APPENDIX A

Support Scripts

Here are full listings for some scripts that were too long to fit in the main text. Also included are scripts used to generate some of the tables and data fixtures used in this book.

These scripts are also available in the *Fluent Python* code repository, along with almost every other code snippet that appears in the book.

Chapter 3: in Operator Performance Test

Example A-1 is the code I used to produce the timings in Table 3-6 using the `timeit` module. The script mostly deals with setting up the `haystack` and `needles` samples and with formatting output.

While coding Example A-1, I found something that really puts `dict` performance in perspective. If the script is run in "verbose mode" (with the `-v` command-line option), the timings I get are nearly twice those in Table 3-5. But note that, in this script, "verbose mode" means only four calls to `print` while setting up the test, and one additional `print` to show the number of needles found when each test finishes. No output happens within the loop that does the actual search of the needles in the haystack, but these five `print` calls take about as much time as searching for 1,000 needles.

Example A-1. container_perftest.py: run it with the name of a built-in collection type as a command-line argument (e.g., container_perftest.py dict)

```
"""
Container ``in`` operator performance test
"""
import sys
import timeit

SETUP = '''
```

```
import array
selected = array.array('d')
with open('selected.arr', 'rb') as fp:
    selected.fromfile(fp, {size})
if {container_type} is dict:
    haystack = dict.fromkeys(selected, 1)
else:
    haystack = {container_type}(selected)
if {verbose}:
    print(type(haystack), end='  ')
    print('haystack: %10d' % len(haystack), end='  ')
needles = array.array('d')
with open('not_selected.arr', 'rb') as fp:
    needles.fromfile(fp, 500)
needles.extend(selected[::{size}//500])
if {verbose}:
    print(' needles: %10d' % len(needles), end='  ')
'''

TEST = '''
found = 0
for n in needles:
    if n in haystack:
        found += 1
if {verbose}:
    print('  found: %10d' % found)
'''

def test(container_type, verbose):
    MAX_EXPONENT = 7
    for n in range(3, MAX_EXPONENT + 1):
        size = 10**n
        setup = SETUP.format(container_type=container_type,
                             size=size, verbose=verbose)
        test = TEST.format(verbose=verbose)
        tt = timeit.repeat(stmt=test, setup=setup, repeat=5, number=1)
        print('|{:{}d}|{:f}'.format(size, MAX_EXPONENT + 1, min(tt)))

if __name__=='__main__':
    if '-v' in sys.argv:
        sys.argv.remove('-v')
        verbose = True
    else:
        verbose = False
    if len(sys.argv) != 2:
        print('Usage: %s <container_type>' % sys.argv[0])
    else:
        test(sys.argv[1], verbose)
```

The script *container_perftest_datagen.py* (Example A-2) generates the data fixture for the script in Example A-1.

Example A-2. container_perftest_datagen.py: generate files with arrays of unique floating point numbers for use in Example A-1

```
"""
Generate data for container performance test
"""

import random
import array

MAX_EXPONENT = 7
HAYSTACK_LEN = 10 ** MAX_EXPONENT
NEEDLES_LEN = 10 ** (MAX_EXPONENT - 1)
SAMPLE_LEN = HAYSTACK_LEN + NEEDLES_LEN // 2

needles = array.array('d')

sample = {1/random.random() for i in range(SAMPLE_LEN)}
print('initial sample: %d elements' % len(sample))

# complete sample, in case duplicate random numbers were discarded
while len(sample) < SAMPLE_LEN:
    sample.add(1/random.random())

print('complete sample: %d elements' % len(sample))

sample = array.array('d', sample)
random.shuffle(sample)

not_selected = sample[:NEEDLES_LEN // 2]
print('not selected: %d samples' % len(not_selected))
print('  writing not_selected.arr')
with open('not_selected.arr', 'wb') as fp:
    not_selected.tofile(fp)

selected = sample[NEEDLES_LEN // 2:]
print('selected: %d samples' % len(selected))
print('  writing selected.arr')
with open('selected.arr', 'wb') as fp:
    selected.tofile(fp)
```

Chapter 3: Compare the Bit Patterns of Hashes

Example A-3 is a simple script to visually show how different are the bit patterns for the hashes of similiar floating-point numbers (e.g., 1.0001, 1.0002, etc.). Its output appears in Example 3-16.

Example A-3. hashdiff.py: display the difference of bit paterns from hash values

```
import sys

MAX_BITS = len(format(sys.maxsize, 'b'))
```

```
print('%s-bit Python build' % (MAX_BITS + 1))

def hash_diff(o1, o2):
    h1 = '{:>0{}b}'.format(hash(o1), MAX_BITS)
    h2 = '{:>0{}b}'.format(hash(o2), MAX_BITS)
    diff = ''.join('!' if b1 != b2 else ' ' for b1, b2 in zip(h1, h2))
    count = '!= {}'.format(diff.count('!'))
    width = max(len(repr(o1)), len(repr(o2)), 8)
    sep = '-' * (width * 2 + MAX_BITS)
    return '{!r:{width}} {}\n{:{width}} {} {}\n{!r:{width}} {}\n{}'.format(
                o1, h1, ' ' * width, diff, count, o2, h2, sep, width=width)

if __name__ == '__main__':
    print(hash_diff(1, 1.0))
    print(hash_diff(1.0, 1.0001))
    print(hash_diff(1.0001, 1.0002))
    print(hash_diff(1.0002, 1.0003))
```

Chapter 9: RAM Usage With and Without __slots__

The *memtest.py* script was used for a demostration in "Saving Space with the __slots__ Class Attribute" on page 264: Example 9-12.

The *memtest.py* script takes a module name in the command line and loads it. Assuming the module defines a class named `Vector`, *memtest.py* creates a list with 10 million instances, reporting the memory usage before and after the list is created.

Example A-4. memtest.py: create lots of Vector instances reporting memory usage

```
import importlib
import sys
import resource

NUM_VECTORS = 10**7

if len(sys.argv) == 2:
    module_name = sys.argv[1].replace('.py', '')
    module = importlib.import_module(module_name)
else:
    print('Usage: {} <vector-module-to-test>'.format())
    sys.exit(1)

fmt = 'Selected Vector2d type: {.__name__}.{.__name__}'
print(fmt.format(module, module.Vector2d))

mem_init = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss
print('Creating {:,} Vector2d instances'.format(NUM_VECTORS))

vectors = [module.Vector2d(3.0, 4.0) for i in range(NUM_VECTORS)]

mem_final = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss
```

```
print('Initial RAM usage: {:14,}'.format(mem_init))
print('  Final RAM usage: {:14,}'.format(mem_final))
```

Chapter 14: isis2json.py Database Conversion Script

Example A-5 is the *isis2json.py* script discussed in "Case Study: Generators in a Database Conversion Utility" on page 437 (Chapter 14). It uses generator functions to lazily convert CDS/ISIS databases to JSON for loading to CouchDB or MongoDB.

Note that this is a Python 2 script, designed to run on CPython or Jython, versions 2.5 to 2.7, but not on Python 3. Under CPython it can read only *.iso* files; with Jython it can also read *.mst* files, using the `Bruma` library available on the *fluentpython/isis2json* repository in GitHub. See usage documentation in that repository.

Example A-5. isis2json.py: dependencies and documentation available on GitHub repository fluentpython/isis2json

```
# this script works with Python or Jython (versions >=2.5 and <3)

import sys
import argparse
from uuid import uuid4
import os

try:
    import json
except ImportError:
    if os.name == 'java':  # running Jython
        from com.xhaus.jyson import JysonCodec as json
    else:
        import simplejson as json

SKIP_INACTIVE = True
DEFAULT_QTY = 2**31
ISIS_MFN_KEY = 'mfn'
ISIS_ACTIVE_KEY = 'active'
SUBFIELD_DELIMITER = '^'
INPUT_ENCODING = 'cp1252'


def iter_iso_records(iso_file_name, isis_json_type):    ❶
    from iso2709 import IsoFile
    from subfield import expand

    iso = IsoFile(iso_file_name)
    for record in iso:
        fields = {}
        for field in record.directory:
            field_key = str(int(field.tag))  # remove leading zeroes
            field_occurrences = fields.setdefault(field_key, [])
            content = field.value.decode(INPUT_ENCODING, 'replace')
```

```
            if isis_json_type == 1:
                field_occurrences.append(content)
            elif isis_json_type == 2:
                field_occurrences.append(expand(content))
            elif isis_json_type == 3:
                field_occurrences.append(dict(expand(content)))
            else:
                raise NotImplementedError('ISIS-JSON type %s conversion '
                    'not yet implemented for .iso input' % isis_json_type)

        yield fields
    iso.close()


def iter_mst_records(master_file_name, isis_json_type):    ❷
    try:
        from bruma.master import MasterFactory, Record
    except ImportError:
        print('IMPORT ERROR: Jython 2.5 and Bruma.jar '
              'are required to read .mst files')
        raise SystemExit
    mst = MasterFactory.getInstance(master_file_name).open()
    for record in mst:
        fields = {}
        if SKIP_INACTIVE:
            if record.getStatus() != Record.Status.ACTIVE:
                continue
        else:  # save status only there are non-active records
            fields[ISIS_ACTIVE_KEY] = (record.getStatus() ==
                                       Record.Status.ACTIVE)
        fields[ISIS_MFN_KEY] = record.getMfn()
        for field in record.getFields():
            field_key = str(field.getId())
            field_occurrences = fields.setdefault(field_key, [])
            if isis_json_type == 3:
                content = {}
                for subfield in field.getSubfields():
                    subfield_key = subfield.getId()
                    if subfield_key == '*':
                        content['_'] = subfield.getContent()
                    else:
                        subfield_occurrences = content.setdefault(subfield_key, [])
                        subfield_occurrences.append(subfield.getContent())
                field_occurrences.append(content)
            elif isis_json_type == 1:
                content = []
                for subfield in field.getSubfields():
                    subfield_key = subfield.getId()
                    if subfield_key == '*':
                        content.insert(0, subfield.getContent())
                    else:
                        content.append(SUBFIELD_DELIMITER + subfield_key +
```

```
                                    subfield.getContent())
                field_occurrences.append(''.join(content))
            else:
                raise NotImplementedError('ISIS-JSON type %s conversion '
                    'not yet implemented for .mst input' % isis_json_type)
        yield fields
    mst.close()


def write_json(input_gen, file_name, output, qty, skip, id_tag,    ❸
               gen_uuid, mongo, mfn, isis_json_type, prefix,
               constant):
    start = skip
    end = start + qty
    if id_tag:
        id_tag = str(id_tag)
        ids = set()
    else:
        id_tag = ''
    for i, record in enumerate(input_gen):
        if i >= end:
            break
        if not mongo:
            if i == 0:
                output.write('[')
            elif i > start:
                output.write(',')
        if start <= i < end:
            if id_tag:
                occurrences = record.get(id_tag, None)
                if occurrences is None:
                    msg = 'id tag #%s not found in record %s'
                    if ISIS_MFN_KEY in record:
                        msg = msg + (' (mfn=%s)' % record[ISIS_MFN_KEY])
                    raise KeyError(msg % (id_tag, i))
                if len(occurrences) > 1:
                    msg = 'multiple id tags #%s found in record %s'
                    if ISIS_MFN_KEY in record:
                        msg = msg + (' (mfn=%s)' % record[ISIS_MFN_KEY])
                    raise TypeError(msg % (id_tag, i))
                else:  # ok, we have one and only one id field
                    if isis_json_type == 1:
                        id = occurrences[0]
                    elif isis_json_type == 2:
                        id = occurrences[0][0][1]
                    elif isis_json_type == 3:
                        id = occurrences[0]['_']
                    if id in ids:
                        msg = 'duplicate id %s in tag #%s, record %s'
                        if ISIS_MFN_KEY in record:
                            msg = msg + (' (mfn=%s)' % record[ISIS_MFN_KEY])
                        raise TypeError(msg % (id, id_tag, i))
```

```
                    record['_id'] = id
                    ids.add(id)
            elif gen_uuid:
                record['_id'] = unicode(uuid4())
            elif mfn:
                record['_id'] = record[ISIS_MFN_KEY]
            if prefix:
                # iterate over a fixed sequence of tags
                for tag in tuple(record):
                    if str(tag).isdigit():
                        record[prefix+tag] = record[tag]
                        del record[tag]  # this is why we iterate over a tuple
                        # with the tags, and not directly on the record dict
            if constant:
                constant_key, constant_value = constant.split(':')
                record[constant_key] = constant_value
            output.write(json.dumps(record).encode('utf-8'))
            output.write('\n')
    if not mongo:
        output.write(']\n')


def main():   ❹
    # create the parser
    parser = argparse.ArgumentParser(
        description='Convert an ISIS .mst or .iso file to a JSON array')

    # add the arguments
    parser.add_argument(
        'file_name', metavar='INPUT.(mst|iso)',
        help='.mst or .iso file to read')
    parser.add_argument(
        '-o', '--out', type=argparse.FileType('w'), default=sys.stdout,
        metavar='OUTPUT.json',
        help='the file where the JSON output should be written'
             ' (default: write to stdout)')
    parser.add_argument(
        '-c', '--couch', action='store_true',
        help='output array within a "docs" item in a JSON document'
             ' for bulk insert to CouchDB via POST to db/_bulk_docs')
    parser.add_argument(
        '-m', '--mongo', action='store_true',
        help='output individual records as separate JSON dictionaries, one'
             ' per line for bulk insert to MongoDB via mongoimport utility')
    parser.add_argument(
        '-t', '--type', type=int, metavar='ISIS_JSON_TYPE', default=1,
        help='ISIS-JSON type, sets field structure: 1=string, 2=alist,'
             ' 3=dict (default=1)')
    parser.add_argument(
        '-q', '--qty', type=int, default=DEFAULT_QTY,
        help='maximum quantity of records to read (default=ALL)')
    parser.add_argument(
```

```
        '-s', '--skip', type=int, default=0,
        help='records to skip from start of .mst (default=0)')
    parser.add_argument(
        '-i', '--id', type=int, metavar='TAG_NUMBER', default=0,
        help='generate an "_id" from the given unique TAG field number'
             ' for each record')
    parser.add_argument(
        '-u', '--uuid', action='store_true',
        help='generate an "_id" with a random UUID for each record')
    parser.add_argument(
        '-p', '--prefix', type=str, metavar='PREFIX', default='',
        help='concatenate prefix to every numeric field tag'
             ' (ex. 99 becomes "v99")')
    parser.add_argument(
        '-n', '--mfn', action='store_true',
        help='generate an "_id" from the MFN of each record'
             ' (available only for .mst input)')
    parser.add_argument(
        '-k', '--constant', type=str, metavar='TAG:VALUE', default='',
        help='Include a constant tag:value in every record (ex. -k type:AS)')

    '''
    # TODO: implement this to export large quantities of records to CouchDB
    parser.add_argument(
        '-r', '--repeat', type=int, default=1,
        help='repeat operation, saving multiple JSON files'
             ' (default=1, use -r 0 to repeat until end of input)')
    '''
    # parse the command line
    args = parser.parse_args()
    if args.file_name.lower().endswith('.mst'):
        input_gen_func = iter_mst_records    ❺
    else:
        if args.mfn:
            print('UNSUPORTED: -n/--mfn option only available for .mst input.')
            raise SystemExit
        input_gen_func = iter_iso_records    ❻
    input_gen = input_gen_func(args.file_name, args.type)    ❼
    if args.couch:
        args.out.write('{ "docs" : ')
    write_json(input_gen, args.file_name, args.out, args.qty,    ❽
               args.skip, args.id, args.uuid, args.mongo, args.mfn,
               args.type, args.prefix, args.constant)
    if args.couch:
        args.out.write('}\n')
    args.out.close()


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

❶ `iter_iso_records` generator function reads *.iso* file, yields records.

❷ `iter_mst_records` generator function reads *.mst* file, yields records.

❸ `write_json` iterates over `input_gen` generator and outputs the *.json* file.

❹ Main function reads command-line arguments then…

❺ …selects `iter_iso_records` or…

❻ …`iter_mst_records` depending on input file extension.

❼ A generator object is built from the selected generator function.

❽ `write_json` is called with the generator as the first argument.

Chapter 16: Taxi Fleet Discrete Event Simulation

Example A-6 is the full listing for *taxi_sim.py* discussed in "The Taxi Fleet Simulation" on page 490.

Example A-6. taxi_sim.py: the taxi fleet simulator

```
"""
Taxi simulator
==============

Driving a taxi from the console::

    >>> from taxi_sim import taxi_process
    >>> taxi = taxi_process(ident=13, trips=2, start_time=0)
    >>> next(taxi)
    Event(time=0, proc=13, action='leave garage')
    >>> taxi.send(_.time + 7)
    Event(time=7, proc=13, action='pick up passenger')
    >>> taxi.send(_.time + 23)
    Event(time=30, proc=13, action='drop off passenger')
    >>> taxi.send(_.time + 5)
    Event(time=35, proc=13, action='pick up passenger')
    >>> taxi.send(_.time + 48)
    Event(time=83, proc=13, action='drop off passenger')
    >>> taxi.send(_.time + 1)
    Event(time=84, proc=13, action='going home')
    >>> taxi.send(_.time + 10)
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    StopIteration

Sample run with two cars, random seed 10. This is a valid doctest::

    >>> main(num_taxis=2, seed=10)
    taxi: 0  Event(time=0, proc=0, action='leave garage')
    taxi: 0  Event(time=5, proc=0, action='pick up passenger')
    taxi: 1     Event(time=5, proc=1, action='leave garage')
    taxi: 1     Event(time=10, proc=1, action='pick up passenger')
```

```
    taxi: 1     Event(time=15, proc=1, action='drop off passenger')
    taxi: 0  Event(time=17, proc=0, action='drop off passenger')
    taxi: 1     Event(time=24, proc=1, action='pick up passenger')
    taxi: 0  Event(time=26, proc=0, action='pick up passenger')
    taxi: 0  Event(time=30, proc=0, action='drop off passenger')
    taxi: 0  Event(time=34, proc=0, action='going home')
    taxi: 1     Event(time=46, proc=1, action='drop off passenger')
    taxi: 1     Event(time=48, proc=1, action='pick up passenger')
    taxi: 1     Event(time=110, proc=1, action='drop off passenger')
    taxi: 1     Event(time=139, proc=1, action='pick up passenger')
    taxi: 1     Event(time=140, proc=1, action='drop off passenger')
    taxi: 1     Event(time=150, proc=1, action='going home')
    *** end of events ***

See longer sample run at the end of this module.

"""

import random
import collections
import queue
import argparse
import time

DEFAULT_NUMBER_OF_TAXIS = 3
DEFAULT_END_TIME = 180
SEARCH_DURATION = 5
TRIP_DURATION = 20
DEPARTURE_INTERVAL = 5

Event = collections.namedtuple('Event', 'time proc action')


# BEGIN TAXI_PROCESS
def taxi_process(ident, trips, start_time=0):
    """Yield to simulator issuing event at each state change"""
    time = yield Event(start_time, ident, 'leave garage')
    for i in range(trips):
        time = yield Event(time, ident, 'pick up passenger')
        time = yield Event(time, ident, 'drop off passenger')

    yield Event(time, ident, 'going home')
    # end of taxi process
# END TAXI_PROCESS


# BEGIN TAXI_SIMULATOR
class Simulator:

    def __init__(self, procs_map):
        self.events = queue.PriorityQueue()
        self.procs = dict(procs_map)
```

```
    def run(self, end_time):
        """Schedule and display events until time is up"""
        # schedule the first event for each cab
        for _, proc in sorted(self.procs.items()):
            first_event = next(proc)
            self.events.put(first_event)

        # main loop of the simulation
        sim_time = 0
        while sim_time < end_time:
            if self.events.empty():
                print('*** end of events ***')
                break

            current_event = self.events.get()
            sim_time, proc_id, previous_action = current_event
            print('taxi:', proc_id, proc_id * '   ', current_event)
            active_proc = self.procs[proc_id]
            next_time = sim_time + compute_duration(previous_action)
            try:
                next_event = active_proc.send(next_time)
            except StopIteration:
                del self.procs[proc_id]
            else:
                self.events.put(next_event)
        else:
            msg = '*** end of simulation time: {} events pending ***'
            print(msg.format(self.events.qsize()))
# END TAXI_SIMULATOR


def compute_duration(previous_action):
    """Compute action duration using exponential distribution"""
    if previous_action in ['leave garage', 'drop off passenger']:
        # new state is prowling
        interval = SEARCH_DURATION
    elif previous_action == 'pick up passenger':
        # new state is trip
        interval = TRIP_DURATION
    elif previous_action == 'going home':
        interval = 1
    else:
        raise ValueError('Unknown previous_action: %s' % previous_action)
    return int(random.expovariate(1/interval)) + 1


def main(end_time=DEFAULT_END_TIME, num_taxis=DEFAULT_NUMBER_OF_TAXIS,
         seed=None):
    """Initialize random generator, build procs and run simulation"""
    if seed is not None:
        random.seed(seed)  # get reproducible results
```

```
    taxis = {i: taxi_process(i, (i+1)*2, i*DEPARTURE_INTERVAL)
             for i in range(num_taxis)}
    sim = Simulator(taxis)
    sim.run(end_time)


if __name__ == '__main__':

    parser = argparse.ArgumentParser(
                        description='Taxi fleet simulator.')
    parser.add_argument('-e', '--end-time', type=int,
                        default=DEFAULT_END_TIME,
                        help='simulation end time; default = %s'
                        % DEFAULT_END_TIME)
    parser.add_argument('-t', '--taxis', type=int,
                        default=DEFAULT_NUMBER_OF_TAXIS,
                        help='number of taxis running; default = %s'
                        % DEFAULT_NUMBER_OF_TAXIS)
    parser.add_argument('-s', '--seed', type=int, default=None,
                        help='random generator seed (for testing)')

    args = parser.parse_args()
    main(args.end_time, args.taxis, args.seed)


"""

Sample run from the command line, seed=3, maximum elapsed time=120::

# BEGIN TAXI_SAMPLE_RUN
$ python3 taxi_sim.py -s 3 -e 120
taxi: 0  Event(time=0, proc=0, action='leave garage')
taxi: 0  Event(time=2, proc=0, action='pick up passenger')
taxi: 1     Event(time=5, proc=1, action='leave garage')
taxi: 1     Event(time=8, proc=1, action='pick up passenger')
taxi: 2        Event(time=10, proc=2, action='leave garage')
taxi: 2        Event(time=15, proc=2, action='pick up passenger')
taxi: 2        Event(time=17, proc=2, action='drop off passenger')
taxi: 0  Event(time=18, proc=0, action='drop off passenger')
taxi: 2        Event(time=18, proc=2, action='pick up passenger')
taxi: 2        Event(time=25, proc=2, action='drop off passenger')
taxi: 1     Event(time=27, proc=1, action='drop off passenger')
taxi: 2        Event(time=27, proc=2, action='pick up passenger')
taxi: 0  Event(time=28, proc=0, action='pick up passenger')
taxi: 2        Event(time=40, proc=2, action='drop off passenger')
taxi: 2        Event(time=44, proc=2, action='pick up passenger')
taxi: 1     Event(time=55, proc=1, action='pick up passenger')
taxi: 1     Event(time=59, proc=1, action='drop off passenger')
taxi: 0  Event(time=65, proc=0, action='drop off passenger')
taxi: 1     Event(time=65, proc=1, action='pick up passenger')
taxi: 2        Event(time=65, proc=2, action='drop off passenger')
```

```
taxi: 2        Event(time=72, proc=2, action='pick up passenger')
taxi: 0  Event(time=76, proc=0, action='going home')
taxi: 1     Event(time=80, proc=1, action='drop off passenger')
taxi: 1     Event(time=88, proc=1, action='pick up passenger')
taxi: 2        Event(time=95, proc=2, action='drop off passenger')
taxi: 2        Event(time=97, proc=2, action='pick up passenger')
taxi: 2        Event(time=98, proc=2, action='drop off passenger')
taxi: 1     Event(time=106, proc=1, action='drop off passenger')
taxi: 2        Event(time=109, proc=2, action='going home')
taxi: 1     Event(time=110, proc=1, action='going home')
*** end of events ***
# END TAXI_SAMPLE_RUN

"""
```

Chapter 17: Cryptographic Examples

These scripts were used to show the use of `futures.ProcessPoolExecutor` to run CPU-intensive tasks.

Example A-7 encrypts and decrypts random byte arrays with the RC4 algorithm. It depends on the *arcfour.py* module (Example A-8) to run.

Example A-7. arcfour_futures.py: futures.ProcessPoolExecutor example

```
import sys
import time
from concurrent import futures
from random import randrange
from arcfour import arcfour

JOBS = 12
SIZE = 2**18

KEY = b"'Twas brillig, and the slithy toves\nDid gyre"
STATUS = '{} workers, elapsed time: {:.2f}s'


def arcfour_test(size, key):
    in_text = bytearray(randrange(256) for i in range(size))
    cypher_text = arcfour(key, in_text)
    out_text = arcfour(key, cypher_text)
    assert in_text == out_text, 'Failed arcfour_test'
    return size


def main(workers=None):
    if workers:
        workers = int(workers)
    t0 = time.time()

    with futures.ProcessPoolExecutor(workers) as executor:
```

```
        actual_workers = executor._max_workers
        to_do = []
        for i in range(JOBS, 0, -1):
            size = SIZE + int(SIZE / JOBS * (i - JOBS/2))
            job = executor.submit(arcfour_test, size, KEY)
            to_do.append(job)

        for future in futures.as_completed(to_do):
            res = future.result()
            print('{:.1f} KB'.format(res/2**10))

    print(STATUS.format(actual_workers, time.time() - t0))

if __name__ == '__main__':
    if len(sys.argv) == 2:
        workers = int(sys.argv[1])
    else:
        workers = None
    main(workers)
```

Example A-8 implements the RC4 encryption algorithm in pure Python.

Example A-8. arcfour.py: RC4 compatible algorithm

```
"""RC4 compatible algorithm"""

def arcfour(key, in_bytes, loops=20):

    kbox = bytearray(256)  # create key box
    for i, car in enumerate(key):  # copy key and vector
        kbox[i] = car
    j = len(key)
    for i in range(j, 256):  # repeat until full
        kbox[i] = kbox[i-j]

    # [1] initialize sbox
    sbox = bytearray(range(256))

    # repeat sbox mixing loop, as recommened in CipherSaber-2
    # http://ciphersaber.gurus.com/faq.html#cs2
    j = 0
    for k in range(loops):
        for i in range(256):
            j = (j + sbox[i] + kbox[i]) % 256
            sbox[i], sbox[j] = sbox[j], sbox[i]

    # main loop
    i = 0
    j = 0
    out_bytes = bytearray()

    for car in in_bytes:
        i = (i + 1) % 256
```

```
        # [2] shuffle sbox
        j = (j + sbox[i]) % 256
        sbox[i], sbox[j] = sbox[j], sbox[i]
        # [3] compute t
        t = (sbox[i] + sbox[j]) % 256
        k = sbox[t]
        car = car ^ k
        out_bytes.append(car)

    return out_bytes


def test():
    from time import time
    clear = bytearray(b'1234567890' * 100000)
    t0 = time()
    cipher = arcfour(b'key', clear)
    print('elapsed time: %.2fs' % (time() - t0))
    result = arcfour(b'key', cipher)
    assert result == clear, '%r != %r' % (result, clear)
    print('elapsed time: %.2fs' % (time() - t0))
    print('OK')


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

Example A-9 applies the SHA-256 hash algorithm to random byte arrays. It uses `hashlib` from the standard library, which in turn uses the OpenSSL library written in C.

Example A-9. sha_futures.py: futures.ProcessPoolExecutor example

```
import sys
import time
import hashlib
from concurrent import futures
from random import randrange

JOBS = 12
SIZE = 2**20
STATUS = '{} workers, elapsed time: {:.2f}s'


def sha(size):
    data = bytearray(randrange(256) for i in range(size))
    algo = hashlib.new('sha256')
    algo.update(data)
    return algo.hexdigest()


def main(workers=None):
    if workers:
```

```
        workers = int(workers)
    t0 = time.time()

    with futures.ProcessPoolExecutor(workers) as executor:
        actual_workers = executor._max_workers
        to_do = (executor.submit(sha, SIZE) for i in range(JOBS))
        for future in futures.as_completed(to_do):
            res = future.result()
            print(res)

    print(STATUS.format(actual_workers, time.time() - t0))

if __name__ == '__main__':
    if len(sys.argv) == 2:
        workers = int(sys.argv[1])
    else:
        workers = None
    main(workers)
```

Chapter 17: flags2 HTTP Client Examples

All `flags2` examples from "Downloads with Progress Display and Error Handling" on page 520 use functions from the *flags2_common.py* module (Example A-10).

Example A-10. flags2_common.py

```
"""Utilities for second set of flag examples.
"""

import os
import time
import sys
import string
import argparse
from collections import namedtuple
from enum import Enum


Result = namedtuple('Result', 'status data')

HTTPStatus = Enum('Status', 'ok not_found error')

POP20_CC = ('CN IN US ID BR PK NG BD RU JP '
            'MX PH VN ET EG DE IR TR CD FR').split()

DEFAULT_CONCUR_REQ = 1
MAX_CONCUR_REQ = 1

SERVERS = {
    'REMOTE': 'http://flupy.org/data/flags',
    'LOCAL':  'http://localhost:8001/flags',
    'DELAY':  'http://localhost:8002/flags',
```

```
    'ERROR':  'http://localhost:8003/flags',
}
DEFAULT_SERVER = 'LOCAL'

DEST_DIR = 'downloads/'
COUNTRY_CODES_FILE = 'country_codes.txt'


def save_flag(img, filename):
    path = os.path.join(DEST_DIR, filename)
    with open(path, 'wb') as fp:
        fp.write(img)


def initial_report(cc_list, actual_req, server_label):
    if len(cc_list) <= 10:
        cc_msg = ', '.join(cc_list)
    else:
        cc_msg = 'from {} to {}'.format(cc_list[0], cc_list[-1])
    print('{} site: {}'.format(server_label, SERVERS[server_label]))
    msg = 'Searching for {} flag{}: {}'
    plural = 's' if len(cc_list) != 1 else ''
    print(msg.format(len(cc_list), plural, cc_msg))
    plural = 's' if actual_req != 1 else ''
    msg = '{} concurrent connection{} will be used.'
    print(msg.format(actual_req, plural))


def final_report(cc_list, counter, start_time):
    elapsed = time.time() - start_time
    print('-' * 20)
    msg = '{} flag{} downloaded.'
    plural = 's' if counter[HTTPStatus.ok] != 1 else ''
    print(msg.format(counter[HTTPStatus.ok], plural))
    if counter[HTTPStatus.not_found]:
        print(counter[HTTPStatus.not_found], 'not found.')
    if counter[HTTPStatus.error]:
        plural = 's' if counter[HTTPStatus.error] != 1 else ''
        print('{} error{}.'.format(counter[HTTPStatus.error], plural))
    print('Elapsed time: {:.2f}s'.format(elapsed))


def expand_cc_args(every_cc, all_cc, cc_args, limit):
    codes = set()
    A_Z = string.ascii_uppercase
    if every_cc:
        codes.update(a+b for a in A_Z for b in A_Z)
    elif all_cc:
        with open(COUNTRY_CODES_FILE) as fp:
            text = fp.read()
        codes.update(text.split())
    else:
```

```
        for cc in (c.upper() for c in cc_args):
            if len(cc) == 1 and cc in A_Z:
                codes.update(cc+c for c in A_Z)
            elif len(cc) == 2 and all(c in A_Z for c in cc):
                codes.add(cc)
            else:
                msg = 'each CC argument must be A to Z or AA to ZZ.'
                raise ValueError('*** Usage error: '+msg)
    return sorted(codes)[:limit]


def process_args(default_concur_req):
    server_options = ', '.join(sorted(SERVERS))
    parser = argparse.ArgumentParser(
                description='Download flags for country codes. '
                'Default: top 20 countries by population.')
    parser.add_argument('cc', metavar='CC', nargs='*',
                help='country code or 1st letter (eg. B for BA...BZ)')
    parser.add_argument('-a', '--all', action='store_true',
                help='get all available flags (AD to ZW)')
    parser.add_argument('-e', '--every', action='store_true',
                help='get flags for every possible code (AA...ZZ)')
    parser.add_argument('-l', '--limit', metavar='N', type=int,
                help='limit to N first codes', default=sys.maxsize)
    parser.add_argument('-m', '--max_req', metavar='CONCURRENT', type=int,
                default=default_concur_req,
                help='maximum concurrent requests (default={})'
                      .format(default_concur_req))
    parser.add_argument('-s', '--server', metavar='LABEL',
                default=DEFAULT_SERVER,
                help='Server to hit; one of {} (default={})'
                      .format(server_options, DEFAULT_SERVER))
    parser.add_argument('-v', '--verbose', action='store_true',
                help='output detailed progress info')
    args = parser.parse_args()
    if args.max_req < 1:
        print('*** Usage error: --max_req CONCURRENT must be >= 1')
        parser.print_usage()
        sys.exit(1)
    if args.limit < 1:
        print('*** Usage error: --limit N must be >= 1')
        parser.print_usage()
        sys.exit(1)
    args.server = args.server.upper()
    if args.server not in SERVERS:
        print('*** Usage error: --server LABEL must be one of',
              server_options)
        parser.print_usage()
        sys.exit(1)
    try:
        cc_list = expand_cc_args(args.every, args.all, args.cc, args.limit)
    except ValueError as exc:
```

```
        print(exc.args[0])
        parser.print_usage()
        sys.exit(1)

    if not cc_list:
        cc_list = sorted(POP20_CC)
    return args, cc_list


def main(download_many, default_concur_req, max_concur_req):
    args, cc_list = process_args(default_concur_req)
    actual_req = min(args.max_req, max_concur_req, len(cc_list))
    initial_report(cc_list, actual_req, args.server)
    base_url = SERVERS[args.server]
    t0 = time.time()
    counter = download_many(cc_list, base_url, args.verbose, actual_req)
    assert sum(counter.values()) == len(cc_list), \
        'some downloads are unaccounted for'
    final_report(cc_list, counter, t0)
```

The *flags2_sequential.py* script (Example A-11) is the baseline for comparison with the concurrent implementations. *flags2_threadpool.py* (Example 17-14) also uses the `get_flag` and `download_one` functions from *flags2_sequential.py*.

Example A-11. flags2_sequential.py

```
"""Download flags of countries (with error handling).

Sequential version

Sample run::

    $ python3 flags2_sequential.py -s DELAY b
    DELAY site: http://localhost:8002/flags
    Searching for 26 flags: from BA to BZ
    1 concurrent connection will be used.
    --------------------
    17 flags downloaded.
    9 not found.
    Elapsed time: 13.36s

"""

import collections

import requests
import tqdm

from flags2_common import main, save_flag, HTTPStatus, Result


DEFAULT_CONCUR_REQ = 1
```

```
MAX_CONCUR_REQ = 1

# BEGIN FLAGS2_BASIC_HTTP_FUNCTIONS
def get_flag(base_url, cc):
    url = '{}/{cc}/{cc}.gif'.format(base_url, cc=cc.lower())
    resp = requests.get(url)
    if resp.status_code != 200:
        resp.raise_for_status()
    return resp.content


def download_one(cc, base_url, verbose=False):
    try:
        image = get_flag(base_url, cc)
    except requests.exceptions.HTTPError as exc:
        res = exc.response
        if res.status_code == 404:
            status = HTTPStatus.not_found
            msg = 'not found'
        else:
            raise
    else:
        save_flag(image, cc.lower() + '.gif')
        status = HTTPStatus.ok
        msg = 'OK'

    if verbose:
        print(cc, msg)

    return Result(status, cc)
# END FLAGS2_BASIC_HTTP_FUNCTIONS

# BEGIN FLAGS2_DOWNLOAD_MANY_SEQUENTIAL
def download_many(cc_list, base_url, verbose, max_req):
    counter = collections.Counter()
    cc_iter = sorted(cc_list)
    if not verbose:
        cc_iter = tqdm.tqdm(cc_iter)
    for cc in cc_iter:
        try:
            res = download_one(cc, base_url, verbose)
        except requests.exceptions.HTTPError as exc:
            error_msg = 'HTTP error {res.status_code} - {res.reason}'
            error_msg = error_msg.format(res=exc.response)
        except requests.exceptions.ConnectionError as exc:
            error_msg = 'Connection error'
        else:
            error_msg = ''
            status = res.status

        if error_msg:
            status = HTTPStatus.error
```

```
        counter[status] += 1
        if verbose and error_msg:
            print('*** Error for {}: {}'.format(cc, error_msg))

    return counter
# END FLAGS2_DOWNLOAD_MANY_SEQUENTIAL

if __name__ == '__main__':
    main(download_many, DEFAULT_CONCUR_REQ, MAX_CONCUR_REQ)
```

Chapter 19: OSCON Schedule Scripts and Tests

Example A-12 is the test script for the *schedule1.py* module (Example 19-9). It uses the `py.test` library and test runner.

Example A-12. test_schedule1.py

```
import shelve
import pytest

import schedule1 as schedule


@pytest.yield_fixture
def db():
    with shelve.open(schedule.DB_NAME) as the_db:
        if schedule.CONFERENCE not in the_db:
            schedule.load_db(the_db)
        yield the_db


def test_record_class():
    rec = schedule.Record(spam=99, eggs=12)
    assert rec.spam == 99
    assert rec.eggs == 12


def test_conference_record(db):
    assert schedule.CONFERENCE in db


def test_speaker_record(db):
    speaker = db['speaker.3471']
    assert speaker.name == 'Anna Martelli Ravenscroft'


def test_event_record(db):
    event = db['event.33950']
    assert event.name == 'There *Will* Be Bugs'


def test_event_venue(db):
```

```
    event = db['event.33950']
    assert event.venue_serial == 1449
```

Example A-13 is the full listing of the *schedule2.py* example presented in "Linked Record Retrieval with Properties" on page 598 in four parts.

Example A-13. schedule2.py

```
"""
schedule2.py: traversing OSCON schedule data

    >>> import shelve
    >>> db = shelve.open(DB_NAME)
    >>> if CONFERENCE not in db: load_db(db)

# BEGIN SCHEDULE2_DEMO

    >>> DbRecord.set_db(db)
    >>> event = DbRecord.fetch('event.33950')
    >>> event
    <Event 'There *Will* Be Bugs'>
    >>> event.venue
    <DbRecord serial='venue.1449'>
    >>> event.venue.name
    'Portland 251'
    >>> for spkr in event.speakers:
    ...     print('{0.serial}: {0.name}'.format(spkr))
    ...
    speaker.3471: Anna Martelli Ravenscroft
    speaker.5199: Alex Martelli

# END SCHEDULE2_DEMO

    >>> db.close()

"""

# BEGIN SCHEDULE2_RECORD
import warnings
import inspect

import osconfeed

DB_NAME = 'data/schedule2_db'
CONFERENCE = 'conference.115'


class Record:
    def __init__(self, **kwargs):
        self.__dict__.update(kwargs)

    def __eq__(self, other):
```

```
        if isinstance(other, Record):
            return self.__dict__ == other.__dict__
        else:
            return NotImplemented
# END SCHEDULE2_RECORD


# BEGIN SCHEDULE2_DBRECORD
class MissingDatabaseError(RuntimeError):
    """Raised when a database is required but was not set."""


class DbRecord(Record):

    __db = None

    @staticmethod
    def set_db(db):
        DbRecord.__db = db

    @staticmethod
    def get_db():
        return DbRecord.__db

    @classmethod
    def fetch(cls, ident):
        db = cls.get_db()
        try:
            return db[ident]
        except TypeError:
            if db is None:
                msg = "database not set; call '{}.set_db(my_db)'"
                raise MissingDatabaseError(msg.format(cls.__name__))
            else:  # ❶
                raise

    def __repr__(self):
        if hasattr(self, 'serial'):
            cls_name = self.__class__.__name__
            return '<{} serial={!r}>'.format(cls_name, self.serial)
        else:
            return super().__repr__()
# END SCHEDULE2_DBRECORD


# BEGIN SCHEDULE2_EVENT
class Event(DbRecord):

    @property
    def venue(self):
        key = 'venue.{}'.format(self.venue_serial)
        return self.__class__.fetch(key)
```

```
    @property
    def speakers(self):
        if not hasattr(self, '_speaker_objs'):
            spkr_serials = self.__dict__['speakers']
            fetch = self.__class__.fetch
            self._speaker_objs = [fetch('speaker.{}'.format(key))
                                  for key in spkr_serials]
        return self._speaker_objs

    def __repr__(self):
        if hasattr(self, 'name'):
            cls_name = self.__class__.__name__
            return '<{} {!r}>'.format(cls_name, self.name)
        else:
            return super().__repr__()
# END SCHEDULE2_EVENT


# BEGIN SCHEDULE2_LOAD
def load_db(db):
    raw_data = osconfeed.load()
    warnings.warn('loading ' + DB_NAME)
    for collection, rec_list in raw_data['Schedule'].items():
        record_type = collection[:-1]
        cls_name = record_type.capitalize()
        cls = globals().get(cls_name, DbRecord)
        if inspect.isclass(cls) and issubclass(cls, DbRecord):
            factory = cls
        else:
            factory = DbRecord
        for record in rec_list:
            key = '{}.{}'.format(record_type, record['serial'])
            record['serial'] = key
            db[key] = factory(**record)
# END SCHEDULE2_LOAD
```

Example A-14 was used to test Example A-13 with `py.test`.

Example A-14. test_schedule2.py

```
import shelve
import pytest

import schedule2 as schedule


@pytest.yield_fixture
def db():
    with shelve.open(schedule.DB_NAME) as the_db:
        if schedule.CONFERENCE not in the_db:
            schedule.load_db(the_db)
        yield the_db
```

```
def test_record_attr_access():
    rec = schedule.Record(spam=99, eggs=12)
    assert rec.spam == 99
    assert rec.eggs == 12


def test_record_repr():
    rec = schedule.DbRecord(spam=99, eggs=12)
    assert 'DbRecord object at 0x' in repr(rec)
    rec2 = schedule.DbRecord(serial=13)
    assert repr(rec2) == "<DbRecord serial=13>"


def test_conference_record(db):
    assert schedule.CONFERENCE in db


def test_speaker_record(db):
    speaker = db['speaker.3471']
    assert speaker.name == 'Anna Martelli Ravenscroft'


def test_missing_db_exception():
    with pytest.raises(schedule.MissingDatabaseError):
        schedule.DbRecord.fetch('venue.1585')


def test_dbrecord(db):
    schedule.DbRecord.set_db(db)
    venue = schedule.DbRecord.fetch('venue.1585')
    assert venue.name == 'Exhibit Hall B'


def test_event_record(db):
    event = db['event.33950']
    assert repr(event) == "<Event 'There *Will* Be Bugs'>"


def test_event_venue(db):
    schedule.Event.set_db(db)
    event = db['event.33950']
    assert event.venue_serial == 1449
    assert event.venue == db['venue.1449']
    assert event.venue.name == 'Portland 251'


def test_event_speakers(db):
    schedule.Event.set_db(db)
    event = db['event.33950']
    assert len(event.speakers) == 2
```

```
    anna_and_alex = [db['speaker.3471'], db['speaker.5199']]
    assert event.speakers == anna_and_alex


def test_event_no_speakers(db):
    schedule.Event.set_db(db)
    event = db['event.36848']
    assert len(event.speakers) == 0
```