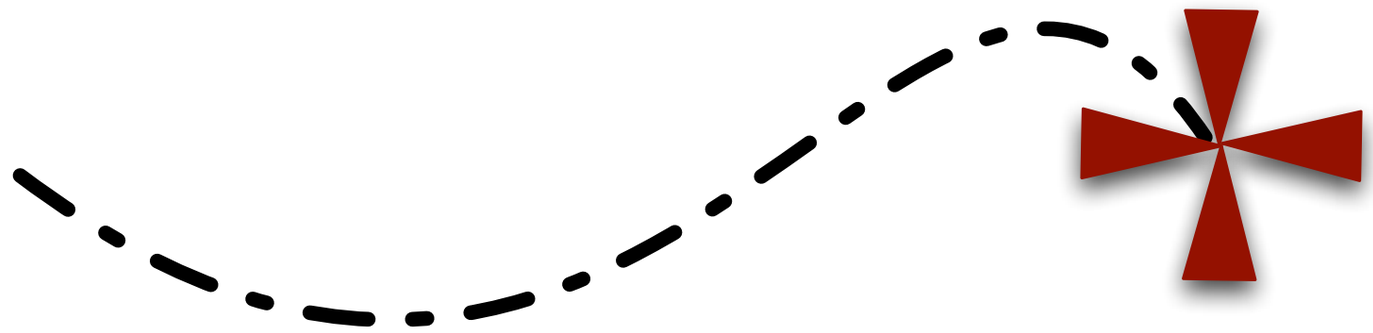
Hidden Treasures of the Standard Library

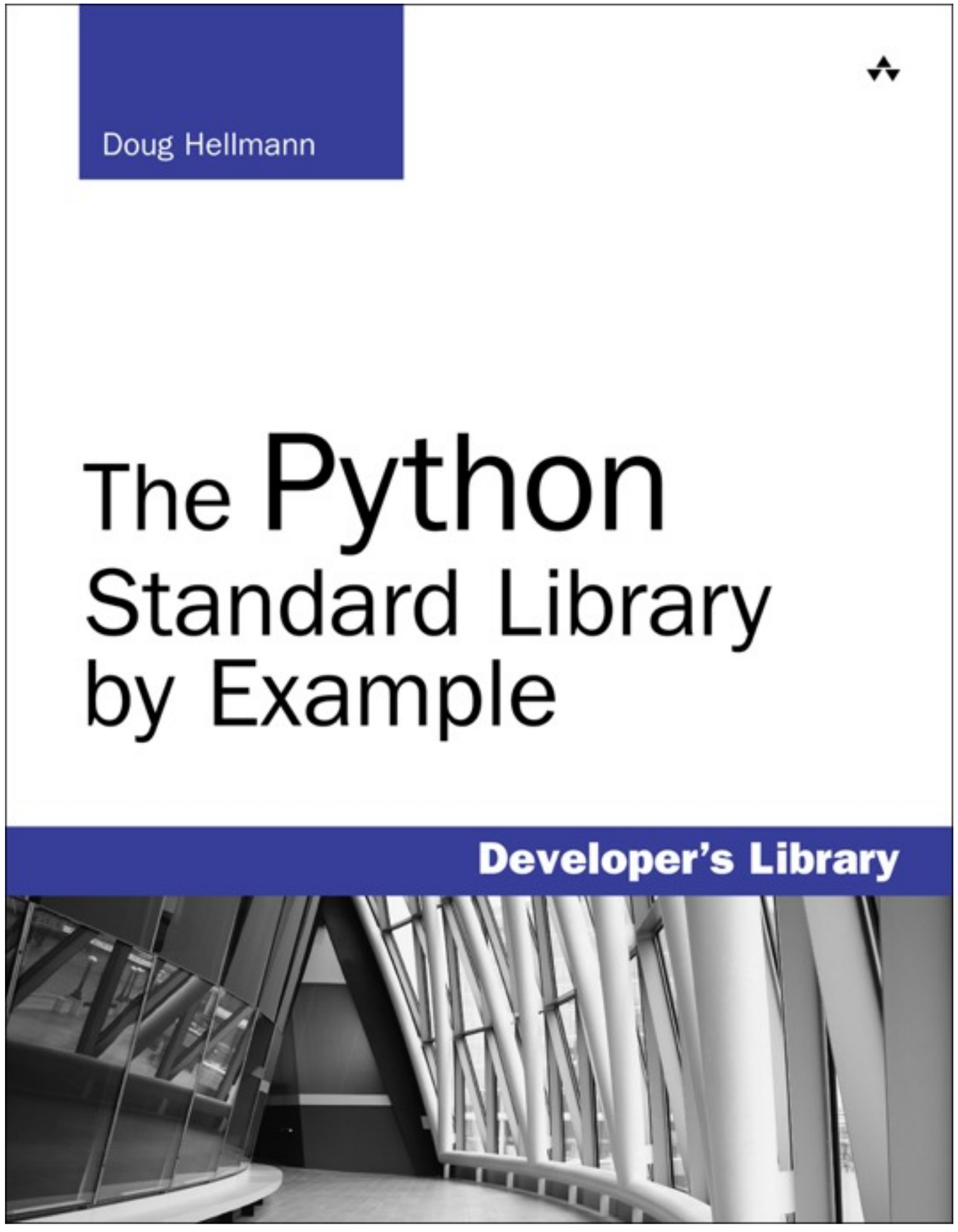
Doug Hellmann
PyCon
February, 2011

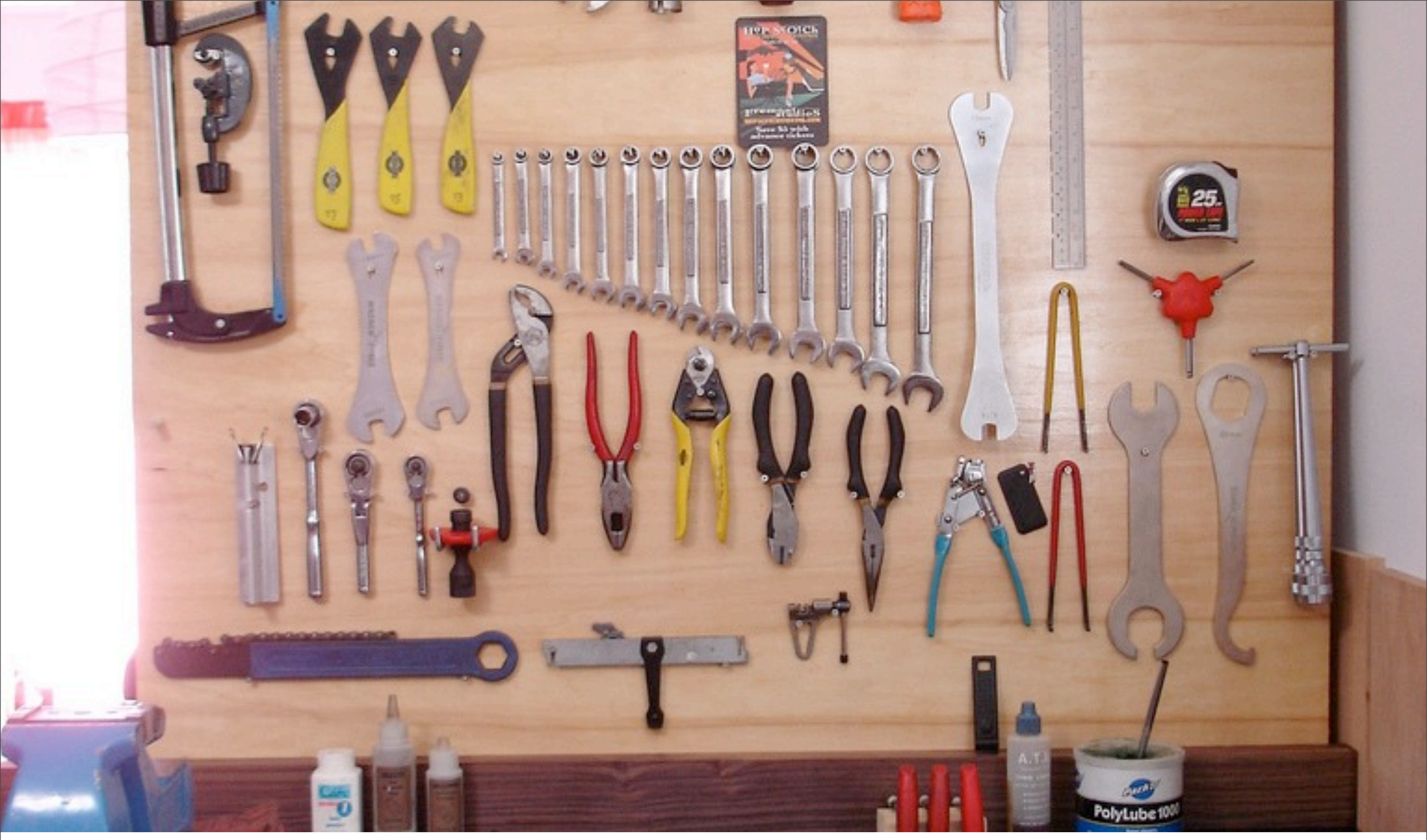


Python Module of the Week:
“I read the docs, so you don’t have to.”

Read This!

Pre-order now





Goals

Usable Tips

Look Around | Change Focus

Look Around

100+ Modules

Full Name	User Name	Email
Doug Hellmann	dhellmann	doug.hellmann@gmail.com
Guest account, no login	guest	guest@example.com

Parsing Structured Data

csv dialects

Test Database

```
#!/bin/sh

rm -f test.db

sqlite3 test.db <<EOF

CREATE TABLE users (
  fullname text,
  username text,
  email    text
);

INSERT INTO users (fullname, username, email)
VALUES ('Doug Hellmann', 'dhellmann', 'doug.hellmann@gmail.com');

INSERT INTO users (fullname, username, email)
VALUES ('Guest account, no login', 'guest', 'guest@example.com');

EOF

```

Test Database

```
#!/bin/sh

rm -f test.db

sqlite3 test.db <<EOF

CREATE TABLE users (
  fullname text,
  username text,
  email    text
);

INSERT INTO users (fullname, username, email)
VALUES ('Doug Hellmann', 'dhellmann', 'doug.hellmann@gmail.com');

INSERT INTO users (fullname, username, email)
VALUES ('Guest account, no login', 'guest', 'guest@example.com');

EOF

```

Database Contents

```
$ ./hidden_stdlib/csv_mkdb.sh


$ sqlite3 -noheader test.db 'select * from users'
Doug Hellmann|dhellmann|doug.hellmann@gmail.com
Guest account, no login|guest|guest@example.com
```

Database Contents

```
$ ./hidden_stdlib/csv_mkdb.sh


$ sqlite3 -noheader test.db 'select * from users'
Doug Hellmann|dhellmann|doug.hellmann@gmail.com
Guest account, no login|guest|guest@example.com
```

csv Dialect

```
 9 csv.register_dialect('pipes', delimiter='|')
10
11 reader = csv.reader(sys.stdin, dialect='pipes')
12 for row in reader:
13     print row
```

csv Dialect

```
 9 csv.register_dialect('pipes', delimiter='|')
10
11 reader = csv.reader(sys.stdin, dialect='pipes')
12 for row in reader:
13     print row
```

csv Dialect

```
 9 csv.register_dialect('pipes', delimiter='|')
10
11 reader = csv.reader(sys.stdin, dialect='pipes')
12 for row in reader:
13     print row
```

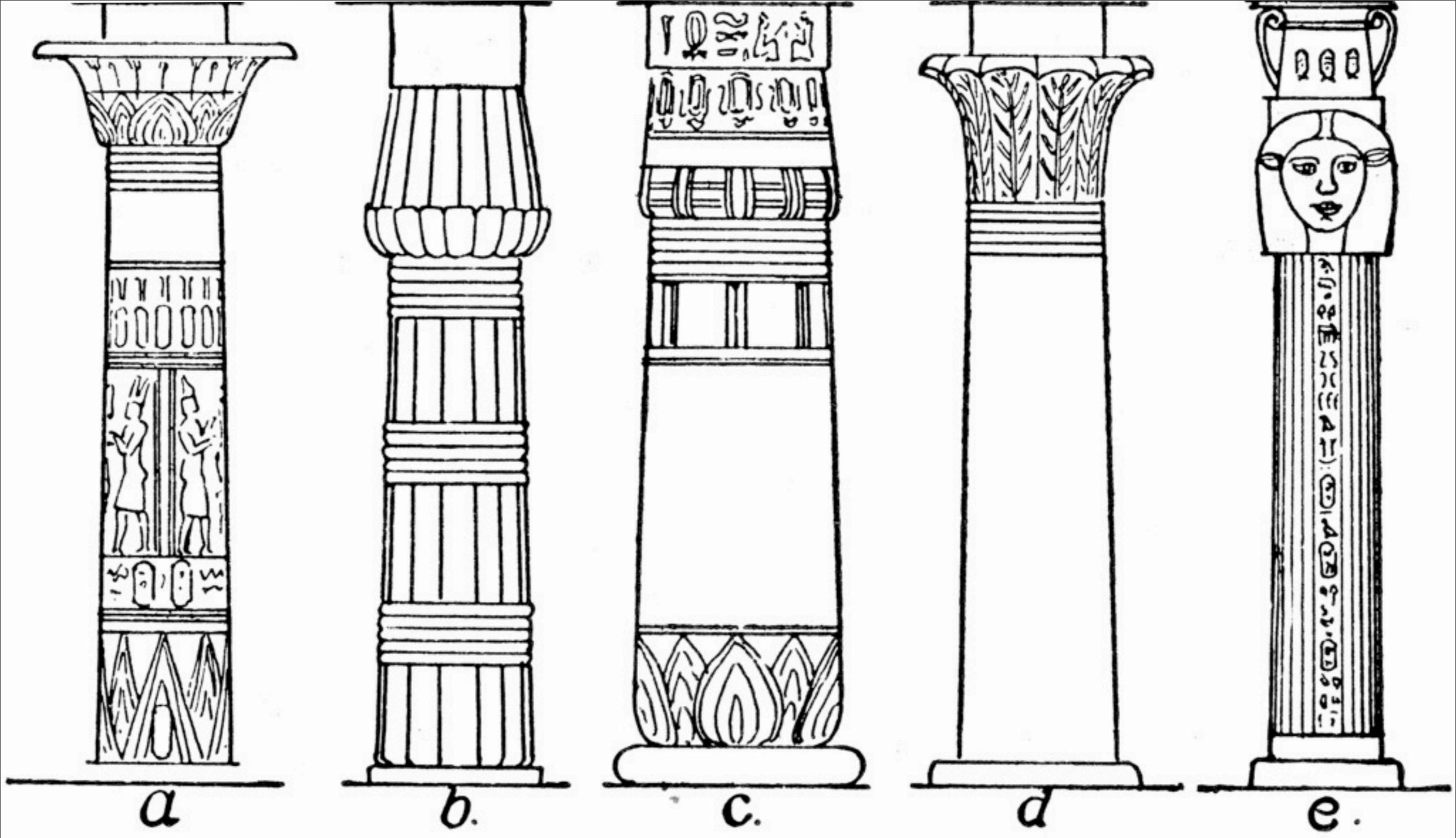
Database Contents

```
$ ./hidden_stdlib/csv_mkdb.sh


$ sqlite3 -noheader test.db 'select * from users'
Doug Hellmann|dhellmann|doug.hellmann@gmail.com
Guest account, no login|guest|guest@example.com
```

```
$ sqlite3 -noheader test.db 'select * from users' \
 | python ./hidden_stdlib/csv_pipes.py
['Doug Hellmann', 'dhellmann', 'doug.hellmann@gmail.com']
['Guest account, no login', 'guest', 'guest@example.com']
```

Database Contents

```
$ ./hidden_stdlib/csv_mkdb.sh


$ sqlite3 -noheader test.db 'select * from users'
Doug Hellmann|dhellmann|doug.hellmann@gmail.com
Guest account, no login|guest|guest@example.com


$ sqlite3 -noheader test.db 'select * from users' \
 | python ./hidden_stdlib/csv_pipes.py
['Doug Hellmann', 'dhellmann', 'doug.hellmann@gmail.com']
['Guest account, no login', 'guest', 'guest@example.com']
```



Custom Column Types

sqlite3

Custom SQLite Column Types

```
42      # Insert the objects into the database
43      print 'Inserting:'
44      to_save = [ (MyObj('this is a value to save'),),
45                  (MyObj(42),),
46                  ]
47      cursor.executemany(
48          "insert into obj (data) values (?)",
49          to_save)
50
51      # Query the database for the objects just saved
52      print '\nQuerying:'
53      cursor.execute("select id, data from obj")
54      for obj_id, obj in cursor.fetchall():
55          print 'Retrieved', obj_id, type(obj), obj
```

Custom SQLite Column Types

```
42      # Insert the objects into the database
43      print 'Inserting:'
44      to_save = [ (MyObj('this is a value to save'),),
45                  (MyObj(42),),
46                  ]
47      cursor.executemany(
48          "insert into obj (data) values (?)",
49          to_save)
50
51      # Query the database for the objects just saved
52      print '\nQuerying:'
53      cursor.execute("select id, data from obj")
54      for obj_id, obj in cursor.fetchall():
55          print 'Retrieved', obj_id, type(obj), obj
```

Custom SQLite Column Types

```
42      # Insert the objects into the database
43      print 'Inserting:'
44      to_save = [ (MyObj('this is a value to save'),),
45                  (MyObj(42),),
46                  ]
47      cursor.executemany(
48          "insert into obj (data) values (?)",
49          to_save)
50
51      # Query the database for the objects just saved
52      print '\nQuerying:'
53      cursor.execute("select id, data from obj")
54      for obj_id, obj in cursor.fetchall():
55          print 'Retrieved', obj_id, type(obj), obj
```

Custom SQLite Column Types

```
 9 class MyObj(object):
10     def __init__(self, arg):
11         self.arg = arg
12     def __str__(self):
13         return 'MyObj(%r)' % self.arg
```

Custom SQLite Column Types

```
17 def adapter_func(obj):
18     """memory -> storage"""
19     print 'Saving:', obj
20     return pickle.dumps(obj)
21
22 sqlite3.register_adapter(MyObj, adapter_func)
23
24 def converter_func(data):
25     """storage -> memory"""
26     return pickle.loads(data)
27
28 sqlite3.register_converter("MyObj", converter_func)
```

Custom SQLite Column Types

```
17 def adapter_func(obj):
18     """memory -> storage"""
19     print 'Saving:', obj
20     return pickle.dumps(obj)
21
22 sqlite3.register_adapter(MyObj, adapter_func)
23
24 def converter_func(data):
25     """storage -> memory"""
26     return pickle.loads(data)
27
28 sqlite3.register_converter("MyObj", converter_func)
```

Custom SQLite Column Types

```
17 def adapter_func(obj):
18     """memory -> storage"""
19     print 'Saving:', obj
20     return pickle.dumps(obj)
21
22 sqlite3.register_adapter(MyObj, adapter_func)
23
24 def converter_func(data):
25     """storage -> memory"""
26     return pickle.loads(data)
27
28 sqlite3.register_converter("MyObj", converter_func)
```

Custom SQLite Column Types

```
17 def adapter_func(obj):
18     """memory -> storage"""
19     print 'Saving:', obj
20     return pickle.dumps(obj)
21
22 sqlite3.register_adapter(MyObj, adapter_func)
23
24 def converter_func(data):
25     """storage -> memory"""
26     return pickle.loads(data)
27
28 sqlite3.register_converter("MyObj", converter_func)
```

Custom SQLite Column Types

```
30 with sqlite3.connect(
31         'type_demo.db',
32         detect_types=sqlite3.PARSE_DECLTYPES) as conn:
33     # Create a table with column of type "MyObj"
34     conn.execute("""
35     create table if not exists obj (
36         id    integer primary key autoincrement not null,
37         data  MyObj
38     )
39     """)
```

Custom SQLite Column Types

```
30 with sqlite3.connect(
31         'type_demo.db',
32         detect_types=sqlite3.PARSE_DECLTYPES) as conn:
33     # Create a table with column of type "MyObj"
34     conn.execute("""
35     create table if not exists obj (
36         id    integer primary key autoincrement not null,
37         data  MyObj
38     )
39     """)
```

Custom SQLite Column Types

```
$ python hidden_stdlib/sqlite3_custom_type.py
Inserting:
Saving: MyObj('this is a value to save')
Saving: MyObj(42)

Querying:
Retrieved 1 <class '__main__.MyObj'> MyObj('this is a value to
save')
Retrieved 2 <class '__main__.MyObj'> MyObj(42)
```

Custom SQLite Column Types

```
$ python hidden_stdlib/sqlite3_custom_type.py
Inserting:
Saving: MyObj('this is a value to save')
Saving: MyObj(42)

Querying:
Retrieved 1 <class '__main__.MyObj'> MyObj('this is a value to
save')
Retrieved 2 <class '__main__.MyObj'> MyObj(42)
```

Custom SQLite Column Types

```
$ python hidden_stdlib/sqlite3_custom_type.py
Inserting:
Saving: MyObj('this is a value to save')
Saving: MyObj(42)

Querying:
Retrieved 1 <class '__main__.MyObj'> MyObj('this is a value to
save')
Retrieved 2 <class '__main__.MyObj'> MyObj(42)
```

Signed Serialized Data

hmac

hmac

```
11 message = 'The message'
12 encoded_message = pickle.dumps(message)
13
14 digest_maker = hmac.new('shared-secret-value')
15 digest_maker.update(encoded_message)
16 signature = digest_maker.hexdigest()
17
18 print 'Outgoing signature:', signature
19
20 # Simulate sending the message
21 buffer = StringIO('%s\n%d\n%s' % (signature,
22                                   len(encoded_message),
23                                   encoded_message
24                                   ))
```

hmac

```
11 message = 'The message'
12 encoded_message = pickle.dumps(message)
13
14 digest_maker = hmac.new('shared-secret-value')
15 digest_maker.update(encoded_message)
16 signature = digest_maker.hexdigest()
17
18 print 'Outgoing signature:', signature
19
20 # Simulate sending the message
21 buffer = StringIO('%s\n%d\n%s' % (signature,
22                                   len(encoded_message),
23                                   encoded_message
24                                   ))
```

hmac

```
11 message = 'The message'
12 encoded_message = pickle.dumps(message)
13
14 digest_maker = hmac.new('shared-secret-value')
15 digest_maker.update(encoded_message)
16 signature = digest_maker.hexdigest()
17
18 print 'Outgoing signature:', signature
19
20 # Simulate sending the message
21 buffer = StringIO('%s\n%d\n%s' % (signature,
22                                   len(encoded_message),
23                                   encoded_message
24                                   ))
```

hmac

```
26 # "Receive" the message
27 read_signature = buffer.readline().rstrip()
28 message_len = int(buffer.readline())
29 read_message = buffer.read(message_len)
30
31 # Check the signature of the incoming data
32 digest_maker = hmac.new('shared-secret-value',
33                         read_message)
34 computed_signature = digest_maker.hexdigest()
35 print 'Computed signature:', signature
```

hmac

```
26 # "Receive" the message
27 read_signature = buffer.readline().rstrip()
28 message_len = int(buffer.readline())
29 read_message = buffer.read(message_len)
30
31 # Check the signature of the incoming data
32 digest_maker = hmac.new('shared-secret-value',
33                         read_message)
34 computed_signature = digest_maker.hexdigest()
35 print 'Computed signature:', signature
```

hmac

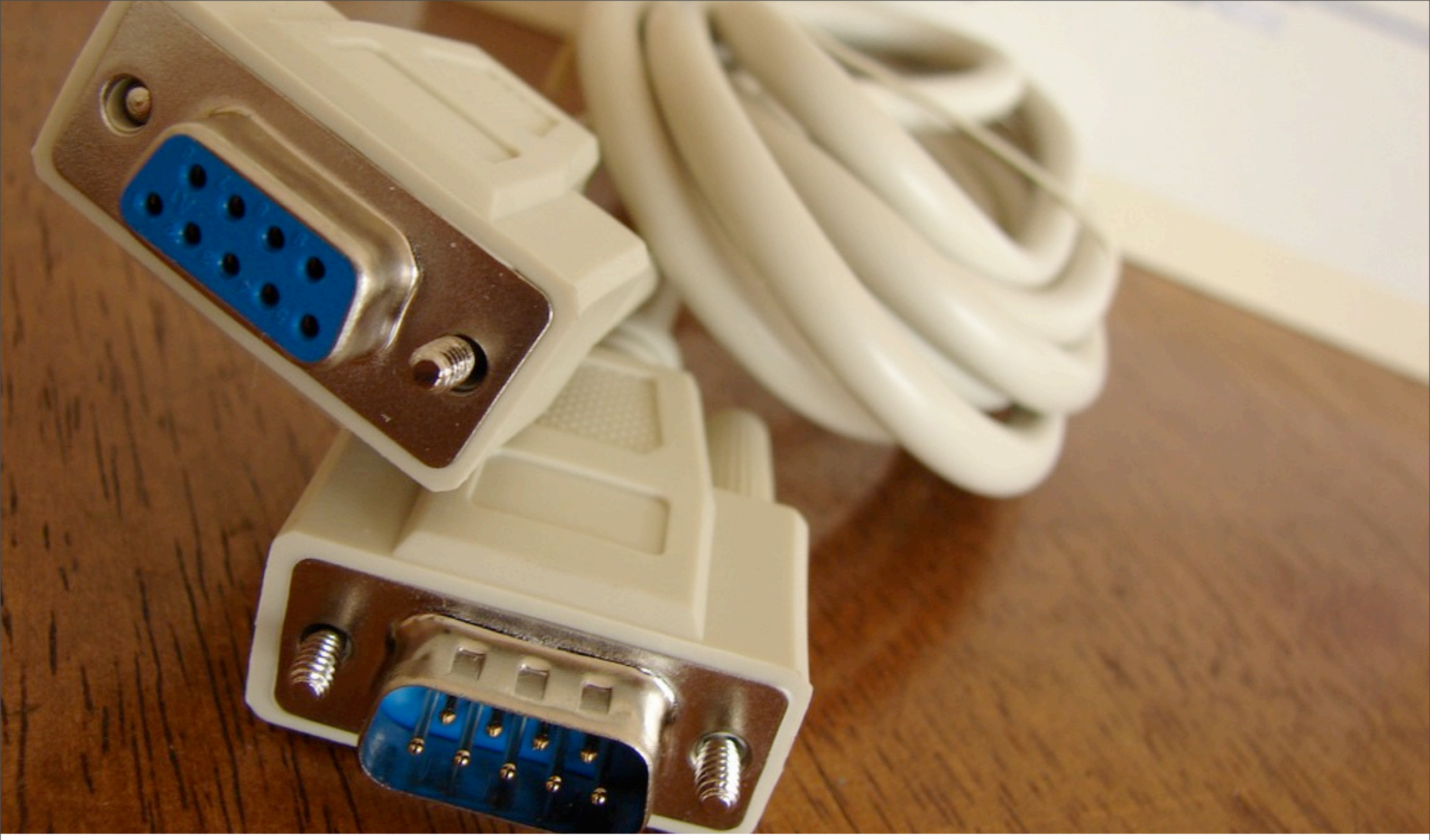
```
37 if computed_signature == read_signature:
38     print '\nValid message, processed'
39     safe_message = pickle.loads(read_message)
40     print 'Message:', safe_message
41 else:
42     raise ValueError('Invalid message, discarded')
```

hmac

```
37 if computed_signature == read_signature:
38     print '\nValid message, processed'
39     safe_message = pickle.loads(read_message)
40     print 'Message:', safe_message
41 else:
42     raise ValueError('Invalid message, discarded')
```

hmac

```
$ python hidden_stdlib/hmac_message_signing.py
Outgoing signature: 3498b6fa42a75dc4603da6f4c67505a2
Computed signature: 3498b6fa42a75dc4603da6f4c67505a2

Valid message, processed
Message: The message
```



Serializing Objects

json

json

```
10 def convert_to_builtin_type(obj):
11     """object->dictionary"""
12     print 'convert to builtin type(%r)' % obj
13     class_name = obj.__class__.__name__
14     module_name = obj.__module__
15     digest_maker = hmac.new('PyCon2011',
16                             module_name + class_name)
17     signature = digest_maker.hexdigest()
18     d = { '__class__':class_name,
19           '__module__':module_name,
20           '__signature__':signature,
21           }
22     d.update(obj.__dict__)
23     return d
24
25 obj = json_myobj.MyObj('instance value goes here')
26 print json.dumps(obj, default=convert_to_builtin_type)
```

json

```
10 def convert_to_builtin_type(obj):
11     """object->dictionary"""
12     print 'convert_to_builtin_type(%r)' % obj
13     class_name = obj.__class__.__name__
14     module_name = obj.__module__
15     digest_maker = hmac.new('PyCon2011',
16                             module_name + class_name)
17     signature = digest_maker.hexdigest()
18     d = { '__class__':class_name,
19           '__module__':module_name,
20           '__signature__':signature,
21           }
22     d.update(obj.__dict__)
23     return d
24
25 obj = json_myobj.MyObj('instance value goes here')
26 print json.dumps(obj, default=convert_to_builtin_type)
```

json

```
10 def convert_to_builtin_type(obj):
11     """object->dictionary"""
12     print 'convert_to_builtin_type(%r)' % obj
13     class_name = obj.__class__.__name__
14     module_name = obj.__module__
15     digest_maker = hmac.new('PyCon2011',
16                             module_name + class_name)
17     signature = digest_maker.hexdigest()
18     d = { '__class__':class_name,
19           '__module__':module_name,
20           '__signature__':signature,
21           }
22     d.update(obj.__dict__)
23     return d
24
25 obj = json_myobj.MyObj('instance value goes here')
26 print json.dumps(obj, default=convert_to_builtin_type)
```

json

```
10 def convert_to_builtin_type(obj):
11     """object->dictionary"""
12     print 'convert_to_builtin_type(%r)' % obj
13     class_name = obj.__class__.__name__
14     module_name = obj.__module__
15     digest_maker = hmac.new('PyCon2011',
16                             module_name + class_name)
17     signature = digest_maker.hexdigest()
18     d = { '__class__':class_name,
19           '__module__':module_name,
20           '__signature__':signature,
21           }
22     d.update(obj.__dict__)
23     return d
24
25 obj = json_myobj.MyObj('instance value goes here')
26 print json.dumps(obj, default=convert_to_builtin_type)
```

json

```
$ python hidden_stdlib/json_dump_default.py
convert_to_builtin_type(<MyObj(instance value goes here)>)
{"s": "instance value goes here", "__module__": "json_myobj",
"__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
"__class__": "MyObj"}
```

json

```
$ python hidden_stdlib/json_dump_default.py
convert_to_builtin_type(<MyObj(instance value goes here)>)
{"s": "instance value goes here", "__module__": "json_myobj",
"__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
"__class__": "MyObj"}
```

json

```
$ python hidden_stdlib/json_dump_default.py
convert_to_builtin_type(<MyObj(instance value goes here)>)
{"s": "instance value goes here", "__module__": "json_myobj",
"__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
"__class__": "MyObj"}
```

json

```
$ python hidden_stdlib/json_dump_default.py
convert_to_builtin_type(<MyObj(instance value goes here)>)
{"s": "instance value goes here", "__module__": "json_myobj",
"__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
"__class__": "MyObj"}
```

json

```
$ python hidden_stdlib/json_dump_default.py
convert_to_builtin_type(<MyObj(instance value goes here)>)
{"s": "instance value goes here", "__module__": "json_myobj",
"__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
"__class__": "MyObj"}
```

json

```
 9 def dict_to_object(d):
10     if '__class__' not in d:
11         return d
12
13     class_name = d.pop('__class__')
14     module_name = d.pop('__module__')
15     signature = d.pop('__signature__')
16
17     digest_maker = hmac.new('PyCon2011',
18                             module_name + class_name)
19     expected_signature = digest_maker.hexdigest()
20     if signature != expected_signature:
21         raise ValueError('Invalid signature')
```

json

```
 9 def dict_to_object(d):
10     if '__class__' not in d:
11         return d
12
13     class_name = d.pop('__class__')
14     module_name = d.pop('__module__')
15     signature = d.pop('__signature__')
16
17     digest_maker = hmac.new('PyCon2011',
18                             module_name + class_name)
19     expected_signature = digest_maker.hexdigest()
20     if signature != expected_signature:
21         raise ValueError('Invalid signature')
```

json

```
 9 def dict_to_object(d):
10     if '__class__' not in d:
11         return d
12
13     class_name = d.pop('__class__')
14     module_name = d.pop('__module__')
15     signature = d.pop('__signature__')
16
17     digest_maker = hmac.new('PyCon2011',
18                             module_name + class_name)
19     expected_signature = digest_maker.hexdigest()
20     if signature != expected_signature:
21         raise ValueError('Invalid signature')
```

json

```
23      print 'Loading "%s" from "%s"' % \
24          (class_name, module_name)
25      module = __import__(module_name)
26      class_ = getattr(module, class_name)
27
28      args = dict( (key.encode('ascii'), value)
29                   for key, value in d.items())
30      print 'Instantiating with', args
31
32      inst = class_(**args)
33      return inst
```

json

```
23      print 'Loading "%s" from "%s"' % \
24          (class_name, module_name)
25      module =   import  (module_name)
26      class_ = getattr(module, class_name)
27
28      args = dict( (key.encode('ascii'), value)
29                   for key, value in d.items())
30      print 'Instantiating with', args
31
32      inst = class_(**args)
33      return inst
```

json

```
23      print 'Loading "%s" from "%s"' % \
24          (class_name, module_name)
25      module = __import__(module_name)
26      class_ = getattr(module, class_name)
27
28      args = dict( (key.encode('ascii'), value)
29                   for key, value in d.items())
30      print 'Instantiating with', args
31
32      inst = class_(**args)
33      return inst
```

json

```
23      print 'Loading "%s" from "%s"' % \
24          (class_name, module_name)
25      module = __import__(module_name)
26      class_ = getattr(module, class_name)
27
28      args = dict( (key.encode('ascii'), value)
29                   for key, value in d.items())
30      print 'Instantiating with', args
31
32      inst = class_(**args)
33      return inst
```

json

```
35 for encoded_object in [
36     '''
37     [{"s": "instance value goes here",
38       "__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
39       "__module__": "json_myobj", "__class__": "MyObj"}]
40     ''',
41
42     # careful!
43     '''
44     [{"path": "/etc/passwd",
45       "__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
46       "__module__": "os", "__class__": "unlink"}]
47     ''',
48     ]:
```

json

```
35 for encoded_object in [
36     '''
37     [{"s": "instance value goes here",
38       "__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
39       "__module__": "json_myobj", "__class__": "MyObj"}]
40     ''',
41
42     # careful!
43     '''
44     [{"path": "/etc/passwd",
45       "__signature__": "426f662f9fe3b3533d9ce7f9dcf8af77",
46       "__module__": "os", "__class__": "unlink"}]
47     ''',
48     ]:
```

json

```
49      try:
50          myobj_instance = json.loads(
51              encoded_object,
52              object_hook=dict_to_object,
53              )
54          print myobj_instance
55      except Exception as err:
56          print 'ERROR:', err
57      print
```

json

```
$ python hidden_stdlib/json_load_object_hook.py
Loading "MyObj" from "json_myobj"
Instantiating with {'s': u'instance value goes here'}
[<MyObj(instance value goes here)>]


ERROR: Invalid signature
```

json

```
$ python hidden_stdlib/json_load_object_hook.py
Loading "MyObj" from "json_myobj"
Instantiating with {'s': u'instance value goes here'}
[<MyObj(instance value goes here)>]
```

```
ERROR: Invalid signature
```

Error Handling

sys.excepthook

sys.excepthook

```
 6 def main():
 7     # do some work
 8     raise RuntimeError('Helpful error message')
 9
10 if __name__ == '__main__':
11     main()
```

```
$ python hidden_stdlib/sys_excepthook_no_handler.py
Traceback (most recent call last):
  File ".../hidden_stdlib/sys_excepthook_no_handler.py", line 11, in <module>
    main()
  File ".../hidden_stdlib/sys_excepthook_no_handler.py", line 8, in main
    raise RuntimeError('Helpful error message')
RuntimeError: Helpful error message
```

sys.excepthook

```
 6 import sys
 7
 8 def main():
 9     try:
10         # do some work
11         raise RuntimeError('Helpful error message')
12     except Exception as err:
13         sys.stderr.write('ERROR: %s\n' % err)
14
15 if __name__ == '__main__':
16     main()
```

```
$ python hidden_stdlib/sys_excepthook_big_block.py
ERROR: Helpful error message
```

sys.excepthook

```
 6 import sys
 7
 8 def quiet_errors(exc_type, exc_value, traceback):
 9     sys.stderr.write('ERROR: %s\n' % exc_value)
10
11 sys.excepthook = quiet_errors
12
13 def main():
14     # do some work
15     raise RuntimeError('Helpful error message')
16
17 if __name__ == '__main__':
18     main()
```

```
$ python hidden_stdlib/sys_excepthook.py

ERROR: Helpful error message
```

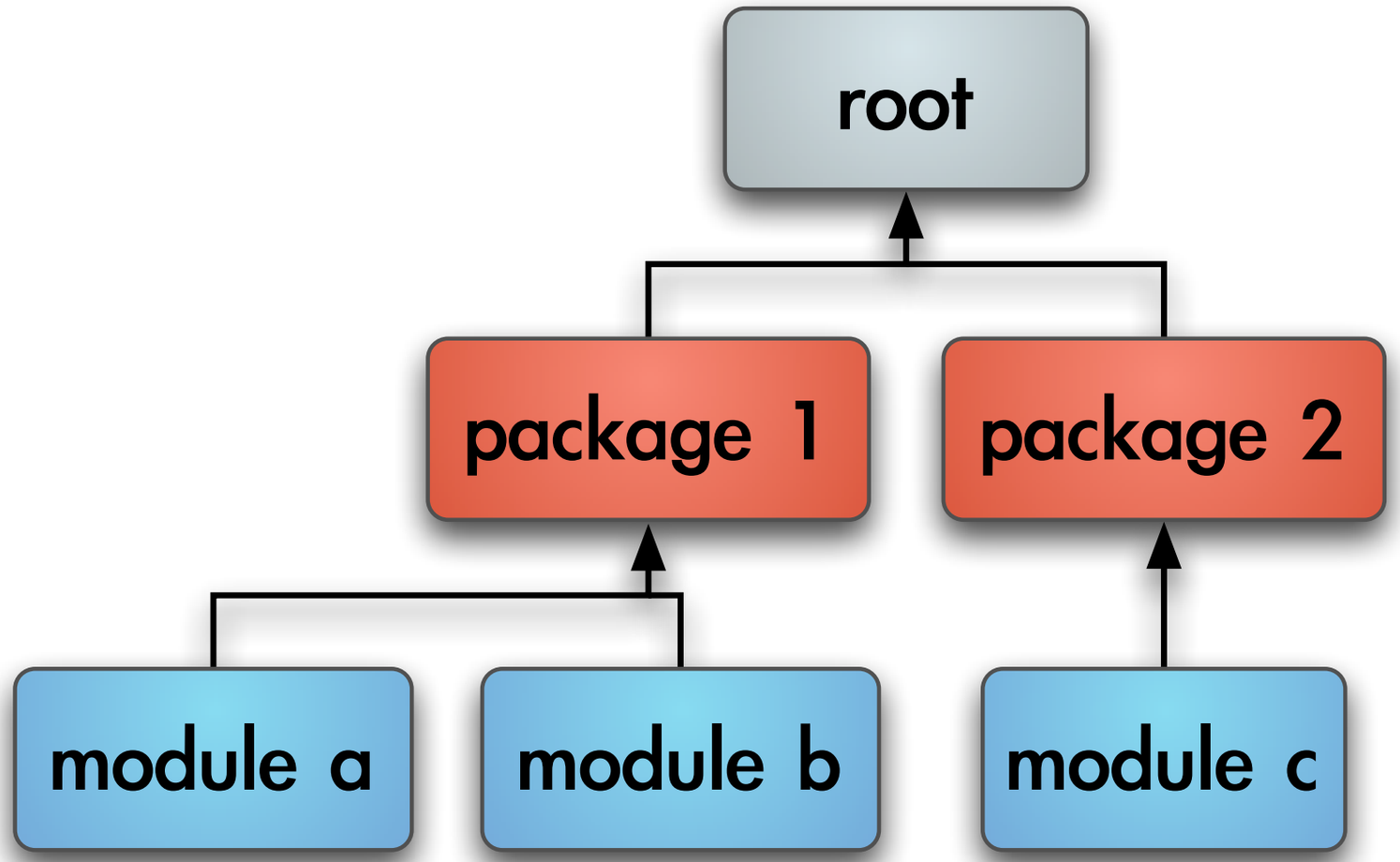
sys.excepthook

```
 6 import sys
 7
 8 def quiet_errors(exc_type, exc_value, traceback):
 9     sys.stderr.write('ERROR: %s\n' % exc_value)
10
11 sys.excepthook = quiet_errors
12
13 def main():
14     # do some work
15     raise RuntimeError('Helpful error message')
16
17 if __name__ == '__main__':
18     main()
```

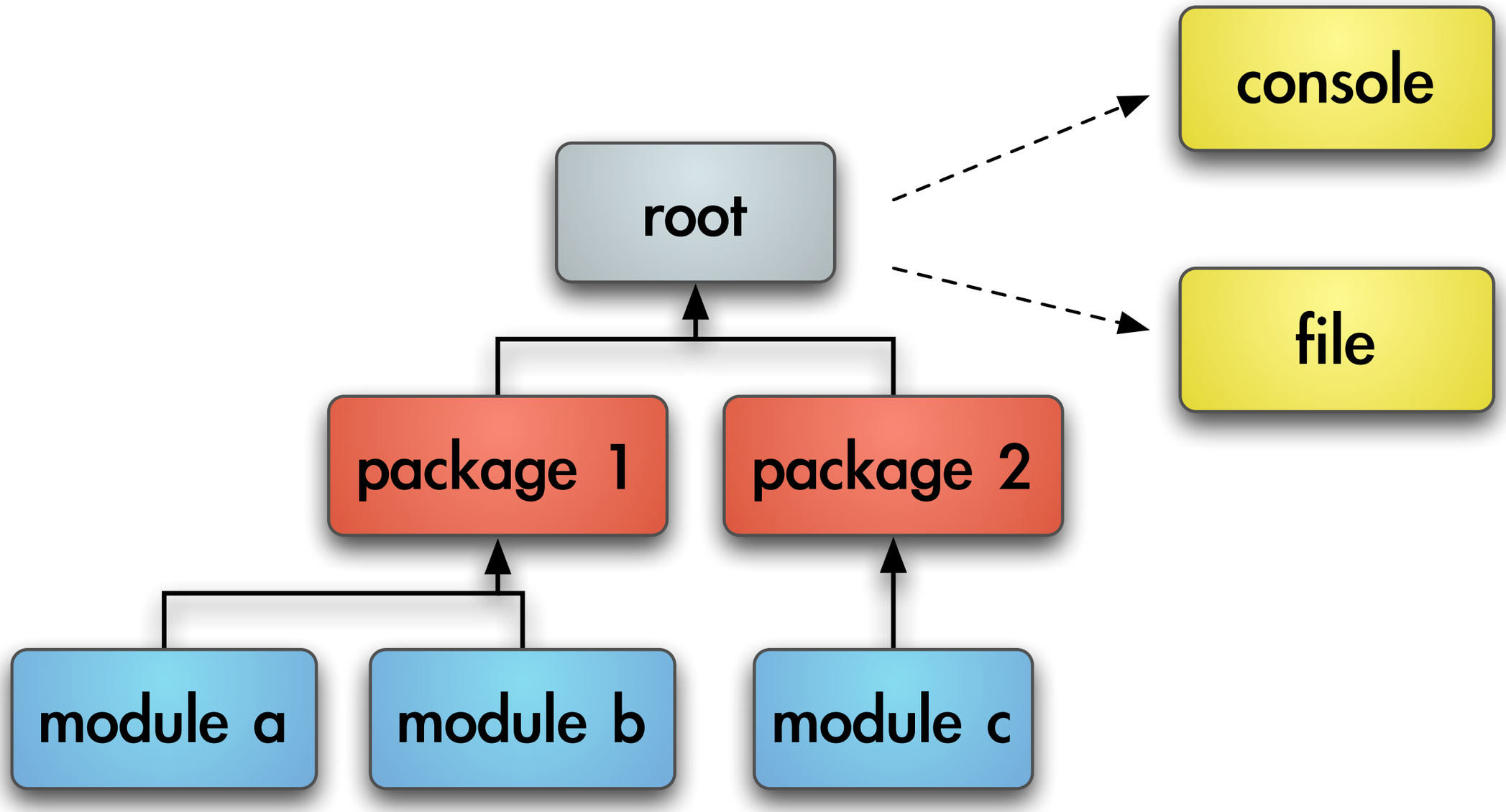
```
$ python hidden_stdlib/sys_excepthook.py

ERROR: Helpful error message
```

Communicating with Users

logging



logging | loggers



logging | loggers

logging

```
10 # Log verbosely
11 root_logger = logging.getLogger('')
12 root_logger.setLevel(logging.DEBUG)
13
14 # Set up console output to stderr
15 console = logging.StreamHandler(sys.stderr)
16 console_format = '%(message)s'
17 console.setFormatter(logging.Formatter(console_format))
18 console.setLevel(logging.INFO) # TODO: command line switch
19 root_logger.addHandler(console)
```

logging

```
10 # Log verbosely
11 root_logger = logging.getLogger('')
12 root_logger.setLevel(logging.DEBUG)
13
14 # Set up console output to stderr
15 console = logging.StreamHandler(sys.stderr)
16 console_format = '%(message)s'
17 console.setFormatter(logging.Formatter(console_format))
18 console.setLevel(logging.INFO) # TODO: command line switch
19 root_logger.addHandler(console)
```

logging

```
10 # Log verbosely
11 root_logger = logging.getLogger('')
12 root_logger.setLevel(logging.DEBUG)
13
14 # Set up console output to stderr
15 console = logging.StreamHandler(sys.stderr)
16 console_format = '%(message)s'
17 console.setFormatter(logging.Formatter(console_format))
18 console.setLevel(logging.INFO) # TODO: command line switch
19 root_logger.addHandler(console)
```

logging

```
10 # Log verbosely
11 root_logger = logging.getLogger('')
12 root_logger.setLevel(logging.DEBUG)
13
14 # Set up console output to stderr
15 console = logging.StreamHandler(sys.stderr)
16 console_format = '%(message)s'
17 console.setFormatter(logging.Formatter(console_format))
18 console.setLevel(logging.INFO) # TODO: command line switch
19 root_logger.addHandler(console)
```

logging

```
10 # Log verbosely
11 root_logger = logging.getLogger('')
12 root_logger.setLevel(logging.DEBUG)
13
14 # Set up console output to stderr
15 console = logging.StreamHandler(sys.stderr)
16 console_format = '%(message)s'
17 console.setFormatter(logging.Formatter(console_format))
18 console.setLevel(logging.INFO) # TODO: command line switch
19 root_logger.addHandler(console)
```

logging

```
21 # Include debug messages when logging to a file
22 file_handler = logging.handlers.RotatingFileHandler(
23     'logging_example.log', # use a full path
24     )
25 file_format = '%(asctime)s %(levelname)6s %(name)s %
(message)s'
26 file_handler.setFormatter(logging.Formatter(file_format))
27 file_handler.setLevel(logging.DEBUG)
28 root_logger.addHandler(file_handler)
```

logging

```
21 # Include debug messages when logging to a file
22 file_handler = logging.handlers.RotatingFileHandler(
23     'logging_example.log', # use a full path
24     )
25 file_format = '%(asctime)s %(levelname)6s %(name)s %
(message)s'
26 file_handler.setFormatter(logging.Formatter(file_format))
27 file_handler.setLevel(logging.DEBUG)
28 root_logger.addHandler(file_handler)
```

logging

```
21 # Include debug messages when logging to a file
22 file_handler = logging.handlers.RotatingFileHandler(
23     'logging_example.log', # use a full path
24     )
25 file_format = '%(asctime)s %(levelname)6s %(name)s %
(message)s'
26 file_handler.setFormatter(logging.Formatter(file_format))
27 file_handler.setLevel(logging.DEBUG)
28 root_logger.addHandler(file_handler)
```

logging

```
30 # Log sample messages with different levels
31 log = logging.getLogger(__name__)
32 log.info('on the console and in the file')
33 log.debug('only in the file')
34 log.error('simple error message')
35
36 # Replace excepthook with logger
37 def log_exception(exc_type, exc_value, traceback):
38     logging.getLogger(__name__).error(exc_value)
39 sys.excepthook = log_exception
40
41 # Send exceptions to the logger automatically
42 raise RuntimeError('failure message')
```

logging

```
30 # Log sample messages with different levels
31 log = logging.getLogger(  name  )
32 log.info('on the console and in the file')
33 log.debug('only in the file')
34 log.error('simple error message')
35
36 # Replace excepthook with logger
37 def log_exception(exc_type, exc_value, traceback):
38     logging.getLogger(__name__).error(exc_value)
39 sys.excepthook = log_exception
40
41 # Send exceptions to the logger automatically
42 raise RuntimeError('failure message')
```

logging

```
30 # Log sample messages with different levels
31 log = logging.getLogger(__name__)
32 log.info('on the console and in the file')
33 log.debug('only in the file')
34 log.error('simple error message')
35
36 # Replace excepthook with logger
37 def log_exception(exc_type, exc_value, traceback):
38     logging.getLogger(__name__).error(exc_value)
39 sys.excepthook = log_exception
40
41 # Send exceptions to the logger automatically
42 raise RuntimeError('failure message')
```

logging

```
$ python hidden_stdlib/logging_example.py
on the console and in the file
simple error message
failure message
```

logging

```
$ python hidden_stdlib/logging_example.py
on the console and in the file
simple error message
failure message

$ cat logging_example.log
2011-02-13 11:28:30,036   INFO __main__ on the console and in
the file
2011-02-13 11:28:30,036  DEBUG __main__ only in the file
2011-02-13 11:28:30,036  ERROR __main__ simple error message
2011-02-13 11:28:30,036  ERROR __main__ failure message
```

logging

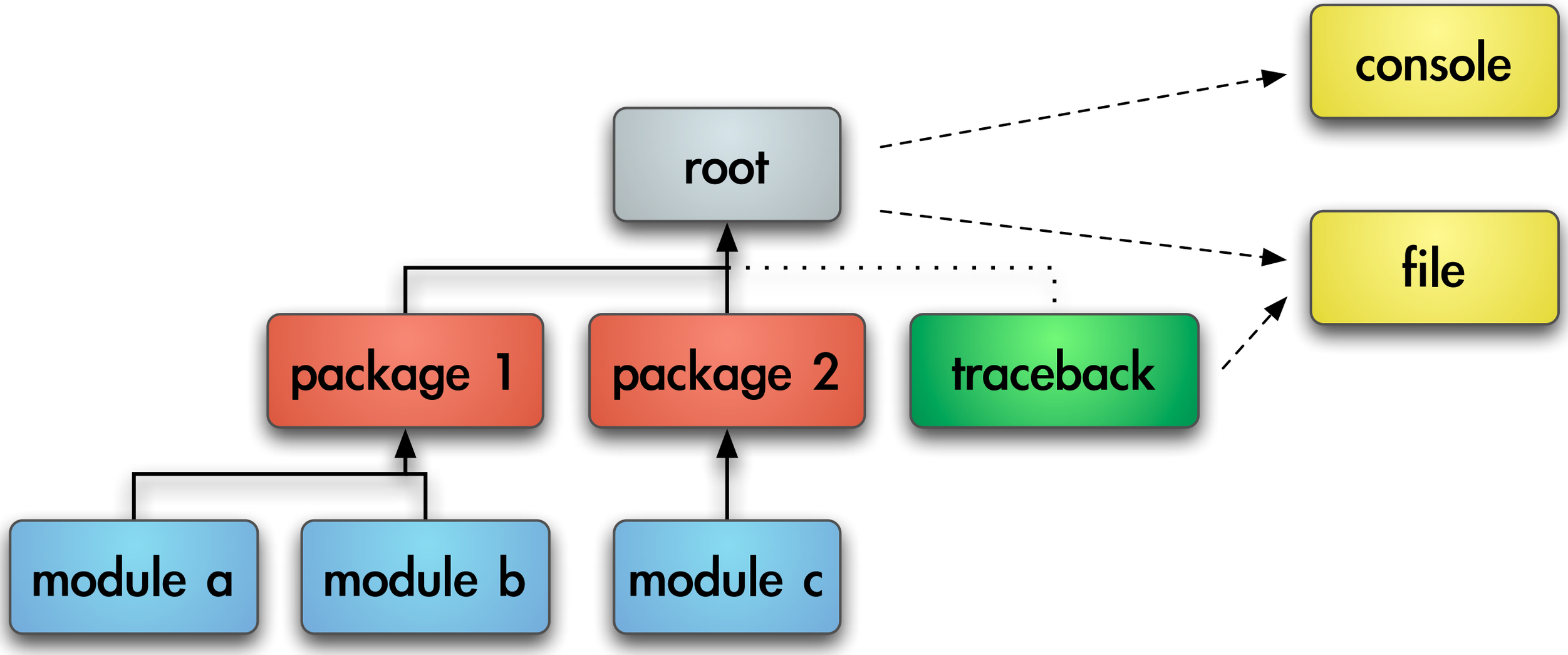
```
$ python hidden_stdlib/logging_example.py
on the console and in the file
simple error message
failure message

$ cat logging_example.log
2011-02-13 11:28:30,036  INFO __main__ on the console and in
the file
2011-02-13 11:28:30,036 DEBUG __main__ only in the file
2011-02-13 11:28:30,036 ERROR __main__ simple error message
2011-02-13 11:28:30,036 ERROR __main__ failure message
```

logging

```
$ python hidden_stdlib/logging_example.py
on the console and in the file
simple error message
failure message

$ cat logging_example.log
2011-02-13 11:28:30,036   INFO __main__ on the console and in
the file
2011-02-13 11:28:30,036  DEBUG __main__ only in the file
2011-02-13 11:28:30,036  ERROR __main__ simple error message
2011-02-13 11:28:30,036  ERROR __main__ failure message
```

logging

```
$ python hidden_stdlib/logging_example.py
on the console and in the file
simple error message
failure message

$ cat logging_example.log
2011-02-13 11:28:30,036   INFO __main__ on the console and in
the file
2011-02-13 11:28:30,036  DEBUG __main__ only in the file
2011-02-13 11:28:30,036  ERROR __main__ simple error message
2011-02-13 11:28:30,036  ERROR __main__ failure message
```



logging

Traceback Handler

logging

```
36 # Tracebacks should only go to the file
37 traceback_log = logging.getLogger('traceback')
38 traceback_log.propagate = False
39 traceback_log.setLevel(logging.ERROR)
40 traceback_log.addHandler(file_handler)
41
42 # Replace excepthook with logger
43 def log_exception(exc_type, exc_value, traceback):
44     logging.getLogger(__name__).error(exc_value)
45     logging.getLogger('traceback').error(
46         exc_value,
47         exc_info=(exc_type, exc_value, traceback),
48         )
49 sys.excepthook = log_exception
50
51 # Send exceptions to the logger automatically
52 raise RuntimeError('failure message')
```

logging

```
36 # Tracebacks should only go to the file
37 traceback_log = logging.getLogger('traceback')
38 traceback_log.propagate = False
39 traceback_log.setLevel(logging.ERROR)
40 traceback_log.addHandler(file_handler)
41
42 # Replace excepthook with logger
43 def log_exception(exc_type, exc_value, traceback):
44     logging.getLogger(__name__).error(exc_value)
45     logging.getLogger('traceback').error(
46         exc_value,
47         exc_info=(exc_type, exc_value, traceback),
48         )
49 sys.excepthook = log_exception
50
51 # Send exceptions to the logger automatically
52 raise RuntimeError('failure message')
```

logging

```
$ python hidden_stdlib/logging_example_tracebacks.py
on the console and in the file
simple error message
failure message

$ cat logging_example.log
2011-02-13 11:33:22,592   INFO __main__ on the console and in the file
2011-02-13 11:33:22,592  DEBUG __main__ only in the file
2011-02-13 11:33:22,592  ERROR __main__ simple error message
2011-02-13 11:33:22,592  ERROR __main__ failure message
2011-02-13 11:33:22,593  ERROR traceback failure message
Traceback (most recent call last):
  File "hidden_stdlib/logging_example_tracebacks.py", line 52, in <module>
    raise RuntimeError('failure message')
RuntimeError: failure message
```

More Information

- http://docs.python.org/
- http://www.doughellmann.com/PyMOTW/
- https://bitbucket.org/dhellmann/hidden_stdlib/

- *The Python Standard Library By Example,* Doug Hellmann
- *Python Essential Reference*, David Beazley

Image Credits

2. http://www.uottawa.ca.libguides.com/content.php?pid=14825&sid=117658
4. http://www.flickr.com/photos/seditiouscanary/1279041211/
5. http://www.flickr.com/photos/elrentaplats/4902419885/in/photostream/
6. http://www.flickr.com/photos/elrentaplats/4903005444/in/photostream/
18. http://etc.usf.edu/clipart/58000/58063/58063_column_types.htm
34. http://www.flickr.com/photos/akeg/3077866744/
44. http://www.flickr.com/photos/yum9me/2807475045/
64. http://commons.wikimedia.org/wiki/File:Zeichen_trissturis_error101.png
72. http://www.flickr.com/photos/myklroventine/3545127104/in/faves-40068198@N04/