python-cheatsheet Documentation

Release 0.1.0

crazyguitar

Feb 05, 2018

Contents

1 Python basic cheatsheet **1**

1.1 Python Naming Rule . . . 2
1.2 Using __future__ backport features . . . 3
1.3 Check object attributes . . . 4
1.4 Define a function __doc__ . . . 4
1.5 Check instance type . . . 4
1.6 Check, Get, Set attribute . . . 5
1.7 Check inheritance . . . 5
1.8 Check all global variables . . . 5
1.9 Check **callable** . . . 6
1.10 Get function/class name . . . 6
1.11 __new__ & __init__ . . . 6
1.12 The diamond problem . . . 7
1.13 Representations of your class behave . . . 7
1.14 Break up a long string . . . 7
1.15 Get list item **SMART** . . . 8
1.16 Get dictionary item **SMART** . . . 9
1.17 Set a list/dict **SMART** . . . 10
1.18 set operations . . . 10
1.19 NamedTuple . . . 12
1.20 __iter__ - Delegating Iteration . . . 12
1.21 Using Generator as Iterator . . . 12
1.22 Emulating a list . . . 12
1.23 Emulating a dictionary . . . 13
1.24 Emulating a matrix multiplication . . . 14
1.25 Decorator . . . 15
1.26 Decorator with arguments . . . 16
1.27 for: exp else: exp . . . 16
1.28 try: exp else: exp . . . 17
1.29 Lambda function . . . 17
1.30 Option arguments - (*args, **kwargs) . . . 18
1.31 type() declare (create) a class . . . 18
1.32 Callable object . . . 19
1.33 Context Manager - with statement . . . 19
1.34 Using @contextmanager . . . 20
1.35 Using with statement open file . . . 20

1.36 Reading file chunk . . . 20
1.37 Property - Managed attributes . . . 20
1.38 Computed attributes - Using property . . . 21
1.39 Descriptor - manage attributes . . . 21
1.40 `@staticmethod`, `@classmethod` . . . 22
1.41 Abstract method - Metaclass . . . 23
1.42 Common Use **Magic** . . . 23
1.43 Parsing csv string . . . 24
1.44 Using `__slots__` to save memory . . . 25
1.45 Using annotation to check type . . . 26

2 Python unicode cheatsheet **29**
2.1 Encode: unicode code point to bytes . . . 29
2.2 Decode: bytes to unicode code point . . . 29
2.3 Get unicode code point . . . 30
2.4 python2 `str` is equivalent to byte string . . . 30
2.5 python3 `str` is equivalent to unicode string . . . 30
2.6 python2 take `str` char as byte character . . . 31
2.7 python3 take `str` char as unicode character . . . 31
2.8 unicode normalization . . . 31

3 Python generator cheatsheet **33**
3.1 Glossary of Generator . . . 34
3.2 Produce value via generator . . . 35
3.3 Unpacking Generators . . . 35
3.4 Implement Iterable object via generator . . . 36
3.5 Send message to generator . . . 36
3.6 `yield from` expression . . . 37
3.7 yield (from) EXPR return RES . . . 38
3.8 Generate sequences . . . 38
3.9 What `RES = yield from EXP` actually do? . . . 39
3.10 `for _ in gen()` simulate `yield from` . . . 40
3.11 Check generator type . . . 40
3.12 Check Generator State . . . 40
3.13 Simple compiler . . . 41
3.14 Context manager and generator . . . 43
3.15 What `@contextmanager` actually doing? . . . 43
3.16 profile code block . . . 44
3.17 `yield from` and `__iter__` . . . 44
3.18 `yield from == await` expression . . . 45
3.19 Closure in Python - using generator . . . 46
3.20 Implement a simple scheduler . . . 47
3.21 Simple round-robin with blocking . . . 48
3.22 simple round-robin with blocking and non-blocking . . . 49
3.23 Asynchronous Generators . . . 50
3.24 Asynchronous generators can have `try..finally` blocks . . . 51
3.25 send value and throw exception into async generator . . . 51
3.26 Simple async round-robin . . . 52
3.27 Async generator get better performance than async iterator . . . 52
3.28 Asynchronous Comprehensions . . . 53

4 Python Regular Expression cheatsheet **57**
4.1 Compare HTML tags . . . 58
4.2 `re.findall()` match string . . . 58

4.3 Group Comparison . . . 59
4.4 Non capturing group . . . 59
4.5 Back Reference . . . 59
4.6 Named Grouping (?P<name>) . . . 60
4.7 Substitute String . . . 60
4.8 Look around . . . 61
4.9 Match common username or password . . . 61
4.10 Match hex color value . . . 61
4.11 Match email . . . 61
4.12 Match URL . . . 62
4.13 Match IP address . . . 62
4.14 Match Mac address . . . 62
4.15 Lexer . . . 63

5 Python socket cheatsheet **65**
5.1 Get Hostname . . . 66
5.2 Transform Host & Network Endian . . . 66
5.3 IP dotted-quad string & byte format convert . . . 67
5.4 Mac address & byte format convert . . . 67
5.5 Simple TCP Echo Server . . . 67
5.6 Simple TCP Echo Server through IPv6 . . . 68
5.7 Disable IPv6 Only . . . 69
5.8 Simple TCP Echo Server Via SocketServer . . . 70
5.9 Simple TLS/SSL TCP Echo Server . . . 70
5.10 Set ciphers on TLS/SSL TCP Echo Server . . . 71
5.11 Simple UDP Echo Server . . . 72
5.12 Simple UDP Echo Server Via SocketServer . . . 72
5.13 Simple UDP client - Sender . . . 73
5.14 Broadcast UDP Packets . . . 73
5.15 Simple UNIX Domain Socket . . . 74
5.16 Simple duplex processes communication . . . 74
5.17 Simple Asynchronous TCP Server - Thread . . . 75
5.18 Simple Asynchronous TCP Server - select . . . 76
5.19 Simple Asynchronous TCP Server - poll . . . 76
5.20 Simple Asynchronous TCP Server - epoll . . . 79
5.21 Simple Asynchronous TCP Server - kqueue . . . 81
5.22 High-Level API - selectors . . . 83
5.23 Simple Non-blocking TLS/SSL socket via selectors . . . 84
5.24 “socketpair” - Similar to PIPE . . . 86
5.25 Using sendfile do copy . . . 87
5.26 Sending a file through sendfile . . . 88
5.27 Linux kernel Crypto API - AF_ALG . . . 89
5.28 AES-CBC encrypt/decrypt via AF_ALG . . . 90
5.29 AES-GCM encrypt/decrypt via AF_ALG . . . 91
5.30 AES-GCM encrypt/decrypt file with sendfile . . . 93
5.31 Compare the performance of AF_ALG to cryptography . . . 95
5.32 Sniffer IP packets . . . 97
5.33 Sniffer TCP packet . . . 99
5.34 Sniffer ARP packet . . . 102

6 Python cryptography cheatsheet **105**
6.1 Simple https server . . . 106
6.2 Check certificate information . . . 106
6.3 Generate a self-signed certificate . . . 108

6.4 Prepare a Certificate Signing Request (csr) . . . 109
6.5 Generate RSA keyfile without passphrase . . . 110
6.6 Sign a file by a given private key . . . 111
6.7 Verify a file from a signed digest . . . 111
6.8 Simple RSA encrypt via pem file . . . 112
6.9 Simple RSA encrypt via RSA module . . . 113
6.10 Simple RSA decrypt via pem file . . . 114
6.11 Simple RSA encrypt with OAEP . . . 114
6.12 Simple RSA decrypt with OAEP . . . 115
6.13 Using DSA to proof of identity . . . 116
6.14 Using AES CBC mode encrypt a file . . . 117
6.15 Using AES CBC mode decrypt a file . . . 118
6.16 AES CBC mode encrypt via password (using cryptography) . . . 119
6.17 AES CBC mode decrypt via password (using cryptography) . . . 120
6.18 AES CBC mode encrypt via password (using pycrypto) . . . 122
6.19 AES CBC mode decrypt via password (using pycrytpo) . . . 123
6.20 Ephemeral Diffie Hellman Key Exchange via cryptography . . . 124
6.21 Calculate DH shared key manually via cryptography . . . 124
6.22 Calculate DH shared key from (p, g, pubkey) . . . 125

7 Python Concurrency Cheatsheet 127
7.1 Execute a shell command . . . 128
7.2 Create a thread via “threading” . . . 128
7.3 Performance Problem - GIL . . . 129
7.4 Consumer and Producer . . . 129
7.5 Thread Pool Template . . . 130
7.6 Using multiprocessing ThreadPool . . . 131
7.7 Mutex lock . . . 131
7.8 Deadlock . . . 132
7.9 Implement “Monitor” . . . 133
7.10 Control primitive resources . . . 133
7.11 Ensure tasks has done . . . 134
7.12 Thread-safe priority queue . . . 135
7.13 Multiprocessing . . . 136
7.14 Custom multiprocessing map . . . 137
7.15 Graceful way to kill all child processes . . . 137
7.16 Simple round-robin scheduler . . . 138
7.17 Scheduler with blocking function . . . 138
7.18 PoolExecutor . . . 140
7.19 What “with ThreadPoolExecutor” doing? . . . 141
7.20 Future Object . . . 141
7.21 Future error handling . . . 142

8 Python SQLAlchemy Cheatsheet 143
8.1 Set a database URL . . . 144
8.2 Sqlalchemy Support DBAPI - PEP249 . . . 144
8.3 Transaction and Connect Object . . . 145
8.4 Metadata - Generating Database Schema . . . 145
8.5 Inspect - Get Database Information . . . 146
8.6 Reflection - Loading Table from Existing Database . . . 146
8.7 Get Table from MetaData . . . 146
8.8 Create all Tables Store in “MetaData” . . . 147
8.9 Create Specific Table . . . 147
8.10 Create table with same columns . . . 148

8.11 Drop a Table . . . 148
8.12 Some Table Object Operation . . . 149
8.13 SQL Expression Language . . . 149
8.14 insert() - Create an "INSERT" Statement . . . 150
8.15 select() - Create a "SELECT" Statement . . . 151
8.16 join() - Joined Two Tables via "JOIN" Statement . . . 152
8.17 Delete Rows from Table . . . 152
8.18 Check Table Existing . . . 153
8.19 Create multiple tables at once . . . 153
8.20 Create tables with dynamic columns (Table) . . . 154
8.21 Object Relational add data . . . 155
8.22 Object Relational update data . . . 156
8.23 Object Relational delete row . . . 157
8.24 Object Relational relationship . . . 158
8.25 Object Relational self association . . . 158
8.26 Object Relational basic query . . . 160
8.27 mapper: Map `Table` to `class` . . . 163
8.28 Get table dynamically . . . 164
8.29 Object Relational join two tables . . . 165
8.30 join on relationship and group_by count . . . 167
8.31 Create tables with dynamic columns (ORM) . . . 168
8.32 Close database connection . . . 169
8.33 Cannot use the object after close the session . . . 170

9 Python asyncio cheatsheet **171**
9.1 What is @asyncio.coroutine? . . . 172
9.2 What is a Task? . . . 173
9.3 What event loop doing? (Without polling) . . . 174
9.4 What `asyncio.wait` doing? . . . 175
9.5 Future like object . . . 176
9.6 Future like object `__await__` other task . . . 177
9.7 Patch loop runner `_run_once` . . . 177
9.8 Put blocking task into Executor . . . 178
9.9 Socket with asyncio . . . 178
9.10 Event Loop with polling . . . 179
9.11 Transport and Protocol . . . 181
9.12 Transport and Protocol with SSL . . . 181
9.13 What `loop.create_server` do? . . . 182
9.14 Inline callback . . . 183
9.15 Asynchronous Iterator . . . 184
9.16 What is asynchronous iterator . . . 185
9.17 Asynchronous context manager . . . 185
9.18 What is asynchronous context manager . . . 186
9.19 What *loop.sock_** do? . . . 186
9.20 Simple asyncio connection pool . . . 188
9.21 Simple asyncio UDP echo server . . . 190
9.22 Simple asyncio web server . . . 192
9.23 Simple HTTPS asyncio web server . . . 192
9.24 Simple asyncio WSGI web server . . . 195

10 Python test cheatsheet **197**
10.1 A simple Python unittest . . . 198
10.2 Python unittest setup & teardown hierarchy . . . 198
10.3 Different module of setUp & tearDown hierarchy . . . 199

10.4 Run tests via unittest.TextTestRunner . . . 201
10.5 Test raise exception . . . 201
10.6 Pass arguments into a TestCase . . . 202
10.7 Group multiple testcases into a suite . . . 202
10.8 Group multiple tests from different TestCase . . . 203
10.9 Skip some tests in the TestCase . . . 203
10.10 Monolithic Test . . . 204
10.11 Cross-module variables to Test files . . . 204
10.12 skip setup & teardown when the test is skipped . . . 205
10.13 Re-using old test code . . . 206
10.14 Testing your document is right . . . 206
10.15 Re-using doctest to unittest . . . 207
10.16 Customize test report . . . 208
10.17 Mock - using `@patch` substitute original method . . . 210
10.18 What `with unittest.mock.patch` do? . . . 211
10.19 Mock - substitute `open` . . . 212

11 Python C API cheatsheet **213**
11.1 Performance of ctypes . . . 213
11.2 Error handling when use ctypes . . . 214
11.3 Getting File System Type . . . 216
11.4 Doing Zero-copy via sendfile . . . 217
11.5 PyObject header . . . 218
11.6 Python C API Template . . . 219
11.7 PyObject with Member and Methods . . . 220

12 Python Design Pattern in C **227**
12.1 Decorator in C . . . 227
12.2 A Set of Functions . . . 229
12.3 Closure in C . . . 230
12.4 Generator . . . 231
12.5 Context Manager in C . . . 232
12.6 Tuple in C . . . 233
12.7 Error Handling . . . 233
12.8 Simple `try: exp except: exp finally:` in C . . . 234
12.9 Simple coroutine in C . . . 236
12.10 Keyword Arguments in C . . . 238
12.11 Function “MAP” . . . 238
12.12 foreach in C . . . 239
12.13 Simple OOP in C . . . 240

CHAPTER 1

Python basic cheatsheet

Table of Contents

- *Python basic cheatsheet*
  - *Python Naming Rule*
  - *Using `__future__` backport features*
  - *Check object attributes*
  - *Define a function `__doc__`*
  - *Check instance type*
  - *Check, Get, Set attribute*
  - *Check inheritance*
  - *Check all global variables*
  - *Check **callable***
  - *Get function/class name*
  - *`__new__` & `__init__`*
  - *The diamond problem*
  - *Representations of your class behave*
  - *Break up a long string*
  - *Get list item **SMART***
  - *Get dictionary item **SMART***
  - *Set a list/dict **SMART***
  - *`set` operations*

- *NamedTuple*
- *`__iter__` - Delegating Iteration*
- *Using Generator as Iterator*
- *Emulating a list*
- *Emulating a dictionary*
- *Emulating a matrix multiplication*
- *Decorator*
- *Decorator with arguments*
- *for: exp else: exp*
- *try: exp else: exp*
- *Lambda function*
- *Option arguments - (*args, **kwargs)*
- *`type()` declare (create) a `class`*
- *Callable object*
- *Context Manager - `with` statement*
- *Using `@contextmanager`*
- *Using `with` statement open file*
- *Reading file chunk*
- *Property - Managed attributes*
- *Computed attributes - Using property*
- *Descriptor - manage attributes*
- *`@staticmethod`, `@classmethod`*
- *Abstract method - Metaclass*
- *Common Use **Magic***
- *Parsing csv string*
- *Using `__slots__` to save memory*
- *Using annotation to check type*

1.1 Python Naming Rule

```
# see: PEP 8

# for class
#
# good:
#   MyClass
# bad:
#   myClass, my_class
```

```
MyClass

# for func, module, package, variables
#
# good:
#   var_underscore_separate
# bad:
#   varCamel, VarCamel
var_underscore_separate

# for public use
var

# for internal use
_var

# convention to avoid conflict keyword
var_

# for private use in class
__var

# for protect use in class
_var_

# "magic" method or attributes
# ex: __init__, __file__, __main__
__var__

# for "internal" use throwaway variable
# usually used in loop
# ex: [_ for _ in range(10)]
# or variable not used
# for _, a in [(1,2),(3,4)]: print a
_
```

1.2 Using `__future__` backport features

```
# PEP 236 - Back to the __future__

# backport python3 print_function in python2

>>> print "Hello World"  # print is a statement
Hello World
>>> from __future__ import print_function
>>> print "Hello World"
  File "<stdin>", line 1
    print "Hello World"
                      ^
SyntaxError: invalid syntax
>>> print("Hello World") # print become a function
Hello World

# backport python3 unicode_literals in python2
```

```
>>> type("Guido") # string type is str in python2
<type 'str'>
>>> from __future__ import unicode_literals
>>> type("Guido") # string type become unicode
<type 'unicode'>

# backport PEP 238 -- Changing the Division Operator

>>> 1/2
0
>>> from __future__ import division
>>> 1/2   # return a float (classic division)
0.5
>>> 1//2  # return a int (floor division)
0
```

Note: `from __future__ import feature` is a future statement. It use for backporting features of other python version to current python version, not like original import.

1.3 Check object attributes

```
# example of check list attributes
>>> dir(list)
['__add__', '__class__', ...]
```

1.4 Define a function `__doc__`

```
# Define a function document
>>> def example():
...    """ This is an example function """
...    print "Example function"
...
>>> example.__doc__
' This is an example function '

# Or using help function
>>> help(example)
```

1.5 Check instance type

```
>>> ex = 10
>>> isinstance(ex,int)
True
```

1.6 Check, Get, Set attribute

```
>>> class Example(object):
...   def __init__(self):
...     self.name = "ex"
...   def printex(self):
...     print "This is an example"
...

# Check object has attributes
# hasattr(obj, 'attr')
>>> ex = Example()
>>> hasattr(ex,"name")
True
>>> hasattr(ex,"printex")
True
>>> hasattr(ex,"print")
False

# Get object attribute
# getattr(obj, 'attr')
>>> getattr(ex,'name')
'ex'

# Set object attribute
# setattr(obj, 'attr', value)
>>> setattr(ex,'name','example')
>>> ex.name
'example'
```

1.7 Check inheritance

```
>>> class Example(object):
...   def __init__(self):
...     self.name = "ex"
...   def printex(self):
...     print "This is an Example"
...
>>> issubclass(Example, object)
True
```

1.8 Check all global variables

```
# globals() return a dictionary
# {'variable name': variable value}
>>> globals()
{'args': (1, 2, 3, 4, 5), ...}
```

1.9 Check callable

```
>>> a = 10
>>> def fun():
...   print "I am callable"
...
>>> callable(a)
False
>>> callable(fun)
True
```

1.10 Get function/class name

```
>>> class ExampleClass(object):
...   pass
...
>>> def example_function():
...   pass
...
>>> ex = ExampleClass()
>>> ex.__class__.__name__
'ExampleClass'
>>> example_function.__name__
'example_function'
```

1.11 __new__ & __init__

```
# __init__ will invoke
>>> class ClassA(object):
...     def __new__(cls, arg):
...         print '__new__ ' + arg
...         return object.__new__(cls, arg)
...     def __init__(self, arg):
...         print '__init__ ' + arg
...
>>> o = ClassA("Hello")
__new__ Hello
__init__ Hello

# __init__ won't be invoke
>>> class ClassB(object):
...     def __new__(cls, arg):
...         print '__new__ ' + arg
...         return object
...     def __init__(self, arg):
...         print '__init__ ' + arg
...
>>> o = ClassB("Hello")
__new__ Hello
```

1.12 The diamond problem

```
# The problem of multiple inheritance in searching a method

>>> def foo_a(self):
...     print("This is ClsA")
...
>>> def foo_b(self):
...     print("This is ClsB")
...
>>> def foo_c(self):
...     print("This is ClsC")
...
>>> class Type(type):
...     def __repr__(cls):
...         return cls.__name__
...
>>> ClsA = Type("ClsA", (object,), {'foo': foo_a})
>>> ClsB = Type("ClsB", (ClsA,), {'foo': foo_b})
>>> ClsC = Type("ClsC", (ClsA,), {'foo': foo_c})
>>> ClsD = Type("ClsD", (ClsB, ClsC), {})
>>> ClsD.mro()
[ClsD, ClsB, ClsC, ClsA, <type 'object'>]
>>> ClsD().foo()
This is ClsB
```

1.13 Representations of your class behave

```
>>> class Example(object):
...    def __str__(self):
...       return "Example __str__"
...    def __repr__(self):
...       return "Example __repr__"
...
>>> print str(Example())
Example __str__
>>> Example()
Example __repr__
```

1.14 Break up a long string

```
# original long string
>>> s = 'This is a very very very long python string'
>>> s
'This is a very very very long python string'

# single quote with an escaping backslash
>>> s = "This is a very very very " \
...     "long python string"
>>> s
'This is a very very very long python string'
```

```
# using brackets
>>> s = ("This is a very very very "
...      "long python string")
>>> s
'This is a very very very long python string'

# using '+'
>>> s = ("This is a very very very " +
...      "long python string")
>>> s
'This is a very very very long python string'

# using triple-quote with an escaping backslash
>>> s = '''This is a very very very \
... long python string'''
>>> s
'This is a very very very long python string'
```

1.15 Get list item SMART

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0]
1
>>> a[-1]
5
>>> a[0:]
[1, 2, 3, 4, 5]
>>> a[:-1]
[1, 2, 3, 4]

# a[start:end:step]
>>> a[0:-1:2]
[1, 3]

# using slice object
# slice(start,end,step)
>>> s = slice(0, -1, 2)
>>> a[s]
[1, 3]

# Get index and item in loop
>>> a = range(3)
>>> for idx, item in enumerate(a):
...   print (idx,item),
...
(0, 0) (1, 1) (2, 2)

# Transfer two list into tuple list
>>> a = [1, 2, 3, 4, 5]
>>> b = [2, 4, 5, 6, 8]
>>> zip(a, b)
[(1, 2), (2, 4), (3, 5), (4, 6), (5, 8)]

# with filter
>>> [x for x in range(5) if x > 1]
```

```
[2, 3, 4]
>>> l = ['1', '2', 3, 'Hello', 4]
>>> predicate = lambda x: isinstance(x, int)
>>> filter(predicate, l)
[3, 4]

# collect distinct objects
>>> a = [1, 2, 3, 3, 3]
>>> list({_ for _ in a})
[1, 2, 3]
# or
>>> list(set(a))
[1, 2, 3]

# reverse
>>> a = [1, 2, 3, 4, 5]
>>> a[::-1]
[5, 4, 3, 2, 1]

# be careful
>>> a = [[]] * 3
>>> b = [[] for _ in range(3)]
>>> a[0].append("Hello")
>>> a
[['Hello'], ['Hello'], ['Hello']]
>>> b[0].append("Python")
>>> b
[['Python'], [], []]
```

1.16 Get dictionary item SMART

```
# get dictionary all keys
>>> a = {"1":1, "2":2, "3":3}
>>> b = {"2":2, "3":3, "4":4}
>>> a.keys()
['1', '3', '2']

# get dictionary key and value as tuple
>>> a.items()
[('1', 1), ('3', 3), ('2', 2)]

# find same key between two dictionary
>>> [_ for _ in a.keys() if _ in b.keys()]
['3', '2']
# better way
>>> c = set(a).intersection(set(b))
>>> list(c)
['3', '2']
# or
>>> [_ for _ in a if _ in b]
['3', '2']

# update dictionary
>>> a.update(b)
>>> a
```

```
{'1': 1, '3': 3, '2': 2, '4': 4}
```

1.17 Set a list/dict SMART

```
# get a list with init value
>>> ex = [0] * 10
>>> ex
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# extend two list
>>> a = [1, 2, 3]; b = ['a', 'b']
>>> a + b
[1, 2, 3, 'a', 'b']

# using list comprehension
>>> [x for x in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> fn = lambda x: x**2
>>> [fn(x) for x in range(5)]
[0, 1, 4, 9, 16]
>>> {'{0}'.format(x): x for x in range(3)}
{'1': 1, '0': 0, '2': 2}

# using builtin function "map"
>>> map(fn, range(5))
[0, 1, 4, 9, 16]
```

1.18 `set` operations

```
# set comprehension
>>> a = [1, 2, 5, 6, 6, 6, 7]
>>> s = {x for x in a}
>>> s
set([1, 2, 5, 6, 7])
>>> s = {x for x in a if x > 3}
>>> s
set([5, 6, 7])
>>> s = {x if x > 3 else -1 for x in a}
>>> s
set([6, 5, -1, 7])

# uniquify list
>>> a = [1, 2, 2, 2, 3, 4, 5, 5]
>>> a
[1, 2, 2, 2, 3, 4, 5, 5]
>>> ua = list(set(a))
>>> ua
[1, 2, 3, 4, 5]

# union two set
>>> a = set([1, 2, 2, 2, 3])
>>> b = set([5, 5, 6, 6, 7])
```

```
>>> a | b
set([1, 2, 3, 5, 6, 7])
# or
>>> a = [1, 2, 2, 2, 3]
>>> b = [5, 5, 6, 6, 7]
>>> set(a + b)
set([1, 2, 3, 5, 6, 7])

# append item to set
>>> a = set([1, 2, 3, 3, 3])
>>> a.add(5)
>>> a
set([1, 2, 3, 5])
# or
>>> a = set([1, 2, 3, 3, 3])
>>> a |= set([1, 2, 3, 4, 5, 6])
>>> a
set([1, 2, 3, 4, 5, 6])

# intersection two set
>>> a = set([1, 2, 2, 2, 3])
>>> b = set([1, 5, 5, 6, 6, 7])
>>> a & b
set([1])

# get two list common items
>>> a = [1, 1, 2, 3]
>>> b = [1, 3, 5, 5, 6, 6]
>>> com = list(set(a) & set(b))
>>> com
[1, 3]

# b contains a
>>> a = set([1, 2])
>>> b = set([1, 2, 5, 6])
>>> a <=b
True

# a contains b
>>> a = set([1, 2, 5, 6])
>>> b = set([1, 5, 6])
>>> a >= b
True

# set diff
>>> a = set([1, 2, 3])
>>> b = set([1, 5, 6, 7, 7])
>>> a - b
set([2, 3])

# symmetric diff
>>> a = set([1,2,3])
>>> b = set([1, 5, 6, 7, 7])
>>> a ^ b
set([2, 3, 5, 6, 7])
```

1.19 NamedTuple

```
# namedtuple(typename, field_names)
# replace define class without method
>>> from collections import namedtuple
>>> Example = namedtuple("Example",'a b c')
>>> e = Example(1, 2, 3)
>>> print e.a, e[1], e[1] + e.b
1 2 4
```

1.20 `__iter__` - Delegating Iteration

```
# __iter__ return an iterator object
# Be careful: list is an "iterable" object not an "iterator"
>>> class Example(object):
...     def __init__(self,list_):
...         self._list = list_
...     def __iter__(self):
...       return iter(self._list)
...
>>> ex = Example([1, 2, 3, 4, 5])
>>> for _ in ex: print _,
...
1 2 3 4 5
```

1.21 Using Generator as Iterator

```
# see: PEP289
>>> a = (_ for _ in range(10))
>>> for _ in a: print _,
...
0 1 2 3 4 5 6 7 8 9

# equivalent to
>>> def generator():
...   for _ in range(10):
...     yield _
...
>>> for _ in generator(): print _,
...
0 1 2 3 4 5 6 7 8 9
```

1.22 Emulating a list

```
>>> class EmuList(object):
...   def __init__(self, list_):
...     self._list = list_
...   def __repr__(self):
...     return "EmuList: " + repr(self._list)
```

```
...   def append(self, item):
...     self._list.append(item)
...   def remove(self, item):
...     self._list.remove(item)
...   def __len__(self):
...     return len(self._list)
...   def __getitem__(self, sliced):
...     return self._list[sliced]
...   def __setitem__(self, sliced, val):
...     self._list[sliced] = val
...   def __delitem__(self, sliced):
...     del self._list[sliced]
...   def __contains__(self, item):
...     return item in self._list
...   def __iter__(self):
...     return iter(self._list)
...
>>> emul = EmuList(range(5))
>>> emul
EmuList: [0, 1, 2, 3, 4]
>>> emul[1:3]  #  __getitem__
[1, 2]
>>> emul[0:4:2]  #  __getitem__
[0, 2]
>>> len(emul)  #  __len__
5
>>> emul.append(5)
>>> emul
EmuList: [0, 1, 2, 3, 4, 5]
>>> emul.remove(2)
>>> emul
EmuList: [0, 1, 3, 4, 5]
>>> emul[3] = 6  # __setitem__
>>> emul
EmuList: [0, 1, 3, 6, 5]
>>> 0 in emul  # __contains__
True
```

1.23 Emulating a dictionary

```
>>> class EmuDict(object):
...   def __init__(self, dict_):
...     self._dict = dict_
...   def __repr__(self):
...     return "EmuDict: " + repr(self._dict)
...   def __getitem__(self, key):
...     return self._dict[key]
...   def __setitem__(self, key, val):
...     self._dict[key] = val
...   def __delitem__(self, key):
...     del self._dict[key]
...   def __contains__(self, key):
...     return key in self._dict
...   def __iter__(self):
...     return iter(self._dict.keys())
```

```
...
>>> _ = {"1":1, "2":2, "3":3}
>>> emud = EmuDict(_)
>>> emud  # __repr__
EmuDict: {'1': 1, '2': 2, '3': 3}
>>> emud['1']  # __getitem__
1
>>> emud['5'] = 5  # __setitem__
>>> emud
EmuDict: {'1': 1, '2': 2, '3': 3, '5': 5}
>>> del emud['2']  # __delitem__
>>> emud
EmuDict: {'1': 1, '3': 3, '5': 5}
>>> for _ in emud: print emud[_],  # __iter__
...
1 3 5
>>> '1' in emud  # __contains__
True
```

1.24 Emulating a matrix multiplication

```
# PEP 465 - "@" represent matrix multiplication
#
# Need Python-3.5 or above

>>> class Arr:
...     def __init__(self, *arg):
...         self._arr = arg
...     def __matmul__(self, other):
...         if not isinstance(other, Arr):
...             raise TypeError
...         if len(self) != len(other):
...             raise ValueError
...         return sum([x*y for x, y in zip(self._arr, other._arr)])
...     def __imatmul__(self, other):
...         if not isinstance(other, Arr):
...             raise TypeError
...         if len(self) != len(other):
...             raise ValueError
...         res = sum([x*y for x, y in zip(self._arr, other._arr)])
...         self._arr = [res]
...         return self
...     def __len__(self):
...         return len(self._arr)
...     def __str__(self):
...         return self.__repr__()
...     def __repr__(self):
...         return "Arr({})".format(repr(self._arr))
...
>>> a = Arr(9, 5, 2, 7)
>>> b = Arr(5, 5, 6, 6)
>>> a @ b  # __matmul__
124
>>> a @= b  # __imatmul__
>>> a
```

```
Arr([124])
```

1.25 Decorator

```
# see: PEP318
>>> from functools import wraps
>>> def decorator(func):
...   @wraps(func)
...   def wrapper(*args, **kwargs):
...     print "Before calling {}.".format(func.__name__)
...     ret = func(*args, **kwargs)
...     print "After calling {}.".format(func.__name__)
...     return ret
...   return wrapper
...
>>> @decorator
... def example():
...   print "Inside example function."
...
>>> example()
Before calling example.
Inside example function.
After calling example.

# equivalent to
... def example():
...   print "Inside example function."
...
>>> example = decorator(example)
>>> example()
Before calling example.
Inside example function.
After calling example.
```

Note: `@wraps` preserve attributes of the original function, otherwise attributes of decorated function will be replaced by **wrapper function**

```
# without @wraps
>>> def decorator(func):
...     def wrapper(*args, **kwargs):
...         print('wrap function')
...         return func(*args, **kwargs)
...     return wrapper
...
>>> @decorator
... def example(*a, **kw):
...     pass
...
>>> example.__name__  # attr of function lose
'wrapper'

# with @wraps
>>> from functools import wraps
```

```
>>> def decorator(func):
...     @wraps(func)
...     def wrapper(*args, **kwargs):
...         print('wrap function')
...         return func(*args, **kwargs)
...     return wrapper
...
>>> @decorator
... def example(*a, **kw):
...     pass
...
>>> example.__name__  # attr of function preserve
'example'
```

1.26 Decorator with arguments

```
>>> from functools import wraps
>>> def decorator_with_argument(val):
...   def decorator(func):
...     @wraps(func)
...     def wrapper(*args, **kwargs):
...       print "Val is {0}".format(val)
...       return func(*args, **kwargs)
...     return wrapper
...   return decorator
...
>>> @decorator_with_argument(10)
... def example():
...   print "This is example function."
...
>>> example()
Val is 10
This is example function.

# equivalent to
>>> def example():
...   print "This is example function."
...
>>> example = decorator_with_argument(10)(example)
>>> example()
Val is 10
This is example function.
```

1.27 for: exp else: exp

```
# see document: More Control Flow Tools
# forloop's else clause runs when no break occurs
>>> for _ in range(5):
...   print _,
... else:
...   print "\nno break occurred"
...
```

```
0 1 2 3 4
no break occurred
>>> for _ in range(5):
...   if _ % 2 == 0:
...     print "break occurred"
...     break
... else:
...   print "no break occurred"
...
break occurred

# above statement equivalent to
flag = False
for _ in range(5):
    if _ % 2 == 0:
        flag = True
        print "break occurred"
        break
if flag == False:
    print "no break occurred"
```

1.28 try: exp else: exp

```
# No exception occur will go into else.
>>> try:
...   print "No exception"
... except:
...   pass
... else:
...   print "No exception occurred"
...
No exception
No exception occurred
```

1.29 Lambda function

```
>>> fn = lambda x: x**2
>>> fn(3)
9
>>> (lambda x: x**2)(3)
9
>>> (lambda x: [x*_ for _ in range(5)])(2)
[0, 2, 4, 6, 8]
>>> (lambda x: x if x>3 else 3)(5)
5

# multiline lambda example
>>> (lambda x:
... True
... if x>0
... else
```

```
... False)(3)
True
```

1.30 Option arguments - (*args, **kwargs)

```
>>> def example(a, b=None, *args, **kwargs):
...   print a, b
...   print args
...   print kwargs
...
>>> example(1, "var", 2, 3, word="hello")
1 var
(2, 3)
{'word': 'hello'}
>>> a_tuple = (1, 2, 3, 4, 5)
>>> a_dict = {"1":1, "2":2, "3":3}
>>> example(1, "var", *a_tuple, **a_dict)
1 var
(1, 2, 3, 4, 5)
{'1': 1, '2': 2, '3': 3}
```

1.31 `type()` declare (create) a `class`

```
>>> def fib(self, n):
...     if n <= 2:
...         return 1
...     return fib(self, n-1) + fib(self, n-2)
...
>>> Fib = type('Fib', (object,), {'val': 10,
...                               'fib': fib})
>>> f = Fib()
>>> f.val
10
>>> f.fib(f.val)
55

# equal to
>>> class Fib(object):
...     val = 10
...     def fib(self, n):
...         if n <=2:
...             return 1
...         return self.fib(n-1)+self.fib(n-2)
...
>>> f = Fib()
>>> f.val
10
>>> f.fib(f.val)
55
```

1.32 Callable object

```
>>> class CallableObject(object):
...   def example(self, *args, **kwargs):
...     print "I am callable!"
...   def __call__(self, *args, **kwargs):
...     self.example(*args, **kwargs)
...
>>> ex = CallableObject()
>>> ex()
I am callable!
```

1.33 Context Manager - `with` statement

```
# replace try: ... finally: ...
# see: PEP343
# common use in open and close

import socket

class Socket(object):
    def __init__(self,host,port):
        self.host = host
        self.port = port

    def __enter__(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind((self.host,self.port))
        sock.listen(5)
        self.sock = sock
        return self.sock

    def __exit__(self,*exc_info):
        if exc_info[0] is not None:
            import traceback
            traceback.print_exception(*exc_info)
        self.sock.close()

if __name__=="__main__":
    host = 'localhost'
    port = 5566
    with Socket(host, port) as s:
        while True:
            conn, addr = s.accept()
            msg = conn.recv(1024)
            print msg
            conn.send(msg)
            conn.close()
```

1.34 Using `@contextmanager`

```
from contextlib import contextmanager

@contextmanager
def opening(filename, mode='r'):
   f = open(filename, mode)
   try:
      yield f
   finally:
      f.close()

with opening('example.txt') as fd:
   fd.read()
```

1.35 Using `with` statement open file

```
>>> with open("/etc/passwd",'r') as f:
...     content = f.read()
```

1.36 Reading file chunk

```
>>> chunk_size = 16
>>> content = ''
>>> with open('/etc/hosts') as f:
...     for c in iter(lambda: f.read(chunk_size), ''):
...         content += c
...
>>> print(content)
127.0.0.1   localhost
255.255.255.255     broadcasthost
::1             localhost

10.245.1.3  www.registry.io
```

1.37 Property - Managed attributes

```
>>> class Example(object):
...     def __init__(self, value):
...         self._val = value
...     @property
...     def val(self):
...         return self._val
...     @val.setter
...     def val(self, value):
...         if not isintance(value, int):
...             raise TypeError("Expected int")
...         self._val = value
...     @val.deleter
```

```
...     def val(self):
...         del self._val
...
>>> ex = Example(123)
>>> ex.val = "str"
Traceback (most recent call last):
  File "", line 1, in
  File "test.py", line 12, in val
    raise TypeError("Expected int")
TypeError: Expected int

# equivalent to
>>> class Example(object):
...     def __init__(self, value):
...        self._val = value
...
...     def _val_getter(self):
...         return self._val
...
...     def _val_setter(self, value):
...         if not isintance(value, int):
...             raise TypeError("Expected int")
...         self._val = value
...
...     def _val_deleter(self):
...         del self._val
...
...     val = property(fget=_val_getter, fset=_val_setter, fdel=_val_deleter,␣
↪doc=None)
...
```

1.38 Computed attributes - Using property

```
>>> class Example(object):
...   @property
...   def square3(self):
...     return 2**3
...
>>> ex = Example()
>>> ex.square3
8
```

Note: `@property` compute the value of attribute only when we need. Not store in memory previously.

1.39 Descriptor - manage attributes

```
>>> class Integer(object):
...   def __init__(self, name):
...     self._name = name
...   def __get__(self, inst, cls):
```

```
...       if inst is None:
...         return self
...       else:
...         return inst.__dict__[self._name]
...     def __set__(self, inst, value):
...       if not isinstance(value, int):
...         raise TypeError("Expected int")
...       inst.__dict__[self._name] = value
...     def __delete__(self,inst):
...       del inst.__dict__[self._name]
...
>>> class Example(object):
...     x = Integer('x')
...     def __init__(self, val):
...       self.x = val
...
>>> ex1 = Example(1)
>>> ex1.x
1
>>> ex2 = Example("str")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in __init__
  File "<stdin>", line 11, in __set__
TypeError: Expected an int
>>> ex3 = Example(3)
>>> hasattr(ex3, 'x')
True
>>> del ex3.x
>>> hasattr(ex3, 'x')
False
```

1.40 @staticmethod, @classmethod

```
# @classmethod: bound to class
# @staticmethod: like python function but in class
>>> class example(object):
...     @classmethod
...     def clsmethod(cls):
...       print "I am classmethod"
...     @staticmethod
...     def stmethod():
...       print "I am staticmethod"
...     def instmethod(self):
...       print "I am instancemethod"
...
>>> ex = example()
>>> ex.clsmethod()
I am classmethod
>>> ex.stmethod()
I am staticmethod
>>> ex.instmethod()
I am instancemethod
>>> example.clsmethod()
I am classmethod
```

```
>>> example.stmethod()
I am staticmethod
>>> example.instmethod()
Traceback (most recent call last):
  File "", line 1, in
TypeError: unbound method instmethod() ...
```

1.41 Abstract method - Metaclass

```
# usually using in define methods but not implement
>>> from abc import ABCMeta, abstractmethod
>>> class base(object):
...   __metaclass__ = ABCMeta
...   @abstractmethod
...   def absmethod(self):
...     """ Abstract method """
...
>>> class example(base):
...   def absmethod(self):
...     print "abstract"
...
>>> ex = example()
>>> ex.absmethod()
abstract

# another better way to define a meta class
>>> class base(object):
...   def absmethod(self):
...     raise NotImplementedError
...
>>> class example(base):
...   def absmethod(self):
...     print "abstract"
...
>>> ex = example()
>>> ex.absmethod()
abstract
```

1.42 Common Use Magic

```
# see python document: data model
# For command class
__main__
__name__
__file__
__module__
__all__
__dict__
__class__
__doc__
__init__(self, [...)
__str__(self)
```

```
__repr__(self)
__del__(self)

# For Descriptor
__get__(self, instance, owner)
__set__(self, instance, value)
__delete__(self, instance)

# For Context Manager
__enter__(self)
__exit__(self, exc_ty, exc_val, tb)

# Emulating container types
__len__(self)
__getitem__(self, key)
__setitem__(self, key, value)
__delitem__(self, key)
__iter__(self)
__contains__(self, value)

# Controlling Attribute Access
__getattr__(self, name)
__setattr__(self, name, value)
__delattr__(self, name)
__getattribute__(self, name)

# Callable object
__call__(self, [args...])

# Compare related
__cmp__(self, other)
__eq__(self, other)
__ne__(self, other)
__lt__(self, other)
__gt__(self, other)
__le__(self, other)
__ge__(self, other)

# arithmetical operation related
__add__(self, other)
__sub__(self, other)
__mul__(self, other)
__div__(self, other)
__mod__(self, other)
__and__(self, other)
__or__(self, other)
__xor__(self, other)
```

1.43 Parsing csv string

```
# python2 and python3 compatible

>>> try:
...     from StringIO import StringIO # for py2
... except ImportError:
```

```
...     from io import StringIO # for py3
...
>>> import csv
>>> s = "foo,bar,baz"
>>> f = StringIO(s)
>>> for x in csv.reader(f): print(x)
...
['foo', 'bar', 'baz']

# or

>>> import csv
>>> s = "foo,bar,baz"
>>> for x in csv.reader([s]): print(x)
...
['foo', 'bar', 'baz']
```

1.44 Using `__slots__` to save memory

```
#!/usr/bin/env python3

import resource
import platform
import functools


def profile_mem(func):
    @functools.wraps(func)
    def wrapper(*a, **k):
        s = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss
        ret = func(*a, **k)
        e = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss

        uname = platform.system()
        if uname == "Linux":
            print(f"mem usage: {e - s} kByte")
        elif uname == "Darwin":
            print(f"mem usage: {e - s} Byte")
        else:
            raise Exception("not support")
        return ret
    return wrapper


class S(object):
    __slots__ = ['attr1', 'attr2', 'attr3']

    def __init__(self):
        self.attr1 = "Foo"
        self.attr2 = "Bar"
        self.attr3 = "Baz"


class D(object):
```

```
    def __init__(self):
        self.attr1 = "Foo"
        self.attr2 = "Bar"
        self.attr3 = "Baz"


@profile_mem
def alloc(cls):
    _ = [cls() for _ in range(1000000)]


alloc(S)
alloc(D)
```

output:

```
$ python3.6 s.py
mem usage: 70922240 Byte
mem usage: 100659200 Byte
```

1.45 Using annotation to check type

```
# need python3 (PEP: 3107)
from functools import wraps

import inspect

ANNO_EMPTY = inspect._empty

def check_args(sig, *a, **k):
    bind = sig.bind(*a, **k)
    params = sig.parameters
    for name, val in bind.arguments.items():
        anno = params[name].annotation
        if anno is ANNO_EMPTY:
            continue
        if isinstance(val, anno):
            continue
        atype = type(val)
        raise TypeError(f"type({name}) is '{anno}', not '{atype}'")


def check_ret(sig, ret):
    anno = sig.return_annotation
    if anno is ANNO_EMPTY:
        return ret
    elif isinstance(ret, anno):
        return ret

    rtype = type(ret)
    raise TypeError(f"type(ret) is '{anno}', not '{rtype}'")



def typechecked(func):
```

```
    sig = inspect.signature(func)

    @wraps(func)
    def wrapper(*a, **k):
        check_args(sig, *a, **k)
        return check_ret(sig, func(*a, **k))
    return wrapper


@typechecked
def test1(a: int)->int:
    return a

@typechecked
def test2(a: int):
    return a

@typechecked
def test3(a)->str:
    return a

@typechecked
def test4(a, b: str, c: str="c")->list:
    return [a, b, c]

print(test1(9527))
print(test2(9487))
print(test3("Hello Python3"))
print(test4(9487, "bb", c="cc"))

try:
    print(test3(9487))
except TypeError as e:
    print(e)

try:
    print(test4(5566, 9527))
except TypeError as e:
    print(e)

try:
    print(test4(123, "b", c=5566))
except TypeError as e:
    print(e)
```

output:

```
9527
9487
Hello Python3
[9487, 'bb', 'cc']
type(ret) is '<class 'str'>', not '<class 'int'>'
type(b) is '<class 'str'>', not '<class 'int'>'
type(c) is '<class 'str'>', not '<class 'int'>'
```

CHAPTER 2

Python unicode cheatsheet

Table of Contents

- *Python unicode cheatsheet*
  - *Encode: unicode code point to bytes*
  - *Decode: bytes to unicode code point*
  - *Get unicode code point*
  - *python2 `str` is equivalent to byte string*
  - *python3 `str` is equivalent to unicode string*
  - *python2 take `str` char as byte character*
  - *python3 take `str` char as unicode character*
  - *unicode normalization*

2.1 Encode: unicode code point to bytes

```
>>> s = u'Café'
>>> type(s.encode('utf-8'))
<class 'bytes'>
```

2.2 Decode: bytes to unicode code point

```
>>> s = bytes('Café', encoding='utf-8')
>>> s.decode('utf-8')
'Café'
```

2.3 Get unicode code point

```
>>> s = u'Café'
>>> for _c in s: print('U+%04x' % ord(_c))
...
U+0043
U+0061
U+0066
U+00e9
>>> u = ''
>>> for _c in u: print('U+%04x' % ord(_c))
...
U+4e2d
U+6587
```

2.4 python2 `str` is equivalent to byte string

```
>>> s = 'Café'  # byte string
>>> s
'Caf\xc3\xa9'
>>> type(s)
<type 'str'>
>>> u = u'Café' # unicode string
>>> u
u'Caf\xe9'
>>> type(u)
<type 'unicode'>
```

2.5 python3 `str` is equivalent to unicode string

```
>>> s = 'Café'
>>> type(s)
<class 'str'>
>>> s
'Café'
>>> s.encode('utf-8')
b'Caf\xc3\xa9'
>>> s.encode('utf-8').decode('utf-8')
'Café'
```

2.6 python2 take `str` char as byte character

```
>>> s= 'Café'
>>> print [_c for _c in s]
['C', 'a', 'f', '\xc3', '\xa9']
>>> len(s)
5
>>> s = u'Café'
>>> print [_c for _c in s]
[u'C', u'a', u'f', u'\xe9']
>>> len(s)
4
```

2.7 python3 take `str` char as unicode character

```
 >>> s = 'Café'
 >>> print([_c for _c in s])
 ['C', 'a', 'f', 'é']
 >>> len(s)
 4
 >>> bs = bytes(s, encoding='utf-8')
 >>> print(bs)
 b'Caf\xc3\xa9'
 >>> len(bs)
5
```

2.8 unicode normalization

```
# python 3
>>> u1 = 'Café'        # unicode string
>>> u2 = 'Cafe\u0301'
>>> u1, u2
('Café', 'Cafe')
>>> len(u1), len(u2)
(4, 5)
>>> u1 == u2
False
>>> u1.encode('utf-8') # get u1 byte string
b'Caf\xc3\xa9'
>>> u2.encode('utf-8') # get u2 byte string
b'Cafe\xcc\x81'
>>> from unicodedata import normalize
>>> s1 = normalize('NFC', u1)  # get u1 NFC format
>>> s2 = normalize('NFC', u2)  # get u2 NFC format
>>> s1 == s2
True
>>> s1.encode('utf-8'), s2.encode('utf-8')
(b'Caf\xc3\xa9', b'Caf\xc3\xa9')
>>> s1 = normalize('NFD', u1)  # get u1 NFD format
>>> s2 = normalize('NFD', u2)  # get u2 NFD format
>>> s1, s2
('Cafe', 'Cafe')
```

```
>>> s1 == s2
True
>>> s1.encode('utf-8'), s2.encode('utf-8')
(b'Cafe\xcc\x81', b'Cafe\xcc\x81')
```

CHAPTER 3

Python generator cheatsheet

Table of Contents

- *Python generator cheatsheet*
  - *Glossary of Generator*
  - *Produce value via generator*
  - *Unpacking Generators*
  - *Implement Iterable object via generator*
  - *Send message to generator*
  - *`yield from` expression*
  - *yield (from) EXPR return RES*
  - *Generate sequences*
  - *What `RES = yield from EXP` actually do?*
  - *`for _ in gen()` simulate `yield from`*
  - *Check generator type*
  - *Check Generator State*
  - *Simple compiler*
  - *Context manager and generator*
  - *What `@contextmanager` actually doing?*
  - *profile code block*
  - *`yield from` and `__iter__`*
  - *`yield from == await` expression*

- *Closure in Python - using generator*
- *Implement a simple scheduler*
- *Simple round-robin with blocking*
- *simple round-robin with blocking and non-blocking*
- *Asynchronous Generators*
- *Asynchronous generators can have `try..finally` blocks*
- *send value and throw exception into async generator*
- *Simple async round-robin*
- *Async generator get better performance than async iterator*
- *Asynchronous Comprehensions*

3.1 Glossary of Generator

```
# generator function

>>> def gen_func():
...     yield 5566
...
>>> gen_func
<function gen_func at 0x1019273a>

# generator
#
# calling the generator function returns a generator

>>> g = gen_func()
>>> g
<generator object gen_func at 0x101238fd>
>>> next(g)
5566
>>> next(g)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration

# generator expression
#
# generator expression evaluating directly to a generator

>>> g = (x for x in range(2))
>>> g
<generator object <genexpr> at 0x10a9c191>
>>> next(g)
0
>>> next(g)
1
>>> next(g)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
StopIteration
```

3.2 Produce value via generator

```
>>> from __future__ import print_function
>>> def prime(n):
...     p = 2
...     while n > 0:
...         for x in range(2, p):
...             if p % x == 0:
...                 break
...         else:
...             yield p
...             n -= 1
...         p += 1
...
>>> p = prime(3)
>>> next(p)
2
>>> next(p)
3
>>> next(p)
5
>>> next(p)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>> for x in prime(5):
...     print(x, end=" ")
...
2 3 5 7 11 >>>
```

3.3 Unpacking Generators

```
# PEP 448

# unpacking inside a list

>>> g1 = (x for x in range(3))
>>> g2 = (x**2 for x in range(2))
>>> [1, *g1, 2, *g2]
[1, 0, 1, 2, 2, 0, 1]
>>> # equal to
>>> g1 = (x for x in range(3))
>>> g2 = (x**2 for x in range(2))
>>> [1] + list(g1) + [2] + list(g2)
[1, 0, 1, 2, 2, 0, 1]

# unpacking inside a set

>>> g = (x for x in [5, 5, 6, 6])
>>> {*g}
```

```
{5, 6}

# unpacking to variables

>>> g = (x for x in range(3))
>>> a, b, c = g
>>> print(a, b, c)
0 1 2
>>> g = (x for x in range(6))
>>> a, b, *c, d = g
>>> print(a, b, d)
0 1 5
>>> print(c)
[2, 3, 4]

# unpacking inside a function

>>> print(*(x for x in range(3)))
0 1 2
```

3.4 Implement Iterable object via generator

```
>>> from __future__ import print_function
>>> class Count(object):
...     def __init__(self, n):
...         self._n = n
...     def __iter__(self):
...         n = self._n
...         while n > 0:
...             yield n
...             n -= 1
...     def __reversed__(self):
...         n = 1
...         while n <= self._n:
...             yield n
...             n += 1
...
>>> for x in Count(5):
...     print(x, end=" ")
...
5 4 3 2 1 >>>
>>> for x in reversed(Count(5)):
...     print(x, end=" ")
...
1 2 3 4 5 >>>
```

3.5 Send message to generator

```
>>> def spam():
...     msg = yield
...     print("Message:", msg)
...
```

```
>>> try:
...     g = spam()
...     # start generator
...     next(g)
...     # send message to generator
...     g.send("Hello World!")
... except StopIteration:
...     pass
...
Message: Hello World!
```

3.6 `yield from` expression

```
# delegating gen do nothing(pipe)
>>> def subgen():
...     try:
...         yield 9527
...     except ValueError:
...         print("get value error")
...
>>> def delegating_gen():
...     yield from subgen()
...
>>> g = delegating_gen()
>>> try:
...     next(g)
...     g.throw(ValueError)
... except StopIteration:
...     print("gen stop")
...
9527
get value error
gen stop

# yield from + yield from
>>> import inspect
>>> def subgen():
...     yield from range(5)
...
>>> def delegating_gen():
...     yield from subgen()
...
>>> g = delegating_gen()
>>> inspect.getgeneratorstate(g)
'GEN_CREATED'
>>> next(g)
0
>>> inspect.getgeneratorstate(g)
'GEN_SUSPENDED'
>>> g.close()
>>> inspect.getgeneratorstate(g)
'GEN_CLOSED'
```

3.7 yield (from) EXPR return RES

```
>>> def average():
...     total = .0
...     count = 0
...     avg = None
...     while True:
...         val = yield
...         if not val:
...             break
...         total += val
...         count += 1
...         avg = total / count
...     return avg
...
>>> g = average()
>>> next(g) # start gen
>>> g.send(3)
>>> g.send(5)
>>> try:
...     g.send(None)
... except StopIteration as e:
...     ret = e.value
...
>>> ret
4.0

# yield from EXP return RES
>>> def subgen():
...     yield 9527
...
>>> def delegating_gen():
...     yield from subgen()
...     return 5566
...
>>> try:
...     g = delegating_gen()
...     next(g)
...     next(g)
... except StopIteration as _e:
...     print(_e.value)
...
9527
5566
```

3.8 Generate sequences

```
# get a list via generator

>>> def chain():
...     for x in 'ab':
...         yield x
...     for x in range(3):
...         yield x
...
```

```
>>> a = list(chain())
>>> a
['a', 'b', 0, 1, 2]

# equivalent to

>>> def chain():
...     yield from 'ab'
...     yield from range(3)
...
>>> a = list(chain())
>>> a
['a', 'b', 0, 1, 2]
```

3.9 What `RES = yield from EXP` actually do?

```
# ref: pep380
>>> def subgen():
...     for x in range(3):
...         yield x
...
>>> EXP = subgen()
>>> def delegating_gen():
...     _i = iter(EXP)
...     try:
...         _y = next(_i)
...     except StopIteration as _e:
...         RES = _e.value
...     else:
...         while True:
...             _s = yield _y
...             try:
...                 _y = _i.send(_s)
...             except StopIteration as _e:
...                 RES = _e.value
...                 break
...
>>> g = delegating_gen()
>>> next(g)
0
>>> next(g)
1
>>> next(g)
2

# equivalent to
>>> EXP = subgen()
>>> def delegating_gen():
...     RES = yield from EXP
...
>>> g = delegating_gen()
>>> next(g)
0
>>> next(g)
1
```

3.10 `for _ in gen()` simulate `yield from`

```
>>> def subgen(n):
...     for x in range(n): yield x
...
>>> def gen(n):
...     yield from subgen(n)
...
>>> g = gen(3)
>>> next(g)
0
>>> next(g)
1

# equal to

>>> def gen(n):
...     for x in subgen(n): yield x
...
>>> g = gen(3)
>>> next(g)
0
>>> next(g)
1
```

3.11 Check generator type

```
>>> from types import GeneratorType
>>> def gen_func():
...     yield 5566
...
>>> g = gen_func()
>>> isinstance(g, GeneratorType)
True
>>> isinstance(123, GeneratorType)
False
```

3.12 Check Generator State

```
>>> import inspect
>>> def gen_func():
...     yield 9527
...
>>> g = gen_func()
>>> inspect.getgeneratorstate(g)
'GEN_CREATED'
>>> next(g)
9527
>>> inspect.getgeneratorstate(g)
'GEN_SUSPENDED'
>>> g.close()
```

```
>>> inspect.getgeneratorstate(g)
'GEN_CLOSED'
```

3.13 Simple compiler

```
# David Beazley - Generators: The Final Frontier

import re
import types
from collections import namedtuple

tokens = [
    r'(?P<NUMBER>\d+)',
    r'(?P<PLUS>\+)',
    r'(?P<MINUS>-)',
    r'(?P<TIMES>\*)',
    r'(?P<DIVIDE>/)',
    r'(?P<WS>\s+)']

Token = namedtuple('Token', ['type', 'value'])
lex = re.compile('|'.join(tokens))

def tokenize(text):
    scan = lex.scanner(text)
    gen = (Token(m.lastgroup, m.group())
           for m in iter(scan.match, None) if m.lastgroup != 'WS')
    return gen


class Node:
    _fields = []
    def __init__(self, *args):
        for attr, value in zip(self._fields, args):
            setattr(self, attr, value)

class Number(Node):
    _fields = ['value']

class BinOp(Node):
    _fields = ['op', 'left', 'right']

def parse(toks):
    lookahead, current = next(toks, None), None

    def accept(*toktypes):
        nonlocal lookahead, current
        if lookahead and lookahead.type in toktypes:
            current, lookahead = lookahead, next(toks, None)
            return True

    def expr():
        left = term()
        while accept('PLUS', 'MINUS'):
            left = BinOp(current.value, left)
            left.right = term()
```

```
        return left

    def term():
        left = factor()
        while accept('TIMES', 'DIVIDE'):
            left = BinOp(current.value, left)
            left.right = factor()
        return left

    def factor():
        if accept('NUMBER'):
            return Number(int(current.value))
        else:
            raise SyntaxError()
    return expr()


class NodeVisitor:
    def visit(self, node):
        stack = [self.genvisit(node)]
        ret = None
        while stack:
            try:
                node = stack[-1].send(ret)
                stack.append(self.genvisit(node))
                ret = None
            except StopIteration as e:
                stack.pop()
                ret = e.value
        return ret

    def genvisit(self, node):
        ret = getattr(self, 'visit_' + type(node).__name__)(node)
        if isinstance(ret, types.GeneratorType):
            ret = yield from ret
        return ret

class Evaluator(NodeVisitor):
    def visit_Number(self, node):
        return node.value

    def visit_BinOp(self, node):
        leftval = yield node.left
        rightval = yield node.right
        if node.op == '+':
            return leftval + rightval
        elif node.op == '-':
            return leftval - rightval
        elif node.op == '*':
            return leftval * rightval
        elif node.op == '/':
            return leftval / rightval

def evaluate(exp):
    toks = tokenize(exp)
    tree = parse(toks)
    return Evaluator().visit(tree)
```

```
exp = '2 * 3 + 5 / 2'
print(evaluate(exp))
exp = '+'.join([str(x) for x in range(10000)])
print(evaluate(exp))
```

output:

```
python3 compiler.py
8.5
49995000
```

3.14 Context manager and generator

```
>>> import contextlib
>>> @contextlib.contextmanager
... def mylist():
...     try:
...         l = [1, 2, 3, 4, 5]
...         yield l
...     finally:
...         print("exit scope")
...
>>> with mylist() as l:
...   print(l)
...
[1, 2, 3, 4, 5]
exit scope
```

3.15 What `@contextmanager` actually doing?

```
# ref: PyCon 2014 - David Beazley
# define a context manager class

class GeneratorCM(object):

    def __init__(self,gen):
        self._gen = gen

    def __enter__(self):
        return next(self._gen)

    def __exit__(self, *exc_info):
        try:
            if exc_info[0] is None:
                next(self._gen)
            else:
                self._gen.throw(*exc_info)
            raise RuntimeError
        except StopIteration:
            return True
        except:
```

```
            raise

# define a decorator
def contextmanager(func):
    def run(*a, **k):
        return GeneratorCM(func(*a, **k))
    return run

# example of context manager
@contextmanager
def mylist():
    try:
        l = [1, 2, 3, 4, 5]
        yield l
    finally:
        print "exit scope"

with mylist() as l:
    print l
```

output:

```
$ python ctx.py
[1, 2, 3, 4, 5]
exit scope
```

3.16 profile code block

```
>>> import time
>>> @contextmanager
... def profile(msg):
...     try:
...         s = time.time()
...         yield
...     finally:
...         e = time.time()
...         print('{} cost time: {}'.format(msg, e - s))
...
>>> with profile('block1'):
...     time.sleep(1)
...
block1 cost time: 1.00105595589
>>> with profile('block2'):
...     time.sleep(3)
...
block2 cost time: 3.00104284286
```

3.17 `yield from` and `__iter__`

```
>>> class FakeGen:
...     def __iter__(self):
...         n = 0
```

```
...         while True:
...             yield n
...             n += 1
...     def __reversed__(self):
...         n = 9527
...         while True:
...            yield n
...            n -= 1
...
>>> def spam():
...     yield from FakeGen()
...
>>> s = spam()
>>> next(s)
0
>>> next(s)
1
>>> next(s)
2
>>> next(s)
3
>>> def reversed_spam():
...     yield from reversed(FakeGen())
...
>>> g = reversed_spam()
>>> next(g)
9527
>>> next(g)
9526
>>> next(g)
9525
```

3.18 `yield from == await` expression

```
# "await" include in pyhton3.5
import asyncio
import socket

# set socket and event loop
loop = asyncio.get_event_loop()
host = 'localhost'
port = 5566
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM,0)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
sock.setblocking(False)
sock.bind((host, port))
sock.listen(10)

@asyncio.coroutine
def echo_server():
    while True:
        conn, addr = yield from loop.sock_accept(sock)
        loop.create_task(handler(conn))

@asyncio.coroutine
```

```
def handler(conn):
    while True:
        msg = yield from loop.sock_recv(conn, 1024)
        if not msg:
            break
        yield from loop.sock_sendall(conn, msg)
    conn.close()

# equal to
async def echo_server():
    while True:
        conn, addr = await loop.sock_accept(sock)
        loop.create_task(handler(conn))

async def handler(conn):
    while True:
        msg = await loop.sock_recv(conn, 1024)
        if not msg:
            break
        await loop.sock_sendall(conn, msg)
    conn.close()

loop.create_task(echo_server())
loop.run_forever()
```

output: (bash 1)

```
$ nc localhost 5566
Hello
Hello
```

output: (bash 2)

```
$ nc localhost 5566
World
World
```

3.19 Closure in Python - using generator

```
# nonlocal version
>>> def closure():
...     x = 5566
...     def inner_func():
...         nonlocal x
...         x += 1
...         return x
...     return inner_func
...
>>> c = closure()
>>> c()
5567
>>> c()
5568
>>> c()
5569
```

```
# class version
>>> class Closure:
...     def __init__(self):
...         self._x = 5566
...     def __call__(self):
...         self._x += 1
...         return self._x
...
>>> c = Closure()
>>> c()
5567
>>> c()
5568
>>> c()
5569

# generator version (best)
>>> def closure_gen():
...     x = 5566
...     while True:
...         x += 1
...         yield x
...
>>> g = closure_gen()
>>> next(g)
5567
>>> next(g)
5568
>>> next(g)
5569
```

3.20 Implement a simple scheduler

```
# idea: write an event loop(scheduler)
>>> def fib(n):
...     if n <= 2:
...         return 1
...     return fib(n-1) + fib(n-2)
...
>>> def g_fib(n):
...     for x in range(1, n + 1):
...         yield fib(x)
...
>>> from collections import deque
>>> t = [g_fib(3), g_fib(5)]
>>> q = deque()
>>> q.extend(t)
>>> def run():
...     while q:
...         try:
...             t = q.popleft()
...             print(next(t))
...             q.append(t)
...         except StopIteration:
```

```
...             print("Task done")
...
>>> run()
1
1
1
1
2
2
Task done
3
5
Task done
```

3.21 Simple round-robin with blocking

```
# ref: PyCon 2015 - David Beazley
# skill: using task and wait queue

from collections import deque
from select import select
import socket

tasks = deque()
w_read = {}
w_send = {}

def run():
    while any([tasks, w_read, w_send]):
        while not tasks:
            # polling tasks
            can_r, can_s,_ = select(w_read, w_send, [])
            for _r in can_r:
                tasks.append(w_read.pop(_r))
            for _w in can_s:
                tasks.append(w_send.pop(_w))
        try:
            task = tasks.popleft()
            why,what = next(task)
            if why == 'recv':
                w_read[what] = task
            elif why == 'send':
                w_send[what] = task
            else:
                raise RuntimeError
        except StopIteration:
            pass

def server():
    host = ('localhost',5566)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(host)
    sock.listen(5)
    while True:
```

```
        # tell scheduler want block
        yield 'recv', sock
        conn,addr = sock.accept()
        tasks.append(client_handler(conn))

def client_handler(conn):
    while True:
        # tell scheduler want block
        yield 'recv', conn
        msg = conn.recv(1024)
        if not msg:
            break
        # tell scheduler want block
        yield 'send', conn
        conn.send(msg)
    conn.close()

tasks.append(server())
run()
```

3.22 simple round-robin with blocking and non-blocking

```
# this method will cause blocking hunger
from collections import deque
from select import select
import socket

tasks = deque()
w_read = {}
w_send = {}

def run():
    while any([tasks, w_read, w_send]):
        while not tasks:
            # polling tasks
            can_r,can_s,_ = select(w_read, w_send,[])
            for _r in can_r:
                tasks.append(w_read.pop(_r))
            for _w in can_s:
                tasks.append(w_send.pop(_w))
        try:
            task = tasks.popleft()
            why,what = next(task)
            if why == 'recv':
                w_read[what] = task
            elif why == 'send':
                w_send[what] = task
            elif why == 'continue':
                print what
                tasks.append(task)
            else:
                raise RuntimeError
        except StopIteration:
            pass
```

```
def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)

def g_fib(n):
    for x in range(1, n + 1):
        yield 'continue', fib(x)

tasks.append(g_fib(15))

def server():
    host = ('localhost',5566)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(host)
    sock.listen(5)
    while True:
        yield 'recv', sock
        conn,addr = sock.accept()
        tasks.append(client_handler(conn))

def client_handler(conn):
    while True:
        yield 'recv', conn
        msg = conn.recv(1024)
        if not msg:
            break
        yield 'send', conn
        conn.send(msg)
    conn.close()

tasks.append(server())
run()
```

3.23 Asynchronous Generators

```
# PEP 525
#
# Need python-3.6 or above

>>> import asyncio
>>> async def slow_gen(n, t):
...     for x in range(n):
...         await asyncio.sleep(t)
...         yield x
...
>>> async def task(n):
...     async for x in slow_gen(n, 0.1):
...         print(x)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(task(3))
0
1
```

```
2
```

3.24 Asynchronous generators can have `try..finally` blocks

```
# Need python-3.6 or above

>>> import asyncio
>>> async def agen(t):
...     try:
...         await asyncio.sleep(t)
...         yield 1 / 0
...     finally:
...         print("finally part")
...
>>> async def main(t=1):
...     try:
...         g = agen(t)
...         await g.__anext__()
...     except Exception as e:
...         print(repr(e))
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(main(1))
finally part
ZeroDivisionError('division by zero',)
```

3.25 send value and throw exception into async generator

```
# Need python-3.6 or above

>>> import asyncio
>>> async def agen(n, t=0.1):
...     try:
...         for x in range(n):
...             await asyncio.sleep(t)
...             val = yield x
...             print(f'get val: {val}')
...     except RuntimeError as e:
...         await asyncio.sleep(t)
...         yield repr(e)
...
>>> async def main(n):
...     g = agen(n)
...     ret = await g.asend(None) + await g.asend('foo')
...     print(ret)
...     ret = await g.athrow(RuntimeError('Get RuntimeError'))
...     print(ret)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(main(5))
get val: foo
```

```
1
RuntimeError('Get RuntimeError',)
```

3.26 Simple async round-robin

```
# Need python-3.6 or above

>>> import asyncio
>>> from collections import deque
>>> async def agen(n, t=0.1):
...     for x in range(n):
...         await asyncio.sleep(t)
...         yield x
...
>>> async def main():
...     q = deque([agen(3), agen(5)])
...     while q:
...         try:
...             g = q.popleft()
...             ret = await g.__anext__()
...             print(ret)
...             q.append(g)
...         except StopAsyncIteration:
...             pass
...
>>> loop.run_until_complete(main())
0
0
1
1
2
2
3
4
```

3.27 Async generator get better performance than async iterator

```
# Need python-3.6 or above

>>> import time
>>> import asyncio
>>> class AsyncIter:
...     def __init__(self, n):
...         self._n = n
...     def __aiter__(self):
...         return self
...     async def __anext__(self):
...         ret = self._n
...         if self._n == 0:
...             raise StopAsyncIteration
...         self._n -= 1
...         return ret
```

```
...
>>> async def agen(n):
...     for i in range(n):
...         yield i
...
>>> async def task_agen(n):
...     s = time.time()
...     async for _ in agen(n): pass
...     cost = time.time() - s
...     print(f"agen cost time: {cost}")
...
>>> async def task_aiter(n):
...     s = time.time()
...     async for _ in AsyncIter(n): pass
...     cost = time.time() - s
...     print(f"aiter cost time: {cost}")
...
>>> n = 10 ** 7
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(task_agen(n))
agen cost time: 1.2698817253112793
>>> loop.run_until_complete(task_aiter(n))
aiter cost time: 4.168368101119995
```

3.28 Asynchronous Comprehensions

```
# PEP 530
#
# Need python-3.6 or above

>>> import asyncio
>>> async def agen(n, t):
...     for x in range(n):
...         await asyncio.sleep(t)
...         yield x
>>> async def main():
...     ret = [x  async for x in agen(5, 0.1)]
...     print(*ret)
...     ret = [x async for x in agen(5, 0.1) if x < 3]
...     print(*ret)
...     ret = [x if x < 3 else -1 async for x in agen(5, 0.1)]
...     print(*ret)
...     ret = {f'{x}': x async for x in agen(5, 0.1)}
...     print(ret)

>>> loop.run_until_complete(main())
0 1 2 3 4
0 1 2
0 1 2 -1 -1
{'0': 0, '1': 1, '2': 2, '3': 3, '4': 4}

# await in Comprehensions

>>> async def foo(t):
...     await asyncio.sleep(t)
```

```
...     return "foo"
...
>>> async def bar(t):
...     await asyncio.sleep(t)
...     return "bar"
...
>>> async def baz(t):
...     await asyncio.sleep(t)
...     return "baz"
...
>>> async def gen(*f, t=0.1):
...     for x in f:
...         await asyncio.sleep(t)
...         yield x
...
>>> async def await_simple_task():
...     ret = [await f(0.1) for f in [foo, bar]]
...     print(ret)
...     ret = {await f(0.1) for f in [foo, bar]}
...     print(ret)
...     ret = {f.__name__: await f(0.1) for f in [foo, bar]}
...     print(ret)
...
>>> async def await_other_task():
...     ret = [await f(0.1) for f in [foo, bar] if await baz(1)]
...     print(ret)
...     ret = {await f(0.1) for f in [foo, bar] if await baz(1)}
...     print(ret)
...     ret = {f.__name__: await f(0.1) for f in [foo, bar] if await baz(1)}
...     print(ret)
...
>>> async def await_aiter_task():
...     ret = [await f(0.1) async for f in gen(foo, bar)]
...     print(ret)
...     ret = {await f(0.1) async for f in gen(foo, bar)}
...     print(ret)
...     ret = {f.__name__: await f(0.1) async for f in gen(foo, bar)}
...     print(ret)
...     ret = [await f(0.1) async for f in gen(foo, bar) if await baz(1)]
...     print(ret)
...     ret = {await f(0.1) async for f in gen(foo, bar) if await baz(1)}
...     print(ret)
...     ret = {f.__name__: await f(0.1) async for f in gen(foo, bar) if await baz(1)}
...
>>> import asyncio
>>> asyncio.get_event_loop()
>>> loop.run_until_complete(await_simple_task())
['foo', 'bar']
{'bar', 'foo'}
{'foo': 'foo', 'bar': 'bar'}
>>> loop.run_until_complete(await_other_task())
['foo', 'bar']
{'bar', 'foo'}
{'foo': 'foo', 'bar': 'bar'}
>>> loop.run_until_complete(await_gen_task())
['foo', 'bar']
{'bar', 'foo'}
{'foo': 'foo', 'bar': 'bar'}
```

```
['foo', 'bar']
{'bar', 'foo'}
{'foo': 'foo', 'bar': 'bar'}
```

CHAPTER 4

Python Regular Expression cheatsheet

Table of Contents

- *Python Regular Expression cheatsheet*
  - *Compare HTML tags*
  - *`re.findall()` match string*
  - *Group Comparison*
  - *Non capturing group*
  - *Back Reference*
  - *Named Grouping `(?P<name>)`*
  - *Substitute String*
  - *Look around*
  - *Match common username or password*
  - *Match hex color value*
  - *Match email*
  - *Match URL*
  - *Match IP address*
  - *Match Mac address*
  - *Lexer*

4.1 Compare HTML tags

tag type	format	example
all tag	<[^>]+>	 , <a>
open tag	<[^/>][^>]*>	<a>, <table>
close tag	</[^>]+>	</p>, </a>
self close	<[^/>]+/>	

```
# open tag
>>> re.search('<[^/>][^>]*>', '<table>') != None
True
>>> re.search('<[^/>][^>]*>', '<a href="#label">') != None
True
>>> re.search('<[^/>][^>]*>', '<img src="/img">') != None
True
>>> re.search('<[^/>][^>]*>', '</table>') != None
False

# close tag
>>> re.search('</[^>]+>', '</table>') != None
True

# self close
>>> re.search('<[^/>]+/>', '<br />') != None
True
```

4.2 `re.findall()` match string

```
# split all string
>>> source = "Hello World Ker HAHA"
>>> re.findall('[\w]+', source)
['Hello', 'World', 'Ker', 'HAHA']

# parsing python.org website
>>> import urllib
>>> import re
>>> s = urllib.urlopen('https://www.python.org')
>>> html = s.read()
>>> s.close()
>>> print "open tags"
open tags
>>> re.findall('<[^/>][^>]*>', html)[0:2]
['<!doctype html>', '<!--[if lt IE 7]>']
>>> print "close tags"
close tags
>>> re.findall('</[^>]+>', html)[0:2]
['</script>', '</title>']
>>> print "self-closing tags"
```

4.3 Group Comparison

```
# (...) group a regular expression
>>> m = re.search(r'(\d{4})-(\d{2})-(\d{2})', '2016-01-01')
>>> m
<_sre.SRE_Match object; span=(0, 10), match='2016-01-01'>
>>> m.groups()
('2016', '01', '01')
>>> m.group()
'2016-01-01'
>>> m.group(1)
'2016'
>>> m.group(2)
'01'
>>> m.group(3)
'01'

# Nesting groups
>>> m = re.search(r'(((\d{4})-\d{2})-\d{2})', '2016-01-01')
>>> m.groups()
('2016-01-01', '2016-01', '2016')
>>> m.group()
'2016-01-01'
>>> m.group(1)
'2016-01-01'
>>> m.group(2)
'2016-01'
>>> m.group(3)
'2016'
```

4.4 Non capturing group

```
# non capturing group
>>> url = 'http://stackoverflow.com/'
>>> m = re.search('(?:http|ftp)://([^/\r\n]+)(/[^\r\n]*)?', url)
>>> m.groups()
('stackoverflow.com', '/')

# capturing group
>>> m = re.search('(http|ftp)://([^/\r\n]+)(/[^\r\n]*)?', url)
>>> m.groups()
('http', 'stackoverflow.com', '/')
```

4.5 Back Reference

```
# compare 'aa', 'bb'
>>> re.search(r'([a-z])\1$','aa') != None
True
>>> re.search(r'([a-z])\1$','bb') != None
True
>>> re.search(r'([a-z])\1$','ab') != None
False
```

```
# compare open tag and close tag
>>> pattern = r'<([^>]+)>[\s\S]*?</\1>'
>>> re.search(pattern, '<bold> test </bold>') != None
True
>>> re.search(pattern, '<h1> test </h1>') != None
True
>>> re.search(pattern, '<bold> test </h1>') != None
False
```

4.6 Named Grouping `(?P<name>)`

```
# group reference ``(?P<name>...)``
>>> pattern = '(?P<year>\d{4})-(?P<month>\d{2})-(?P<day>\d{2})'
>>> m = re.search(pattern, '2016-01-01')
>>> m.group('year')
'2016'
>>> m.group('month')
'01'
>>> m.group('day')
'01'

# back reference ``(?P=name)``
>>> re.search('^(?P<char>[a-z])(?P=char)','aa')
<_sre.SRE_Match object at 0x10ae0f288>
```

4.7 Substitute String

```
# basic substitute
>>> res = "1a2b3c"
>>> re.sub(r'[a-z]',' ', res)
'1 2 3 '

# substitute with group reference
>>> date = r'2016-01-01'
>>> re.sub(r'(\d{4})-(\d{2})-(\d{2})',r'\2/\3/\1/',date)
'01/01/2016/'

# camelcase to underscore
>>> def convert(s):
...     res = re.sub(r'(.)([A-Z][a-z]+)',r'\1_\2', s)
...     return re.sub(r'([a-z])([A-Z])',r'\1_\2', res).lower()
...
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('SimpleHTTPServer')
'simple_http_server'
```

4.8 Look around

notation	compare direction
(?=...)	left to right
(?!...)	left to right
(?<=...)	right to left
(?!<...)	right to left

```
# basic
>>> re.sub('(?=\d{3})', ' ', '12345')
' 1 2 345'
>>> re.sub('(?!\d{3})', ' ', '12345')
'123 4 5 '
>>> re.sub('(?<=\d{3})', ' ', '12345')
'123 4 5 '
>>> re.sub('(?<!\d{3})', ' ', '12345')
' 1 2 345'
```

4.9 Match common username or password

```
>>> re.match('^[a-zA-Z0-9-_]{3,16}$', 'Foo') is not None
True
>>> re.match('^\w|[-_]{3,16}$', 'Foo') is not None
True
```

4.10 Match hex color value

```
>>> re.match('^#?([a-f0-9]{6}|[a-f0-9]{3})$', '#ffffff')
<_sre.SRE_Match object at 0x10886f6c0>
>>> re.match('^#?([a-f0-9]{6}|[a-f0-9]{3})$', '#ffffh')
<_sre.SRE_Match object at 0x10886f288>
```

4.11 Match email

```
>>> re.match('^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$',
...          'hello.world@example.com')
<_sre.SRE_Match object at 0x1087a4d40>

# or

>>> exp = re.compile(r'''^([a-zA-Z0-9._%-]+@
...                     [a-zA-Z0-9.-]+
                        \.[a-zA-Z]{2,4})*$''', re.X)
>>> exp.match('hello.world@example.hello.com')
<_sre.SRE_Match object at 0x1083efd50>
>>> exp.match('hello%world@example.hello.com')
<_sre.SRE_Match object at 0x1083efeb8>
```

4.12 Match URL

```
>>> exp = re.compile(r'''^(https?:\/\/)? # match http or https
...                ([\da-z\.-]+)        # match domain
...                \.([a-z\.]{2,6})     # match domain
...                ([\/\w \.-]*)\/?$    # match api or file
...                ''', re.X)
>>>
>>> exp.match('www.google.com')
<_sre.SRE_Match object at 0x10f01ddf8>
>>> exp.match('http://www.example')
<_sre.SRE_Match object at 0x10f01dd50>
>>> exp.match('http://www.example/file.html')
<_sre.SRE_Match object at 0x10f01ddf8>
>>> exp.match('http://www.example/file!.html')
```

4.13 Match IP address

notation	description
(?:...)	Don't capture group
25[0-5]	Match 251-255 pattern
2[0-4][0-9]	Match 200-249 pattern
[1]?[0-9][0-9]	Match 0-199 pattern

```
>>> exp = re.compile(r'''^(?:(?:25[0-5]
...                       |2[0-4][0-9]
...                       |[1]?[0-9][0-9]?)\.){3}
...                       (?:25[0-5]
...                       |2[0-4][0-9]
...                       |[1]?[0-9][0-9]?)$''', re.X)
>>> exp.match('192.168.1.1')
<_sre.SRE_Match object at 0x108f47ac0>
>>> exp.match('255.255.255.0')
<_sre.SRE_Match object at 0x108f47b28>
>>> exp.match('172.17.0.5')
<_sre.SRE_Match object at 0x108f47ac0>
>>> exp.match('256.0.0.0') is None
True
```

4.14 Match Mac address

```
>>> import random
>>> mac = [random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f)]
>>> mac = ':'.join(map(lambda x: "%02x" % x, mac))
>>> mac
```

```
'3c:38:51:05:03:1e'
>>> exp = re.compile(r'''[0-9a-f]{2}([:])
...                       [0-9a-f]{2}
...                       (\1[0-9a-f]{2}){4}$''', re.X)
>>> exp.match(mac) is not None
True
```

4.15 Lexer

```
>>> import re
>>> from collections import namedtuple
>>> tokens = [r'(?P<NUMBER>\d+)',
...           r'(?P<PLUS>\+)',
...           r'(?P<MINUS>-)',
...           r'(?P<TIMES>\*)',
...           r'(?P<DIVIDE>/)',
...           r'(?P<WS>\s+)']
>>> lex = re.compile('|'.join(tokens))
>>> Token = namedtuple('Token', ['type', 'value'])
>>> def tokenize(text):
...     scan = lex.scanner(text)
...     return (Token(m.lastgroup, m.group())
...         for m in iter(scan.match, None) if m.lastgroup != 'WS')
...
>>> for _t in tokenize('9 + 5 * 2 - 7'):
...     print(_t)
...
Token(type='NUMBER', value='9')
Token(type='PLUS', value='+')
Token(type='NUMBER', value='5')
Token(type='TIMES', value='*')
Token(type='NUMBER', value='2')
Token(type='MINUS', value='-')
Token(type='NUMBER', value='7')
```

CHAPTER 5

Python socket cheatsheet

Table of Contents

- *Python socket cheatsheet*
  - *Get Hostname*
  - *Transform Host & Network Endian*
  - *IP dotted-quad string & byte format convert*
  - *Mac address & byte format convert*
  - *Simple TCP Echo Server*
  - *Simple TCP Echo Server through IPv6*
  - *Disable IPv6 Only*
  - *Simple TCP Echo Server Via SocketServer*
  - *Simple TLS/SSL TCP Echo Server*
  - *Set ciphers on TLS/SSL TCP Echo Server*
  - *Simple UDP Echo Server*
  - *Simple UDP Echo Server Via SocketServer*
  - *Simple UDP client - Sender*
  - *Broadcast UDP Packets*
  - *Simple UNIX Domain Socket*
  - *Simple duplex processes communication*
  - *Simple Asynchronous TCP Server - Thread*
  - *Simple Asynchronous TCP Server - select*

- *Simple Asynchronous TCP Server - poll*
- *Simple Asynchronous TCP Server - epoll*
- *Simple Asynchronous TCP Server - kqueue*
- *High-Level API - selectors*
- *Simple Non-blocking TLS/SSL socket via selectors*
- *"socketpair" - Similar to PIPE*
- *Using sendfile do copy*
- *Sending a file through sendfile*
- *Linux kernel Crypto API - AF_ALG*
- *AES-CBC encrypt/decrypt via AF_ALG*
- *AES-GCM encrypt/decrypt via AF_ALG*
- *AES-GCM encrypt/decrypt file with sendfile*
- *Compare the performance of AF_ALG to cryptography*
- *Sniffer IP packets*
- *Sniffer TCP packet*
- *Sniffer ARP packet*

5.1 Get Hostname

```
>>> import socket
>>> socket.gethostname()
'MacBookPro-4380.local'
>>> hostname = socket.gethostname()
>>> socket.gethostbyname(hostname)
'172.20.10.4'
>>> socket.gethostbyname('localhost')
'127.0.0.1'
```

5.2 Transform Host & Network Endian

```
# little-endian machine
>>> import socket
>>> a = 1 # host endian
>>> socket.htons(a) # network endian
256
>>> socket.htonl(a) # network endian
16777216
>>> socket.ntohs(256) # host endian
1
>>> socket.ntohl(16777216) # host endian
1

# big-endian machine
```

```
>>> import socket
>>> a = 1 # host endian
>>> socket.htons(a) # network endian
1
>>> socket.htonl(a) # network endian
1L
>>> socket.ntohs(1) # host endian
1
>>> socket.ntohl(1) # host endian
1L
```

5.3 IP dotted-quad string & byte format convert

```
>>> import socket
>>> addr = socket.inet_aton('127.0.0.1')
>>> addr
'\x7f\x00\x00\x01'
>>> socket.inet_ntoa(addr)
'127.0.0.1'
```

5.4 Mac address & byte format convert

```
>>> mac = '00:11:32:3c:c3:0b'
>>> byte = binascii.unhexlify(mac.replace(':',''))
>>> byte
'\x00\x112<\xc3\x0b'
>>> binascii.hexlify(byte)
'0011323cc30b'
```

5.5 Simple TCP Echo Server

```
import socket

class Server(object):
    def __init__(self,host,port):
        self._host = host
        self._port = port
    def __enter__(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
        sock.bind((self._host,self._port))
        sock.listen(10)
        self._sock = sock
        return self._sock
    def __exit__(self,*exc_info):
        if exc_info[0]:
            import traceback
            traceback.print_exception(*exc_info)
        self._sock.close()
```

```
if __name__ == '__main__':
    host = 'localhost'
    port = 5566
    with Server(host,5566) as s:
        while True:
            conn, addr = s.accept()
            msg = conn.recv(1024)
            conn.send(msg)
            conn.close()
```

output:

```
$ nc localhost 5566
Hello World
Hello World
```

5.6 Simple TCP Echo Server through IPv6

```
import contextlib
import socket

host = "::1"
port = 5566


@contextlib.contextmanager
def server(host, port):
    s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
    try:
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind((host, port))
        s.listen(10)
        yield s
    finally:
        s.close()


with server(host, port) as s:
    try:
        while True:
            conn, addr = s.accept()
            msg = conn.recv(1024)

            if msg:
                conn.send(msg)

            conn.close()
    except KeyboardInterrupt:
        pass
```

output:

```
$ python3 ipv6.py &
[1] 25752
```

```
$ nc -6 ::1 5566
Hello IPv6
Hello IPv6
```

5.7 Disable IPv6 Only

```
#!/usr/bin/env python3

import contextlib
import socket

host = "::"
port = 5566

@contextlib.contextmanager
def server(host: str, port: int):
    s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
    try:
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_V6ONLY, 0)
        s.bind((host, port))
        s.listen(10)
        yield s
    finally:
        s.close()


with server(host, port) as s:
    try:
        while True:
            conn, addr = s.accept()
            remote = conn.getpeername()
            print(remote)
            msg = conn.recv(1024)

            if msg:
                conn.send(msg)

            conn.close()
    except KeyboardInterrupt:
        pass
```

output:

```
$ python3 ipv6.py
[1] 23914
$ nc -4 127.0.0.1 5566
('::ffff:127.0.0.1', 42604, 0, 0)
Hello IPv4
Hello IPv4
$ nc -6 ::1 5566
('::1', 50882, 0, 0)
Hello IPv6
Hello IPv6
$ nc -6 fe80::a00:27ff:fe9b:50ee%enp0s3 5566
```

```
('fe80::a00:27ff:fe9b:50ee%enp0s3', 42042, 0, 2)
Hello IPv6
Hello IPv6
```

5.8 Simple TCP Echo Server Via SocketServer

```
>>> import SocketServer
>>> bh = SocketServer.BaseRequestHandler
>>> class handler(bh):
...   def handle(self):
...     data = self.request.recv(1024)
...     print self.client_address
...     self.request.sendall(data)
...
>>> host = ('localhost',5566)
>>> s = SocketServer.TCPServer(
...   host, handler)
>>> s.serve_forever()
```

output:

```
$ nc localhost 5566
Hello World
Hello World
```

5.9 Simple TLS/SSL TCP Echo Server

```
import socket
import ssl

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('localhost', 5566))
sock.listen(10)

sslctx = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
sslctx.load_cert_chain(certfile='./root-ca.crt',
                       keyfile='./root-ca.key')

try:
    while True:
        conn, addr = sock.accept()
        sslconn = sslctx.wrap_socket(conn, server_side=True)
        msg = sslconn.recv(1024)
        if msg:
            sslconn.send(msg)
        sslconn.close()
finally:
    sock.close()
```

output:

```
# console 1
$ openssl genrsa -out root-ca.key 2048
$ openssl req -x509 -new -nodes -key root-ca.key -days 365 -out root-ca.crt
$ python3 ssl_tcp_server.py

# console 2
$ openssl s_client -connect localhost:5566
...
Hello SSL
Hello SSL
read:errno=0
```

5.10 Set ciphers on TLS/SSL TCP Echo Server

```
import socket
import json
import ssl

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('localhost', 5566))
sock.listen(10)

sslctx = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
sslctx.load_cert_chain(certfile='cert.pem',
                       keyfile='key.pem')
# set ssl ciphers
sslctx.set_ciphers('ECDH-ECDSA-AES128-GCM-SHA256')
print(json.dumps(sslctx.get_ciphers(), indent=2))

try:
    while True:
        conn, addr = sock.accept()
        sslconn = sslctx.wrap_socket(conn, server_side=True)
        msg = sslconn.recv(1024)
        if msg:
            sslconn.send(msg)
        sslconn.close()
finally:
    sock.close()
```

output:

```
$ openssl ecparam -out key.pem -genkey -name prime256v1
$ openssl req -x509 -new -key key.pem -out cert.pem
$ python3 tls.py&
[2] 64565
[
  {
    "id": 50380845,
    "name": "ECDH-ECDSA-AES128-GCM-SHA256",
    "protocol": "TLSv1/SSLv3",
    "description": "ECDH-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH/ECDSA Au=ECDH␣
↪Enc=AESGCM(128) Mac=AEAD",
    "strength_bits": 128,
```

```
    "alg_bits": 128
  }
]
$ openssl s_client -connect localhost:5566 -cipher "ECDH-ECDSA-AES128-GCM-SHA256"
...
---
Hello ECDH-ECDSA-AES128-GCM-SHA256
Hello ECDH-ECDSA-AES128-GCM-SHA256
read:errno=0
```

5.11 Simple UDP Echo Server

```
import socket

class UDPServer(object):
    def __init__(self,host,port):
        self._host = host
        self._port = port

    def __enter__(self):
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.bind((self._host,self._port))
        self._sock = sock
        return sock
   def __exit__(self,*exc_info):
        if exc_info[0]:
            import traceback
            traceback.print_exception(*exc_info)
        self._sock.close()

if __name__ == '__main__':
    host = 'localhost'
    port = 5566
    with UDPServer(host,port) as s:
        while True:
            msg, addr = s.recvfrom(1024)
            s.sendto(msg, addr)
```

output:

```
$ nc -u localhost 5566
Hello World
Hello World
```

5.12 Simple UDP Echo Server Via SocketServer

```
>>> import SocketServer
>>> bh = SocketServer.BaseRequestHandler
>>> class handler(bh):
...   def handle(self):
...     m,s = self.request
...     s.sendto(m,self.client_address)
```

```
...     print self.client_address
...
>>> host = ('localhost',5566)
>>> s = SocketServer.UDPServer(
...   host, handler)
>>> s.serve_forever()
```

output:

```
$ nc -u localhost 5566
Hello World
Hello World
```

5.13 Simple UDP client - Sender

```
>>> import socket
>>> import time
>>> sock = socket.socket(
...   socket.AF_INET,
...   socket.SOCK_DGRAM)
>>> host = ('localhost',5566)
>>> while True:
...   sock.sendto("Hello\n",host)
...   time.sleep(5)
...
```

output:

```
$ nc -lu localhost 5566
Hello
Hello
```

5.14 Broadcast UDP Packets

```
>>> import socket
>>> import time
>>> sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
>>> sock.bind(('',0))
>>> sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST,1)
>>> while True:
...   m = '{0}\n'.format(time.time())
...   sock.sendto(m,('<broadcast>',5566))
...   time.sleep(5)
...
```

output:

```
$ nc -k -w 1 -ul 5566
1431473025.72
```

5.15 Simple UNIX Domain Socket

```
import socket
import contextlib
import os

@contextlib.contextmanager
def DomainServer(addr):
    try:
        if os.path.exists(addr):
            os.unlink(addr)
        sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
        sock.bind(addr)
        sock.listen(10)
        yield sock
    finally:
        sock.close()
        if os.path.exists(addr):
            os.unlink(addr)

addr = "./domain.sock"
with DomainServer(addr) as sock:
    while True:
        conn, _ = sock.accept()
        msg = conn.recv(1024)
        conn.send(msg)
        conn.close()
```

output:

```
$ nc -U ./domain.sock
Hello
Hello
```

5.16 Simple duplex processes communication

```
import os
import socket

child, parent = socket.socketpair()
pid = os.fork()
try:

    if pid == 0:
        print('chlid pid: {}'.format(os.getpid()))

        child.send(b'Hello Parent')
        msg = child.recv(1024)
        print('p[{}] <--- c[{}]: {}'.format(
            os.getppid(), os.getpid(), msg))
    else:
        print('parent pid: {}'.format(os.getpid()))

        # simple echo server (parent)
        msg = parent.recv(1024)
```

```
        print('c[{}] ---> p[{}]: {}'.format(
                pid, os.getpid(), msg))
        parent.send(msg)

except KeyboardInterrupt:
    pass
finally:
    child.close()
    parent.close()
```

output:

```
$ python3 socketpair_demo.py
parent pid: 9497
chlid pid: 9498
c[9498] ---> p[9497]: b'Hello Parent'
p[9497] ---> c[9498]: b'Hello Parent'
```

5.17 Simple Asynchronous TCP Server - Thread

```
>>> from threading import Thread
>>> import socket
>>> def work(conn):
...   while True:
...     msg = conn.recv(1024)
...     conn.send(msg)
...
>>> sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
>>> sock.bind(('localhost',5566))
>>> sock.listen(5)
>>> while True:
...   conn,addr = sock.accept()
...   t=Thread(target=work,args=(conn,))
...   t.daemon=True
...   t.start()
...
```

output: (bash 1)

```
$ nc localhost 5566
Hello
Hello
```

output: (bash 2)

```
$ nc localhost 5566
Ker Ker
Ker Ker
```

5.18 Simple Asynchronous TCP Server - select

```
from select import select
import socket

host = ('localhost',5566)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
sock.bind(host)
sock.listen(5)
rl = [sock]
wl = []
ml = {}
try:
    while True:
        r, w, _ = select(rl,wl,[])
        # process ready to ready
        for _ in r:
            if _ == sock:
                conn, addr = sock.accept()
                rl.append(conn)
            else:
                msg = _.recv(1024)
                ml[_.fileno()] = msg
                wl.append(_)
        # process ready to write
        for _ in w:
            msg = ml[_.fileno()]
            _.send(msg)
            wl.remove(_)
            del ml[_.fileno()]
except:
    sock.close()
```

output: (bash 1)

```
$ nc localhost 5566
Hello
Hello
```

output: (bash 2)

```
$ nc localhost 5566
Ker Ker
Ker Ker
```

5.19 Simple Asynchronous TCP Server - poll

```
from __future__ import print_function, unicode_literals

import socket
import select
import contextlib

host = 'localhost'
```

```
port = 5566

con = {}
req = {}
resp = {}

@contextlib.contextmanager
def Server(host,port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.setblocking(False)
        s.bind((host,port))
        s.listen(10)
        yield s
    except socket.error:
        print("Get socket error")
        raise
    finally:
        if s: s.close()


@contextlib.contextmanager
def Poll():
    try:
        e = select.poll()
        yield e
    finally:
        for fd, c in con.items():
            e.unregister(fd)
            c.close()


def accept(server, poll):
    conn, addr = server.accept()
    conn.setblocking(False)
    fd = conn.fileno()
    poll.register(fd, select.POLLIN)
    req[fd] = conn
    con[fd] = conn


def recv(fd, poll):
    if fd not in req:
        return

    conn = req[fd]
    msg = conn.recv(1024)
    if msg:
        resp[fd] = msg
        poll.modify(fd, select.POLLOUT)
    else:
        conn.close()
        del con[fd]

    del req[fd]
```

```
def send(fd, poll):
    if fd not in resp:
        return

    conn = con[fd]
    msg = resp[fd]
    b = 0
    total = len(msg)
    while total > b:
        l = conn.send(msg)
        msg = msg[l:]
        b += l

    del resp[fd]
    req[fd] = conn
    poll.modify(fd, select.POLLIN)

try:
    with Server(host, port) as server, Poll() as poll:

        poll.register(server.fileno())

        while True:
            events = poll.poll(1)
            for fd, e in events:
                if fd == server.fileno():
                    accept(server, poll)
                elif e & (select.POLLIN | select.POLLPRI):
                    recv(fd, poll)
                elif e & select.POLLOUT:
                    send(fd, poll)
except KeyboardInterrupt:
    pass
```

output: (bash 1)

```
$ python3 poll.py &
[1] 3036
$ nc localhost 5566
Hello poll
Hello poll
Hello Python Socket Programming
Hello Python Socket Programming
```

output: (bash 2)

```
$ nc localhost 5566
Hello Python
Hello Python
Hello Awesome Python
Hello Awesome Python
```

5.20 Simple Asynchronous TCP Server - epoll

```
from __future__ import print_function, unicode_literals

import socket
import select
import contextlib


host = 'localhost'
port = 5566

con = {}
req = {}
resp = {}

@contextlib.contextmanager
def Server(host,port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.setblocking(False)
        s.bind((host,port))
        s.listen(10)
        yield s
    except socket.error:
        print("Get socket error")
        raise
    finally:
        if s: s.close()


@contextlib.contextmanager
def Epoll():
    try:
        e = select.epoll()
        yield e
    finally:
        for fd in con: e.unregister(fd)
        e.close()


def accept(server, epoll):
    conn, addr = server.accept()
    conn.setblocking(0)
    fd = conn.fileno()
    epoll.register(fd, select.EPOLLIN)
    req[fd] = conn
    con[fd] = conn


def recv(fd, epoll):
    if fd not in req:
        return

    conn = req[fd]
    msg = conn.recv(1024)
    if msg:
```

```
        resp[fd] = msg
        epoll.modify(fd, select.EPOLLOUT)
    else:
        conn.close()
        del con[fd]

    del req[fd]


def send(fd, epoll):
    if fd not in resp:
        return

    conn = con[fd]
    msg = resp[fd]
    b = 0
    total = len(msg)
    while total > b:
        l = conn.send(msg)
        msg = msg[l:]
        b += l

    del resp[fd]
    req[fd] = conn
    epoll.modify(fd, select.EPOLLIN)


try:
    with Server(host, port) as server, Epoll() as epoll:

        epoll.register(server.fileno())

        while True:
            events = epoll.poll(1)
            for fd, e in events:
                if fd == server.fileno():
                    accept(server, epoll)
                elif e & select.EPOLLIN:
                    recv(fd, epoll)
                elif e & select.EPOLLOUT:
                    send(fd, epoll)
except KeyboardInterrupt:
    pass
```

output: (bash 1)

```
$ python3 epoll.py &
[1] 3036
$ nc localhost 5566
Hello epoll
Hello epoll
Hello Python Socket Programming
Hello Python Socket Programming
```

output: (bash 2)

```
$ nc localhost 5566
Hello Python
```

```
Hello Python
Hello Awesome Python
Hello Awesome Python
```

5.21 Simple Asynchronous TCP Server - kqueue

```
from __future__ import print_function, unicode_literals

import socket
import select
import contextlib

if not hasattr(select, 'kqueue'):
    print("Not support kqueue")
    exit(1)


host = 'localhost'
port = 5566

con = {}
req = {}
resp = {}

@contextlib.contextmanager
def Server(host,port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.setblocking(False)
        s.bind((host,port))
        s.listen(10)
        yield s
    except socket.error:
        print("Get socket error")
        raise
    finally:
        if s: s.close()


@contextlib.contextmanager
def Kqueue():
    try:
        kq = select.kqueue()
        yield kq
    finally:
        kq.close()
        for fd, c in con.items(): c.close()


def accept(server, kq):
    conn, addr = server.accept()
    conn.setblocking(False)
    fd = conn.fileno()
    ke = select.kevent(conn.fileno(),
```

```
                        select.KQ_FILTER_READ,
                        select.KQ_EV_ADD)
    kq.control([ke], 0)
    req[fd] = conn
    con[fd] = conn


def recv(fd, kq):
    if fd not in req:
        return

    conn = req[fd]
    msg = conn.recv(1024)
    if msg:
        resp[fd] = msg
        # remove read event
        ke = select.kevent(fd,
                           select.KQ_FILTER_READ,
                           select.KQ_EV_DELETE)
        kq.control([ke], 0)
        # add write event
        ke = select.kevent(fd,
                           select.KQ_FILTER_WRITE,
                           select.KQ_EV_ADD)
        kq.control([ke], 0)
        req[fd] = conn
        con[fd] = conn
    else:
        conn.close()
        del con[fd]

    del req[fd]


def send(fd, kq):
    if fd not in resp:
        return

    conn = con[fd]
    msg = resp[fd]
    b = 0
    total = len(msg)
    while total > b:
        l = conn.send(msg)
        msg = msg[l:]
        b += l

    del resp[fd]
    req[fd] = conn
    # remove write event
    ke = select.kevent(fd,
                       select.KQ_FILTER_WRITE,
                       select.KQ_EV_DELETE)
    kq.control([ke], 0)
    # add read event
    ke = select.kevent(fd,
                       select.KQ_FILTER_READ,
                       select.KQ_EV_ADD)
```

```
    kq.control([ke], 0)


try:
    with Server(host, port) as server, Kqueue() as kq:

        max_events = 1024
        timeout = 1

        ke = select.kevent(server.fileno(),
                           select.KQ_FILTER_READ,
                           select.KQ_EV_ADD)

        kq.control([ke], 0)
        while True:
            events = kq.control(None, max_events, timeout)
            for e in events:
                fd = e.ident
                if fd == server.fileno():
                    accept(server, kq)
                elif e.filter == select.KQ_FILTER_READ:
                    recv(fd, kq)
                elif e.filter == select.KQ_FILTER_WRITE:
                    send(fd, kq)
except KeyboardInterrupt:
    pass
```

output: (bash 1)

```
$ python3 kqueue.py &
[1] 3036
$ nc localhost 5566
Hello kqueue
Hello kqueue
Hello Python Socket Programming
Hello Python Socket Programming
```

output: (bash 2)

```
$ nc localhost 5566
Hello Python
Hello Python
Hello Awesome Python
Hello Awesome Python
```

5.22 High-Level API - selectors

```
# Pyton3.4+ only
# Reference: selectors
import selectors
import socket
import contextlib

@contextlib.contextmanager
def Server(host,port):
```

```
   try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind((host,port))
        s.listen(10)
        sel = selectors.DefaultSelector()
        yield s, sel
    except socket.error:
        print("Get socket error")
        raise
    finally:
        if s:
            s.close()

def read_handler(conn, sel):
    msg = conn.recv(1024)
    if msg:
        conn.send(msg)
    else:
        sel.unregister(conn)
        conn.close()

def accept_handler(s, sel):
    conn, _ = s.accept()
    sel.register(conn, selectors.EVENT_READ, read_handler)

host = 'localhost'
port = 5566
with Server(host, port) as (s,sel):
    sel.register(s, selectors.EVENT_READ, accept_handler)
    while True:
        events = sel.select()
        for sel_key, m in events:
            handler = sel_key.data
            handler(sel_key.fileobj, sel)
```

output: (bash 1)

```
$ nc localhost 5566
Hello
Hello
```

output: (bash 1)

```
$ nc localhost 5566
Hi
Hi
```

5.23 Simple Non-blocking TLS/SSL socket via selectors

```
import socket
import selectors
import contextlib
import ssl
```

```
from functools import partial

sslctx = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
sslctx.load_cert_chain(certfile="cert.pem", keyfile="key.pem")

@contextlib.contextmanager
def Server(host,port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind((host,port))
        s.listen(10)
        sel = selectors.DefaultSelector()
        yield s, sel
    except socket.error:
        print("Get socket error")
        raise
    finally:
        if s: s.close()
        if sel: sel.close()


def accept(s, sel):
    conn, _ = s.accept()
    sslconn = sslctx.wrap_socket(conn,
                                 server_side=True,
                                 do_handshake_on_connect=False)
    sel.register(sslconn, selectors.EVENT_READ, do_handshake)


def do_handshake(sslconn, sel):
    sslconn.do_handshake()
    sel.modify(sslconn, selectors.EVENT_READ, read)


def read(sslconn, sel):
    msg = sslconn.recv(1024)
    if msg:
        sel.modify(sslconn,
                   selectors.EVENT_WRITE,
                   partial(write, msg=msg))
    else:
        sel.unregister(sslconn)
        sslconn.close()


def write(sslconn, sel, msg=None):
    if msg:
        sslconn.send(msg)
    sel.modify(sslconn, selectors.EVENT_READ, read)


host = 'localhost'
port = 5566
try:
    with Server(host, port) as (s,sel):
        sel.register(s, selectors.EVENT_READ, accept)
        while True:
```

```
            events = sel.select()
            for sel_key, m in events:
                handler = sel_key.data
                handler(sel_key.fileobj, sel)
except KeyboardInterrupt:
    pass
```

output:

```
# console 1
$ openssl genrsa -out key.pem 2048
$ openssl req -x509 -new -nodes -key key.pem -days 365 -out cert.pem
$ python3 ssl_tcp_server.py &
$ openssl s_client -connect localhost:5566
...
---
Hello TLS
Hello TLS

# console 2
$ openssl s_client -connect localhost:5566
...
---
Hello SSL
Hello SSL
```

5.24 “socketpair” - Similar to PIPE

```
import socket
import os
import time

c_s, p_s = socket.socketpair()
try:
    pid = os.fork()
except OSError:
    print "Fork Error"
    raise

if pid:
    # parent process
    c_s.close()
    while True:
        p_s.sendall("Hi! Child!")
        msg = p_s.recv(1024)
        print msg
        time.sleep(3)
    os.wait()
else:
    # child process
    p_s.close()
    while True:
        msg = c_s.recv(1024)
        print msg
        c_s.sendall("Hi! Parent!")
```

output:

```
$ python ex.py
Hi! Child!
Hi! Parent!
Hi! Child!
Hi! Parent!
...
```

5.25 Using sendfile do copy

```
# need python 3.3 or above
from __future__ import print_function, unicode_literals

import os
import sys

if len(sys.argv) != 3:
    print("Usage: cmd src dst")
    exit(1)

src = sys.argv[1]
dst = sys.argv[2]

with open(src, 'r') as s, open(dst, 'w') as d:
    st = os.fstat(s.fileno())

    offset = 0
    count = 4096
    s_len = st.st_size

    sfd = s.fileno()
    dfd = d.fileno()

    while s_len > 0:
        ret = os.sendfile(dfd, sfd, offset, count)
        offset += ret
        s_len -= ret
```

output:

```
$ dd if=/dev/urandom of=dd.in bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 108.02 s, 9.9 MB/s
$ python3 sendfile.py dd.in dd.out
$ md5sum dd.in
e79afdd6aba71b7174142c0bbc289674  dd.in
$ md5sum dd.out
e79afdd6aba71b7174142c0bbc289674  dd.out
```

5.26 Sending a file through sendfile

```
# need python 3.5 or above
from __future__ import print_function, unicode_literals

import os
import sys
import time
import socket
import contextlib

@contextlib.contextmanager
def server(host, port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind((host, port))
        s.listen(10)
        yield s
    finally:
        s.close()


@contextlib.contextmanager
def client(host, port):
    try:
        c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        c.connect((host, port))
        yield c
    finally:
        c.close()


def do_sendfile(fout, fin, count, fin_len):
    l = fin_len
    offset = 0
    while l > 0:
        ret = fout.sendfile(fin, offset, count)
        offset += ret
        l -= ret


def do_recv(fout, fin):
    while True:
        data = fin.recv(4096)

        if not data: break

        fout.write(data)


host = 'localhost'
port = 5566

if len(sys.argv) != 3:
    print("usage: cmd src dst")
    exit(1)
```

```
src = sys.argv[1]
dst = sys.argv[2]
offset = 0

pid = os.fork()

if pid ==  0:
    # client
    time.sleep(3)
    with client(host, port) as c, open(src, 'rb') as f:
        fd = f.fileno()
        st = os.fstat(fd)
        count = 4096

        flen = st.st_size
        do_sendfile(c, f, count, flen)

else:
    # server
    with server(host, port) as s, open(dst, 'wb') as f:
        conn, addr = s.accept()
        do_recv(f, conn)
```

output:

```
$ dd if=/dev/urandom of=dd.in bs=1M count=512
512+0 records in
512+0 records out
536870912 bytes (537 MB, 512 MiB) copied, 3.17787 s, 169 MB/s
$ python3 sendfile.py dd.in dd.out
$ md5sum dd.in
eadfd96c85976b1f46385e89dfd9c4a8  dd.in
$ md5sum dd.out
eadfd96c85976b1f46385e89dfd9c4a8  dd.out
```

5.27 Linux kernel Crypto API - AF_ALG

```
# need python 3.6 or above & Linux >=2.6.38
import socket
import hashlib
import contextlib

@contextlib.contextmanager
def create_alg(typ, name):
    s = socket.socket(socket.AF_ALG, socket.SOCK_SEQPACKET, 0)
    try:
        s.bind((typ, name))
        yield s
    finally:
        s.close()

msg = b'Python is awesome!'

with create_alg('hash', 'sha256') as algo:
    op, _ = algo.accept()
```

```
    with op:
        op.sendall(msg)
        data = op.recv(512)
        print(data.hex())

        # check data
        h = hashlib.sha256(msg).digest()
        if h != data:
            raise Exception(f"sha256({h}) != af_alg({data})")
```

output:

```
$ python3 af_alg.py
9d50bcac2d5e33f936ec2db7dc7b6579cba8e1b099d77c31d8564df46f66bdf5
```

5.28 AES-CBC encrypt/decrypt via AF_ALG

```
# need python 3.6 or above & Linux >=4.3
import contextlib
import socket
import os

BS = 16  # Bytes
pad = lambda s: s + (BS - len(s) % BS) * \
                chr(BS - len(s) % BS).encode('utf-8')

upad = lambda s : s[0:-s[-1]]


@contextlib.contextmanager
def create_alg(typ, name):
    s = socket.socket(socket.AF_ALG, socket.SOCK_SEQPACKET, 0)
    try:
        s.bind((typ, name))
        yield s
    finally:
        s.close()


def encrypt(plaintext, key, iv):
    ciphertext = None
    with create_alg('skcipher', 'cbc(aes)') as algo:
        algo.setsockopt(socket.SOL_ALG, socket.ALG_SET_KEY, key)
        op, _ = algo.accept()
        with op:
            plaintext = pad(plaintext)
            op.sendmsg_afalg([plaintext],
                             op=socket.ALG_OP_ENCRYPT,
                             iv=iv)
            ciphertext = op.recv(len(plaintext))

    return ciphertext


def decrypt(ciphertext, key, iv):
```

```
    plaintext = None
    with create_alg('skcipher', 'cbc(aes)') as algo:
        algo.setsockopt(socket.SOL_ALG, socket.ALG_SET_KEY, key)
        op, _ = algo.accept()
        with op:
            op.sendmsg_afalg([ciphertext],
                             op=socket.ALG_OP_DECRYPT,
                             iv=iv)
            plaintext = op.recv(len(ciphertext))

    return upad(plaintext)


key = os.urandom(32)
iv  = os.urandom(16)

plaintext = b"Demo AF_ALG"
ciphertext = encrypt(plaintext, key, iv)
plaintext = decrypt(ciphertext, key, iv)

print(ciphertext.hex())
print(plaintext)
```

output:

```
$ python3 aes_cbc.py
01910e4bd6932674dba9bebd4fdf6cf2
b'Demo AF_ALG'
```

5.29 AES-GCM encrypt/decrypt via AF_ALG

```
# need python 3.6 or above & Linux >=4.9
import contextlib
import socket
import os

@contextlib.contextmanager
def create_alg(typ, name):
    s = socket.socket(socket.AF_ALG, socket.SOCK_SEQPACKET, 0)
    try:
        s.bind((typ, name))
        yield s
    finally:
        s.close()


def encrypt(key, iv, assoc, taglen, plaintext):
    """ doing aes-gcm encrypt

    :param key: the aes symmetric key
    :param iv: initial vector
    :param assoc: associated data (integrity protection)
    :param taglen: authenticator tag len
    :param plaintext: plain text data
    """
```

```
    assoclen = len(assoc)
    ciphertext = None
    tag = None

    with create_alg('aead', 'gcm(aes)') as algo:
        algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_KEY, key)
        algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_AEAD_AUTHSIZE,
                        None,
                        assoclen)

        op, _ = algo.accept()
        with op:
            msg = assoc + plaintext
            op.sendmsg_afalg([msg],
                             op=socket.ALG_OP_ENCRYPT,
                             iv=iv,
                             assoclen=assoclen)

            res = op.recv(assoclen + len(plaintext) + taglen)
            ciphertext = res[assoclen:-taglen]
            tag = res[-taglen:]

    return ciphertext, tag


def decrypt(key, iv, assoc, tag, ciphertext):
    """ doing aes-gcm decrypt

    :param key: the AES symmetric key
    :param iv: initial vector
    :param assoc: associated data (integrity protection)
    :param tag: the GCM authenticator tag
    :param ciphertext: cipher text data
    """
    plaintext = None
    assoclen = len(assoc)

    with create_alg('aead', 'gcm(aes)') as algo:
        algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_KEY, key)
        algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_AEAD_AUTHSIZE,
                        None,
                        assoclen)
        op, _ = algo.accept()
        with op:
            msg = assoc + ciphertext + tag
            op.sendmsg_afalg([msg],
                             op=socket.ALG_OP_DECRYPT, iv=iv,
                             assoclen=assoclen)

            taglen = len(tag)
            res = op.recv(len(msg) - taglen)
            plaintext = res[assoclen:]
```

```
    return plaintext

key = os.urandom(16)
iv  = os.urandom(12)
assoc = os.urandom(16)

plaintext = b"Hello AES-GCM"
ciphertext, tag = encrypt(key, iv, assoc, 16, plaintext)
plaintext = decrypt(key, iv, assoc, tag, ciphertext)

print(ciphertext.hex())
print(plaintext)
```

output:

```
$ $ python3 aes_gcm.py
2e27b67234e01bcb0ab6b451f4f870ce
b'Hello AES-GCM'
```

5.30 AES-GCM encrypt/decrypt file with sendfile

```
# need python 3.6 or above & Linux >=4.9
import contextlib
import socket
import sys
import os

@contextlib.contextmanager
def create_alg(typ, name):
    s = socket.socket(socket.AF_ALG, socket.SOCK_SEQPACKET, 0)
    try:
        s.bind((typ, name))
        yield s
    finally:
        s.close()


def encrypt(key, iv, assoc, taglen, pfile):
    assoclen = len(assoc)
    ciphertext = None
    tag = None

    pfd = pfile.fileno()
    offset = 0
    st = os.fstat(pfd)
    totalbytes = st.st_size

    with create_alg('aead', 'gcm(aes)') as algo:
        algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_KEY, key)
        algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_AEAD_AUTHSIZE,
                        None,
                        assoclen)
```

```
        op, _ = algo.accept()
        with op:
            op.sendmsg_afalg(op=socket.ALG_OP_ENCRYPT,
                             iv=iv,
                             assoclen=assoclen,
                             flags=socket.MSG_MORE)

            op.sendall(assoc, socket.MSG_MORE)

            # using sendfile to encrypt file data
            os.sendfile(op.fileno(), pfd, offset, totalbytes)

            res = op.recv(assoclen + totalbytes + taglen)
            ciphertext = res[assoclen:-taglen]
            tag = res[-taglen:]

    return ciphertext, tag


def decrypt(key, iv, assoc, tag, ciphertext):
    plaintext = None
    assoclen = len(assoc)

    with create_alg('aead', 'gcm(aes)') as algo:
        algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_KEY, key)
        algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_AEAD_AUTHSIZE,
                        None,
                        assoclen)
        op, _ = algo.accept()
        with op:
            msg = assoc + ciphertext + tag
            op.sendmsg_afalg([msg],
                             op=socket.ALG_OP_DECRYPT, iv=iv,
                             assoclen=assoclen)

            taglen = len(tag)
            res = op.recv(len(msg) - taglen)
            plaintext = res[assoclen:]

    return plaintext

key = os.urandom(16)
iv  = os.urandom(12)
assoc = os.urandom(16)

if len(sys.argv) != 2:
    print("usage: cmd plain")
    exit(1)

plain = sys.argv[1]

with open(plain, 'r') as pf:
    ciphertext, tag = encrypt(key, iv, assoc, 16, pf)
    plaintext = decrypt(key, iv, assoc, tag, ciphertext)

    print(ciphertext.hex())
```

```
    print(plaintext)
```

output:

```
$ echo "Test AES-GCM with sendfile" > plain.txt
$ python3 aes_gcm.py plain.txt
b3800044520ed07fa7f20b29c2695bae9ab596065359db4f009dd6
b'Test AES-GCM with sendfile\n'
```

5.31 Compare the performance of AF_ALG to cryptography

```
# need python 3.6 or above & Linux >=4.9
import contextlib
import socket
import time
import os

from cryptography.hazmat.primitives.ciphers.aead import AESGCM

@contextlib.contextmanager
def create_alg(typ, name):
    s = socket.socket(socket.AF_ALG, socket.SOCK_SEQPACKET, 0)
    try:
        s.bind((typ, name))
        yield s
    finally:
        s.close()


def encrypt(key, iv, assoc, taglen, op, pfile, psize):
    assoclen = len(assoc)
    ciphertext = None
    tag = None
    offset = 0

    pfd = pfile.fileno()
    totalbytes = psize

    op.sendmsg_afalg(op=socket.ALG_OP_ENCRYPT,
                     iv=iv,
                     assoclen=assoclen,
                     flags=socket.MSG_MORE)

    op.sendall(assoc, socket.MSG_MORE)

    # using sendfile to encrypt file data
    os.sendfile(op.fileno(), pfd, offset, totalbytes)

    res = op.recv(assoclen + totalbytes + taglen)
    ciphertext = res[assoclen:-taglen]
    tag = res[-taglen:]

    return ciphertext, tag
```

```
def decrypt(key, iv, assoc, tag, op, ciphertext):
    plaintext = None
    assoclen = len(assoc)

    msg = assoc + ciphertext + tag
    op.sendmsg_afalg([msg],
                     op=socket.ALG_OP_DECRYPT, iv=iv,
                     assoclen=assoclen)

    taglen = len(tag)
    res = op.recv(len(msg) - taglen)
    plaintext = res[assoclen:]

    return plaintext


key = os.urandom(16)
iv  = os.urandom(12)
assoc = os.urandom(16)
assoclen = len(assoc)

count = 1000000
plain = "tmp.rand"

# crate a tmp file
with open(plain, 'wb') as f:
    f.write(os.urandom(4096))
    f.flush()


# profile AF_ALG with sendfile (zero-copy)
with open(plain, 'rb') as pf,\
     create_alg('aead', 'gcm(aes)') as enc_algo,\
     create_alg('aead', 'gcm(aes)') as dec_algo:

    enc_algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_KEY, key)
    enc_algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_AEAD_AUTHSIZE,
                        None,
                        assoclen)

    dec_algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_KEY, key)
    dec_algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_AEAD_AUTHSIZE,
                        None,
                        assoclen)

    enc_op, _ = enc_algo.accept()
    dec_op, _ = dec_algo.accept()

    st = os.fstat(pf.fileno())
    psize = st.st_size

    with enc_op, dec_op:

        s = time.time()
```

```
        for _ in range(count):
            ciphertext, tag = encrypt(key, iv, assoc, 16, enc_op, pf, psize)
            plaintext = decrypt(key, iv, assoc, tag, dec_op, ciphertext)

        cost = time.time() - s

        print(f"total cost time: {cost}. [AF_ALG]")


# profile cryptography (no zero-copy)
with open(plain, 'rb') as pf:

    aesgcm = AESGCM(key)

    s = time.time()

    for _ in range(count):
        pf.seek(0, 0)
        plaintext = pf.read()
        ciphertext = aesgcm.encrypt(iv, plaintext, assoc)
        plaintext = aesgcm.decrypt(iv, ciphertext, assoc)

    cost = time.time() - s

    print(f"total cost time: {cost}. [cryptography]")

# clean up
os.remove(plain)
```

output:

```
$ python3 aes-gcm.py
total cost time: 15.317010641098022. [AF_ALG]
total cost time: 50.256704807281494. [cryptography]
```

5.32 Sniffer IP packets

```
from ctypes import *
import socket
import struct

# ref: IP protocol numbers
PROTO_MAP = {
        1 : "ICMP",
        2 : "IGMP",
        6 : "TCP",
        17: "UDP",
        27: "RDP"}

class IP(Structure):
    ''' IP header Structure

    In linux api, it define as below:
```

```
    strcut ip {
        u_char          ip_hl:4; /* header_len */
        u_char          ip_v:4;  /* version */
        u_char          ip_tos;  /* type of service */
        short           ip_len;  /* total len */
        u_short         ip_id;   /* identification */
        short           ip_off;  /* offset field */
        u_char          ip_ttl;  /* time to live */
        u_char          ip_p;    /* protocol */
        u_short         ip_sum;  /* checksum */
        struct in_addr ip_src;  /* source */
        struct in_addr ip_dst;  /* destination */
    };
    '''
    _fields_ = [("ip_hl" , c_ubyte, 4), # 4 bit
                ("ip_v"  , c_ubyte, 4), # 1 byte
                ("ip_tos", c_uint8),    # 2 byte
                ("ip_len", c_uint16),   # 4 byte
                ("ip_id" , c_uint16),   # 6 byte
                ("ip_off", c_uint16),   # 8 byte
                ("ip_ttl", c_uint8),    # 9 byte
                ("ip_p"  , c_uint8),    # 10 byte
                ("ip_sum", c_uint16),   # 12 byte
                ("ip_src", c_uint32),   # 16 byte
                ("ip_dst", c_uint32)]   # 20 byte

    def __new__(cls, buf=None):
        return cls.from_buffer_copy(buf)
    def __init__(self, buf=None):
        src = struct.pack("<L", self.ip_src)
        self.src = socket.inet_ntoa(src)
        dst = struct.pack("<L", self.ip_dst)
        self.dst = socket.inet_ntoa(dst)
        try:
            self.proto = PROTO_MAP[self.ip_p]
        except KeyError:
            print "{} Not in map".format(self.ip_p)
            raise

host = '0.0.0.0'
s = socket.socket(socket.AF_INET,
                  socket.SOCK_RAW,
                  socket.IPPROTO_ICMP)
s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
s.bind((host, 0))

print "Sniffer start..."
try:
    while True:
        buf = s.recvfrom(65535)[0]
        ip_header = IP(buf[:20])
        print '{0}: {1} -> {2}'.format(ip_header.proto,
                                       ip_header.src,
                                       ip_header.dst)
except KeyboardInterrupt:
    s.close()
```

output: (bash 1)

```
python sniffer.py
Sniffer start...
ICMP: 127.0.0.1 -> 127.0.0.1
ICMP: 127.0.0.1 -> 127.0.0.1
ICMP: 127.0.0.1 -> 127.0.0.1
```

output: (bash 2)

```
$ ping -c 3 localhost
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.063 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.159 ms

--- localhost ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.063/0.103/0.159/0.041 ms
```

5.33 Sniffer TCP packet

```
#!/usr/bin/env python3.6
"""
Based on RFC-793, the following figure shows the TCP header format:

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

In linux api (uapi/linux/tcp.h), it defines the TCP header:

struct tcphdr {
    __be16  source;
    __be16  dest;
    __be32  seq;
    __be32  ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u16   res1:4,
            doff:4,
            fin:1,
```

```
            syn:1,
            rst:1,
            psh:1,
            ack:1,
            urg:1,
            ece:1,
            cwr:1;
#elif defined(__BIG_ENDIAN_BITFIELD)
    __u16   doff:4,
            res1:4,
            cwr:1,
            ece:1,
            urg:1,
            ack:1,
            psh:1,
            rst:1,
            syn:1,
            fin:1;
#else
#error      "Adjust your <asm/byteorder.h> defines"
#endif
    __be16  window;
    __sum16 check;
    __be16  urg_ptr;
};
"""
import sys
import socket
import platform

from struct import unpack
from contextlib import contextmanager

un = platform.system()
if un != "Linux":
    print(f"{un} is not supported!")
    sys.exit(1)

@contextmanager
def create_socket():
    ''' Create a TCP raw socket '''
    s = socket.socket(socket.AF_INET,
                      socket.SOCK_RAW,
                      socket.IPPROTO_TCP)
    try:
        yield s
    finally:
        s.close()


try:
    with create_socket() as s:
        while True:
            pkt, addr = s.recvfrom(65535)

            # the first 20 bytes are ip header
            iphdr = unpack('!BBHHHBBH4s4s', pkt[0:20])
            iplen = (iphdr[0] & 0xf) * 4
```

```
            # the next 20 bytes are tcp header
            tcphdr = unpack('!HHLLBBHHH', pkt[iplen:iplen+20])
            source = tcphdr[0]
            dest = tcphdr[1]
            seq = tcphdr[2]
            ack_seq = tcphdr[3]
            dr = tcphdr[4]
            flags = tcphdr[5]
            window = tcphdr[6]
            check = tcphdr[7]
            urg_ptr = tcphdr[8]

            doff = dr >> 4
            fin = flags & 0x01
            syn = flags & 0x02
            rst = flags & 0x04
            psh = flags & 0x08
            ack = flags & 0x10
            urg = flags & 0x20
            ece = flags & 0x40
            cwr = flags & 0x80

            tcplen = (doff) * 4
            h_size = iplen + tcplen

            #get data from the packet
            data = pkt[h_size:]

            if not data:
                continue

            print("------------ TCP_HEADER --------------")
            print(f"Source Port:           {source}")
            print(f"Destination Port:      {dest}")
            print(f"Sequence Number:       {seq}")
            print(f"Acknowledgment Number: {ack_seq}")
            print(f"Data offset:           {doff}")
            print(f"FIN:                   {fin}")
            print(f"SYN:                   {syn}")
            print(f"RST:                   {rst}")
            print(f"PSH:                   {psh}")
            print(f"ACK:                   {ack}")
            print(f"URG:                   {urg}")
            print(f"ECE:                   {ece}")
            print(f"CWR:                   {cwr}")
            print(f"Window:                {window}")
            print(f"Checksum:              {check}")
            print(f"Urgent Point:          {urg_ptr}")
            print("--------------- DATA -----------------")
            print(data)

except KeyboardInterrupt:
    pass
```

output:

```
$ python3.6 tcp.py
------------ TCP_HEADER --------------
Source Port:           38352
Destination Port:      8000
Sequence Number:       2907801591
Acknowledgment Number: 398995857
Data offset:           8
FIN:                   0
SYN:                   0
RST:                   0
PSH:                   8
ACK:                   16
URG:                   0
ECE:                   0
CWR:                   0
Window:                342
Checksum:              65142
Urgent Point:          0
---------------- DATA -----------------
b'GET / HTTP/1.1\r\nHost: localhost:8000\r\nUser-Agent: curl/7.47.0\r\nAccept: */
↪*\r\n\r\n'
```

5.34 Sniffer ARP packet

```
"""
Ehternet Packet Header

struct ethhdr {
    unsigned char h_dest[ETH_ALEN];   /* destination eth addr */
    unsigned char h_source[ETH_ALEN]; /* source ether addr    */
    __be16        h_proto;            /* packet type ID field */
} __attribute__((packed));

ARP Packet Header

struct arphdr {
    uint16_t htype;    /* Hardware Type           */
    uint16_t ptype;    /* Protocol Type           */
    u_char   hlen;     /* Hardware Address Length */
    u_char   plen;     /* Protocol Address Length */
    uint16_t opcode;   /* Operation Code          */
    u_char   sha[6];   /* Sender hardware address */
    u_char   spa[4];   /* Sender IP address       */
    u_char   tha[6];   /* Target hardware address */
    u_char   tpa[4];   /* Target IP address       */
};
"""

import socket
import struct
import binascii

rawSocket = socket.socket(socket.AF_PACKET,
                          socket.SOCK_RAW,
                          socket.htons(0x0003))
```

```
while True:

    packet = rawSocket.recvfrom(2048)
    ethhdr = packet[0][0:14]
    eth = struct.unpack("!6s6s2s", ethhdr)

    arphdr = packet[0][14:42]
    arp = struct.unpack("2s2s1s1s2s6s4s6s4s", arphdr)
    # skip non-ARP packets
    ethtype = eth[2]
    if ethtype != '\x08\x06': continue

    print "---------------- ETHERNET_FRAME ----------------"
    print "Dest MAC:        ", binascii.hexlify(eth[0])
    print "Source MAC:      ", binascii.hexlify(eth[1])
    print "Type:            ", binascii.hexlify(ethtype)
    print "----------------- ARP_HEADER -------------------"
    print "Hardware type:   ", binascii.hexlify(arp[0])
    print "Protocol type:   ", binascii.hexlify(arp[1])
    print "Hardware size:   ", binascii.hexlify(arp[2])
    print "Protocol size:   ", binascii.hexlify(arp[3])
    print "Opcode:          ", binascii.hexlify(arp[4])
    print "Source MAC:      ", binascii.hexlify(arp[5])
    print "Source IP:       ", socket.inet_ntoa(arp[6])
    print "Dest MAC:        ", binascii.hexlify(arp[7])
    print "Dest IP:         ", socket.inet_ntoa(arp[8])
    print "------------------------------------------------\n"
```

output:

```
$ python arp.py
---------------- ETHERNET_FRAME ----------------
Dest MAC:         ffffffffffff
Source MAC:       f0257252f5ca
Type:             0806
----------------- ARP_HEADER -------------------
Hardware type:    0001
Protocol type:    0800
Hardware size:    06
Protocol size:    04
Opcode:           0001
Source MAC:       f0257252f5ca
Source IP:        140.112.91.254
Dest MAC:         000000000000
Dest IP:          140.112.91.20
------------------------------------------------
```

CHAPTER 6

Python cryptography cheatsheet

Table of Contents

- *Python cryptography cheatsheet*
  - *Simple https server*
  - *Check certificate information*
  - *Generate a self-signed certificate*
  - *Prepare a Certificate Signing Request (csr)*
  - *Generate RSA keyfile without passphrase*
  - *Sign a file by a given private key*
  - *Verify a file from a signed digest*
  - *Simple RSA encrypt via pem file*
  - *Simple RSA encrypt via RSA module*
  - *Simple RSA decrypt via pem file*
  - *Simple RSA encrypt with OAEP*
  - *Simple RSA decrypt with OAEP*
  - *Using DSA to proof of identity*
  - *Using AES CBC mode encrypt a file*
  - *Using AES CBC mode decrypt a file*
  - *AES CBC mode encrypt via password (using cryptography)*
  - *AES CBC mode decrypt via password (using cryptography)*
  - *AES CBC mode encrypt via password (using pycrypto)*

- *AES CBC mode decrypt via password (using pycrytpo)*
- *Ephemeral Diffie Hellman Key Exchange via cryptography*
- *Calculate DH shared key manually via cryptography*
- *Calculate DH shared key from (p, g, pubkey)*

6.1 Simple https server

```
# python2

>>> import BaseHTTPServer, SimpleHTTPServer
>>> import ssl
>>> host, port = 'localhost', 5566
>>> handler = SimpleHTTPServer.SimpleHTTPRequestHandler
>>> httpd = BaseHTTPServer.HTTPServer((host, port), handler)
>>> httpd.socket = ssl.wrap_socket(httpd.socket,
...                                certfile='./cert.crt',
...                                keyfile='./cert.key',
...                                server_side=True)
>>> httpd.serve_forever()

# python3

>>> from http import server
>>> handler = server.SimpleHTTPRequestHandler
>>> import ssl
>>> host, port = 'localhost', 5566
>>> httpd = server.HTTPServer((host, port), handler)
>>> httpd.socket = ssl.wrap_socket(httpd.socket,
...                                certfile='./cert.crt',
...                                keyfile='./cert.key',
...                                server_side=True)
...
>>> httpd.serve_forever()
```

6.2 Check certificate information

```
from cryptography import x509
from cryptography.hazmat.backends import default_backend

backend = default_backend()
with open('./cert.crt', 'rb') as f:
    crt_data = f.read()
    cert = x509.load_pem_x509_certificate(crt_data, backend)

class Certificate:

    _fields = ['country_name',
               'state_or_province_name',
               'locality_name',
               'organization_name',
```

```
                'organizational_unit_name',
                'common_name',
                'email_address']

    def __init__(self, cert):
        assert isinstance(cert, x509.Certificate)
        self._cert = cert
        for attr in self._fields:
            oid = getattr(x509, 'OID_' + attr.upper())
            subject = cert.subject
            info = subject.get_attributes_for_oid(oid)
            setattr(self, attr, info)


cert = Certificate(cert)
for attr in cert._fields:
    for info in getattr(cert, attr):
        print("{}: {}".format(info._oid._name, info._value))
```

output:

```
$ genrsa -out cert.key
Generating RSA private key, 1024 bit long modulus
..........++++++
...++++++
e is 65537 (0x10001)
$ openssl req -x509 -new -nodes \
>       -key cert.key -days 365 \
>       -out cert.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:TW
State or Province Name (full name) [Some-State]:Taiwan
Locality Name (eg, city) []:Taipei
Organization Name (eg, company) [Internet Widgits Pty Ltd]:personal
Organizational Unit Name (eg, section) []:personal
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:test@example.com
$ python3 cert.py
countryName: TW
stateOrProvinceName: Taiwan
localityName: Taipei
organizationName: personal
organizationalUnitName: personal
commonName: localhost
emailAddress: test@example.com
```

6.3 Generate a self-signed certificate

```
from __future__ import print_function, unicode_literals

from datetime import datetime, timedelta
from OpenSSL import crypto

# load private key
ftype = crypto.FILETYPE_PEM
with open('key.pem', 'rb') as f: k = f.read()
k = crypto.load_privatekey(ftype, k)

now    = datetime.now()
expire = now + timedelta(days=365)

# country (countryName, C)
# state or province name (stateOrProvinceName, ST)
# locality (locality, L)
# organization (organizationName, O)
# organizational unit (organizationalUnitName, OU)
# common name (commonName, CN)

cert = crypto.X509()
cert.get_subject().C  = "TW"
cert.get_subject().ST = "Taiwan"
cert.get_subject().L  = "Taipei"
cert.get_subject().O  = "pysheeet"
cert.get_subject().OU = "cheat sheet"
cert.get_subject().CN = "pythonsheets.com"
cert.set_serial_number(1000)
cert.set_notBefore(now.strftime("%Y%m%d%H%M%SZ").encode())
cert.set_notAfter(expire.strftime("%Y%m%d%H%M%SZ").encode())
cert.set_issuer(cert.get_subject())
cert.set_pubkey(k)
cert.sign(k, 'sha1')

with open('cert.pem', "wb") as f:
    f.write(crypto.dump_certificate(ftype, cert))
```

output:

```
$ openssl genrsa -out key.pem 2048
Generating RSA private key, 2048 bit long modulus
.............+++
.................................+++
e is 65537 (0x10001)
$ python3 x509.py
$ openssl x509 -subject -issuer -noout -in cert.pem
subject= /C=TW/ST=Taiwan/L=Taipei/O=pysheeet/OU=cheat sheet/CN=pythonsheets.com
issuer= /C=TW/ST=Taiwan/L=Taipei/O=pysheeet/OU=cheat sheet/CN=pythonsheets.com
```

6.4 Prepare a Certificate Signing Request (csr)

```
from __future__ import print_function, unicode_literals

from OpenSSL import crypto

# load private key
ftype = crypto.FILETYPE_PEM
with open('key.pem', 'rb') as f: key = f.read()
key = crypto.load_privatekey(ftype, key)
req    = crypto.X509Req()

alt_name  = [ b"DNS:www.pythonsheeets.com",
              b"DNS:doc.pythonsheeets.com" ]
key_usage = [ b"Digital Signature",
              b"Non Repudiation",
              b"Key Encipherment" ]

# country (countryName, C)
# state or province name (stateOrProvinceName, ST)
# locality (locality, L)
# organization (organizationName, O)
# organizational unit (organizationalUnitName, OU)
# common name (commonName, CN)

req.get_subject().C  = "TW"
req.get_subject().ST = "Taiwan"
req.get_subject().L  = "Taipei"
req.get_subject().O  = "pysheeet"
req.get_subject().OU = "cheat sheet"
req.get_subject().CN = "pythonsheets.com"
req.add_extensions([
    crypto.X509Extension( b"basicConstraints",
                          False,
                          b"CA:FALSE"),
    crypto.X509Extension( b"keyUsage",
                          False,
                          b",".join(key_usage)),
    crypto.X509Extension( b"subjectAltName",
                          False,
                          b",".join(alt_name))
])

req.set_pubkey(key)
req.sign(key, "sha256")

csr = crypto.dump_certificate_request(ftype, req)
with open("cert.csr", 'wb') as f: f.write(csr)
```

output:

```
# create a root ca
$ openssl genrsa -out ca-key.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....................................+++
e is 65537 (0x10001)
$ openssl req -x509 -new -nodes -key ca-key.pem \
```

```
> -days 10000 -out ca.pem -subj "/CN=root-ca"

# prepare a csr
$ openssl genrsa -out key.pem 2048
Generating RSA private key, 2048 bit long modulus
....+++
.....................................+++
e is 65537 (0x10001)
$ python3 x509.py

# prepare openssl.cnf
cat <<EOF > openssl.cnf
> [req]
> req_extensions = v3_req
> distinguished_name = req_distinguished_name
> [req_distinguished_name]
> [ v3_req ]
> basicConstraints = CA:FALSE
> keyUsage = nonRepudiation, digitalSignature, keyEncipherment
> subjectAltName = @alt_names
> [alt_names]
> DNS.1 = www.pythonsheets.com
> DNS.2 = doc.pythonsheets.com
> EOF

# sign a csr
$ openssl x509 -req -in cert.csr -CA ca.pem \
> -CAkey ca-key.pem -CAcreateserial -out cert.pem \
> -days 365 -extensions v3_req -extfile openssl.cnf
Signature ok
subject=/C=TW/ST=Taiwan/L=Taipei/O=pysheeet/OU=cheat sheet/CN=pythonsheets.com
Getting CA Private Key

# check
$ openssl x509 -in cert.pem -text -noout
```

6.5 Generate RSA keyfile without passphrase

```
# $ openssl genrsa cert.key 2048

>>> from cryptography.hazmat.backends import default_backend
>>> from cryptography.hazmat.primitives import serialization
>>> from cryptography.hazmat.primitives.asymmetric import rsa
>>> key = rsa.generate_private_key(
... public_exponent=65537,
... key_size=2048,
... backend=default_backend())
...
>>> with open('cert.key', 'wb') as f:
...     f.write(key.private_bytes(
...     encoding=serialization.Encoding.PEM,
...     format=serialization.PrivateFormat.TraditionalOpenSSL,
...     encryption_algorithm=serialization.NoEncryption()))
```

6.6 Sign a file by a given private key

```
from __future__ import print_function, unicode_literals

from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256


def signer(privkey, data):
    rsakey = RSA.importKey(privkey)
    signer = PKCS1_v1_5.new(rsakey)
    digest = SHA256.new()
    digest.update(data)
    return signer.sign(digest)


with open('private.key', 'rb') as f: key = f.read()
with open('foo.tgz', 'rb') as f: data = f.read()

sign = signer(key, data)
with open('foo.tgz.sha256', 'wb') as f: f.write(sign)
```

output:

```
# gernerate public & private key
$ openssl genrsa -out private.key 2048
$ openssl rsa -in private.key -pubout -out public.key

$ python3 sign.py
$ openssl dgst -sha256 -verify public.key -signature foo.tgz.sha256 foo.tgz
Verified OK
```

6.7 Verify a file from a signed digest

```
from __future__ import print_function, unicode_literals

import sys

from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256

def verifier(pubkey, sig, data):
    rsakey = RSA.importKey(key)
    signer = PKCS1_v1_5.new(rsakey)
    digest = SHA256.new()

    digest.update(data)
    return signer.verify(digest, sig)


with open("public.key", 'rb') as f: key = f.read()
with open("foo.tgz.sha256", 'rb') as f: sig = f.read()
with open("foo.tgz", 'rb') as f: data = f.read()
```

```
if verifier(key, sig, data):
    print("Verified OK")
else:
    print("Verification Failure")
```

output:

```
# gernerate public & private key
$ openssl genrsa -out private.key 2048
$ openssl rsa -in private.key -pubout -out public.key

# do verification
$ cat /dev/urandom | head -c 512 | base64 > foo.txt
$ tar -zcf foo.tgz foo.txt
$ openssl dgst -sha256 -sign private.key -out foo.tgz.sha256 foo.tgz
$ python3 verify.py
Verified OK

# do verification via openssl
$ openssl dgst -sha256 -verify public.key -signature foo.tgz.sha256 foo.tgz
Verified OK
```

6.8 Simple RSA encrypt via pem file

```
from __future__ import print_function, unicode_literals

import base64
import sys

from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5

key_text = sys.stdin.read()

# import key via rsa module
pubkey = RSA.importKey(key_text)

# create a cipher via PKCS1.5
cipher = PKCS1_v1_5.new(pubkey)

# encrypt
cipher_text = cipher.encrypt(b"Hello RSA!")

# do base64 encode
cipher_text = base64.b64encode(cipher_text)
print(cipher_text.decode('utf-8'))
```

output:

```
$ openssl genrsa -out private.key 2048
$ openssl rsa -in private.key -pubout -out public.key
$ cat public.key                                    |\
> python3 rsa.py                                    |\
> openssl base64 -d -A                              |\
```

```
> openssl rsautl -decrypt -inkey private.key
Hello RSA!
```

6.9 Simple RSA encrypt via RSA module

```
from __future__ import print_function, unicode_literals

import base64
import sys

from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from Crypto.PublicKey.RSA import construct

# prepare public key
e = int('10001', 16)
n = int(sys.stdin.read(), 16)
pubkey = construct((n, e))

# create a cipher via PKCS1.5
cipher = PKCS1_v1_5.new(pubkey)

# encrypt
cipher_text = cipher.encrypt(b"Hello RSA!")

# do base64 encode
cipher_text = base64.b64encode(cipher_text)
print(cipher_text.decode('utf-8'))
```

output:

```
$ openssl genrsa -out private.key 2048
$ openssl rsa -in private.key -pubout -out public.key
$ # check (n, e)
$ openssl rsa -pubin -inform PEM -text -noout < public.key
Public-Key: (2048 bit)
Modulus:
    00:93:d5:58:0c:18:cf:91:f0:74:af:1b:40:09:73:
    0c:d8:13:23:6c:44:60:0d:83:71:e6:f9:61:85:e5:
    b2:d0:8a:73:5c:02:02:51:9a:4f:a7:ab:05:d5:74:
    ff:4d:88:3d:e2:91:b8:b0:9f:7e:a9:a3:b2:3c:99:
    1c:9a:42:4d:ac:2f:6a:e7:eb:0f:a7:e0:a5:81:e5:
    98:49:49:d5:15:3d:53:42:12:08:db:b0:e7:66:2d:
    71:5b:ea:55:4e:2d:9b:40:79:f8:7d:6e:5d:f4:a7:
    d8:13:cb:13:91:c9:ac:5b:55:62:70:44:25:50:ca:
    94:de:78:5d:97:e8:a9:33:66:4f:90:10:00:62:21:
    b6:60:52:65:76:bd:a3:3b:cf:2a:db:3f:66:5f:0d:
    a3:35:ff:29:34:26:6d:63:a2:a6:77:96:5a:84:c7:
    6a:0c:4f:48:52:70:11:8f:85:11:a0:78:f8:60:4b:
    5d:d8:4b:b2:64:e5:ec:99:72:c5:a8:1b:ab:5c:09:
    e1:80:70:91:06:22:ba:97:33:56:0b:65:d8:f3:35:
    66:f8:f9:ea:b9:84:64:8e:3c:14:f7:3d:1f:2c:67:
    ce:64:cf:f9:c5:16:6b:03:a1:7a:c7:fa:4c:38:56:
    ee:e0:4d:5f:ec:46:7e:1f:08:7c:e6:45:a1:fc:17:
    1f:91
```

```
Exponent: 65537 (0x10001)
$ openssl rsa -pubin -in public.key -modulus -noout |\
> cut -d'=' -f 2                                    |\
> python3 rsa.py                                    |\
> openssl base64 -d -A                              |\
> openssl rsautl -decrypt -inkey private.key
Hello RSA!
```

6.10 Simple RSA decrypt via pem file

```
from __future__ import print_function, unicode_literals

import base64
import sys

from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5

# read key file
with open('private.key') as f: key_text = f.read()

# create a private key object
privkey = RSA.importKey(key_text)

# create a cipher object
cipher = PKCS1_v1_5.new(privkey)

# decode base64
cipher_text = base64.b64decode(sys.stdin.read())

# decrypt
plain_text = cipher.decrypt(cipher_text, None)
print(plain_text.decode('utf-8').strip())
```

output:

```
$ openssl genrsa -out private.key 2048
$ openssl rsa -in private.key -pubout -out public.key
$ echo "Hello openssl RSA encrypt"                  |\
> openssl rsautl -encrypt -pubin -inkey public.key |\
> openssl base64 -e -A                             |\
> python3 rsa.py
Hello openssl RSA encrypt
```

6.11 Simple RSA encrypt with OAEP

```
from __future__ import print_function, unicode_literals

import base64
import sys

from Crypto.PublicKey import RSA
```

```
from Crypto.Cipher import PKCS1_OAEP

# read key file
key_text = sys.stdin.read()

# create a public key object
pubkey = RSA.importKey(key_text)

# create a cipher object
cipher = PKCS1_OAEP.new(pubkey)

# encrypt plain text
cipher_text = cipher.encrypt(b"Hello RSA OAEP!")

# encode via base64
cipher_text = base64.b64encode(cipher_text)
print(cipher_text.decode('utf-8'))
```

output:

```
$ openssl genrsa -out private.key 2048
$ openssl rsa -in private.key -pubout -out public.key
$ cat public.key          |\
> python3 rsa.py          |\
> openssl base64 -d -A    |\
> openssl rsautl -decrypt -oaep -inkey private.key
Hello RSA OAEP!
```

6.12 Simple RSA decrypt with OAEP

```
from __future__ import print_function, unicode_literals

import base64
import sys

from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

# read key file
with open('private.key') as f: key_text = f.read()

# create a private key object
privkey = RSA.importKey(key_text)

# create a cipher object
cipher = PKCS1_OAEP.new(privkey)

# decode base64
cipher_text = base64.b64decode(sys.stdin.read())

# decrypt
plain_text = cipher.decrypt(cipher_text)
print(plain_text.decode('utf-8').strip())
```

output:

```
$ openssl genrsa -out private.key 2048
$ openssl rsa -in private.key -pubout -out public.key
$ echo "Hello RSA encrypt via OAEP"                        |\
> openssl rsautl -encrypt -pubin -oaep -inkey public.key |\
> openssl base64 -e -A                                   |\
> python3 rsa.py
Hello RSA encrypt via OAEP
```

6.13 Using DSA to proof of identity

```
import socket

from cryptography.exceptions import InvalidSignature
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import dsa

alice, bob = socket.socketpair()

def gen_dsa_key():
    private_key = dsa.generate_private_key(
        key_size=2048, backend=default_backend())
    return private_key, private_key.public_key()


def sign_data(data, private_key):
    signature = private_key.sign(data, hashes.SHA256())
    return signature


def verify_data(data, signature, public_key):
    try:
        public_key.verify(signature, data, hashes.SHA256())
    except InvalidSignature:
        print("recv msg: {} not trust!".format(data))
    else:
        print("check msg: {} success!".format(data))


# generate alice private & public key
alice_private_key, alice_public_key = gen_dsa_key()

# alice send message to bob, then bob recv
alice_msg = b"Hello Bob"
b = alice.send(alice_msg)
bob_recv_msg = bob.recv(1024)

# alice send signature to bob, then bob recv
signature = sign_data(alice_msg, alice_private_key)
b = alice.send(signature)
bob_recv_signature = bob.recv(1024)

# bob check message recv from alice
verify_data(bob_recv_msg, bob_recv_signature, alice_public_key)
```

```
# attacker modify the msg will make the msg check fail
verify_data(b"I'm attacker!", bob_recv_signature, alice_public_key)
```

output:

```
$ python3 test_dsa.py
check msg: b'Hello Bob' success!
recv msg: b"I'm attacker!" not trust!
```

6.14 Using AES CBC mode encrypt a file

```
from __future__ import print_function, unicode_literals

import struct
import sys
import os

from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import (
    Cipher,
    algorithms,
    modes)

backend = default_backend()
key = os.urandom(32)
iv  = os.urandom(16)

def encrypt(ptext):
    pad = padding.PKCS7(128).padder()
    ptext = pad.update(ptext) + pad.finalize()

    alg = algorithms.AES(key)
    mode = modes.CBC(iv)
    cipher = Cipher(alg, mode, backend=backend)
    encryptor = cipher.encryptor()
    ctext = encryptor.update(ptext) + encryptor.finalize()

    return ctext

print("key: {}".format(key.hex()))
print("iv: {}".format(iv.hex()))

if len(sys.argv) != 3:
    raise Exception("usage: cmd [file] [enc file]")

# read plain text from file
with open(sys.argv[1], 'rb') as f:
    plaintext = f.read()

# encrypt file
ciphertext = encrypt(plaintext)
with open(sys.argv[2], 'wb') as f:
    f.write(ciphertext)
```

output:

```
$ echo "Encrypt file via AES-CBC" > test.txt
$ python3 aes.py test.txt test.enc
key: f239d9609e3f318b7afda7e4bb8db5b8734f504cf67f55e45dfe75f90d24fefc
iv: 8d6383b469f100d25293fb244ccb951e
$ openssl aes-256-cbc -d -in test.enc -out secrets.txt.new            \
> -K f239d9609e3f318b7afda7e4bb8db5b8734f504cf67f55e45dfe75f90d24fefc \
> -iv 8d6383b469f100d25293fb244ccb951e
$ cat secrets.txt.new
Encrypt file via AES-CBC
```

6.15 Using AES CBC mode decrypt a file

```
from __future__ import print_function, unicode_literals

import struct
import sys
import os

from binascii import unhexlify

from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import (
    Cipher,
    algorithms,
    modes)

backend = default_backend()

def decrypt(key, iv, ctext):
    alg = algorithms.AES(key)
    mode = modes.CBC(iv)
    cipher = Cipher(alg, mode, backend=backend)
    decryptor = cipher.decryptor()
    ptext = decryptor.update(ctext) + decryptor.finalize()

    unpadder = padding.PKCS7(128).unpadder() # 128 bit
    ptext = unpadder.update(ptext) + unpadder.finalize()

    return ptext

if len(sys.argv) != 4:
    raise Exception("usage: cmd [key] [iv] [file]")

# read cipher text from file
with open(sys.argv[3], 'rb') as f:
    ciphertext = f.read()

# decrypt file
key, iv = unhexlify(sys.argv[1]), unhexlify(sys.argv[2])
plaintext = decrypt(key, iv, ciphertext)
print(plaintext)
```

output:

```
$ echo "Encrypt file via AES-CBC" > test.txt
$ key=`openssl rand -hex 32`
$ iv=`openssl rand -hex 16`
$ openssl enc -aes-256-cbc -in test.txt -out test.enc -K $key -iv $iv
$ python3 aes.py $key $iv test.enc
```

6.16 AES CBC mode encrypt via password (using cryptography)

```
from __future__ import print_function, unicode_literals

import base64
import struct
import sys
import os

from hashlib import md5, sha1

from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import (
    Cipher,
    algorithms,
    modes)

backend = default_backend()

def EVP_ByteToKey(pwd, md, salt, key_len, iv_len):
    buf = md(pwd + salt).digest()
    d = buf
    while len(buf) < (iv_len + key_len):
        d = md(d + pwd + salt).digest()
        buf += d
    return buf[:key_len], buf[key_len:key_len + iv_len]


def aes_encrypt(pwd, ptext, md):
    key_len, iv_len = 32, 16

    # generate salt
    salt = os.urandom(8)

    # generate key, iv from password
    key, iv = EVP_ByteToKey(pwd, md, salt, key_len, iv_len)

    # pad plaintext
    pad = padding.PKCS7(128).padder()
    ptext = pad.update(ptext) + pad.finalize()

    # create an encryptor
    alg = algorithms.AES(key)
    mode = modes.CBC(iv)
    cipher = Cipher(alg, mode, backend=backend)
    encryptor = cipher.encryptor()

    # encrypt plain text
```

```
    ctext = encryptor.update(ptext) + encryptor.finalize()
    ctext = b'Salted__' + salt + ctext

    # encode base64
    ctext = base64.b64encode(ctext)
    return ctext


if len(sys.argv) != 2: raise Exception("usage: CMD [md]")

md = globals()[sys.argv[1]]

plaintext = sys.stdin.read().encode('utf-8')
pwd = b"password"

print(aes_encrypt(pwd, plaintext, md).decode('utf-8'))
```

output:

```
# with md5 digest
$ echo "Encrypt plaintext via AES-CBC from a given password" |\
> python3 aes.py md5                                         |\
> openssl base64 -d -A                                       |\
> openssl aes-256-cbc -md md5 -d -k password
Encrypt plaintext via AES-CBC from a given password

# with sha1 digest
$ echo "Encrypt plaintext via AES-CBC from a given password" |\
> python3 aes.py sha1                                        |\
> openssl base64 -d -A                                       |\
> openssl aes-256-cbc -md sha1 -d -k password
Encrypt plaintext via AES-CBC from a given password
```

6.17 AES CBC mode decrypt via password (using cryptography)

```
from __future__ import print_function, unicode_literals

import base64
import struct
import sys
import os

from hashlib import md5, sha1

from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import (
    Cipher,
    algorithms,
    modes)

backend = default_backend()

def EVP_ByteToKey(pwd, md, salt, key_len, iv_len):
    buf = md(pwd + salt).digest()
```

```
    d = buf
    while len(buf) < (iv_len + key_len):
        d = md(d + pwd + salt).digest()
        buf += d
    return buf[:key_len], buf[key_len:key_len + iv_len]


def aes_decrypt(pwd, ctext, md):
    ctext = base64.b64decode(ctext)

    # check magic
    if ctext[:8] != b'Salted__':
        raise Exception("bad magic number")

    # get salt
    salt = ctext[8:16]

    # generate key, iv from password
    key, iv = EVP_ByteToKey(pwd, md, salt, 32, 16)

    # decrypt
    alg = algorithms.AES(key)
    mode = modes.CBC(iv)
    cipher = Cipher(alg, mode, backend=backend)
    decryptor = cipher.decryptor()
    ptext = decryptor.update(ctext[16:]) + decryptor.finalize()

    # unpad plaintext
    unpadder = padding.PKCS7(128).unpadder() # 128 bit
    ptext = unpadder.update(ptext) + unpadder.finalize()
    return ptext.strip()

if len(sys.argv) != 2: raise Exception("usage: CMD [md]")

md = globals()[sys.argv[1]]

ciphertext = sys.stdin.read().encode('utf-8')
pwd = b"password"

print(aes_decrypt(pwd, ciphertext, md).decode('utf-8'))
```

output:

```
# with md5 digest
$ echo "Decrypt ciphertext via AES-CBC from a given password" |\
> openssl aes-256-cbc -e -md md5 -salt -A -k password         |\
> openssl base64 -e -A                                        |\
> python3 aes.py md5
Decrypt ciphertext via AES-CBC from a given password

# with sha1 digest
$ echo "Decrypt ciphertext via AES-CBC from a given password" |\
> openssl aes-256-cbc -e -md sha1 -salt -A -k password        |\
> openssl base64 -e -A                                        |\
> python3 aes.py sha1
Decrypt ciphertext via AES-CBC from a given password
```

6.18 AES CBC mode encrypt via password (using pycrypto)

```
from __future__ import print_function, unicode_literals

import struct
import base64
import sys

from hashlib import md5, sha1
from Crypto.Cipher import AES
from Crypto.Random.random import getrandbits

# AES CBC requires blocks to be aligned on 16-byte boundaries.
BS = 16

pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS).encode('utf-8')
unpad = lambda s : s[0:-ord(s[-1])]

def EVP_ByteToKey(pwd, md, salt, key_len, iv_len):
    buf = md(pwd + salt).digest()
    d = buf
    while len(buf) < (iv_len + key_len):
        d = md(d + pwd + salt).digest()
        buf += d
    return buf[:key_len], buf[key_len:key_len + iv_len]


def aes_encrypt(pwd, plaintext, md):
    key_len, iv_len = 32, 16

    # generate salt
    salt = struct.pack('=Q', getrandbits(64))

    # generate key, iv from password
    key, iv = EVP_ByteToKey(pwd, md, salt, key_len, iv_len)

    # pad plaintext
    plaintext = pad(plaintext)

    # create a cipher object
    cipher = AES.new(key, AES.MODE_CBC, iv)

    # ref: openssl/apps/enc.c
    ciphertext = b'Salted__' + salt + cipher.encrypt(plaintext)

    # encode base64
    ciphertext = base64.b64encode(ciphertext)
    return ciphertext

if len(sys.argv) != 2: raise Exception("usage: CMD [md]")

md = globals()[sys.argv[1]]

plaintext = sys.stdin.read().encode('utf-8')
pwd = b"password"

print(aes_encrypt(pwd, plaintext, md).decode('utf-8'))
```

output:

```
# with md5 digest
$ echo "Encrypt plaintext via AES-CBC from a given password" |\
> python3 aes.py md5                                         |\
> openssl base64 -d -A                                       |\
> openssl aes-256-cbc -md md5 -d -k password
Encrypt plaintext via AES-CBC from a given password

# with sha1 digest
$ echo "Encrypt plaintext via AES-CBC from a given password" |\
> python3 aes.py sha1                                        |\
> openssl base64 -d -A                                       |\
> openssl aes-256-cbc -md sha1 -d -k password
Encrypt plaintext via AES-CBC from a given password
```

6.19 AES CBC mode decrypt via password (using pycrytpo)

```
from __future__ import print_function, unicode_literals

import struct
import base64
import sys

from hashlib import md5, sha1
from Crypto.Cipher import AES
from Crypto.Random.random import getrandbits

# AES CBC requires blocks to be aligned on 16-byte boundaries.
BS = 16

unpad = lambda s : s[0:-s[-1]]

def EVP_ByteToKey(pwd, md, salt, key_len, iv_len):
    buf = md(pwd + salt).digest()
    d = buf
    while len(buf) < (iv_len + key_len):
        d = md(d + pwd + salt).digest()
        buf += d
    return buf[:key_len], buf[key_len:key_len + iv_len]


def aes_decrypt(pwd, ciphertext, md):
    ciphertext = base64.b64decode(ciphertext)

    # check magic
    if ciphertext[:8] != b'Salted__':
        raise Exception("bad magic number")

    # get salt
    salt = ciphertext[8:16]

    # get key, iv
    key, iv = EVP_ByteToKey(pwd, md, salt, 32, 16)

    # decrypt
```

```
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(ciphertext[16:])).strip()


if len(sys.argv) != 2: raise Exception("usage: CMD [md]")

md = globals()[sys.argv[1]]

ciphertext = sys.stdin.read().encode('utf-8')
pwd = b"password"

print(aes_decrypt(pwd, ciphertext, md).decode('utf-8'))
```

output:

```
# with md5 digest
$ echo "Decrypt ciphertext via AES-CBC from a given password" |\
> openssl aes-256-cbc -e -md md5 -salt -A -k password         |\
> openssl base64 -e -A                                        |\
> python3 aes.py md5
Decrypt ciphertext via AES-CBC from a given password

# with sha1 digest
$ echo "Decrypt ciphertext via AES-CBC from a given password" |\
> openssl aes-256-cbc -e -md sha1 -salt -A -k password        |\
> openssl base64 -e -A                                        |\
> python3 aes.py sha1
Decrypt ciphertext via AES-CBC from a given password
```

6.20 Ephemeral Diffie Hellman Key Exchange via cryptography

```
>>> from cryptography.hazmat.backends import default_backend
>>> from cryptography.hazmat.primitives.asymmetric import dh
>>> params = dh.generate_parameters(2, 512, default_backend())
>>> a_key = params.generate_private_key()  # alice's private key
>>> b_key = params.generate_private_key()  # bob's private key
>>> a_pub_key = a_key.public_key()
>>> b_pub_key = b_key.public_key()
>>> a_shared_key = a_key.exchange(b_pub_key)
>>> b_shared_key = b_key.exchange(a_pub_key)
>>> a_shared_key == b_shared_key
True
```

6.21 Calculate DH shared key manually via cryptography

```
>>> from cryptography.hazmat.backends import default_backend
>>> from cryptography.hazmat.primitives.asymmetric import dh
>>> from cryptography.utils import int_from_bytes
>>> a_key = params.generate_private_key()  # alice's private key
>>> b_key = params.generate_private_key()  # bob's private key
>>> a_pub_key = a_key.public_key()
>>> b_pub_key = b_key.public_key()
```

```
>>> shared_key = int_from_bytes(a_key.exchange(b_pub_key), 'big')
>>> shared_key_manual = pow(a_pub_key.public_numbers().y,
...                         b_key.private_numbers().x,
...                         params.parameter_numbers().p)
>>> shared_key == shared_key_manual
True
```

6.22 Calculate DH shared key from (p, g, pubkey)

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.utils import int_from_bytes

backend = default_backend()

p = int("11859949538425015739337467917303613431031019140213666"
        "12902540730065402658508634532306628480096346320424639"
        "02565679345822604242388444633308879626896424671 23")

g = 2

y = int("32155788395534640648739966373159697798396966919821525"
        "72238852825117261342483718574508213761865276905503199"
        "96990809820334548136646487475937745447668839 1248")

x = int("40936406544967344339783335855892659846934781346881 6037"
        "26845184711698249073345046319492140506999900861723 1539"
        "71470358966874013508773088997328264463377071 28")

params = dh.DHParameterNumbers(p, g)
public = dh.DHPublicNumbers(y, params)
private = dh.DHPrivateNumbers(x, public)

key = private.private_key(backend)
shared_key = key.exchange(public.public_key(backend))

# check shared key
shared_key = int_from_bytes(shared_key, 'big')
shared_key_manual = pow(y, x, p)    # y^x mod p

assert shared_key == shared_key_manual
```

CHAPTER 7

Python Concurrency Cheatsheet

Table of Contents

- *Python Concurrency Cheatsheet*
  - *Execute a shell command*
  - *Create a thread via "threading"*
  - *Performance Problem - GIL*
  - *Consumer and Producer*
  - *Thread Pool Template*
  - *Using multiprocessing ThreadPool*
  - *Mutex lock*
  - *Deadlock*
  - *Implement "Monitor"*
  - *Control primitive resources*
  - *Ensure tasks has done*
  - *Thread-safe priority queue*
  - *Multiprocessing*
  - *Custom multiprocessing map*
  - *Graceful way to kill all child processes*
  - *Simple round-robin scheduler*
  - *Scheduler with blocking function*
  - *PoolExecutor*

- *What "with ThreadPoolExecutor" doing?*
- *Future Object*
- *Future error handling*

7.1 Execute a shell command

```
# get stdout, stderr, returncode

>>> from subprocess import Popen, PIPE
>>> args = ['time', 'echo', 'hello python']
>>> ret = Popen(args, stdout=PIPE, stderr=PIPE)
>>> out, err = ret.communicate()
>>> out
b'hello python\n'
>>> err
b'        0.00 real         0.00 user         0.00 sys\n'
>>> ret.returncode
0
```

7.2 Create a thread via "threading"

```
>>> from threading import Thread
>>> class Worker(Thread):
...   def __init__(self, id):
...     super(Worker, self).__init__()
...     self._id = id
...   def run(self):
...     print "I am worker %d" % self._id
...
>>> t1 = Worker(1)
>>> t2 = Worker(2)
>>> t1.start(); t2.start()
I am worker 1
I am worker 2

# using function could be more flexible
>>> def Worker(worker_id):
...   print "I am worker %d" % worker_id
...
>>> from threading import Thread
>>> t1 = Thread(target=Worker, args=(1,))
>>> t2 = Thread(target=Worker, args=(2,))
>>> t1.start()
I am worker 1
I am worker 2
```

7.3 Performance Problem - GIL

```
# GIL - Global Interpreter Lock
# see: Understanding the Python GIL
>>> from threading import Thread
>>> def profile(func):
...   def wrapper(*args, **kwargs):
...     import time
...     start = time.time()
...     func(*args, **kwargs)
...     end   = time.time()
...     print end - start
...   return wrapper
...
>>> @profile
... def nothread():
...   fib(35)
...   fib(35)
...
>>> @profile
... def hasthread():
...   t1=Thread(target=fib, args=(35,))
...   t2=Thread(target=fib, args=(35,))
...   t1.start(); t2.start()
...   t1.join(); t2.join()
...
>>> nothread()
9.51164007187
>>> hasthread()
11.3131771088
# !Thread get bad Performance
# since cost on context switch
```

7.4 Consumer and Producer

```
# This architecture make concurrency easy
>>> from threading import Thread
>>> from Queue import Queue
>>> from random import random
>>> import time
>>> q = Queue()
>>> def fib(n):
...   if n<=2:
...     return 1
...   return fib(n-1)+fib(n-2)
...
>>> def producer():
...   while True:
...     wt = random()*5
...     time.sleep(wt)
...     q.put((fib,35))
...
>>> def consumer():
...   while True:
...     task,arg = q.get()
```

```
...     print task(arg)
...     q.task_done()
...
>>> t1 = Thread(target=producer)
>>> t2 = Thread(target=consumer)
>>> t1.start();t2.start()
```

7.5 Thread Pool Template

```
# producer and consumer architecture
from Queue import Queue
from threading import Thread

class Worker(Thread):
   def __init__(self,queue):
      super(Worker, self).__init__()
      self._q = queue
      self.daemon = True
      self.start()
   def run(self):
      while True:
         f,args,kwargs = self._q.get()
         try:
            print f(*args, **kwargs)
         except Exception as e:
            print e
         self._q.task_done()

class ThreadPool(object):
   def __init__(self, num_t=5):
      self._q = Queue(num_t)
      # Create Worker Thread
      for _ in range(num_t):
         Worker(self._q)
   def add_task(self,f,*args,**kwargs):
      self._q.put((f, args, kwargs))
   def wait_complete(self):
      self._q.join()

def fib(n):
   if n <= 2:
      return 1
   return fib(n-1)+fib(n-2)

if __name__ == '__main__':
   pool = ThreadPool()
   for _ in range(3):
      pool.add_task(fib,35)
   pool.wait_complete()
```

7.6 Using multiprocessing ThreadPool

```
# ThreadPool is not in python doc
>>> from multiprocessing.pool import ThreadPool
>>> pool = ThreadPool(5)
>>> pool.map(lambda x: x**2, range(5))
[0, 1, 4, 9, 16]
```

Compare with "map" performance

```
# pool will get bad result since GIL
import time
from multiprocessing.pool import \
     ThreadPool

pool = ThreadPool(10)
def profile(func):
    def wrapper(*args, **kwargs):
       print func.__name__
       s = time.time()
       func(*args, **kwargs)
       e = time.time()
       print "cost: {0}".format(e-s)
    return wrapper

@profile
def pool_map():
    res = pool.map(lambda x:x**2,
                   range(999999))

@profile
def ordinary_map():
    res = map(lambda x:x**2,
              range(999999))

pool_map()
ordinary_map()
```

output:

```
$ python test_threadpool.py
pool_map
cost: 0.562669038773
ordinary_map
cost: 0.38525390625
```

7.7 Mutex lock

Simplest synchronization primitive lock

```
>>> from threading import Thread
>>> from threading import Lock
>>> lock = Lock()
>>> def getlock(id):
...   lock.acquire()
```

```
...     print "task{0} get".format(id)
...     lock.release()
...
>>> t1=Thread(target=getlock,args=(1,))
>>> t2=Thread(target=getlock,args=(2,))
>>> t1.start();t2.start()
task1 get
task2 get

# using lock manager
>>> def getlock(id):
...     with lock:
...       print "task%d get" % id
...
>>> t1=Thread(target=getlock,args=(1,))
>>> t2=Thread(target=getlock,args=(2,))
>>> t1.start();t2.start()
task1 get
task2 get
```

7.8 Deadlock

Happen when more than one mutex lock.

```
>>> import threading
>>> import time
>>> lock1 = threading.Lock()
>>> lock2 = threading.Lock()
>>> def task1():
...     with lock1:
...       print "get lock1"
...       time.sleep(3)
...       with lock2:
...         print "No deadlock"
...
>>> def task2():
...     with lock2:
...       print "get lock2"
...       with lock1:
...         print "No deadlock"
...
>>> t1=threading.Thread(target=task1)
>>> t2=threading.Thread(target=task2)
>>> t1.start();t2.start()
get lock1
 get lock2

>>> t1.isAlive()
True
>>> t2.isAlive()
True
```

7.9 Implement "Monitor"

Using RLock

```
# ref: An introduction to Python Concurrency - David Beazley
from threading import Thread
from threading import RLock
import time

class monitor(object):
   lock = RLock()
   def foo(self,tid):
      with monitor.lock:
         print "%d in foo" % tid
         time.sleep(5)
         self.ker(tid)

   def ker(self,tid):
      with monitor.lock:
         print "%d in ker" % tid
m = monitor()
def task1(id):
   m.foo(id)

def task2(id):
   m.ker(id)

t1 = Thread(target=task1,args=(1,))
t2 = Thread(target=task2,args=(2,))
t1.start()
t2.start()
t1.join()
t2.join()
```

output:

```
$ python monitor.py
1 in foo
1 in ker
2 in ker
```

7.10 Control primitive resources

Using Semaphore

```
from threading import Thread
from threading import Semaphore
from random    import random
import time

# limit resource to 3
sema = Semaphore(3)
def foo(tid):
    with sema:
        print "%d acquire sema" % tid
```

```
        wt = random()*5
        time.sleep(wt)
    print "%d release sema" % tid

threads = []
for _t in range(5):
    t = Thread(target=foo,args=(_t,))
    threads.append(t)
    t.start()
for _t in threads:
    _t.join()
```

output:

```
python semaphore.py
0 acquire sema
1 acquire sema
2 acquire sema
0 release sema
 3 acquire sema
2 release sema
 4 acquire sema
1 release sema
4 release sema
3 release sema
```

7.11 Ensure tasks has done

Using 'event'

```
from threading import Thread
from threading import Event
import time

e = Event()

def worker(id):
   print "%d wait event" % id
   e.wait()
   print "%d get event set" % id

t1=Thread(target=worker,args=(1,))
t2=Thread(target=worker,args=(2,))
t3=Thread(target=worker,args=(3,))
t1.start()
t2.start()
t3.start()

# wait sleep task(event) happen
time.sleep(3)
e.set()
```

output:

```
python event.py
1 wait event
2 wait event
3 wait event
2 get event set
 3 get event set
1 get event set
```

7.12 Thread-safe priority queue

Using 'condition'

```
import threading
import heapq
import time
import random

class PriorityQueue(object):
    def __init__(self):
        self._q = []
        self._count = 0
        self._cv = threading.Condition()

    def __str__(self):
        return str(self._q)

    def __repr__(self):
        return self._q

    def put(self, item, priority):
        with self._cv:
            heapq.heappush(self._q, (-priority,self._count,item))
            self._count += 1
            self._cv.notify()

    def pop(self):
        with self._cv:
            while len(self._q) == 0:
                print("wait...")
                self._cv.wait()
            ret = heapq.heappop(self._q)[-1]
        return ret

priq = PriorityQueue()
def producer():
    while True:
        print(priq.pop())

def consumer():
    while True:
        time.sleep(3)
        print("consumer put value")
        priority = random.random()
        priq.put(priority,priority*10)
```

```
for _ in range(3):
    priority = random.random()
    priq.put(priority,priority*10)

t1=threading.Thread(target=producer)
t2=threading.Thread(target=consumer)
t1.start();t2.start()
t1.join();t2.join()
```

output:

```
python3 thread_safe.py
0.6657491871045683
0.5278797439991247
0.20990624606296315
wait...
consumer put value
0.09123101305407577
wait...
```

7.13 Multiprocessing

Solving GIL problem via processes

```
>>> from multiprocessing import Pool
>>> def fib(n):
...     if n <= 2:
...         return 1
...     return fib(n-1) + fib(n-2)
...
>>> def profile(func):
...     def wrapper(*args, **kwargs):
...         import time
...         start = time.time()
...         func(*args, **kwargs)
...         end   = time.time()
...         print end - start
...     return wrapper
...
>>> @profile
... def nomultiprocess():
...     map(fib,[35]*5)
...
>>> @profile
... def hasmultiprocess():
...     pool = Pool(5)
...     pool.map(fib,[35]*5)
...
>>> nomultiprocess()
23.8454811573
>>> hasmultiprocess()
13.2433719635
```

7.14 Custom multiprocessing map

```
from multiprocessing import Process, Pipe
from itertools import izip

def spawn(f):
    def fun(pipe,x):
        pipe.send(f(x))
        pipe.close()
    return fun

def parmap(f,X):
    pipe=[Pipe() for x in X]
    proc=[Process(target=spawn(f),
          args=(c,x))
          for x,(p,c) in izip(X,pipe)]
    [p.start() for p in proc]
    [p.join() for p in proc]
    return [p.recv() for (p,c) in pipe]

print parmap(lambda x:x**x,range(1,5))
```

7.15 Graceful way to kill all child processes

```
from __future__ import print_function

import signal
import os
import time

from multiprocessing import Process, Pipe

NUM_PROCESS = 10

def aurora(n):
    while True:
        time.sleep(n)

if __name__ == "__main__":
    procs = [Process(target=aurora, args=(x,))
                for x in range(NUM_PROCESS)]
    try:
        for p in procs:
            p.daemon = True
            p.start()
        [p.join() for p in procs]
    finally:
        for p in procs:
            if not p.is_alive(): continue
            os.kill(p.pid, signal.SIGKILL)
```

7.16 Simple round-robin scheduler

```
>>> def fib(n):
...   if n <= 2:
...     return 1
...   return fib(n-1)+fib(n-2)
...
>>> def gen_fib(n):
...   for _ in range(1,n+1):
...     yield fib(_)
...
>>> t=[gen_fib(5),gen_fib(3)]
>>> from collections import deque
>>> tasks = deque()
>>> tasks.extend(t)
>>> def run(tasks):
...   while tasks:
...     try:
...       task = tasks.popleft()
...       print task.next()
...       tasks.append(task)
...     except StopIteration:
...       print "done"
...
>>> run(tasks)
1
1
1
1
2
2
3
done
5
done
```

7.17 Scheduler with blocking function

```
# ref: PyCon 2015 - David Beazley
import socket
from select import select
from collections import deque

tasks  = deque()
r_wait = {}
s_wait = {}

def fib(n):
    if n <= 2:
        return 1
    return fib(n-1)+fib(n-2)

def run():
    while any([tasks,r_wait,s_wait]):
        while not tasks:
```

```
            # polling
            rr, sr, _ = select(r_wait, s_wait, {})
            for _ in rr:
                tasks.append(r_wait.pop(_))
            for _ in sr:
                tasks.append(s_wait.pop(_))
        try:
            task = tasks.popleft()
            why,what = task.next()
            if why == 'recv':
                r_wait[what] = task
            elif why == 'send':
                s_wait[what] = task
            else:
                raise RuntimeError
        except StopIteration:
            pass

def fib_server():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    sock.bind(('localhost',5566))
    sock.listen(5)
    while True:
        yield 'recv', sock
        c, a = sock.accept()
        tasks.append(fib_handler(c))

def fib_handler(client):
    while True:
        yield 'recv', client
        req  = client.recv(1024)
        if not req:
            break
        resp = fib(int(req))
        yield 'send', client
        client.send(str(resp)+'\n')
    client.close()

tasks.append(fib_server())
run()
```

output: (bash 1)

```
$ nc loalhost 5566
20
6765
```

output: (bash 2)

```
$ nc localhost 5566
10
55
```

7.18 PoolExecutor

```
# python2.x is module futures on PyPI
# new in Python3.2
>>> from concurrent.futures import \
...     ThreadPoolExecutor
>>> def fib(n):
...     if n<=2:
...         return 1
...     return fib(n-1) + fib(n-2)
...
>>> with ThreadPoolExecutor(3) as e:
...     res= e.map(fib,[1,2,3,4,5])
...     for _ in res:
...         print(_, end=' ')
...
1 1 2 3 5 >>>
# result is generator?!
>>> with ThreadPoolExecutor(3) as e:
...   res = e.map(fib, [1,2,3])
...   inspect.isgenerator(res)
...
True

# demo GIL
from concurrent import futures
import time

def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)

def thread():
    s = time.time()
    with futures.ThreadPoolExecutor(2) as e:
        res = e.map(fib, [35]*2)
        for _ in res:
            print(_)
    e = time.time()
    print("thread cost: {}".format(e-s))

def process():
    s = time.time()
    with futures.ProcessPoolExecutor(2) as e:
        res = e.map(fib, [35]*2)
        for _ in res:
            print(_)
    e = time.time()
    print("pocess cost: {}".format(e-s))


# bash> python3 -i test.py
>>> thread()
9227465
9227465
thread cost: 12.550225019454956
>>> process()
```

```
9227465
9227465
pocess cost: 5.538189888000488
```

7.19 What "with ThreadPoolExecutor" doing?

```
from concurrent import futures

def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)

with futures.ThreadPoolExecutor(3) as e:
    fut = e.submit(fib, 30)
    res = fut.result()
    print(res)

# equal to
e = futures.ThreadPoolExecutor(3)
fut = e.submit(fib, 30)
fut.result()
e.shutdown(wait=True)
print(res)
```

output:

```
$ python3 thread_pool_exec.py
832040
832040
```

7.20 Future Object

```
# future: deferred computation
# add_done_callback
from concurrent import futures

def fib(n):
    if n <= 2:
        return 1
    return fib(n-1) + fib(n-2)

def handler(future):
    res = future.result()
    print("res: {}".format(res))

def thread_v1():
    with futures.ThreadPoolExecutor(3) as e:
        for _ in range(3):
            f = e.submit(fib, 30+_)
            f.add_done_callback(handler)
    print("end")
```

```
def thread_v2():
    to_do = []
    with futures.ThreadPoolExecutor(3) as e:
        for _ in range(3):
            fut = e.submit(fib, 30+_)
            to_do.append(fut)
        for _f in futures.as_completed(to_do):
            res = _f.result()
            print("res: {}".format(res))
    print("end")
```

output:

```
$ python3 -i fut.py
>>> thread_v1()
res: 832040
res: 1346269
res: 2178309
end
>>> thread_v2()
res: 832040
res: 1346269
res: 2178309
end
```

7.21 Future error handling

```
from concurrent import futures

def spam():
    raise RuntimeError

def handler(future):
    print("callback handler")
    try:
        res = future.result()
    except RuntimeError:
        print("get RuntimeError")

def thread_spam():
    with futures.ThreadPoolExecutor(2) as e:
        f = e.submit(spam)
        f.add_done_callback(handler)
```

output:

```
$ python -i fut_err.py
>>> thread_spam()
callback handler
get RuntimeError
```

CHAPTER 8

Python SQLAlchemy Cheatsheet

Table of Contents

- *Python SQLAlchemy Cheatsheet*
  - *Set a database URL*
  - *Sqlalchemy Support DBAPI - PEP249*
  - *Transaction and Connect Object*
  - *Metadata - Generating Database Schema*
  - *Inspect - Get Database Information*
  - *Reflection - Loading Table from Existing Database*
  - *Get Table from MetaData*
  - *Create all Tables Store in "MetaData"*
  - *Create Specific Table*
  - *Create table with same columns*
  - *Drop a Table*
  - *Some Table Object Operation*
  - *SQL Expression Language*
  - *insert() - Create an "INSERT" Statement*
  - *select() - Create a "SELECT" Statement*
  - *join() - Joined Two Tables via "JOIN" Statement*
  - *Delete Rows from Table*
  - *Check Table Existing*

- *Create multiple tables at once*
- *Create tables with dynamic columns (Table)*
- *Object Relational add data*
- *Object Relational update data*
- *Object Relational delete row*
- *Object Relational relationship*
- *Object Relational self association*
- *Object Relational basic query*
- *mapper: Map `Table` to `class`*
- *Get table dynamically*
- *Object Relational join two tables*
- *join on relationship and group_by count*
- *Create tables with dynamic columns (ORM)*
- *Close database connection*
- *Cannot use the object after close the session*

8.1 Set a database URL

```
from sqlalchemy.engine.url import URL

postgres_db = {'drivername': 'postgres',
               'username': 'postgres',
               'password': 'postgres',
               'host': '192.168.99.100',
               'port': 5432}
print URL(**postgres_db)

sqlite_db = {'drivername': 'sqlite', 'database': 'db.sqlite'}
print URL(**sqlite_db)
```

output:

```
$ python sqlalchemy_url.py
postgres://postgres:postgres@192.168.99.100:5432
sqlite:///db.sqlite
```

8.2 Sqlalchemy Support DBAPI - PEP249

```
from sqlalchemy import create_engine

db_uri = "sqlite:///db.sqlite"
engine = create_engine(db_uri)
```

```
# DBAPI - PEP249
# create table
engine.execute('CREATE TABLE "EX1" ('
               'id INTEGER NOT NULL,'
               'name VARCHAR, '
               'PRIMARY KEY (id));')
# insert a raw
engine.execute('INSERT INTO "EX1" '
               '(id, name) '
               'VALUES (1,"raw1")')

# select *
result = engine.execute('SELECT * FROM '
                        '"EX1"')
for _r in result:
   print _r

# delete *
engine.execute('DELETE from "EX1" where id=1;')
result = engine.execute('SELECT * FROM "EX1"')
print result.fetchall()
```

8.3 Transaction and Connect Object

```
from sqlalchemy import create_engine

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# Create connection
conn = engine.connect()
# Begin transaction
trans = conn.begin()
conn.execute('INSERT INTO "EX1" (name) '
             'VALUES ("Hello")')
trans.commit()
# Close connection
conn.close()
```

8.4 Metadata - Generating Database Schema

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# Create a metadata instance
metadata = MetaData(engine)
```

```
# Declare a table
table = Table('Example',metadata,
              Column('id',Integer, primary_key=True),
              Column('name',String))
# Create all tables
metadata.create_all()
for _t in metadata.tables:
   print "Table: ", _t
```

8.5 Inspect - Get Database Information

```
from sqlalchemy import create_engine
from sqlalchemy import inspect

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

inspector = inspect(engine)

# Get table information
print inspector.get_table_names()

# Get column information
print inspector.get_columns('EX1')
```

8.6 Reflection - Loading Table from Existing Database

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# Create a MetaData instance
metadata = MetaData()
print metadata.tables

# reflect db schema to MetaData
metadata.reflect(bind=engine)
print metadata.tables
```

8.7 Get Table from MetaData

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table

db_uri = 'sqlite:///db.sqlite'
```

```
engine = create_engine(db_uri)

# Create MetaData instance
metadata = MetaData(engine, reflect=True)
print metadata.tables

# Get Table
ex_table = metadata.tables['Example']
print ex_table
```

8.8 Create all Tables Store in "MetaData"

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
meta = MetaData(engine)

# Register t1, t2 to metadata
t1 = Table('EX1', meta,
           Column('id',Integer, primary_key=True),
           Column('name',String))

t2 = Table('EX2', meta,
           Column('id',Integer, primary_key=True),
           Column('val',Integer))
# Create all tables in meta
meta.create_all()
```

8.9 Create Specific Table

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

meta = MetaData(engine)
t1 = Table('Table_1', meta,
           Column('id', Integer, primary_key=True),
           Column('name',String))
t2 = Table('Table_2', meta,
           Column('id', Integer, primary_key=True),
           Column('val',Integer))
t1.create()
```

8.10 Create table with same columns

```
from sqlalchemy import (
    create_engine,
    inspect,
    Column,
    String,
    Integer)

from sqlalchemy.ext.declarative import declarative_base

db_url = "sqlite://"
engine = create_engine(db_url)

Base = declarative_base()

class TemplateTable(object):
    id   = Column(Integer, primary_key=True)
    name = Column(String)
    age  = Column(Integer)

class DowntownAPeople(TemplateTable, Base):
    __tablename__ = "downtown_a_people"

class DowntownBPeople(TemplateTable, Base):
    __tablename__ = "downtown_b_people"

Base.metadata.create_all(bind=engine)

# check table exists
ins = inspect(engine)
for _t in ins.get_table_names(): print _t
```

8.11 Drop a Table

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import inspect
from sqlalchemy import Table
from sqlalchemy import Column, Integer, String
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))
m = MetaData()
table = Table('Test', m,
              Column('id', Integer, primary_key=True),
              Column('key', String, nullable=True),
              Column('val', String))

table.create(engine)
```

```
inspector = inspect(engine)
print 'Test' in inspector.get_table_names()

table.drop(engine)
inspector = inspect(engine)
print 'Test' in inspector.get_table_names()
```

output:

```
$ python sqlalchemy_drop.py
$ True
$ False
```

8.12 Some Table Object Operation

```
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

meta = MetaData()
t = Table('ex_table', meta,
          Column('id', Integer, primary_key=True),
          Column('key', String),
          Column('val', Integer))
# Get Table Name
print t.name

# Get Columns
print t.columns.keys()

# Get Column
c = t.c.key
print c.name
# Or
c = t.columns.key
print c.name

# Get Table from Column
print c.table
```

8.13 SQL Expression Language

```
# Think Column as "ColumnElement"
# Implement via overwrite special function
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String
from sqlalchemy import or_

meta = MetaData()
```

```
table = Table('example', meta,
              Column('id', Integer, primary_key=True),
              Column('l_name', String),
              Column('f_name', String))
# sql expression binary object
print repr(table.c.l_name == 'ed')
# exhbit sql expression
print str(table.c.l_name == 'ed')

print repr(table.c.f_name != 'ed')

# comparison operator
print repr(table.c.id > 3)

# or expression
print (table.c.id > 5) | (table.c.id < 2)
# Equal to
print or_(table.c.id > 5, table.c.id < 2)

# compare to None produce IS NULL
print (table.c.l_name == None)
# Equal to
print (table.c.l_name.is_(None))

# + means "addition"
print (table.c.id + 5)
# or means "string concatenation"
print (table.c.l_name + "some name")

# in expression
print (table.c.l_name.in_(['a','b']))
```

8.14 insert() - Create an “INSERT” Statement

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# create table
meta = MetaData(engine)
table = Table('user', meta,
   Column('id', Integer, primary_key=True),
   Column('l_name', String),
   Column('f_name', String))
meta.create_all()

# insert data via insert() construct
ins = table.insert().values(
      l_name='Hello',
```

```
      f_name='World')
conn = engine.connect()
conn.execute(ins)

# insert multiple data
conn.execute(table.insert(),[
   {'l_name':'Hi','f_name':'bob'},
   {'l_name':'yo','f_name':'alice'}])
```

8.15 select() - Create a "SELECT" Statement

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import select
from sqlalchemy import or_

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
conn = engine.connect()

meta = MetaData(engine,reflect=True)
table = meta.tables['user']

# select * from 'user'
select_st = select([table]).where(
   table.c.l_name == 'Hello')
res = conn.execute(select_st)
for _row in res: print _row

# or equal to
select_st = table.select().where(
   table.c.l_name == 'Hello')
res = conn.execute(select_st)
for _row in res: print _row

# combine with "OR"
select_st = select([
   table.c.l_name,
   table.c.f_name]).where(or_(
      table.c.l_name == 'Hello',
      table.c.l_name == 'Hi'))
res = conn.execute(select_st)
for _row in res: print _row

# combine with "ORDER_BY"
select_st = select([table]).where(or_(
      table.c.l_name == 'Hello',
      table.c.l_name == 'Hi')).order_by(table.c.f_name)
res = conn.execute(select_st)
for _row in res: print _row
```

8.16 join() - Joined Two Tables via "JOIN" Statement

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String
from sqlalchemy import select

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

meta = MetaData(engine, reflect=True)
email_t = Table('email_addr', meta,
      Column('id', Integer, primary_key=True),
      Column('email',String),
      Column('name',String))
meta.create_all()

# get user table
user_t = meta.tables['user']

# insert
conn = engine.connect()
conn.execute(email_t.insert(),[
   {'email':'ker@test','name':'Hi'},
   {'email':'yo@test','name':'Hello'}])
# join statement
join_obj = user_t.join(email_t,
           email_t.c.name == user_t.c.l_name)
# using select_from
sel_st = select(
   [user_t.c.l_name, email_t.c.email]).select_from(join_obj)
res = conn.execute(sel_st)
for _row in res: print _row
```

8.17 Delete Rows from Table

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
conn = engine.connect()

meta = MetaData(engine, reflect=True)
user_t = meta.tables['user']

# select * from user_t
sel_st = user_t.select()
res = conn.execute(sel_st)
for _row in res: print _row

# delete l_name == 'Hello'
```

```
del_st = user_t.delete().where(
      user_t.c.l_name == 'Hello')
print '----- delete -----'
res = conn.execute(del_st)

# check rows has been delete
sel_st = user_t.select()
res = conn.execute(sel_st)
for _row in res: print _row
```

8.18 Check Table Existing

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Column
from sqlalchemy import Integer, String
from sqlalchemy import inspect
from sqlalchemy.ext.declarative import declarative_base

Modal = declarative_base()
class Example(Modal):
   __tablename__ = "ex_t"
   id = Column(Integer, primary_key=True)
   name = Column(String(20))

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
Modal.metadata.create_all(engine)

# check register table exist to Modal
for _t in Modal.metadata.tables: print _t

# check all table in database
meta = MetaData(engine, reflect=True)
for _t in meta.tables: print _t

# check table names exists via inspect
ins = inspect(engine)
for _t in ins.get_table_names(): print _t
```

8.19 Create multiple tables at once

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import inspect
from sqlalchemy import Column, String, Integer
from sqlalchemy.engine.url import URL

db = {'drivername': 'postgres',
      'username': 'postgres',
      'password': 'postgres',
```

```
      'host': '192.168.99.100',
      'port': 5432}

url = URL(**db)
engine = create_engine(url)

metadata = MetaData()
metadata.reflect(bind=engine)

def create_table(name, metadata):
    tables = metadata.tables.keys()
    if name not in tables:
        table = Table(name, metadata,
                      Column('id', Integer, primary_key=True),
                      Column('key', String),
                      Column('val', Integer))
        table.create(engine)

tables = ['table1', 'table2', 'table3']
for _t in tables: create_table(_t, metadata)

inspector = inspect(engine)
print inspector.get_table_names()
```

output:

```
$ python sqlalchemy_create.py
[u'table1', u'table2', u'table3']
```

8.20 Create tables with dynamic columns (Table)

```
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String
from sqlalchemy import Table
from sqlalchemy import MetaData
from sqlalchemy import inspect
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

engine = create_engine(URL(**db_url))

def create_table(name, *cols):
    meta = MetaData()
    meta.reflect(bind=engine)
    if name in meta.tables: return

    table = Table(name, meta, *cols)
    table.create(engine)

create_table('Table1',
```

```
            Column('id', Integer, primary_key=True),
            Column('name', String))
create_table('Table2',
            Column('id', Integer, primary_key=True),
            Column('key', String),
            Column('val', String))

inspector = inspect(engine)
for _t in inspector.get_table_names(): print _t
```

output:

```
$ python sqlalchemy_dynamic.py
Table1
Table2
```

8.21 Object Relational add data

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))

Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

data = {'a': 5566, 'b': 9527, 'c': 183}
try:
    for _key, _val in data.items():
        row = TestTable(key=_key, val=_val)
```

```
        session.add(row)
    session.commit()
except SQLAlchemyError as e:
    print e
finally:
    session.close()
```

8.22 Object Relational update data

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))
Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

try:
    # add row to database
    row = TestTable(key="hello", val="world")
    session.add(row)
    session.commit()

    # update row to database
    row = session.query(TestTable).filter(
          TestTable.key == 'hello').first()
    print 'original:', row.key, row.val
    row.key = "Hello"
    row.val = "World"
    session.commit()
```

```
    # check update correct
    row = session.query(TestTable).filter(
          TestTable.key == 'Hello').first()
    print 'update:', row.key, row.val
except SQLAlchemyError as e:
    print e
finally:
    session.close()
```

output:

```
$ python sqlalchemy_update.py
original: hello world
update: Hello World
```

8.23 Object Relational delete row

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL


db_url = {'drivername': 'postgres',
        'username': 'postgres',
        'password': 'postgres',
        'host': '192.168.99.100',
        'port': 5432}
engine = create_engine(URL(**db_url))
Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

row = TestTable(key='hello', val='world')
session.add(row)
query = session.query(TestTable).filter(
        TestTable.key=='hello')
print query.first()
```

```
query.delete()
query = session.query(TestTable).filter(
        TestTable.key=='hello')
print query.all()
```

output:

```
$ python sqlalchemy_delete.py
<__main__.TestTable object at 0x104eb8f50>
[]
```

8.24 Object Relational relationship

```
from sqlalchemy import Column, String, Integer, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    addresses = relationship("Address", backref="user")

class Address(Base):
    __tablename__ = 'address'
    id = Column(Integer, primary_key=True)
    email = Column(String)
    user_id = Column(Integer, ForeignKey('user.id'))

u1 = User()
a1 = Address()
print u1.addresses
print a1.user

u1.addresses.append(a1)
print u1.addresses
print a1.user
```

output:

```
$ python sqlalchemy_relationship.py
[]
None
[<__main__.Address object at 0x10c4edb50>]
<__main__.User object at 0x10c4ed810>
```

8.25 Object Relational self association

```
import json
```

```
from sqlalchemy import (
    Column,
    Integer,
    String,
    ForeignKey,
    Table)

from sqlalchemy.orm import (
    sessionmaker,
    relationship)

from sqlalchemy.ext.declarative import declarative_base

base = declarative_base()

association = Table("Association", base.metadata,
    Column('left', Integer, ForeignKey('node.id'), primary_key=True),
    Column('right', Integer, ForeignKey('node.id'), primary_key=True))

class Node(base):
    __tablename__ = 'node'
    id = Column(Integer, primary_key=True)
    label = Column(String)
    friends = relationship('Node',
                           secondary=association,
                           primaryjoin=id==association.c.left,
                           secondaryjoin=id==association.c.right,
                           backref='left')
    def to_json(self):
        return dict(id=self.id,
                    friends=[_.label for _ in self.friends])

nodes = [Node(label='node_{}'.format(_)) for _ in range(0, 3)]
nodes[0].friends.extend([nodes[1], nodes[2]])
nodes[1].friends.append(nodes[2])

print('----> right')
print(json.dumps([_.to_json() for _ in nodes], indent=2))

print('----> left')
print(json.dumps([_n.to_json() for _n in nodes[1].left], indent=2))
```

output:

```
----> right
[
  {
    "friends": [
      "node_1",
      "node_2"
    ],
    "id": null
  },
  {
    "friends": [
      "node_2"
    ],
    "id": null
```

```
  },
  {
    "friends": [],
    "id": null
  }
]
----> left
[
  {
    "friends": [
      "node_1",
      "node_2"
    ],
    "id": null
  }
]
```

8.26 Object Relational basic query

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, String, Integer, DateTime
from sqlalchemy import or_
from sqlalchemy import desc
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

Base = declarative_base()

class User(Base):
    __tablename__ = 'User'
    id       = Column(Integer, primary_key=True)
    name     = Column(String, nullable=False)
    fullname = Column(String, nullable=False)
    birth    = Column(DateTime)

# create tables
engine = create_engine(URL(**db_url))
Base.metadata.create_all(bind=engine)

users = [
    User(name='ed',
         fullname='Ed Jones',
         birth=datetime(1989,7,1)),
    User(name='wendy',
         fullname='Wendy Williams',
```

```
        birth=datetime(1983,4,1)),
    User(name='mary',
        fullname='Mary Contrary',
        birth=datetime(1990,1,30)),
    User(name='fred',
        fullname='Fred Flinstone',
        birth=datetime(1977,3,12)),
    User(name='justin',
        fullname="Justin Bieber")]

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

# add_all
session.add_all(users)
session.commit()

print "----> order_by(id):"
query = session.query(User).order_by(User.id)
for _row in query.all():
    print _row.name, _row.fullname, _row.birth

print "\n----> order_by(desc(id)):"
query = session.query(User).order_by(desc(User.id))
for _row in query.all():
    print _row.name, _row.fullname, _row.birth

print "\n----> order_by(date):"
query = session.query(User).order_by(User.birth)
for _row in query.all():
    print _row.name, _row.fullname, _row.birth

print "\n----> EQUAL:"
query = session.query(User).filter(User.id == 2)
_row = query.first()
print _row.name, _row.fullname, _row.birth

print "\n----> NOT EQUAL:"
query = session.query(User).filter(User.id != 2)
for _row in query.all():
    print _row.name, _row.fullname, _row.birth

print "\n----> IN:"
query = session.query(User).filter(User.name.in_(['ed', 'wendy']))
for _row in query.all():
    print _row.name, _row.fullname, _row.birth

print "\n----> NOT IN:"
query = session.query(User).filter(~User.name.in_(['ed', 'wendy']))
for _row in query.all():
    print _row.name, _row.fullname, _row.birth

print "\n----> AND:"
query = session.query(User).filter(
        User.name=='ed', User.fullname=='Ed Jones')
_row = query.first()
```

```
print _row.name, _row.fullname, _row.birth

print "\n----> OR:"
query = session.query(User).filter(
        or_(User.name=='ed', User.name=='wendy'))
for _row in query.all():
    print _row.name, _row.fullname, _row.birth

print "\n----> NULL:"
query = session.query(User).filter(User.birth == None)
for _row in query.all():
    print _row.name, _row.fullname

print "\n----> NOT NULL:"
query = session.query(User).filter(User.birth != None)
for _row in query.all():
    print _row.name, _row.fullname

print "\n----> LIKE"
query = session.query(User).filter(User.name.like('%ed%'))
for _row in query.all():
    print _row.name, _row.fullname
```

output:

```
----> order_by(id):
ed Ed Jones 1989-07-01 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
fred Fred Flinstone 1977-03-12 00:00:00
justin Justin Bieber None

----> order_by(desc(id)):
justin Justin Bieber None
fred Fred Flinstone 1977-03-12 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00
ed Ed Jones 1989-07-01 00:00:00

----> order_by(date):
fred Fred Flinstone 1977-03-12 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00
ed Ed Jones 1989-07-01 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
justin Justin Bieber None

----> EQUAL:
wendy Wendy Williams 1983-04-01 00:00:00

----> NOT EQUAL:
ed Ed Jones 1989-07-01 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
fred Fred Flinstone 1977-03-12 00:00:00
justin Justin Bieber None

----> IN:
ed Ed Jones 1989-07-01 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00
```

```
----> NOT IN:
mary Mary Contrary 1990-01-30 00:00:00
fred Fred Flinstone 1977-03-12 00:00:00
justin Justin Bieber None

----> AND:
ed Ed Jones 1989-07-01 00:00:00

----> OR:
ed Ed Jones 1989-07-01 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00

----> NULL:
justin Justin Bieber

----> NOT NULL:
ed Ed Jones
wendy Wendy Williams
mary Mary Contrary
fred Fred Flinstone

----> LIKE
ed Ed Jones
fred Fred Flinstone
```

8.27 mapper: Map `Table` to `class`

```
from sqlalchemy import (
    create_engine,
    Table,
    MetaData,
    Column,
    Integer,
    String,
    ForeignKey)

from sqlalchemy.orm import (
    mapper,
    relationship,
    sessionmaker)

# classical mapping: map "table" to "class"
db_url = 'sqlite://'
engine = create_engine(db_url)

meta = MetaData(bind=engine)

user = Table('User', meta,
             Column('id', Integer, primary_key=True),
             Column('name', String),
             Column('fullname', String),
             Column('password', String))

addr = Table('Address', meta,
```

```
            Column('id', Integer, primary_key=True),
            Column('email', String),
            Column('user_id', Integer, ForeignKey('User.id')))

# map table to class
class User(object):
    def __init__(self, name, fullname, password):
        self.name = name
        self.fullname = fullname
        self.password = password

class Address(object):
    def __init__(self, email):
        self.email = email

mapper(User, user, properties={
       'addresses': relationship(Address, backref='user')})
mapper(Address, addr)

# create table
meta.create_all()

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

u = User(name='Hello', fullname='HelloWorld', password='ker')
a = Address(email='hello@hello.com')
u.addresses.append(a)
try:
    session.add(u)
    session.commit()

    # query result
    u = session.query(User).filter(User.name == 'Hello').first()
    print u.name, u.fullname, u.password

finally:
    session.close()
```

output:

```
$ python map_table_class.py
Hello HelloWorld ker
```

8.28 Get table dynamically

```
from sqlalchemy import (
    create_engine,
    MetaData,
    Table,
    inspect,
    Column,
    String,
```

```
    Integer)

from sqlalchemy.orm import (
    mapper,
    scoped_session,
    sessionmaker)

db_url = "sqlite://"
engine = create_engine(db_url)
metadata = MetaData(engine)

class TableTemp(object):
    def __init__(self, name):
        self.name = name

def get_table(name):
    if name in metadata.tables:
        table = metadata.tables[name]
    else:
        table = Table(name, metadata,
                Column('id', Integer, primary_key=True),
                Column('name', String))
        table.create(engine)

    cls = type(name.title(), (TableTemp,), {})
    mapper(cls, table)
    return cls

# get table first times
t = get_table('Hello')

# get table secone times
t = get_table('Hello')

Session = scoped_session(sessionmaker(bind=engine))
try:
    Session.add(t(name='foo'))
    Session.add(t(name='bar'))
    for _ in Session.query(t).all(): print _.name
except Exception as e:
    Session.rollback()
finally:
    Session.close()
```

output:

```
$ python get_table.py
foo
bar
```

8.29 Object Relational join two tables

```
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
```

```
from sqlalchemy.engine.url import URL
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class User(Base):
    __tablename__ = 'user'
    id    = Column(Integer, primary_key=True)
    name  = Column(String)
    addresses = relationship("Address", backref="user")

class Address(Base):
    __tablename__ = 'address'
    id = Column(Integer, primary_key=True)
    email = Column(String)
    user_id = Column(Integer, ForeignKey('user.id'))

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

# create engine
engine = create_engine(URL(**db_url))

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

user = User(name='user1')
mail1 = Address(email='user1@foo.com')
mail2 = Address(email='user1@bar.com')
user.addresses.extend([mail1, mail2])

session.add(user)
session.add_all([mail1, mail2])
session.commit()

query = session.query(Address, User).join(User)
for _a, _u in query.all(): print _u.name, _a.email
```

output:

```
$ python sqlalchemy_join.py
user1 user1@foo.com
user1 user1@bar.com
```

8.30 join on relationship and group_by count

```
from sqlalchemy import (
    create_engine,
    Column,
    String,
    Integer,
    ForeignKey,
    func)

from sqlalchemy.orm import (
    relationship,
    sessionmaker,
    scoped_session)

from sqlalchemy.ext.declarative import declarative_base

db_url = 'sqlite://'
engine = create_engine(db_url)

Base = declarative_base()

class Parent(Base):
    __tablename__ = 'parent'
    id       = Column(Integer, primary_key=True)
    name     = Column(String)
    children = relationship('Child', back_populates='parent')

class Child(Base):
    __tablename__ = 'child'
    id        = Column(Integer, primary_key=True)
    name      = Column(String)
    parent_id = Column(Integer, ForeignKey('parent.id'))
    parent    = relationship('Parent', back_populates='children')

Base.metadata.create_all(bind=engine)
Session = scoped_session(sessionmaker(bind=engine))

p1 = Parent(name="Alice")
p2 = Parent(name="Bob")

c1 = Child(name="foo")
c2 = Child(name="bar")
c3 = Child(name="ker")
c4 = Child(name="cat")

p1.children.extend([c1, c2, c3])
p2.children.append(c4)

try:
    Session.add(p1)
    Session.add(p2)
    Session.commit()

    # count number of children
    q = Session.query(Parent, func.count(Child.id))\
                .join(Child)\
                .group_by(Parent.id)
```

```
    # print result
    for _p, _c in q.all():
        print 'parent: {}, num_child: {}'.format(_p.name, _c)
finally:
    Session.remove()
```

output:

```
$ python join_group_by.py
parent: Alice, num_child: 3
parent: Bob, num_child: 1
```

8.31 Create tables with dynamic columns (ORM)

```
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String
from sqlalchemy import inspect
from sqlalchemy.engine.url import URL
from sqlalchemy.ext.declarative import declarative_base

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

engine = create_engine(URL(**db_url))
Base = declarative_base()

def create_table(name, cols):
    Base.metadata.reflect(engine)
    if name in Base.metadata.tables: return

    table = type(name, (Base,), cols)
    table.__table__.create(bind=engine)

create_table('Table1', {
             '__tablename__': 'Table1',
             'id': Column(Integer, primary_key=True),
             'name': Column(String)})

create_table('Table2', {
             '__tablename__': 'Table2',
             'id': Column(Integer, primary_key=True),
             'key': Column(String),
             'val': Column(String)})

inspector = inspect(engine)
for _t in inspector.get_table_names(): print _t
```

output:

```
$ python sqlalchemy_dynamic_orm.py
Table1
```

```
Table2
```

8.32 Close database connection

```
from sqlalchemy import (
    create_engine,
    event,
    Column,
    Integer)

from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite://')
base = declarative_base()

@event.listens_for(engine, 'engine_disposed')
def receive_engine_disposed(engine):
    print("engine dispose")

class Table(base):
    __tablename__ = 'example table'
    id = Column(Integer, primary_key=True)

base.metadata.create_all(bind=engine)
session = sessionmaker(bind=engine)()

try:
    try:
        row = Table()
        session.add(row)
    except Exception as e:
        session.rollback()
        raise
    finally:
        session.close()
finally:
    engine.dispose()
```

output:

```
$ python db_dispose.py
engine dispose
```

Warning: Be careful. Close *session* does not mean close database connection. SQLAlchemy *session* generally represents the *transactions*, not connections.

8.33 Cannot use the object after close the session

```
from __future__ import print_function

from sqlalchemy import (
    create_engine,
    Column,
    String,
    Integer)

from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base


url = 'sqlite://'
engine = create_engine(url)
base = declarative_base()

class Table(base):
    __tablename__ = 'table'
    id  = Column(Integer, primary_key=True)
    key = Column(String)
    val = Column(String)

base.metadata.create_all(bind=engine)
session = sessionmaker(bind=engine)()

try:
    t = Table(key="key", val="val")
    try:
        print(t.key, t.val)
        session.add(t)
        session.commit()
    except Exception as e:
        print(e)
        session.rollback()
    finally:
        session.close()

    print(t.key, t.val) # exception raise from here
except Exception as e:
    print("Cannot use the object after close the session")
finally:
    engine.dispose()
```

output:

```
$ python sql.py
key val
Cannot use the object after close the session
```

CHAPTER 9

Python asyncio cheatsheet

Table of Contents

- *Python asyncio cheatsheet*
  - *What is @asyncio.coroutine?*
  - *What is a Task?*
  - *What event loop doing? (Without polling)*
  - *What `asyncio.wait` doing?*
  - *Future like object*
  - *Future like object `__await__` other task*
  - *Patch loop runner `_run_once`*
  - *Put blocking task into Executor*
  - *Socket with asyncio*
  - *Event Loop with polling*
  - *Transport and Protocol*
  - *Transport and Protocol with SSL*
  - *What `loop.create_server` do?*
  - *Inline callback*
  - *Asynchronous Iterator*
  - *What is asynchronous iterator*
  - *Asynchronous context manager*
  - *What is asynchronous context manager*

- *What* loop.sock_* *do?*
- *Simple asyncio connection pool*
- *Simple asyncio UDP echo server*
- *Simple asyncio web server*
- *Simple HTTPS asyncio web server*
- *Simple asyncio WSGI web server*

9.1 What is @asyncio.coroutine?

```
import asyncio
import inspect
from functools import wraps

Future = asyncio.futures.Future
def coroutine(func):
    """Simple prototype of coroutine"""
    @wraps(func)
    def coro(*a, **k):
        res = func(*a, **k)
        if isinstance(res, Future) or inspect.isgenerator(res):
            res = yield from res
        return res
    return coro

@coroutine
def foo():
    yield from asyncio.sleep(1)
    print("Hello Foo")

@asyncio.coroutine
def bar():
    print("Hello Bar")

loop = asyncio.get_event_loop()
tasks = [loop.create_task(foo()),
         loop.create_task(bar())]
loop.run_until_complete(
     asyncio.wait(tasks))
loop.close()
```

output:

```
$ python test.py
Hello Bar
Hello Foo
```

9.2 What is a Task?

```
# goal: supervise coroutine run state
# ref: asyncio/tasks.py

import asyncio
Future = asyncio.futures.Future

class Task(Future):
    """Simple prototype of Task"""

    def __init__(self, gen, *,loop):
        super().__init__(loop=loop)
        self._gen = gen
        self._loop.call_soon(self._step)

    def _step(self, val=None, exc=None):
        try:
            if exc:
                f = self._gen.throw(exc)
            else:
                f = self._gen.send(val)
        except StopIteration as e:
            self.set_result(e.value)
        except Exception as e:
            self.set_exception(e)
        else:
            f.add_done_callback(
                 self._wakeup)

    def _wakeup(self, fut):
        try:
            res = fut.result()
        except Exception as e:
            self._step(None, e)
        else:
            self._step(res, None)

@asyncio.coroutine
def foo():
    yield from asyncio.sleep(3)
    print("Hello Foo")

@asyncio.coroutine
def bar():
    yield from asyncio.sleep(1)
    print("Hello Bar")

loop = asyncio.get_event_loop()
tasks = [Task(foo(), loop=loop),
         loop.create_task(bar())]
loop.run_until_complete(
        asyncio.wait(tasks))
loop.close()
```

output:

```
$ python test.py
Hello Bar
hello Foo
```

9.3 What event loop doing? (Without polling)

```
import asyncio
from collections import deque

def done_callback(fut):
    fut._loop.stop()

class Loop:
    """Simple event loop prototype"""

    def __init__(self):
        self._ready = deque()
        self._stopping = False

    def create_task(self, coro):
        Task = asyncio.tasks.Task
        task = Task(coro, loop=self)
        return task

    def run_until_complete(self, fut):
        tasks = asyncio.tasks
        # get task
        fut = tasks.ensure_future(
                    fut, loop=self)
        # add task to ready queue
        fut.add_done_callback(done_callback)
        # run tasks
        self.run_forever()
        # remove task from ready queue
        fut.remove_done_callback(done_callback)

    def run_forever(self):
        """Run tasks until stop"""
        try:
            while True:
                self._run_once()
                if self._stopping:
                    break
        finally:
            self._stopping = False

    def call_soon(self, cb, *args):
        """Append task to ready queue"""
        self._ready.append((cb, args))
    def call_exception_handler(self, c):
        pass

    def _run_once(self):
        """Run task at once"""
        ntodo = len(self._ready)
```

```
        for i in range(ntodo):
            t, a = self._ready.popleft()
            t(*a)

    def stop(self):
        self._stopping = True

    def close(self):
        self._ready.clear()

    def get_debug(self):
        return False

@asyncio.coroutine
def foo():
    print("Foo")

@asyncio.coroutine
def bar():
    print("Bar")

loop = Loop()
tasks = [loop.create_task(foo()),
         loop.create_task(bar())]
loop.run_until_complete(
        asyncio.wait(tasks))
loop.close()
```

output:

```
$ python test.py
Foo
Bar
```

9.4 What `asyncio.wait` doing?

```
import asyncio

async def wait(fs, loop=None):
    fs = {asyncio.ensure_future(_) for _ in set(fs)}
    if loop is None:
        loop = asyncio.get_event_loop()

    waiter = loop.create_future()
    counter = len(fs)

    def _on_complete(f):
        nonlocal counter
        counter -= 1
        if counter <= 0 and not waiter.done():
             waiter.set_result(None)

    for f in fs:
        f.add_done_callback(_on_complete)
```

```
    # wait all tasks done
    await waiter

    done, pending = set(), set()
    for f in fs:
        f.remove_done_callback(_on_complete)
        if f.done():
            done.add(f)
        else:
            pending.add(f)
    return done, pending

async def slow_task(n):
    await asyncio.sleep(n)
    print('sleep "{}" sec'.format(n))

loop = asyncio.get_event_loop()

try:
    print("---> wait")
    loop.run_until_complete(
            wait([slow_task(_) for _ in range(1,3)]))
    print("---> asyncio.wait")
    loop.run_until_complete(
            asyncio.wait([slow_task(_) for _ in range(1,3)]))
finally:
    loop.close()
```

output:

```
---> wait
sleep "1" sec
sleep "2" sec
---> asyncio.wait
sleep "1" sec
sleep "2" sec
```

9.5 Future like object

```
>>> import sys
>>> PY_35 = sys.version_info >= (3, 5)
>>> import asyncio
>>> loop = asyncio.get_event_loop()
>>> class SlowObj:
...     def __init__(self, n):
...         print("__init__")
...         self._n = n
...     if PY_35:
...         def __await__(self):
...             print("__await__ sleep({})".format(self._n))
...             yield from asyncio.sleep(self._n)
...             print("ok")
...             return self
...
>>> async def main():
```

```
...     obj = await SlowObj(3)
...
>>> loop.run_until_complete(main())
__init__
__await__ sleep(3)
ok
```

9.6 Future like object `__await__` other task

```
>>> import sys
>>> PY_35 = sys.version_info >= (3, 5)
>>> import asyncio
>>> loop = asyncio.get_event_loop()
>>> async def slow_task(n):
...     await asyncio.sleep(n)
...
>>> class SlowObj:
...     def __init__(self, n):
...         print("__init__")
...         self._n = n
...     if PY_35:
...         def __await__(self):
...             print("__await__")
...             yield from slow_task(self._n).__await__()
...             yield from asyncio.sleep(self._n)
...             print("ok")
...             return self
...
>>> async def main():
...     obj = await SlowObj(1)
...
>>> loop.run_until_complete(main())
__init__
__await__
ok
```

9.7 Patch loop runner `_run_once`

```
>>> import asyncio
>>> def _run_once(self):
...     num_tasks = len(self._scheduled)
...     print("num tasks in queue: {}".format(num_tasks))
...     super(asyncio.SelectorEventLoop, self)._run_once()
...
>>> EventLoop = asyncio.SelectorEventLoop
>>> EventLoop._run_once = _run_once
>>> loop = EventLoop()
>>> asyncio.set_event_loop(loop)
>>> async def task(n):
...     await asyncio.sleep(n)
...     print("sleep: {} sec".format(n))
...
```

```
>>> coro = loop.create_task(task(3))
>>> loop.run_until_complete(coro)
num tasks in queue: 0
num tasks in queue: 1
num tasks in queue: 0
sleep: 3 sec
num tasks in queue: 0
>>> loop.close()
```

9.8 Put blocking task into Executor

```
>>> import asyncio
>>> from concurrent.futures import ThreadPoolExecutor
>>> e = ThreadPoolExecutor()
>>> loop = asyncio.get_event_loop()
>>> async def read_file(file_):
...     with open(file_) as f:
...         data = await loop.run_in_executor(e, f.read)
...         return data

>>> task = loop.create_task(read_file('/etc/passwd'))
>>> ret = loop.run_until_complete(task)
```

9.9 Socket with asyncio

```
import asyncio
import socket

host = 'localhost'
port = 9527
loop = asyncio.get_event_loop()
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.setblocking(False)
s.bind((host, port))
s.listen(10)

async def handler(conn):
    while True:
        msg = await loop.sock_recv(conn, 1024)
        if not msg:
            break
        await loop.sock_sendall(conn, msg)
    conn.close()

async def server():
    while True:
        conn, addr = await loop.sock_accept(s)
        loop.create_task(handler(conn))

loop.create_task(server())
```

```
loop.run_forever()
loop.close()
```

output: (bash 1)

```
$ nc localhost 9527
Hello
Hello
```

output: (bash 2)

```
$ nc localhost 9527
World
World
```

9.10 Event Loop with polling

```
# using selectors
# ref: PyCon 2015 - David Beazley

import asyncio
import socket
import selectors
from collections import deque

@asyncio.coroutine
def read_wait(s):
    yield 'read_wait', s

@asyncio.coroutine
def write_wait(s):
    yield 'write_wait', s

class Loop:
    """Simple loop prototype"""

    def __init__(self):
        self.ready = deque()
        self.selector = selectors.DefaultSelector()

    @asyncio.coroutine
    def sock_accept(self, s):
        yield from read_wait(s)
        return s.accept()

    @asyncio.coroutine
    def sock_recv(self, c, mb):
        yield from read_wait(c)
        return c.recv(mb)

    @asyncio.coroutine
    def sock_sendall(self, c, m):
        while m:
            yield from write_wait(c)
            nsent = c.send(m)
```

```
            m = m[nsent:]

    def create_task(self, coro):
        self.ready.append(coro)

    def run_forever(self):
        while True:
            self._run_once()

    def _run_once(self):
        while not self.ready:
            events = self.selector.select()
            for k, _ in events:
                self.ready.append(k.data)
                self.selector.unregister(k.fileobj)

        while self.ready:
            self.cur_t = ready.popleft()
            try:
                op, *a = self.cur_t.send(None)
                getattr(self, op)(*a)
            except StopIteration:
                pass

    def read_wait(self, s):
        self.selector.register(s, selectors.EVENT_READ, self.cur_t)

    def write_wait(self, s):
        self.selector.register(s, selectors.EVENT_WRITE, self.cur_t)

loop = Loop()
host = 'localhost'
port = 9527

s = socket.socket(
        socket.AF_INET,
        socket.SOCK_STREAM, 0)
s.setsockopt(
        socket.SOL_SOCKET,
        socket.SO_REUSEADDR, 1)
s.setblocking(False)
s.bind((host, port))
s.listen(10)

@asyncio.coroutine
def handler(c):
    while True:
        msg = yield from loop.sock_recv(c, 1024)
        if not msg:
            break
        yield from loop.sock_sendall(c, msg)
    c.close()

@asyncio.coroutine
def server():
    while True:
        c, addr = yield from loop.sock_accept(s)
        loop.create_task(handler(c))
```

```
loop.create_task(server())
loop.run_forever()
```

9.11 Transport and Protocol

```
import asyncio

class EchoProtocol(asyncio.Protocol):

    def connection_made(self, transport):
        peername = transport.get_extra_info('peername')
        print('Connection from {}'.format(peername))
        self.transport = transport

    def data_received(self, data):
        msg = data.decode()
        self.transport.write(data)

loop = asyncio.get_event_loop()
coro = loop.create_server(EchoProtocol, 'localhost', 5566)
server = loop.run_until_complete(coro)

try:
    loop.run_forever()
except:
    loop.run_until_complete(server.wait_closed())
finally:
    loop.close()
```

output:

```
# console 1
$ nc localhost 5566
Hello
Hello

# console 2
$ nc localhost 5566
World
World
```

9.12 Transport and Protocol with SSL

```
import asyncio
import ssl

def make_header():
    head  = b'HTTP/1.1 200 OK\r\n'
    head += b'Content-Type: text/html\r\n'
    head += b'\r\n'
    return head
```

```
def make_body():
    resp  = b"<html>"
    resp += b"<h1>Hello SSL</h1>"
    resp += b"</html>"
    return resp

sslctx = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
sslctx.load_cert_chain(certfile='./root-ca.crt',
                       keyfile='./root-ca.key')

class Service(asyncio.Protocol):

    def connection_made(self, tr):
        self.tr = tr
        self.total = 0

    def data_received(self, data):
        if data:
            resp  = make_header()
            resp += make_body()
            self.tr.write(resp)
        self.tr.close()


async def start():
    server = await loop.create_server(Service,
                                      'localhost',
                                      4433,
                                      ssl=sslctx)
    await server.wait_closed()

try:
    loop = asyncio.get_event_loop()
    loop.run_until_complete(start())
finally:
    loop.close()
```

output:

```
$ openssl genrsa -out root-ca.key 2048
$ openssl req -x509 -new -nodes -key root-ca.key -days 365 -out root-ca.crt
$ python3 ssl_web_server.py

# then open browser: https://localhost:4433
```

9.13 What `loop.create_server` do?

```
import asyncio
import socket

loop = asyncio.get_event_loop()

async def create_server(loop, protocol_factory, host,
                        port, *args, **kwargs):
```

```
    sock = socket.socket(socket.AF_INET,
                         socket.SOCK_STREAM, 0)
    sock.setsockopt(socket.SOL_SOCKET,
                    socket.SO_REUSEADDR, 1)
    sock.setblocking(False)
    sock.bind((host, port))
    sock.listen(10)
    sockets = [sock]
    server = asyncio.base_events.Server(loop, sockets)
    loop._start_serving(protocol_factory, sock, None, server)

    return server


class EchoProtocol(asyncio.Protocol):
    def connection_made(self, transport):
        peername = transport.get_extra_info('peername')
        print('Connection from {}'.format(peername))
        self.transport = transport

    def data_received(self, data):
        message = data.decode()
        self.transport.write(data)

# Equal to: loop.create_server(EchoProtocol,
#                              'localhost', 5566)
coro = create_server(loop, EchoProtocol, 'localhost', 5566)
server = loop.run_until_complete(coro)

try:
    loop.run_forever()
finally:
    server.close()
    loop.run_until_complete(server.wait_closed())
    loop.close()
```

output:

```
# console1
$ nc localhost 5566
Hello
Hello

# console2
$ nc localhost 5566
asyncio
asyncio
```

9.14 Inline callback

```
>>> import asyncio
>>> async def foo():
...     await asyncio.sleep(1)
...     return "foo done"
...
```

```
>>> async def bar():
...     await asyncio.sleep(.5)
...     return "bar done"
...
>>> async def ker():
...     await asyncio.sleep(3)
...     return "ker done"
...
>>> async def task():
...     res = await foo()
...     print(res)
...     res = await bar()
...     print(res)
...     res = await ker()
...     print(res)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(task())
foo done
bar done
ker done
```

9.15 Asynchronous Iterator

```
# ref: PEP-0492
# need Python >= 3.5

>>> class AsyncIter:
...     def __init__(self, it):
...         self._it = iter(it)
...     async def __aiter__(self):
...         return self
...     async def __anext__(self):
...         await asyncio.sleep(1)
...         try:
...             val = next(self._it)
...         except StopIteration:
...             raise StopAsyncIteration
...         return val
...
>>> async def foo():
...     it = [1,2,3]
...     async for _ in AsyncIter(it):
...         print(_)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(foo())
1
2
3
```

9.16 What is asynchronous iterator

```
>>> import asyncio
>>> class AsyncIter:
...     def __init__(self, it):
...         self._it = iter(it)
...     async def __aiter__(self):
...         return self
...     async def __anext__(self):
...         await asyncio.sleep(1)
...         try:
...             val = next(self._it)
...         except StopIteration:
...             raise StopAsyncIteration
...         return val
...
>>> async def foo():
...     _ = [1,2,3]
...     running = True
...     it = AsyncIter(_)
...     while running:
...         try:
...             res = await it.__anext__()
...             print(res)
...         except StopAsyncIteration:
...             running = False
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(loop.create_task(foo()))
1
2
3
```

9.17 Asynchronous context manager

```
# ref: PEP-0492
# need Python >= 3.5

>>> class AsyncCtxMgr:
...     async def __aenter__(self):
...         await asyncio.sleep(3)
...         print("__anter__")
...         return self
...     async def __aexit__(self, *exc):
...         await asyncio.sleep(1)
...         print("__aexit__")
...
>>> async def hello():
...     async with AsyncCtxMgr() as m:
...         print("hello block")
...
>>> async def world():
...     print("world block")
...
>>> t = loop.create_task(world())
```

```
>>> loop.run_until_complete(hello())
world block
__anter__
hello block
__aexit__
```

9.18 What is asynchronous context manager

```
>>> import asyncio
>>> class AsyncManager:
...     async def __aenter__(self):
...         await asyncio.sleep(5)
...         print("__aenter__")
...     async def __aexit__(self, *exc_info):
...         await asyncio.sleep(3)
...         print("__aexit__")
...
>>> async def foo():
...     import sys
...     mgr = AsyncManager()
...     await mgr.__aenter__()
...     print("body")
...     await mgr.__aexit__(*sys.exc_info())
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(loop.create_task(foo()))
__aenter__
body
__aexit__
```

9.19 What *loop.sock_** do?

```
import asyncio
import socket

def sock_accept(self, sock, fut=None, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = self.create_future()
    if registed:
        self.remove_reader(fd)
    try:
        conn, addr = sock.accept()
        conn.setblocking(False)
    except (BlockingIOError, InterruptedError):
        self.add_reader(fd, self.sock_accept, sock, fut, True)
    except Exception as e:
        fut.set_exception(e)
    else:
        fut.set_result((conn, addr))
    return fut
```

```
def sock_recv(self, sock, n , fut=None, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = self.create_future()
    if registed:
        self.remove_reader(fd)
    try:
        data = sock.recv(n)
    except (BlockingIOError, InterruptedError):
        self.add_reader(fd, self.sock_recv, sock, n ,fut, True)
    except Exception as e:
        fut.set_exception(e)
    else:
        fut.set_result(data)
    return fut

def sock_sendall(self, sock, data, fut=None, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = self.create_future()
    if registed:
        self.remove_writer(fd)
    try:
        n = sock.send(data)
    except (BlockingIOError, InterruptedError):
        n = 0
    except Exception as e:
        fut.set_exception(e)
        return
    if n == len(data):
        fut.set_result(None)
    else:
        if n:
            data = data[n:]
        self.add_writer(fd, sock, data, fut, True)
    return fut

async def handler(loop, conn):
    while True:
        msg = await loop.sock_recv(conn, 1024)
        if msg: await loop.sock_sendall(conn, msg)
        else: break
    conn.close()

async def server(loop):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.setblocking(False)
    sock.bind(('localhost', 9527))
    sock.listen(10)

    while True:
        conn, addr = await loop.sock_accept(sock)
        loop.create_task(handler(loop, conn))

EventLoop = asyncio.SelectorEventLoop
EventLoop.sock_accept = sock_accept
EventLoop.sock_recv = sock_recv
```

```
EventLoop.sock_sendall = sock_sendall
loop = EventLoop()

try:
    loop.run_until_complete(server(loop))
except KeyboardInterrupt:
    pass
finally:
    loop.close()
```

output:

```
# console 1
$ python3 async_sock.py &
$ nc localhost 9527
Hello
Hello

# console 2
$ nc localhost 9527
asyncio
asyncio
```

9.20 Simple asyncio connection pool

```
import asyncio
import socket
import uuid

class Transport:

    def __init__(self, loop, host, port):
        self.used = False

        self._loop = loop
        self._host = host
        self._port = port
        self._sock = socket.socket(
                socket.AF_INET, socket.SOCK_STREAM)
        self._sock.setblocking(False)
        self._uuid = uuid.uuid1()

    async def connect(self):
        loop, sock = self._loop, self._sock
        host, port = self._host, self._port
        return (await loop.sock_connect(sock, (host, port)))

    async def sendall(self, msg):
        loop, sock = self._loop, self._sock
        return (await loop.sock_sendall(sock, msg))

    async def recv(self, buf_size):
        loop, sock = self._loop, self._sock
        return (await loop.sock_recv(sock, buf_size))
```

```
    def close(self):
        if self._sock: self._sock.close()

    @property
    def alive(self):
        ret = True if self._sock else False
        return ret

    @property
    def uuid(self):
        return self._uuid


class ConnectionPool:

    def __init__(self, loop, host, port, max_conn=3):
        self._host = host
        self._port = port
        self._max_conn = max_conn
        self._loop = loop

        conns = [Transport(loop, host, port) for _ in range(max_conn)]
        self._conns = conns

    def __await__(self):
        for _c in self._conns:
            yield from _c.connect().__await__()
        return self

    def getconn(self, fut=None):
        if fut is None:
            fut = self._loop.create_future()

        for _c in self._conns:
            if _c.alive and not _c.used:
                _c.used = True
                fut.set_result(_c)
                break
        else:
            loop.call_soon(self.getconn, fut)

        return fut

    def release(self, conn):
        if not conn.used:
            return
        for _c in self._conns:
            if _c.uuid != conn.uuid:
                continue
            _c.used = False
            break

    def close(self):
        for _c in self._conns:
            _c.close()


async def handler(pool, msg):
```

```
    conn = await pool.getconn()
    byte = await conn.sendall(msg)
    mesg = await conn.recv(1024)
    pool.release(conn)
    return 'echo: {}'.format(mesg)


async def main(loop, host, port):
    try:
        # creat connection pool
        pool = await ConnectionPool(loop, host, port)

        # generate messages
        msgs = ['coro_{}'.format(_).encode('utf-8') for _ in range(5)]

        # create tasks
        fs = [loop.create_task(handler(pool, _m)) for _m in msgs]

        # wait all tasks done
        done, pending = await asyncio.wait(fs)
        for _ in done: print(_.result())
    finally:
        pool.close()


loop = asyncio.get_event_loop()
host = '127.0.0.1'
port = 9527

try:
    loop.run_until_complete(main(loop, host, port))
except KeyboardInterrupt:
    pass
finally:
    loop.close()
```

output:

```
$ ncat -l 9527 --keep-open --exec "/bin/cat" &
$ python3 conn_pool.py
echo: b'coro_1'
echo: b'coro_0'
echo: b'coro_2'
echo: b'coro_3'
echo: b'coro_4'
```

9.21 Simple asyncio UDP echo server

```
import asyncio
import socket

loop = asyncio.get_event_loop()

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, 0)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
sock.setblocking(False)

host = 'localhost'
port = 3553

sock.bind((host, port))

def recvfrom(loop, sock, n_bytes, fut=None, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = loop.create_future()
    if registed:
        loop.remove_reader(fd)

    try:
        data, addr = sock.recvfrom(n_bytes)
    except (BlockingIOError, InterruptedError):
        loop.add_reader(fd, recvfrom, loop, sock, n_bytes, fut, True)
    else:
        fut.set_result((data, addr))
    return fut

def sendto(loop, sock, data, addr, fut=None, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = loop.create_future()
    if registed:
        loop.remove_writer(fd)
    if not data:
        return

    try:
        n = sock.sendto(data, addr)
    except (BlockingIOError, InterruptedError):
        loop.add_writer(fd, sendto, loop, sock, data, addr, fut, True)
    else:
        fut.set_result(n)
    return fut

async def udp_server(loop, sock):
    while True:
        data, addr = await recvfrom(loop, sock, 1024)
        n_bytes = await sendto(loop, sock, data, addr)

try:
    loop.run_until_complete(udp_server(loop, sock))
finally:
    loop.close()
```

output:

```
$ python3 udp_server.py
$ nc -u localhost 3553
Hello UDP
Hello UDP
```

9.22 Simple asyncio web server

```
import asyncio
import socket

host = 'localhost'
port = 9527
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.setblocking(False)
s.bind((host, port))
s.listen(10)

loop = asyncio.get_event_loop()

def make_header():
    header  = b"HTTP/1.1 200 OK\r\n"
    header += b"Content-Type: text/html\r\n"
    header += b"\r\n"
    return header

def make_body():
    resp  = b'<html>'
    resp += b'<body><h3>Hello World</h3></body>'
    resp += b'</html>'
    return resp

async def handler(conn):
    req = await loop.sock_recv(conn, 1024)
    if req:
        resp = make_header()
        resp += make_body()
        await loop.sock_sendall(conn, resp)
    conn.close()

async def server(sock, loop):
    while True:
        conn, addr = await loop.sock_accept(sock)
        loop.create_task(handler(conn))

try:
    loop.run_until_complete(server(s, loop))
except KeyboardInterrupt:
    pass
finally:
    loop.close()
    s.close()
# Then open browser with url: localhost:9527
```

9.23 Simple HTTPS asyncio web server

```
import asyncio
import socket
import ssl
```

```
def make_header():
    head  = b'HTTP/1.1 200 OK\r\n'
    head += b'Content-type: text/html\r\n'
    head += b'\r\n'
    return head

def make_body():
    resp  = b'<html>'
    resp += b'<h1>Hello SSL</h1>'
    resp += b'</html>'
    return resp

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.setblocking(False)
sock.bind(('localhost' , 4433))
sock.listen(10)

sslctx = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
sslctx.load_cert_chain(certfile='./root-ca.crt',
                       keyfile='./root-ca.key')


def do_handshake(loop, sock, waiter):
    sock_fd = sock.fileno()
    try:
        sock.do_handshake()
    except ssl.SSLWantReadError:
        loop.remove_reader(sock_fd)
        loop.add_reader(sock_fd, do_handshake,
                        loop, sock, waiter)
        return
    except ssl.SSLWantWriteError:
        loop.remove_writer(sock_fd)
        loop.add_writer(sock_fd, do_handshake,
                        loop, sock, waiter)
        return

    loop.remove_reader(sock_fd)
    loop.remove_writer(sock_fd)
    waiter.set_result(None)


def handle_read(loop, conn, waiter):
    try:
        req = conn.recv(1024)
    except ssl.SSLWantReadError:
        loop.remove_reader(conn.fileno())
        loop.add_reader(conn.fileno(), handle_read,
                        loop, conn, waiter)
        return
    loop.remove_reader(conn.fileno())
    waiter.set_result(req)


def handle_write(loop, conn, msg, waiter):
    try:
        resp = make_header()
```

```
        resp += make_body()
        ret = conn.send(resp)
    except ssl.SSLWantReadError:
        loop.remove_writer(conn.fileno())
        loop.add_writer(conn.fileno(), handle_write,
                        loop, conn, waiter)
        return
    loop.remove_writer(conn.fileno())
    conn.close()
    waiter.set_result(None)


async def server(loop):
    while True:
        conn, addr = await loop.sock_accept(sock)
        conn.setblocking(False)
        sslconn = sslctx.wrap_socket(conn,
                                     server_side=True,
                                     do_handshake_on_connect=False)
        # wait SSL handshake
        waiter = loop.create_future()
        do_handshake(loop, sslconn, waiter)
        await waiter

        # wait read request
        waiter = loop.create_future()
        handle_read(loop, sslconn, waiter)
        msg = await waiter

        # wait write response
        waiter = loop.create_future()
        handle_write(loop, sslconn, msg, waiter)
        await waiter

loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(server(loop))
finally:
    loop.close()
```

output:

```
# console 1

$ openssl genrsa -out root-ca.key 2048
$ openssl req -x509 -new -nodes -key root-ca.key -days 365 -out root-ca.crt
$ python3 Simple_https_server.py

# console 2

$ curl https://localhost:4433 -v          \
>      --resolve localhost:4433:127.0.0.1 \
>      --cacert ~/test/root-ca.crt
```

9.24 Simple asyncio WSGI web server

```
# ref: PEP333

import asyncio
import socket
import io
import sys

from flask import Flask, Response

host = 'localhost'
port = 9527
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.setblocking(False)
s.bind((host, port))
s.listen(10)

loop = asyncio.get_event_loop()

class WSGIServer(object):

    def __init__(self, sock, app):
        self._sock = sock
        self._app = app
        self._header = []

    def parse_request(self, req):
        """ HTTP Request Format:

        GET /hello.htm HTTP/1.1\r\n
        Accept-Language: en-us\r\n
        ...
        Connection: Keep-Alive\r\n
        """
        # bytes to string
        req_info = req.decode('utf-8')
        first_line = req_info.splitlines()[0]
        method, path, ver = first_line.split()
        return method, path, ver

    def get_environ(self, req, method, path):
        env = {}

        # Required WSGI variables
        env['wsgi.version']      = (1, 0)
        env['wsgi.url_scheme']   = 'http'
        env['wsgi.input']        = req
        env['wsgi.errors']       = sys.stderr
        env['wsgi.multithread']  = False
        env['wsgi.multiprocess'] = False
        env['wsgi.run_once']     = False

        # Required CGI variables
        env['REQUEST_METHOD']    = method    # GET
        env['PATH_INFO']         = path      # /hello
        env['SERVER_NAME']       = host      # localhost
```

```
        env['SERVER_PORT']       = str(port) # 9527
        return env

    def start_response(self, status, resp_header, exc_info=None):
        header = [('Server', 'WSGIServer 0.2')]
        self.headers_set = [status, resp_header + header]

    async def finish_response(self, conn, data, headers):
        status, resp_header = headers

        # make header
        resp = 'HTTP/1.1 {0}\r\n'.format(status)
        for header in resp_header:
            resp += '{0}: {1}\r\n'.format(*header)
        resp += '\r\n'

        # make body
        resp += '{0}'.format(data)
        try:
            await loop.sock_sendall(conn, str.encode(resp))
        finally:
            conn.close()

    async def run_server(self):
        while True:
            conn, addr = await loop.sock_accept(self._sock)
            loop.create_task(self.handle_request(conn))

    async def handle_request(self, conn):
        # get request data
        req = await loop.sock_recv(conn, 1024)
        if req:
            method, path, ver = self.parse_request(req)
            # get environment
            env = self.get_environ(req, method, path)
            # get application execute result
            res = self._app(env, self.start_response)
            res = [_.decode('utf-8') for _ in list(res)]
            res = ''.join(res)
            loop.create_task(
                 self.finish_response(conn, res, self.headers_set))

app = Flask(__name__)

@app.route('/hello')
def hello():
    return Response("Hello WSGI",mimetype="text/plain")

server = WSGIServer(s, app.wsgi_app)
try:
    loop.run_until_complete(server.run_server())
except:
    pass
finally:
    loop.close()

# Then open browser with url: localhost:9527/hello
```

CHAPTER 10

Python test cheatsheet

Table of Contents

- *Python test cheatsheet*
  - *A simple Python unittest*
  - *Python unittest setup & teardown hierarchy*
  - *Different module of setUp & tearDown hierarchy*
  - *Run tests via unittest.TextTestRunner*
  - *Test raise exception*
  - *Pass arguments into a TestCase*
  - *Group multiple testcases into a suite*
  - *Group multiple tests from different TestCase*
  - *Skip some tests in the TestCase*
  - *Monolithic Test*
  - *Cross-module variables to Test files*
  - *skip setup & teardown when the test is skipped*
  - *Re-using old test code*
  - *Testing your document is right*
  - *Re-using doctest to unittest*
  - *Customize test report*
  - *Mock - using `@patch` substitute original method*
  - *What `with unittest.mock.patch` do?*

- *Mock - substitute open*

10.1 A simple Python unittest

```
# python unittests only run the function with prefix "test"

>>> from __future__ import print_function
>>> import unittest
>>> class TestFoo(unittest.TestCase):
...     def test_foo(self):
...             self.assertTrue(True)
...     def fun_not_run(self):
...             print("no run")
...
>>> unittest.main()
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
>>> import unittest
>>> class TestFail(unittest.TestCase):
...     def test_false(self):
...             self.assertTrue(False)
...
>>> unittest.main()
F
======================================================================
FAIL: test_false (__main__.TestFail)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<stdin>", line 3, in test_false
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

10.2 Python unittest setup & teardown hierarchy

```
from __future__ import print_function

import unittest

def fib(n):
    return 1 if n<=2 else fib(n-1)+fib(n-2)

def setUpModule():
        print("setup module")
def tearDownModule():
        print("teardown module")
```

```
class TestFib(unittest.TestCase):

    def setUp(self):
        print("setUp")
        self.n = 10
    def tearDown(self):
        print("tearDown")
        del self.n
    @classmethod
    def setUpClass(cls):
        print("setUpClass")
    @classmethod
    def tearDownClass(cls):
        print("tearDownClass")
    def test_fib_assert_equal(self):
        self.assertEqual(fib(self.n), 55)
    def test_fib_assert_true(self):
        self.assertTrue(fib(self.n) == 55)

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

output:

```
$ python test.py
setup module
setUpClass
setUp
tearDown
.setUp
tearDown
.tearDownClass
teardown module

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

10.3 Different module of setUp & tearDown hierarchy

```
# test_module.py
from __future__ import print_function

import unittest

class TestFoo(unittest.TestCase):
    @classmethod
    def setUpClass(self):
        print("foo setUpClass")
    @classmethod
    def tearDownClass(self):
        print("foo tearDownClass")
    def setUp(self):
```

```
        print("foo setUp")
    def tearDown(self):
        print("foo tearDown")
    def test_foo(self):
        self.assertTrue(True)

class TestBar(unittest.TestCase):
    def setUp(self):
        print("bar setUp")
    def tearDown(self):
        print("bar tearDown")
    def test_bar(self):
        self.assertTrue(True)

# test.py
from __future__ import print_function

from test_module import TestFoo
from test_module import TestBar
import test_module
import unittest

def setUpModule():
    print("setUpModule")

def tearDownModule():
    print("tearDownModule")


if __name__ == "__main__":
    test_module.setUpModule = setUpModule
    test_module.tearDownModule = tearDownModule
    suite1 = unittest.TestLoader().loadTestsFromTestCase(TestFoo)
    suite2 = unittest.TestLoader().loadTestsFromTestCase(TestBar)
    suite = unittest.TestSuite([suite1,suite2])
    unittest.TextTestRunner().run(suite)
```

output:

```
$ python test.py
setUpModule
foo setUpClass
foo setUp
foo tearDown
.foo tearDownClass
bar setUp
bar tearDown
.tearDownModule

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

10.4 Run tests via unittest.TextTestRunner

```
>>> import unittest
>>> class TestFoo(unittest.TestCase):
...     def test_foo(self):
...         self.assertTrue(True)
...     def test_bar(self):
...         self.assertFalse(False)

>>> suite = unittest.TestLoader().loadTestsFromTestCase(TestFoo)
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_bar (__main__.TestFoo) ... ok
test_foo (__main__.TestFoo) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

10.5 Test raise exception

```
>>> import unittest

>>> class TestRaiseException(unittest.TestCase):
...     def test_raise_except(self):
...         with self.assertRaises(SystemError):
...             raise SystemError
>>> suite_loader = unittest.TestLoader()
>>> suite = suite_loader.loadTestsFromTestCase(TestRaiseException)
>>> unittest.TextTestRunner().run(suite)
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
>>> class TestRaiseFail(unittest.TestCase):
...     def test_raise_fail(self):
...         with self.assertRaises(SystemError):
...             pass
>>> suite = unittest.TestLoader().loadTestsFromTestCase(TestRaiseFail)
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_raise_fail (__main__.TestRaiseFail) ... FAIL

======================================================================
FAIL: test_raise_fail (__main__.TestRaiseFail)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<stdin>", line 4, in test_raise_fail
AssertionError: SystemError not raised

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

10.6 Pass arguments into a TestCase

```
>>> from __future__ import print_function
>>> import unittest
>>> class TestArg(unittest.TestCase):
...     def __init__(self, testname, arg):
...         super(TestArg, self).__init__(testname)
...         self._arg = arg
...     def setUp(self):
...         print("setUp:", self._arg)
...     def test_arg(self):
...         print("test_arg:", self._arg)
...         self.assertTrue(True)
...
>>> suite = unittest.TestSuite()
>>> suite.addTest(TestArg('test_arg', 'foo'))
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_arg (__main__.TestArg) ... setUp: foo
test_arg: foo
ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

10.7 Group multiple testcases into a suite

```
>>> import unittest
>>> class TestFooBar(unittest.TestCase):
...     def test_foo(self):
...         self.assertTrue(True)
...     def test_bar(self):
...         self.assertTrue(True)
...
>>> class TestHelloWorld(unittest.TestCase):
...     def test_hello(self):
...         self.assertEqual("Hello", "Hello")
...     def test_world(self):
...         self.assertEqual("World", "World")
...
>>> suite_loader = unittest.TestLoader()
>>> suite1 = suite_loader.loadTestsFromTestCase(TestFooBar)
>>> suite2 = suite_loader.loadTestsFromTestCase(TestHelloWorld)
>>> suite = unittest.TestSuite([suite1, suite2])
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_bar (__main__.TestFooBar) ... ok
test_foo (__main__.TestFooBar) ... ok
test_hello (__main__.TestHelloWorld) ... ok
test_world (__main__.TestHelloWorld) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

10.8 Group multiple tests from different TestCase

```
>>> import unittest
>>> class TestFoo(unittest.TestCase):
...     def test_foo(self):
...         assert "foo" == "foo"
...
>>> class TestBar(unittest.TestCase):
...     def test_bar(self):
...         assert "bar" == "bar"
...
>>> suite = unittest.TestSuite()
>>> suite.addTest(TestFoo('test_foo'))
>>> suite.addTest(TestBar('test_bar'))
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_foo (__main__.TestFoo) ... ok
test_bar (__main__.TestBar) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

10.9 Skip some tests in the TestCase

```
>>> import unittest
>>> RUN_FOO = False
>>> DONT_RUN_BAR = False
>>> class TestSkip(unittest.TestCase):
...     def test_always_run(self):
...         self.assertTrue(True)
...     @unittest.skip("always skip this test")
...     def test_always_skip(self):
...         raise RuntimeError
...     @unittest.skipIf(RUN_FOO == False, "demo skipIf")
...     def test_skipif(self):
...         raise RuntimeError
...     @unittest.skipUnless(DONT_RUN_BAR == True, "demo skipUnless")
...     def test_skipunless(self):
...         raise RuntimeError
...
>>> suite = unittest.TestLoader().loadTestsFromTestCase(TestSkip)
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_always_run (__main__.TestSkip) ... ok
test_always_skip (__main__.TestSkip) ... skipped 'always skip this test'
test_skipif (__main__.TestSkip) ... skipped 'demo skipIf'
test_skipunless (__main__.TestSkip) ... skipped 'demo skipUnless'

----------------------------------------------------------------------
Ran 4 tests in 0.000s
```

```
OK (skipped=3)
```

10.10 Monolithic Test

```
>>> from __future__ import print_function
>>> import unittest
>>> class Monolithic(unittest.TestCase):
...     def step1(self):
...         print('step1')
...     def step2(self):
...         print('step2')
...     def step3(self):
...         print('step3')
...     def _steps(self):
...         for attr in sorted(dir(self)):
...             if not attr.startswith('step'):
...                 continue
...             yield attr
...     def test_foo(self):
...         for _s in self._steps():
...             try:
...                 getattr(self, _s)()
...             except Exception as e:
...                 self.fail('{} failed({})'.format(attr, e))
...
>>> suite = unittest.TestLoader().loadTestsFromTestCase(Monolithic)
>>> unittest.TextTestRunner().run(suite)
step1
step2
step3
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
<unittest.runner.TextTestResult run=1 errors=0 failures=0>
```

10.11 Cross-module variables to Test files

test_foo.py

```
from __future__ import print_function

import unittest

print(conf)

class TestFoo(unittest.TestCase):
    def test_foo(self):
        print(conf)

    @unittest.skipIf(conf.isskip==True, "skip test")
```

```
    def test_skip(self):
        raise RuntimeError
```

test_bar.py

```
from __future__ import print_function

import unittest
import __builtin__

if __name__ == "__main__":
    conf = type('TestConf', (object,), {})
    conf.isskip = True

    # make a cross-module variable
    __builtin__.conf = conf
    module = __import__('test_foo')
    loader = unittest.TestLoader()
    suite = loader.loadTestsFromTestCase(module.TestFoo)
    unittest.TextTestRunner(verbosity=2).run(suite)
```

output:

```
$ python test_bar.py
<class '__main__.TestConf'>
test_foo (test_foo.TestFoo) ... <class '__main__.TestConf'>
ok
test_skip (test_foo.TestFoo) ... skipped 'skip test'

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK (skipped=1)
```

10.12 skip setup & teardown when the test is skipped

```
>>> from __future__ import print_function
>>> import unittest
>>> class TestSkip(unittest.TestCase):
...     def setUp(self):
...         print("setUp")
...     def tearDown(self):
...         print("tearDown")
...     @unittest.skip("skip this test")
...     def test_skip(self):
...         raise RuntimeError
...     def test_not_skip(self):
...         self.assertTrue(True)
...
>>> suite = unittest.TestLoader().loadTestsFromTestCase(TestSkip)
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_not_skip (__main__.TestSkip) ... setUp
tearDown
ok
test_skip (__main__.TestSkip) ... skipped 'skip this test'
```

```
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK (skipped=1)
```

10.13 Re-using old test code

```
>>> from __future__ import print_function
>>> import unittest
>>> def old_func_test():
...     assert "Hello" == "Hello"
...
>>> def old_func_setup():
...     print("setup")
...
>>> def old_func_teardown():
...     print("teardown")
...
>>> testcase = unittest.FunctionTestCase(old_func_test,
...                                       setUp=old_func_setup,
...                                       tearDown=old_func_teardown)
>>> suite = unittest.TestSuite([testcase])
>>> unittest.TextTestRunner().run(suite)
setup
teardown
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
<unittest.runner.TextTestResult run=1 errors=0 failures=0>
```

10.14 Testing your document is right

```
"""
This is an example of doctest

>>> fib(10)
55
"""

def fib(n):
""" This function calculate fib number.

Example:

    >>> fib(10)
    55
    >>> fib(-1)
    Traceback (most recent call last):
    ...
```

```
    ValueError
"""
if n < 0:
    raise ValueError('')
return 1 if n<=2 else fib(n-1) + fib(n-2)

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

output:

```
$ python demo_doctest.py -v
Trying:
fib(10)
Expecting:
55
ok
Trying:
fib(10)
Expecting:
55
ok
Trying:
fib(-1)
Expecting:
Traceback (most recent call last):
...
ValueError
ok
2 items passed all tests:
1 tests in __main__
2 tests in __main__.fib
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
```

10.15 Re-using doctest to unittest

```
import unittest
import doctest

"""
This is an example of doctest

>>> fib(10)
55
"""

def fib(n):
    """ This function calculate fib number.

    Example:

        >>> fib(10)
```

```
        55
        >>> fib(-1)
        Traceback (most recent call last):
            ...
        ValueError
    """
    if n < 0:
        raise ValueError('')
    return 1 if n<=2 else fib(n-1) + fib(n-2)

if __name__ == "__main__":
    finder = doctest.DocTestFinder()
    suite = doctest.DocTestSuite(test_finder=finder)
    unittest.TextTestRunner(verbosity=2).run(suite)
```

output:

```
fib (__main__)
Doctest: __main__.fib ... ok

----------------------------------------------------------------------
Ran 1 test in 0.023s

OK
```

10.16 Customize test report

```
from unittest import (
        TestCase,
        TestLoader,
        TextTestResult,
        TextTestRunner)

from pprint import pprint
import unittest
import os

OK = 'ok'
FAIL = 'fail'
ERROR = 'error'
SKIP = 'skip'

class JsonTestResult(TextTestResult):

    def __init__(self, stream, descriptions, verbosity):
        super_class = super(JsonTestResult, self)
        super_class.__init__(stream, descriptions, verbosity)

        # TextTestResult has no successes attr
        self.successes = []

    def addSuccess(self, test):
        # addSuccess do nothing, so we need to overwrite it.
        super(JsonTestResult, self).addSuccess(test)
        self.successes.append(test)
```

```
    def json_append(self, test, result, out):
        suite = test.__class__.__name__
        if suite not in out:
            out[suite] = {OK: [], FAIL: [], ERROR:[], SKIP: []}
        if result is OK:
            out[suite][OK].append(test._testMethodName)
        elif result is FAIL:
            out[suite][FAIL].append(test._testMethodName)
        elif result is ERROR:
            out[suite][ERROR].append(test._testMethodName)
        elif result is SKIP:
            out[suite][SKIP].append(test._testMethodName)
        else:
            raise KeyError("No such result: {}".format(result))
        return out

    def jsonify(self):
        json_out = dict()
        for t in self.successes:
            json_out = self.json_append(t, OK, json_out)

        for t, _ in self.failures:
            json_out = self.json_append(t, FAIL, json_out)

        for t, _ in self.errors:
            json_out = self.json_append(t, ERROR, json_out)

        for t, _ in self.skipped:
            json_out = self.json_append(t, SKIP, json_out)

        return json_out

class TestSimple(TestCase):

    def test_ok_1(self):
        foo = True
        self.assertTrue(foo)

    def test_ok_2(self):
        bar = True
        self.assertTrue(bar)

    def test_fail(self):
        baz = False
        self.assertTrue(baz)

    def test_raise(self):
        raise RuntimeError

    @unittest.skip("Test skip")
    def test_skip(self):
        raise NotImplementedError

if __name__ == '__main__':
    # redirector default output of unittest to /dev/null
    with open(os.devnull, 'w') as null_stream:
        # new a runner and overwrite resultclass of runner
```

```
        runner = TextTestRunner(stream=null_stream)
        runner.resultclass = JsonTestResult

        # create a testsuite
        suite = TestLoader().loadTestsFromTestCase(TestSimple)

        # run the testsuite
        result = runner.run(suite)

        # print json output
        pprint(result.jsonify())
```

output:

```
$ python test.py
{'TestSimple': {'error': ['test_raise'],
                'fail': ['test_fail'],
                'ok': ['test_ok_1', 'test_ok_2'],
                'skip': ['test_skip']}}
```

10.17 Mock - using `@patch` substitute original method

```
# python-3.3 or above

>>> from unittest.mock import patch
>>> import os
>>> def fake_remove(path, *a, **k):
...     print("remove done")
...
>>> @patch('os.remove', fake_remove)
... def test():
...     try:
...         os.remove('%$!?&*') # fake os.remove
...     except OSError as e:
...         print(e)
...     else:
...         print('test success')
...
>>> test()
remove done
test success
```

Note: Without mock, above test will always fail.

```
>>> import os
>>> def test():
...     try:
...         os.remove('%$!?&*')
...     except OSError as e:
...         print(e)
...     else:
...         print('test success')
...
```

```
>>> test()
[Errno 2] No such file or directory: '%$!?&*'
```

10.18 What `with unittest.mock.patch` do?

```
from unittest.mock import patch
import os

PATH = '$@!%?&'

def fake_remove(path):
    print("Fake remove")


class SimplePatch:

    def __init__(self, target, new):
        self._target = target
        self._new = new

    def get_target(self, target):
        target, attr = target.rsplit('.', 1)
        getter = __import__(target)
        return getter, attr

    def __enter__(self):
        orig, attr = self.get_target(self._target)
        self.orig, self.attr = orig, attr
        self.orig_attr = getattr(orig, attr)
        setattr(orig, attr, self._new)
        return self._new

    def __exit__(self, *exc_info):
        setattr(self.orig, self.attr, self.orig_attr)
        del self.orig_attr


print('---> inside unittest.mock.patch scope')
with patch('os.remove', fake_remove):
    os.remove(PATH)

print('---> inside simple patch scope')
with SimplePatch('os.remove', fake_remove):
    os.remove(PATH)

print('---> outside patch scope')
try:
    os.remove(PATH)
except OSError as e:
    print(e)
```

output:

```
$ python3 simple_patch.py
---> inside unittest.mock.patch scope
```

```
Fake remove
---> inside simple patch scope
Fake remove
---> outside patch scope
[Errno 2] No such file or directory: '$@!%?&'
```

10.19 Mock - substitute open

```
>>> import urllib
>>> from unittest.mock import patch, mock_open
>>> def send_req(url):
...     with urllib.request.urlopen(url) as f:
...         if f.status == 200:
...             return f.read()
...         raise urllib.error.URLError
...
>>> fake_html = b'<html><h1>Mock Content</h1></html>'
>>> mock_urlopen = mock_open(read_data=fake_html)
>>> ret = mock_urlopen.return_value
>>> ret.status = 200
>>> @patch('urllib.request.urlopen', mock_urlopen)
... def test_send_req_success():
...     try:
...         ret = send_req('http://www.mockurl.com')
...         assert ret == fake_html
...     except Exception as e:
...         print(e)
...     else:
...         print('test send_req success')
...
>>> test_send_req_success()
test send_req success
>>> ret = mock_urlopen.return_value
>>> ret.status = 404
>>> @patch('urllib.request.urlopen', mock_urlopen)
... def test_send_req_fail():
...     try:
...         ret = send_req('http://www.mockurl.com')
...         assert ret == fake_html
...     except Exception as e:
...         print('test fail success')
...
>>> test_send_req_fail()
test fail success
```

CHAPTER 11

Python C API cheatsheet

Table of Contents

- *Python C API cheatsheet*
  - *Performance of ctypes*
  - *Error handling when use ctypes*
  - *Getting File System Type*
  - *Doing Zero-copy via sendfile*
  - *PyObject header*
  - *Python C API Template*
    - *C API source*
    - *Prepare setup.py*
    - *Build C API source*
    - *Run the C module*
  - *PyObject with Member and Methods*
    - *C API source*
    - *Compare performance with pure Python*

11.1 Performance of ctypes

```
// fib.c
unsigned int fib(unsigned int n)
{
```

```
    if ( n < 2) {
        return n;
    }
    return fib(n-1) + fib(n-2);
}
```

Building a libfib.dylib (Mac OSX)

```
clang -Wall -Werror -shared -fPIC -o libfib.dylib fib.c
```

Comparing the performance

```
>>> import time
>>> from ctypes import *
>>> def fib(n):
...     if n < 2:
...         return n
...     return fib(n-1) + fib(n-2)
...
>>> s = time.time(); fib(35); e = time.time()
9227465
>>> print("cost time: {} sec".format(e - s))
cost time: 4.09563493729 sec
>>> libfib = CDLL("./libfib.dylib")
>>> s = time.time(); libfib.fib(35); e = time.time()
9227465
>>> print("cost time: {} sec".format(e - s))
cost time: 0.0819959640503 sec
```

11.2 Error handling when use ctypes

```
from __future__ import print_function

import errno
import os

from ctypes import *
from sys import platform, maxsize

is_64bits = maxsize > 2**32

if is_64bits and platform == 'darwin':
    libc = CDLL("libc.dylib", use_errno=True)
else:
    raise RuntimeError("Not support platform: {}".format(platform))

stat = libc.stat

class Stat(Structure):
    '''
    From /usr/include/sys/stat.h

    struct stat {
        dev_t          st_dev;
        ino_t          st_ino;
```

```
        mode_t          st_mode;
        nlink_t         st_nlink;
        uid_t           st_uid;
        gid_t           st_gid;
        dev_t           st_rdev;
    #ifndef _POSIX_SOURCE
        struct      timespec st_atimespec;
        struct      timespec st_mtimespec;
        struct      timespec st_ctimespec;
    #else
        time_t          st_atime;
        long            st_atimensec;
        time_t          st_mtime;
        long            st_mtimensec;
        time_t          st_ctime;
        long            st_ctimensec;
    #endif
        off_t           st_size;
        int64_t         st_blocks;
        u_int32_t       st_blksize;
        u_int32_t       st_flags;
        u_int32_t       st_gen;
        int32_t         st_lspare;
        int64_t         st_qspare[2];
    };
    '''
    _fields_ = [('st_dev',          c_ulong),
                ('st_ino',          c_ulong),
                ('st_mode',         c_ushort),
                ('st_nlink',        c_uint),
                ('st_uid',          c_uint),
                ('st_gid',          c_uint),
                ('st_rdev',         c_ulong),
                ('st_atime',        c_longlong),
                ('st_atimendesc',   c_long),
                ('st_mtime',        c_longlong),
                ('st_mtimendesc',   c_long),
                ('st_ctime',        c_longlong),
                ('st_ctimendesc',   c_long),
                ('st_size',         c_ulonglong),
                ('st_blocks',       c_int64),
                ('st_blksize',      c_uint32),
                ('st_flags',        c_uint32),
                ('st_gen',          c_uint32),
                ('st_lspare',       c_int32),
                ('st_qspare',       POINTER(c_int64) * 2)]

# stat success
path = create_string_buffer(b"/etc/passwd")
st = Stat()
ret = stat(path, byref(st))
assert ret == 0

# if stat fail, check errno
path = create_string_buffer(b"&%$#@!")
st = Stat()
ret = stat(path, byref(st))
if ret != 0:
```

```
    errno_ = get_errno() # get errno
    errmsg = "stat({}) failed. {}".format(path.raw, os.strerror(errno_))
    raise OSError(errno_, errmsg)
```

output:

```
$ python err_handling.py   # python2
Traceback (most recent call last):
  File "err_handling.py", line 85, in <module>
    raise OSError(errno_, errmsg)
OSError: [Errno 2] stat(&%$#@!) failed. No such file or directory

$ python3 err_handling.py  # python3
Traceback (most recent call last):
  File "err_handling.py", line 85, in <module>
    raise OSError(errno_, errmsg)
FileNotFoundError: [Errno 2] stat(b'&%$#@!\x00') failed. No such file or directory
```

11.3 Getting File System Type

```
from __future__ import print_function

from ctypes import *
from sys import platform

if platform not in ('linux', 'linux2'):
    raise RuntimeError("Not support '{}'".format(platform))


# from Linux/include/uapi/linux/magic.h

EXT_SUPER_MAGIC      = 0x137D
EXT2_OLD_SUPER_MAGIC = 0xEF51
EXT2_SUPER_MAGIC     = 0xEF53
EXT3_SUPER_MAGIC     = 0xEF53
EXT4_SUPER_MAGIC     = 0xEF53
BTRFS_SUPER_MAGIC    = 0x9123683E


class KernelFsid(Structure):
    '''
    From Linux/arch/mips/include/asm/posix_types.h

    typedef struct {
            long    val[2];
    } __kernel_fsid_t;
    '''
    _fields_ = [('val', POINTER(c_long) * 2)]

class Statfs(Structure):
    '''
    From Linux/arch/mips/include/asm/statfs.h

    struct statfs {
            long            f_type;
```

```
    #define f_fstyp f_type
            long            f_bsize;
            long            f_frsize;
            long            f_blocks;
            long            f_bfree;
            long            f_files;
            long            f_ffree;
            long            f_bavail;

            /* Linux specials */
            __kernel_fsid_t f_fsid;
            long            f_namelen;
            long            f_flags;
            long            f_spare[5];
    };
    '''
    _fields_ = [('f_type',    c_long),
                ('f_bsize',   c_long),
                ('f_frsize',  c_long),
                ('f_block',   c_long),
                ('f_bfree',   c_long),
                ('f_files',   c_long),
                ('f_ffree',   c_long),
                ('f_fsid',    KernelFsid),
                ('f_namelen', c_long),
                ('f_flags',   c_long),
                ('f_spare',   POINTER(c_long) * 5)]


libc = CDLL('libc.so.6', use_errno=True)
statfs = libc.statfs

path = create_string_buffer(b'/etc')
fst = Statfs()
ret = statfs(path, byref(fst))
assert ret == 0

print('Is ext4? {}'.format(fst.f_type == EXT4_SUPER_MAGIC))
```

output:

```
$ python3 statfs.py
Is ext4? True
```

11.4 Doing Zero-copy via sendfile

```
from __future__ import print_function, unicode_literals

import os
import sys
import errno
import platform

from ctypes import *
```

```
# check os
p = platform.system()
if p != "Linux":
    raise OSError("Not support '{}'".format(p))

# check linux version
ver = platform.release()
if tuple(map(int, ver.split('.'))) < (2,6,33):
    raise OSError("Upgrade kernel after 2.6.33")

# check input arguments
if len(sys.argv) != 3:
    print("Usage: sendfile.py f1 f2", file=sys.stderr)
    exit(1)

libc = CDLL('libc.so.6', use_errno=True)
sendfile = libc.sendfile

src = sys.argv[1]
dst = sys.argv[2]
src_size = os.stat(src).st_size

# clean destination first
try:
    os.remove(dst)
except OSError as e:
    if e.errno != errno.ENOENT: raise

offset = c_int64(0)

with open(src, 'r') as f1:
    with open(dst, 'w') as f2:
        src_fd = c_int(f1.fileno())
        dst_fd = c_int(f2.fileno())
        ret = sendfile(dst_fd, src_fd, byref(offset), src_size)
        if ret < 0:
            errno_ = get_errno()
            errmsg = "sendfile failed. {}".format(os.strerror(errno_))
            raise OSError(errno_, errmsg)
```

output:

```
$ python3 sendfile.py /etc/resolv.conf resolve.conf; cat resolve.conf
nameserver  192.168.1.1
```

11.5 PyObject header

```
// ref: python source code
// Python/Include/object.c
#define _PyObject_HEAD_EXTRA \
    struct _object *_ob_next;\
    struct _object *_ob_prev;

#define PyObject_HEAD    \
    _PyObject_HEAD_EXTRA \
```

```
    Py_ssize_t ob_refcnt;\
    struct _typeobject *ob_type;
```

11.6 Python C API Template

11.6.1 C API source

```
#include <Python.h>

typedef struct {
    PyObject_HEAD
} spamObj;

static PyTypeObject spamType = {
    PyObject_HEAD_INIT(&PyType_Type)
    0,                  //ob_size
    "spam.Spam",        //tp_name
    sizeof(spamObj),    //tp_basicsize
    0,                  //tp_itemsize
    0,                  //tp_dealloc
    0,                  //tp_print
    0,                  //tp_getattr
    0,                  //tp_setattr
    0,                  //tp_compare
    0,                  //tp_repr
    0,                  //tp_as_number
    0,                  //tp_as_sequence
    0,                  //tp_as_mapping
    0,                  //tp_hash
    0,                  //tp_call
    0,                  //tp_str
    0,                  //tp_getattro
    0,                  //tp_setattro
    0,                  //tp_as_buffer
    Py_TPFLAGS_DEFAULT, //tp_flags
    "spam objects",     //tp_doc
};

static PyMethodDef spam_methods[] = {
    {NULL}  /* Sentinel */
};

/* declarations for DLL import */
#ifndef PyMODINIT_FUNC
#define PyMODINIT_FUNC void
#endif

PyMODINIT_FUNC
initspam(void)
{
    PyObject *m;
    spamType.tp_new = PyType_GenericNew;
    if (PyType_Ready(&spamType) < 0) {
        goto END;
    }
```

```
    m = Py_InitModule3("spam", spam_methods, "Example of Module");
    Py_INCREF(&spamType);
    PyModule_AddObject(m, "spam", (PyObject *)&spamType);
END:
    return;
}
```

11.6.2 Prepare setup.py

```
from distutils.core import setup
from distutils.core import Extension

setup(name="spam",
      version="1.0",
      ext_modules=[Extension("spam", ["spam.c"])])
```

11.6.3 Build C API source

```
$ python setup.py build
$ python setup.py install
```

11.6.4 Run the C module

```
>>> import spam
>>> spam.__doc__
'Example of Module'
>>> spam.spam
<type 'spam.Spam'>
```

11.7 PyObject with Member and Methods

11.7.1 C API source

```
#include <Python.h>
#include <structmember.h>

typedef struct {
    PyObject_HEAD
    PyObject *hello;
    PyObject *world;
    int spam_id;
} spamObj;

static void
spamdealloc(spamObj *self)
{
    Py_XDECREF(self->hello);
    Py_XDECREF(self->world);
```

```
    self->ob_type
        ->tp_free((PyObject*)self);
}

/* __new__ */
static PyObject *
spamNew(PyTypeObject *type, PyObject *args, PyObject *kwds)
{
    spamObj *self = NULL;

    self = (spamObj *)
           type->tp_alloc(type, 0);
    if (self == NULL) {
        goto END;
    }
    /* alloc str to hello */
    self->hello =
        PyString_FromString("");
    if (self->hello == NULL)
    {
        Py_XDECREF(self);
        self = NULL;
        goto END;
    }
    /* alloc str to world */
    self->world =
        PyString_FromString("");
    if (self->world == NULL)
    {
        Py_XDECREF(self);
        self = NULL;
        goto END;
    }
    self->spam_id = 0;
END:
    return (PyObject *)self;
}

/* __init__ */
static int
spamInit(spamObj *self, PyObject *args, PyObject *kwds)
{
    int ret = -1;
    PyObject *hello=NULL,
             *world=NULL,
             *tmp=NULL;

    static char *kwlist[] = {
        "hello",
        "world",
        "spam_id", NULL};

    /* parse input arguments */
    if (! PyArg_ParseTupleAndKeywords(
          args, kwds,
          "|OOi",
          kwlist,
          &hello, &world,
```

```
          &self->spam_id)) {
        goto END;
    }
    /* set attr hello */
    if (hello) {
        tmp = self->hello;
        Py_INCREF(hello);
        self->hello = hello;
        Py_XDECREF(tmp);
    }
    /* set attr world */
    if (world) {
        tmp = self->world;
        Py_INCREF(world);
        self->world = world;
        Py_XDECREF(tmp);
    }
    ret = 0;
END:
    return ret;
}

static long
fib(long n) {
    if (n<=2) {
        return 1;
    }
    return fib(n-1)+fib(n-2);
}

static PyObject *
spamFib(spamObj *self, PyObject *args)
{
    PyObject  *ret = NULL;
    long arg = 0;

    if (!PyArg_ParseTuple(args, "i", &arg)) {
        goto END;
    }
    ret = PyInt_FromLong(fib(arg));
END:
    return ret;
}

//ref: python doc
static PyMemberDef spam_members[] = {
    /* spameObj.hello*/
    {"hello",                   //name
     T_OBJECT_EX,               //type
     offsetof(spamObj, hello),  //offset
     0,                         //flags
     "spam hello"},             //doc
    /* spamObj.world*/
    {"world",
     T_OBJECT_EX,
     offsetof(spamObj, world),
     0,
     "spam world"},
```

```
    /* spamObj.spam_id*/
    {"spam_id",
     T_INT,
     offsetof(spamObj, spam_id),
     0,
     "spam id"},
    /* Sentiel */
    {NULL}
};

static PyMethodDef spam_methods[] = {
    /* fib */
    {"spam_fib",
     (PyCFunction)spamFib,
     METH_VARARGS,
     "Calculate fib number"},
    /* Sentiel */
    {NULL}
};

static PyMethodDef module_methods[] = {
    {NULL}  /* Sentinel */
};

static PyTypeObject spamKlass = {
    PyObject_HEAD_INIT(NULL)
    0,                                  //ob_size
    "spam.spamKlass",                   //tp_name
    sizeof(spamObj),                    //tp_basicsize
    0,                                  //tp_itemsize
    (destructor) spamdealloc,           //tp_dealloc
    0,                                  //tp_print
    0,                                  //tp_getattr
    0,                                  //tp_setattr
    0,                                  //tp_compare
    0,                                  //tp_repr
    0,                                  //tp_as_number
    0,                                  //tp_as_sequence
    0,                                  //tp_as_mapping
    0,                                  //tp_hash
    0,                                  //tp_call
    0,                                  //tp_str
    0,                                  //tp_getattro
    0,                                  //tp_setattro
    0,                                  //tp_as_buffer
    Py_TPFLAGS_DEFAULT |
    Py_TPFLAGS_BASETYPE,                //tp_flags
    "spamKlass objects",                //tp_doc
    0,                                  //tp_traverse
    0,                                  //tp_clear
    0,                                  //tp_richcompare
    0,                                  //tp_weaklistoffset
    0,                                  //tp_iter
    0,                                  //tp_iternext
    spam_methods,                       //tp_methods
    spam_members,                       //tp_members
    0,                                  //tp_getset
    0,                                  //tp_base
```

```
    0,                                  //tp_dict
    0,                                  //tp_descr_get
    0,                                  //tp_descr_set
    0,                                  //tp_dictoffset
    (initproc)spamInit,                 //tp_init
    0,                                  //tp_alloc
    spamNew,                            //tp_new
};

/* declarations for DLL import */
#ifndef PyMODINIT_FUNC
#define PyMODINIT_FUNC void
#endif

PyMODINIT_FUNC
initspam(void)
{
    PyObject* m;

    if (PyType_Ready(&spamKlass) < 0) {
        goto END;
    }

    m = Py_InitModule3(
      "spam",          // Mod name
      module_methods, // Mod methods
      "Spam Module"); // Mod doc

    if (m == NULL) {
        goto END;
    }
    Py_INCREF(&spamKlass);
    PyModule_AddObject(
      m,                              // Module
      "SpamKlass",                    // Class Name
      (PyObject *) &spamKlass);       // Class
END:
    return;
}
```

11.7.2 Compare performance with pure Python

```
>>> import spam
>>> o = spam.SpamKlass()
>>> def profile(func):
...     def wrapper(*args, **kwargs):
...         s = time.time()
...         ret = func(*args, **kwargs)
...         e = time.time()
...         print e-s
...     return wrapper
...
>>> def fib(n):
...     if n <= 2:
...         return n
...     return fib(n-1)+fib(n-2)
```

```
...
>>> @profile
... def cfib(n):
...     o.spam_fib(n)
...
>>> @profile
... def pyfib(n):
...     fib(n)
...
>>> cfib(30)
0.0106310844421
>>> pyfib(30)
0.399799108505
```

CHAPTER 12

Python Design Pattern in C

Table of Contents

- *Python Design Pattern in C*
  - *Decorator in C*
  - *A Set of Functions*
  - *Closure in C*
  - *Generator*
  - *Context Manager in C*
  - *Tuple in C*
  - *Error Handling*
  - *Simple* `try: exp except: exp finally:` *in C*
  - *Simple coroutine in C*
  - *Keyword Arguments in C*
  - *Function "MAP"*
  - *foreach in C*
  - *Simple OOP in C*

12.1 Decorator in C

Python

```
>>> def decorator(func):
...     def wrapper(*args, **kwargs):
...         print("I am decorator")
...         ret = func(*args, **kwargs)
...         return ret
...     return wrapper
...
>>> @decorator
... def hello(str):
...     print("Hello {0}".format(str))
...
>>> @decorator
... def add(a,b):
...     print("add %d+%d=%d" % (a,b,a+b))
...     return a+b
...
>>> hello("KerKer")
I am decorator
Hello KerKer
>>> add(1,2)
I am decorator
add 1+2=3
3
```

C

```
#include <stdio.h>

#define DECORATOR(t, f, declar, input) \
   t decor_##f(declar) { \
      printf("I am decorator\n"); \
      return f(input);\
   }
#define FUNC_DEC(func, ...) \
   decor_##func(__VA_ARGS__)

// Original function
void hello(char *str) {
   printf("Hello %s\n", str);
}

int add(int a, int b) {
   printf("add %d + %d = %d\n",a,b,a+b);
   return a+b;
}
// Patch function
#define DECLAR    char *str
#define INPUT     str
DECORATOR(void, hello, DECLAR, INPUT)
#undef DECLAR
#undef INPUT

#define DECLAR    int a, int b
#define INPUT     a,b
DECORATOR(int, add, DECLAR, INPUT)
#undef DECLAR
#undef INPUT
```

```
int main(int argc, char *argv[]) {
   FUNC_DEC(hello, "KerKer");
   FUNC_DEC(add,1,2);

   return 0;
}
```

output:

```
$ gcc example.c
$ ./a.out
I am decorator
Hello KerKer
I am decorator
add 1 + 2 = 3
```

12.2 A Set of Functions

Python

```
>>> def func_1():
...     print "Hello"
...
>>> def func_2():
...     print "World"
...
>>> def func_3():
...     print "!!!"
...
>>> s = [func_1,func_2,func_3]
>>> for _ in s: _()
...
Hello
World
!!!
```

C

```
#include <stdio.h>

typedef void (*func)(void);

enum func_id{
   FUNC_1,FUNC_2,FUNC_3
};

void func_1() {
   printf("Hello ");
}
void func_2() {
   printf("World ");
}
void func_3() {
   printf("!!!\n");
}
```

```
func gFuncTable[] = {
   func_1,func_2,func_3
};

int main(int argc, char *argv[]) {
   gFuncTable[FUNC_1]();
   gFuncTable[FUNC_2]();
   gFuncTable[FUNC_3]();

   return 0;
}
```

12.3 Closure in C

Python

```
# implement via __call__
>>> class closure(object):
...     def __init__(self):
...         self.val = 5566
...     def __call__(self,var):
...         self.val += var
...
>>> c = closure()
>>> c(9527)
>>> print c.val
15093
# using "global" keyword
>>> x = 0
>>> def closure(val):
...     def wrapper():
...         global x
...         x += val
...         print x
...     wrapper()
...
>>> closure(5566)
5566
>>> closure(9527)
15093
# using "nonlocal" (only in python3)
>>> def closure(val):
...     x = 0
...     def wrapper():
...         nonlocal x
...         x += val
...         print(x)
...     wrapper()
...
>>> closure(5566)
5566
>>> closure(9527)
9527
```

C

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Closure {
   int val;
   void (*add) (struct Closure **, int);
}closure;

void add_func(closure **c, int var) {
   (*c)->val += var;
}

int main(int argc, char *argv[]) {
   closure *c = NULL;
   c = malloc(sizeof(closure));
   c->val = 5566;
   c->add = add_func;
   c->add(&c,9527);
   printf("result: %d\n",c->val);

   return 0;
}
```

12.4 Generator

Python

```
>>> def gen():
...     var = 0
...     while True:
...         var += 1
...         yield var
...
>>> g = gen()
>>> for _ in range(3):
...     print next(g),
...
1 2 3
```

C

```c
#include <stdio.h>
#include <stdlib.h>

struct gen {
   int (*next) (struct gen *);
   int var;
};

int next_func(struct gen *g) {
   printf("var = %d\n",g->var);
   g->var +=1;
   return g->var;
}
```

```
struct gen * new_gen() {
   struct gen *g = NULL;
   g = (struct gen*)
         malloc(sizeof(struct gen));
   g->var = 0;
   g->next = next_func;
   return g;
}

int main(int argc, char *argv[]) {
   struct gen *g = new_gen();
   int i = 0;
   for (i=0;i<3;i++) {
      printf("gen var = %d\n",g->next(g));
   }
   return 0;
}
```

12.5 Context Manager in C

Python

```
>>> class CtxManager(object):
...     def __enter__(self):
...         self._attr = "KerKer"
...         return self._attr
...     def __exit__(self,*e_info):
...         del self._attr
...
>>> with CtxManager() as c:
...     print c
...
KerKer
```

C

```
#include <stdio.h>
#include <stdlib.h>

#define ENTER(type,ptr,len) \
   printf("enter context manager\n");\
   ptr = malloc(sizeof(type)*len);\
   if (NULL == ptr) { \
      printf("malloc get error\n");\
      goto exit;\
   }\

#define EXIT(ptr) \
exit:\
   printf("exit context manager\n");\
   if (NULL != ptr) {\
      free(ptr);\
      ptr = NULL;  \
   }\
```

```
#define CONTEXT_MANAGER(t, p, l,...){\
   ENTER(t,p,l)\
   __VA_ARGS__ \
   EXIT(p)\
}

int main(int argc, char *argv[]) {
   char *ptr;
   CONTEXT_MANAGER(char, ptr, 128,
      sprintf(ptr, "Hello World");
      printf("%s\n",ptr);
   );
   printf("ptr = %s\n",ptr);
   return 0;
}
```

12.6 Tuple in C

Python

```
>>> a = ("Hello","World",123)
>>> for _ in a: print _,
...
Hello World 123
```

C

```
#include <stdio.h>

int main(int argc, char *argv[]) {
   int a = 123;
   void * const x[4] = {"Hello",
                        "World",&a};
   printf("%s %s, %d\n",x[0],x[1],*(int *)x[2]);
   return 0;
}
```

12.7 Error Handling

Python

```
>>> import os
>>> def spam(a,b):
...     try:
...         os.listdir('.')
...     except OSError:
...         print "listdir get error"
...         return
...     try:
...         a/b
...     except ZeroDivisionError:
...         print "zero division"
...         return
```

```
...
>>> spam(1,0)
zero division
# single exit -> using decorator
>>> import time
>>> def profile(func):
...     def wrapper(*args, **kwargs):
...         s = time.time()
...         ret = func(*args, **kwargs)
...         e = time.time()
...         print e - s
...         return ret
...     return wrapper
...
>>> @profile
... def spam(a,b):
...     try:
...         os.listdir('.')
...     except OSError:
...         return
...     try:
...         a/b
...     except ZeroDivisionError:
...         return
...
>>> spam(1,0)
0.000284910202026
```

C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
   int ret = -1;
   char *ptr;
   ptr = malloc(sizeof(char)*128);
   if (NULL == ptr) {
      perror("malloc get error");
      goto exit;
   }
   strcpy(ptr,"KerKer");
   printf("%s\n", ptr);
   ret = 0;
exit:
   if (ptr) {
      free(ptr);
      ptr = NULL;
   }
   return ret;
}
```

12.8 Simple `try: exp except: exp finally:` in C

Python

```
>>> try:
...     # do something...
...     raise OSError
... except OSError as e:
...     print('get error OSError')
... finally:
...     print('finally block')
...
get error OSError
finally block
```

C

```
#include <stdio.h>
#include <string.h>
#include <setjmp.h>

enum {
    ERR_EPERM = 1,
    ERR_ENOENT,
    ERR_ESRCH,
    ERR_EINTR,
    ERR_EIO
};

#define try    do { jmp_buf jmp_env__;              \
                    switch ( setjmp(jmp_env__) ) { \
                        case 0: while(1) {
#define except(exc)         break;                  \
                        case exc:
#define finally         break; }                    \
                    default:
#define end  } } while(0)

#define raise(exc) longjmp(jmp_env__, exc)

int main(int argc, char *argv[])
{
    int ret = 0;

    try {
        raise(ERR_ENOENT);
    } except(ERR_EPERM) {
        printf("get exception: %s\n", strerror(ERR_EPERM));
        ret = -1;
    } except(ERR_ENOENT) {
        printf("get exception: %s\n", strerror(ERR_ENOENT));
        ret = -1;
    } except(ERR_ESRCH) {
        printf("get exception: %s\n", strerror(ERR_ENOENT));
        ret = -1;
    } finally {
        printf("finally block\n");
    } end;
    return ret;
}
```

12.9 Simple coroutine in C

Python

```
from collections import deque

_registry = { }
_msg_queue = deque()

def send(name, msg):
    _msg_queue.append((name, msg))

def actor(func):
    def wrapper(*args, **kwargs):
        gen = func(*args, **kwargs)
        next(gen)
        _registry[func.__name__] = gen
    return wrapper

@actor
def ping():
    """ coroutine ping """
    n = yield
    print('ping %d' % n)
    send('pong', 20001)

    n = yield
    print('ping %d' % n)
    send('pong', 20002)

@actor
def pong():
    """ coroutine pong """
    n = yield
    print('pong %d' % n)
    send('ping', 10001)

    n = yield
    print('pong %d' % n)
    send('ping', 10002)

def run():
    while _msg_queue:
        try:
            name, msg = _msg_queue.popleft()
            _registry[name].send(msg)
        except StopIteration:
            pass

ping()
pong()
send('ping', 10001)
run()
```

output:

```
$ python coro.py
ping 10001
```

```
pong 20001
ping 10001
pong 20002
```

C

```c
#include <stdio.h>
#include <string.h>
#include <setjmp.h>

static jmp_buf jmp_ping, jmp_pong;

#define send(buf_a, buf_b, val)  \
    do {                         \
        r = setjmp(buf_a);       \
        if (r == 0) {            \
            longjmp(buf_b, val); \
        }                        \
    } while(0)

#define GEN_FUNC(func) void func


GEN_FUNC(ping) ();
GEN_FUNC(pong) ();

GEN_FUNC(ping) ()
{
    int r = 0;

    r = setjmp(jmp_ping);
    if (r == 0) pong();
    printf("ping %d\n", r);

    /* ping -- 20001 -> pong */
    send(jmp_ping, jmp_pong, 20001);
    printf("ping %d\n", r);

    /* ping -- 20002 -> pong */
    send(jmp_ping, jmp_pong, 20002);

}

GEN_FUNC(pong) ()
{
    int r = 0;

    /* pong -- 10001 -> ping */
    send(jmp_pong, jmp_ping, 10001);
    printf("pong %d\n", r);

    /* pong -- 10002 -> ping */
    send(jmp_pong, jmp_ping, 10002);
    printf("pong %d\n", r);
}

int main(int argc, char *argv[])
{
```

```
    ping();
    return 0;
}
```

output:

```
$ ./a.out
ping 10001
pong 20001
ping 10002
pong 20002
```

12.10 Keyword Arguments in C

Python

```
>>> def f(str_, float_,int_=0):
...     print(str_, float_, int_)
...
>>> f("KerKer",2.0,2)
KerKer 2.0 2
>>> f("HaHa",3.)
HaHa 3.0 0
```

C

```
#include <stdio.h>

#define FUNC(...) \
   base_func((struct input ){.var=0, ##__VA_ARGS__});

struct input {
   char *str;
   int var;
   double dvar;
};

void base_func(struct input in){
   printf("str = %s, var = %d"
      ", dvar = %lf\n",
      in.str, in.var,in.dvar);
}

int main(int argc, char *argv[]) {
   FUNC(.str="KerKer", 2.0);
   FUNC(2, .str="KerKer");
   FUNC(.var=10, .dvar=2.0, .str="HAHA");
   return 0;
}
```

12.11 Function “MAP”

Python

```
>>> x = [1,2,3,4,5]
>>> y = map(lambda x:2*x, x)
>>> print y
[2, 4, 6, 8, 10]
#or
>>> x = [1,2,3,4,5]
>>> y = [2*_ for _ in x]
>>> print y
[2, 4, 6, 8, 10]
```

C

```
#include <stdio.h>

#define MAP(func, src, dst, len) \
   do {\
      unsigned i=0;\
      for(i=0; i<len; i++) {\
         dst[i] = func(src[i]);\
      }\
   }while(0);

int multi2(int a) {
   return 2*a;
}

int main(int argc, char *argv[]) {
   int x[] = {1,2,3,4,5};
   int y[5] = {0};
   int i = 0;

   MAP(multi2, x, y, 5);
   for(i=0;i<5;i++) {
      printf("%d ",y[i]);
   }
   printf("\n");
}
```

12.12 foreach in C

Python

```
>>> x = ["Hello","World","!!!"]
>>> for _ in x:print _,
...
Hello World !!!
```

C

```
#include <stdio.h>

#define foreach(it, x,...) \
   for(char **it=x;*it;it++) {__VA_ARGS__}

int main(int argc, char *argv[]) {
```

```
   char *x[] = {"Hello","World",
                "!!!",NULL};
   foreach(it,x,
      printf("%s ",*it);
   )
   printf("\n");
   return 0;
}
```

12.13 Simple OOP in C

Python

```
# common declaration
>>> class obj(object):
...     def __init__(self):
...         self.a = 0
...         self.b = 0
...     def add(self):
...         return self.a + self.b
...     def sub(self):
...         return self.a - self.b
...
>>> o = obj()
>>> o.a = 9527
>>> o.b = 5566
>>> o.add()
15093
>>> o.sub()
3961
# patch class (more like ooc)
>>> class obj(object):
...     def __init__(self):
...         self.a = 0
...         self.b = 0
...
>>> def add(self):
...     return self.a+self.b
...
>>> def sub(self):
...     return self.a - self.b
...
>>> obj.add = add
>>> obj.sub = sub
>>> o = obj()
>>> o.a = 9527
>>> o.b = 5566
>>> o.add()
15093
>>> o.sub()
3961
```

C

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct object Obj;
typedef int (*func)(Obj *);

struct object {
   int a;
   int b;
   // virtual
   func add;
   func sub;
};
int add_func(Obj *self) {
   return self->a + self->b;
}
int sub_func(Obj *self) {
   return self->a - self->b;
}
int init_obj(Obj **self) {
   *self = malloc(sizeof(Obj));
   if (NULL == *self) {
      return -1;
   }
   (*self)->a = 0;
   (*self)->b = 0;
   (*self)->add = add_func;
   (*self)->sub = sub_func;
   return 0;
}

int main(int argc, char *argv[]) {
   Obj *o = NULL;
   init_obj(&o);
   o->a = 9527;
   o->b = 5566;
   printf("add = %d\n",o->add(o));
   printf("sub = %d\n",o->sub(o));
   return 0;
}
```