INTRODUCING PYTHON

MODERN COMPUTING IN SIMPLE PACKAGES

2ND EDITION

BY BILL LUBANOVIC

Contents

Figure 1-1. Knitted socks

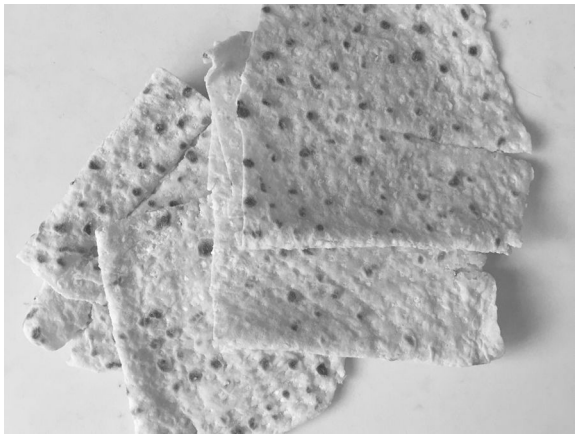

Figure 1-2. Lefse

Example 1-1. countdown.py

```
for countdown in 5, 4, 3, 2, 1, "hey!":
    print(countdown)
```

Example 1-2. spells.py

```
spells = [
    "Riddikulus!",
    "Wingardium Leviosa!",
    "Avada Kedavra!",
    "Expecto Patronum!",
    "Nox!",
    "Lumos!",
    ]
print(spells[3])
```

Example 1-3. quotes.py

```
quotes = {
    "Moe": "A wise guy, huh?",
    "Larry": "Ow!",
    "Curly": "Nyuk nyuk!",
    }
stooge = "Curly"
print(stooge, "says:", quotes[stooge])
```

Example 1-4. archive.py

```
1 import webbrowser
2 import json
3 from urllib.request import urlopen
4
5 print("Let's find an old website.")
6 site = input("Type a website URL: ")
7 era = input("Type a year, month, and day, like 20150613: ")
8 url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
9 response = urlopen(url)
10 contents = response.read()
11 text = contents.decode("utf-8")
12 data = json.loads(text)
13 try:
14     old_site = data["archived_snapshots"]["closest"]["url
15     print("Found this copy: ", old_sit
16     print("It should appear in your browser now.
17     webbrowser.open(old_sit
18 except:
19     print("Sorry, no luck finding", site)
```

Figure 1-3. From the Wayback Machine

Example 1-5. archive2.py

```
1 import webbrowser
2 import requests
3
4 print("Let's find an old website.")
5 site = input("Type a website URL: ")
6 era = input("Type a year, month, and day, like 20150613: ")
7 url = "http://archive.org/wayback/available?url=%s&timestamp=%s" % (site, era)
8 response = requests.get(url)
9 data = response.json()
10 try:
11     old_site = data["archived_snapshots"]["closest"]["url"]
12     print("Found this copy: ", old_site)
13     print("It should appear in your browser now.")
14     webbrowser.open(old_site)
15 except:
16     print("Sorry, no luck finding", site)
```

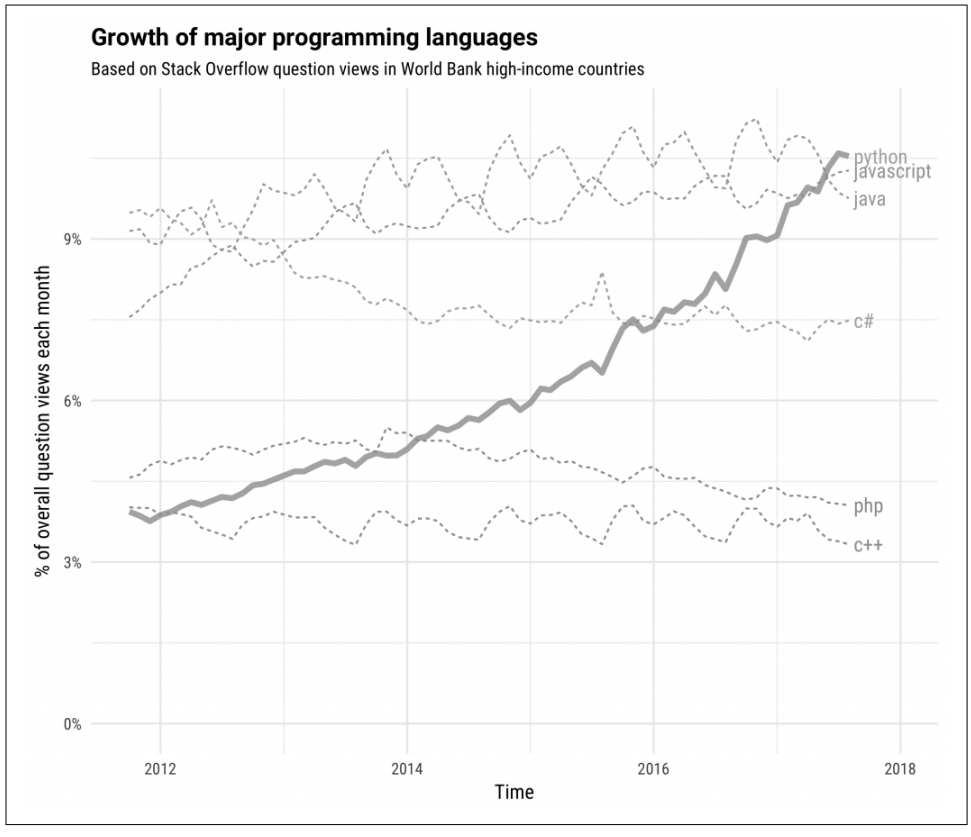

Figure 1-4. Python leads in major programming language growth

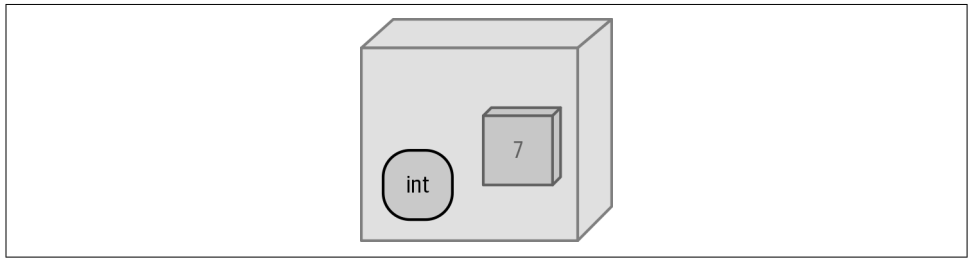

Figure 2-1. An object is like a box; this one is an integer with value 7

Table 2-1. Python's basic data types

Name	Type	Mutable?	Examples	Chapter
Boolean	`bool`	no	`True`, `False`	Chapter 3
Integer	`int`	no	`47`, `25000`, `25_000`	Chapter 3
Floating point	`float`	no	`3.14`, `2.7e5`	Chapter 3
Complex	`complex`	no	`3j`, `5 + 9j`	Chapter 22
Text string	`str`	no	`'alas'`, `"alack"`, `'''a verse attack'''`	Chapter 5
List	`list`	yes	`['Winken', 'Blinken', 'Nod']`	Chapter 7
Tuple	`tuple`	no	`(2, 4, 8)`	Chapter 7
Bytes	`bytes`	no	`b'ab\xff'`	Chapter 12
ByteArray	`bytearray`	yes	`bytearray(...)`	Chapter 12
Set	`set`	yes	`set([3, 5, 7])`	Chapter 8
Frozen set	`frozenset`	no	`frozenset(['Elsa', 'Otto'])`	Chapter 8
Dictionary	`dict`	yes	`{'game': 'bingo', 'dog': 'dingo', 'drummer': 'Ringo'}`	Chapter 8

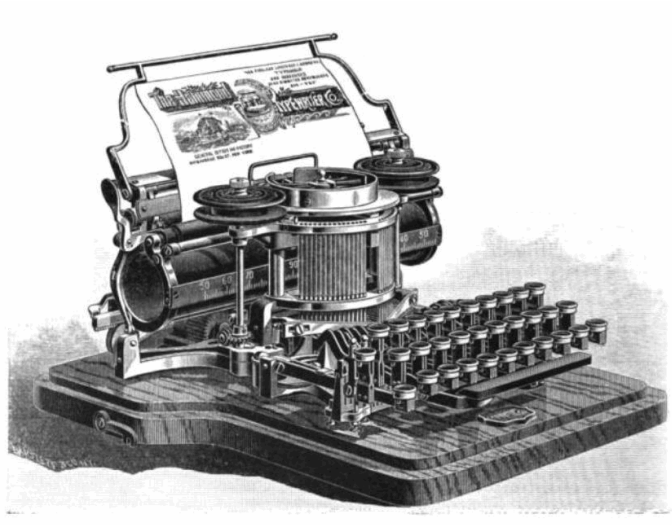

Figure 2-2. Strong typing does not mean push the keys harder

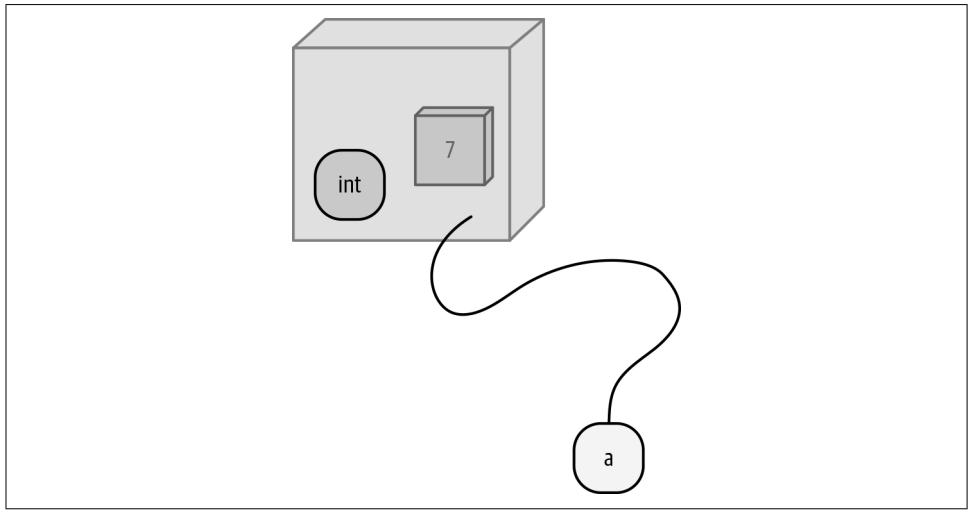

Figure 2-3. Names point to objects (variable a points to an integer object with value 7)

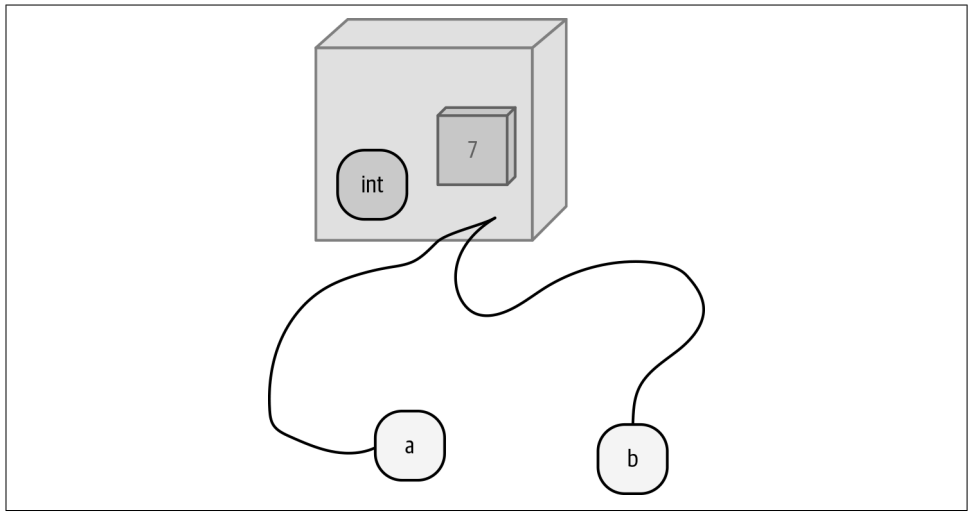

Figure 2-4. Copying a name (now variable b also points to the same integer object)

Figure 3-1. Chester—a fine furry fellow, and inventor of the chesterdigital system

Table 5-1. Conversion types

`%s`	string
`%d`	decimal integer
`%x`	hex integer
`%o`	octal integer
`%f`	decimal float
`%e`	exponential float
`%g`	decimal or exponential
float`%%`	a literal %

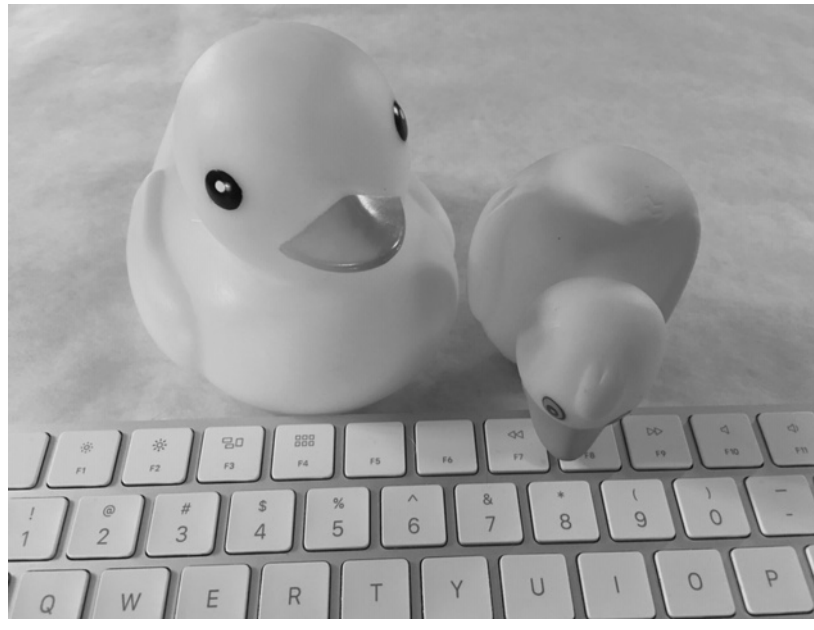

Figure 10-1. Duck typing is not hunt-and-peck

Table 10-1. Magic methods for comparison

Method	Description
`__eq__( self, other )`	*self* == *other*
`__ne__( self, other )`	*self* != *other*
`__lt__( self, other )`	*self* < *other*
`__gt__( self, other )`	*self* > *other*
`__le__( self, other )`	*self* <= *other*
`__ge__( self, other )`	*self* >= *other*

Table 10-2. Magic methods for math

Method	Description
`__add__( self, other )`	*self + other*
`__sub__( self, other )`	*self – other*
`__mul__( self, other )`	*self * other*
`__floordiv__( self, other )`	*self // other*
`__truediv__( self, other )`	*self / other*
`__mod__( self, other )`	*self % other*
`__pow__( self, other )`	*self ** other*

Table 10-3. Other, miscellaneous magic methods

Method	Description
`__str__(`*self*`)`	`str(`*self*`)`
`__repr__(`*self*`)`	`repr(`*self*`)`
`__len__(`*self*`)`	`len(`*self*`)`

Example 11-1. fast.py

```
from random import choice

places = ['McDonalds", "KFC", "Burger King", "Taco Bell",
     "Wendys", "Arbys", "Pizza Hut"]

def pick():  # see the docstring below?
    """Return random fast food place"""
    return choice(places)
```

Example 11-2. lunch.py

```
import fast

place = fast.pick()
print("Let's go to", place)
```

Example 11-3. fast2.py

```
places = ['McDonalds", "KFC", "Burger King", "Taco Bell",
     "Wendys", "Arbys", "Pizza Hut"]

def pick():
    import random
    return random.choice(places)
```

Example 11-4. fast3.py

```
import fast as f
place = f.pick()
print("Let's go to", place)
```

Example 11-5. fast4.py

```
from fast import pick
place = pick()
print("Let's go to", place)
```

Example 11-6. fast5.py

```
from fast import pick as who_cares
place = who_cares()
print("Let's go to", place)
```

Example 11-7. questions.py

```
from sources import fast, advice

print("Let's go to", fast.pick())
print("Should we take out?", advice.give())
```

Example 11-8. choices/fast.py

```
from random import choice

places = ["McDonalds", "KFC", "Burger King", "Taco Bell",
     "Wendys", "Arbys", "Pizza Hut"]

def pick():
    """Return random fast food place"""
    return choice(places)
```

Example 11-9. choices/advice.py

```
from random import choice

answers = ["Yes!", "No!", "Reply hazy", "Sorry, what?"]

def give():
    """Return random advice"""
    return choice(answers)
```

Table 12-1. Encodings

Encoding name	Description
`'ascii'`	Good old seven-bit ASCII
`'utf-8'`	Eight-bit variable-length encoding, and what you almost always want to use
`'latin-1'`	Also known as ISO 8859-1
`'cp-1252'`	A common Windows encoding
`'unicode-escape'`	Python Unicode literal format, `\u`*xxxx* or `\U`*xxxxxxxx*

Table 12-2. Special characters

Pattern	Matches
`\d`	A single digit
`\D`	A single nondigit
`\w`	An alphanumeric character
`\W`	A non-alphanumeric character
`\s`	A whitespace character
`\S`	A nonwhitespace character
`\b`	A word boundary (between a `\w` and a `\W`, in either order)
`\B`	A nonword boundary

Table 12-3. Pattern specifiers

Pattern	Matches
`abc`	Literal `abc`
`(` *expr* `)`	*expr*
expr1 `|` *expr2*	*expr1* or *expr2*
`.`	Any character except `\n`
`^`	Start of source string
`$`	End of source string
prev `?`	Zero or one *prev*
prev `*`	Zero or more *prev*, as many as possible
prev `*?`	Zero or more *prev*, as few as possible
prev `+`	One or more *prev*, as many as possible
prev `+?`	One or more *prev*, as few as possible
prev `{` *m* `}`	*m* consecutive *prev*
prev `{` *m*, *n* `}`	*m* to *n* consecutive *prev*, as many as possible
prev `{` *m*, *n* `}?`	*m* to *n* consecutive *prev*, as few as possible
`[` *abc* `]`	`a` or `b` or `c` (same as `a|b|c`)
`[^` *abc* `]`	*not* (`a` or `b` or `c`)
prev `(?=` *next* `)`	*prev* if followed by *next*
prev `(?!` *next* `)`	*prev* if *not* followed by *next*
`(?<=` *prev* `)` *next*	*next* if preceded by *prev*
`(?<!` *prev* `)` *next*	*next* if *not* preceded by *prev*

Figure 12-1. The O'Reilly tarsier

Table 12-4. Endian specifiers

Specifier	Byte order
<	Little endian
>	Big endian

Table 12-5. Format specifiers

Specifier	Description	Bytes
`x`	Skip a byte	1
`b`	Signed byte	1
`B`	Unsigned byte	1
`h`	Signed short integer	2
`H`	Unsigned short integer	2
`i`	Signed integer	4
`I`	Unsigned integer	4
`l`	Signed long integer	4
`L`	Unsigned long integer	4
`Q`	Unsigned long long integer	8
`f`	Single-precision float	4
`d`	Double-precision float	8
`p`	*count* and characters	1 + *count*
`s`	Characters	*count*

Table 12-6. Bit-level integer operators

Operator	Description	Example	Decimal result	Binary result
`&`	And	`x & y`	`1`	`0b0001`
`|`	Or	`x | y`	`5`	`0b0101`
`^`	Exclusive or	`x ^ y`	`4`	`0b0100`
`~`	Flip bits	`~x`	`-6`	*binary representation depends on int size*
`<<`	Left shift	`x << 1`	`10`	`0b1010`
`>>`	Right shift	`x >> 1`	`2`	`0b0010`

Example 12-1. mammoth.txt

```
We have seen thee, queen of cheese,
Lying quietly at your ease,
Gently fanned by evening breeze,
Thy fair form no flies dare seize.

All gaily dressed soon you'll go
To the great Provincial show,
To be admired by many a beau
In the city of Toronto.

Cows numerous as a swarm of bees,
Or as the leaves upon the trees,
It did require to make thee please,
And stand unrivalled, queen of cheese.

May you not receive a scar as
We have heard that Mr. Harris
Intends to send you off as far as
The great world's show at Paris.

Of the youth beware of these,
For some of them might rudely squeeze
And bite your cheek, then songs or glees
We could not sing, oh! queen of cheese.

We'rt thou suspended from balloon,
You'd cast a shade even at noon,
Folks would think it was the moon
About to fall and crush them soon.
```

Table 13-1. `struct_time` values

Index	Name	Meaning	Values
0	`tm_year`	Year	`0000` to `9999`
1	`tm_mon`	Month	`1` to `12`
2	`tm_mday`	Day of month	`1` to `31`
3	`tm_hour`	Hour	`0` to `23`
4	`tm_min`	Minute	`0` to `59`
5	`tm_sec`	Second	`0` to `61`
6	`tm_wday`	Day of week	`0` (Monday) to `6` (Sunday)
7	`tm_yday`	Day of year	`1` to `366`
8	`tm_isdst`	Daylight savings?	`0` = no, `1` = yes, `-1` = unknown

Table 13-2. Output specifiers for `strftime()`

Format string	Date/time unit	Range
`%Y`	year	`1900-...`
`%m`	month	`01-12`
`%B`	month name	`January, ...`
`%b`	month abbrev	`Jan, ...`
`%d`	day of month	`01-31`
`%A`	weekday name	`Sunday, ...`
`a`	weekday abbrev	`Sun, ...`
`%H`	hour (24 hr)	`00-23`
`%I`	hour (12 hr)	`01-12`
`%p`	AM/PM	`AM, PM`
`%M`	minute	`00-59`
`%S`	second	`00-59`

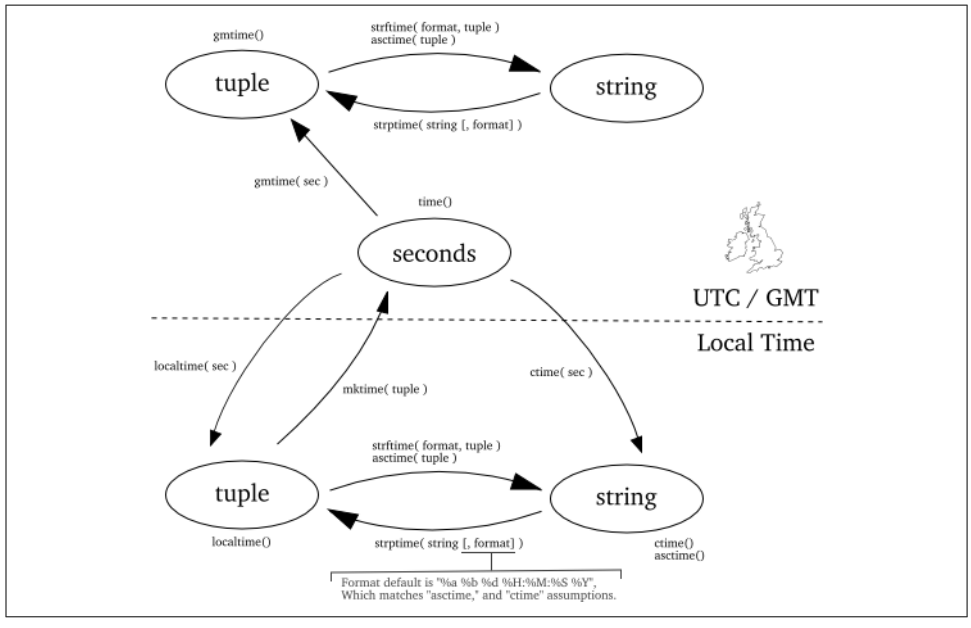

Figure 13-1. Date and time conversions

Example 14-1. convert_image.py

```
from io import BytesIO
from PIL import Image
import sys

def data_to_img(data):
    """Return PIL Image object, with data from in-memory <data>"""
    fp = BytesIO(data)
    return Image.open(fp)    # reads from memory

def img_to_data(img, fmt=None):
    """Return image data from PIL Image <img>, in <fmt> format"""
    fp = BytesIO()
    if not fmt:
        fmt = img.format     # keeps the original format
    img.save(fp, fmt)        # writes to memory
    return fp.getvalue()

def convert_image(data, fmt=None):
    """Convert image <data> to PIL <fmt> image data"""
    img = data_to_img(data)
    return img_to_data(img, fmt)

def get_file_data(name):
    """Return PIL Image object for image file <name>"""
    img = Image.open(name)
    print("img", img, img.format)
    return img_to_data(img)

if __name__ == "__main__":
    for name in sys.argv[1:]:
        data = get_file_data(name)
        print("in", len(data), data[:10])
        for fmt in ("gif", "png", "jpeg"):
            out_data = convert_image(data, fmt)
            print("out", len(out_data), out_data[:10])
```

Example 15-1. mp.py

```
import multiprocessing
import os

def whoami(what):
    print("Process %s says: %s" % (os.getpid(), what))
if __name__ == "__main__":
    whoami("I'm the main program")
    for n in range(4):
        p = multiprocessing.Process(target=whoami,
          args=("I'm function %s" % n,))
        p.start()
```

Example 15-2. mp2.py

```
import multiprocessing
import time
import os

def whoami(name):
    print("I'm %s, in process %s" % (name, os.getpid()))

def loopy(name):
    whoami(name)
    start = 1
    stop = 1000000
    for num in range(start, stop):
        print("\tNumber %s of %s. Honk!" % (num, stop))
        time.sleep(1)

if __name__ == "__main__":
    whoami("main")
    p = multiprocessing.Process(target=loopy, args=("loopy",))
    p.start()
    time.sleep(5)
    p.terminate()
```

Example 15-3. tasks.py

```
from invoke import task

@task
def mytime(ctx):
    import time
    now = time.time()
    time_str = time.asctime(time.localtime(now))
    print("Local time is", timestr)
```

Example 15-4. dishes.py

```
import multiprocessing as mp

def washer(dishes, output):
    for dish in dishes:
        print('Washing', dish, 'dish')
        output.put(dish)

def dryer(input):
    while True:
        dish = input.get()
        print('Drying', dish, 'dish')
        input.task_done()

dish_queue = mp.JoinableQueue()
dryer_proc = mp.Process(target=dryer, args=(dish_queue,))
dryer_proc.daemon = True
dryer_proc.start()

dishes = ['salad', 'bread', 'entree', 'dessert']
washer(dishes, dish_queue)
dish_queue.join()
```

Example 15-5. thread1.py

```
import threading

def do_this(what):
    whoami(what)

def whoami(what):
    print("Thread %s says: %s" % (threading.current_thread(), what))

if __name__ == "__main__":
    whoami("I'm the main program")
    for n in range(4):
        p = threading.Thread(target=do_this,
          args=("I'm function %s" % n,))
        p.start()
```

Example 15-6. thread_dishes.py

```
import threading, queue
import time

def washer(dishes, dish_queue):
    for dish in dishes:
        print ("Washing", dish)
        time.sleep(5)
        dish_queue.put(dish)

def dryer(dish_queue):
    while True:
        dish = dish_queue.get()
        print ("Drying", dish)
        time.sleep(10)
        dish_queue.task_done()


dish_queue = queue.Queue()
for n in range(2):
    dryer_thread = threading.Thread(target=dryer, args=(dish_queue,))
    dryer_thread.start()

dishes = ['salad', 'bread', 'entree', 'dessert']
washer(dishes, dish_queue)
dish_queue.join()
```

Example 15-7. cf.py

```
from concurrent import futures
import math
import time
import sys

def calc(val):
    time.sleep(1)
    result = math.sqrt(float(val))
    return result

def use_threads(num, values):
    t1 = time.time()
    with futures.ThreadPoolExecutor(num) as tex:
        results = tex.map(calc, values)
    t2 = time.time()
    return t2 - t1

def use_processes(num, values):
    t1 = time.time()
    with futures.ProcessPoolExecutor(num) as pex:
        results = pex.map(calc, values)
    t2 = time.time()
    return t2 - t1

def main(workers, values):
    print(f"Using {workers} workers for {len(values)} values")
    t_sec = use_threads(workers, values)
    print(f"Threads took {t_sec:.4f} seconds")
    p_sec = use_processes(workers, values)
    print(f"Processes took {p_sec:.4f} seconds")

if __name__ == '__main__':
    workers = int(sys.argv[1])
    values = list(range(1, 6)) # 1 .. 5
    main(workers, values)
```

Example 15-8. cf2.py

```
from concurrent import futures
import math
import sys

def calc(val):
    result = math.sqrt(float(val))
    return val, result

def use_threads(num, values):
    with futures.ThreadPoolExecutor(num) as tex:
        tasks = [tex.submit(calc, value) for value in values]
        for f in futures.as_completed(tasks):
             yield f.result()

def use_processes(num, values):
    with futures.ProcessPoolExecutor(num) as pex:
        tasks = [pex.submit(calc, value) for value in values]
        for f in futures.as_completed(tasks):

def main(workers, values):
    print(f"Using {workers} workers for {len(values)} values")
    print("Using threads:")
    for val, result in use_threads(workers, values):
        print(f'{val} {result:.4f}')
    print("Using processes:")
    for val, result in use_processes(workers, values):
        print(f'{val} {result:.4f}')

if __name__ == '__main__':
    workers = 3
    if len(sys.argv) > 1:
        workers = int(sys.argv[1])
    values = list(range(1, 6)) # 1 .. 5
    main(workers, values)
```

Example 15-9. gevent_test.py

```
import gevent
from gevent import socket
hosts = ['www.crappytaxidermy.com', 'www.walterpottertaxidermy.com',
    'www.antique-taxidermy.com']
jobs = [gevent.spawn(gevent.socket.gethostbyname, host) for host in hosts]
gevent.joinall(jobs, timeout=5)
for job in jobs:
    print(job.value)
```

Example 15-10. gevent_monkey.py

```
import gevent
from gevent import monkey; monkey.patch_all()
import socket
hosts = ['www.crappytaxidermy.com', 'www.walterpottertaxidermy.com',
    'www.antique-taxidermy.com']
jobs = [gevent.spawn(socket.gethostbyname, host) for host in hosts]
gevent.joinall(jobs, timeout=5)
for job in jobs:
    print(job.value)
```

Example 15-11. knock_server.py

```
from twisted.internet import protocol, reactor

class Knock(protocol.Protocol):
    def dataReceived(self, data):
        print('Client:', data)
        if data.startswith("Knock knock"):
            response = "Who's there?"
        else:
            response = data + " who?"
        print('Server:', response)
        self.transport.write(response)

class KnockFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return Knock()

reactor.listenTCP(8000, KnockFactory())
reactor.run()
```

Example 15-12. knock_client.py

```
from twisted.internet import reactor, protocol

class KnockClient(protocol.Protocol):
    def connectionMade(self):
        self.transport.write("Knock knock")

    def dataReceived(self, data):
        if data.startswith("Who's there?"):
            response = "Disappearing client"
            self.transport.write(response)
        else:
            self.transport.loseConnection()
            reactor.stop()

class KnockFactory(protocol.ClientFactory):
    protocol = KnockClient

def main():
    f = KnockFactory()
    reactor.connectTCP("localhost", 8000, f)
    reactor.run()

if __name__ == '__main__':
    main()
    Client: Disappearing client
    Server: Disappearing client who?
```

Example 15-13. redis_washer.py

```
import redis
conn = redis.Redis()
print('Washer is starting')
dishes = ['salad', 'bread', 'entree', 'dessert']
for dish in dishes:
    msg = dish.encode('utf-8')
    conn.rpush('dishes', msg)
    print('Washed', dish)
conn.rpush('dishes', 'quit')
print('Washer is done')
```

Example 15-14. redis_dryer.py

```
import redis
conn = redis.Redis()
print('Dryer is starting')
while True:
    msg = conn.blpop('dishes')
    if not msg:
        break
    val = msg[1].decode('utf-8')
    if val == 'quit':
        break
    print('Dried', val)
print('Dishes are dried')
```

Example 15-15. redis_dryer2.py

```
def dryer():
    import redis
    import os
    import time
    conn = redis.Redis()
    pid = os.getpid()
    timeout = 20
    print('Dryer process %s is starting' % pid)
    while True:
        msg = conn.blpop('dishes', timeout)
        if not msg:
            break
        val = msg[1].decode('utf-8')
        if val == 'quit':
            break
        print('%s: dried %s' % (pid, val))
        time.sleep(0.1)
    print('Dryer process %s is done' % pid)

import multiprocessing
DRYERS=3
for num in range(DRYERS):
    p = multiprocessing.Process(target=dryer)
    p.start()
```

Example 16-1. villains.csv

```
first,last
Doctor,No
Rosa,Klebb
Mister,Big
Auric,Goldfinger
Ernst,Blofeld
```

Example 16-2. Read CSV with Pandas

```
>>> import pandas
>>>
>>> data = pandas.read_csv('villains.csv')
>>> print(data)
    first        last
0  Doctor          No
1    Rosa       Klebb
2  Mister         Big
3   Auric  Goldfinger
4   Ernst     Blofeld
```

Table 16-1. Basic SQL DDL commands

Operation	SQL pattern	SQL example
Create a database	`CREATE DATABASE` *dbname*	`CREATE DATABASE d`
Select current database	`USE` *dbname*	`USE d`
Delete a database and its tables	`DROP DATABASE` *dbname*	`DROP DATABASE d`
Create a table	`CREATE TABLE` *tbname* `(` *coldefs* `)`	`CREATE TABLE t (id INT, count INT)`
Delete a table	`DROP TABLE` *tbname*	`DROP TABLE t`
Remove all rows from a table	`TRUNCATE TABLE` *tbname*	`TRUNCATE TABLE t`

Table 16-2. Basic SQL DML commands

Operation	SQL pattern	SQL example
Add a row	`INSERT INTO` *tbname* `VALUES( ... )`	`INSERT INTO t VALUES(7, 40)`
Select all rows and columns	`SELECT * FROM` *tbname*	`SELECT * FROM t`
Select all rows, some columns	`SELECT` *cols* `FROM` *tbname*	`SELECT id, count FROM t`
Select some rows, some columns	`SELECT` *cols* `FROM` *tbname* `WHERE` *condition*	`SELECT id, count from t WHERE count > 5 AND id = 9`
Change some rows in a column	`UPDATE` *tbname* `SET` *col* = *value* `WHERE` *condition*	`UPDATE t SET count=3 WHERE id=5`
Delete some rows	`DELETE FROM` *tbname* `WHERE` *condition*	`DELETE FROM t WHERE count <= 10 OR id = 16`

Table 16-3. MySQL drivers

Name	Link	Pypi package	Import as	Notes
mysqlclient	*https://https://mysqlclient.readthedocs.io*	mysql-connector-python	`MySQLdb`	
MySQL Connector	*http://bit.ly/mysql-cpdg*	mysql-connector-python	`mysql.connector`	
PYMySQL	*https://github.com/petehunt/PyMySQL*	pymysql	`pymysql`	
oursql	*http://pythonhosted.org/oursql*	oursql	`oursql`	Requires the MySQL C client libraries

Table 16-4. PostgreSQL drivers

Name	Link	Pypi package	Import as	Notes
psycopg2	*http://initd.org/psycopg*	psycopg2	psycopg2	Needs `pg_config` from PostgreSQL client tools
py-postgresql	*https://pypi.org/project/py-postgresql*	py-postgresql	postgresql	

Table 16-5. SQLAlchemy connection

dialect	driver
`sqlite`	`pysqlite` (or omit)
`mysql`	`mysqlconnector`
`mysql`	`pymysql`
`mysql`	`oursql`
`postgresql`	`psycopg2`
`postgresql`	`pypostgresql`

Table 16-6. Document databases

Database	Python API
Mongo (*https://www.mongodb.com*)	tools (*https://api.mongodb.com/python/current/tools.html*)
DynamoDB (*https://aws.amazon.com/dynamodb*)	`boto3` (*https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Python.html*)
CouchDB (*http://couchdb.apache.org*)	`couchdb` (*https://couchdb-python.readthedocs.io/en/latest/index.html*)

Table 16-7. Temporal databases

Database	Python API
InfluxDB (*https://www.influxdata.com*)	`influx-client` (*https://pypi.org/project/influx-client*)
kdb+ (*https://kx.com*)	PyQ (*https://code.kx.com/v2/interfaces/pyq/*)
Prometheus (*https://prometheus.io*)	`prometheus_client` (*https://github.com/prometheus/client_python/blob/master/README.md*)
TimescaleDB (*https://www.timescale.com*)	(PostgreSQL clients)
OpenTSDB (*http://opentsdb.net*)	`potsdb` (*https://pypi.org/project/potsdb*)
PyStore (*https://github.com/ranaroussi/pystore*)	`PyStore` (*https://pypi.org/project/PyStore*)

Table 16-8. Graph databases

Database	Python API
Neo4J (*https://neo4j.com*)	`py2neo` (*https://py2neo.org/v3*)
OrientDB (*https://orientdb.com*)	`pyorient` (*https://orientdb.com/docs/last/PyOrient.html*)
ArangoDB (*https://www.arangodb.com*)	`pyArango` (*https://github.com/ArangoDB-Community/pyArango*)

Table 16-9. NoSQL databases

Database	Python API
Cassandra (*http://cassandra.apache.org*)	`pycassa` (*https://github.com/pycassa/pycassa*)
CouchDB (*http://couchdb.apache.org*)	`couchdb-python` (*https://github.com/djc/couchdb-python*)
HBase (*http://hbase.apache.org*)	`happybase` (*https://github.com/wbolster/happybase*)
Kyoto Cabinet (*http://fallabs.com/kyotocabinet*)	`kyotocabinet` (*http://bit.ly/kyotocabinet*)
MongoDB (*http://www.mongodb.org*)	`mongodb` (*http://api.mongodb.org/python/current*)
Pilosa (*https://www.pilosa.com*)	`python-pilosa` (*https://github.com/pilosa/python-pilosa*)
Riak (*http://basho.com/riak*)	`riak-python-client` (*https://github.com/basho/riak-python-client*)

Table 16-10. Full-text databases

Site	Python API
Lucene (*http://lucene.apache.org*)	`pylucene` (*http://lucene.apache.org/pylucene*)
Solr (*http://lucene.apache.org/solr*)	`SolPython` (*http://wiki.apache.org/solr/SolPython*)
ElasticSearch (*http://www.elasticsearch.org*)	`elasticsearch` (*https://elasticsearch-py.readthedocs.io*)
Sphinx (*http://sphinxsearch.com*)	`sphinxapi` (*http://bit.ly/sphinxapi*)
Xapian (*http://xapian.org*)	`xappy` (*https://code.google.com/p/xappy*)
Whoosh (*http://bit.ly/mchaput-whoosh*)	(written in Python, includes an API)

Example 17-1. udp_server.py

```
from datetime import datetime
import socket

server_address = ('localhost', 6789)
max_size = 4096

print('Starting the server at', datetime.now())
print('Waiting for a client to call.')
server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(server_address)

data, client = server.recvfrom(max_size)

print('At', datetime.now(), client, 'said', data)
server.sendto(b'Are you talking to me?', client)
server.close()
```

Example 17-2. udp_client.py

```
import socket
from datetime import datetime

server_address = ('localhost', 6789)
max_size = 4096

print('Starting the client at', datetime.now())
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client.sendto(b'Hey!', server_address)
data, server = client.recvfrom(max_size)
print('At', datetime.now(), server, 'said', data)
client.close()
```

Example 17-3. tcp_client.py

```
import socket
from datetime import datetime

address = ('localhost', 6789)
max_size = 1000

print('Starting the client at', datetime.now())
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(address)
client.sendall(b'Hey!')
data = client.recv(max_size)
print('At', datetime.now(), 'someone replied', data)
client.close()
```

Example 17-4. tcp_server.py

```
from datetime import datetime
import socket

address = ('localhost', 6789)
max_size = 1000

print('Starting the server at', datetime.now())
print('Waiting for a client to call.')
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(address)
server.listen(5)

client, addr = server.accept()
data = client.recv(max_size)

print('At', datetime.now(), client, 'said', data)
client.sendall(b'Are you talking to me?')
client.close()
server.close()
```

Example 17-5. zmq_server.py

```
import zmq

host = '127.0.0.1'
port = 6789
context = zmq.Context()
server = context.socket(zmq.REP)
server.bind("tcp://%s:%s" % (host, port))
while True:
    #  Wait for next request from client
    request_bytes = server.recv()
    request_str = request_bytes.decode('utf-8')
    print("That voice in my head says: %s" % request_str)
    reply_str = "Stop saying: %s" % request_str
    reply_bytes = bytes(reply_str, 'utf-8')
    server.send(reply_bytes)
```

Example 17-6. zmq_client.py

```
import zmq

host = '127.0.0.1'
port = 6789
context = zmq.Context()
client = context.socket(zmq.REQ)
client.connect("tcp://%s:%s" % (host, port))
for num in range(1, 6):
    request_str = "message #%s" % num
    request_bytes = request_str.encode('utf-8')
    client.send(request_bytes)
    reply_bytes = client.recv()
    reply_str = reply_bytes.decode('utf-8')
    print("Sent %s, received %s" % (request_str, reply_str))
```

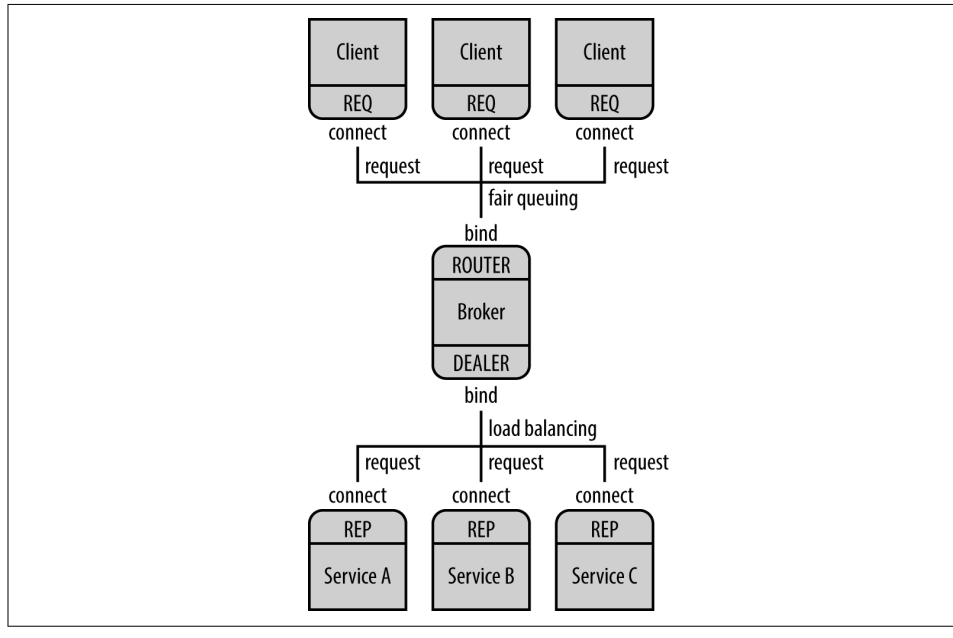

Figure 17-1. Using a broker to connect multiple clients and services.

Example 17-7. redis_pub.py

```
import redis
import random

conn = redis.Redis()
cats = ['siamese', 'persian', 'maine coon', 'norwegian forest']
hats = ['stovepipe', 'bowler', 'tam-o-shanter', 'fedora']
for msg in range(10):
    cat = random.choice(cats)
    hat = random.choice(hats)
    print('Publish: %s wears a %s' % (cat, hat))
    conn.publish(cat, hat)
```

Example 17-8. redis_sub.py

```
import redis
conn = redis.Redis()

topics = ['maine coon', 'persian']
sub = conn.pubsub()
sub.subscribe(topics)
for msg in sub.listen():
    if msg['type'] == 'message':
        cat = msg['channel']
        hat = msg['data']
        print('Subscribe: %s wears a %s' % (cat, hat))
```

Example 17-9. zmq_pub.py

```
import zmq
import random
import time
host = '*'
port = 6789
ctx = zmq.Context()
pub = ctx.socket(zmq.PUB)
pub.bind('tcp://%s:%s' % (host, port))
cats = ['siamese', 'persian', 'maine coon', 'norwegian forest']
hats = ['stovepipe', 'bowler', 'tam-o-shanter', 'fedora']
time.sleep(1)
for msg in range(10):
    cat = random.choice(cats)
    cat_bytes = cat.encode('utf-8')
    hat = random.choice(hats)
    hat_bytes = hat.encode('utf-8')
    print('Publish: %s wears a %s' % (cat, hat))
    pub.send_multipart([cat_bytes, hat_bytes])
```

Example 17-10. zmq_sub.py

```
import zmq
host = '127.0.0.1'
port = 6789
ctx = zmq.Context()
sub = ctx.socket(zmq.SUB)
sub.connect('tcp://%s:%s' % (host, port))
topics = ['maine coon', 'persian']
for topic in topics:
    sub.setsockopt(zmq.SUBSCRIBE, topic.encode('utf-8'))
while True:
    cat_bytes, hat_bytes = sub.recv_multipart()
    cat = cat_bytes.decode('utf-8')
    hat = hat_bytes.decode('utf-8')
    print('Subscribe: %s wears a %s' % (cat, hat))
```

Example 17-11. xmlrpc_server.py

```
from xmlrpc.server import SimpleXMLRPCServer

def double(num):
    return num * 2

server = SimpleXMLRPCServer(("localhost", 6789))
server.register_function(double, "double")
server.serve_forever()
```

Example 17-12. xmlrpc_client.py

```
import xmlrpc.client

proxy = xmlrpc.client.ServerProxy("http://localhost:6789/")
num = 7
result = proxy.double(num)
print("Double %s is %s" % (num, result))
```

Example 17-13. jsonrpc_server.py

```
from jsonrpcserver import method, serve

@method
def double(num):
    return num * 2

if __name__ == "__main__":
    serve()
```

Example 17-14. jsonrpc_client.py

```
from jsonrpcclient import request

num = 7
response = request("http://localhost:5000", "double", num=num)
print("Double", num, "is", response.data.result)
```

Example 17-15. msgpack_server.py

```
from msgpackrpc import Server, Address

class Services():
    def double(self, num):
        return num * 2

server = Server(Services())
server.listen(Address("localhost", 6789))
server.start()
```

Example 17-16. msgpack_client.py

```
from msgpackrpc import Client, Address

client = Client(Address("localhost", 6789))
num = 8
result =  client.call('double', num)
print("Double %s is %s" % (num, result))
```

Example 17-17. zerorpc_server.py

```
import zerorpc

class RPC():
    def double(self, num):
        return 2 * num

server = zerorpc.Server(RPC())
server.bind("tcp://0.0.0.0:4242")
server.run()
```

Example 17-18. zerorpc_client.py

```
import zerorpc

client = zerorpc.Client()
client.connect("tcp://127.0.0.1:4242")
num = 7
result = client.double(num)
print("Double", num, "is", result)
```

Example 18-1. ia.py

```
import json
import sys

import requests

def search(title):
     url = "http://archive.org/advancedsearch.php"
     params = {"q": f"title:({title})",
               "output": "json",
               "fields": "identifier,title",
               "rows": 50,
               "page": 1,}
     resp = requests.get(url, params=params)
     return resp.json()

if __name__ == "__main__":
     title = sys.argv[1]
     data = search(title)
     docs = data["response"]["docs"]
     print(f"Found {len(docs)} items, showing first 10")
     print("identifier\ttitle")
     for row in docs[:10]:
         print(row["identifier"], row["title"], sep="\t")
```

Example 18-2. home.wsgi

```
import bottle

application = bottle.default_app()

@bottle.route('/')

def home():
    return "apache and wsgi, sitting in a tree"
```

Example 18-3. bottle1.py

```
from bottle import route, run

@route('/')
def home():
  return "It isn't fancy, but it's my home page"

run(host='localhost', port=9999)
```

Example 18-4. bottle2.py

```
from bottle import route, run, static_file

@route('/')
def main():
    return static_file('index.html', root='.')

run(host='localhost', port=9999)
```

Example 18-5. bottle3.py

```
from bottle import route, run, static_file

@route('/')
def home():
    return static_file('index.html', root='.')

@route('/echo/<thing>')
def echo(thing):
    return "Say hello to my little friend: %s!" % thing

run(host='localhost', port=9999)
```

Example 18-6. bottle_test.py

```
import requests

resp = requests.get('http://localhost:9999/echo/Mothra')
if resp.status_code == 200 and \
  resp.text == 'Say hello to my little friend: Mothra!':
    print('It worked! That almost never happens!')
else:
    print('Argh, got this:', resp.text)
```

Example 18-7. flask1.py

```
from flask import Flask

app = Flask(__name__, static_folder='.', static_url_path='')

@app.route('/')
def home():
    return app.send_static_file('index.html')

@app.route('/echo/<thing>')
def echo(thing):
    return "Say hello to my little friend: %s" % thing

app.run(port=9999, debug=True)
```

Example 18-8. flask2.html

```
<html>
<head>
<title>Flask2 Example</title>
</head>
<body>
Say hello to my little friend: {{ thing }}
</body>
</html>
```

Example 18-9. flask2.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/echo/<thing>')
def echo(thing):
    return render_template('flask2.html', thing=thing)

app.run(port=9999, debug=True)
```

Example 18-10. flask3a.py

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/echo/<thing>/<place>')
def echo(thing, place):
    return render_template('flask3.html', thing=thing, place=place)

app.run(port=9999, debug=True)
```

Example 18-11. flask3b.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/echo/')
def echo():
    thing = request.args.get('thing')
    place = request.args.get('place')
    return render_template('flask3.html', thing=thing, place=place)
app.run(port=9999, debug=True)
```

Example 18-12. flask3c.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/echo/')
def echo():
    kwargs = {}
    kwargs['thing'] = request.args.get('thing')
    kwargs['place'] = request.args.get('place')
    return render_template('flask3.html', **kwargs)

app.run(port=9999, debug=True)
```

Figure 18-1. `webview` display window

Example 18-13. links.py

```
def get_links(url):
    import requests
    from bs4 import BeautifulSoup as soup
    result = requests.get(url)
    page = result.text
    doc = soup(page)
    links = [element.get('href') for element in doc.find_all('a')]
    return links

if __name__ == '__main__':
    import sys
    for url in sys.argv[1:]:
        print('Links in', url)
        for num, link in enumerate(get_links(url), start=1):
            print(num, link)
        print()
```

Example 18-14. iamovies.py

```
"""Find a video at the Internet Archive
by a partial title match and display it."""

import sys
import webbrowser
import requests

def search(title):
    """Return a list of 3-item tuples (identifier,
       title, description) about videos
       whose titles partially match :title."""
    search_url = "https://archive.org/advancedsearch.php"
    params = {
        "q": "title:({}) AND mediatype:(movies)".format(title),
        "fl": "identifier,title,description",
        "output": "json",
        "rows": 10,
        "page": 1,
        }
    resp = requests.get(search_url, params=params)
    data = resp.json()
    docs = [(doc["identifier"], doc["title"], doc["description"])
            for doc in data["response"]["docs"]]
    return docs

def choose(docs):
    """Print line number, title and truncated description for
       each tuple in :docs. Get the user to pick a line
       number. If it's valid, return the first item in the
       chosen tuple (the "identifier"). Otherwise, return None."""
    last = len(docs) -
    for num, doc in enumerate(docs
        print(f"{num}: ({doc[1]}) {doc[2][:30]}...
    index = input(f"Which would you like to see (0 to {last})?
    tr
        return docs[int(index)][
    excep
        return No

def display(identifier):
    """Display the Archive video with :identifier in the browser"
    details_url = "https://archive.org/details/{}".format(identifie
    print("Loading", details_ur
    webbrowser.open(details_ur
```

```
45
46 def main(title):
47     """Find any movies that match :title.
48        Get the user's choice and display it in the browser."""
49     identifiers = search(title)
50     if identifiers:
51         identifier = choose(identifiers)
52         if identifier:
53             display(identifier)
54         else:
55             print("Nothing selected")
56     else:
57         print("Nothing found for", title)
58
59 if __name__ == "__main__":
60     main(sys.argv[1])
```

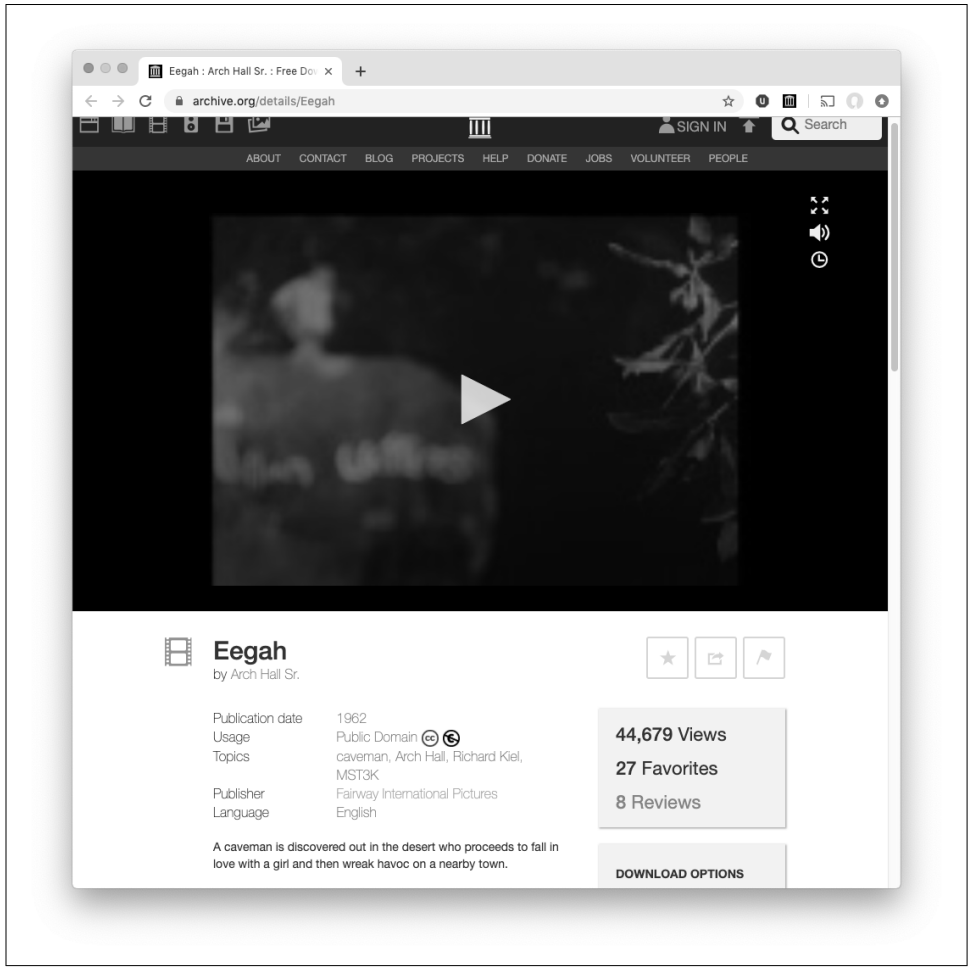

Figure 18-2. Movie search result

Figure 19-1. Startup screen for PyCharm

Example 19-1. ftoc1.py

```
def ftoc(f_temp):
    "Convert Fahrenheit temperature <f_temp> to Celsius and return it."
    f_boil_temp = 212.0
    f_freeze_temp = 32.0
    c_boil_temp = 100.0
    c_freeze_temp = 0.0
    f_range = f_boil_temp - f_freeze_temp
    c_range = c_boil_temp - c_freeze_temp
    f_c_ratio = c_range / f_range
    c_temp = (f_temp - f_freeze_temp) * f_c_ratio + c_freeze_temp
    return c_temp

if __name__ == '__main__':
    for f_temp in [-40.0, 0.0, 32.0, 100.0, 212.0]:
        c_temp = ftoc(f_temp)
        print('%f F => %f C' % (f_temp, c_temp))
```

Example 19-2. ftoc2.py

```
F_BOIL_TEMP = 212.0
F_FREEZE_TEMP = 32.0
C_BOIL_TEMP = 100.0
C_FREEZE_TEMP = 0.0
F_RANGE = F_BOIL_TEMP - F_FREEZE_TEMP
C_RANGE = C_BOIL_TEMP - C_FREEZE_TEMP
F_C_RATIO = C_RANGE / F_RANGE

def ftoc(f_temp):
    "Convert Fahrenheit temperature <f_temp> to Celsius and return it."
    c_temp = (f_temp - F_FREEZE_TEMP) * F_C_RATIO + C_FREEZE_TEMP
    return c_temp

if __name__ == '__main__':
    for f_temp in [-40.0, 0.0, 32.0, 100.0, 212.0]:
        c_temp = ftoc(f_temp)
        print('%f F => %f C' % (f_temp, c_temp))
```

Example 19-3. style1.py

```
a = 1
b = 2
print(a)
print(b)
print(c)
```

Example 19-4. style2.py

```
a = 1
b = 2
c = 3
print(a)
print(b)
print(c)
```

```
$ pylint style2.py

No config file found, using default configuration
************* Module style2
C:  1,0: Missing docstring
C:  1,0: Invalid name "a" for type constant
  (should match (([A-Z_][A-Z0-9_]*)|(__.*__))$)
C:  2,0: Invalid name "b" for type constant
  (should match (([A-Z_][A-Z0-9_]*)|(__.*__))$)
C:  3,0: Invalid name "c" for type constant
  (should match (([A-Z_][A-Z0-9_]*)|(__.*__))$)
```

Example 19-5. style3.py

```
"Module docstring goes here"

def func():
    "Function docstring goes here. Hi, Mom!"
    first = 1
    second = 2
    third = 3
    print(first)
    print(second)
    print(third)

func()
```

```
$ pylint style3.py
No config file found, using default configuration
```

Example 19-6. cap.py

```
def just_do_it(text):
    return text.capitalize()
```

Example 19-7. test_cap.py

```
import unittest
import cap

class TestCap(unittest.TestCase):

    def setUp(self):
        pass

    def tearDown(self):
        pass

    def test_one_word(self):
        text = 'duck'
        result = cap.just_do_it(text)
        self.assertEqual(result, 'Duck')

    def test_multiple_words(self):
        text = 'a veritable flock of ducks'
        result = cap.just_do_it(text)
        self.assertEqual(result, 'A Veritable Flock Of Ducks')


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

Example 19-8. cap.py, revised

```
def just_do_it(text):
    return text.title()
```

Example 19-9. test_cap.py, revised

```
    def test_words_with_apostrophes(self):
        text = "I'm fresh out of ideas"
        result = cap.just_do_it(text)
        self.assertEqual(result, "I'm Fresh Out Of Ideas")
```

Example 19-10. cap.py, re-revised

```
def just_do_it(text):
    from string import capwords
    return capwords(text)
```

```
$ python test_cap.py

...
----------------------------------------------------------------------
Ran 3 tests in 0.004s

OK
```

Example 19-11. test_cap.py, re-revised

```
    def test_words_with_quotes(self):
        text = "\"You're despicable,\" said Daffy Duck"
        result = cap.just_do_it(text)
        self.assertEqual(result, "\"You're Despicable,\" Said Daffy Duck")
```

Example 19-12. cap2.py

```
def just_do_it(text):
    """
    >>> just_do_it('duck')
    'Duck'
    >>> just_do_it('a veritable flock of ducks')
    'A Veritable Flock Of Ducks'
    >>> just_do_it("I'm fresh out of ideas")
    "I'm Fresh Out Of Ideas"
    """
    from string import capwords
    return capwords(text)

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Example 19-13. test_cap_nose.py

```
import cap2
from nose.tools import eq_

def test_one_word():
    text = 'duck'
    result = cap.just_do_it(text)
    eq_(result, 'Duck')

def test_multiple_words():
    text = 'a veritable flock of ducks'
    result = cap.just_do_it(text)
    eq_(result, 'A Veritable Flock Of Ducks')

def test_words_with_apostrophes():
    text = "I'm fresh out of ideas"
    result = cap.just_do_it(text)
    eq_(result, "I'm Fresh Out Of Ideas")

def test_words_with_quotes():
    text = "\"You're despicable,\" said Daffy Duck"
    result = cap.just_do_it(text)
    eq_(result, "\"You're Despicable,\" Said Daffy Duck")
```

Example 19-14. dump.py

```
def dump(func):
    "Print input arguments and output value(s)"
    def wrapped(*args, **kwargs):
        print("Function name:", func.__name__)
        print("Input arguments:", ' '.join(map(str, args)))
        print("Input keyword arguments:", kwargs.items())
        output = func(*args, **kwargs)
        print("Output:", output)
        return output
    return wrapped
```

Example 19-15. test_dump.py

```
from dump import dump

@dump
def double(*args, **kwargs):
    "Double every argument"
    output_list = [ 2 * arg for arg in args ]
    output_dict =  { k:2*v for k,v in kwargs.items() }
    return output_list, output_dict

if __name__ == '__main__':
    output = double(3, 5, first=100, next=98.6, last=-40)
```

Example 19-16. cities.csv

```
France, Paris
venuzuela,caracas
  LithuaniA,vilnius
     quit
```

Example 19-17. capitals.py

```
def process_cities(filename):
    with open(filename, 'rt') as file:
        for line in file:
            line = line.strip()
            if 'quit' in line.lower():
                return
            country, city = line.split(',')
            city = city.strip()
            country = country.strip()
            print(city.title(), country.title(), sep=',')

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

Example 19-18. cities2.csv

```
argentina,buenos aires
bolivia,la paz
brazil,brasilia
chile,santiago
colombia,Bogotá
ecuador,quito
falkland islands,stanley
french guiana,cayenne
guyana,georgetown
paraguay,Asunción
peru,lima
suriname,paramaribo
uruguay,montevideo
venezuela,caracas
quit
```

Example 19-19. capitals2.py

```
def process_cities(filename):
    with open(filename, 'rt') as file:
        for line in file:
            line = line.strip()
            if 'quit' == line.lower():
                return
            country, city = line.split(',')
            city = city.strip()
            country = country.strip()
            print(city.title(), country.title(), sep=',')

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

Example 19-20. time1.py

```
from time import time

t1 = time()
num = 5
num *= 2
print(time() - t1)
```

Example 19-21. time2.py

```
from time import time, sleep

t1 = time()
sleep(1.0)
print(time() - t1)
```

Example 19-22. timeit1.py

```
from timeit import timeit
print(timeit('num = 5; num *= 2', number=1))
```

Example 19-23. timeit2.py

```
from timeit import repeat
print(repeat('num = 5; num *= 2', number=1, repeat=3))
```

Example 19-24. time_lists.py

```
from timeit import timeit

def make_list_1():
    result = []
    for value in range(1000):
        result.append(value)
    return result

def make_list_2():
    result = [value for value in range(1000)]
    return result

print('make_list_1 takes', timeit(make_list_1, number=1000), 'seconds')
print('make_list_2 takes', timeit(make_list_2, number=1000), 'seconds')
```

Example 19-25. test.py

```
print('Oops')
```

Example 19-26. test.py, revised

```
print('Ops!')
```

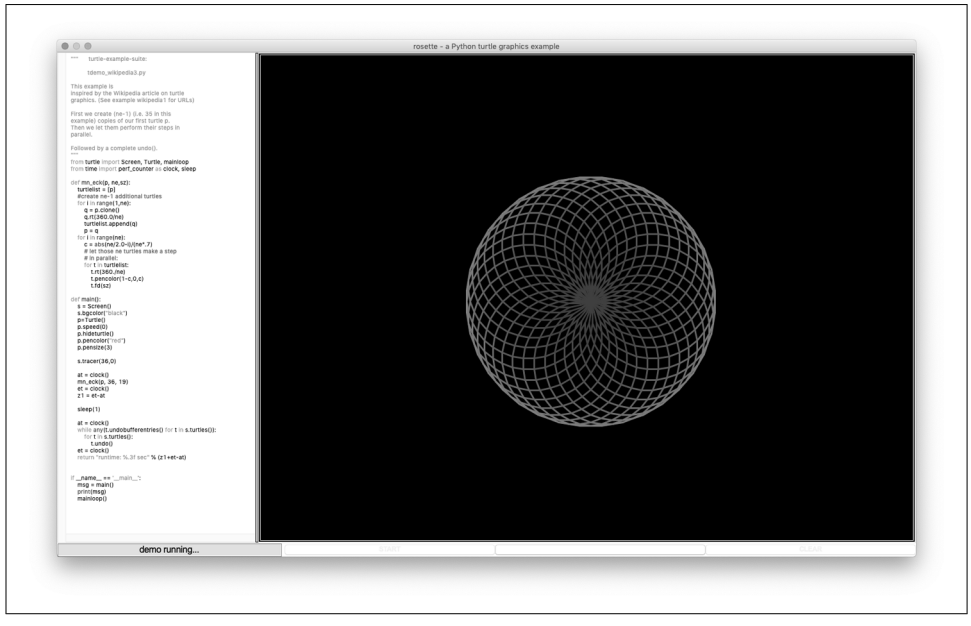

Figure 20-1. Image from turtledemo

Figure 20-2. Image displayed with the Python Image Library

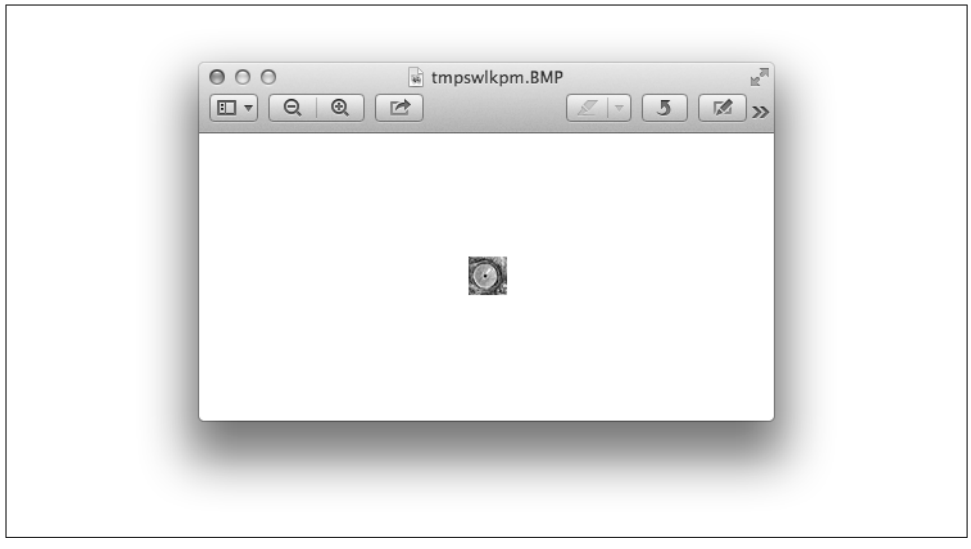

Figure 20-3. The cropped image

Figure 20-4. Beloved ur-critter

Figure 20-5. Alien technology

Example 20-1. ch20_critter.py

```
from PIL import Image

critter = Image.open('ch20_critter.png')
stache = Image.open('ch20_stache.png')
stache.putalpha(100)
img = Image.new('RGBA', critter.size, (255, 255, 255, 0))
img.paste(critter, (0, 0))
img.paste(stache, (45, 90), mask=stache)
img.show()
```

Figure 20-6. Our new, dapper mascot

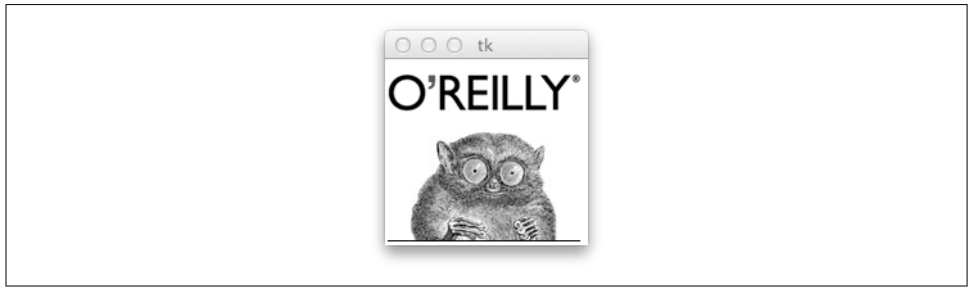

Figure 20-7. Image displayed with Tkinter

Figure 20-8. Image displayed with Matplotlib

Example 20-2. ch20_matplotlib.py

```
import matplotlib.pyplot as plt
from random import randint

linear = list(range(1, 21))
wiggly = list(num + randint(-1, 1) for num in linear)

fig, plots = plt.subplots(nrows=1, ncols=3)

ticks = list(range(0, 21, 5))
for plot in plots:
    plot.set_xticks(ticks)
    plot.set_yticks(ticks)

plots[0].scatter(linear, wiggly)
plots[1].plot(linear, wiggly)
plots[2].plot(linear, wiggly, 'o-')

plt.show()
```

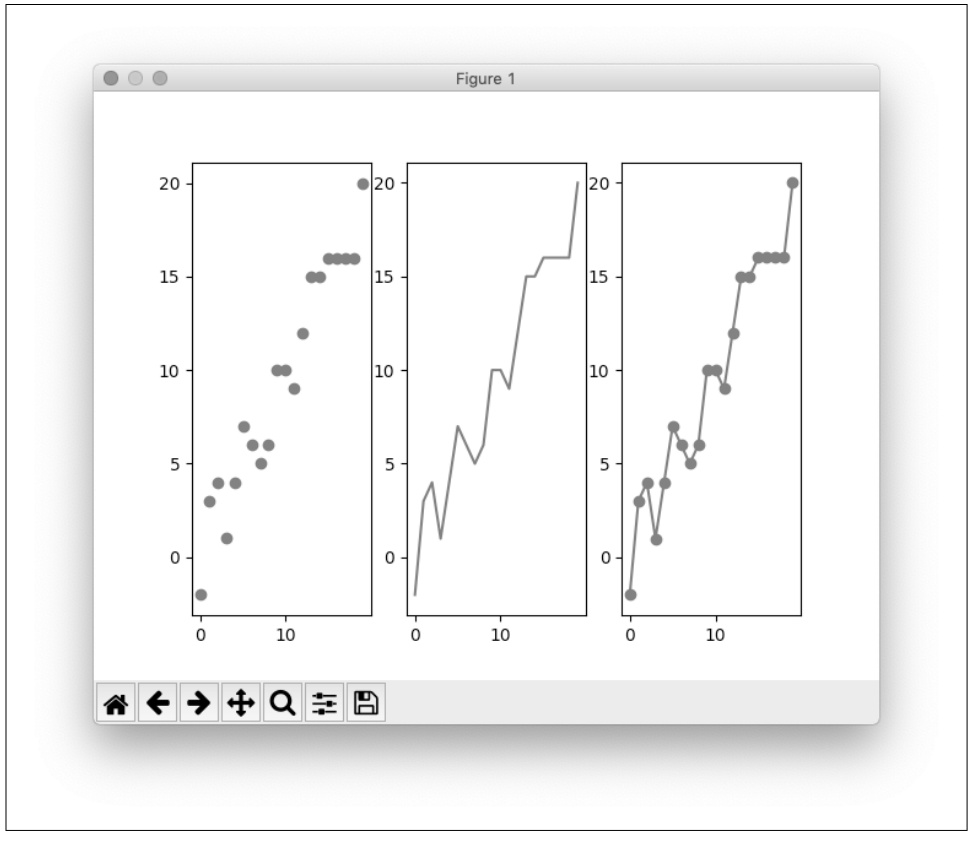

Figure 20-9. Basic Matplotlib scatter and line plots

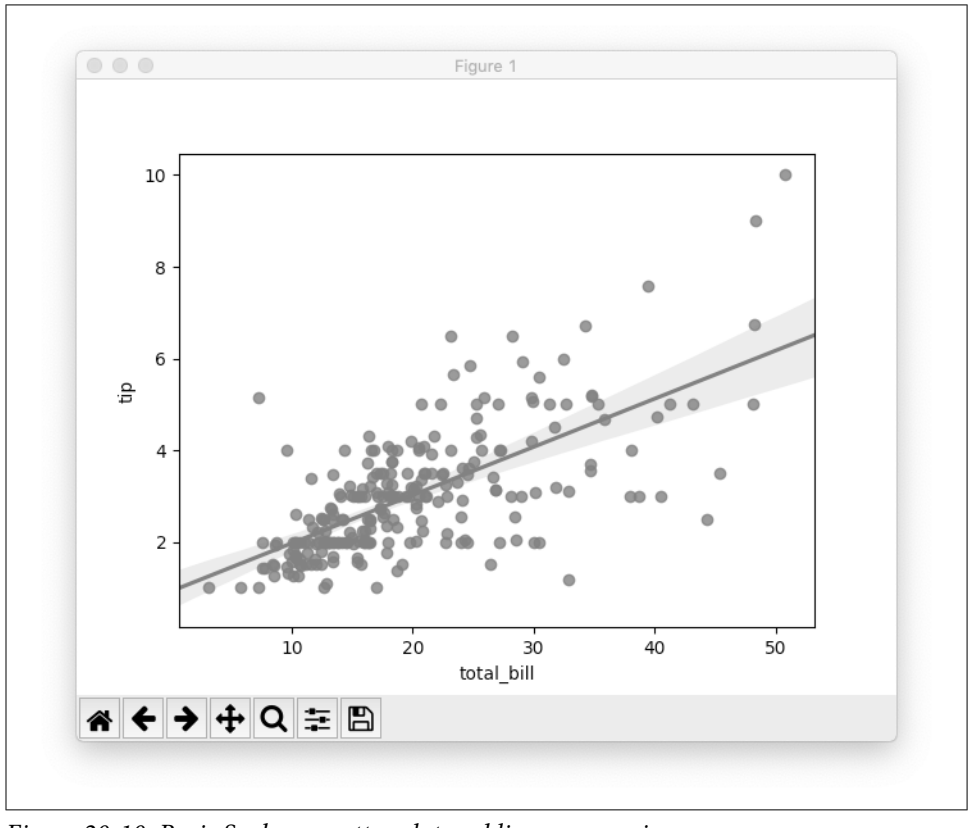

Figure 20-10. Basic Seaborn scatter plot and linear regression

Example 20-3. ch20_seaborn.py

```
import seaborn as sns
import matplotlib.pyplot as plt

tips = sns.load_dataset("tips")
sns.regplot(x="total_bill", y="tip", data=tips);
plt.show()
```

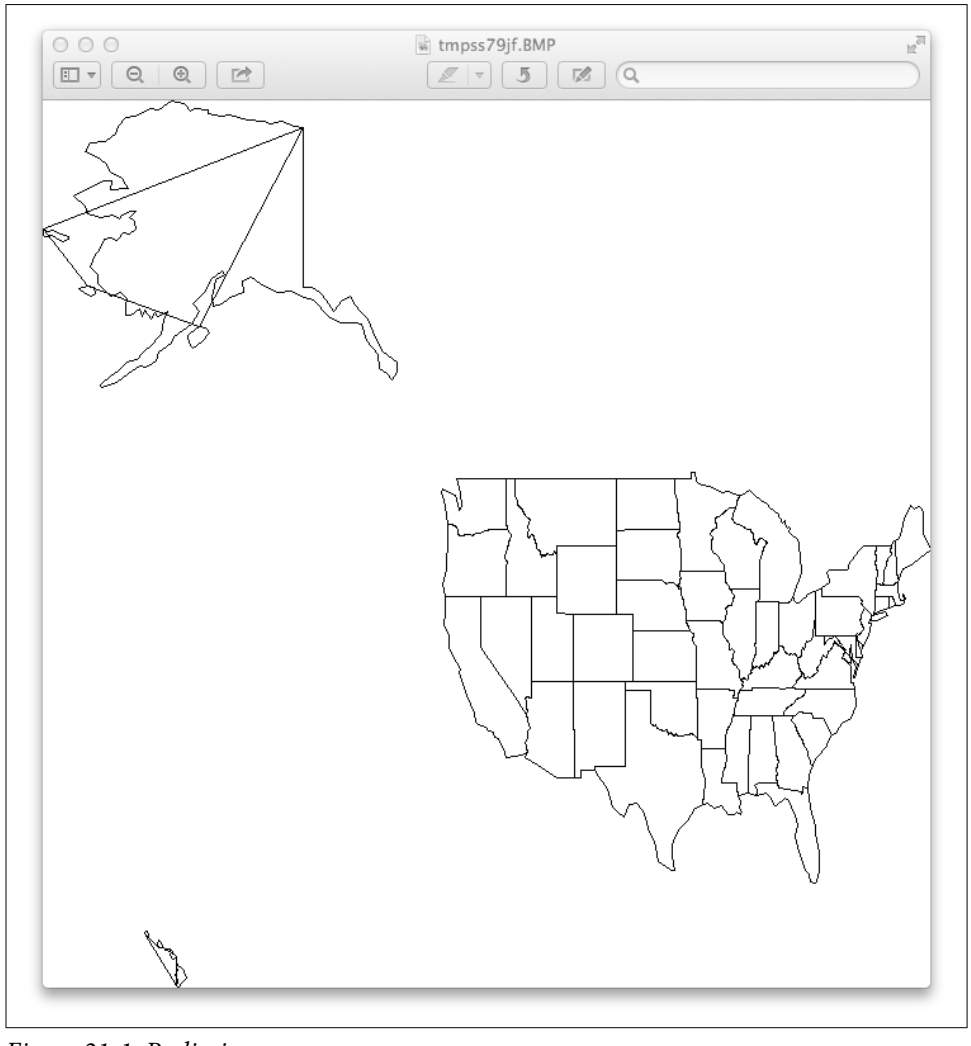

Figure 21-1. Preliminary map

Example 21-1. geopandas.py

```
import geopandas
import matplotlib.pyplot as plt

world_file = geopandas.datasets.get_path('naturalearth_lowres')
world = geopandas.read_file(world_file)
cities_file = geopandas.datasets.get_path('naturalearth_cities')
cities = geopandas.read_file(cities_file)
base = world.plot(color='orchid')
cities.plot(ax=base, color='black', markersize=2)
plt.show()
```

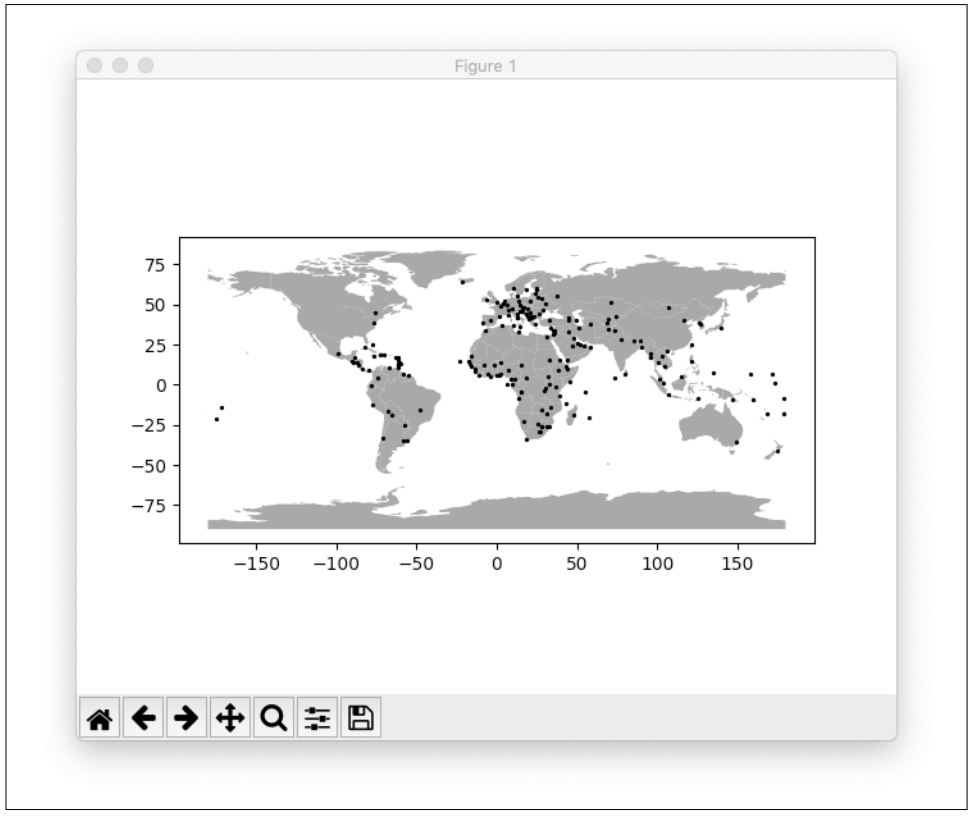

Figure 21-2. Geopandas map

Table A-1. Relative access times

Location	Time	Ratio	Relative Time	Relative Distance
CPU	0.4 ns	1	1 sec	1 in
L1 cache	0.9 ns	2	2 sec	2 in
L2 cache	2.8 ns	7	7 sec	7 in
L3 cache	28 ns	70	1 min	6 ft
RAM	100 ns	250	4 min	20 ft
SSD	100 µs	250,000	3 days	4 miles
Mag disk	10 ms	25,000,000	9 months	400 miles
Internet: SF→NY	65 ms	162,500,000	5 years	2,500 miles

Table A-2. Pets versus livestock

Pets	Livestock
Individually named	Automatically numbered
Customized care	Standardized
Nurse back to health	Replace

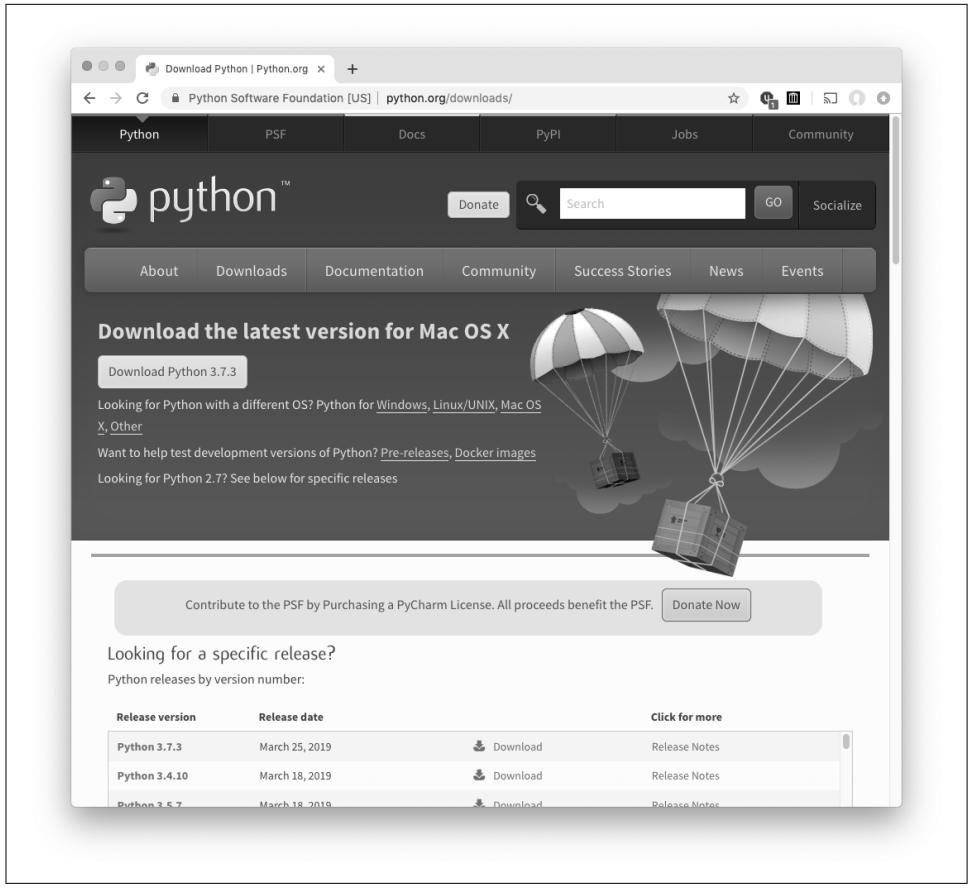

Figure B-1. Sample download page

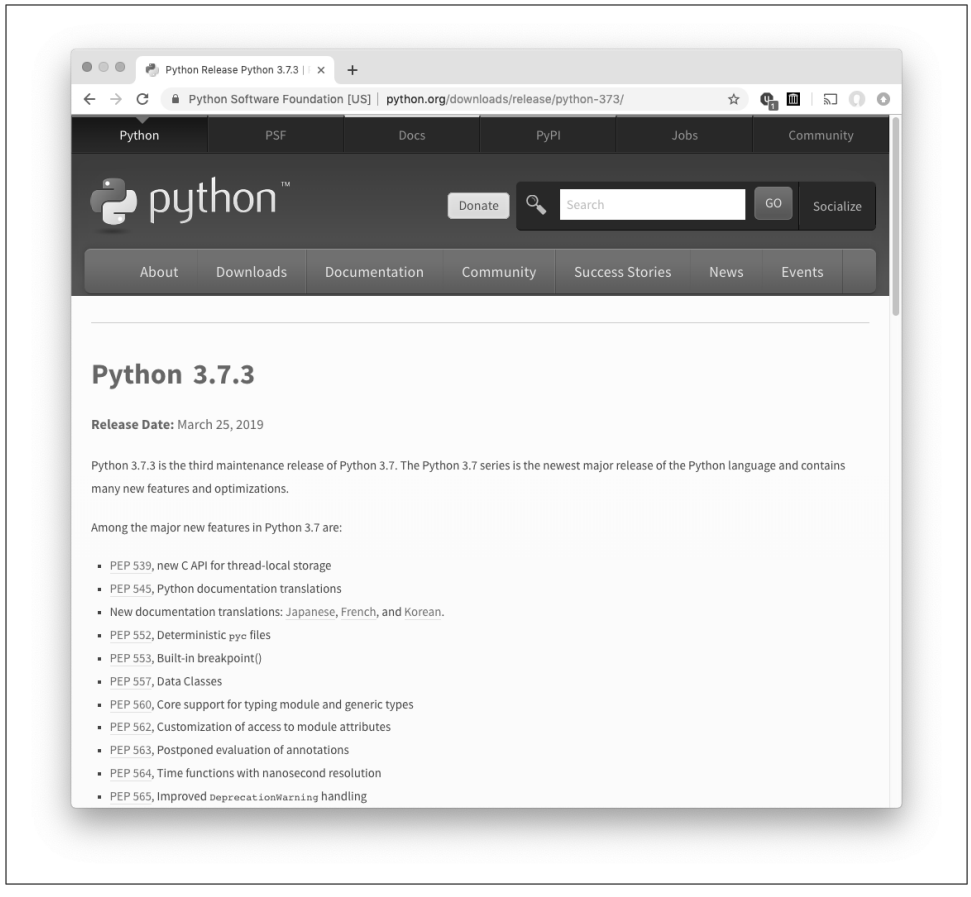

Figure B-2. Detail page for download

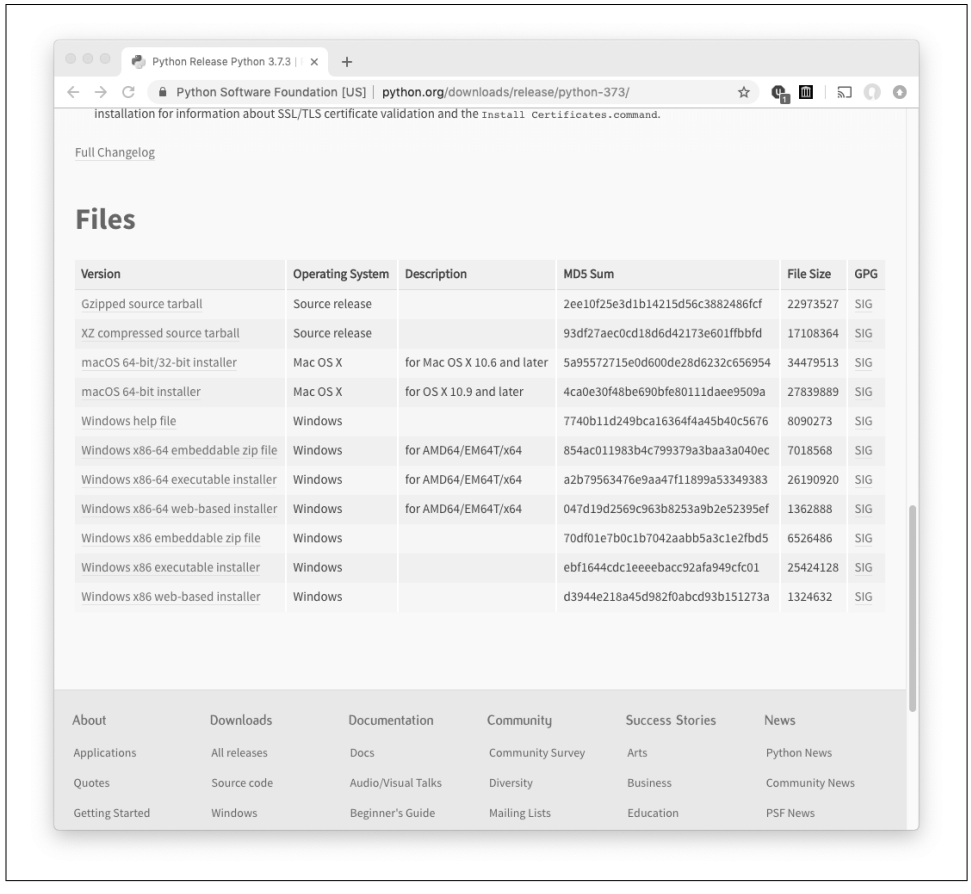

Figure B-3. Bottom of page offering downloads

Figure B-4. Mac install dialog 1

Figure B-5. Mac install dialog 9

Example C-1. trio_asks_sites.py

```
import time

import asks
import trio

asks.init("trio")

urls = [
    'https://boredomtherapy.com/bad-taxidermy/',
    'http://www.badtaxidermy.com/',
    'https://crappytaxidermy.com/',
    'https://www.ranker.com/list/bad-taxidermy-pictures/ashley-reign',
]

async def get_one(url, t1):
    r = await asks.get(url)
    t2 = time.time()
    print(f"{(t2-t1):.04}\t{len(r.content)}\t{url}")

async def get_sites(sites):
    t1 = time.time()
    async with trio.open_nursery() as nursery:
        for url in sites:
            nursery.start_soon(get_one, url, t1)

if __name__ == "__main__":
    print("seconds\tbytes\turl")
    trio.run(get_sites, urls)
```

APPENDIX D

Answers to Exercises

1. A Taste of Py

1.1 If you don't already have Python 3 installed on your computer, do it now. Read Appendix B for the details for your computer system.

1.2 Start the Python 3 interactive interpreter. Again, details are in Appendix B. It should print a few lines about itself and then a single line starting with `>>>`. That's your prompt to type Python commands.

Here's what it looks like on my Mac:

```
$ python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 16:39:00)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

1.3 Play with the interpreter a little. Use it like a calculator and type this: `8 * 9`. Press the Enter key to see the result. Python should print `72`.

```
>>> 8 * 9
72
```

1.4 Type the number `47` and press the Enter key. Did it print 47 for you on the next line?

```
>>> 47
47
```

1.5 Now type `print(47)` and press Enter. Did that also print 47 for you on the next line?

```
>>> print(47)
47
```

2. Data: Types, Values, Variables, and Names

2.1 Assign the integer value `99` to the variable `prince`, and print it.

```
>>> prince = 99
>>> print(prince)
99
>>>
```

2.2 What type is the value `5`?

```
>>> type(5)
<class 'int'>
```

2.3 What type is the value `2.0`?

```
>>> type(2.0)
<class 'float'>
```

2.4 What type is the expression `5 + 2.0`?

```
>>> type(5 + 2.0)
<class 'float'>
```

3. Numbers

3.1 How many seconds are in an hour? Use the interactive interpreter as a calculator and multiply the number of seconds in a minute (`60`) by the number of minutes in an hour (also `60`).

```
>>> 60 * 60
3600
```

3.2 Assign the result from the previous task (seconds in an hour) to a variable called `seconds_per_hour`.

```
>>> seconds_per_hour = 60 * 60
>>> seconds_per_hour
3600
```

3.3 How many seconds are in a day? Use your `seconds_per_hour` variable.

```
>>> seconds_per_hour * 24
86400
```

3.4 Calculate seconds per day again, but this time save the result in a variable called `seconds_per_day`.

```
>>> seconds_per_day = seconds_per_hour * 24
>>> seconds_per_day
86400
```

3.5 Divide `seconds_per_day` by `seconds_per_hour`. Use floating-point (`/`) division.

```
>>> seconds_per_day / seconds_per_hour
24.0
```

3.6 Divide `seconds_per_day` by `seconds_per_hour`, using integer (`//`) division. Did this number agree with the floating-point value from the previous question, aside from the final `.0`?

```
>>> seconds_per_day // seconds_per_hour
24
```

4. Choose with if

4.1 Choose a number between 1 and 10 and assign it to the variable `secret`. Then, select another number between 1 and 10 and assign it to the variable `guess`. Next, write the conditional tests (`if`, `else`, and `elif`) to print the string `'too low'` if `guess` is less than `secret`, `'too high'` if greater than `secret`, and `'just right'` if equal to `secret`.

Did you choose `7` for `secret`? I bet a lot of people do, because there's something about `7`.

```
secret = 7
guess = 5
if guess < secret:
    print('too low')
elif guess > secret:
    print('too high')
else:
    print('just right')
```

Run this program and you should see the following:

```
too low
```

4.2 Assign `True` or `False` to the variables `small` and `green`. Write some `if/else` statements to print which of these matches those choices: cherry, pea, watermelon, pumpkin.

```
>>> small = False
>>> green = True
>>> if small:
...     if green:
...         print("pea")
...     else:
...         print("cherry")
... else:
...     if green:
...         print("watermelon")
...     else:
...         print("pumpkin")
...
watermelon
```

5. Text Strings

5.1 Capitalize the word starting with `m`:

```
>>> song = """When an eel grabs your arm,
... And it causes great harm,
... That's - a moray!"""
```

Don't forget the space before the `m`:

```
>>> song = """When an eel grabs your arm,
... And it causes great harm,
... That's - a moray!"""
>>> song = song.replace(" m", " M")
>>> print(song)
When an eel grabs your arm,
And it causes great harm,
That's - a Moray!
```

5.2 Print each list question with its correctly matching answer, in the form:

Q: *question*
A: *answer*

```
>>> questions = [
...     "We don't serve strings around here. Are you a string?",
...     "What is said on Father's Day in the forest?",
```

```
...     "What makes the sound 'Sis! Boom! Bah!'?"
...     ]
>>> answers = [
...     "An exploding sheep.",
...     "No, I'm a frayed knot.",
...     "'Pop!' goes the weasel."
...     ]
```

You could print each item in `questions` with its mate from `answers` in many ways. Let's try a tuple sandwich (tuples in a tuple) to pair them, and tuple unpacking to retrieve them for printing:

```
questions = [
    "We don't serve strings around here. Are you a string?",
    "What is said on Father's Day in the forest?",
    "What makes the sound 'Sis! Boom! Bah!'?"
    ]
answers = [
    "An exploding sheep.",
    "No, I'm a frayed knot.",
    "'Pop!' goes the weasel."
    ]

q_a = ( (0, 1), (1,2), (2, 0) )
for q, a in q_a:
    print("Q:", questions[q])
    print("A:", answers[a])
    print()
```

Output:

```
$ python qanda.py
Q: We don't serve strings around here. Are you a string?
A: No, I'm a frayed knot.

Q: What is said on Father's Day in the forest?
A: 'Pop!' goes the weasel.

Q: What makes the sound 'Sis! Boom! Bah!'?
A: An exploding sheep.
```

5.3 Write the following poem by using old-style formatting. Substitute the strings `'roast beef'`, `'ham'`, `'head'`, and `'clam'` into this string:

```
My kitty cat likes %s,
My kitty cat likes %s,
My kitty cat fell on his %s
And now thinks he's a %s.
```

```
>>> poem = '''
... My kitty cat likes %s,
... My kitty cat likes %s,
... My kitty cat fell on his %s
... And now thinks he's a %s.
... '''
>>> args = ('roast beef', 'ham', 'head', 'clam')
>>> print(poem % args)

My kitty cat likes roast beef,
My kitty cat likes ham,
My kitty cat fell on his head
And now thinks he's a clam.
```

5.4 Write a form letter by using new-style formatting. Save the following string as `letter` (you'll use it in the next exercise):

```
Dear {salutation} {name},

Thank you for your letter. We are sorry that our {product}
{verbed} in your {room}. Please note that it should never
be used in a {room}, especially near any {animals}.

Send us your receipt and {amount} for shipping and handling.
We will send you another {product} that, in our tests,
is {percent}% less likely to have {verbed}.

Thank you for your support.

Sincerely,
{spokesman}
{job_title}
```

```
>>> letter = '''
... Dear {salutation} {name},
...
... Thank you for your letter. We are sorry that our {product}
... {verbed} in your {room}. Please note that it should never
... be used in a {room}, especially near any {animals}.
...
... Send us your receipt and {amount} for shipping and handling.
... We will send you another {product} that, in our tests,
... is {percent}% less likely to have {verbed}.
...
... Thank you for your support.
...
... Sincerely,
```

```
... {spokesman}
... {job_title}
... '''
```

5.5 Assign values to variable strings named `'salutation'`, `'name'`, `'product'`, `'verbed'` (past tense verb), `'room'`, `'animals'`, `'percent'`, `'spokesman'`, and `'job_title'`. Print `letter` with these values, using `letter.format()`.

```
>>> print (
...     letter.format(salutation='Ambassador',
...                   name='Nibbler',
...                   product='pudding',
...                   verbed='evaporated',
...                   room='gazebo',
...                   animals='octothorpes',
...                   amount='$1.99',
...                   percent=88,
...                   spokesman='Shirley Iugeste',
...                   job_title='I Hate This Job')
...     )

Dear Ambassador Nibbler,

Thank you for your letter. We are sorry that our pudding
evaporated in your gazebo. Please note that it should never
be used in a gazebo, especially near any octothorpes.

Send us your receipt and $1.99 for shipping and handling.
We will send you another pudding that, in our tests,
is 88% less likely to have evaporated.

Thank you for your support.

Sincerely,
Shirley Iugeste
I Hate This Job
```

5.6 After public polls to name things, a pattern emerged: an English submarine (Boaty McBoatface), an Australian racehorse (Horsey McHorseface), and a Swedish train (Trainy McTrainface). Use `%` formatting to print the winning name at the state fair for a prize duck, gourd, and spitz.

Example D-1. mcnames1.py

```
names = ["duck", "gourd", "spitz"]
for name in names:
    cap_name = name.capitalize()
    print("%sy Mc%sface" % (cap_name, cap_name))
```

Output:

```
Ducky McDuckface
Gourdy McGourdface
Spitzy McSpitzface
```

5.7 Do the same, with `format()` formatting.

Example D-2. mcnames2.py

```
names = ["duck", "gourd", "spitz"]
for name in names:
    cap_name = name.capitalize()
    print("{}y Mc{}face".format(cap_name, cap_name))
```

5.8 Once more, with feeling, and *f strings*.

Example D-3. mcnames3.py

```
names = ["duck", "gourd", "spitz"]
for name in names:
    cap_name = name.capitalize()
    print(f"{cap_name}y Mc{cap_name}face")
```

6. Loop with while and for

6.1 Use a `for` loop to print the values of the list `[3, 2, 1, 0]`.

```
>>> for value in [3, 2, 1, 0]:
...     print(value)
...
3
2
1
0
```

6.2 Assign the value `7` to the variable `guess_me`, and the value `1` to the variable `number`. Write a `while` loop that compares `number` with `guess_me`. Print `'too low'` if `number` is less than `guess_me`. If `number` equals `guess_me`, print `'found it!'` and then exit the loop. If `number` is greater than `guess_me`, print `'oops'` and then exit the loop. Increment `number` at the end of the loop.

```
guess_me = 7
number = 1
while True:
    if number < guess_me:
        print('too low')
    elif number == guess_me:
        print('found it!')
        break
    elif number > guess_me:
        print('oops')
        break
    number += 1
```

If you did this right, you should see this:

```
too low
too low
too low
too low
too low
too low
found it!
```

Notice that the `elif start > guess_me:` line could have been a simple `else:`, because if `start` is not less than or equal to `guess_me`, it must be greater—at least in this universe.

6.3 Assign the value `5` to the variable `guess_me`. Use a `for` loop to iterate a variable called `number` over `range(10)`. If `number` is less than `guess_me`, print `'too low'`. If it equals `guess_me`, print `found it!` and then break out of the for loop. If `number` is greater than `guess_me`, print `'oops'` and then exit the loop.

```
>>> guess_me = 5
>>> for number in range(10):
...     if number < guess_me:
...         print("too low")
...     elif number == guess_me:
...         print("found it!")
...         break
...     else:
...         print("oops")
...         break
...
too low
too low
too low
too low
too low
found it!
```

7. Tuples and Lists

7.1 Create a list called `years_list`, starting with the year of your birth, and each year thereafter until the year of your fifth birthday. For example, if you were born in 1980, the list would be `years_list = [1980, 1981, 1982, 1983, 1984, 1985]`.

If you were born in 1980, you would type:

```
>>> years_list = [1980, 1981, 1982, 1983, 1984, 1985]
```

7.2 In which of these years was your third birthday? Remember, you were 0 years of age for your first year.

You want offset `3`. Thus, if you were born in 1980:

```
>>> years_list[3]
1983
```

7.3 In which year in `years_list` were you the oldest?

You want the last year, so use offset `-1`. You could also say `5` because you know this list has six items, but `-1` gets the last item from a list of any size. For a 1980-vintage person:

```
>>> years_list[-1]
1985
```

7.4 Make and print a list called `things` with these three strings as elements: `"mozzarella"`, `"cinderella"`, `"salmonella"`.

```
>>> things = ["mozzarella", "cinderella", "salmonella"]
>>> things
['mozzarella', 'cinderella', 'salmonella']
```

7.5 Capitalize the element in `things` that refers to a person and then print the list. Did it change the element in the list?

This capitalizes the word but doesn't change it in the list:

```
>>> things[1].capitalize()
'Cinderella'
>>> things
['mozzarella', 'cinderella', 'salmonella']
```

If you want to change it in the list, you need to assign it back:

```
>>> things[1] = things[1].capitalize()
>>> things
['mozzarella', 'Cinderella', 'salmonella']
```

7.6 Make the cheesy element of `things` all uppercase and then print the list.

```
>>> things[0] = things[0].upper()
>>> things
['MOZZARELLA', 'Cinderella', 'salmonella']
```

7.7 Delete the disease element, collect your Nobel Prize, and then print the list.

This would remove it by value:

```
>>> things.remove("salmonella")
>>> things
['MOZZARELLA', 'Cinderella']
```

Because it was last in the list, the following would have worked also:

```
>>> del things[-1]
```

And you could have deleted by offset from the beginning:

```
>>> del things[2]
```

7.8 Create a list called `surprise` with the elements `"Groucho"`, `"Chico"`, and `"Harpo"`.

```
>>> surprise = ['Groucho', 'Chico', 'Harpo']
>>> surprise
['Groucho', 'Chico', 'Harpo']
```

7.9 Lowercase the last element of the `surprise` list, reverse it, and then capitalize it.

```
>>> surprise[-1] = surprise[-1].lower()
>>> surprise[-1] = surprise[-1][::-1]
>>> surprise[-1].capitalize()
'Oprah'
```

7.10 Use a list comprehension to make a list called `even` of the even numbers in `range(10)`.

```
>>> even = [number for number in range(10) if number % 2 == 0]
>>> even
[0, 2, 4, 6, 8]
```

7.11 Let's create a jumprope rhyme maker. You'll print a series of two-line rhymes. Start with this program fragment:

```
start1 = ["fee", "fie", "foe"]
rhymes = [
    ("flop", "get a mop"),
    ("fope", "turn the rope"),
    ("fa", "get your ma"),
    ("fudge", "call the judge"),
    ("fat", "pet the cat"),
    ("fog", "walk the dog"),
    ("fun", "say we're done"),
    ]
start2 = "Someone better"
```

For each string pair (`first`, `second`) in `rhymes`:

For the first line:

- Print each string in `start1`, capitalized and followed by an exclamation point and a space.
- Print `first`, also capitalized and followed by an exclamation point.

For the second line:

- Print `start2` and a space.
- Print `second` and a period.

```
start1 = ["fee", "fie", "foe"]
rhymes = [
    ("flop", "get a mop"),
    ("fope", "turn the rope"),
    ("fa", "get your ma"),
    ("fudge", "call the judge"),
    ("fat", "pet the cat"),
    ("fog", "pet the dog"),
    ("fun", "say we're done"),
    ]
start2 = "Someone better"
start1_caps = " ".join([word.capitalize() + "!" for word in start1])
for first, second in rhymes:
    print(f"{start1_caps} {first.capitalize()}!")
    print(f"{start2} {second}.")
```

Output:

```
Fee! Fie! Foe! Flop!
Someone better get a mop.
```

```
Fee! Fie! Foe! Fope!
Someone better turn the rope.
Fee! Fie! Foe! Fa!
Someone better get your ma.
Fee! Fie! Foe! Fudge!
Someone better call the judge.
Fee! Fie! Foe! Fat!
Someone better pet the cat.
Fee! Fie! Foe! Fog!
Someone better walk the dog.
Fee! Fie! Foe! Fun!
Someone better say we're done.
```

8. Dictionaries

8.1 Make an English-to-French dictionary called `e2f` and print it. Here are your starter words: `dog` is `chien`, `cat` is `chat`, and `walrus` is `morse`.

```
>>> e2f = {'dog': 'chien', 'cat': 'chat', 'walrus': 'morse'}
>>> e2f
{'cat': 'chat', 'walrus': 'morse', 'dog': 'chien'}
```

8.2 Using your three-word dictionary `e2f`, print the French word for `walrus`.

```
>>> e2f['walrus']
'morse'
```

8.3 Make a French-to-English dictionary called `f2e` from `e2f`. Use the `items` method.

```
>>> f2e = {}
>>> for english, french in e2f.items():
    f2e[french] = english
>>> f2e
{'morse': 'walrus', 'chien': 'dog', 'chat': 'cat'}
```

8.4 Print the English equivalent of the French word `chien`.

```
>>> f2e['chien']
'dog'
```

8.5 Print the set of English words from `e2f`.

```
>>> set(e2f.keys())
{'cat', 'walrus', 'dog'}
```

8.6 Make a multilevel dictionary called `life`. Use these strings for the topmost keys: `'animals'`, `'plants'`, and `'other'`. Make the `'animals'` key refer to another dictionary with the keys `'cats'`, `'octopi'`, and `'emus'`. Make the `'cats'` key refer to a list of strings with the values `'Henri'`, `'Grumpy'`, and `'Lucy'`. Make all the other keys refer to empty dictionaries.

This is a hard one, so don't feel bad if you peeked here first.

```
>>> life = {
...     'animals': {
...         'cats': [
...             'Henri', 'Grumpy', 'Lucy'
...             ],
...         'octopi': {},
...         'emus': {}
...         },
...     'plants': {},
...     'other': {}
...     }
>>>
```

8.7 Print the top-level keys of `life`.

```
>>> print(life.keys())
dict_keys(['animals', 'other', 'plants'])
```

Python 3 includes that `dict_keys` stuff. To print them as a plain list, use this:

```
>>> print(list(life.keys()))
['animals', 'other', 'plants']
```

By the way, you can use spaces to make your code easier to read:

```
>>> print ( list ( life.keys() ) )
['animals', 'other', 'plants']
```

8.8 Print the keys for `life['animals']`.

```
>>> print(life['animals'].keys())
dict_keys(['cats', 'octopi', 'emus'])
```

8.9 Print the values for `life['animals']['cats']`.

```
>>> print(life['animals']['cats'])
['Henri', 'Grumpy', 'Lucy']
```

8.10 Use a dictionary comprehension to create the dictionary `squares`. Use `range(10)` to return the keys, and use the square of each key as its value.

```
>>> squares = {key: key*key for key in range(10)}
>>> squares
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

8.11 Use a set comprehension to create the set `odd` from the odd numbers in `range(10)`.

```
>>> odd = {number for number in range(10) if number % 2 == 1}
>>> odd
{1, 3, 5, 7, 9}
```

8.12 Use a generator comprehension to return the string `'Got '` and a number for the numbers in `range(10)`. Iterate through this by using a `for` loop.

```
>>> for thing in ('Got %s' % number for number in range(10)):
...     print(thing)
...
Got 0
Got 1
Got 2
Got 3
Got 4
Got 5
Got 6
Got 7
```

```
Got 8
Got 9
```

8.13 Use `zip()` to make a dictionary from the key tuple `('optimist', 'pessimist', 'troll')` and the values tuple `('The glass is half full', 'The glass is half empty', 'How did you get a glass?')`.

```
>>> keys = ('optimist', 'pessimist', 'troll')
>>> values = ('The glass is half full',
...     'The glass is half empty',
...     'How did you get a glass?')
>>> dict(zip(keys, values))
{'optimist': 'The glass is half full',
'pessimist': 'The glass is half empty',
'troll': 'How did you get a glass?'}
```

8.14 Use `zip()` to make a dictionary called `movies` that pairs these lists: `titles = ['Creature of Habit', 'Crewel Fate', 'Sharks On a Plane']` and `plots = ['A nun turns into a monster', 'A haunted yarn shop', 'Check your exits']`

```
>>> titles = ['Creature of Habit',
...     'Crewel Fate',
...     'Sharks On a Plane']
>>> plots = ['A nun turns into a monster',
...     'A haunted yarn shop',
...     'Check your exits']
>>> movies = dict(zip(titles, plots))
>>> movies
{'Creature of Habit': 'A nun turns into a monster',
'Crewel Fate': 'A haunted yarn shop',
'Sharks On a Plane': 'Check your exits'}
>>>
```

9. Functions

9.1 Define a function called `good()` that returns the following list: `['Harry', 'Ron', 'Hermione']`.

```
>>> def good():
...     return ['Harry', 'Ron', 'Hermione']
...
```

```
>>> good()
['Harry', 'Ron', 'Hermione']
```

9.2 Define a generator function called `get_odds()` that returns the odd numbers from `range(10)`. Use a `for` loop to find and print the third value returned.

```
>>> def get_odds():
...     for number in range(1, 10, 2):
...         yield number
...
>>> count = 1
>>> for number in get_odds():
...     if count == 3:
...         print("The third odd number is", number)
...         break
...     count += 1
...
The third odd number is 5
```

9.3 Define a decorator called `test` that prints `'start'` when a function is called, and `'end'` when it finishes.

```
>>> def test(func):
...     def new_func(*args, **kwargs):
...         print('start')
...         result = func(*args, **kwargs)
...         print('end')
...         return result
...     return new_func
...
>>>
>>> @test
... def greeting():
...     print("Greetings, Earthling")
...
>>> greeting()
start
Greetings, Earthling
end
```

9.4 Define an exception called `OopsException`. Raise this exception to see what happens. Then, write the code to catch this exception and print `'Caught an oops'`.

```
>>> class OopsException(Exception):
...     pass
...
>>> raise OopsException()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
__main__.OopsException
>>>
>>> try:
...     raise OopsException
... except OopsException:
...     print('Caught an oops')
...
Caught an oops
```

10. Oh Oh: Objects and Classes

10.1 Make a class called `Thing` with no contents and print it. Then, create an object called `example` from this class and also print it. Are the printed values the same or different?

```
>>> class Thing:
...     pass
...
>>> print(Thing)
<class '__main__.Thing'>
>>> example = Thing()
>>> print(example)
<__main__.Thing object at 0x1006f3fd0>
```

10.2 Make a new class called `Thing2` and assign the value `'abc'` to a class variable called `letters`. Print `letters`.

```
>>> class Thing2:
...     letters = 'abc'
...
>>> print(Thing2.letters)
abc
```

10.3 Make yet another class called (of course) `Thing3`. This time, assign the value `'xyz'` to an instance (object) variable called `letters`. Print `letters`. Do you need to make an object from the class to do this?

```
>>> class Thing3:
...     def __init__(self):
...         self.letters = 'xyz'
...
```

The variable `letters` belongs to any objects made from `Thing3`, not the `Thing3` class itself:

```
>>> print(Thing3.letters)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: type object 'Thing3' has no attribute 'letters'
>>> something = Thing3()
>>> print(something.letters)
xyz
```

10.4 Make a class called `Element`, with instance attributes `name`, `symbol`, and `number`. Create an object called `hydrogen` of this class with the values `'Hydrogen'`, `'H'`, and `1`.

```
>>> class Element:
...     def __init__(self, name, symbol, number):
...         self.name = name
...         self.symbol = symbol
...         self.number = number
...
>>> hydrogen = Element('Hydrogen', 'H', 1)
```

10.5 Make a dictionary with these keys and values: `'name': 'Hydrogen', 'symbol': 'H', 'number': 1`. Then, create an object called `hydrogen` from class `Element` using this dictionary.

Start with the dictionary:

```
>>> el_dict = {'name': 'Hydrogen', 'symbol': 'H', 'number': 1}
```

This works, although it takes a bit of typing:

```
>>> hydrogen = Element(el_dict['name'], el_dict['symbol'], el_dict['number'])
```

Let's check that it worked:

```
>>> hydrogen.name
'Hydrogen'
```

However, you can also initialize the object directly from the dictionary, because its key names match the arguments to `__init__` (refer to Chapter 9 for a discussion of keyword arguments):

```
>>> hydrogen = Element(**el_dict)
>>> hydrogen.name
'Hydrogen'
```

10.6 For the `Element` class, define a method called `dump()` that prints the values of the object's attributes (`name`, `symbol`, and `number`). Create the `hydrogen` object from this new definition and use `dump()` to print its attributes.

```
>>> class Element:
...     def __init__(self, name, symbol, number):
...         self.name = name
...         self.symbol = symbol
...         self.number = number
...     def dump(self):
...         print('name=%s, symbol=%s, number=%s' %
...             (self.name, self.symbol, self.number))
...
>>> hydrogen = Element(**el_dict)
>>> hydrogen.dump()
name=Hydrogen, symbol=H, number=1
```

10.7 Call `print(hydrogen)`. In the definition of `Element`, change the name of the method `dump` to `__str__`, create a new `hydrogen` object, and call `print(hydrogen)` again.

```
>>> print(hydrogen)
<__main__.Element object at 0x1006f5310>
>>> class Element:
...     def __init__(self, name, symbol, number):
...         self.name = name
...         self.symbol = symbol
...         self.number = number
...     def __str__(self):
...         return ('name=%s, symbol=%s, number=%s' %
...             (self.name, self.symbol, self.number))
...
>>> hydrogen = Element(**el_dict)
>>> print(hydrogen)
name=Hydrogen, symbol=H, number=1
```

`__str__()` is one of Python's *magic methods*. The `print` function calls an object's `__str__()` method to get its string representation. If it doesn't have a `__str__()` method, it gets the default method from its parent `Object` class, which returns a string like `<__main__.Element object at 0x1006f5310>`.

10.8 Modify `Element` to make the attributes `name`, `symbol`, and `number` private. Define a getter property for each to return its value.

```
>>> class Element:
...     def __init__(self, name, symbol, number):
...         self.__name = name
...         self.__symbol = symbol
...         self.__number = number
...     @property
...     def name(self):
...         return self.__name
...     @property
...     def symbol(self):
...         return self.__symbol
...     @property
...     def number(self):
...         return self.__number
...
>>> hydrogen = Element('Hydrogen', 'H', 1)
>>> hydrogen.name
'Hydrogen'
>>> hydrogen.symbol
'H'
>>> hydrogen.number
1
```

10.9 Define three classes: `Bear`, `Rabbit`, and `Octothorpe`. For each, define only one method: `eats()`. This should return `'berries'` (`Bear`), `'clover'` (`Rabbit`), and `'campers'` (`Octothorpe`). Create one object from each and print what it eats.

```
>> class Bear:
...     def eats(self):
...         return 'berries'
...
>>> class Rabbit:
...     def eats(self):
...         return 'clover'
...
>>> class Octothorpe:
...     def eats(self):
```

```
...         return 'campers'
...
>>> b = Bear()
>>> r = Rabbit()
>>> o = Octothorpe()
>>> print(b.eats())
berries
>>> print(r.eats())
clover
>>> print(o.eats())
campers
```

10.10 Define these classes: `Laser`, `Claw`, and `SmartPhone`. Each has only one method: `does()`. This returns `'disintegrate'` (`Laser`), `'crush'` (`Claw`), or `'ring'` (`SmartPhone`). Then, define the class `Robot` that has one instance (object) of each of these. Define a `does()` method for the `Robot` that prints what its component objects do.

```
>>> class Laser:
...     def does(self):
...         return 'disintegrate'
...
>>> class Claw:
...     def does(self):
...         return 'crush'
...
>>> class SmartPhone:
...     def does(self):
...         return 'ring'
...
>>> class Robot:
...     def __init__(self):
...         self.laser = Laser()
...         self.claw = Claw()
...         self.smartphone = SmartPhone()
...     def does(self):
...         return '''I have many attachments:
... My laser, to %s.
... My claw, to %s.
... My smartphone, to %s.''' % (
...     self.laser.does(),
...     self.claw.does(),
...     self.smartphone.does() )
...
>>> robbie = Robot()
>>> print( robbie.does() )
I have many attachments:
My laser, to disintegrate.
```

```
My claw, to crush.
My smartphone, to ring.
```

11. Modules, Packages, and Goodies

11.1 Make a file called *zoo.py*. In it, define a function called `hours` that prints the string `'Open 9-5 daily'`. Then, use the interactive interpreter to import the `zoo` module and call its `hours` function.

Here's *zoo.py*:

```
def hours():
    print('Open 9-5 daily')
```

And now, let's import it interactively:

```
>>> import zoo
>>> zoo.hours()
Open 9-5 daily
```

11.2 In the interactive interpreter, import the `zoo` module as `menagerie` and call its `hours()` function.

```
>>> import zoo as menagerie
>>> menagerie.hours()
Open 9-5 daily
```

11.3 Staying in the interpreter, import the `hours()` function from `zoo` directly and call it.

```
>>> from zoo import hours
>>> hours()
Open 9-5 daily
```

11.4 Import the `hours()` function as `info` and call it.

```
>>> from zoo import hours as info
>>> info()
Open 9-5 daily
```

11.6 Make an `OrderedDict` called `fancy` from the same pairs and print it. Did it print in the same order as `plain`?

```
>>> from collections import OrderedDict
>>> fancy = OrderedDict([('a', 1), ('b', 2), ('c', 3)])
>>> fancy
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```

11.7 Make a `defaultdict` called `dict_of_lists` and pass it the argument `list`. Make the list `dict_of_lists['a']` and append the value `'something for a'` to it in one assignment. Print `dict_of_lists['a']`.

```
>>> from collections import defaultdict
>>> dict_of_lists = defaultdict(list)
>>> dict_of_lists['a'].append('something for a')
>>> dict_of_lists['a']
['something for a']
```

12. Wrangle and Mangle Data

12.1 Create a Unicode string called `mystery` and assign it the value `'\U0001f984'`. Print `mystery` and its Unicode name.

```
>>> import unicodedata
>>> mystery = '\U0001f4a9'
>>> mystery
'💩'
>>> unicodedata.name(mystery)
'PILE OF POO'
```

Oh my. What else have they got in there?

12.2 Encode `mystery`, this time using UTF-8, into the `bytes` variable `popbytes`. Print `pop_bytes`.

```
>>> pop_bytes = mystery.encode('utf-8')
>>> pop_bytes
b'\xf0\x9f\x92\xa9'
```

12.3 Using UTF-8, decode `popbytes` into the string variable `pop_string`. Print `pop_string`. Is `pop_string` equal to `mystery`?

```
>>> pop_string = pop_bytes.decode('utf-8')
>>> pop_string
'☕'
>>> pop_string == mystery
True
```

12.4 When you're working with text, regular expressions come in very handy. We'll apply them in a number of ways to our featured text sample. It's a poem titled "Ode on the Mammoth Cheese," written by James McIntyre in 1866 in homage to a seven-thousand-pound cheese that was crafted in Ontario and sent on an international tour. If you'd rather not type all of it, use your favorite search engine and cut and paste the words into your Python program, or just grab it from Project Gutenberg (*http://bit.ly/mcintyre-poetry*). Call the text string `mammoth`.

```
>>> mammoth = '''
... We have seen thee, queen of cheese,
... Lying quietly at your ease,
... Gently fanned by evening breeze,
... Thy fair form no flies dare seize.
...
... All gaily dressed soon you'll go
... To the great Provincial show,
... To be admired by many a beau
... In the city of Toronto.
...
... Cows numerous as a swarm of bees,
... Or as the leaves upon the trees,
... It did require to make thee please,
... And stand unrivalled, queen of cheese.
...
... May you not receive a scar as
... We have heard that Mr. Harris
... Intends to send you off as far as
... The great world's show at Paris.
...
... Of the youth beware of these,
... For some of them might rudely squeeze
... And bite your cheek, then songs or glees
... We could not sing, oh! queen of cheese.
...
... We'rt thou suspended from balloon,
... You'd cast a shade even at noon,
... Folks would think it was the moon
```

```
... About to fall and crush them soon.
... '''
```

12.5 Import the `re` module to use Python's regular expression functions. Use the `re.findall()` to print all the words that begin with `c`.

We'll define the variable `pat` for the pattern and then search for it in `mammoth`:

```
>>> import re
>>> pat = r'\bc\w*'
>>> re.findall(pat, mammoth)
['cheese', 'city', 'cheese', 'cheek', 'could', 'cheese', 'cast', 'crush']
```

The `\b` means to begin at a boundary between a word and a nonword. Use this to specify either the beginning or end of a word. The literal `c` is the first letter of the words we're looking for. The `\w` means any *word character*, which includes letters, digits, and underscores (_). The `*` means *zero or more* of these word characters. Together, this finds words that begin with `c`, including `'c'` itself. If you didn't use a raw string (with an `r` right before the starting quote), Python would interpret `\b` as a backspace and the search would mysteriously fail:

```
>>> pat = '\bc\w*'
>>> re.findall(pat, mammoth)
[]
```

12.6 Find all four-letter words that begin with `c`.

```
>>> pat = r'\bc\w{3}\b'
>>> re.findall(pat, mammoth)
['city', 'cast']
```

You need that final `\b` to indicate the end of the word. Otherwise, you'll get the first four letters of all words that begin with `c` and have at least four letters:

```
>>> pat = r'\bc\w{3}'
>>> re.findall(pat, mammoth)
['chee', 'city', 'chee', 'chee', 'coul', 'chee', 'cast', 'crus']
```

12.7 Find all the words that end with `r`.

This is a little tricky. We get the right result for words that end with `r`:

```
>>> pat = r'\b\w*r\b'
>>> re.findall(pat, mammoth)
['your', 'fair', 'Or', 'scar', 'Mr', 'far', 'For', 'your', 'or']
```

However, the results aren't so good for words that end with `l`:

```
>>> pat = r'\b\w*l\b'
>>> re.findall(pat, mammoth)
['All', 'll', 'Provincial', 'fall']
```

But what's that `ll` doing there? The `\w` pattern matches only letters, numbers, and underscores—not ASCII apostrophes. As a result, it grabs the final `ll` from `you'll`. We can handle this by adding an apostrophe to the set of characters to match. Our first try fails:

```
>>> pat = r'\b[\w']*l\b'
  File "<stdin>", line 1
    pat = r'\b[\w']*l\b'
```

Python points to the vicinity of the error, but it might take a while to see that the mistake was that the pattern string is surrounded by the same apostrophe/quote character. One way to solve this is to escape it with a backslash:

```
>>> pat = r'\b[\w\']*l\b'
>>> re.findall(pat, mammoth)
['All', "you'll", 'Provincial', 'fall']
```

Another way is to surround the pattern string with double quotes:

```
>>> pat = r"\b[\w']*l\b"
>>> re.findall(pat, mammoth)
['All', "you'll", 'Provincial', 'fall']
```

12.8 Find all the words that contain exactly three vowels in a row.

Begin with a word boundary, any number of *word* characters, three vowels, and then any nonvowel characters to the end of the word:

```
>>> pat = r'\b[^aeiou]*[aeiou]{3}[^aeiou]*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'beau\nIn', 'queen', 'squeeze', 'queen']
```

This looks right, except for that `'beau\nIn'` string. We searched `mammoth` as a single multiline string. Our `[^aeiou]` matches any nonvowels, including `\n` (line feed, which marks the end of a text line). We need to add one more thing to the ignore set: `\s` matches any space characters, including `\n`:

```
>>> pat = r'\b\w*[aeiou]{3}[^aeiou\s]\w*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'queen', 'squeeze', 'queen']
```

We didn't find `beau` this time, so we need one more tweak to the pattern: match any number (even zero) of nonvowels after the three vowels. Our previous pattern always matched one nonvowel.

```
>>> pat = r'\b\w*[aeiou]{3}[^aeiou\s]*\w*\b'
>>> re.findall(pat, mammoth)
['queen', 'quietly', 'beau', 'queen', 'squeeze', 'queen']
```

What does all of this show? Among other things, that regular expressions can do a lot, but they can be very tricky to get right.

12.9 Use `unhexlify()` to convert this hex string (combined from two strings to fit on a page) to a `bytes` variable called `gif`:

```
'47494638396101000100800000000000ffffff21f9' +
'0401000000002c000000000100010000020144003b'
```

```
>>> import binascii
>>> hex_str = '47494638396101000100800000000000ffffff21f9' + \
...     '0401000000002c000000000100010000020144003b'
>>> gif = binascii.unhexlify(hex_str)
>>> len(gif)
42
```

12.10 The bytes in `gif` define a one-pixel transparent GIF file, one of the most common graphics file formats. A legal GIF starts with the string *GIF89a*. Does `gif` match this?

```
>>> gif[:6] == b'GIF89a'
True
```

Notice that we needed to use a `b` to define a byte string rather than a Unicode character string. You can compare bytes with bytes, but you cannot compare bytes with strings:

```
>>> gif[:6] == 'GIF89a'
False
>>> type(gif)
<class 'bytes'>
>>> type('GIF89a')
<class 'str'>
>>> type(b'GIF89a')
<class 'bytes'>
```

12.11 The pixel width of a GIF is a 16-bit little-endian integer starting at byte offset 6, and the height is the same size, starting at offset 8. Extract and print these values for `gif`. Are they both `1`?

```
>>> import struct
>>> width, height = struct.unpack('<HH', gif[6:10])
>>> width, height
(1, 1)
```

13. Calendars and Clocks

13.1 Write the current date as a string to the text file *today.txt*.

```
>>> from datetime import date
>>> now = date.today()
>>> now_str = now.isoformat()
>>> with open('today.txt', 'wt') as output:
...     print(now_str, file=output)
>>>
```

When I ran this, here's what I got in *today.txt*:

```
2019-07-23
```

Instead of `print`, you could have also said something like `output.write(now_str)`. Using `print` adds the final newline.

13.2 Read the text file *today.txt* into the string `today_string`.

```
>>> with open('today.txt', 'rt') as input:
...     today_string = input.read()
...
>>> today_string
'2019-07-23\n'
```

13.3 Parse the date from `today_string`.

```
>>> fmt = '%Y-%m-%d\n'
>>> datetime.strptime(today_string, fmt)
datetime.datetime(2019, 7, 23, 0, 0)
```

If you wrote that final newline to the file, you need to match it in the format string.

13.4 Create a date object of your day of birth.

Let's say that you were born on August 14, 1982:

```
>>> my_day = date(1982, 8, 14)
>>> my_day
datetime.date(1982, 8, 14)
```

13.5 What day of the week was your day of birth?

```
>>> my_day.weekday()
5
>>> my_day.isoweekday()
6
```

With `weekday()`, Monday is 0 and Sunday is 6. With `isoweekday()`, Monday is 1 and Sunday is 7. Therefore, this date was a Saturday.

13.6 When will you be (or when were you) 10,000 days old?

```
>>> from datetime import timedelta
>>> party_day = my_day + timedelta(days=10000)
>>> party_day
datetime.date(2009, 12, 30)
```

If August 14, 1982 was your birthday, you probably missed an excuse for a party.

14. Files and Directories

14.1 List the files in your current directory.

If your current directory is *ohmy* and contains three files named after animals, it might look like this:

```
>>> import os
>>> os.listdir('.')
['bears', 'lions', 'tigers']
```

14.2 List the files in your parent directory.

If your parent directory contained two files plus the current *ohmy* directory, it might look like this:

```
>>> import os
>>> os.listdir('..')
['ohmy', 'paws', 'whiskers']
```

14.3 Assign the string `'This is a test of the emergency text system'` to the variable `test1`, nd write `test1` to a file called *test.txt*.

```
>>> test1 = 'This is a test of the emergency text system'
>>> len(test1)
43
```

Here's how to do it by using `open`, `write`, and `close`:

```
>>> outfile = open('test.txt', 'wt')
>>> outfile.write(test1)
43
>>> outfile.close()
```

Or, you can use `with` and avoid calling `close` (Python does it for you):

```
>>> with open('test.txt', 'wt') as outfile:
...     outfile.write(test1)
...
43
```

14.4 Open the file *test.txt* and read its contents into the string `test2`. Are `test1` and `test2` the same?

```
>>> with open('test.txt', 'rt') as infile:
...     test2 = infile.read()
...
>>> len(test2)
43
>>> test1 == test2
True
```

15. Data in Time: Processes and Concurrency

15.1 Use `multiprocessing` to create three separate processes. Make each one wait a random number of seconds between zero and one, print the current time, and then exit.

```
import multiprocessing

def now(seconds):
    from datetime import datetime
    from time import sleep
    sleep(seconds)
    print('wait', seconds, 'seconds, time is', datetime.utcnow())

if __name__ == '__main__':
    import random
    for n in range(3):
        seconds = random.random()
        proc = multiprocessing.Process(target=now, args=(seconds,))
        proc.start()
```

```
$ python multi_times.py
wait 0.10720361113059229 seconds, time is 2019-07-24 00:19:23.951594
wait 0.5825144002370065 seconds, time is 2019-07-24 00:19:24.425047
wait 0.6647690569029477 seconds, time is 2019-07-24 00:19:24.509995
```

16. Data in a Box: Persistent Storage

16.1 Save the following text lines to a file called *books.csv* (notice that if the fields are separated by commas, you need to surround a field with quotes if it contains a comma):

```
author,book
J R R Tolkien,The Hobbit
Lynne Truss,"Eats, Shoots & Leaves"
```

```
>>> text = '''author,book
... J R R Tolkien,The Hobbit
... Lynne Truss,"Eats, Shoots & Leaves"
... '''
>>> with open('test.csv', 'wt') as outfile:
...     outfile.write(text)
...
73
```

16.2 Use the `csv` module and its `DictReader` method to read *books.csv* to the variable `books`. Print the values in `books`. Did `DictReader` handle the quotes and commas in the second book's title?

```
>>> with open('books.csv', 'rt') as infile:
...     books = csv.DictReader(infile)
...     for book in books:
...         print(book)
...
{'book': 'The Hobbit', 'author': 'J R R Tolkien'}
{'book': 'Eats, Shoots & Leaves', 'author': 'Lynne Truss'}
```

16.3 Create a CSV file called *books2.csv* by using these lines:

```
title,author,year
The Weirdstone of Brisingamen,Alan Garner,1960
Perdido Street Station,China Miéville,2000
Thud!,Terry Pratchett,2005
The Spellman Files,Lisa Lutz,2007
Small Gods,Terry Pratchett,1992
```

```
>>> text = '''title,author,year
... The Weirdstone of Brisingamen,Alan Garner,1960
... Perdido Street Station,China Miéville,2000
... Thud!,Terry Pratchett,2005
... The Spellman Files,Lisa Lutz,2007
... Small Gods,Terry Pratchett,1992
... '''
>>> with open('books2.csv', 'wt') as outfile:
...     outfile.write(text)
...
201
```

16.4 Use the `sqlite3` module to create a SQLite database called *books.db* and a table called `books` with these fields: `title` (text), `author` (text), and `year` (integer).

```
>>> import sqlite3
>>> db = sqlite3.connect('books.db')
>>> curs = db.cursor()
>>> curs.execute('''create table book (title text, author text, year int)''')
<sqlite3.Cursor object at 0x1006e3b90>
>>> db.commit()
```

16.5 Read *books2.csv* and insert its data into the book table.

```
>>> import csv
>>> import sqlite3
>>> ins_str = 'insert into book values(?, ?, ?)'
>>> with open('books.csv', 'rt') as infile:
...   books = csv.DictReader(infile)
...   for book in books:
...     curs.execute(ins_str, (book['title'], book['author'], book['year']))
...
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
<sqlite3.Cursor object at 0x1007b21f0>
>>> db.commit()
```

16.6 Select and print the `title` column from the book table in alphabetical order.

```
>>> sql = 'select title from book order by title asc'
>>> for row in db.execute(sql):
...     print(row)
...
('Perdido Street Station',)
('Small Gods',)
('The Spellman Files',)
('The Weirdstone of Brisingamen',)
('Thud!',)
```

If you just wanted to print the `title` value without that tuple stuff (parentheses and comma), try this:

```
>>> for row in db.execute(sql):
...     print(row[0])
...
Perdido Street Station
Small Gods
The Spellman Files
The Weirdstone of Brisingamen
Thud!
```

If you want to ignore the initial `'The'` in titles, you need a little extra SQL fairy dust:

```
>>> sql = '''select title from book order by
... case when (title like "The %")
... then substr(title, 5)
... else title end'''
>>> for row in db.execute(sql):
...     print(row[0])
```

```
...
Perdido Street Station
Small Gods
The Spellman Files
Thud!
The Weirdstone of Brisingamen
```

16.7 Select and print all columns from the book table in order of publication.

```
>>> for row in db.execute('select * from book order by year'):
...     print(row)
...
('The Weirdstone of Brisingamen', 'Alan Garner', 1960)
('Small Gods', 'Terry Pratchett', 1992)
('Perdido Street Station', 'China Miéville', 2000)
('Thud!', 'Terry Pratchett', 2005)
('The Spellman Files', 'Lisa Lutz', 2007)
```

To print all the fields in each row, just separate with a comma and space:

```
>>> for row in db.execute('select * from book order by year'):
...     print(*row, sep=', ')
...
The Weirdstone of Brisingamen, Alan Garner, 1960
Small Gods, Terry Pratchett, 1992
Perdido Street Station, China Miéville, 2000
Thud!, Terry Pratchett, 2005
The Spellman Files, Lisa Lutz, 2007
```

16.8 Use the `sqlalchemy` module to connect to the sqlite3 database *books.db* that you just made in exercise 8.6. As in 8.8, select and print the `title` column from the book table in alphabetical order.

```
>>> import sqlalchemy
>>> conn = sqlalchemy.create_engine('sqlite:///books.db')
>>> sql = 'select title from book order by title asc'
>>> rows = conn.execute(sql)
>>> for row in rows:
...     print(row)
...
('Perdido Street Station',)
('Small Gods',)
('The Spellman Files',)
('The Weirdstone of Brisingamen',)
('Thud!',)
```

16.9 Install the Redis server (see Appendix B) and the Python `redis` library (`pip install redis`) on your machine. Create a Redis hash called `test` with the fields `count` (`1`) and `name` (`'Fester Bestertester'`). Print all the fields for `test`.

```
>>> import redis
>>> conn = redis.Redis()
>>> conn.delete('test')
1
>>> conn.hmset('test', {'count': 1, 'name': 'Fester Bestertester'})
True
>>> conn.hgetall('test')
{b'name': b'Fester Bestertester', b'count': b'1'}
```

16.10 Increment the `count` field of `test` and print it.

```
>>> conn.hincrby('test', 'count', 3)
4
>>> conn.hget('test', 'count')
b'4'
```

17. Data in Space: Networks

17.1 Use a plain `socket` to implement a current-time service. When a client sends the string `'time'` to the server, return the current date and time as an ISO string.

Here's one way to write the server, *udp_time_server.py*:

```
from datetime import datetime
import socket

address = ('localhost', 6789)
max_size = 4096

print('Starting the server at', datetime.now())
print('Waiting for a client to call.')
server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(address)
while True:
    data, client_addr = server.recvfrom(max_size)
    if data == b'time':
        now = str(datetime.utcnow())
        data = now.encode('utf-8')
        server.sendto(data, client_addr)
```

```
            print('Server sent', data)
    server.close()
```

And the client, *udp_time_client.py*:

```
import socket
from datetime import datetime
from time import sleep

address    = ('localhost', 6789)
max_size   = 4096

print('Starting the client at', datetime.now())
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
    sleep(5)
    client.sendto(b'time', address)
    data, server_addr = client.recvfrom(max_size)
    print('Client read', data)
client.close()
```

I put in a `sleep(5)` call at the top of the client loop to make the data exchange less supersonic. Start the server in one window:

```
$ python udp_time_server.py
Starting the server at 2014-06-02 20:28:47.415176
Waiting for a client to call.
```

Start the client in another window:

```
$ python udp_time_client.py
Starting the client at 2014-06-02 20:28:51.454805
```

After five seconds, you'll start getting output in both windows. Here are the first three lines from the server:

```
Server sent b'2014-06-03 01:28:56.462565'
Server sent b'2014-06-03 01:29:01.463906'
Server sent b'2014-06-03 01:29:06.465802'
```

And here are the first three from the client:

```
Client read b'2014-06-03 01:28:56.462565'
Client read b'2014-06-03 01:29:01.463906'
Client read b'2014-06-03 01:29:06.465802'
```

Both of these programs run forever, so you'll need to cancel them manually.

17.2. Use ZeroMQ REQ and REP sockets to do the same thing.

```
import zmq
from datetime import datetime
```

```
host = '127.0.0.1'
port = 6789
context = zmq.Context()
server = context.socket(zmq.REP)
server.bind("tcp://%s:%s" % (host, port))
print('Server started at', datetime.utcnow())
while True:
    #  Wait for next request from client
    message = server.recv()
    if message == b'time':
        now = datetime.utcnow()
        reply = str(now)
        server.send(bytes(reply, 'utf-8'))
        print('Server sent', reply)
```

```
import zmq
from datetime import datetime
from time import sleep

host = '127.0.0.1'
port = 6789
context = zmq.Context()
client = context.socket(zmq.REQ)
client.connect("tcp://%s:%s" % (host, port))
print('Client started at', datetime.utcnow())
while True:
    sleep(5)
    request = b'time'
    client.send(request)
    reply = client.recv()
    print("Client received %s" % reply)
```

With plain sockets, you need to start the server first. With ZeroMQ, you can start either the server or client first.

```
$ python zmq_time_server.py
Server started at 2014-06-03 01:39:36.933532
```

```
$ python zmq_time_client.py
Client started at 2014-06-03 01:39:42.538245
```

After 15 seconds or so, you should have some lines from the server:

```
Server sent 2014-06-03 01:39:47.539878
Server sent 2014-06-03 01:39:52.540659
Server sent 2014-06-03 01:39:57.541403
```

Here's what you should see from the client:

```
Client received b'2014-06-03 01:39:47.539878'
Client received b'2014-06-03 01:39:52.540659'
Client received b'2014-06-03 01:39:57.541403'
```

17.3. Try the same with XMLRPC.

From the server:

```
from xmlrpc.server import SimpleXMLRPCServer

def now():
    from datetime import datetime
    data = str(datetime.utcnow())
    print('Server sent', data)
    return data

server = SimpleXMLRPCServer(("localhost", 6789))
server.register_function(now, "now")
server.serve_forever()
```

And from the client:

```
import xmlrpc.client
from time import sleep

proxy = xmlrpc.client.ServerProxy("http://localhost:6789/")
while True:
    sleep(5)
    data = proxy.now()
    print('Client received', data)
```

Start the server:

```
$ python xmlrpc_time_server.py
```

Start the client:

```
$ python xmlrpc_time_client.py
```

Wait 15 seconds or so. Here are the first three lines of server output:

```
Server sent 2014-06-03 02:14:52.299122
127.0.0.1 - - [02/Jun/2014 21:14:52] "POST / HTTP/1.1" 200 -
Server sent 2014-06-03 02:14:57.304741
127.0.0.1 - - [02/Jun/2014 21:14:57] "POST / HTTP/1.1" 200 -
Server sent 2014-06-03 02:15:02.310377
127.0.0.1 - - [02/Jun/2014 21:15:02] "POST / HTTP/1.1" 200 -
```

And here are the first three lines from the client:

```
Client received 2014-06-03 02:14:52.299122
Client received 2014-06-03 02:14:57.304741
Client received 2014-06-03 02:15:02.310377
```

17.4 You may have seen the classic *I Love Lucy* television episode in which Lucy and Ethel worked in a chocolate factory. The duo fell behind as the conveyor belt that supplied the confections for them to process began operating at an ever-faster rate. Write a simulation that pushes different types of chocolates to a Redis list, and Lucy is a client doing blocking pops of this list. She needs 0.5 seconds to handle a piece of chocolate. Print the time and type of each chocolate as Lucy gets it, and how many remain to be handled.

redis_choc_supply.py supplies the infinite treats:

```
import redis
import random
from time import sleep

conn = redis.Redis()
varieties = ['truffle', 'cherry', 'caramel', 'nougat']
conveyor = 'chocolates'
while True:
    seconds = random.random()
    sleep(seconds)
    piece = random.choice(varieties)
    conn.rpush(conveyor, piece)
```

redis_lucy.py might look like this:

```
import redis
from datetime import datetime
from time import sleep

conn = redis.Redis()
timeout = 10
conveyor = 'chocolates'
while True:
    sleep(0.5)
    msg = conn.blpop(conveyor, timeout)
    remaining = conn.llen(conveyor)
    if msg:
        piece = msg[1]
        print('Lucy got a', piece, 'at', datetime.utcnow(),
        ', only', remaining, 'left')
```

Start them in either order. Because Lucy takes a half second to handle each, and they're being produced every half second on average, it's a race to keep up. The more of a head start that you give to the conveyor belt, the harder you make Lucy's life.

```
$ python redis_choc_supply.py &

$ python redis_lucy.py
Lucy got a b'nougat' at 2014-06-03 03:15:08.721169 , only 4 left
Lucy got a b'cherry' at 2014-06-03 03:15:09.222816 , only 3 left
Lucy got a b'truffle' at 2014-06-03 03:15:09.723691 , only 5 left
```

```
Lucy got a b'truffle' at 2014-06-03 03:15:10.225008 , only 4 left
Lucy got a b'cherry' at 2014-06-03 03:15:10.727107 , only 4 left
Lucy got a b'cherry' at 2014-06-03 03:15:11.228226 , only 5 left
Lucy got a b'cherry' at 2014-06-03 03:15:11.729735 , only 4 left
Lucy got a b'truffle' at 2014-06-03 03:15:12.230894 , only 6 left
Lucy got a b'caramel' at 2014-06-03 03:15:12.732777 , only 7 left
Lucy got a b'cherry' at 2014-06-03 03:15:13.234785 , only 6 left
Lucy got a b'cherry' at 2014-06-03 03:15:13.736103 , only 7 left
Lucy got a b'caramel' at 2014-06-03 03:15:14.238152 , only 9 left
Lucy got a b'cherry' at 2014-06-03 03:15:14.739561 , only 8 left
```

Poor Lucy.

17.5 Use ZeroMQ to publish the poem from exercise 12.4 (from Example 12-1), one word at a time. Write a ZeroMQ consumer that prints every word that starts with a vowel, and another that prints every word that contains five letters. Ignore punctuation characters.

Here's the server, *poem_pub.py*, which plucks each word from the poem and publishes it to the topic `vowels` if it starts with a vowel, and the topic `five` if it has five letters. Some words might be in both topics, some in neither.

```
import string
import zmq

host = '127.0.0.1'
port = 6789
ctx = zmq.Context()
pub = ctx.socket(zmq.PUB)
pub.bind('tcp://%s:%s' % (host, port))

with open('mammoth.txt', 'rt') as poem:
    words = poem.read()
for word in words.split():
    word = word.strip(string.punctuation)
    data = word.encode('utf-8')
    if word.startswith(('a','e','i','o','u','A','e','i','o','u')):
        pub.send_multipart([b'vowels', data])
    if len(word) == 5:
        pub.send_multipart([b'five', data])
```

The client, *poem_sub.py*, subscribes to the topics `vowels` and `five` and prints the topic and word:

```
import string
import zmq

host = '127.0.0.1'
port = 6789
```

```
ctx = zmq.Context()
sub = ctx.socket(zmq.SUB)
sub.connect('tcp://%s:%s' % (host, port))
sub.setsockopt(zmq.SUBSCRIBE, b'vowels')
sub.setsockopt(zmq.SUBSCRIBE, b'five')
while True:
    topic, word = sub.recv_multipart()
    print(topic, word)
```

If you start these and run them, they *almost* work. Your code looks fine but nothing happens. You need to read the ZeroMQ guide (*http://zguide.zeromq.org/page:all*) to learn about the *slow joiner* problem: even if you start the client before the server, the server begins pushing data immediately after starting, and the client takes a little time to connect to the server. If you're publishing a constant stream of something and don't really care when the subscribers jump in, it's no problem. But in this case, the data stream is so short that it's flown past before the subscriber blinks, like a fastball past a batter.

The easiest way to fix this is to make the publisher sleep a second after it calls `bind()` and before it starts sending messages. Call this version *poem_pub_sleep.py*:

```
import string
import zmq
from time import sleep

host = '127.0.0.1'
port = 6789
ctx = zmq.Context()
pub = ctx.socket(zmq.PUB)
pub.bind('tcp://%s:%s' % (host, port))

sleep(1)

with open('mammoth.txt', 'rt') as poem:
    words = poem.read()
for word in words.split():
    word = word.strip(string.punctuation)
    data = word.encode('utf-8')
    if word.startswith(('a','e','i','o','u','A','e','i','o','u')):
        print('vowels', data)
        pub.send_multipart([b'vowels', data])
    if len(word) == 5:
        print('five', data)
        pub.send_multipart([b'five', data])
```

Start the subscriber and then the sleepy publisher:

```
$ python poem_sub.py
```

```
$ python poem_pub_sleep.py
```

Now, the subscriber has time to grab its two topics. Here are the first few lines of its output:

```
b'five' b'queen'
b'vowels' b'of'
b'five' b'Lying'
b'vowels' b'at'
b'vowels' b'ease'
b'vowels' b'evening'
b'five' b'flies'
b'five' b'seize'
b'vowels' b'All'
b'five' b'gaily'
b'five' b'great'
b'vowels' b'admired'
```

If you can't add a `sleep()` to your publisher, you can synchronize publisher and subscriber programs by using `REQ` and `REP` sockets. See the *publisher.py* and *subscriber.py* examples on GitHub (*http://bit.ly/pyzmq-gh*).

18. The Web, Untangled

18.1 If you haven't installed `flask` yet, do so now. This will also install `werkzeug`, `jinja2`, and possibly other packages.

18.2 Build a skeleton website, using Flask's debug/reload development web server. Ensure that the server starts up for hostname `localhost` on default port `5000`. If your machine is already using port 5000 for something else, use another port number.

Here's *flask1.py*:

```
from flask import Flask

app = Flask(__name__)

app.run(port=5000, debug=True)
```

Gentlemen, start your engines:

```
$ python flask1.py
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

18.3 Add a `home()` function to handle requests for the home page. Set it up to return the string `It's alive!`.

What should we call this one, *flask2.py*?

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "It's alive!"

app.run(debug=True)
```

Start the server:

```
$ python flask2.py
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

Finally, access the home page via a browser, command-line HTTP program such as `curl` or `wget`, or even `telnet`:

```
$ curl http://localhost:5000/
It's alive!
```

18.4 Create a Jinja2 template file called *home.html* with the following contents:

```
I'm of course referring to {{thing}},
which is {{height}} feet tall and {{color}}.
```

Make a directory called *templates* and create the file *home.html* with the contents just shown. If your Flask server is still running from the previous examples, it will detect the new content and restart itself.

18.5 Modify your server's `home()` function to use the *home.html* template. Provide it with three `GET` parameters: `thing`, `height`, and `color`.

Here comes *flask3.py*:

```
from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/')
def home():
    thing = request.values.get('thing')
    height = request.values.get('height')
    color = request.values.get('color')
    return render_template('home.html',
        thing=thing, height=height, color=color)

app.run(debug=True)
```

Go to this address in your web client:

```
http://localhost:5000/?thing=Octothorpe&height=7&color=green
```

You should see the following:

```
I'm of course referring to Octothorpe, which is 7 feet tall and green.
```

19. Be a Pythonista

(Pythonistas don't have homework today.)

20. Py Art

20.1 Install `matplotlib`. Draw a scatter diagram of these (x, y) pairs: `( (0, 0), (3, 5), (6, 2), (9, 8), (14, 10) )`.

20.2 Draw a line graph of the same data.

20.3 Draw a plot (a line graph with markers) of the same data

This has all three as subplots:

```
import matplotlib.pyplot as plt

x = (0, 3, 6, 9, 14)
y = (0, 5, 2, 8, 10)
fig, plots = plt.subplots(nrows=1, ncols=3)

plots[0].scatter(x, y)
plots[1].plot(x, y)
plots[2].plot(x, y, 'o-')

plt.show()
```

21. Py at Work

21.1 Install `geopandas` and run Example 21-1. Try modifying things like colors and marker sizes.

22. PySci

22.1 Install Pandas. Get the CSV file in Example 16-1. Run the program in Example 16-2. Experiment with some of the Pandas commands.

APPENDIX E
Cheat Sheets

I find myself looking up certain things a little too often. Here are some tables that I hope you'll find useful.

Operator Precedence

This table is a remix of the official documentation on precedence in Python 3, with the *highest* precedence operators at the top.

Operator	Description and examples
`[`*v*, ...`]`, `{`*v1*, ...`}`, `{`*k1*`:`*v1*, ...`}`, `(`...`)`	List/set/dict/generator creation or comprehension, parenthesized expression
seq`[`*n*`]`, *seq*`[`*n*`:`*m*`]`, *func*`(`*args*...`)`, *obj*`.`*attr*	Index, slice, function call, attribute reference
`**`	Exponentiation
`+`*n*, `-`*n*, `~`*n*	Positive, negative, bitwise `not`
`*`, `/`, `//`, `%`	Multiplication, float division, int division, remainder
`+`, `-`	Addition, subtraction
`<<`, `>>`	Bitwise left, right shifts
`&`	Bitwise `and`
`|`	Bitwise `or`
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	Membership and equality tests
`not` *x*	Boolean (logical) `not`
`and`	Boolean `and`
`or`	Boolean `or`
`if` ... `else`	Conditional expression
`lambda` ...	`lambda` expression

String Methods

Python offers both string *methods* (can be used with any `str` object) and a `string` module with some useful definitions. Let's use these test variables:

```
>>> s = "OH, my paws and whiskers!"
>>> t = "I'm late!"
```

In the following examples, the Python shell prints the result of the method call, but the original variables `s` and `t` are not changed.

Change Case

```
>>> s.capitalize()
'Oh, my paws and whiskers!'
>>> s.lower()
'oh, my paws and whiskers!'
>>> s.swapcase()
'oh, MY PAWS AND WHISKERS!'
>>> s.title()
'Oh, My Paws And Whiskers!'
>>> s.upper()
'OH, MY PAWS AND WHISKERS!'
```

Search

```
>>> s.count('w')
2
>>> s.find('w')
9
>>> s.index('w')
9
>>> s.rfind('w')
16
>>> s.rindex('w')
16
>>> s.startswith('OH')
True
```

Modify

```
>>> ''.join(s)
'OH, my paws and whiskers!'
>>> ' '.join(s)
'O H ,   m y   p a w s   a n d   w h i s k e r s !'
>>> ' '.join((s, t))
"OH, my paws and whiskers! I'm late!"
>>> s.lstrip('HO')
', my paws and whiskers!'
>>> s.replace('H', 'MG')
'OMG, my paws and whiskers!'
```

```
>>> s.rsplit()
['OH,', 'my', 'paws', 'and', 'whiskers!']
>>> s.rsplit(' ', 1)
['OH, my paws and', 'whiskers!']
>>> s.split(' ', 1)
['OH,', 'my paws and whiskers!']
>>> s.split(' ')
['OH,', 'my', 'paws', 'and', 'whiskers!']
>>> s.splitlines()
['OH, my paws and whiskers!']
>>> s.strip()
'OH, my paws and whiskers!'
>>> s.strip('s!')
'OH, my paws and whisker'
```

Format

```
>>> s.center(30)
'  OH, my paws and whiskers!   '
>>> s.expandtabs()
'OH, my paws and whiskers!'
>>> s.ljust(30)
'OH, my paws and whiskers!     '
>>> s.rjust(30)
'     OH, my paws and whiskers!'
```

String Type

```
>>> s.isalnum()
False
>>> s.isalpha()
False
>>> s.isprintable()
True
>>> s.istitle()
False
>>> s.isupper()
False
>>> s.isdecimal()
False
>>> s.isnumeric()
False
```

String Module Attributes

These are class attributes that are used as constant definitions.

Attribute	Example
ascii_letters	'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
ascii_lowercase	'abcdefghijklmnopqrstuvwxyz'
ascii_uppercase	'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
digits	'0123456789'
hexdigits	'0123456789abcdefABCDEF'
octdigits	'01234567'
punctuation	'!"#$%&\'()*+,-./:;<=>?@[\\]^_\{|}~`'
printable	digits + ascii_letters + punctuation + whitespace
whitespace	' \t\n\r\x0b\x0c'

Coda

Chester wants to express his appreciation for your diligence. If you need him, he's taking a nap…

Figure E-1. Chester[1]

1 He's moved about a foot to the right since Figure 3-1.

…but Lucy is available to answer any questions.

Figure E-2. Lucy