time of last modification of path. (seconds since epoch).
getctime(path)	Return the time of last modification of the path's **properties** (not creation time). (seconds since epoch).
getsize(path)	Return the size, in bytes, of path. Raise os.error if path does not exist or cannot be accessed.
isabs(path)	Return True if path is an absolute pathname (begins with a slash).
isfile(path)	Return True if path is an existing regular file. This follows symbolic links.
isdir(path)	Return True if path is an existing directory. Follows symbolic links.
islink(path)	Return True if path refers to a directory entry that is a symbolic link. Always False on Windows.
ismount(path)	Return True if pathname path is a mount point (Unix only).
join(path1[, path2[, ...]])	Join one or more path components intelligently.
normcase(path)	Normalize the case of a pathname. On Unix, this returns the path unchanged; on case-insensitive filesystems, it converts the path to lowercase. On Windows, it also converts forward slashes to backward slashes.
normpath(ph)	Normalize a pathname. This collapses redundant separators and up-level references so that A//B, A/./B and A/foo/../B all become A/B.

Method	Description
realpath(path)	Return the canonical path of the specified filename, eliminating any symbolic links encountered in the path.
samefile(path1, path2)	Return True if both pathname arguments refer to the same file or directory (as indicated by device number and i-node number). Raise an exception if a os.stat() call on either pathname fails. Availability: Macintosh, Unix.
sameopenfile(fp1, fp2)	Return True if the file descriptors fp1 and fp2 refer to the same file. Availability: Macintosh, Unix.
samestat(stat1, stat2)	Return True if the stat tuples stat1 and stat2 refer to the same file. These structures may have been returned by fstat(), lstat(), or stat(). Availability: Macintosh, Unix.
split(path)	Split the pathname path into a pair, (head, tail) where tail is the last pathname component and head is everything leading up to that. The tail part will never contain a slash.
splitdrive(path)	Split the pathname path into a pair (drive, tail) where drive is either a drive specification or the empty string. On systems which do not use drive specifications, drive will always be the empty string..
splitext(path)	Split the pathname path into a pair (root, ext) such that root + ext == path, and ext is empty or begins with a period and contains at most one period.
splitunc(path)	Split the pathname path into a pair (unc, rest) so that unc is the UNC mount point (such as r'\\host\mount'), if present, and rest the rest of the path (such as r'\path\file.ext'). For paths containing drive letters, unc will always be the empty string. Availability: Windows.
walk(path, visit, arg)	Calls the function visit with arguments (arg, dirname, names) for each directory in the directory tree rooted at path (including path itself, if it is a directory). Note: The newer os.walk() generator supplies similar functionality and can be easier to use. (Like File::Find in perl)
supports_unicode_filenames	True if arbitrary Unicode strings can be used as file names (within limitations imposed by the file system), and if os.listdir() returns Unicode strings for a Unicode argument. New in version 2.3.

Checking for existence

- Use os.path.exists()

The **exists()** method of os.path can be used to determine whether a file or folder exists.

NOTE	os.access() can also be used

Example

existence.py

```
#!/usr/bin/env python

import os

for path in ('c:/windows', '/etc', '../DATA', '/wombat', '/tmp', 'RESULTS', 'C:/temp'):
①
    print(path, end=' ')  ②
    if os.path.exists(path):  ③
        print('exists')
    else:
        print('does not exist')
```

① loop through some files and folders that may or may not exist

② print path to be checked, but stay on same line (output space instead of newline)

③ check for existence

existence.py

```
c:/windows does not exist
/etc exists
../DATA exists
/wombat does not exist
/tmp exists
RESULTS does not exist
C:/temp does not exist
```

NOTE	On Unix-like systems, exists() returns False for broken symbolic links

Getting the contents of a directory

- Use os.listdir() or os.scandir()
- Use os.path.join() to create path

To get the contents of a directory, there are two functions in os.path. Use os.listdir() to return the contents of a directory as a list of strings.

Use os.scandir() to return a generator of **DirEntry** objects. Each DirEntry has attributes for the name of the entry as well as some other data, including the result of **stat()**.

scandir() is a little faster than *listdir(

Example

listing_directories.py

```
#!/usr/bin/env python
import os

DIRECTORY = '../DATA'

for i, entry_name in enumerate(os.listdir(DIRECTORY)):
    print(entry_name)
    if i == 9:
        break

print('-' * 60)

for i, entry in enumerate(os.scandir(DIRECTORY)):
    print("{:25s} {:6s} {:6s} {:6o}".format(entry.name, str(entry.is_dir()),
str(entry.is_file()), entry.stat().st_mode))
    if i == 9:
        break
```

listing_directories.py

```
hyper.xlsx
presidents.csv
Bicycle_Counts.csv
wetprf
uri-schemes-1.csv
presidents.html
presidents.xlsx
pokemon_data.csv
presidents_plus_biden.txt
baby_names
------------------------------------------------------------
hyper.xlsx                False  True   100644
presidents.csv            False  True   100755
Bicycle_Counts.csv        False  True   100644
wetprf                    True   False   40755
uri-schemes-1.csv         False  True   100644
presidents.html           False  True   100755
presidents.xlsx           False  True   100644
pokemon_data.csv          False  True   100644
presidents_plus_biden.txt False  True   100755
baby_names                True   False   40755
```

Getting file attributes

- Simple to get size, timestamps, other info
- Import os
- Use datetime.fromtimestamp() to convert timestamps

The os.access() method tests whether a file exists (like os.path.exists()), or whether the user running the script has read, ``write, or execute permission on the file. Use the constants os.F_OK, os.R_OK, os.W_OK, or os.X_OK with os.access().

The os.path.getsize() method will return the size of a filesystem node (file, directory, or folder) in bytes. The os.path.getmtime() method returns the timestamp from the last time the file was modified; the os.path.getatime() method returns the timestamp from the last time the node was read.

The os.stat() method returns a named tuple of information with information from the files's internal structures (on Unix-like systems, the inode).

On Unix-like systems, os.stat() follows symbolic links; use os.lstat() to provide information on the target even if it's a symbolic link.

Example

file_access.py

```
#!/usr/bin/env python

import sys
import os

if len(sys.argv) < 2:
    start_dir = "."
else:
    start_dir = sys.argv[1]

for base_name in os.listdir(start_dir):  ①
    file_name = os.path.join(start_dir, base_name)
    if os.access(file_name, os.W_OK):  ②
        print(file_name, "is writable")
```

① os.listdir() lists the contents of a directory

② os.access() returns True if file has specified permissions (can be os.W_OK, os.R_OK, or os.X_OK, combined with | (OR))

file_access.py

```
./boto3_build_function.py is writable
./db_mysql_basics.py is writable
./bs4_parse_longest_rivers.py is writable
./db_postgres_get_version.py is writable
./bs4_parse_html.py is writable
./xml_from_presidents.py is writable
./fmt_types.py is writable
./map_vs_listcomp2.py is writable
./existence.py is writable
./px_save_range_to_sheet.py is writable
```

Example

file_attrib.py

```
#!/usr/bin/env python
import os
from datetime import datetime


def format_time(time_stamp):
    """convert time stamp to YYYY-MM-DD HH:MM string"""
    date_time = datetime.fromtimestamp(time_stamp)  ①
    time_string = date_time.strftime('%Y-%m-%d %H:%M')  ②

    return time_string


filenames = (  ③
    '../DATA/alice.txt',
    '../ANSWERS/sieve.py',
    'file_attrib.py',
)

for filename in filenames:
    size = os.path.getsize(filename)  ④
    mtime_ts = os.path.getmtime(filename)  ⑤
    mtime = format_time(mtime_ts)

    atime_ts = os.path.getatime(filename)  ⑥
    atime = format_time(atime_ts)

    print('{0:20s} {1:8d}  {2}  {3}'.format(filename, size, mtime, atime))
```

① convert epoch time to Python datetime

② format datetime object

③ files to test

④ get size of file

⑤ get file timestamp (last time file was modified)

⑥ get file timestamp (last time file was read)

file_attrib.py

```
../DATA/alice.txt      148544  2016-02-14 09:46  2020-12-25 00:33
../ANSWERS/sieve.py       304  2020-01-28 11:17  2020-12-25 00:28
file_attrib.py            707  2019-05-25 07:47  2020-12-25 00:28
```

Example

file_stat.py

```
#!/usr/bin/env python
import os

info = os.stat('../DATA/parrot.txt')  ①
print(info)  ②

print('size is', info[6])  ③
print('size is', info.st_size)  ④
```

① get stat data for file

② print all data, which is a named tuple

③ get the file size by numeric index

④ get the file size by field name

file_stat.py

```
os.stat_result(st_mode=33261, st_ino=516354, st_dev=16777221, st_nlink=1, st_uid=501,
st_gid=20, st_size=1437, st_atime=1608874384, st_mtime=1455461206, st_ctime=1608856398)
size is 1437
size is 1437
```

Walking directory trees

- import os module
- Use the walk() iterator
- Returns tuple for each directory starting with the specified top directory
- Tuple contains full path to directory, list of subdirectories, and list of files
- syntax:

```
for currdir,subdirs,files in os.walk("start-dir"):
    pass
```

The os.walk() method provides a way to easily walk a directory tree. It provides an iterator for a directory and all its subdirectories. For each directory, it returns a tuple with three elements.

The first element is the full (absolute) path to the directory; the second element is a list of the directory's subdirectories (relative names); the third element is a list of the non-directory files in the subdirectory (also relative names).

Don't use **dir** or **file** as variables when looping through the iterator, because they will overwrite builtins. (No matter how tempting they are).

Example

os_walk.py

```
#!/usr/bin/env python

# count number of files and dirs in  a directory tree
# note "files" includes devices, symbolic links, and pipes
import os
import sys

if sys.platform == 'win32':  ①
    target = 'C:/Users'
else:
    target = '/etc'

total_files = 0  ②
total_dirs = 0

for currdir, subdirs, files in os.walk(target):  ③
    total_dirs += 1  # increment number of directories seen
    total_files += len(files)  # add the number of files in this dir

print("{} contains {} dirs and {} files".format(target, total_dirs, total_files))  ④
```

① check platform

② initialize counters

③ visit target folder and each subfolder; os.walk() returns 3-tuple at each folder

④ output results

os_walk.py

```
/etc contains 38 dirs and 342 files
```

Example

os_walk2.py

```
#!/usr/bin/env python
"""
    find files whose size is greater than or equal to specified number of bytes
"""
import sys
import os

MINIMUM_SIZE = 1000

if len(sys.argv) < 2: ①
    print('Syntax: walk2.py START-DIR')
    sys.exit(1)

for currdir, subdirs, files in os.walk(sys.argv[1]):
    for file in files: ②
        fullpath = os.path.join(currdir, file) ③
        if os.path.isfile(fullpath): ④
            fsize = os.path.getsize(fullpath) ⑤
            if fsize >= MINIMUM_SIZE: ⑥
                print("{:40s} {:8d}".format(fullpath, fsize))
```

① make sure there is at least one command line argument

② in each folder, iterate over files

③ get full path to file

④ make sure it's a file (not a link or device, etc.)

⑤ get file size

⑥ check size to see if it's at least MINIMUM_SIZE

os_walk2.py

```
./xml_from_presidents.py                 2458
./boto3_create_folders.py                1324
./basic_auth_requests.py                 1545
./dc_carddeck.py                         1595
./paramiko_remote_cmd.py                 1217
./bm_ordered_unique.py                   1422
./presidents_hidden_sheet.xlsx          11186
./sa_movie_models.py                     1349
./example.zip                           56345
./xml_create_knights.py                  1396
./.pylintrc                             14755
./db_mysql_metadata.py                   1431
./new_magic.py                           1387
./iterable_recipes.py                    8359
./sa_movie_populate_db.py                1034
./flask_simple_form.py                   1247
./nba_procs.py                           1526
./db_mysql_bulk_insert.py                2316
./rxml.py                                1146
./tauntmodule.c                          3137
```

...

File operations with shutil

- shutil provides cross-platform high-level file operations
- Move and copy files and folders

The **shutil** module provides a simple, cross-platform way to move and copy files and file trees.

The basic function is copy(), which will make a copy of a file, preserving permission bits. The destination for copy() can be either a file name or a folder.

To copy a file tree, use copytree().

To relocate or rename a file or folder, use move(). As with copy(), the destination can be either a filename or folder.

Some other useful functions: copyfile() which is like copy(), but requires the destination to be a file, and does not preserve permissions.

Example

shutil_ex.py

```
#!/usr/bin/env python
#
import shutil
import os

shutil.copy('../DATA/alice.txt', 'betsy.txt') ①

print("betsy.txt exists:", os.path.exists('betsy.txt'))

shutil.move('betsy.txt', 'fred.txt') ②
print("betsy.txt exists:", os.path.exists('betsy.txt'))
print("fred.txt exists:", os.path.exists('fred.txt'))

new_folder = 'remove_me'

os.mkdir(new_folder) ③
shutil.move('fred.txt', new_folder)

shutil.make_archive(new_folder, 'zip', new_folder) ④

print("{}.zip exists:".format(new_folder), os.path.exists(new_folder + '.zip'))

print("{} exists:".format(new_folder), os.path.exists(new_folder))

shutil.rmtree(new_folder) ⑤

print("{} exists:".format(new_folder), os.path.exists(new_folder))
```

① copy file

② rename file

③ create new folder

④ make a zip archive of new folder

⑤ recursively remove folder

shutil_ex.py

```
betsy.txt exists: True
betsy.txt exists: False
fred.txt exists: True
remove_me.zip exists: True
remove_me exists: True
remove_me exists: False
```

Chapter 2 Exercises

Exercise 2-1 (file_date.py)

Write a script which accepts one or more filenames on the command line, and prints out each file name with its date of last modification in the format 'Mar 12, 2013'.

Exercise 2-2 (get_file_size.py)

Write a script that accepts one or more files on the command line, and prints out the size, one file per line. If any argument is a directory, print out a warning message.

Exercise 2-3 (for_usage.py)

Write a script to count how many times a for loop is used in all .py files under py3master (the root of the student files).

Exercise 2-4 (copy_csv.py)

Write a script to copy all CSV files from the DATA folder into the TEMP folder.

TIP	use shutil

Chapter 3: OS Services

Objectives

- Working with the OS
- Running external programs
- Walking through a directory tree
- Working with path names

The os module

- Provides OS-specific services

The **os** module provides many basic services from the operating system. The interface is the same for different operating systems. These services include file and folder utilities, as well as working with dates and times, running external programs, and many others.

Table 7. The os module

Method or Data	Description
path	either posixpath or ntpath
ctermid()	Return name of the controlling terminal
device_encoding()	Return string describing the encoding of the device
dup()	Return a duplicate of a file descriptor.
dup2()	Duplicate file descriptor.
exec…()	Execute file, with different configurations of arguments, environment, etc.
fchdir()	Change to directory of given file descriptor.
fchmod()	Change permissions of file given by file descriptor
fchown()	Change owner/group id of the file given by file descriptor
fdatasync()	force write of file with file descriptor to disk.
fork()	Fork a child process.
forkpty()	Fork a new process with a new pseudo-terminal
fpathconf()	Return the configuration limit name for the file descriptor
fstat()	Return stat result for an open file descriptor.
fstatvfs()	Return stat result for open file descriptor on virtual file system
fsync()	force write of file with filedescriptor to disk.
ftruncate()	Truncate a file to a specified length.
getcwd()	Return unicode string representing current working directory.
getegid()	Return the current process’s effective group id.
getenv()	Get specified environment variable or None/Default (returns string)
getenvb()	Get specified environment variable or None/Default (returns bytes)
geteuid()	Return the current process’s effective user id.
getgid()	Return the current process’s group id.
getgroups()	Return list of supplemental group IDs for the process.
getloadavg()	Return number of processes averaged over 1, 5, and 15 minutes
getlogin()	Return the actual login name.
getpgid()	Call the system call getpgid().
getpgrp()	Return the current process group id.
getpid()	Return the current process id
getppid()	Return the parent’s process id.

Method or Data	Description
getresgid()	Return tuple of real, effective, saved group IDs
getresuid()	Return tuple of real, effective, saved user IDs
getsid()	Call the system call getsid().
getuid()	Return the current process's user id.
initgroups()	Initialize the group access list with all groups of which the specified username is a member, plus the specified group id.
isatty()	Return True if file descriptor is an open file descriptor
kill()	Kill a process with a signal.
killpg()	Kill a process group with a signal.
lchown()	Change owner/group of path (don't follow symlinks)
link()	Create a hard link to a file.
listdir()	Return list of all entries in the directory.
lseek()	Set the current position of a file descriptor.
lstat()	Like stat(path), but do not follow symbolic links.
major()	Extracts device major number from a raw device number.
makedev()	Composes a raw device number from major/minor device numbers.
makedirs()	Super-mkdir (like unix mkdir -p)
minor()	Extracts device minor number from a raw device number.
mkdir()	Create a directory.
mkfifo()	Create a FIFO (a POSIX named pipe).
mknod()	Create a filesystem node
nice()	Decrease priority of process by inc and return new priority.
open()	Open a file (for low level IO).
openpty()	Open a pseudo-terminal
pathconf()	Return configuration limit name for file or directory path.
pipe()	Create a pipe.
putenv()	Change or add an environment variable.
read()	Read a file descriptor.
readlink()	Return string representation of symlink target
remove()	Remove a file (same as unlink(path)).
removedirs(name)	Super-rmdir; remove leaf directory and all empty intermediate ones

Method or Data	Description
rename()	Rename a file or directory.
renames()	Super-rename; create directories as necessary
rmdir()	Remove a directory.
setegid()	Set the current process's effective group id.
seteuid()	Set the current process's effective user id.
setgid()	Set the current process's group id.
setgroups()	Set the groups of the current process to list.
setpgid()	Call the system call setpgid().
setpgrp()	Make this process a session leader.
setregid()	Set the current process's real and effective group ids.
setresgid()	Set the current process's real, effective, and saved group ids.
setresuid()	Set the current process's real, effective, and saved user ids.
setreuid()	Set the current process's real and effective user ids.
setsid()	Call the system call setsid().
setuid()	Set the current process's user id.
spawn...()	Execute file with arguments from args in a subprocess.
stat()	Perform a stat system call on the given path.
stat_float_times()	Determine whether os.[lf]stat represents time stamps as float objects.
statvfs()	Perform a statvfs system call on the given path.
strerror()	Translate an error code to a message string.
symlink()	Create a symbolic link
sysconf()	Return an integer-valued system configuration variable.
system()	Execute the command (a string) in a subshell.
tcgetpgrp()	Return the process group associated with the terminal given by a fd.
tcsetpgrp()	Set the process group associated with the terminal given by a fd.
times()	Return tuple of floats indicating process times.
ttyname()	Return the name of the terminal device
umask()	Set the current numeric umask and return the previous umask.
uname()	Return a tuple identifying the current operating system.
unlink()	Remove a file (same as remove(path)).
unsetenv()	Delete an environment variable.

Method or Data	Description
utime()	Set the access and modified time of file
wait…()	Wait for completion of a child process.
walk()	Directory tree generator.
write()	Write a string to a file descriptor.

Paths, directories and file names

- import os.path module
- Many routines for working with file and folder attributes

The os.path module provides many functions for working with file and directory names and paths. These are all about the file and directories *attributes*, not the contents.

Some of the more common methods are

os.path.abspath()
os.path.basename
os.path.dirname()
os.path.getmtime()
os.path.getsize()
os.path.isdir()
os.path.isfile()
os.path.join()
os.path.exists()

Example

paths.py

```
#!/usr/bin/env python
import sys
import os.path

unix_p1 = "bin/spam.txt"  ①
unix_p2 = "/usr/local/bin/ham"  ②

win_p1 = r"spam\ham.doc"  ③
win_p2 = r"\\spam\ham\eggs\toast\jam.doc"  ④

if sys.platform == 'win32':  ⑤
    print("win_p1:", win_p1)
    print("win_p2:", win_p2)
    print("dirname(win_p1):", os.path.dirname(win_p1))  ⑥
    print("dirname(win_p2):", os.path.dirname(win_p2))
    print("basename(win_p1):", os.path.basename(win_p1))  ⑦
    print("basename(win_p2):", os.path.basename(win_p2))
    print("isabs(win_p1):", os.path.isabs(win_p1))  ⑧
    print("isabs(win_p2):", os.path.isabs(win_p2))
else:
    print("unix_p1:", unix_p1)
    print("unix_p2:", unix_p2)
    print("dirname(unix_p1):", os.path.dirname(unix_p1))  ⑥
    print("dirname(unix_p2):", os.path.dirname(unix_p2))
    print("basename(unix_p1):", os.path.basename(unix_p1))  ⑦
    print("basename(unix_p2):", os.path.basename(unix_p2))
    print("isabs(unix_p1):", os.path.isabs(unix_p1))  ⑧
    print("isabs(unix_p2):", os.path.isabs(unix_p2))
    print(
        'format("cp spam.txt {}".format(os.path.expanduser("~"))):',  ⑨
        format("cp spam.txt {}".format(os.path.expanduser("~"))),
    )
    print(
        'format("cd {}".format(os.path.expanduser("~root"))):',  ⑩
        format("cd {}".format(os.path.expanduser("~root"))),
    )
```

① Unix relative path

② Unix absolute path

③ Windows relative path

④ Windows UNC path

⑤ What platform are we on?

⑥ Just the folder name

⑦ Just the file (or folder) name

⑧ Is it an absolute path?

⑨ ~ is current user's home

⑩ ~NAME is NAME's home

paths.py

```
unix_p1: bin/spam.txt
unix_p2: /usr/local/bin/ham
dirname(unix_p1): bin
dirname(unix_p2): /usr/local/bin
basename(unix_p1): spam.txt
basename(unix_p2): ham
isabs(unix_p1): False
isabs(unix_p2): True
format("cp spam.txt {}".format(os.path.expanduser("~"))): cp spam.txt /Users/jstrick
format("cd {}".format(os.path.expanduser("~root"))): cd /var/root
```

Table 8. os.path methods

Method	Description
abspath(path)	Return normalized absolutized version of the pathname path.
basename(path)	Return the base name of pathname path.
commonprefix(list)	Return the longest path prefix (taken character-by-character) that is a prefix of all paths in list. If list is empty, return the empty string ('').
dirname(path)	Return the directory name of pathname path.
exists(path)	Return True if path refers to an existing path. Returns False for broken symbolic links. May be subject to permissions
lexists(path)	Return True if path refers to an existing path. Returns True for broken symbolic links.
expanduser(path)	On Unix, return the argument with an initial component of "~" or "~user" replaced by that user's home directory. Only "~" is supported on Windows.
expandvars(path)	Return the argument with environment variables expanded. Substrings of the form "$name" or "${name}" are replaced by the value of environment variable name. Malformed variable names and references to non-existing variables are left unchanged.
getatime(path)	Return the time of last access of path. (seconds since epoch).
getmtime(path)	Return the time of last modification of path. (seconds since epoch).
getctime(path)	Return the system's ctime. (seconds since epoch).
getsize(path)	Return the size, in bytes, of path. Raise os.error if path does not exist or cannot be accessed.
isabs(path)	Return True if path is an absolute pathname (begins with a slash).
isfile(path)	Return True if path is an existing regular file. This follows symbolic links.
isdir(path)	Return True if path is an existing directory. Follows symbolic links.
islink(path)	Return True if path refers to a directory entry that is a symbolic link. Always False on Windows.
ismount(path)	Return True if pathname path is a mount point (Unix only).
join(path1[, path2[, ...]])	Join one or more path components intelligently.
normcase(path)	Normalize the case of a pathname. On Unix, this returns the path unchanged; on case-insensitive filesystems, it converts the path to lowercase. On Windows, it also converts forward slashes to backward slashes.

Method	Description
normpath(ph)	Normalize a pathname. This collapses redundant separators and up-level references so that A//B, A/./B and A/foo/../B all become A/B.
realpath(path)	Return the canonical path of the specified filename, eliminating any symbolic links encountered in the path.
samefile(path1, path2)	Return True if both pathname arguments refer to the same file or directory (as indicated by device number and i-node number). Raise an exception if a os.stat() call on either pathname fails. Availability: Macintosh, Unix.
sameopenfile(fp1, fp2)	Return True if the file descriptors fp1 and fp2 refer to the same file. Availability: Macintosh, Unix.
samestat(stat1, stat2)	Return True if the stat tuples stat1 and stat2 refer to the same file. These structures may have been returned by fstat(), lstat(), or stat(). Availability: Macintosh, Unix.
split(path)	Split the pathname path into a pair, (head, tail) where tail is the last pathname component and head is everything leading up to that. The tail part will never contain a slash.
splitdrive(path)	Split the pathname path into a pair (drive, tail) where drive is either a drive specification or the empty string. On systems which do not use drive specifications, drive will always be the empty string..
splitext(path)	Split the pathname path into a pair (root, ext) such that root + ext == path, and ext is empty or begins with a period and contains at most one period.
splitunc(path)	Split the pathname path into a pair (unc, rest) so that unc is the UNC mount point (such as r'\\host\mount'), if present, and rest the rest of the path (such as r'\path\file.ext'). For paths containing drive letters, unc will always be the empty string. Availability: Windows.
walk(path, visit, arg)	Calls the function visit with arguments (arg, dirname, names) for each directory in the directory tree rooted at path (including path itself, if it is a directory). Note: The newer os.walk() generator supplies similar functionality and can be easier to use. (Like File::Find in Perl)
supports_unicode_filenames( )	True if arbitrary Unicode strings can be used as file names (within limitations imposed by the file system), and if os.listdir() returns Unicode strings for a Unicode argument. New in version 2.3.

Walking directory trees

- Use os.walk()
- Returns tuple for each directory
- Tuple contains directory path, subdirectories, and files

The os.walk method provides a way to easily walk a directory tree. It provides an iterator for a directory and all its subdirectories. For each directory, it returns a tuple with three values.

The first element is the full (absolute) path to the directory; the second element is a list of the directory's subdirectories (relative names); the third element is a list of the non-directory entries in the subdirectory (also relative names).

Be sure to use os.path.join() to put together the directory and the file or subdirectory name.

Do not use "dir" as a variable when looping through the iterator, because it will overwrite Python's builtin **dir** function.

Example

os_walk_ex.py

```
#!/usr/bin/env python
"""print size of every python file whose name starts with "m" """

import os

START_DIR = ".."  # start in root of student files  ①


def main():
    for currdir, subdirs, files in os.walk(START_DIR):  ②
        for file in files:  ③
            if file.endswith('.py') and file.startswith('m'):
                fullpath = os.path.join(currdir, file)  ④
                fsize = os.path.getsize(fullpath)
                print("{:8d} {:s}".format(fsize, fullpath))


if __name__ == '__main__':
    main()
```

① starting location

② walk folder tree

③ loop over file names

④ get file path

os_walk_ex.py

```
     469 ../custom/pynavy/1.0/f5_week2/EXAMPLES/modtest.py
     263 ../py3jpmcadv/production/junk/py3jpmcadv/ANSWERS/make_zip.py
     603 ../py3jpmcadv/production/junk/py3jpmcadv/ANSWERS/maxlist.py
    2433 ../py3jpmcadv/production/junk/py3jpmcadv/EXAMPLES/metaclass_generic.py
    1052 ../py3jpmcadv/production/junk/py3jpmcadv/EXAMPLES/multiindexlist.py
     458 ../py3jpmcadv/production/junk/py3jpmcadv/EXAMPLES/meta_monkey.py
    1959 ../py3jpmcadv/production/junk/py3jpmcadv/EXAMPLES/multi_processing.py
     881 ../py3jpmcadv/production/junk/py3jpmcadv/EXAMPLES/metaclass_singleton.py
     243 ../DATA/make_potus.py
    1176 ../acc_django_outlines/py3sci3day/EXAMPLES/mammal.py
```

...

Environment variables

- Shell or OS variables
- Same for Windows and non-Windows
- Syntax

```
value = os.environ[varname]
ivalue = os.environ.get(varname)
value = os.getenv(varname)
value = os.getenv(varname,default)
str2 = os.path.expandvars(str1)
```

There are several ways to access environment variables from Python.

The most direct is to use os.environ, which is a dictionary of the current environment. If a non-existent variable name is specified, a KeyError will be raised, so it is safer to use os.environ.get(varname[,default]) than os.environ[varname].

You can also use the os.getenv(varname[,default]) method. It takes the name of an environment variable and returns that variable's value. An optional second argument provides a default value if the variable is not defined.

Another way to use environment variables is to expand a string that contains them, using the expandvars(string) method of the os.path object. This takes a string containing one or more environment variables and returns the strings with environment variables expanded to their values.

If the variables do not exist in the environment, they are left unexpanded.

Example

getenv_ex.py

```
#!/usr/bin/env python

import sys
import os.path

if sys.platform == 'win32':
    user_key = 'USERNAME'
else:
    user_key = 'USER'

count_key = 'COUNT'

os.environ[count_key] = "42"  ①
print("count is", os.environ[count_key], "user is", os.environ[user_key])  ②
print("count is", os.environ.get(count_key), "user is", os.environ.get(user_key))  ③
user = os.getenv(user_key)  ④
count = os.getenv(count_key)
print("count is", count, "user is", user)
cmd = "count is ${} user is ${}".format(count_key, user_key)
print("cmd:", cmd)
print(os.path.expandvars(cmd))  ⑤
```

① set environment variable

② os.environ is a dictionary

③ os.environ.get() is safer than os.environ[]

④ os.getenv() is shortcut for os.environ.get()

⑤ expand variables in place; handy for translating shell scripts

getenv_ex.py

```
count is 42 user is jstrick
count is 42 user is jstrick
count is 42 user is jstrick
cmd: count is $COUNT user is $USER
count is 42 user is jstrick
```

Launching external programs

- Different ways to launch programs
  - Just launch (use system())
  - Capture output (use popen())
- import os module
- Use system() or popen() methods

In Python, you can launch an external command using the os module functions **os.system()** and **os.popen()**.

os.system() launches any external command, as though you had typed it at a command prompt. popen() opens a command, returning a file-like object. You can read the output of the command with any of the methods used for a file.

You can open a process for writing as well, by specifying a mode of "w".

TIP	For more sophisticated control of processes, see the **subprocess** module.

Example

external_programs.py

```
#!/usr/bin/env python
import os

os.system("hostname")  ①

with os.popen('netstat -an') as netstat_in:  ②
    for entry in netstat_in:  ③
        if 'ESTAB' in entry:  ④
            print(entry, end='')
print()
```

① Just run "hostname"

② Open command line "netstat -an" as a file-like object

③ Iterate over lines in output of "netstat -an"

④ Check to see if line contains "ESTAB"

external_programs.py

```
Johns-Macbook.attlocal.net
tcp6       0      0  2600:1700:3901:6.61923 2406:da00:ff00::.443   ESTABLISHED
tcp6       0      0  2600:1700:3901:6.61922 2607:f8b0:4002:c.443   ESTABLISHED
tcp6       0      0  2600:1700:3901:6.61917 2a03:2880:f011:1.443   ESTABLISHED
tcp6       0      0  2600:1700:3901:6.61888 2a03:2880:f011:1.443   ESTABLISHED
tcp4       0      0  192.168.1.242.61887    140.82.114.26.443      ESTABLISHED
tcp6       0      0  2600:1700:3901:6.61882 2a03:2880:f011:1.443   ESTABLISHED
tcp4       0      0  192.168.1.242.61880    52.109.16.72.443       ESTABLISHED
tcp6       0      0  2600:1700:3901:6.61813 2a03:2880:f011:1.443   ESTABLISHED
tcp6       0      0  2600:1700:3901:6.61272 2607:f8b0:4002:c.5228  ESTABLISHED
tcp6       0      0  2600:1700:3901:6.60211 2a03:2880:f011:1.443   ESTABLISHED
tcp4       0      0  192.168.1.242.58663    162.247.243.147.443    ESTABLISHED
tcp6       0      0  2600:1700:3901:6.59378 2a01:111:f100:20.443   ESTABLISHED
tcp6       0      0  fe80::aede:48ff:.58127 fe80::aede:48ff:.49198 ESTABLISHED
tcp6       0      0  fe80::aede:48ff:.58126 fe80::aede:48ff:.49184 ESTABLISHED
```

...

Chapter 3 Exercises

Exercise 3-1 (path_files.py)

List each folder in the PATH environment variable, together with the number of files it contains.

Output should look something like this:

Mac/Linux

```
/usr/bin            2376
/usr/local/bin        17
/usr/local/sbin        1
/usr/sbin            263
```

Windows

```
C:\ProgramData\Anaconda3: 84
C:\ProgramData\Anaconda3\Library\mingw-w64\bin: 13
WARNING: C:\ProgramData\Anaconda3\Library\usr\bin is in PATH, but does not exist
C:\ProgramData\Anaconda3\Library\bin: 387
C:\ProgramData\Anaconda3\Scripts: 304
C:\ProgramData\Anaconda3\bin: 1
C:\ProgramData\Anaconda3\condabin: 7
C:\ProgramData\Oracle\Java\javapath: 3
C:\WINDOWS\system32: 4536
C:\WINDOWS: 113
C:\WINDOWS\System32\Wbem: 382
C:\WINDOWS\System32\WindowsPowerShell\v1.0: 28
C:\WINDOWS\System32\OpenSSH: 7
C:\Program Files\Git\cmd: 6
```

TIP	Use os to get the pathsep value; then use os.listdir to get the contents of each directory after splitting PATH.

Exercise 3-2 (oldest_file.py)

Write a script that, given a directory on the command line, prints out the oldest file in that directory. If there is more than one file sharing the oldest timestamp, print any one of them.

TIP	Use os.path.getmtime()

Exercise 3-3 (all_python_lines.py)

Write a script that finds all the Python files (.py) in the student files (starting at py3nsextra), and counts the total number of lines in all of them.

Chapter 4: Serializing Data

Objectives

- Have a good understanding of the XML format
- Know which modules are available to process XML
- Use lxml ElementTree to create a new XML file
- Parse an existing XML file with ElementTree
- Using XPath for searching XML nodes
- Load JSON data from strings or files
- Write JSON data to strings or files
- Read and write CSV data
- Read and write YAML data

Which XML module to use?

- Bewildering array of XML modules
- Some are SAX, some are DOM
- Use xml.etree.ElementTree

When you are ready to process Python with XML, you turn to the standard library, only to find a number of different modules with confusing names.

To cut to the chase, use **lxml.etree**, which is based on **ElementTree** with some nice extra features, such as pretty-printing. While not part of the core Python library, it is provided by the Anaconda bundle.

If `lxml.etree` is not available, you can use **xml.etree.ElementTree** from the core library.

Getting Started With ElementTree

- Import xml.etree.ElementTree (or lxml.etree) as ET for convenience
- Parse XML or create empty ElementTree

ElementTree is part of the Python standard library; lxml is included with the Anaconda distribution.

Since putting "xml.etree.ElementTree" in front of its methods requires a lot of extra typing , it is typical to alias xml.etree.ElementTree to just ET when importing it: import xml.etree.ElementTree as ET

You can check the version of ElementTree via the VERSION attribute:

```
import xml.etree.ElementTree as ET
print(ET.VERSION)
```

How ElementTree Works

- ElementTree contains root Element
- Document is tree of Elements

In ElementTree, an XML document consists of a nested tree of Element objects. Each Element corresponds to an XML tag.

An ElementTree object serves as a wrapper for reading or writing the XML text.

If you are parsing existing XML, use ElementTree.parse(); this creates the ElementTree wrapper and the tree of Elements. You can then navigate to, or search for, Elements within the tree. You can also insert and delete new elements.

If you are creating a new document from scratch, create a top-level (AKA "root") element, then create child elements as needed.

```
element = root.find('sometag')
for subelement in element:
    print(subelement.tag)
print(element.get('someattribute'))
```

Elements

- Element has
  - Tag name
  - Attributes (implemented as a dictionary)
  - Text
  - Tail
  - Child elements (implemented as a list) (if any)
- SubElement creates child of Element

When creating a new Element, you can initialize it with the tag name and any attributes. Once created, you can add the text that will be contained within the element's tags, or add other attributes.

When you are ready to save the XML into a file, initialize an ElementTree with the root element.

The **Element** class is a hybrid of list and dictionary. You access child elements by treating it as a list. You access attributes by treating it as a dictionary. (But you can't use subscripts for the attributes – you must use the get() method).

The Element object also has several useful properties: **tag** is the element's tag; **text** is the text contained inside the element; **tail** is any text following the element, before the next element.

The **SubElement** class is a convenient way to add children to an existing Element.

TIP	Only the tag property of an Element is required; other properties are optional.

Table 9. Element properties and methods

Property	Description
append(element)	Add a subelement element to end of subelements
attrib	Dictionary of element's attributes
clear()	Remove all subelements
find(path)	Find first subelement matching path
findall(path)	Find all subelements matching path
findtext(path)	Shortcut for find(path).text
get(attr)	Get an attribute; Shortcut for attrib.get()
getiterator()	Returns an iterator over all descendants
getiterator(path)	Returns an iterator over all descendants matching path
insert(pos,element)	Insert subelement element at position pos
items()	Get all attribute values; Shortcut for attrib.items()
keys()	Get all attribute names; Shortcut for attrib.keys()
remove(element)	Remove subelement element
set(attrib,value)	Set an attribute value; shortcut for attr[attrib] = value
tag	The element's tag
tail	Text following the element
text	Text contained within the element

Table 10. ElementTree properties and methods

Property	Description
find(path)	Finds the first toplevel element with given tag; shortcut for getroot().find(path).
findall(path)	Finds all toplevel elements with the given tag; shortcut for getroot().findall(path).
findtext(path)	Finds element text for first toplevel element with given tag; shortcut for getroot().findtext(path).
getiterator(path)	Returns an iterator over all descendants of root node matching path. (All nodes if path not specified)
getroot()	Return the root node of the document
parse(filename) parse(fileobj)	Parse an XML source (filename or file-like object)
write(filename,encoding)	Writes XML document to filename, using encoding (Default us-ascii).

Creating a New XML Document

- Create root element
- Add descendants via SubElement
- Use keyword arguments for attributes
- Add text after element created
- Create ElementTree for import/export

To create a new XML document, first create the root (top-level) element. This will be a container for all other elements in the tree. If your XML document contains books, for instance, the root document might use the "books" tag. It would contain one or more "book" elements, each of which might contain author, title, and ISBN elements.

Once the root element is created, use SubElement to add elements to the root element, and then nested Elements as needed. SubElement returns the new element, so you can assign the contents of the tag to the **text** attribute.

Once all the elements are in place, you can create an ElementTree object to contain the elements and allow you to write out the XML text. From the ElementTree object, call write.

To output an XML string from your elements, call ET.tostring(), passing the root of the element tree as a parameter. It will return a bytes object (pure ASCII), so use .decode() to convert it to a normal Python string.

For an example of creating an XML document from a data file, see **xml_create_knights.py** in the EXAMPLES folder

Example

xml_create_movies.py

```
#!/usr/bin/env python

# from xml.etree import ElementTree as ET
import lxml.etree as ET

movie_data = [
    ('Jaws', 'Spielberg, Stephen'),
    ('Vertigo', 'Alfred Hitchcock'),
    ('Blazing Saddles', 'Brooks, Mel'),
    ('Princess Bride', 'Reiner, Rob'),
    ('Avatar', 'Cameron, James'),
]

movies = ET.Element('movies')

for name, director in movie_data:
    movie = ET.SubElement(movies, 'movie', name=name)
    ET.SubElement(movie, 'director').text = director

print(ET.tostring(movies, pretty_print=True).decode())

doc = ET.ElementTree(movies)

doc.write('movies.xml')
```

xml_create_movies.py

```
<movies>
  <movie name="Jaws">
    <director>Spielberg, Stephen</director>
  </movie>
  <movie name="Vertigo">
    <director>Alfred Hitchcock</director>
  </movie>
  <movie name="Blazing Saddles">
    <director>Brooks, Mel</director>
  </movie>
  <movie name="Princess Bride">
    <director>Reiner, Rob</director>
  </movie>
  <movie name="Avatar">
    <director>Cameron, James</director>
  </movie>
</movies>
```

Parsing An XML Document

- Use ElementTree.parse()
- returns an ElementTree object
- Use get* or find* methods to select an element

Use the parse() method to parse an existing XML document. It returns an ElementTree object, from which you can find the root, or any other element within the document.

To get the root element, use the getroot() method.

Example

```
import xml.etree.ElementTree as ET

doc = ET.parse('solar.xml')

root = doc.getroot()
```

Navigating the XML Document

- Use find() or findall()
- Element is iterable of it children
- findtext() retrieves text from element

To find the first child element with a given tag, use find('tag'). This will return the first matching element. The findtext('tag') method is the same, but returns the text within the tag.

To get all child elements with a given tag, use the findall('tag') method, which returns a list of elements.

to see whether a node was found, say

```
if node is None:
```

but to check for existence of child elements, say

```
if len(node) > 0:
```

A node with no children tests as false because it is an empty list, but it is not None.

TIP The ElementTree object also supports the find() and findall() methods of the Element object, searching from the root object.

Example

xml_planets_nav.py

```
#!/usr/bin/env python
'''Use etree navigation to extract planets from solar.xml'''
import lxml.etree as ET


def main():
    '''Program entry point'''
    doc = ET.parse('../DATA/solar.xml')  ①

    solar_system = doc.getroot()  ②

    print(solar_system)
    print()

    inner = solar_system.find('innerplanets')  ③
    print('Inner:')

    for planet in inner:  ④
        if planet.tag == 'planet':
            print('\t', planet.get("planetname", "NO NAME"))

    outer = solar_system.find('outerplanets')
    print('Outer:')

    for planet in outer:
        print('\t', planet.get("planetname"))

    plutoids = solar_system.find('dwarfplanets')
    print('Dwarf:')

    for planet in plutoids:
        print('\t', planet.get("planetname"))


if __name__ == '__main__':
    main()
```

xml_planets_nav.py

```
<Element solarsystem at 0x7fd3a821c5a0>

Inner:
     Mercury
     Venus
     Earth
     Mars
Outer:
     Jupiter
     Saturn
     Uranus
     Neptune
Dwarf:
     Pluto
```

Example

xml_read_movies.py

```
#!/usr/bin/env python

# import xml.etree.ElementTree as ET
import lxml.etree as ET

movies_doc = ET.parse('movies.xml')  ①

movies = movies_doc.getroot()  ②

for movie in movies:  ③
    print('{} by {}'.format(
        movie.get('name'),  ④
        movie.findtext('director'),  ⑤
    )
    )
```

① read and parse the XML file

② get the root element (<movies>)

③ loop through children of root element

④ get 'name' attribute of movie element

⑤ get 'director' attribute of movie element

xml_read_movies.py

```
Jaws by Spielberg, Stephen
Vertigo by Alfred Hitchcock
Blazing Saddles by Brooks, Mel
Princess Bride by Reiner, Rob
Avatar by Cameron, James
```

Using XPath

- Use simple XPath patterns Works with find* methods

When a simple tag is specified, the find* methods only search for subelements of the current element. For more flexible searching, the find* methods work with simplified **XPath** patterns. To find all tags named 'spam', for instance, use `.//spam`.

```
.//movie
presidents/president/name/last
```

Example

xml_planets_xpath1.py

```
#!/usr/bin/env python

# import xml.etree.ElementTree as ET
import lxml.etree as ET

doc = ET.parse('../DATA/solar.xml')  ①

inner_nodes = doc.findall('innerplanets/planet')  ②

outer_nodes = doc.findall('outerplanets/planet')  ③

print('Inner:')
for planet in inner_nodes:  ④
    print('\t', planet.get("planetname"))  ⑤

print('Outer:')
for planet in outer_nodes:  ④
    print('\t', planet.get("planetname"))  ⑤
```

① parse XML file

② find all elements (relative to root element) with tag "planet" under "innerplanets" element

③ find all elements with tag "planet" under "outerplanets" element

④ loop through search results

⑤ print "name" attribute of planet element

xml_planets_xpath1.py

```
Inner:
     Mercury
     Venus
     Earth
     Mars
Outer:
     Jupiter
     Saturn
     Uranus
     Neptune
```

Example

xml_planets_xpath2.py

```
#!/usr/bin/env python

# import xml.etree.ElementTree as ET
import lxml.etree as ET

doc = ET.parse('../DATA/solar.xml')

jupiter = doc.find('.//planet[@planetname="Jupiter"]')

if jupiter is not None:
    for moon in jupiter:
        print(moon.text)  # grab attribute
```

xml_planets_xpath2.py

```
Metis
Adrastea
Amalthea
Thebe
Io
Europa
Gannymede
Callista
Themisto
Himalia
Lysithea
Elara
```

Table 11. ElementTree XPath Summary

Syntax	Meaning
tag	Selects all child elements with the given tag. For example, “spam” selects all child elements named “spam”, “spam/egg” selects all grandchildren named “egg” in all child elements named “spam”. You can use universal names (“{url}local”) as tags.
*	Selects all child elements. For example, “*/egg” selects all grandchildren named “egg”.
.	Select the current node. This is mostly useful at the beginning of a path, to indicate that it’s a relative path.
//	Selects all subelements, on all levels beneath the current element (search the entire subtree). For example, “.//egg” selects all “egg” elements in the entire tree.
..	Selects the parent element.
[@attrib]	Selects all elements that have the given attribute. For example, “.//a[@href]” selects all “a” elements in the tree that has a “href” attribute.
[@attrib=’value’]	Selects all elements for which the given attribute has the given value. For example, “.//div[@class=’sidebar’]” selects all “div” elements in the tree that has the class “sidebar”. In the current release, the value cannot contain quotes.
parent_tag[*child_tag*]	Selects all parent elements that has a child element named *child_tag*. In the current version, only a single tag can be used (i.e. only immediate children are supported). Parent tag can be *****.

About JSON

- Lightweight, human-friendly format for data
- Contains dictionaries and lists
- Stands for JavaScript Object Notation
- Looks like Python
- Basic types: Number, String, Boolean, Array, Object
- White space is ignored
- Stricter rules than Python

JSON is a lightweight and human-friendly format for sharing or storing data. It was developed and popularized by Douglas Crockford starting in 2001.

A JSON file contains objects and arrays, which correspond exactly to Python dictionaries and lists.

White space is ignored, so JSON may be formatted for readability.

Data types are Number, String, and Boolean. Strings are enclosed in double quotes (only); numbers look like integers or floats; Booleans are represented by true or false; null (None in Python) is represented by null.

Reading JSON

- json module in standard library
- json.load() parse from file-like object
- json.loads() parse from string
- Both methods return Python dict or list

To read a JSON file, import the json module. Use json.loads() to parse a string containing valid JSON. Use json.load() to read JSON from a file-like object0.

Both methods return a Python dictionary containing all the data from the JSON file.

Example

json_read.py

```
#!/usr/bin/env python

import json

with open('../DATA/solar.json') as solar_in:  ①
    solar = json.load(solar_in)  ②

# json.loads(STRING)
# json.load(FILE_OBJECT)

# print(solar)

print(solar['innerplanets'])  ③
print('*' * 60)
print(solar['innerplanets'][0]['name'])
print('*' * 60)
for planet in solar['innerplanets'] + solar['outerplanets']:
    print(planet['name'])

print("*" * 60)
for group in solar:
    if group.endswith('planets'):
        for planet in solar[group]:
            print(planet['name'])
```

① open JSON file for reading

② load from file object and convert to Python data structure

③ solar is just a Python dictionary

json_read.py

```
[{'name': 'Mercury', 'moons': None}, {'name': 'Venus', 'moons': None}, {'name': 'Earth',
'moons': ['Moon']}, {'name': 'Mars', 'moons': ['Deimos', 'Phobos']}]
************************************************************
Mercury
************************************************************
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
************************************************************
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
Pluto
```

Writing JSON

- Use json.dumps() or json.dump()

To output JSON to a string, use json.dumps(). To output JSON to a file, pass a file-like object to json.dump(). In both cases, pass a Python data structure as the data to be output.

Example

json_write.py

```
#!/usr/bin/env python

import json

george = [
    {
        'num': 1,
        'lname': 'Washington',
        'fname': 'George',
        'dstart': [1789, 4, 30],
        'dend': [1797, 3, 4],
        'birthplace': 'Westmoreland County',
        'birthstate': 'Virginia',
        'dbirth': [1732, 2, 22],
        'ddeath': [1799, 12, 14],
        'assassinated': False,
        'party': None,
    },
    {
        'spam': 'ham',
        'eggs': [1.2, 2.3, 3.4],
        'toast': {'a': 5, 'm': 9, 'c': 4},
    }
]  ①

js = json.dumps(george, indent=4)  ②
print(js)

with open('george.json', 'w') as george_out:  ③
    json.dump(george, george_out, indent=4)  ④
```

① Python data structure

② dump structure to JSON string

③ open file for writing

④ dump structure to JSON file using open file object

json_write.py

```
[
    {
        "num": 1,
        "lname": "Washington",
        "fname": "George",
        "dstart": [
            1789,
            4,
            30
        ],
        "dend": [
            1797,
            3,
            4
        ],
        "birthplace": "Westmoreland County",
        "birthstate": "Virginia",
        "dbirth": [
            1732,
            2,
            22
        ],
        "ddeath": [
            1799,
            12,
            14
        ],
        "assassinated": false,
        "party": null
    },
    {
        "spam": "ham",
        "eggs": [
            1.2,
            2.3,
            3.4
        ],
        "toast": {
            "a": 5,
            "m": 9,
            "c": 4
        }
    }
]
```

Customizing JSON

- JSON data types limited
- simple cases — dump dict
- create custom encoders

The JSON spec only supports a limited number of datatypes. If you try to dump a data structure contains dates, user-defined classes, or many other types, the json encoder will not be able to handle it.

You can a custom encoder for various data types. To do this, write a function that expects one Python object, and returns some object that JSON can parse, such as a string or dictionary. The function can be called anything. Specify the function with the **default** parameter to json.dump().

The function should check the type of the object. If it is a type that needs special handling, return a JSON-friendly version, otherwise just return the original object.

Table 12. Python types that JSON can encode

Python	JSON
dict	object
list	array
str	string
int	number (int)
float	number (real)
True	true
False	false
None	null

NOTE see the file **json_custom_singledispatch.py** in EXAMPLES for how to use the **singledispatch** decorator (in the **functools** module to handle multiple data types.

Example

json_custom_encoding.py

```
#!/usr/bin/env python
#
import json
from datetime import date


class Parrot():  ①
    def __init__(self, name, color):
        self._name = name
        self._color = color

    @property
    def name(self):  ②
        return self._name

    @property
    def color(self):
        return self._color


parrots = [  ③
    Parrot('Polly', 'green'),  #
    Parrot('Peggy', 'blue'),
    Parrot('Roger', 'red'),
]


def encode(obj):  ④
    if isinstance(obj, date):  ⑤
        return obj.ctime()  ⑥
    elif isinstance(obj, Parrot):  ⑦
        return {'name': obj.name, 'color': obj.color}  ⑧
    return obj  ⑨


data = {  ⑩
    'spam': [1, 2, 3],
    'ham': ('a', 'b', 'c'),
    'toast': date(2014, 8, 1),
    'parrots': parrots,
}

print(json.dumps(data, default=encode, indent=4))  ⑪
```

① sample user-defined class (not JSON-serializable)

② JSON does not understand arbitrary properties

③ list of Parrot objects

④ custom JSON encoder function

⑤ check for date object

⑥ convert date to string

⑦ check for Parrot object

⑧ convert Parrot to dictionary

⑨ if not processed, return object for JSON to parse with default parser

⑩ dictionary of arbitrary data

⑪ convert Python data to JSON data; 'default' parameter specifies function for custom encoding; 'indent' parameter says to indent and add newlines for readability

json_custom_encoding.py

```
{
    "spam": [
        1,
        2,
        3
    ],
    "ham": [
        "a",
        "b",
        "c"
    ],
    "toast": "Fri Aug  1 00:00:00 2014",
    "parrots": [
        {
            "name": "Polly",
            "color": "green"
        },
        {
            "name": "Peggy",
            "color": "blue"
        },
        {
            "name": "Roger",
            "color": "red"
        }
    ]
}
```

Reading and writing YAML

- yaml module from PYPI
- syntax like **json** module
- yaml.load(), dump() parse from/to file-like object
- yaml.loads(), dumps() parse from/to string

YAML is a structured data format which is a superset of JSON. However, YAML allows for a more compact and readable format.

Reading and writing YAML uses the same syntax as JSON, other than using the **yaml** module, which is NOT in the standard library. To install the **yaml** module:

```
pip install pyyaml
```

To read a YAML file (or string) into a Python data structure, use `yaml.load(file_object)` or `yaml.loads(string)`.

To write a data structure to a YAML file or string, use `yaml.dump(data, file_object)` or `yaml.dumps(data)`.

You can also write custom YAML processors.

NOTE	YAML parsers will parse JSON data

Example

yaml_read_solar.py

```
import yaml

PLANET_SECTIONS = "inner outer plutoid".split()

with open('../DATA/solar.yaml') as solar_in:
    solar_data = yaml.load(solar_in, Loader=yaml.FullLoader)

star = solar_data['star']
print("Our star is {}\n".format(star))

for section in PLANET_SECTIONS:
    for planet in solar_data[section]:
        print(planet['name'])
        for moon in planet['moons']:
            print("\t{}".format(moon))
```

yaml_read_solar.py

```
Our star is Sun

Mercury
    None
Venus
    None
Earth
    Moon
Mars
    Deimos
    Phobos
    Metis
Jupiter
    Adrastea
    Amalthea
    Thebe
    Io
    Europa
    Gannymede
    Callista
    Themisto
    Himalia
    Lysithea
    Elara
Saturn
    Rhea
    Hyperion
    Titan
    Iapetus
    Mimas
```

...

Example

yaml_create_file.py

```
import sys
from datetime import date
import yaml

potus = {
    'presidents': [
         {
            'lastname': 'Washington',
            'firstname': 'George',
            'dob': date(1732, 2, 22),
            'dod': date(1799, 12, 14),
            'birthplace': 'Westmoreland County',
            'birthstate': 'Virginia',
            'term': [ date(1789, 4, 30), date(1797, 3, 4) ],
            'assassinated': False,
            'party': None,
        },
        {
            'lastname': 'Adams',
            'firstname': 'John',
            'dob': date(1735, 10, 30),
            'dod': date(1826, 7, 4),
            'birthplace': 'Braintree, Norfolk',
            'birthstate': 'Massachusetts',
            'term': [date(1797, 3, 4), date(1801, 3, 4)],
            'assassinated': False,
            'party': 'Federalist',

        }
    ]
}

with open('potus.yaml', 'w') as potus_out:
    yaml.dump(potus, potus_out)

yaml.dump(potus, sys.stdout)
```

yaml_create_file.py

```
presidents:
- assassinated: false
  birthplace: Westmoreland County
  birthstate: Virginia
  dob: 1732-02-22
  dod: 1799-12-14
  firstname: George
  lastname: Washington
  party: null
  term:
  - 1789-04-30
  - 1797-03-04
- assassinated: false
  birthplace: Braintree, Norfolk
  birthstate: Massachusetts
  dob: 1735-10-30
  dod: 1826-07-04
  firstname: John
  lastname: Adams
  party: Federalist
  term:
  - 1797-03-04
  - 1801-03-04
```

Reading CSV data

- Use csv module
- Create a reader with any iterable (e.g. file object)
- Understands Excel CSV and tab-delimited files
- Can specify alternate configuration
- Iterate through reader to get rows as lists of columns

To read CSV data, use the reader() method in the csv module.

To create a reader with the default settings, use the reader() constructor. Pass in an iterable – typically, but not necessarily, a file object.

You can also add parameters to control the type of quoting, or the output delimiters.

Example

csv_read.py

```
#!/usr/bin/env python
import csv

with open('../DATA/knights.csv') as knights_in:
    rdr = csv.reader(knights_in)  ①
    for name, title, color, quest, comment, number, ladies in rdr:  ②
        print('{:4s} {:9s} {}'.format(
            title, name, quest
        ))
```

① create CSV reader

② Read and unpack records one at a time; each record is a list

csv_read.py

```
King Arthur    The Grail
Sir  Lancelot  The Grail
Sir  Robin     Not Sure
Sir  Bedevere  The Grail
Sir  Gawain    The Grail
```

...

Nonstandard CSV

- Variations in how CSV data is written
- Most common alternate is for Excel
- Add parameters to reader/writer

You can customize how the CSV parser and generator work by passing extra parameters to csv.reader() or csv.writer(). You can change the field and row delimiters, the escape character, and for output, what level of quoting.

You can also create a "dialect", which is a custom set of CSV parameters. The csv module includes one extra dialect, **excel**, which handles CSV files generated by Microsoft Excel. To use it, specify the *dialect* parameter:

```
rdr = csv.reader(csvfile, dialect='excel')
```

Table 13. CSV reader()/writer() Parameters

Parameter	Meaning
quotechar	One-character string to use as quoting character (default: '"')
delimiter	One-character string to use as field separator (default: ',')
skipinitialspace	If True, skip white space after field separator (default: False)
lineterminator	The character sequence which terminates rows (default: depends on OS)
quoting	When should quotes be generated when writing CSV csv.QUOTE_MINIMAL – only when needed (default) csv.QUOTE_ALL – quote all fields csv.QUOTE_NONNUMERIC – quote all fields that are not numbers csv.QUOTE_NONE – never put quotes around fields
escapechar	One-character string to escape delimiter when quoting is set to csv.QUOTE_NONE
doublequote	Control quote handling inside fields. When True, two consecutive quotes are read as one, and one quote is written as two. (default: True)

Example

csv_nonstandard.py

```
#!/usr/bin/env python
import csv

with open('../DATA/computer_people.txt') as computer_people_in:
    rdr = csv.reader(computer_people_in, delimiter=';')  ①

    for first_name, last_name, known_for, birth_date in rdr:  ②
        print('{}: {}'.format(last_name, known_for))
```

① specify alternate field delimiter

② iterate over rows of data — csv reader is a generator

csv_nonstandard.py

```
Gates: Gates Foundation
Jobs: Apple
Wall: Perl
Allen: Microsoft
Ellison: Oracle
Gates: Microsoft
Zuckerberg: Facebook
Brin: Google
Page: Google
Torvalds: Linux
```

Using csv.DictReader

- Returns each row as dictionary
- Keys are field names
- Use header or specify

Instead of the normal reader, you can create a dictionary-based reader by using the DictReader class.

If the CSV file has a header, it will parse the header line and use it as the field names. Otherwise, you can specify a list of field names with the **fieldnames** parameter. For each row, you can look up a field by name, rather than position.

Example

csv_dictreader.py

```
#!/usr/bin/env python
import csv

field_names = ['term', 'firstname', 'lastname', 'birthplace', 'state', 'party'] ①

with open('../DATA/presidents.csv') as presidents_in:
    rdr = csv.DictReader(presidents_in, fieldnames=field_names)  ②
    for row in rdr:  ③
        print('{:25s} {:12s} {}'.format(row['firstname'], row['lastname'], row['party']))
④
        # string .format can use keywords from an unpacked dict as well:
        # print('{firstname:25s} {lastname:12s} {party}'.format(**row))
```

① field names, which will become dictionary keys on each row

② create reader, passing in field names (if not specified, uses first row as field names)

③ iterate over rows in file

④ print results with formatting

csv_dictreader.py

```
George                  Washington   no party
John                    Adams        Federalist
Thomas                  Jefferson    Democratic - Republican
James                   Madison      Democratic - Republican
James                   Monroe       Democratic - Republican
John Quincy             Adams        Democratic - Republican
Andrew                  Jackson      Democratic
Martin                  Van Buren    Democratic
William Henry           Harrison     Whig
John                    Tyler        Whig
James Knox              Polk         Democratic
Zachary                 Taylor       Whig
Millard                 Fillmore     Whig
Franklin                Pierce       Democratic
James                   Buchanan     Democratic
Abraham                 Lincoln      Republican
Andrew                  Johnson      Republican
Ulysses Simpson         Grant        Republican
Rutherford Birchard     Hayes        Republican
James Abram             Garfield     Republican
```

...

Writing CSV Data

- Use csv.writer()
- Parameter is file-like object (must implement write() method)
- Can specify parameters to writer constructor
- Use writerow() or writerows() to output CSV data

To output data in CSV format, first create a writer using csv.writer(). Pass in a file-like object.

For each row to write, call the writerow() method of the writer, passing in an iterable with the values for that row.

To modify how data is written out, pass parameters to the writer.

TIP On Windows, to prevent double-spaced output, add `lineterminator='\n'` when creating a CSV writer.

Example

csv_write.py

```
#!/usr/bin/env python
import sys
import csv

chicago_data = [
    ['Name', 'Position Title', 'Department', 'Employee Annual Salary'],
    ['BONADUCE,  MICHAEL J', 'POLICE OFFICER', 'POLICE', '$80724.00'],
    ['MELLON,  MATTHEW J "Matt"', 'POLICE OFFICER', 'POLICE', '$75372.00'],
    ['FIERI,  JOHN J', 'FIREFIGHTER-EMT', 'FIRE', '$75342.00'],
    ['GALAHAD,  MERLE S', 'CLERK III', 'BUSINESS AFFAIRS', '$45828.00'],
    ['ORCATTI,  JENNIFER L', 'FIRE COMMUNICATIONS OPERATOR I', 'OEMC', '$63121.68'],
    ['ASHE,  JOHN W', 'FOREMAN OF MACHINISTS', 'AVIATION', '$96553.60'],
    ['SADINSKY BLAKE,  MICHAEL G', 'POLICE OFFICER', 'POLICE', '$78012.00'],
    ['GRANT,  CRAIG A', 'SANITATION LABORER', 'STREETS & SAN', '$69576.00'],
    ['MILLER,  JONATHAN D', 'POLICE OFFICER', 'POLICE', '$75372.00'],
    ['FRANK,  ARTHUR R',
    'POLICE OFFICER/EXPLSV DETECT, K9 HNDLR',
    'POLICE',
    '$87918.00'],
    ['POVOTTI,  JAMES S "Jimmy P"', 'TRAFFIC CONTROL AIDE-HOURLY', 'OEMC', '$19167.20'],
    ['TRAWLER,  DANIEL J', 'POLICE OFFICER', 'POLICE', '$75372.00'],
    ['SCUBA,  ANDREW G', 'POLICE OFFICER', 'POLICE', '$75372.00'],
    ['SWINE,  MATTHEW W', 'SERGEANT', 'POLICE', '$99756.00'],
    ['''RYDER,  MYRTA T "Lil'Myrt"''', 'POLICE OFFICER', 'POLICE', '$83706.00'],
    ['KORSHAK,  ROMAN', 'PARAMEDIC', 'FIRE', '$75372.00']
]

with open('../TEMP/chi_data.csv', 'w') as chi_out:
    # if sys.platform == 'win32':
    wtr = csv.writer(chi_out, lineterminator='\n') ①
    # else:
    #     wtr = csv.writer(stuff_in) ①
    for data_row in chicago_data:
        data_row[-1] = data_row[-1].lstrip('$')  # strip leading $ from last field
        wtr.writerow(data_row) ②
```

① create CSV writer from file object that is opened for writing; on windows, need to set output line terminator to '\n'

② write one row (of iterables) to output file

Pickle

- Use the pickle module
- Create a binary stream that can be saved to file
- Can also be transmitted over the network

Python uses the pickle module for data serialization.

To create pickled data, use either pickle.dump() or pickle.dumps(). Both functions take a data structure as the first argument. dumps() returns the pickled data as a string. dump () writes the data to a file-like object which has been specified as the second argument. The file-like object must be opened for writing.

To read pickled data, use pickle.load(), which takes a file-like object that has been open for writing, or pickle.loads() which reads from a string. Both functions return the original data structure that had been pickled.

NOTE	The syntax of the **json** module is based on the **pickle** module.

Example

pickling.py

```
#!/usr/bin/env python
import pickle
from pprint import pprint

①
airports = {
    'RDU': 'Raleigh-Durham', 'IAD': 'Dulles', 'MGW': 'Morgantown',
    'EWR': 'Newark', 'LAX': 'Los Angeles', 'ORD': 'Chicago'
}

colors = [
    'red', 'blue', 'green', 'yellow', 'black',
    'white', 'orange', 'brown', 'purple'
]

data = [  ②
    colors,
    airports,
]

with open('../TEMP/pickled_data.pic', 'wb') as pic_out:  ③
    pickle.dump(data, pic_out)  ④

with open('../TEMP/pickled_data.pic', 'rb') as pic_in:  ⑤
    pickled_data = pickle.load(pic_in)  ⑥

pprint(pickled_data)  ⑦
```

① some data structures

② list of data structures

③ open pickle file for writing in binary mode

④ serialize data structures to pickle file

⑤ open pickle file for reading in binary mode

⑥ de-serialize pickle file back into data structures

⑦ view data structures

pickling.py

```
[['red',
  'blue',
  'green',
  'yellow',
  'black',
  'white',
  'orange',
  'brown',
  'purple'],
 {'EWR': 'Newark',
  'IAD': 'Dulles',
  'LAX': 'Los Angeles',
  'MGW': 'Morgantown',
  'ORD': 'Chicago',
  'RDU': 'Raleigh-Durham'}]
```

Chapter 4 Exercises

Exercise 4-1 (xwords.py)

Using ElementTree, create a new XML file containing all the words that start with 'x' from words.txt. The root tag should be named 'words', and each word should be contained in a 'word' tag. The finished file should look like this:

```
<words>
   <word>xanthan</word>
   <word>xanthans</words>
   and so forth
</words>
```

Exercise 4-2 (xpresidents.py)

Use ElementTree to parse presidents.xml. Loop through and print out each president's first and last names and their state of birth.

Exercise 4-3 (jpresidents.py)

Rewrite xpresidents.py to parse presidents.json using the json module.

Exercise 4-4 (cpresidents.py)

Rewrite xpresidents.py to parse presidents.csv using the csv module.

Exercise 4-5 (pickle_potus.py)

Write a script which reads the data from presidents.csv into an dictionary where the key is the term number, and the value is another dictionary of data for one president.

Using the pickle module, Write the entire dictionary out to a file named presidents.pic.

Exercise 4-6 (unpickle_potus.py)

Write a script to open presidents.pic, and restore the data back into a dictionary.

Then loop through the array and print out each president's first name, last name, and party.

Chapter 5: Network Programming

Objectives

- Download web pages or file from the Internet
- Consume web services
- Send e-mail using a mail server
- Learn why requests is the best HTTP client

Making HTTP requests

- Use the **requests** module
- Pythonic front end to urllib, urllib2, httplib, etc
- Makes HTTP transactions simple

The standard library provides the **urllib** package. It and its friends are powerful libraries, but their interfaces are complex for non-trivial tasks. There is a lot of code to write if you want to provide authentication, proxies, headers, or data, among other things.

The **requests** module is a much easier to use HTTP client module. It is included with the **Anaconda** distribution, or is readily available from **PyPI**.

requests implements GET, POST, PUT, and other HTTP verbs, and takes care of all the protocol housekeeping needed to send data on the URL, to send a username/password, and to retrieve data in various formats.

To use **requests**, import the module and then call **requests.*VERB***, where *VERB* is "get", "post", "put", "patch", "delete", or "head". The first argument to any of these methods is the URL, followed by any of the named parameters for fine-tuning the request.

These methods return an HTTPResponse object, which contains the headers and data returned from the HTTP server. If the URL refers to a web page, then the **text** attribute contains the text of the page as a Python string.

In all cases, the **content** attribute contains the raw content from the server as a **bytes** string. If the returned data is a JSON string, the **json()** method converts the JSON data into a Python nested list or dictionary.

The **status_code** attribute contains the HTTP status code, normally 200 for a successful request.

For GET requests, URL parameters can be specified as a dictionary, using the **params** parameter.

For POST, PUT, or PATCH requests, the data to be uploaed can be specified as a dictionary using the **data** parameter.

TIP See details of the **requests** API at http://docs.python-requests.org/en/v3.0.0/api/#main-interface

Example

read_html_requests.py

```
#!/usr/bin/env python
import requests

response = requests.get("https://www.python.org")  ①

for header, value in sorted(response.headers.items()): ②
    print("{:20.20s} {}".format(header, value))
print()

print(response.text[:200])   ③
print('...')
print(response.text[-200:])   ④
```

① requests.get() returns HTTP response object

② response.headers is a dictionary of the headers

③ The text is returned as a bytes object, so it needs to be decoded to a string; print the first 200 bytes

④ print the last 200 bytes

Example

read_pdf_requests.py

```
#!/usr/bin/env python

import sys
import os

import requests

url = 'https://www.nasa.gov/pdf/739318main_ISS%20Utilization%20Brochure
%202012%20Screenres%203-8-13.pdf'  ①
saved_pdf_file = 'nasa_iss.pdf'  ②

response = requests.get(url)  ③
if response.status_code == requests.codes.OK:  ④
    if response.headers.get('content-type') == 'application/pdf':
        with open(saved_pdf_file, 'wb') as pdf_in:  ⑤
            pdf_in.write(response.content)  ⑥

        if sys.platform == 'win32':  ⑦
            cmd = saved_pdf_file
        elif sys.platform == 'darwin':
            cmd = 'open ' + saved_pdf_file
        else:
            cmd = 'acroread ' + saved_pdf_file

        os.system(cmd)  ⑧
```

① target URL

② name of PDF file for saving

③ open the URL

④ check status code

⑤ open local file

⑥ write data to a local file in binary mode; response.content is data from URL

⑦ select platform and choose the app to open the PDF file

⑧ launch the app

Example

web_content_consumer_requests.py

```
import sys
import requests

BASE_URL = 'https://www.dictionaryapi.com/api/v3/references/collegiate/json/'  ①

API_KEY = 'b619b55d-faa3-442b-a119-dd906adc79c8' ②


def main(args):
    if len(args) < 1:
        print("Please specify a search term")
        sys.exit(1)

    response = requests.get(
        BASE_URL + args[0],
        params={'key': API_KEY},
        # ssl, proxy, cookies, headers, etc.
    )  ③

    if response.status_code == requests.codes.OK:  # 200?
        data = response.json()  ④
        for entry in data: ⑤
            if isinstance(entry, dict):
                meta = entry.get("meta")
                if meta:
                    part_of_speech = '({})'.format(entry.get('fl'))
                    word_id = meta.get("id")
                    print("{} {}".format(word_id.upper(), part_of_speech))
                if "shortdef" in entry:
                    print('\n'.join(entry['shortdef']))
                print()
            else:
                print(entry)

    else:
        print("Sorry, HTTP response", response.status_code)

if __name__ == '__main__':
    main(sys.argv[1:])
```

① base URL of resource site

② credentials

③ send HTTP request and get HTTP response

④ convert JSON content to Python data structure

⑤ check for results

web_content_consumer_requests.py wombat

```
WOMBAT (noun)
any of several stocky burrowing Australian marsupials (genera Vombatus and Lasiorhinus of
the family Vombatidae) resembling small bears
```

Table 14. Keyword Parameters for **requests** *methods*

Option	Data Type	Description
allow_redirects	bool	set to True if PUT/POST/DELETE redirect following is allowed
auth	tuple	authentication pair (user/token,password/key)
cert	str or tuple	path to cert file or ('cert', 'key') tuple
cookies	dict or CookieJar	cookies to send with request
data	dict	parameters for a POST or PUT request
files	dict	files for multipart upload
headers	dict	HTTP headers
json	str	JSON data to send in request body
params	dict	parameters for a GET request
proxies	dict	map protocol to proxy URL
stream	bool	if False, immediately download content
timeout	float or tuple	timeout in seconds or (connect timeout, read timeout) tuple
verify	bool	if True, then verify SSL cert

NOTE These can be used with any of the HTTP request types, as appropriate.

Table 15. **requests.Response** *attributes*

Attribute	Definition
apparent_encoding	Returns the apparent encoding
close()	Closes the connection to the server
content	Content of the response, in bytes
cookies	A CookieJar object with the cookies sent back from the server
elapsed	A timedelta object with the time elapsed from sending the request to the arrival of the response
encoding	The encoding used to decode r.text
headers	A dictionary of response headers
history	A list of response objects holding the history of request (url)
is_permanent_redirect	True if the response is the permanent redirected url, otherwise False
is_redirect	True if the response was redirected, otherwise False
iter_content()	Iterates over the response
iter_lines()	Iterates over the lines of the response
json()	A JSON object of the result (if the result was written in JSON format, if not it raises an error)
links	The header links
next	A PreparedRequest object for the next request in a redirection
ok	True if status_code is less than 400, otherwise False
raise_for_status()	If an error occur, this method a HTTPError object
reason	A text corresponding to the status code
request	The request object that requested this response
status_code	A number that indicates the status (200 is OK, 404 is Not Found)
text	The content of the response, in unicode
url	The URL of the response

Authentication with requests

- Options
  - Basic-Auth
  - Digest
  - Custom
- Use **auth** argument

requests makes it eaasy to provide basic authentication to a web site.

In the simplest case, create a `requests.auth.HTTPBasicAuth` object with the username and password, then pass that to requests with the `auth` argument. Since this is a common use case, you can also just pass a `(user, password)` tuple to the `auth` parameter.

For digest authentication, use `requests.auth.HTTPDigestAuth` with the username and password.

For custom authentication, you can create your own auth class by inheriting from `requests.auth.AuthBase`.

For OAuth 1, OAuth 2, and OpenID, install `requests-oauthlib`. This additional module provides auth objects that can be passed in with the `auth` parameter, as above.

See https://docs.python-requests.org/en/latest/user/authentication/ for more details.

Example

basic_auth_requests.py

```
import requests
from requests.auth import HTTPBasicAuth, HTTPDigestAuth

# base URL for httpbin
BASE_URL = 'https://httpbin.org'

# formats for httpbin
BASIC_AUTH_FMT = "/basic-auth/{}/{}"
DIGEST_AUTH_FMT = "/digest-auth/{}/{}/{}"

USERNAME = "spam"
PASSWORD = "ham"
BAD_PASSWORD = "toast"

REPORT_FMT = "{:35s} {}"

def main():
    basic_auth()
    digest()

def basic_auth():
    auth = HTTPBasicAuth(USERNAME, PASSWORD)
    response = requests.get(
        BASE_URL + BASIC_AUTH_FMT.format(USERNAME, PASSWORD),
        auth=auth,
    )
    print(REPORT_FMT.format("Basic auth good password", response))

    response = requests.get(
        BASE_URL + BASIC_AUTH_FMT.format(USERNAME, PASSWORD),
        auth=(USERNAME, PASSWORD),
    )
    print(REPORT_FMT.format("Basic auth good password (shortcut)", response))

    response = requests.get(
        BASE_URL + BASIC_AUTH_FMT.format(USERNAME, BAD_PASSWORD),
        auth=auth,
    )
    print(REPORT_FMT.format("Basic auth bad password", response))

def digest():
    auth = HTTPDigestAuth(USERNAME, PASSWORD)
    response = requests.get(
        BASE_URL + DIGEST_AUTH_FMT.format('WOMBAT', USERNAME, PASSWORD),
```

```
        auth=auth,
    )
    print(REPORT_FMT.format("Digest auth good password", response))

    auth = HTTPDigestAuth(USERNAME, BAD_PASSWORD)
    response = requests.get(
        BASE_URL + DIGEST_AUTH_FMT.format('WOMBAT', USERNAME, PASSWORD),
        auth=auth,
    )
    print(REPORT_FMT.format("Digest auth bad password", response))


if __name__ == '__main__':
    main()
```

basic_auth_requests.py

```
Basic auth good password            <Response [200]>
Basic auth good password (shortcut) <Response [200]>
Basic auth bad password             <Response [401]>
Digest auth good password           <Response [200]>
Digest auth bad password            <Response [401]>
```

Grabbing a web page the hard way

- import urlopen() from urllib.request
- urlopen() similar to open()
- Read response
- Use info() for metadata

While **requests** simplifies creating an HTTP client, the standard library module **urllib.request** includes **urlopen()**. It returns a file-like object. You can iterate over lines of HTML, or read all of the contents with read().

The URL is opened in binary mode ; you can download any kind of file which a URL represents – PDF, MP3, JPG, and so forth – by using read().

NOTE	When downloading HTML or other text, a bytes object is returned; use decode() to convert it to a string.

In general, the preferred approach is to install and use **requests**.

Example

read_html_urllib.py

```
#!/usr/bin/env python

import urllib.request

u = urllib.request.urlopen("https://www.python.org")

print(u.info())  ①
print()

print(u.read(500).decode())   ②
```

① .info() returns a dictionary of HTTP headers

② The text is returned as a bytes object, so it needs to be decoded to a string

read_html_urllib.py

```
Connection: close
Content-Length: 49816
Server: nginx
Content-Type: text/html; charset=utf-8
X-Frame-Options: DENY
Via: 1.1 vegur, 1.1 varnish, 1.1 varnish
Accept-Ranges: bytes
Date: Wed, 12 Jan 2022 13:02:30 GMT
Age: 2415
X-Served-By: cache-iad-kcgs7200147-IAD, cache-pdk17834-PDK
X-Cache: HIT, HIT
X-Cache-Hits: 1, 1
X-Timer: S1641992550.163781,VS0,VE1
Vary: Cookie
Strict-Transport-Security: max-age=63072000; includeSubDomains



<!doctype html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">   <![endif]-->
<!--[if IE 7]>      <html class="no-js ie7 lt-ie8 lt-ie9">          <![endif]-->
<!--[if IE 8]>      <html class="no-js ie8 lt-ie9">                 <![endif]-->
<!--[if gt IE 8]><!--><html class="no-js" lang="en" dir="ltr">  <!--<![endif]-->

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <link rel="prefetch" href="//ajax.googleapis.com/ajax/libs/jqu
```

Example

read_pdf_urllib.py

```
#!/usr/bin/env python

import sys
import os
from urllib.request import urlopen
from urllib.error import HTTPError

# url to download a PDF file of a NASA ISS brochure

url = 'https://www.nasa.gov/pdf/739318main_ISS%20Utilization%20Brochure
%202012%20Screenres%203-8-13.pdf'  ①

saved_pdf_file = 'nasa_iss.pdf'  ②

try:
    URL = urlopen(url)  ③
except HTTPError as e:  ④
    print("Unable to open URL:", e)
    sys.exit(1)

pdf_contents = URL.read()  ⑤
URL.close()

with open(saved_pdf_file, 'wb') as pdf_in:
    pdf_in.write(pdf_contents)  ⑥

if sys.platform == 'win32':  ⑦
    cmd = saved_pdf_file
elif sys.platform == 'darwin':
    cmd = 'open ' + saved_pdf_file
else:
    cmd = 'acroread ' + saved_pdf_file

os.system(cmd)  ⑧
```

① target URL

② name of PDF file for saving

③ open the URL

④ catch any HTTP errors

⑤ read all data from URL in binary mode

⑥ write data to a local file in binary mode

⑦ select platform and choose the app to open the PDF file

⑧ launch the app

Consuming Web services the hard way

- Use urllib.parse to URL encode the query.
- Use urllib.request.Request
- Specify data type in header
- Open URL with urlopen Read data and parse as needed

To consume Web services, use the urllib.request module from the standard library. Create a urllib.request.Request object, and specify the desired data type for the service to return.

If needed, add a headers parameter to the request. Its value should be a dictionary of HTTP header names and values.

For URL encoding the query, use urllib.parse.urlencode(). It takes either a dictionary or an iterable of key/value pairs, and returns a single string in the format "K1=V1&K2=V2&..." suitable for appending to a URL.

Pass the Request object to urlopen(), and it will return a file-like object which you can read by calling its read() method.

The data will be a bytes object, so to use it as a string, call decode() on the data. It can then be parsed as appropriate, depending on the content type.

NOTE	the example program on the next page queries the Merriam-Webster dictionary API. It requires a word on the command line, which will be looked up in the online dictionary.
TIP	List of public RESTful APIs: http://www.programmableweb.com/apis/directory/1?protocol=REST

Example

web_content_consumer_urllib.py

```
#!/usr/bin/env python
"""
Fetch a word definition from Merriam-Webster's API
"""
import sys
from urllib.request import Request, urlopen
import json
# from pprint import pprint

DATA_TYPE = 'application/json'

API_KEY = 'b619b55d-faa3-442b-a119-dd906adc79c8'

URL_TEMPLATE =
'https://www.dictionaryapi.com/api/v3/references/collegiate/json/{}?key={}'  ①

def main(args):
    if len(args) < 1:
        print("Please specify a word to look up")
        sys.exit(1)

    search_term = args[0].replace(' ', '+')

    url = URL_TEMPLATE.format(search_term, API_KEY)  ②

    do_query(url)

def do_query(url):
    print("URL:", url)
    request = Request(url)
    response = urlopen(request)  ③
    raw_json_string = response.read().decode()  ④
    data = json.loads(raw_json_string)  ⑤
    # print("RAW DATA:")
    # pprint(data)
    for entry in data:  ⑥
        if isinstance(entry, dict):
            meta = entry.get("meta") ⑦
            if meta:
                part_of_speech = '({})'.format(entry.get('fl'))
                word_id = meta.get("id")
                print("{} {}".format(word_id.upper(), part_of_speech))
            if "shortdef" in entry:
                print('\n'.join(entry['shortdef']))
```

```
            print()
        else:
            print(entry)
if __name__ == '__main__':
    main(sys.argv[1:])
```

① base URL of resource site

② build search URL

③ send HTTP request and get HTTP response

④ read content from web site and decode() from bytes to str

⑤ convert JSON string to Python data structure

⑥ iterate over each entry in results

⑦ retrieve items from results (JSON convert to lists and dicts)

web_content_consumer_urllib.py dewars

```
URL: https://www.dictionaryapi.com/api/v3/references/collegiate/json/wombat?key=b619b55d-faa3-442b-a119-dd906adc79c8
WOMBAT (noun)
any of several stocky burrowing Australian marsupials (genera Vombatus and Lasiorhinus of the family Vombatidae) resembling small bears
```

sending e-mail

- import smtplib module
- Create an SMTP object specifying server
- Call sendmail() method from SMTP object

You can send e-mail messages from Python using the smtplib module. All you really need is one smtplib object, and one method – sendmail().

Create the smtplib object, then call the sendmail() method with the sender, recipient(s), and the message body (including any headers).

The recipients list should be a list or tuple, or could be a plain string containing a single recipient.

Example

email_simple.py

```
#!/usr/bin/env python
from getpass import getpass  ①
import smtplib  ②
from email.message import EmailMessage  ③
from datetime import datetime

TIMESTAMP = datetime.now().ctime()  ④

SENDER = 'jstrick@mindspring.com'
RECIPIENTS = ['jstrickler@gmail.com']
MESSAGE_SUBJECT = 'Python SMTP example'

MESSAGE_BODY = """
Hello at {}.

Testing email from Python
""".format(TIMESTAMP)

SMTP_USER = 'pythonclass'
SMTP_PASSWORD = getpass("Enter SMTP server password:")  ⑤

smtpserver = smtplib.SMTP("smtp2go.com", 2525)  ⑥
smtpserver.login(SMTP_USER, SMTP_PASSWORD)  ⑦

msg = EmailMessage()  ⑧
msg.set_content(MESSAGE_BODY)  ⑨
msg['Subject'] = MESSAGE_SUBJECT  ⑩
msg['from'] = SENDER  ⑪
msg['to'] = RECIPIENTS  ⑫

try:
    smtpserver.send_message(msg)  ⑬
except smtplib.SMTPException as err:
    print("Unable to send mail:", err)
finally:
    smtpserver.quit()  ⑭
```

① module for hiding password

② module for sending email

③ module for creating message

④ get a time string for the current date/time

⑤ get password (not echoed to screen)

⑥ connect to SMTP server

⑦ log into SMTP server

⑧ create empty email message

⑨ add the message body

⑩ add the message subject

⑪ add the sender address

⑫ add a list of recipients

⑬ send the message

⑭ disconnect from SMTP server

Email attachments

- Create MIME multipart message
- Create MIME objects
- Attach MIME objects
- Serialize message and send

To send attachments, you need to create a MIME multipart message, then create MIME objects for each of the attachments, and attach them to the main message. This is done with various classes provided by the **email.mime** module.

These modules include **multipart** for the main message, **text** for text attachments, **image** for image attachments, **audio** for audio files, and **application** for miscellaneous binary data.

One the attachments are created and attached, the message must be serialized with the **as_string()** method. The actual transport uses **smptlib**, just like simple email messages described earlier.

Example

email_attach.py

```
#!/usr/bin/env python
import smtplib
from datetime import datetime
from imghdr import what  ①
from email.message import EmailMessage ②
from getpass import getpass ③


SMTP_SERVER = "smtp2go.com"  ④
SMTP_PORT = 2525

SMTP_USER = 'pythonclass'

SENDER = 'jstrick@mindspring.com'
RECIPIENTS = ['jstrickler@gmail.com']


def main():
    smtp_server = create_smtp_server()
    now = datetime.now()
    msg = create_message(
        SENDER,
        RECIPIENTS,
        'Here is your attachment',
        'Testing email attachments from python class at {}\n\n'.format(now),
    )
    add_text_attachment('../DATA/parrot.txt', msg)
    add_image_attachment('../DATA/felix_auto.jpeg', msg)
    send_message(smtp_server, msg)


def create_message(sender, recipients, subject, body):
    msg = EmailMessage()  ⑤
    msg.set_content(body)  ⑥
    msg['From'] = sender
    msg['To'] = recipients
    msg['Subject'] = subject
    return msg


def add_text_attachment(file_name, message):
    with open(file_name) as file_in:  ⑦
        attachment_data = file_in.read()
    message.add_attachment(attachment_data)  ⑧
```

```
def add_image_attachment(file_name, message):
    with open(file_name, 'rb') as file_in:  ⑨
        attachment_data = file_in.read()
    image_type = what(None, h=attachment_data)  ⑩
    message.add_attachment(attachment_data, maintype='image', subtype=image_type)  ⑪


def create_smtp_server():
    password = getpass("Enter SMTP server password:")  ⑫
    smtpserver = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)  ⑬
    smtpserver.login(SMTP_USER, password)  ⑭

    return smtpserver


def send_message(server, message):
    try:
        server.send_message(message)  ⑮
    finally:
        server.quit()


if __name__ == '__main__':
    main()
```

① module to determine image type

② module for creating email message

③ module for reading password privately

④ global variables for external information (IRL should be from environment — command line, config file, etc.)

⑤ create instance of EmailMessage to hold message

⑥ set content (message text) and various headers

⑦ read data for text attachment

⑧ add text attachment to message

⑨ read data for binary attachment

⑩ get type of binary data

⑪ add binary attachment to message, including type and subtype (e.g., "image/jpg)"

⑫ get password from user (don’t hardcode sensitive data in script)

⑬ create SMTP server connection

⑭ log into SMTP connection

⑮ send message

Remote Access

- Use paramiko (not part of standard library)
- Create ssh client
- Create transport object to use sftp and other tools

For remote access to other computers, you generally use the SSH protocol. Python has several ways to use SSH.

The current best way is to use paramiko. It is a pure-Python module for connecting to other computers using SSH. It is not part of the standard library, but is included with the Anaconda distribution.

NOTE

Paramiko is used by Ansible and other sys admin tools.

Find out more about paramiko at http://www.lag.net/paramiko/
Find out more about Ansible at http://www.ansible.com/
Find out more about **ssh2-python**, an alternative to Paramiko, at https://parallel-ssh.org/post/ssh2-python/

Auto-adding hosts

- Interactive SSH prompts to add new host
- Programmatic interface can't do that
- Use **set_missing_host_key_policy()**
- Adds to list of known hosts.

The first time you connect to a new host with SSH, you get the following message:

```
The authenticity of host HOSTNAME can't be established.
ECDSA key fingerprint is HOSTNAME
Are you sure you want to continue connecting...
```

To avoid the message when using Paramiko, call **set_missing_host_key_policy()** from the Paramiko SSH client object:

```
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

Remote commands

- Use SSHClient
- Access standard I/O channels

To run commands on a remote computer, use SSHClient. Once you connect to the remote host, you can execute commands and access the standard I/O of the remote program.

The **exec_command()** method executes a command on the remote host, and returns a 3-tuple with the remote command's stdin, stdout, and stderr as file-like objects.

You can read from stdout and stderr, and write to stdin.

NOTE	With some versions of **paramiko**, the *stdin* object returned by **exec_command()** must be explicitly set to **None**, or deleted with **DEL** after use. Otherwise, an error will be raised.

Example

paramiko_commands.py

```
#!/usr/bin/env python

import paramiko

with paramiko.SSHClient() as ssh:  ①

    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  ②

    ssh.connect('localhost', username='python', password='l0lz')  ③

    stdin, stdout, stderr = ssh.exec_command('whoami')  ④
    print(stdout.read().decode())  ⑤
    print('-' * 60)

    stdin, stdout, stderr = ssh.exec_command('ls -l')  ④
    print(stdout.read().decode())  ⑤
    print('-' * 60)

    stdin, stdout, stderr = ssh.exec_command('ls -l /etc/passwd /etc/horcrux')  ④
    print("STDOUT:")
    print(stdout.read().decode())  ⑤
    print("STDERR:")
    print(stderr.read().decode())  ⑥
    print('-' * 60)

    del stdin  # workaround for paramiko bug!
```

① create paramiko client

② ignore missing keys (this is safe)

③ connect to remote host

④ execute remote command; returns standard I/O objects

⑤ read stdout of command

⑥ read stderr of command

paramiko_commands.py

```
python

------------------------------------------------------------
total 384
drwx------+  3 python  staff      96 Feb 11  2021 Desktop
drwx------+  3 python  staff      96 Feb 11  2021 Documents
drwx------+  3 python  staff      96 Feb 11  2021 Downloads
drwx------@ 50 python  staff    1600 Sep 14 07:12 Library
drwx------+  3 python  staff      96 Feb 11  2021 Movies
drwx------+  3 python  staff      96 Feb 11  2021 Music
drwx------+  3 python  staff      96 Feb 11  2021 Pictures
drwxr-xr-x+  4 python  staff     128 Feb 11  2021 Public
-rw-r--r--   1 python  staff  148544 May 27  2021 alice.txt
drwxr-xr-x   2 python  staff      64 May 27  2021 foo
drwxr-xr-x   2 python  staff      64 May 27  2021 testing
drwxr-xr-x   3 python  staff      96 Feb 18  2021 text_files

------------------------------------------------------------
STDOUT:
-rw-r--r--  1 root  wheel  6946 Jun  5  2020 /etc/passwd

STDERR:
ls: /etc/horcrux: No such file or directory

------------------------------------------------------------
```

Copying files with SFTP

- Create transport
- Create SFTP client with transport

To copy files with paramiko, first create a **Transport** object. Using a **with** block will automatically close the Transport object.

From the transport object you can create an SFTPClient. Once you have this, call standard FTP/SFTP methods on that object.

Some common methods include listdir_iter(), get(), put(), mkdir(), and rmdir().

Example

paramiko_copy_files.py

```
#!/usr/bin/env python
import os
import paramiko

REMOTE_DIR = 'text_files'

with paramiko.Transport(('localhost', 22)) as transport:  ①
    transport.connect(username='python', password='l0lz')  ②
    sftp = paramiko.SFTPClient.from_transport(transport)  ③
    for item in sftp.listdir_iter():  ④
        print(item)
    print('-' * 60)

    remote_file = os.path.join(REMOTE_DIR, 'betsy.txt')  ⑤
    sftp.mkdir("testing")

    # sftp.put(local-file)
    # sftp.put(local-file, remote-file)
    sftp.put('../DATA/alice.txt', 'text_files/betsy.txt')  ⑥
    sftp.put('../DATA/alice.txt', 'alice.txt')
    sftp.put('../DATA/alice.txt', 'text_files')
    sftp.get(remote_file, 'eileen.txt')  ⑦
```

① create paramiko Transport instance

② connect to remote host

③ create SFTP client using Transport instance

④ get list of items on default (login) folder (listdir_iter() returns a generator)

⑤ create path for remote file

⑥ create a folder on the remote host

⑦ copy a file to the remote host

⑧ copy a file from the remote host

⑨ use SSHClient to confirm operations (not needed, just for illustration)

paramiko_copy_files.py

```
drwx------   1 503      20             96 11 Feb 2021  Music
-r--------   1 503      20              7 14 Sep 07:09 .CFUserTextEncoding
drwx------   1 503      20             96 11 Feb 2021  Pictures
drwxr-xr-x   1 503      20             96 18 Feb 2021  text_files
-rw-r--r--   1 503      20         148544 27 May 2021  alice.txt
-rw-------   1 503      20            135 14 Sep 07:13 .zsh_history
drwx------   1 503      20             96 11 Feb 2021  Desktop
drwx------   1 503      20           1600 14 Sep 07:12 Library
drwxr-xr-x   1 503      20             64 27 May 2021  testing
drwxr-xr-x   1 503      20            128 11 Feb 2021  Public
drwxr-xr-x   1 503      20             64 27 May 2021  foo
drwx------   1 503      20             96 11 Feb 2021  Movies
drwx------   1 503      20             96 11 Feb 2021  Documents
drwx------   1 503      20             96 11 Feb 2021  Downloads
----------------------------------------------------------------
```

Interactive remote access

- Write to stdin
- Read response from stdout

To interact with a remote program, write to the stdin object returned by ***ssh_object*.exec_command()**.

```
stdin.write("command input....\n")
```

Be sure to add a newline ('\n') for each line of input you send.

To get the response, read the next line(s) of code with *stdout*.readline()

Example

paramiko_interactive.py

```
#!/usr/bin/env python
import paramiko
# bc is an interactive calculator that comes with Unix-like systems (Linux, Mac, etc.)

with paramiko.SSHClient() as ssh:  ①
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  ②

    ssh.connect('localhost', username='python', password='l0lz')  ③

    stdin, stdout, stderr = ssh.exec_command('bc')  ④

    stdin.write("17 + 25\n")  ⑤
    result = stdout.readline()  ⑥
    print("Result is:", result)

    stdin.write("scale = 3\n")  ⑦
    stdin.write("738.3/191.9\n")
    result = stdout.readline()
    print("Result is:", result)

    stdin.write("quit\n")  ⑧
    stdin = None    ⑨
```

① create paramiko SSH client

② auto-add remote host

③ log into to remote host

④ execute command; returns file-like objects representing stdio

⑤ write to command's stdin

⑥ read output of command

⑦ set scale (# decimal points) to 3 (bc-specific command)

paramiko_interactive.py

```
Result is: 42

Result is: 3.847
```

Chapter 5 Exercises

Exercise 5-1 (fetch_xkcd_requests.py, fetch_xkcd_urllib.py)

Write a script to fetch the following image from the Internet and display it. http://imgs.xkcd.com/comics/python.png

Exercise 5-2 (wiki_links_requests.py, wiki_links_urllib.py)

Write a script to count how many links are on the home page of Wikipedia. To do this, read the page into memory, then look for occurrences of the string "href". (For *real* screen-scraping, you can use the Beautiful Soup module.)

You can use the string method **find()**, which can be called like S.find('text', start, stop), which finds on a slice of the string, moving forward each time the string is found.

Exercise 5-3 (send_chimp.py)

If the class conditions allow it (i.e., if you have access to the Internet, and an SMTP account), send an email to yourself with the image **chimp.bmp** (from the DATA folder) attached.

Chapter 6: Binary Data

Objectives

- Know the difference between text and binary data
- Open files in text or binary mode
- Use Struct to process binary data streams

"Binary" (raw, or non-delimited) data

- Open file in binary mode
- Use read()
- Specify number of bytes to read
- Read entire file when no size given
- Returns **bytes** object

A file can be opened in binary mode. This allows for "raw", or non-delimited reads, which do not treat newlines and carriage returns as special.

In binary mode, read() will return a **bytes** object (array of 8-bit integers), not a Python string (array of Unicode characters). Use **.decode()** to convert the bytes object to a string.

Use **write()** to write raw data to a file.

Use **seek()** to position the next read, and **tell()** to determine the current location within the file.

Binary vs Text data

- Networks use binary data
- Convert to standard if mixing platforms
- Need to know layout of data

When you read data from a network application, such as getting the HTML source of a web page, it is retrieved as binary data, even thought it is "text". It is typically encoded as ASCII or UTF-8. This is represented by a **bytes** object, which is an array of bytes.

To convert a bytes object to a string, call the **decode()** method. When going the other direction, as in writing some text out to a network application, you will need to convert from a Python string, which is an in-memory representation, to a string of bytes. Do do this, call the string's **encode()** method.

Using Struct

- Struct class from struct module
- Translates between Python to native/standard formats
- Format string describes data layout

If you need to process a **raw** binary file

The **struct** module provides the Struct class. You can instantiate a Struct with a format string representing the binary data layout. From the instance, you can call **unpack()** to decode a binary stream, or **pack()** to encode it.

The **size** property is the number of bytes needed for the data.

The format string describes the data layout using format codes. Each code is a letter representing a data type, and can be preceded with a size or repeat count (depending on data type), and a prefix which specifies the byte order and alignment.

"Native" byte order or alignment refers to the same byte order or alignment used by the C compiler on the current platform. "Standard" refers to a standard set of sizes for typical numerical objects, such as shorts, ints, longs, floats and doubles. The default is native.

Table 16. Struct format codes

Format	C Type	Python Type	Standard size	Notes
x	pad byte	no value	n/a	
c	char	bytes of length 1	1	
b	signed char	integer	1	(1),(3)
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2),(3)
Q	unsigned long long	integer	8	(2), (3)
n	ssize_t	integer		(4)
N	size_t	integer		(4)
f	float	float	4	(5)
d	double	float	8	(5)
s	char[]	bytes		
p	char[]	bytes		
P	void *	integer		(6)

Notes

1. The '?' conversion code corresponds to the _Bool type defined by C99. If this type is not available, it is simulated using a char. In standard mode, it is always represented by one byte.
2. The 'q' and 'Q' conversion codes are available in native mode only if the platform C compiler supports C long long, or, on Windows, __int64. They are always available in standard modes.
3. When attempting to pack a non-integer using any of the integer conversion codes, if the non-integer has a *index*() method then that method is called to convert the argument to an integer before packing.
4. The 'n' and 'N' conversion codes are only available for the native size (selected as the default or with the '@' byte order character). For the standard size, you can use whichever of the other integer

formats fits your application.

5. For the 'f' and 'd' conversion codes, the packed representation uses the IEEE 754 binary32 (for 'f') or binary64 (for 'd') format, regardless of the floating-point format used by the platform.
6. The 'P' format character is only available for the native byte ordering (selected as the default or with the '@' byte order character). The byte order character '=' chooses to use little- or big-endian ordering based on the host system. The struct module does not interpret this as native ordering, so the 'P' format is not available.

Table 17. Struct byte order/size/alignment flags

Flag	Byte order	Size and byte alignment
@ default	Native	Native
=	Native	Standard
<	Little-endian	Standard
> or !	Big-endian	Standard

Example

binary_data.py

```
#!/usr/bin/env python

from struct import Struct

values = 7, 6, 42.3, b'Guido' ①

demo = Struct('iif10s')  ②

print("Size of data: {} bytes".format(demo.size)) ③

binary_stream = demo.pack(*values) ④

int1, int2, float1, raw_bytes = demo.unpack(binary_stream) ⑤
str1 = raw_bytes.decode().rstrip('\x00')  ⑥

print(raw_bytes)
print(int1, int2, float1, str1)
```

① create some assoted values

② create Struct object with desired data layout

③ size property gives size of data in bytes

④ pack() converts values into binary stream using format

⑤ unpack() converts binary stream into list of values

⑥ decode the raw bytes into a string, and strip off trailing null bytes (that were added by pack())

binary_data.py

```
Size of data: 22 bytes
b'Guido\x00\x00\x00\x00\x00'
7 6 42.29999923706055 Guido
```

Example

parse_bmp.py

```
#!/usr/bin/env python

from struct import Struct

# short int short short int (native, unsigned)
s = Struct('=ccIHHI')  ①

with open('../DATA/chimp.bmp', 'rb') as chimp_in:
    chimp_bmp = chimp_in.read(s.size)  ②

(sig1, sig2, size, reserved1, reserved2, offset) = s.unpack(chimp_bmp)  ③

print("signature:", (sig1 + sig2).decode())  ④
print('size:', size)
print('reserved1:', reserved1)
print('reserved2:', reserved2)
print('offset:', offset)
```

① define layout of bitmap header

② read the first 14 bytes of bitmap file in binary mode

③ unpack the binary header into individual values

④ output the individual values

parse_bmp.py

```
signature: BM
size: 5498
reserved1: 0
reserved2: 0
offset: 1074
```

Example

read_binary.py

```
#!/usr/bin/env python

# print out a file 10 bytes at a time

with open("../DATA/parrot.txt", "rb") as parrot_in: ①
    while True:
        chunk = parrot_in.read(10)  ②
        if chunk == b"":   ③
            break
        print(chunk.decode()) ④
```

① Add "b" to "r", "w", or "a" for binary mode

② Use read() to read a specified number of bytes

③ Read returns **bytes**, not **str**

④ Use decode() to convert bytes to str

read_binary.py | head -10

```
So there's
 this fell
a with a p
arrot. And
 this parr
ot swears

like a sai
lor, I mea
n he's a p
```

Bitwise operations

- Operators
  - & and
  - | or
  - ~ complement
  - ^ xor (exclusive or)
  - << left shift
  - >> right shift
- Integers only
- bin() displays in binary

Python has bitwise operations to compare individual bits in an integer. This is sometimes used for flags, or for packing more information into a byte. (Instead of using a 32-bit integer to store that something is true, you can use one bit.

The and and or operators work on two integers of the same size, and return a new integer with the bits modified according to the operator. The and operator can be used to clear bits. It is typically used with a *mask* composed of ones for all the bits you don't want to clear.

When comparing two numbers, the and operator, **&**, sets a result bit to 1 (one) if the corresponding bits in both numbers are both set to one. Otherwise, it sets the result bit to 0 (zero).

The or operator, |, sets a result bit to 1 (one) if *either* of the corresponding bits in both numbers are set to one. Otherwise, it sets the result bit to 0 (zero).

The complement operator, ~ reverses the values of bits — i.e., it changes ones to zeros and zeros to ones.

The xor operator, ^, sets a result bit to 1 (one) if the corresponding bits have *different* values.. Otherwise, it sets the result bit to 0 (zero).

The left shift operator, <<, moves bits to the left, a specified number of places.

The right shift operator, >>, works like left shift, but moves bits to the right.

Table 18. Struct bitwise operators

operator	**meaning**	**a**	**b**	**a** *op* **b**
&	and	1	1	1
		1	0	0
		0	1	0
		0	0	0
|	or	1	1	1
		1	0	1
		0	1	1
		0	0	0
^	xor (exclusive or)	1	1	0
		1	0	1
		0	1	1
		0	0	0
~	complement			
<<	left shift			
>>	right shift			

Example

bitwise_ops.py

```
#!/usr/bin/env python

a = 0b10101010  ①
b = 0b11110000

c = a & b  ②
print("  {:08b}".format(a))
print("& {:08b}".format(b))
print("  --------")
print("  {:08b}".format(c))
print()

c = a | b ③
print("  {:08b}".format(a))
print("| {:08b}".format(b))
print("  --------")
print("  {:08b}".format(c))
print()

c = a ^ b ④
print("  {:08b}".format(a))
print("^ {:08b}".format(b))
print("  --------")
print("  {:08b}".format(c))
print()

c = ~a ⑤
print(" ~ {:09b}".format(a))
print("   {:09b}".format(c))
print()

c = a >> 1 ⑥
print("{:08b} >> 1".format(a))
print("{:08b}".format(c))
print()

c = a >> 3 ⑦
print("{:08b} >> 3".format(a))
print("{:08b}".format(c))
print()

c = a << 1 ⑧
print("{:012b} << 1".format(a))
print("{:012b}".format(c))
```

```
print()

c = a << 3 ⑨
print("{:012b} << 3".format(a))
print("{:012b}".format(c))
print()
```

① a and b are integers

② bitwise AND

③ bitwise OR

④ bitwise XOR

⑤ complement (flip bit values)

⑥ shift right 1 bit

⑦ shift right 3 bits

⑧ shift left 1 bit

⑨ shift left 3 bits

bitwise_ops.py

```
  10101010
& 11110000
  --------
  10100000

  10101010
| 11110000
  --------
  11111010

  10101010
^ 11110000
  --------
  01011010

 ~ 010101010
   -10101011

10101010 >> 1
01010101

10101010 >> 3
00010101

000010101010 << 1
000101010100

000010101010 << 3
010101010000
```

Chapter 6 Exercises

Exercise 6-1 (demystify.py)

Write a program that prints out every third byte (starting with the first byte) of the file named **mystery**. The output will be an ASCII art picture.

TIP	read the file into a bytes object (be sure to use binary mode), then use slice notation to select the bytes, then decode into a string for printing.

Exercise 6-2 (pypuzzle.py)

The file **puzzle.data** has a well-known name encoded in it. The ASCII values of the characters in the name are represented by a series of integers and floats of various sizes.

The layout is: float, int, float, int, float, short, unsigned short, float, unsigned int, double, float, double, unsigned int, int, unsigned int, short

To decode, read the file into a bytes object and use a Struct object to decode the raw data into the values. Then convert the values into integers, and use chr() to convert the integers into ASCII characters. Finally, you can join the characters together and print them out.

Chapter 7: Using openpyxl with Excel spreadsheets

Objectives

- Learn the basics of **openpyxl**
- Open **Excel** spreadsheets and extract data
- Update spreadsheets
- Create new spreadsheets
- Add styles and conditional formatting

The openpyxl module

- Provides full read/write access to Excel spreadsheets
- Creates new workbooks/worksheets
- Does not require Excel

The **openpyxl** module allows you to read, write, and create **Excel** spreadsheets.

openpyxl does not require Excel to be installed.

You can open existing workbooks or create new ones. You can do most anything that you could do manually in a spreadsheet – update or insert data, create formulas, add or change styles, even hide columns.

When you open an existing spreadsheet or create a new one, openpyxl creates an instance of `Workbook`. A Workbook contains one or more `Worksheet` objects.

From a worksheet you can access cells directly, or create a range of cells.

The data in each cell can be manipulated through its `.value` property. Other properties, such as `.font` and `.number_format`, control the display of the data.

View the full documentation at http://openpyxl.readthedocs.org/en/latest/index.html.

TIP	To save typing, import **openpyxl** as **px**.

Reading an existing spreadsheet

- Use `load_workbook()` to open file
- Get active worksheet with `WB.active`
- List all worksheets with `get_sheet_by_name()`
- Get named worksheet with `WB['worksheet-name']`

To open and read an existing spreadsheet, use the `load_workbook()` function. This returns a WorkBook object.

There are several ways to get a worksheet from a workbook. To list all the sheets by name, use `WB.get_sheet_names()`.

To get a particular worksheet, index the workbook by sheet name, e.g. `WB['sheetname']`.

A workbook is also an iterable of all the worksheets it contains, so to work on all the worksheets one at a time, you can loop over the workbook.

```
for ws in WB:
    print(ws.title)
```

The `.active` property of a workbook is the currently active worksheet. `WB.active` may be used as soon as a workbook is open.

The `.title` property of a worksheet lets you get or set the title (name) of the worksheet.

Example

px_load_worksheet.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    wb = px.load_workbook('../DATA/presidents.xlsx')

# three ways to get to a worksheet:

    # 1
    print(wb.sheetnames, '\n')
    ws = wb['US Presidents']
    print(ws, '\n')

    # 2
    for ws in wb:
        print(ws.title, ws.dimensions)
    print()

    # 3
    ws = wb.active
    print(ws, '\n')

    print(ws['B2'].value)


if __name__ == '__main__':
    main()
```

px_load_worksheet.py

```
['US Presidents', 'President Names']

<Worksheet "US Presidents">

US Presidents A1:J47
President Names B2:C47

<Worksheet "President Names">

Washington
```

Worksheet info

- Worksheet attributes
  - dimensions
  - min_row
  - max_row
  - min_column
  - max_column
  - *many others…*

Once a worksheet is opened, you can get information about the worksheet directly from the worksheet object.

The dimensions are based on the extent of the cells that actually contain data.

NOTE	Worksheets can have a maximum of 1,048,576 rows and 16,384 columns.

Example

px_worksheet_info.py

```
#!/usr/bin/env python
import openpyxl as px


def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    print("Title:", ws.title)
    print("Dimensions:", ws.dimensions)
    print("Minimum column:", ws.min_column)
    print("Minimum row:", ws.min_row)
    print("Maximum column:", ws.max_column)
    print("Maximum row:", ws.max_row)
    print("Parent:", ws.parent)
    print("Active cell:", ws.active_cell)

if __name__ == '__main__':
    main()
```

px_worksheet_info.py

```
Title: US Presidents
Dimensions: A1:J47
Minimum column: 1
Minimum row: 1
Maximum column: 10
Maximum row: 47
Parent: <openpyxl.workbook.workbook.Workbook object at 0x7ff4b88e5f10>
Active cell: J48
```

Table 19. Useful Worksheet Attributes

Attribute	Data type	Description
`active_cell`	str	coordinates ("A1"-style) of active cell
`columns`	generator	iterable of all columns, as tuples of Cell objects
`dimensions`	str	coordinate range ("A1:B2") of all cells containing data
`encoding`	str	text encoding of worksheet
`max_column`	int	maximum column index (1-based)
`max_row`	int	maximum row index (1-based)
`mime_type`	str	MIME type of document
`min_column`	int	minimum column index (1-based)
`min_row`	int	minimum row index (1-based)
`parent`	Workbook	Workbook object that this worksheet belongs to
`rows`	generator	iterable of all rows, as tuples of Cell objects
`selected_cell`	str	coordinates of currently selected cell
`tables`	dict	dictionary of tables
`title`	str	title of this worksheet
`values`	generator	iterable of all values in the worksheet (actual values, not Cell objects)

NOTE min and max row/column refer to extent of cells containing data

Accessing cells

- Each cell is instance of Cell
- Attributes
  - value
  - number_format
  - font
  - *and others*
- Get cell with
  - `ws["COORDINATES"]`
  - `ws.cell(row, column)`

A worksheet consists of *cells*. There are two ways to access an individual cell:

- lookup the cell using the cell coordinates, e.g. `ws["B3"]`.
- specify the row and column as integers (1-based) using the `.cell` method of the worksheet, e.g. `ws.cell(4, 5)`. You can used named arguments for the row and column: `ws.cell(row=4, column=5)`.

In both cases, to get the actual value, use the `.value` attribute of the cell.

NOTE	Cell coordinates are case-insensitive.

Example

px_access_cells.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    # access cell by cell name
    print(ws['A1'].value)
    print(ws['C2'].value, ws['B2'].value)
    print()

    # same, but lower-case
    print(ws['a1'].value)
    print(ws['c2'].value, ws['b2'].value)
    print()

    # access cell by row/column (1-based)
    print(
        ws.cell(row=27, column=3).value,  # "C27"
        ws.cell(row=27, column=2).value,  # "B27"
    )
    print()

    # same, without argument names
    print(
        ws.cell(27, 3).value,  # "C27"
        ws.cell(27, 2).value,  # "B27"
    )
    print()

if __name__ == '__main__':
    main()
```

px_access_cells.py

```
Term
George Washington

Term
George Washington

Theodore Roosevelt

Theodore Roosevelt
```

Table 20. Cell attributes

Attribute	Type	Description
alignment	openpyxl.styles.alignment.Alignment	display alignment info
border	openpyxl.styles.borders.Border	border info
col_idx	int	column index as integer
column	int	column index as integer
column_letter	str	column index as string
comment	any	cell comment
coordinate	str	coordinate of cell (e.g. ("B2")
data_type	str	data type code (s=str, n=numeric, etc)
encoding	str	text encoding (e.g. 'utf-8')
fill	openpyxl.styles.fills.PatternFill	fill (background) style
font	openpyxl.styles.fonts.Font	font color, family, style
has_style	bool	True if cell has style assigned
hyperlink	openpyxl.worksheet.hyperlink.Hyperlink	hyperlink for cell
internal_value	any	value of cell
is_date	bool	True if value is a date
number_format	str	Code for number format, e.g. "0.0"
parent	openpyxl.worksheet.worksheet.Worksheet	worksheet in which cell is located
protection	openpyxl.styles.protection.Protection	protection settings (e.g., hidden, locked)
quotePrefix	bool	character used for quoting
row	int	row index
style	str	name of style
value	any	actual value of cell

Getting raw values

- Use `worksheet.values`
- Returns row generator
- Each row is a tuple of column values

To iterate over all the values in the spreadsheet, use `worksheet.values`. It is a generator of the rows in the worksheet. Each element is a tuple of column values.

Only populated cells will be part of the returned data.

Example

px_raw_values.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']  ①
    headers = next(ws.values)   ②
    for row in ws.values:  ③
        print(row[:5])   ④


if __name__ == '__main__':
    main()
```

① get active sheet

② read first row from generator

③ loop over rows in generator

④ print first 5 elements of row tuple

px_raw_values.py

```
('Term', 'Last Name', 'First Name', 'Birth Date', 'Death Date')
(1, 'Washington', 'George', '1732-02-22', '1799-12-14')
(2, 'Adams', 'John', '1735-10-30', '1826-07-04')
(3, 'Jefferson', 'Thomas', '1743-04-13', '1826-07-04')
(4, 'Madison', 'James', '1751-03-16', '1836-06-28')
(5, 'Monroe', 'James', '1758-04-28', '1831-07-04')
(6, 'Adams', 'John Quincy', '1767-07-11', '1848-02-23')
(7, 'Jackson', 'Andrew', '1767-03-15', '1845-06-08')
(8, 'Van Buren', 'Martin', '1782-12-05', '1862-07-24')
(9, 'Harrison', 'William Henry', '1773-02-09', '1841-04-04')
```

```
...
```

```
(37, 'Nixon', 'Richard Milhous', '1913-01-09', '1994-04-22')
(38, 'Ford', 'Gerald Rudolph', '1913-07-14', '2006-12-26')
(39, 'Carter', "James Earl 'Jimmy'", '1924-10-01', 'NONE')
(40, 'Reagan', 'Ronald Wilson', '1911-02-06', '2004-06-05')
(41, 'Bush', 'George Herbert Walker', '1924-06-12', datetime.datetime(2018, 11, 30, 0,
0))
(42, 'Clinton', "William Jefferson 'Bill'", '1946-08-19', 'NONE')
(43, 'Bush', 'George Walker', '1946-07-06', 'NONE')
(44, 'Obama', 'Barack Hussein', '1961-08-04', 'NONE')
(45, 'Trump', 'Donald J', '1946-06-14', 'NONE')
(46, 'Biden', 'Joseph Robinette', datetime.datetime(1942, 11, 10, 0, 0), 'NONE')
```

Working with ranges

- Range represents a rectangle of cells
- Use slice notation
- Iterate through rows, then columns

To get a range of cells, use slice notation on the worksheet object and standard cell notation, e.g. `WS['A1':'M9']` or `WS['A1:M9']`. Note that the range can consist of one string containing the range, or two strings separated by `:`.

The range is a virtual list of rows, and so can be iterated over. Each element of a row is a Cell object. Use the .value attribute to get or set the cell value.

Example

px_get_ranges.py

```
#!/usr/bin/env python
import openpyxl as px


def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    print_first_and_last_names(ws)


def print_first_and_last_names(ws):
    """Print first and last names of all presidents"""
    pres_range = ws['B2':'C47']  # cell range
    for row in pres_range:  # row object
        print(row[1].value, row[0].value)


if __name__ == '__main__':
    main()
```

px_get_ranges.py

```
George Washington
John Adams
Thomas Jefferson
James Madison
James Monroe
John Quincy Adams
Andrew Jackson
Martin Van Buren
William Henry Harrison
John Tyler
```

...

Modifying a worksheet

- Assign to cells
  - WS.cell(row=ROW, column=COLUMN).value = *value*
  - WS.cell(ROW, COLUMN).value = *value*
  - WS[*coordinate*] = *value*

To modify a worksheet, you can either iterate through rows and columns as described above, or assign to the `.value` attribute of individual cells using either `WS.cell()` or `WS["coordinates"]`.

Use `ws.append(iterable)` to append a new row to the spreadsheet.

Use `workbook.save('name.xlsx')` to save the changes. To save to the original workbook, use its name.

TIP Assigning to *cell* is a shortcut for assigning to *cell*.value(). That is, you can say `ws['B4'] = 10`.

NOTE See the later section on inserting and moving rows and columns.

Example

px_modify_sheet.py

```
#!/usr/bin/env python
from datetime import date
import openpyxl as px


def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    add_age_at_inauguration(ws)

    wb.save('presidents1.xlsx')  # save as ...


def make_date(date_str):
    """Convert date string returned by CELL.value into Python date object"""
    year, month, day = date_str.split('-')
    return date(int(year), int(month), int(day))


def add_age_at_inauguration(ws):
    """Add a new column with age of inauguration"""
    new_col = ws.max_column + 1
    print(new_col)
    ws.cell(row=1, column=new_col).value = 'Age at Inauguration'
    for row in range(2, 47):
        birth_date = make_date(ws.cell(row=row, column=4).value)  # treat date as string
        inaugural_date = make_date(ws.cell(row=row, column=8).value)
        raw_age_took_office = inaugural_date - birth_date
        age_took_office = raw_age_took_office.days / 365.25
        ws.cell(row=row, column=new_col).value = age_took_office


if __name__ == '__main__':
    main()
```

Working with formulas

- Assign to cell value as a string
- Be sure to start with '='

To add or update a formula, assign the formula as a string to the cell value.

> **NOTE** Remember that **openpyxl** can not *recalculate* a worksheet.

Example

px_formulas.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    add_age_at_inauguration(ws)

    wb.save('presidents_formula.xlsx')

def add_age_at_inauguration(ws):
    """Add a new column with age of inauguration"""
    new_col = ws.max_column + 1
    print(new_col)
    ws.cell(row=1, column=new_col).value = 'Age at Inauguration'
    for row in range(2, 47):
        new_cell = ws.cell(row=row, column=new_col)
        new_cell.value = '=(H{0}-D{0})/365.25'.format(row)
        new_cell.number_format = '0.0'


if __name__ == '__main__':
    main()
```

Creating a new spreadsheet

- Use the Workbook() function
- One worksheet created by default
- Add worksheets with WB.create_sheet(n)
- Copy worksheets with WB.copy_sheet(n)
- Add data rows with WS.append(iterable)

To create a new spreadsheet file, use the Workbook() function. It creates a new workbook, with a default worksheet named "Sheet1".

Add worksheets with `WB.create_sheet(n)`. The parameter indicates where to insert the new worksheet; if not specified, it is appended.

To get or set the name of the worksheet, use its `.title` property.

To easily add rows to the worksheet, use `WS.append(iterable)`, where *iterable* is an iterable of column values.

Example

px_create_worksheet.py

```
#!/usr/bin/env python
import openpyxl as px

fruits = [
    "pomegranate", "cherry", "apricot", "date", "apple", "lemon",
    "kiwi", "orange", "lime", "watermelon", "guava", "papaya",
    "fig", "pear", "banana", "tamarind", "persimmon", "elderberry",
    "peach", "blueberry", "lychee", "grape"
]

wb = px.Workbook()

ws = wb.active

ws.title = 'fruits'

ws.append(['Fruit', 'Length'])

for fruit in fruits:
    ws.append([fruit, len(fruit)])

# hard way
# for i, fruit in enumerate(fruits, 1):
#     ws.cell(row=i, column=1).value = fruit
#     ws.cell(row=i, column=2).value = len(fruit)

wb.save('fruits.xlsx')
```

Inserting, Deleting, and moving cells

- Insert
  - `ws.insert_rows(row_index, num_rows=1)`
  - `ws.insert_cols(col_index, num_cols=1)`
- Delete
  - `ws.delete_rows(row_index, num_rows)`
  - `ws.delete_cols(col_index, num_cols)`
- Move
  - `ws.move_range(range, rows=row_delta, cols=col_delta)`
- Append
  - `ws.append(iterable)`

To insert one or more blank rows or columns, use `ws.insert_rows()` or `ws.insert_cols()`. The first argument is the positional index of the row or column (1-based), and the second argument is how many columns to insert.

To delete rows or columns, use `ws.delete_rows()` or `ws.delete_cols()`. The first argument is the index of the first row or column to delete; the second is the number of rows or columns.

To move a range of rows and columns, use `ws.move_range()`. The first argument is a range string such as `A1:F10`. Add named arguments `rows` and `cols` to specify how many cells to move. Positive values move down or right, and negative values move up or left. Existing data will be overwritten at the new location of the moved cells.

To append a row of data to a worksheet, pass an iterable to `ws.append()`.

Example

px_insert_delete_move.py

```
#!/usr/bin/env python
import openpyxl as px

RAW_DATA = [47, "Mouse", "Mickey", None, None, "Anaheim", "California", "2025-01-20",
None, "Imagineer"]

def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    insert_cells(ws)
    delete_cells(ws)
    move_cells(ws)
    append_cells(ws)

    print(ws.dimensions)

    wb.save('presidents_insert_delete_move.xlsx')

def append_cells(ws):
    ws.append(RAW_DATA)

def insert_cells(ws):
    ws.insert_rows(1, 3)  # insert three rows at top
    ws.insert_cols(5)  # insert one col at position 5

def delete_cells(ws):
    ws.delete_rows(15,5)
    ws.delete_cols(6)

def move_cells(ws):
    ws.move_range('A43:K45', rows=6, cols=3)


if __name__ == '__main__':
    main()
```

Hiding and freezing columns and sheets

- Hide
  - `ws.column_dimensions[column]`
  - `ws.column_dimensions.group(column, ⋯)`
- Freeze
  - `ws.freeze_panes = 'coordinate'`
- Hide entire sheet
  - `ws.sheet_state = 'hidden'`

You can hide a column by using the `.column_dimensions` property of a worksheet. Specify a column letter inside square brackets, and assign `True` to the `.hidden` property. To hide multiple columns, use `.column_dimensions.group()`. Specify start and end columns, and set the `hidden` argument to `True`

```
ws.column_dimensions['M'].hidden = True
ws.column_dimensions.group(start="C", end="F", hidden=True)
```

To freeze rows and columns for scrolling, assign the coordinates of the first row and column that you want to scroll to `ws.freeze_panes`. For example, to freeze the first 3 columns and start scrolling with column 'D', use

```
ws.freeze_panes = 'A4'
```

You can also hide a worksheet by setting `ws.sheet_state` to `"hidden"`.

Example

px_hide_freeze.py

```
#!/usr/bin/env python
import openpyxl as px


def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    hide_columns(ws)

    wb.save('presidents_hidden.xlsx')

    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    freeze_columns(ws)

    create_hidden_sheet(wb)

    wb.save('presidents_frozen.xlsx')

    wb = px.load_workbook('../DATA/presidents.xlsx')

    create_hidden_sheet(wb)

    wb.save('presidents_hidden_sheet.xlsx')


def hide_columns(ws):
    """Hide single columnn and multiple columns"""

    # hide birthplace column
    ws.column_dimensions['F'].hidden = True

    # hide inauguration columns
    ws.column_dimensions.group(start='H', end='I', hidden=True)

def freeze_columns(ws):
    """Freeze the first 2 columns"""
    ws.freeze_panes = "C1"

def create_hidden_sheet(wb):
    """Add a hidden worksheet to the workbook"""
```

```
    ws = wb.create_sheet(title="secret plans")
    ws.sheet_state = "hidden"


if __name__ == '__main__':
    main()
```

Setting Styles

- Must be set on each cell individually
- Cannot change, once assigned (but can be replaced)
- Copy style to make changes

Each cell has a group of attributes that control its styles and formatting. Most of these have a corresponding class; to change styles, create an instance of the appropriate class and assign it to the attribute.

You can also make a copy of an existing style object, and just change the attributes you need.

Example

px_styles.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    update_last_names(ws)

    wb.save('presidents_styles.xlsx')

def update_last_names(ws):
    """Make the last name column blue and bold"""
    for row in ws['B2:B47']:
        cell = row[0]
        cell.value = cell.value.upper()
        cell.font = px.styles.Font(color='FF0000FF')

if __name__ == '__main__':
    main()
```

Table 21. openpyxl Cell Style Attributes

Cell attribute	Class	Parameters	Default value
font	Font		
		name	'Calibri'
		size	11
		bold	False
		italic	False
		vertAlign	None
		underline	'none'
		strike	False
		color	'FF000000'
fill	PatternFill		
		fill_type	None
		start_color	'FFFFFFFF'
		end_color	'FF000000'
border	Border		
		left	Side(border_style=None, color='FF000000')
		right	Side(border_style=None, color='FF000000')
		top	Side(border_style=None, color='FF000000')
		bottom	Side(border_style=None, color='FF000000')
		diagonal	Side(border_style=None, color='FF000000')
		diagonal_direction	
		outline	Side(border_style=None, color='FF000000')
		vertical	Side(border_style=None, color='FF000000')
		horizontal	Side(border_style=None, color='FF000000')
alignment	Alignment		

Cell attribute	Class	Parameters	Default value
		horizontal	'general'
		vertical	'bottom'
		text_rotation	0
		wrap_text	False
		shrink_to_fit	False
		indent	0
number_format	None	N/A	'General'
protection	Protection		
		locked	True
		hidden	False

Conditional Formatting

- Apply styles per values
- Types
  - Builtin
  - Standard
  - Custom
- Components
  - Differential Style
  - Rule
    - Formula

Conditional formatting means applying styles to cells based on their values. In openpyxl, conditional formatting can be a little complicated.

There are three kinds of rules for conditional formatting:

- Builtin — predefined rules with predefined styles
- Standard — predefined rules with custom styles
- Custom — custom rules with custom styles

Because this is complicated, there are some convenience functions for generating some formats.

Components

Formatting requres *styles*, which are either builtin or configured via a `DifferentialStyle` object, and *rules*, which are embedded in the formats. For custom rules, you provide a *formula* that defines when the rule should be used.

Builtin formats

There are three conditional formats: `ColorScale`, `IconSet`, and `DataBar`. These formats contain various settings, which compare the value to ann integer using one of these types: `num`, `percent`, `max`, `min`, `formula`, or `percentile`.

ColorScale

`ColorScale` provides a rule for a gradient from one color to another for the values within a range. You can add a second `ColorScale` for two gradients.

The convenience function for `ColorScale` is `openpyxl.formatting.rule.ColorScaleRule()`.

IconSet

`IconSet` provides a rule for applying different icons to different values.

The convenience function for `IconSet` is `openpyxl.formatting.rule.IconSetRule()`.

DataBar

`DataBar` provides a rule for adding "data bars", similar to the bars used by mobile phones to indicate signal strength.

The convenience function for `DataBar` is `openpyxl.formatting.rule.DataBarRule()`.

Example

px_conditional_styles.py

```
#!/usr/bin/env python
import openpyxl as px
from openpyxl.formatting.rule import (
    Rule, ColorScale, FormatObject, IconSet, DataBar
)

from openpyxl.styles import Font, PatternFill, Color
from openpyxl.styles.differential import DifferentialStyle

CONDITIONAL_CONFIG = {
    'Republican': {
        'font_color': "FF0000",
        'fill': "FFC0CB",
    },
    'Democratic': {
        'font_color': "0000FF",
        'fill': "ADD8E6",
    },
    'Whig': {
        'font_color': "008000",
        'fill': "98FB98",
    }
}

def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    colorscale_values(ws)

    color_potus_parties(ws)

    icon_values(ws)

    wb.save('presidents_conditional.xlsx')

    wb = px.load_workbook('../DATA/columns_of_numbers.xlsx')
    icon_values(wb.active)
    databar_values(wb.active)
    wb.save('columns_with_icons.xlsx')

def colorscale_values(ws):
    """
```

```
    Add conditional style to the "TERM" column using a builtin
    type.

    :param ws: the worksheet
    :return: None
    """
    first = FormatObject(type="min")
    last = FormatObject(type="max")
    colors = [Color('AA0000'), Color('00AA00')]
    cs2 = ColorScale(cfvo=[first, last], color=colors)
    rule = Rule(type='colorScale', colorScale=cs2)

    last_row = ws.max_row
    ws.conditional_formatting.add(f'A2:A{last_row}', rule)



def color_potus_parties(ws):
    """
        Make Republicans red and Democrats blue, etc.

        This is a custom rule with a custom formula.

        :param ws: Worksheet to format
        :returns: None
    """
    for text, config in CONDITIONAL_CONFIG.items():
        font = Font(color=config['font_color'])
        fill = PatternFill(bgColor=config['fill'])
        dxf = DifferentialStyle(font=font, fill=fill)

        # make a rule for this condition
        rule = Rule(type="expression", dxf=dxf)

        # add an Excel formula to the rule. Cell must be first cell of
        # range; otherwise formatting is offset by difference from first
        # cell to specified cell
        #
        # can use any Excel text operations here
        rule.formula=[f'EXACT("{text}",$J2)']

        # add the rule to desired range
        ws.conditional_formatting.add('J2:J47', rule)

def icon_values(ws):
    """
    Add icons for numeric values in column.
```

```
    :param ws: worksheet to format
    :return: None
    """
    thresholds = [0, 33, 67]
    icons = [FormatObject(type='percent', val=t) for t in thresholds]
    iconset = IconSet(iconSet='3TrafficLights1', cfvo=icons)
    rule = Rule(type='iconSet', iconSet=iconset)
    format_range = f"A2:A{ws.max_row}"
    ws.conditional_formatting.add(format_range, rule)

def databar_values(ws):
    """
    Add conditional databars to worksheet.

    :param ws: worksheet to format
    :return: None
    """
    first = FormatObject(type='min')
    second = FormatObject(type='max')
    data_bar = DataBar(cfvo=[first, second], color="638EC6")
    rule = Rule(type='dataBar', dataBar = data_bar)
    format_range = f"F2:F{ws.max_row}"
    ws.column_dimensions['F'].width = 25  # make column wider for data bar
    ws.conditional_formatting.add(format_range, rule)


if __name__ == '__main__':
    main()
```

Chapter 7 Exercises

Exercise 7-1 (age_of_geeks.py,age_of_geeks_formula.py)

Write a script to compute the average age of the people on the worksheet 'people' in **computer_people.xlsx**. First, you'll have to calculate the age from the birth date. (Some of the people in the worksheet have died. Just include them for purposes of this exercise).

TIP TO calculate the age, get today's date (`datetime.datetime.now()`), subtract the DOB cell value from today's date. This gets a **timedelta** object. Use the **days** attribute of the timedelta divided by 365 to get the age.

NOTE Another way to do this lab (if Excel is available) is to use this Excel formula: `=DATEDIF(DOB,TODAY(),"y")` where DOB is the cell containing the birthdate, such as "D2". Add an additional column. Then add a formula to average the values in this new column. Since OpenPyXL can't recalculate a sheet, open the sheet in Excel to the the results. (You could also use Libre Office or Open Office to open the workbook).

Print out the average.

Exercise 7-2 (knights_to_spreadsheet.py, knights_to_spreadsheet_extra.py

Write a script to create a new spreadsheet with data from the knights.txt file. The first row of the spreadsheet should have the column headings:

Name, Title, Favorite Color, Quest, Comment

The data should start after that.

Save the workbook as knights.xlsx.

NOTE For extra fun, make the headers bold, the name fields red, and the comments in italics.

Index

