
python-cheatsheet Documentation

Release 0.1.0

crazyguitar

Oct 12, 2018

1	Python basic cheatsheet	3
1.1	Python Naming Rule	4
1.2	Using <code>__future__</code> backport features	5
1.3	Check python version	6
1.4	Check object attributes	6
1.5	Define a function <code>__doc__</code>	6
1.6	Check instance type	7
1.7	Check, Get, Set attribute	7
1.8	Check inheritance	7
1.9	Check all global variables	8
1.10	Check callable	8
1.11	Get function/class name	8
1.12	<code>__new__</code> & <code>__init__</code>	8
1.13	The diamond problem	9
1.14	Representations of your class behave	9
1.15	Break up a long string	10
1.16	Get list item SMART	10
1.17	Get dictionary item SMART	12
1.18	Set a list/dict SMART	12
1.19	set operations	13
1.20	NamedTuple	14
1.21	<code>__iter__</code> - Delegating Iteration	15
1.22	Using Generator as Iterator	15
1.23	Emulating a list	15
1.24	Emulating a dictionary	16
1.25	Decorator	17
1.26	Decorator with arguments	18
1.27	for: exp else: exp	19
1.28	try: exp else: exp	19
1.29	Lambda function	20
1.30	Option arguments - (*args, **kwargs)	20
1.31	<code>type()</code> declare (create) a class	20
1.32	Callable object	21
1.33	Context Manager - with statement	21
1.34	Using <code>@contextmanager</code>	22
1.35	Using with statement open file	22

1.36	Reading file chunk	22
1.37	Property - Managed attributes	23
1.38	Computed attributes - Using property	24
1.39	Descriptor - manage attributes	24
1.40	@staticmethod, @classmethod	25
1.41	Abstract method - Metaclass	25
1.42	Common Use Magic	26
1.43	Parsing csv string	27
1.44	Using <code>__slots__</code> to save memory	27
1.45	Using annotation to check type	28
2	New in Python3 cheatsheet	31
2.1	<code>print</code> is a function	32
2.2	String is unicode	32
2.3	Division Operator	33
2.4	New dict implementation	34
2.5	Keyword-Only Arguments	34
2.6	New Super	35
2.7	Remove <code><></code>	35
2.8	Not allow from module <code>import *</code> inside function	36
2.9	Add <code>nonlocal</code> keyword	36
2.10	Extended iterable unpacking	36
2.11	General unpacking	37
2.12	Function annotations	37
2.13	Variable annotations	38
2.14	Core support for typing module and generic types	38
2.15	Format byte string	39
2.16	<code>fstring</code>	39
2.17	Suppressing exception	40
2.18	Generator delegation	40
2.19	<code>async</code> and <code>await</code> syntax	41
2.20	Asynchronous generators	42
2.21	Asynchronous comprehensions	42
2.22	Matrix multiplication	43
2.23	Data Classes	44
2.24	Built-in <code>breakpoint()</code>	45
3	Python typing cheatsheet	47
3.1	Without type check	48
3.2	With type check	48
3.3	Basic types	49
3.4	Functions	50
3.5	Classes	50
3.6	Generator	51
3.7	Asynchronous Generator	51
3.8	Context Manager	52
3.9	Asynchronous Contents Manager	52
3.10	Avoid <code>None</code> access	52
3.11	Positional-only arguments	53
3.12	Multiple return values	53
3.13	<code>Union[Any, None] == Optional[Any]</code>	54
3.14	Be careful of <code>Optional</code>	54
3.15	Be careful of casting	55
3.16	Forward references	55

3.17	Postponed Evaluation of Annotations	56
3.18	Type alias	57
3.19	Define a <code>NewType</code>	57
3.20	Using <code>TypeVar</code> as template	58
3.21	Using <code>TypeVar</code> and <code>Generic</code> as class template	59
3.22	Scoping rules for <code>TypeVar</code>	60
3.23	Restricting to a fixed set of possible types	61
3.24	<code>TypeVar</code> with an upper bound	62
3.25	<code>@overload</code>	64
3.26	Stub Files	66
4	Python unicode cheatsheet	67
4.1	Encode: unicode code point to bytes	67
4.2	Decode: bytes to unicode code point	68
4.3	Get unicode code point	68
4.4	python2 <code>str</code> is equivalent to byte string	68
4.5	python3 <code>str</code> is equivalent to unicode string	68
4.6	python2 take <code>str</code> char as byte character	69
4.7	python3 take <code>str</code> char as unicode character	69
4.8	unicode normalization	69
4.9	Avoid <code>UnicodeDecodeError</code>	70
5	Python generator cheatsheet	71
5.1	Glossary of Generator	72
5.2	Produce value via generator	73
5.3	Unpacking Generators	73
5.4	Implement Iterable object via generator	74
5.5	Send message to generator	74
5.6	<code>yield from</code> expression	75
5.7	<code>yield (from) EXPR</code> return <code>RES</code>	76
5.8	Generate sequences	76
5.9	What <code>RES = yield from EXP</code> actually do?	77
5.10	<code>for _ in gen()</code> simulate <code>yield from</code>	78
5.11	Check generator type	78
5.12	Check Generator State	78
5.13	Simple compiler	79
5.14	Context manager and generator	81
5.15	What <code>@contextmanager</code> actually doing?	81
5.16	profile code block	82
5.17	<code>yield from</code> and <code>__iter__</code>	83
5.18	<code>yield from == await</code> expression	83
5.19	Closure in Python - using generator	84
5.20	Implement a simple scheduler	85
5.21	Simple round-robin with blocking	86
5.22	simple round-robin with blocking and non-blocking	87
5.23	Asynchronous Generators	89
5.24	Asynchronous generators can have <code>try..finally</code> blocks	89
5.25	send value and throw exception into async generator	89
5.26	Simple async round-robin	90
5.27	Async generator get better performance than async iterator	91
5.28	Asynchronous Comprehensions	91
6	Python Regular Expression cheatsheet	95
6.1	Compare HTML tags	96

6.2	<code>re.findall()</code> match string	96
6.3	Group Comparison	97
6.4	Non capturing group	97
6.5	Back Reference	97
6.6	Named Grouping (<code>?P<name></code>)	98
6.7	Substitute String	98
6.8	Look around	99
6.9	Match common username or password	99
6.10	Match hex color value	99
6.11	Match email	99
6.12	Match URL	100
6.13	Match IP address	100
6.14	Match Mac address	100
6.15	Lexer	101
7	Python socket cheatsheet	103
7.1	Get Hostname	104
7.2	Get address family and socket address from string	104
7.3	Transform Host & Network Endian	105
7.4	IP dotted-quad string & byte format convert	106
7.5	Mac address & byte format convert	106
7.6	Simple TCP Echo Server	106
7.7	Simple TCP Echo Server through IPv6	107
7.8	Disable IPv6 Only	107
7.9	Simple TCP Echo Server Via SocketServer	108
7.10	Simple TLS/SSL TCP Echo Server	109
7.11	Set ciphers on TLS/SSL TCP Echo Server	110
7.12	Simple UDP Echo Server	111
7.13	Simple UDP Echo Server Via SocketServer	111
7.14	Simple UDP client - Sender	112
7.15	Broadcast UDP Packets	112
7.16	Simple UNIX Domain Socket	112
7.17	Simple duplex processes communication	113
7.18	Simple Asynchronous TCP Server - Thread	114
7.19	Simple Asynchronous TCP Server - select	114
7.20	Simple Asynchronous TCP Server - poll	115
7.21	Simple Asynchronous TCP Server - epoll	117
7.22	Simple Asynchronous TCP Server - kqueue	119
7.23	High-Level API - selectors	122
7.24	Simple Non-blocking TLS/SSL socket via selectors	123
7.25	“socketpair” - Similar to PIPE	125
7.26	Using sendfile do copy	126
7.27	Sending a file through sendfile	126
7.28	Linux kernel Crypto API - AF_ALG	128
7.29	AES-CBC encrypt/decrypt via AF_ALG	129
7.30	AES-GCM encrypt/decrypt via AF_ALG	130
7.31	AES-GCM encrypt/decrypt file with sendfile	132
7.32	Compare the performance of AF_ALG to cryptography	134
7.33	Sniffer IP packets	136
7.34	Sniffer TCP packet	138
7.35	Sniffer ARP packet	141
8	Python security cheatsheet	143
8.1	Simple https server	144

8.2	Generate a SSH key pair	144
8.3	Get certificate information	145
8.4	Generate a self-signed certificate	146
8.5	Prepare a Certificate Signing Request (csr)	147
8.6	Generate RSA keyfile without passphrase	149
8.7	Sign a file by a given private key	149
8.8	Verify a file from a signed digest	150
8.9	Simple RSA encrypt via pem file	151
8.10	Simple RSA encrypt via RSA module	151
8.11	Simple RSA decrypt via pem file	152
8.12	Simple RSA encrypt with OAEP	153
8.13	Simple RSA decrypt with OAEP	154
8.14	Using DSA to proof of identity	154
8.15	Using AES CBC mode encrypt a file	155
8.16	Using AES CBC mode decrypt a file	156
8.17	AES CBC mode encrypt via password (using cryptography)	157
8.18	AES CBC mode decrypt via password (using cryptography)	159
8.19	AES CBC mode encrypt via password (using pycrypto)	160
8.20	AES CBC mode decrypt via password (using pycrypto)	162
8.21	Ephemeral Diffie Hellman Key Exchange via cryptography	163
8.22	Calculate DH shared key manually via cryptography	163
8.23	Calculate DH shared key from (p, g, pubkey)	164
9	Python Concurrency Cheatsheet	165
9.1	Execute a shell command	166
9.2	Create a thread via “threading”	166
9.3	Performance Problem - GIL	167
9.4	Consumer and Producer	167
9.5	Thread Pool Template	168
9.6	Using multiprocessing ThreadPool	169
9.7	Mutex lock	169
9.8	Deadlock	170
9.9	Implement “Monitor”	171
9.10	Control primitive resources	171
9.11	Ensure tasks has done	172
9.12	Thread-safe priority queue	173
9.13	Multiprocessing	174
9.14	Custom multiprocessing map	175
9.15	Graceful way to kill all child processes	175
9.16	Simple round-robin scheduler	176
9.17	Scheduler with blocking function	176
9.18	PoolExecutor	178
9.19	How to use ThreadPoolExecutor?	179
9.20	What “with ThreadPoolExecutor” doing?	179
9.21	Future Object	180
9.22	Future error handling	181
10	Python SQLAlchemy Cheatsheet	183
10.1	Set a database URL	184
10.2	Sqlalchemy Support DBAPI - PEP249	184
10.3	Transaction and Connect Object	185
10.4	Metadata - Generating Database Schema	185
10.5	Inspect - Get Database Information	186
10.6	Reflection - Loading Table from Existing Database	186

10.7	Get Table from MetaData	186
10.8	Create all Tables Store in “MetaData”	187
10.9	Create Specific Table	187
10.10	Create table with same columns	188
10.11	Drop a Table	188
10.12	Some Table Object Operation	189
10.13	SQL Expression Language	190
10.14	insert() - Create an “INSERT” Statement	190
10.15	select() - Create a “SELECT” Statement	191
10.16	join() - Joined Two Tables via “JOIN” Statement	192
10.17	Delete Rows from Table	193
10.18	Check Table Existing	193
10.19	Create multiple tables at once	194
10.20	Create tables with dynamic columns (Table)	195
10.21	Object Relational add data	195
10.22	Object Relational update data	196
10.23	Object Relational delete row	197
10.24	Object Relational relationship	198
10.25	Object Relational self association	199
10.26	Object Relational basic query	200
10.27	mapper: Map Table to class	204
10.28	Get table dynamically	205
10.29	Object Relational join two tables	206
10.30	join on relationship and group_by count	207
10.31	Create tables with dynamic columns (ORM)	209
10.32	Close database connection	209
10.33	Cannot use the object after close the session	210
11	Python asyncio cheatsheet	213
11.1	What is @asyncio.coroutine?	214
11.2	What is a Task?	215
11.3	What event loop doing? (Without polling)	216
11.4	What asyncio.wait doing?	217
11.5	Future like object	218
11.6	Future like object __await__ other task	219
11.7	Patch loop runner _run_once	219
11.8	Put blocking task into Executor	220
11.9	asyncio.run	220
11.10	Simple asyncio.run	221
11.11	Socket with asyncio	221
11.12	Event Loop with polling	222
11.13	Transport and Protocol	224
11.14	Transport and Protocol with SSL	224
11.15	What loop.create_server do?	225
11.16	Inline callback	226
11.17	Asynchronous Iterator	227
11.18	What is asynchronous iterator	228
11.19	Asynchronous context manager	228
11.20	What is asynchronous context manager	229
11.21	decorator @asynccontextmanager	229
11.22	What loop.sock_* do?	230
11.23	Simple asyncio connection pool	232
11.24	Get domain name	234
11.25	Simple asyncio UDP echo server	234

11.26 Simple asyncio Web server	236
11.27 Simple HTTPS Web Server	236
11.28 Simple HTTPS Web server (low-level api)	237
11.29 TLS Upgrade	239
11.30 Using sendfile	240
11.31 Simple asyncio WSGI web server	241
12 Python test cheatsheet	245
12.1 A simple Python unittest	246
12.2 Python unittest setup & teardown hierarchy	246
12.3 Different module of setUp & tearDown hierarchy	247
12.4 Run tests via unittest.TextTestRunner	249
12.5 Test raise exception	249
12.6 Pass arguments into a TestCase	250
12.7 Group multiple testcases into a suite	250
12.8 Group multiple tests from different TestCase	251
12.9 Skip some tests in the TestCase	251
12.10 Monolithic Test	252
12.11 Cross-module variables to Test files	252
12.12 skip setup & teardown when the test is skipped	253
12.13 Re-using old test code	254
12.14 Testing your document is right	254
12.15 Re-using doctest to unittest	255
12.16 Customize test report	256
12.17 Mock - using @patch substitute original method	258
12.18 What with unittest.mock.patch do?	259
12.19 Mock - substitute open	260
13 Python C Extensions cheatsheet	261
13.1 Simple setup.py for c extension	262
13.2 Customize CFLAGS	262
13.3 Doc string	262
13.4 Simple C Extension	263
13.5 Get Reference Count	263
13.6 Parse Arguments	264
13.7 Calling Python Functions from C	265
13.8 Raise Exception	266
13.9 Customize Exception	267
13.10 Iterate a List	268
13.11 Iterate a Dictionary	269
13.12 Simple Class	270
13.13 Simple Class with Members and Methods	271
13.14 Simple Class with Getter and Setter	274
13.15 Inherit from Other Class	277
13.16 Run a Python command from C	281
13.17 Run a Python file from C	281
13.18 Import a Python Module from C	282
13.19 Import All Attributes of a Module	284
13.20 Access Attributes	285
13.21 Performance of C Extension	286
13.22 Performance of ctypes	287
13.23 Error handling when using ctypes	288

Welcome to pysheet. This project aims at collecting useful python snippets in order to enhance pythoneers' coding experiences. Any snippets are welcome. If you'd like to contribute, [fork pysheet on GitHub](#). If there is any question or suggestion, please create an issue on [GitHub Issues](#).

Table of Contents

- *Python basic cheatsheet*
 - *Python Naming Rule*
 - *Using `__future__` backport features*
 - *Check python version*
 - *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*
- *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
```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```
>>> 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 python version

```
>>> import sys
>>> print(sys.version)
3.6.5 (default, Apr 24 2018, 10:37:34)
[GCC 4.2.1 Compatible Apple LLVM 7.3.0 (clang-703.0.31)]
>>> py36 = sys.version_info >= (3, 6)
>>> py36
True
>>> py37 = sys.version_info >= (3, 7)
>>> py37
False
```

1.4 Check object attributes

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

1.5 Define a function `__doc__`

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

(continues on next page)

(continued from previous page)

```
>>> example.__doc__
' This is an example function '

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

1.6 Check instance type

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

1.7 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'
```

1.8 Check inheritance

```
>>> class Example(object):
...     def __init__(self):
...         self.name = "ex"
...     def printex(self):
```

(continues on next page)

(continued from previous page)

```
...     print("This is an Example")
...
>>> isinstance(Example, object)
True
```

1.9 Check all global variables

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

1.10 Check callable

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

1.11 Get function/class name

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

1.12 __new__ & __init__

```
# __init__ will be invoked
>>> class ClassA(object):
...     def __new__(cls, arg):
...         print('__new__ ' + arg)
...         return object.__new__(cls, arg)
...     def __init__(self, arg):
```

(continues on next page)

(continued from previous page)

```

...         print('__init__ ' + arg)
...
>>> o = ClassA("Hello")
__new__ Hello
__init__ Hello

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

```

1.13 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

```

1.14 Representations of your class behave

```

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

```

(continues on next page)

(continued from previous page)

```
>>> print(str(Example()))
Example __str__
>>> Example()
Example __repr__
```

1.15 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'
```

1.16 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]
```

(continues on next page)

(continued from previous page)

```

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

# Get index and item in loop
>>> for i, v in enumerate(range(3)):
...     print((i, v))
...
(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'], [], []]

```

1.17 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}

# merge two dictionaries (python 3.4 or lower)
>>> a = {"x": 55, "y": 66}
>>> b = {"a": "foo", "b": "bar"}
>>> c = a.copy()
>>> c.update(b)
>>> c
{'y': 66, 'x': 55, 'b': 'bar', 'a': 'foo'}

# merge two dictionaries (python 3.5 or above)
>>> a = {"x": 55, "y": 66}
>>> b = {"a": "foo", "b": "bar"}
>>> c = {**a, **b}
>>> c
{'x': 55, 'y': 66, 'a': 'foo', 'b': 'bar'}
```

1.18 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']
```

(continues on next page)

(continued from previous page)

```

# 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]

```

1.19 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])

# unify 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])

```

(continues on next page)

(continued from previous page)

```
>>> 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])
```

1.20 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
```


1.21 `__iter__` - Delegating Iteration

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

1.22 Using Generator as Iterator

```
# see: PEP289
>>> for x in g:
...     print(x, end=' ')
... else:
...     print()
...
0 1 2 3 4 5 6 7 8 9

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

1.23 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):
```

(continues on next page)

(continued from previous page)

```

...     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

```

1.24 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}

```

(continues on next page)

(continued from previous page)

```

>>> 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[_], end=' ') # __iter__
... else:
...     print()
...
1 3 5
>>> '1' in emud # __contains__
True

```

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():
```

(continues on next page)

(continued from previous page)

```
...     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 x in range(5):
...     print(x, end=' ')
... else:
...     print("\nno break occurred")
...
0 1 2 3 4
no break occurred
>>> for x in range(5):
...     if x % 2 == 0:
...         print("break occurred")
...         break
...     else:
...         print("no break occurred")
...
break occurred

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

(continues on next page)

(continued from previous page)

```
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)
... 
```

(continues on next page)

(continued from previous page)

```

>>> 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)

```

(continues on next page)

(continued from previous page)

```
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
... 
```

(continues on next page)

(continued from previous page)

```
>>> 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 isinstance(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 isinstance(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):
```

(continues on next page)

(continued from previous page)

```
...     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)
```

(continues on next page)

(continued from previous page)

```

__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):

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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"))

```

(continues on next page)

(continued from previous page)

```
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'>'
```


Table of Contents

- *New in Python3 cheatsheet*
 - *print is a function*
 - *String is unicode*
 - *Division Operator*
 - *New dict implementation*
 - *Keyword-Only Arguments*
 - *New Super*
 - *Remove <>*
 - *Not allow from module import * inside function*
 - *Add nonlocal keyword*
 - *Extended iterable unpacking*
 - *General unpacking*
 - *Function annotations*
 - *Variable annotations*
 - *Core support for typing module and generic types*
 - *Format byte string*
 - *fstring*
 - *Suppressing exception*
 - *Generator delegation*

- *async and await syntax*
- *Asynchronous generators*
- *Asynchronous comprehensions*
- *Matrix multiplication*
- *Data Classes*
- *Built-in breakpoint()*

2.1 print is a function

New in Python 3.0

- PEP 3105 - Make print a function

Python 2

```
>>> print "print is a statement"
print is a statement
>>> for x in range(3):
...     print x,
...
0 1 2
```

Python 3

```
>>> print("print is a function")
print is a function
>>> print()
>>> for x in range(3):
...     print(x, end=' ')
... else:
...     print()
...
0 1 2
```

2.2 String is unicode

New in Python 3.0

- PEP 3138 - String representation in Python 3000
- PEP 3120 - Using UTF-8 as the default source encoding
- PEP 3131 - Supporting Non-ASCII Identifiers

Python 2

```
>>> s = 'Café' # byte string
>>> s
'Caf\xca3\xa9'
>>> type(s)
<type 'str'>
```

(continues on next page)

(continued from previous page)

```
>>> u = u'Caf ' # unicode string
>>> u
u'Caf xe9'
>>> type(u)
<type 'unicode'>
>>> len([_c for _c in 'Caf '])
5
```

Python 3

```
>>> s = 'Caf '
>>> s
'Caf '
>>> type(s)
<class 'str'>
>>> s.encode('utf-8')
b'Caf xc3 xa9'
>>> s.encode('utf-8').decode('utf-8')
'Caf '
>>> len([_c for _c in 'Caf '])
4
```

2.3 Division Operator

New in Python 3.0

- [PEP 238](#) - Changing the Division Operator

Python2

```
>>> 1 / 2
0
>>> 1 // 2
0
>>> 1. / 2
0.5

# back port "true division" to python2

>>> from __future__ import division
>>> 1 / 2
0.5
>>> 1 // 2
0
```

Python3

```
>>> 1 / 2
0.5
>>> 1 // 2
0
```

2.4 New dict implementation

New in Python 3.6

- PEP 468 - Preserving the order of `**kwargs` in a function
- PEP 520 - Preserving Class Attribute Definition Order
- bpo 27350 - More compact dictionaries with faster iteration

Before Python 3.5

```
>>> import sys
>>> sys.getsizeof({str(i):i for i in range(1000)})
49248

>>> d = {'timmy': 'red', 'barry': 'green', 'guido': 'blue'}
>>> d      # without order-preserving
{'barry': 'green', 'timmy': 'red', 'guido': 'blue'}
```

Python 3.6

- Memory usage is smaller than Python 3.5
- Preserve insertion ordered

```
>>> import sys
>>> sys.getsizeof({str(i):i for i in range(1000)})
36968

>>> d = {'timmy': 'red', 'barry': 'green', 'guido': 'blue'}
>>> d      # preserve insertion ordered
{'timmy': 'red', 'barry': 'green', 'guido': 'blue'}
```

2.5 Keyword-Only Arguments

New in Python 3.0

- PEP 3102 - Keyword-Only Arguments

```
>>> def f(a, b, *, kw):
...     print(a, b, kw)
...
>>> f(1, 2, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: f() takes 2 positional arguments but 3 were given
>>> f(1, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: f() missing 1 required keyword-only argument: 'kw'
>>> f(1, 2, kw=3)
1 2 3
```

2.6 New Super

New in Python 3.0

- PEP 3135 - New Super

Python 2

```
>>> class ParentCls(object):
...     def foo(self):
...         print "call parent"
...
>>> class ChildCls(ParentCls):
...     def foo(self):
...         super(ChildCls, self).foo()
...         print "call child"
...
>>> p = ParentCls()
>>> c = ChildCls()
>>> p.foo()
call parent
>>> c.foo()
call parent
call child
```

Python 3

```
>>> class ParentCls(object):
...     def foo(self):
...         print("call parent")
...
>>> class ChildCls(ParentCls):
...     def foo(self):
...         super().foo()
...         print("call child")
...
>>> p = ParentCls()
>>> c = ChildCls()
>>> p.foo()
call parent
>>> c.foo()
call parent
call child
```

2.7 Remove <>

New in Python 3.0

Python 2

```
>>> a = "Python2"
>>> a <> "Python3"
True

# equal to !=
```

(continues on next page)

(continued from previous page)

```
>>> a != "Python3"
True
```

Python 3

```
>>> a = "Python3"
>>> a != "Python2"
True
```

2.8 Not allow from module import * inside function

New in Python 3.0

```
>>> def f():
...     from os import *
...
File "<stdin>", line 1
SyntaxError: import * only allowed at module level
```

2.9 Add nonlocal keyword

New in Python 3.0

PEP 3104 - Access to Names in Outer Scopes

Note: nonlocal allow assigning directly to a variable in an outer (but non-global) scope

```
>>> def outf():
...     o = "out"
...     def inf():
...         nonlocal o
...         o = "change out"
...     inf()
...     print(o)
...
>>> outf()
change out
```

2.10 Extended iterable unpacking

New in Python 3.0

- PEP 3132 - Extended Iterable Unpacking

```
>>> a, *b, c = range(5)
>>> a, b, c
(0, [1, 2, 3], 4)
>>> for a, *b in [(1, 2, 3), (4, 5, 6, 7)]:
```

(continues on next page)

(continued from previous page)

```
...     print(a, b)
...
1 [2, 3]
4 [5, 6, 7]
```

2.11 General unpacking

New in Python 3.5

- [PEP 448](#) - Additional Unpacking Generalizations

Python 2

```
>>> def func(*a, **k):
...     print(a)
...     print(k)
...
>>> func(*[1,2,3,4,5], **{"foo": "bar"})
(1, 2, 3, 4, 5)
{'foo': 'bar'}
```

Python 3

```
>>> print(*[1, 2, 3], 4, *[5, 6])
1 2 3 4 5 6
>>> [*range(4), 4]
[0, 1, 2, 3, 4]
>>> {"foo": "Foo", "bar": "Bar", **{"baz": "baz"}}
{'foo': 'Foo', 'bar': 'Bar', 'baz': 'baz'}
>>> def func(*a, **k):
...     print(a)
...     print(k)
...
>>> func(*[1], *[4,5], **{"foo": "FOO"}, **{"bar": "BAR"})
(1, 4, 5)
{'foo': 'FOO', 'bar': 'BAR'}
```

2.12 Function annotations

New in Python 3.0

- [PEP 3107](#) - Function Annotations
- [PEP 484](#) - Type Hints
- [PEP 483](#) - The Theory of Type Hints

```
>>> import types
>>> generator = types.GeneratorType
>>> def fib(n: int) -> generator:
...     a, b = 0, 1
...     for _ in range(n):
...         yield a
```

(continues on next page)

(continued from previous page)

```
...         b, a = a + b, b
...
>>> [f for f in fib(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

2.13 Variable annotations

New in Python 3.6

- [PEP 526](#) - Syntax for Variable Annotations

```
>>> from typing import List
>>> x: List[int] = [1, 2, 3]
>>> x
[1, 2, 3]

>>> from typing import List, Dict
>>> class Cls(object):
...     x: List[int] = [1, 2, 3]
...     y: Dict[str, str] = {"foo": "bar"}
...
>>> o = Cls()
>>> o.x
[1, 2, 3]
>>> o.y
{'foo': 'bar'}
```

2.14 Core support for typing module and generic types

New in Python 3.7

- [PEP 560](#) - Core support for typing module and generic types

Before Python 3.7

```
>>> from typing import Generic, TypeVar
>>> from typing import Iterable
>>> T = TypeVar('T')
>>> class C(Generic[T]): ...
...
>>> def func(l: Iterable[C[int]]) -> None:
...     for i in l:
...         print(i)
...
>>> func([1,2,3])
1
2
3
```

Python 3.7 or above


```

>>> from typing import Iterable
>>> class C:
...     def __class_getitem__(cls, item):
...         return f"{cls.__name__}[{item.__name__}]"
...
>>> def func(l: Iterable[C[int]]) -> None:
...     for i in l:
...         print(i)
...
>>> func([1,2,3])
1
2
3

```

2.15 Format byte string

New in Python 3.5

- PEP 461 - Adding % formatting to bytes and bytearray

```

>>> b'abc %b %b' % (b'foo', b'bar')
b'abc foo bar'
>>> b'%d %f' % (1, 3.14)
b'1 3.140000'
>>> class Cls(object):
...     def __repr__(self):
...         return "repr"
...     def __str__(self):
...         return "str"
...
'repr'
>>> b'%a' % Cls()
b'repr'

```

2.16 fstring

New in Python 3.6

- PEP 498 - Literal String Interpolation

```

>>> py = "Python3"
>>> f'Awesome {py}'
'Awesome Python3'
>>> x = [1, 2, 3, 4, 5]
>>> f'{x}'
'[1, 2, 3, 4, 5]'
>>> def foo(x:int) -> int:
...     return x + 1
...
>>> f'{foo(0)}'
'1'
>>> f'{123.567:1.3}'
'1.24e+02'

```

2.17 Suppressing exception

New in Python 3.3

- PEP 409 - Suppressing exception context

Without raise Exception from None

```
>>> def func():
...     try:
...         1 / 0
...     except ZeroDivisionError:
...         raise ArithmeticError
...
>>> func()
Traceback (most recent call last):
  File "<stdin>", line 3, in func
ZeroDivisionError: division by zero

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in func
ArithmeticError
```

With raise Exception from None

```
>>> def func():
...     try:
...         1 / 0
...     except ZeroDivisionError:
...         raise ArithmeticError from None
...
>>> func()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in func
ArithmeticError

# debug

>>> try:
...     func()
... except ArithmeticError as e:
...     print(e.__context__)
...
division by zero
```

2.18 Generator delegation

New in Python 3.3

- PEP 380 - Syntax for Delegating to a Subgenerator

```
>>> def fib(n: int):
...     a, b = 0, 1
...     for _ in range(n):
...         yield a
...         b, a = a + b, b
...
>>> def delegate(n: int):
...     yield from fib(n)
...
>>> list(delegate(10))
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

2.19 async and await syntax

New in Python 3.5

- [PEP 492](#) - Coroutines with async and await syntax

Before Python 3.5

```
>>> import asyncio
>>> @asyncio.coroutine
... def fib(n: int):
...     a, b = 0, 1
...     for _ in range(n):
...         b, a = a + b, b
...     return a
...
>>> @asyncio.coroutine
... def coro(n: int):
...     for x in range(n):
...         yield from asyncio.sleep(1)
...         f = yield from fib(x)
...         print(f)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(coro(3))
0
1
1
```

Python 3.5 or above

```
>>> import asyncio
>>> async def fib(n: int):
...     a, b = 0, 1
...     for _ in range(n):
...         b, a = a + b, b
...     return a
...
>>> async def coro(n: int):
...     for x in range(n):
...         await asyncio.sleep(1)
...         f = await fib(x)
...         print(f)
```

(continues on next page)

(continued from previous page)

```
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(coro(3))
0
1
1
```

2.20 Asynchronous generators

New in Python 3.6

- [PEP 525](#) - Asynchronous Generators

```
>>> import asyncio
>>> async def fib(n: int):
...     a, b = 0, 1
...     for _ in range(n):
...         await asyncio.sleep(1)
...         yield a
...         b, a = a + b, b
...
>>> async def coro(n: int):
...     ag = fib(n)
...     f = await ag.asend(None)
...     print(f)
...     f = await ag.asend(None)
...     print(f)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(coro(5))
0
1
```

2.21 Asynchronous comprehensions

New in Python 3.6

- [PEP 530](#) - Asynchronous Comprehensions

```
>>> import asyncio
>>> async def fib(n: int):
...     a, b = 0, 1
...     for _ in range(n):
...         await asyncio.sleep(1)
...         yield a
...         b, a = a + b, b
...
# async for ... else

>>> async def coro(n: int):
...     async for f in fib(n):
```

(continues on next page)

(continued from previous page)

```

...     print(f, end=" ")
...     else:
...         print()
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(coro(5))
0 1 1 2 3

# async for in list

>>> async def coro(n: int):
...     return [f async for f in fib(n)]
...
>>> loop.run_until_complete(coro(5))
[0, 1, 1, 2, 3]

# await in list

>>> async def slowfmt(n: int) -> str:
...     await asyncio.sleep(0.5)
...     return f'{n}'
...
>>> async def coro(n: int):
...     return [await slowfmt(f) async for f in fib(n)]
...
>>> loop.run_until_complete(coro(5))
['0', '1', '1', '2', '3']

```

2.22 Matrix multiplication

New in Python 3.5

- [PEP 465](#) - A dedicated infix operator for matrix multiplication

```

>>> # "@" represent matrix multiplication
>>> 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)

```

(continues on next page)

(continued from previous page)

```
...     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])
```

2.23 Data Classes

New in Python 3.7

PEP 557 - Data Classes

Mutable Data Class

```
>>> from dataclasses import dataclass
>>> @dataclass
... class DCls(object):
...     x: str
...     y: str
...
>>> d = DCls("foo", "bar")
>>> d
DCls(x='foo', y='bar')
>>> d = DCls(x="foo", y="baz")
>>> d
DCls(x='foo', y='baz')
>>> d.z = "bar"
```

Immutable Data Class

```
>>> from dataclasses import dataclass
>>> from dataclasses import FrozenInstanceError
>>> @dataclass(frozen=True)
... class DCls(object):
...     x: str
...     y: str
...
>>> try:
...     d.x = "baz"
... except FrozenInstanceError as e:
...     print(e)
...
cannot assign to field 'x'
>>> try:
...     d.z = "baz"
... except FrozenInstanceError as e:
...     print(e)
```

(continues on next page)

(continued from previous page)

```
...
cannot assign to field 'z'
```

2.24 Built-in breakpoint ()

New in Python 3.7

- [PEP 553](#) - Built-in breakpoint()

```
>>> for x in range(3):
...     print(x)
...     breakpoint()
...
0
> <stdin>(1) <module>()->None
(Pdb) c
1
> <stdin>(1) <module>()->None
(Pdb) c
2
> <stdin>(1) <module>()->None
(Pdb) c
```


Table of Contents

- *Python typing cheatsheet*
 - *Without type check*
 - *With type check*
 - *Basic types*
 - *Functions*
 - *Classes*
 - *Generator*
 - *Asynchronous Generator*
 - *Context Manager*
 - *Asynchronous Contents Manager*
 - *Avoid None access*
 - *Positional-only arguments*
 - *Multiple return values*
 - *Union[Any, None] == Optional[Any]*
 - *Be careful of Optional*
 - *Be careful of casting*
 - *Forward references*
 - *Postponed Evaluation of Annotations*
 - *Type alias*

- *Define a NewType*
- *Using TypeVar as template*
- *Using TypeVar and Generic as class template*
- *Scoping rules for TypeVar*
- *Restricting to a fixed set of possible types*
- *TypeVar with an upper bound*
- *@overload*
- *Stub Files*

3.1 Without type check

```
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        b, a = a + b, b

print([n for n in fib(3.6)])
```

output:

```
# errors will not be detected until runtime

$ python fib.py
Traceback (most recent call last):
  File "fib.py", line 8, in <module>
    print([n for n in fib(3.5)])
  File "fib.py", line 8, in <listcomp>
    print([n for n in fib(3.5)])
  File "fib.py", line 3, in fib
    for _ in range(n):
TypeError: 'float' object cannot be interpreted as an integer
```

3.2 With type check

```
# give a type hint
from typing import Generator

def fib(n: int) -> Generator:
    a: int = 0
    b: int = 1
    for _ in range(n):
        yield a
        b, a = a + b, b

print([n for n in fib(3.6)])
```

output:

```
# errors will be detected before running

$ mypy --strict fib.py
fib.py:12: error: Argument 1 to "fib" has incompatible type "float"; expected "int"
```

3.3 Basic types

```
import io
import re

from collections import deque, namedtuple
from typing import (
    Dict,
    List,
    Tuple,
    Set,
    Deque,
    NamedTuple,
    IO,
    Pattern,
    Match,
    Text,
    Optional,
    Sequence,
    Iterable,
    Mapping,
    MutableMapping,
    Any,
)

# without initializing
x: int

# any type
y: Any
y = 1
y = "1"

# built-in
var_int: int = 1
var_str: str = "Hello Typing"
var_byte: bytes = b"Hello Typing"
var_bool: bool = True
var_float: float = 1.
var_unicode: Text = u'\u2713'

# could be none
var_could_be_none: Optional[int] = None
var_could_be_none = 1

# collections
var_set: Set[int] = {i for i in range(3)}
var_dict: Dict[str, str] = {"foo": "Foo"}
var_list: List[int] = [i for i in range(3)]
var_Tuple: Tuple = (1, 2, 3)
```

(continues on next page)

(continued from previous page)

```

var_deque: Deque = deque([1, 2, 3])
var_namedtuple: NamedTuple = namedtuple('P', ['x', 'y'])

# io
var_io_str: IO[str] = io.StringIO("Hello String")
var_io_byte: IO[bytes] = io.BytesIO(b"Hello Bytes")
var_io_file_str: IO[str] = open(__file__)
var_io_file_byte: IO[bytes] = open(__file__, 'rb')

# re
p: Pattern = re.compile("(https?)://([^\r\n]+)(/[^\r\n]*)?")
m: Optional[Match] = p.match("https://www.python.org/")

# duck types: list-like
var_seq_list: Sequence[int] = [1, 2, 3]
var_seq_tuple: Sequence[int] = (1, 2, 3)
var_iter_list: Iterable[int] = [1, 2, 3]
var_iter_tuple: Iterable[int] = (1, 2, 3)

# duck types: dict-like
var_map_dict: Mapping[str, str] = {"foo": "Foo"}
var_mutable_dict: MutableMapping[str, str] = {"bar": "Bar"}

```

3.4 Functions

```

from typing import Generator, Callable

# function
def gcd(a: int, b: int) -> int:
    while b:
        a, b = b, a % b
    return a

# callback
def fun(cb: Callable[[int, int], int]) -> int:
    return cb(55, 66)

# lambda
f: Callable[[int], int] = lambda x: x * 2

```

3.5 Classes

```

from typing import ClassVar, Dict, List

class Foo:

    x: int = 1 # instance variable. default = 1
    y: ClassVar[str] = "class var" # class variable

    def __init__(self) -> None:
        self.i: List[int] = [0]

```

(continues on next page)

(continued from previous page)

```

    def foo(self, a: int, b: str) -> Dict[int, str]:
        return {a: b}

foo = Foo()
foo.x = 123

print(foo.x)
print(foo.i)
print(foo.y)
print(foo.foo(1, "abc"))

```

3.6 Generator

```

from typing import Generator

# Generator[YieldType, SendType, ReturnType]
def fib(n: int) -> Generator[int, None, None]:
    a: int = 0
    b: int = 1
    while n > 0:
        yield a
        b, a = a + b, b
        n -= 1

g: Generator = fib(10)
i: Iterator[int] = (x for x in range(3))

```

3.7 Asynchronous Generator

```

import asyncio

from typing import AsyncGenerator, AsyncIterator

async def fib(n: int) -> AsyncGenerator:
    a: int = 0
    b: int = 1
    while n > 0:
        await asyncio.sleep(0.1)
        yield a

        b, a = a + b, b
        n -= 1

async def main() -> None:
    async for f in fib(10):
        print(f)

    ag: AsyncIterator = (f async for f in fib(10))

```

(continues on next page)

(continued from previous page)

```
loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

3.8 Context Manager

```
from typing import ContextManager, Generator, IO
from contextlib import contextmanager

@contextmanager
def open_file(name: str) -> Generator:
    f = open(name)
    yield f
    f.close()

cm: ContextManager[IO] = open_file(__file__)
with cm as f:
    print(f.read())
```

3.9 Asynchronous Contents Manager

```
import asyncio

from typing import AsyncContextManager, AsyncGenerator, IO
from contextlib import asynccontextmanager

# need python 3.7 or above
@asynccontextmanager
async def open_file(name: str) -> AsyncGenerator:
    await asyncio.sleep(0.1)
    f = open(name)
    yield f
    await asyncio.sleep(0.1)
    f.close()

async def main() -> None:
    acm: AsyncContextManager[IO] = open_file(__file__)
    async with acm as f:
        print(f.read())

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

3.10 Avoid None access

```
import re

from typing import Pattern, Dict, Optional
```

(continues on next page)

(continued from previous page)

```
# like c++
# std::regex url("https?://([^\r\n]+)(/[^\r\n]*)?");
# std::regex color("^#?([a-f0-9]{6}|[a-f0-9]{3})$");

url: Pattern = re.compile("https?://([^\r\n]+)(/[^\r\n]*)?")
color: Pattern = re.compile("^#?([a-f0-9]{6}|[a-f0-9]{3})$")

x: Dict[str, Pattern] = {"url": url, "color": color}
y: Optional[Pattern] = x.get("baz", None)

print(y.match("https://www.python.org/"))
```

output:

```
$ mypy --strict foo.py
foo.py:15: error: Item "None" of "Optional[Pattern[Any]]" has no attribute "match"
```

3.11 Positional-only arguments

```
# define arguments with names beginning with __

def fib(__n: int) -> int: # positional only arg
    a, b = 0, 1
    for _ in range(__n):
        b, a = a + b, b
    return a

def gcd(*, a: int, b: int) -> int: # keyword only arg
    while b:
        a, b = b, a % b
    return a

print(fib(__n=10)) # error
print(gcd(10, 5)) # error
```

output:

```
mypy --strict foo.py
foo.py:1: note: "fib" defined here
foo.py:14: error: Unexpected keyword argument "__n" for "fib"
foo.py:15: error: Too many positional arguments for "gcd"
```

3.12 Multiple return values

```
from typing import Tuple, Iterable, Union

def foo(x: int, y: int) -> Tuple[int, int]:
    return x, y
```

(continues on next page)

(continued from previous page)

```
# or

def bar(x: int, y: str) -> Iterable[Union[int, str]]:
    # XXX: not recommend declaring in this way
    return x, y

a: int
b: int
a, b = foo(1, 2)      # ok
c, d = bar(3, "bar")  # ok
```

3.13 Union[Any, None] == Optional[Any]

```
from typing import List, Union

def first(l: List[Union[int, None]]) -> Union[int, None]:
    return None if len(l) == 0 else l[0]

first([None])

# equal to

from typing import List, Optional

def first(l: List[Optional[int]]) -> Optional[int]:
    return None if len(l) == 0 else l[0]

first([None])
```

3.14 Be careful of Optional

```
from typing import cast, Optional

def fib(n):
    a, b = 0, 1
    for _ in range(n):
        b, a = a + b, b
    return a

def cal(n: Optional[int]) -> None:
    print(fib(n))

cal(None)
```

output:

```
# mypy will not detect errors
$ mypy foo.py
```

Explicitly declare


```

from typing import Optional

def fib(n: int) -> int: # declare n to be int
    a, b = 0, 1
    for _ in range(n):
        b, a = a + b, b
    return a

def cal(n: Optional[int]) -> None:
    print(fib(n))

```

output:

```

# mypy can detect errors even we do not check None
$ mypy --strict foo.py
foo.py:11: error: Argument 1 to "fib" has incompatible type "Optional[int]"; expected
↳ "int"

```

3.15 Be careful of casting

```

from typing import cast, Optional

def gcd(a: int, b: int) -> int:
    while b:
        a, b = b, a % b
    return a

def cal(a: Optional[int], b: Optional[int]) -> None:
    # XXX: Avoid casting
    ca, cb = cast(int, a), cast(int, b)
    print(gcd(ca, cb))

cal(None, None)

```

output:

```

# mypy will not detect type errors
$ mypy --strict foo.py

```

3.16 Forward references

Based on PEP 484, if we want to reference a type before it has been declared, we have to use **string literal** to imply that there is a type of that name later on in the file.

```

from typing import Optional

class Tree:
    def __init__(
        self, data: int,
        left: Optional["Tree"], # Forward references.
        right: Optional["Tree"]
    ):

```

(continues on next page)

(continued from previous page)

```
) -> None:
    self.data = data
    self.left = left
    self.right = right
```

Note: There are some issues that mypy does not complain about Forward References. Get further information from [Issue#948](#).

```
class A:
    def __init__(self, a: A) -> None: # should fail
        self.a = a
```

output:

```
$ mypy --strict type.py
$ echo $?
0
$ python type.py # get runtime fail
Traceback (most recent call last):
  File "type.py", line 1, in <module>
    class A:
  File "type.py", line 2, in A
    def __init__(self, a: A) -> None: # should fail
NameError: name 'A' is not defined
```

3.17 Postponed Evaluation of Annotations

New in Python 3.7

- [PEP 563](#) - Postponed Evaluation of Annotations

Before Python 3.7

```
>>> class A:
...     def __init__(self, a: A) -> None:
...         self._a = a
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in A
NameError: name 'A' is not defined
```

After Python 3.7 (include 3.7)

```
>>> from __future__ import annotations
>>> class A:
...     def __init__(self, a: A) -> None:
...         self._a = a
...
```

Note: Annotation can only be used within the scope which names have already existed. Therefore, **forward reference** does not support the case which names are not available in the current scope. **Postponed evaluation of annotations**

will become the default behavior in Python 4.0.

3.18 Type alias

Like `typedef` or using in `c/c++`

```
#include <iostream>
#include <string>
#include <regex>
#include <vector>

typedef std::string Url;
template<typename T> using Vector = std::vector<T>;

int main(int argc, char *argv[])
{
    Url url = "https://python.org";
    std::regex p("(https?):\\/([^\r\n]+)\\/([^\r\n]*)?");
    bool m = std::regex_match(url, p);
    Vector<int> v = {1, 2};

    std::cout << m << std::endl;
    for (auto it : v) std::cout << it << std::endl;
    return 0;
}
```

Type aliases are defined by simple variable assignments

```
import re

from typing import Pattern, List

# Like typedef, using in c/c++
# PEP 484 recommend capitalizing alias names
Url = str

url: Url = "https://www.python.org/"

p: Pattern = re.compile("(https?):\\/([^\r\n]+)\\/([^\r\n]*)?")
m = p.match(url)

Vector = List[int]
v: Vector = [1., 2.]
```

3.19 Define a NewType

Unlike alias, `NewType` returns a separate type but is identical to the original type at runtime.

```
from sqlalchemy import Column, String, Integer
from sqlalchemy.ext.declarative import declarative_base
from typing import NewType, Any
```

(continues on next page)

(continued from previous page)

```
# check mypy #2477
Base: Any = declarative_base()

# create a new type
Id = NewType('Id', int) # not equal alias, it's a 'new type'

class User(Base):
    __tablename__ = 'User'
    id = Column(Integer, primary_key=True)
    age = Column(Integer, nullable=False)
    name = Column(String, nullable=False)

    def __init__(self, id: Id, age: int, name: str) -> None:
        self.id = id
        self.age = age
        self.name = name

# create users
user1 = User(Id(1), 62, "Guido van Rossum") # ok
user2 = User(2, 48, "David M. Beazley")      # error
```

output:

```
$ python foo.py
$ mypy --ignore-missing-imports foo.py
foo.py:24: error: Argument 1 to "User" has incompatible type "int"; expected "Id"
```

Further reading:

- [Issue#1284](#)

3.20 Using TypeVar as template

Like c++ template <typename T>

```
#include <iostream>

template <typename T>
T add(T x, T y) {
    return x + y;
}

int main(int argc, char *argv[])
{
    std::cout << add(1, 2) << std::endl;
    std::cout << add(1., 2.) << std::endl;
    return 0;
}
```

Python using TypeVar

```
from typing import TypeVar

T = TypeVar("T")
```

(continues on next page)

(continued from previous page)

```
def add(x: T, y: T) -> T:
    return x + y

add(1, 2)
add(1., 2.)
```

3.21 Using `TypeVar` and `Generic` as class template

Like `c++` `template <typename T> class`

```
#include <iostream>

template<typename T>
class Foo {
public:
    Foo(T foo) {
        foo_ = foo;
    }
    T Get() {
        return foo_;
    }
private:
    T foo_;
};

int main(int argc, char *argv[])
{
    Foo<int> f(123);
    std::cout << f.Get() << std::endl;
    return 0;
}
```

Define a generic class in Python

```
from typing import Generic, TypeVar

T = TypeVar("T")

class Foo(Generic[T]):
    def __init__(self, foo: T) -> None:
        self.foo = foo

    def get(self) -> T:
        return self.foo

f: Foo[str] = Foo("Foo")
v: int = f.get()
```

output:

```
$ mypy --strict foo.py
foo.py:13: error: Incompatible types in assignment (expression has type "str",
↳ variable has type "int")
```

3.22 Scoping rules for TypeVar

- TypeVar used in different generic function will be inferred to be different types.

```
from typing import TypeVar

T = TypeVar("T")

def foo(x: T) -> T:
    return x

def bar(y: T) -> T:
    return y

a: int = foo(1)      # ok: T is inferred to be int
b: int = bar("2")    # error: T is inferred to be str
```

output:

```
$ mypy --strict foo.py
foo.py:12: error: Incompatible types in assignment (expression has type "str",
↳ variable has type "int")
```

- TypeVar used in a generic class will be inferred to be same types.

```
from typing import TypeVar, Generic

T = TypeVar("T")

class Foo(Generic[T]):

    def foo(self, x: T) -> T:
        return x

    def bar(self, y: T) -> T:
        return y

f: Foo[int] = Foo()
a: int = f.foo(1)      # ok: T is inferred to be int
b: str = f.bar("2")    # error: T is expected to be int
```

output:

```
$ mypy --strict foo.py
foo.py:15: error: Incompatible types in assignment (expression has type "int",
↳ variable has type "str")
foo.py:15: error: Argument 1 to "bar" of "Foo" has incompatible type "str"; expected
↳ "int"
```

- TypeVar used in a method but did not match any parameters which declare in Generic can be inferred to be different types.

```
from typing import TypeVar, Generic

T = TypeVar("T")
S = TypeVar("S")
```

(continues on next page)

(continued from previous page)

```

class Foo(Generic[T]):    # S does not match params

    def foo(self, x: T, y: S) -> S:
        return y

    def bar(self, z: S) -> S:
        return z

f: Foo[int] = Foo()
a: str = f.foo(1, "foo")  # S is inferred to be str
b: int = f.bar(12345678)  # S is inferred to be int

```

output:

```
$ mypy --strict foo.py
```

- TypeVar should not appear in body of method/function if it is unbound type.

```

from typing import TypeVar, Generic

T = TypeVar("T")
S = TypeVar("S")

def foo(x: T) -> None:
    a: T = x      # ok
    b: S = 123    # error: invalid type

```

output:

```
$ mypy --strict foo.py
foo.py:8: error: Invalid type "foo.S"
```

3.23 Restricting to a fixed set of possible types

`T = TypeVar('T', ClassA, ...)` means we create a **type variable with a value restriction**.

```

from typing import TypeVar

# restrict T = int or T = float
T = TypeVar("T", int, float)

def add(x: T, y: T) -> T:
    return x + y

add(1, 2)
add(1., 2.)
add("1", 2)
add("hello", "world")

```

output:

```

# mypy can detect wrong type
$ mypy --strict foo.py

```

(continues on next page)

(continued from previous page)

```
foo.py:10: error: Value of type variable "T" of "add" cannot be "object"
foo.py:11: error: Value of type variable "T" of "add" cannot be "str"
```

3.24 TypeVar with an upper bound

`T = TypeVar('T', bound=BaseClass)` means we create a **type variable with an upper bound**. The concept is similar to **polymorphism** in c++.

```
#include <iostream>

class Shape {
public:
    Shape(double width, double height) {
        width_ = width;
        height_ = height;
    };
    virtual double Area() = 0;
protected:
    double width_;
    double height_;
};

class Rectangle: public Shape {
public:
    Rectangle(double width, double height)
    :Shape(width, height)
    {};

    double Area() {
        return width_ * height_;
    };
};

class Triangle: public Shape {
public:
    Triangle(double width, double height)
    :Shape(width, height)
    {};

    double Area() {
        return width_ * height_ / 2;
    };
};

double Area(Shape &s) {
    return s.Area();
}

int main(int argc, char *argv[])
{
    Rectangle r(1., 2.);
    Triangle t(3., 4.);

    std::cout << Area(r) << std::endl;
```

(continues on next page)

(continued from previous page)

```
std::cout << Area(t) << std::endl;
return 0;
}
```

Like c++, create a base class and `TypeVar` which bounds to the base class. Then, static type checker will take every subclass as type of base class.

```
from typing import TypeVar

class Shape:
    def __init__(self, width: float, height: float) -> None:
        self.width = width
        self.height = height

    def area(self) -> float:
        return 0

class Rectangle(Shape):
    def area(self) -> float:
        width: float = self.width
        height: float = self.height
        return width * height

class Triangle(Shape):
    def area(self) -> float:
        width: float = self.width
        height: float = self.height
        return width * height / 2

S = TypeVar("S", bound=Shape)

def area(s: S) -> float:
    return s.area()

r: Rectangle = Rectangle(1, 2)
t: Triangle = Triangle(3, 4)
i: int = 5566

print(area(r))
print(area(t))
print(area(i))
```

output:

```
$ mypy --strict foo.py
foo.py:40: error: Value of type variable "S" of "area" cannot be "int"
```

3.25 @overload

Sometimes, we use `Union` to infer that the return of a function has multiple different types. However, type checker cannot distinguish which type do we want. Therefore, following snippet shows that type checker cannot determine which type is correct.

```
from typing import List, Union

class Array(object):
    def __init__(self, arr: List[int]) -> None:
        self.arr = arr

    def __getitem__(self, i: Union[int, str]) -> Union[int, str]:
        if isinstance(i, int):
            return self.arr[i]
        if isinstance(i, str):
            return str(self.arr[int(i)])

arr = Array([1, 2, 3, 4, 5])
x:int = arr[1]
y:str = arr["2"]
```

output:

```
$ mypy --strict foo.py
foo.py:16: error: Incompatible types in assignment (expression has type "Union[int, ↵
↵str]", variable has type "int")
foo.py:17: error: Incompatible types in assignment (expression has type "Union[int, ↵
↵str]", variable has type "str")
```

Although we can use `cast` to solve the problem, it cannot avoid typo and `cast` is not safe.

```
from typing import List, Union, cast

class Array(object):
    def __init__(self, arr: List[int]) -> None:
        self.arr = arr

    def __getitem__(self, i: Union[int, str]) -> Union[int, str]:
        if isinstance(i, int):
            return self.arr[i]
        if isinstance(i, str):
            return str(self.arr[int(i)])

arr = Array([1, 2, 3, 4, 5])
x: int = cast(int, arr[1])
y: str = cast(str, arr[2]) # typo. we want to assign arr["2"]
```

output:

```
$ mypy --strict foo.py
$ echo ??
0
```

Using `@overload` can solve the problem. We can declare the return type explicitly.

```
from typing import Generic, List, Union, overload

class Array(object):
    def __init__(self, arr: List[int]) -> None:
        self.arr = arr

    @overload
    def __getitem__(self, i: str) -> str:
        ...

    @overload
    def __getitem__(self, i: int) -> int:
        ...

    def __getitem__(self, i: Union[int, str]) -> Union[int, str]:
        if isinstance(i, int):
            return self.arr[i]
        if isinstance(i, str):
            return str(self.arr[int(i)])

arr = Array([1, 2, 3, 4, 5])
x: int = arr[1]
y: str = arr["2"]
```

output:

```
$ mypy --strict foo.py
$ echo $?
0
```

Warning: Based on PEP 484, the `@overload` decorator just **for type checker only**, it does not implement the real overloading like c++/java. Thus, we have to implement one exactly non-`@overload` function. At the runtime, calling the `@overload` function will raise `NotImplementedError`.

```
from typing import List, Union, overload

class Array(object):
    def __init__(self, arr: List[int]) -> None:
        self.arr = arr

    @overload
    def __getitem__(self, i: Union[int, str]) -> Union[int, str]:
        if isinstance(i, int):
            return self.arr[i]
        if isinstance(i, str):
            return str(self.arr[int(i)])

arr = Array([1, 2, 3, 4, 5])
try:
```

(continues on next page)

(continued from previous page)

```
x: int = arr[1]
except NotImplementedError as e:
    print("NotImplementedError")
```

output:

```
$ python foo.py
NotImplementedError
```

3.26 Stub Files

Stub files just like header files which we usually use to define our interfaces in c/c++. In python, we can define our interfaces in the same module directory or `export MYPYPATH=${stubs}`

First, we need to create a stub file (interface file) for module.

```
$ mkdir fib
$ touch fib/__init__.py fib/__init__.pyi
```

Then, define the interface of the function in `__init__.pyi` and implement the module.

```
# fib/__init__.pyi
def fib(n: int) -> int: ...

# fib/__init__.py
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        b, a = a + b, b
    return a
```

Then, write a test.py for testing fib module.

```
# touch test.py
import sys

from pathlib import Path

p = Path(__file__).parent / "fib"
sys.path.append(str(p))

from fib import fib

print(fib(10.0))
```

output:

```
$ mypy --strict test.py
test.py:10: error: Argument 1 to "fib" has incompatible type "float"; expected "int"
```

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*
 - *Avoid UnicodeDecodeError*

4.1 Encode: unicode code point to bytes

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

4.2 Decode: bytes to unicode code point

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

4.3 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
```

4.4 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'>
```

4.5 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'
```

4.6 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
```

4.7 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
```

4.8 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\xc3\x01'
>>> 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
```

(continues on next page)

(continued from previous page)

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

4.9 Avoid UnicodeDecodeError

```
# raise a UnicodeDecodeError

>>> u = b"0xff"
>>> u.decode('utf-8')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 0: invalid start_
↳byte

# raise a UnicodeDecodeError

>>> u.decode('utf-8', "strict")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 0: invalid start_
↳byte

# use U+FFFD, REPLACEMENT CHARACTER

>>> u.decode('utf-8', "replace")
'\ufffd'

# inserts a \xNN escape sequence

>>> u.decode('utf-8', "backslashreplace")
'\\xff'

# leave the character out of the Unicode result

>>> u.decode('utf-8', "ignore")
''
```

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*

5.1 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>
```

(continues on next page)

(continued from previous page)

StopIteration

5.2 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 >>>
```

5.3 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])
```

(continues on next page)

(continued from previous page)

```
>>> {*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
```

5.4 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 >>>
```

5.5 Send message to generator

```
>>> def spam():
...     msg = yield
```

(continues on next page)

(continued from previous page)

```

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

```

5.6 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'

```

5.7 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
```

5.8 Generate sequences

```
# get a list via generator

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

(continues on next page)

(continued from previous page)

```

...
>>> 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]

```

5.9 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

```

(continues on next page)

(continued from previous page)

```
>>> next(g)
1
```

5.10 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
```

5.11 Check generator type

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

5.12 Check Generator State

```
>>> import inspect
>>> def gen_func():
...     yield 9527
...
>>> g = gen_func()
>>> inspect.getgeneratorstate(g)
```

(continues on next page)

(continued from previous page)

```
'GEN_CREATED'
>>> next(g)
9527
>>> inspect.getgeneratorstate(g)
'GEN_SUSPENDED'
>>> g.close()
>>> inspect.getgeneratorstate(g)
'GEN_CLOSED'
```

5.13 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)
```

(continues on next page)

(continued from previous page)

```

        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

```

(continues on next page)

(continued from previous page)

```

        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

```

5.14 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

```

5.15 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):

```

(continues on next page)

(continued from previous page)

```

    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

```

5.16 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)

```

(continues on next page)

(continued from previous page)

```
...
block2 cost time: 3.00104284286
```

5.17 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
```

5.18 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)
```

(continues on next page)

(continued from previous page)

```

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

```

5.19 Closure in Python - using generator

```

# nonlocal version
>>> def closure():
...     x = 5566

```

(continues on next page)

(continued from previous page)

```

...     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

```

5.20 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):

```

(continues on next page)

(continued from previous page)

```

...     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

```

5.21 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':

```

(continues on next page)

(continued from previous page)

```

        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()

```

5.22 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:

```

(continues on next page)

(continued from previous page)

```

        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()

```

5.23 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
```

5.24 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',)
```

5.25 send value and throw exception into async generator

```
# Need python-3.6 or above

>>> import asyncio
```

(continues on next page)

(continued from previous page)

```

>>> 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',)

```

5.26 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

```

5.27 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
```

5.28 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)]
```

(continues on next page)

(continued from previous page)

```

...     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)]

```

(continues on next page)

(continued from previous page)

```
...     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'}
```

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*

6.1 Compare HTML tags

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

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

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

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

6.2 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"
```

6.3 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'
```

6.4 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', '/')
```

6.5 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
```

(continues on next page)

(continued from previous page)

```
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
```

6.6 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>
```

6.7 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'
```

6.8 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'
```

6.9 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
```

6.10 Match hex color value

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

6.11 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>
```

6.12 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')
```

6.13 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]?|0[0-9]?)\.){3}
...                 (?:25[0-5]
...                 |2[0-4][0-9]
...                 |1[0-9][0-9]?|0[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
```

6.14 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))
```

(continues on next page)

(continued from previous page)

```
>>> 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
```

6.15 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')
```


Table of Contents

- *Python socket cheatsheet*
 - *Get Hostname*
 - *Get address family and socket address from string*
 - *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*

7.1 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'
```

7.2 Get address family and socket address from string

```
import socket
import sys

try:
    for res in socket.getaddrinfo(sys.argv[1], None,
                                   proto=socket.IPPROTO_TCP):
        family = res[0]
        sockaddr = res[4]
        print(family, sockaddr)
```

(continues on next page)

(continued from previous page)

```
except socket.gaierror:
    print("Invalid")
```

Output:

```
$ gai.py 192.0.2.244
AddressFamily.AF_INET ('192.0.2.244', 0)
$ gai.py 2001:db8:f00d::1:d
AddressFamily.AF_INET6 ('2001:db8:f00d::1:d', 0, 0, 0)
$ gai.py www.google.com
AddressFamily.AF_INET6 ('2607:f8b0:4006:818::2004', 0, 0, 0)
AddressFamily.AF_INET ('172.217.10.132', 0)
```

It handles unusual cases, valid and invalid:

```
$ gai.py 10.0.0.256 # octet overflow
Invalid
$ gai.py not-exist.example.com # unresolvable
Invalid
$ gai.py fe80::1%eth0 # scoped
AddressFamily.AF_INET6 ('fe80::1%eth0', 0, 0, 2)
$ gai.py ::ffff:192.0.2.128 # IPv4-Mapped
AddressFamily.AF_INET6 ('::ffff:192.0.2.128', 0, 0, 0)
$ gai.py 0xc000027b # IPv4 in hex
AddressFamily.AF_INET ('192.0.2.123', 0)
$ gai.py 3221226198 # IPv4 in decimal
AddressFamily.AF_INET ('192.0.2.214', 0)
```

7.3 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
```

7.4 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'
```

7.5 Mac address & byte format convert

```
>>> mac = '00:11:32:3c:c3:0b'
>>> byte = binascii.unhexlify(mac.replace(':', ''))
>>> byte
'\x00\x11<\xc3\x0b'
>>> binascii.hexlify(byte)
'0011323cc30b'
```

7.6 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
```

7.7 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
```

7.8 Disable IPv6 Only

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

import contextlib
import socket
```

(continues on next page)

(continued from previous page)

```

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

```

7.9 Simple TCP Echo Server Via SocketServer

```

>>> import SocketServer
>>> bh = SocketServer.BaseRequestHandler
>>> class handler(bh):

```

(continues on next page)

(continued from previous page)

```
...     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
```

7.10 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
```

7.11 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
```


7.12 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
```

7.13 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
```

7.14 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
```

7.15 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
```

7.16 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)
```

(continues on next page)

(continued from previous page)

```

        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

```

7.17 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
child pid: 9498
c[9498] ---> p[9497]: b'Hello Parent'
p[9497] ---> c[9498]: b'Hello Parent'
```

7.18 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
```

7.19 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:
```

(continues on next page)

(continued from previous page)

```

while True:
    r, w, _ = select(rl,wl,[])
    # process ready to read
    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

```

7.20 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))

```

(continues on next page)

(continued from previous page)

```

        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

```

(continues on next page)

(continued from previous page)

```

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

```

7.21 Simple Asynchronous TCP Server - epoll

```

from __future__ import print_function, unicode_literals

import socket
import select
import contextlib

host = 'localhost'
port = 5566

con = {}
req = {}

```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```

    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

```

7.22 Simple Asynchronous TCP Server - kqueue

```

from __future__ import print_function, unicode_literals

import socket

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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(),

```

(continues on next page)

(continued from previous page)

```

        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

```

7.23 High-Level API - selectors

```

# Python3.4+ 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")

```

(continues on next page)

(continued from previous page)

```

        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

```

7.24 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:

```

(continues on next page)

(continued from previous page)

```
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
```

7.25 “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!
```

(continues on next page)

(continued from previous page)

```
Hi! Parent!  
...
```

7.26 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
```

7.27 Sending a file through sendfile

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

(continues on next page)

(continued from previous page)

```

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()

```

(continues on next page)

(continued from previous page)

```

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

```

7.28 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())

```

(continues on next page)

(continued from previous page)

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

output:

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

7.29 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)
```

(continues on next page)

(continued from previous page)

```

    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'

```

7.30 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)

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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'

```

7.31 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)

```

(continues on next page)

(continued from previous page)

```

    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)

```

(continues on next page)

(continued from previous page)

```
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'
```

7.32 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
```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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]

```

7.33 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):

```

(continues on next page)

(continued from previous page)

```

''' IP header Structure

In linux api, it define as below:

struct 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,

```

(continues on next page)

(continued from previous page)

```

                                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

```

7.34 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|                                |
|                               |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;

```

(continues on next page)

(continued from previous page)

```

__be16  dest;
__be32  seq;
__be32  ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
__u16   resl: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,
        resl: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()

```

(continues on next page)

(continued from previous page)

```

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('!HLLBBHHH', 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}")

```

(continues on next page)

(continued from previous page)

```

        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'

```

7.35 Sniffer ARP packet

```

"""
Ethernet 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 */
};

```

(continues on next page)

(continued from previous page)

```

"""

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("-----")

```

output:

```

$ python arp.py
----- ETHERNET_FRAME -----
Dest MAC:      ffffffff
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
-----

```


Table of Contents

- *Python security cheatsheet*
 - *Simple https server*
 - *Generate a SSH key pair*
 - *Get 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 pycrypto)
- Ephemeral Diffie Hellman Key Exchange via cryptography
- Calculate DH shared key manually via cryptography
- Calculate DH shared key from (p, g, pubkey)

8.1 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()
```

8.2 Generate a SSH key pair

```
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.backends import default_backend

key = rsa.generate_private_key(
    backend=default_backend(),
    public_exponent=65537,
    key_size=2048
)
private_key = key.private_bytes(
    serialization.Encoding.PEM,
    serialization.PrivateFormat.PKCS8,
```

(continues on next page)

(continued from previous page)

```

        serialization.NoEncryption(),
    )
    public_key = key.public_key().public_bytes(
        serialization.Encoding.OpenSSH,
        serialization.PublicFormat.OpenSSH
    )

    with open('id_rsa', 'wb') as f, open('id_rsa.pub', 'wb') as g:
        f.write(private_key)
        g.write(public_key)

```

8.3 Get 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 \

```

(continues on next page)

(continued from previous page)

```
> -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
```

8.4 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 = "pysheet"
cert.get_subject().OU = "cheat sheet"
cert.get_subject().CN = "pythonsheets.com"
```

(continues on next page)

(continued from previous page)

```

cert.set_serial_number(1000)
cert.set_notBefore(now.strftime("%Y%m%d%H%M%S").encode())
cert.set_notAfter(expire.strftime("%Y%m%d%H%M%S").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=pysheet/OU=cheat sheet/CN=pythonsheets.com
issuer= /C=TW/ST=Taiwan/L=Taipei/O=pysheet/OU=cheat sheet/CN=pythonsheets.com

```

8.5 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.pythonsheets.com",
              b"DNS:doc.pythonsheets.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 = "pysheet"
req.get_subject().OU = "cheat sheet"
req.get_subject().CN = "pythonsheets.com"
req.add_extensions([

```

(continues on next page)

(continued from previous page)

```

    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=pysheet/OU=cheat sheet/CN=pythonsheets.com

```

(continues on next page)

(continued from previous page)

Getting CA Private Key

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

8.6 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()))
```

8.7 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:

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

(continues on next page)

(continued from previous page)

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

8.8 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:

```
# generate 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
```


8.9 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!
```

8.10 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)
```

(continues on next page)

(continued from previous page)

```
# 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!
```

8.11 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()
```

(continues on next page)

(continued from previous page)

```
# 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
```

8.12 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 | \
```

(continues on next page)

(continued from previous page)

```
> python3 rsa.py          |\
> openssl base64 -d -A    |\
> openssl rsautl -decrypt -oaep -inkey private.key
Hello RSA OAEP!
```

8.13 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
```

8.14 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()
```

(continues on next page)

(continued from previous page)

```

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!

```

8.15 Using AES CBC mode encrypt a file

```

from __future__ import print_function, unicode_literals

import struct
import sys
import os

```

(continues on next page)

(continued from previous page)

```

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

```

8.16 Using AES CBC mode decrypt a file

```

from __future__ import print_function, unicode_literals

```

(continues on next page)

(continued from previous page)

```

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

```

8.17 AES CBC mode encrypt via password (using cryptography)

```

from __future__ import print_function, unicode_literals

import base64
import struct
import sys

```

(continues on next page)

(continued from previous page)

```

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"

```

(continues on next page)

(continued from previous page)

```
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
```

8.18 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")
```

(continues on next page)

(continued from previous page)

```

# 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

```

8.19 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

```

(continues on next page)

(continued from previous page)

```

# 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

```

(continues on next page)

(continued from previous page)

```
$ 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
```

8.20 AES CBC mode decrypt 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

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]]
```

(continues on next page)

(continued from previous page)

```

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

```

8.21 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

```

8.22 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

```

8.23 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"
        "0256567934582260424238844463330887962689642467123")

g = 2

y = int("32155788395534640648739966373159697798396966919821525"
        "72238852825117261342483718574508213761865276905503199"
        "969908098203345481366464874759377454476688391248")

x = int("409364065449673443397833358558926598469347813468816037"
        "268451847116982490733450463194921405069999008617231539"
        "7147035896687401350877308899732826446337707128")

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
```

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*

- *How to use ThreadPoolExecutor?*
- *What “with ThreadPoolExecutor” doing?*
- *Future Object*
- *Future error handling*

9.1 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
```

9.2 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
```


9.3 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
```

9.4 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:
```

(continues on next page)

(continued from previous page)

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

9.5 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()
```

9.6 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
```

9.7 Mutex lock

Simplest synchronization primitive lock

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

(continues on next page)

(continued from previous page)

```
...     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
```

9.8 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
```

9.9 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
```

9.10 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)
```

(continues on next page)

(continued from previous page)

```
        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
```

9.11 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
```

9.12 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)
```

(continues on next page)

(continued from previous page)

```
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...
```

9.13 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
```


9.14 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)))
```

9.15 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)
```

9.16 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
```

9.17 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]):
```

(continues on next page)

(continued from previous page)

```

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 localhost 5566
20
6765

```

output: (bash 2)

```

$ nc localhost 5566
10
55

```

9.18 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
```

(continues on next page)

(continued from previous page)

```
>>> process()
9227465
9227465
process cost: 5.538189888000488
```

9.19 How to use ThreadPoolExecutor?

```
from concurrent.futures import ThreadPoolExecutor

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

with ThreadPoolExecutor(max_workers=3) as ex:
    futs = []
    for x in range(3):
        futs.append(ex.submit(fib, 30+x))

    res = [fut.result() for fut in futs]

print(res)
```

output:

```
$ python3 thread_pool_ex.py
[832040, 1346269, 2178309]
```

9.20 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
```

9.21 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
```

9.22 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
```


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*

10.1 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
```

10.2 Sqlalchemy Support DBAPI - PEP249

```
from sqlalchemy import create_engine

db_uri = "sqlite:///db.sqlite"
engine = create_engine(db_uri)
```

(continues on next page)

(continued from previous page)

```
# DBAPI - PEP249
# create table
engine.execute('CREATE TABLE "EX1" ('
                'id INTEGER NOT NULL,'
                'name VARCHAR, '
                'PRIMARY KEY (id));')

# insert a row
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())
```

10.3 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()
```

10.4 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
```

(continues on next page)

(continued from previous page)

```
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)
```

10.5 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'))
```

10.6 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)
```

10.7 Get Table from MetaData

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
```

(continues on next page)

(continued from previous page)

```

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)

```

10.8 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()

```

10.9 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,

```

(continues on next page)

(continued from previous page)

```
        Column('id', Integer, primary_key=True),
        Column('val', Integer))
t1.create()
```

10.10 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)
```

10.11 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',
```

(continues on next page)

(continued from previous page)

```

        '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

```

10.12 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)

```

10.13 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'))
# exhibit 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']))
```

10.14 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)
```

(continues on next page)

(continued from previous page)

```

# 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'}])

```

10.15 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_(

```

(continues on next page)

(continued from previous page)

```
        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)
```

10.16 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)
```

10.17 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)
```

10.18 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
```

(continues on next page)

(continued from previous page)

```
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)
```

10.19 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']
```

10.20 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

```

10.21 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

```

(continues on next page)

(continued from previous page)

```

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()

```

10.22 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()

```

(continues on next page)

(continued from previous page)

```

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

```

10.23 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

```

(continues on next page)

(continued from previous page)

```

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>
[]

```

10.24 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")

```

(continues on next page)

(continued from previous page)

```

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>

```

10.25 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,

```

(continues on next page)

(continued from previous page)

```

        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
  }
]

```

10.26 Object Relational basic query

```

from datetime import datetime

from sqlalchemy import create_engine

```

(continues on next page)

(continued from previous page)

```

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)

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```
ed Ed Jones
fred Fred Flinstone
```

10.27 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()
```

(continues on next page)

(continued from previous page)

```
# 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
```

10.28 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),
```

(continues on next page)

(continued from previous page)

```

        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

```

10.29 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',

```

(continues on next page)

(continued from previous page)

```

        '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

```

10.30 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)

```

(continues on next page)

(continued from previous page)

```

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

```

10.31 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

```

10.32 Close database connection

```

from sqlalchemy import (
    create_engine,
    event,
    Column,
    Integer)

from sqlalchemy.orm import sessionmaker

```

(continues on next page)

(continued from previous page)

```

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.

10.33 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)

```

(continues on next page)

(continued from previous page)

```
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
```


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*
 - *asyncio.run*
 - *Simple asyncio.run*
 - *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*
- *decorator @asynccontextmanager*
- *What loop.sock_* do?*
- *Simple asyncio connection pool*
- *Get domain name*
- *Simple asyncio UDP echo server*
- *Simple asyncio Web server*
- *Simple HTTPS Web Server*
- *Simple HTTPS Web server (low-level api)*
- *TLS Upgrade*
- *Using sendfile*
- *Simple asyncio WSGI web server*

11.1 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
```

11.2 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()
```

(continues on next page)

(continued from previous page)

```
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
```

11.3 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
```

(continues on next page)

(continued from previous page)

```

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

```

11.4 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)

```

(continues on next page)

(continued from previous page)

```

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

```

11.5 Future like object

```

>>> import sys
>>> PY_35 = sys.version_info >= (3, 5)
>>> import asyncio
>>> loop = asyncio.get_event_loop()
>>> class SlowObj:

```

(continues on next page)

(continued from previous page)

```

...     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

```

11.6 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

```

11.7 Patch loop runner `_run_once`

```

>>> import asyncio
>>> def _run_once(self):

```

(continues on next page)

(continued from previous page)

```

...     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()

```

11.8 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)

```

11.9 asyncio.run

New in Python 3.7

```

>>> import asyncio
>>> from concurrent.futures import ThreadPoolExecutor
>>> e = ThreadPoolExecutor()
>>> async def read_file(file_):
...     loop = asyncio.get_event_loop()
...     with open(file_) as f:
...         return (await loop.run_in_executor(e, f.read))
...
>>> ret = asyncio.run(read_file('/etc/passwd'))

```

11.10 Simple asyncio.run

```
>>> import asyncio
>>> async def getaddrinfo(host, port):
...     loop = asyncio.get_event_loop()
...     return (await loop.getaddrinfo(host, port))
...
>>> def run(main):
...     loop = asyncio.new_event_loop()
...     asyncio.set_event_loop(loop)
...     return loop.run_until_complete(main)
...
>>> ret = run(getaddrinfo('google.com', 443))
>>> ret = asyncio.run(getaddrinfo('google.com', 443))
```

11.11 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
```

11.12 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:
```

(continues on next page)

(continued from previous page)

```

        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()

```

11.13 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
```

11.14 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>"
```

(continues on next page)

(continued from previous page)

```

    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

```

11.15 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,

```

(continues on next page)

(continued from previous page)

```

        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

```

11.16 Inline callback

```

>>> import asyncio
>>> async def foo():
...     await asyncio.sleep(1)
...     return "foo done"
...

```

(continues on next page)

(continued from previous page)

```
>>> 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
```

11.17 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
```

11.18 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
```

11.19 Asynchronous context manager

```
# ref: PEP-0492
# need Python >= 3.5

>>> class AsyncCtxMgr:
...     async def __aenter__(self):
...         await asyncio.sleep(3)
...         print("__aenter__")
...         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")
...
```

(continues on next page)

(continued from previous page)

```
>>> t = loop.create_task(world())
>>> loop.run_until_complete(hello())
world block
__aenter__
hello block
__aexit__
```

11.20 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__
```

11.21 decorator @asynccontextmanager

New in Python 3.7

- Issue 29679 - Add @contextlib.asynccontextmanager

```
>>> import asyncio
>>> from contextlib import asynccontextmanager
>>> @asynccontextmanager
... async def coro(msg):
...     await asyncio.sleep(1)
...     yield msg
...     await asyncio.sleep(0.5)
...     print('done')
...
>>> async def main():
...     async with coro("Hello") as m:
...         await asyncio.sleep(1)
...         print(m)
...
>>> loop = asyncio.get_event_loop()
```

(continues on next page)

(continued from previous page)

```
>>> loop.run_until_complete(main())
Hello
done
```

11.22 What *loop.sock_** do?

```
import asyncio
import socket

def sock_accept(self, sock, fut=None, registered=False):
    fd = sock.fileno()
    if fut is None:
        fut = self.create_future()
    if registered:
        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, registered=False):
    fd = sock.fileno()
    if fut is None:
        fut = self.create_future()
    if registered:
        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, registered=False):
    fd = sock.fileno()
    if fut is None:
        fut = self.create_future()
    if registered:
        self.remove_writer(fd)
    try:
        n = sock.send(data)
    except (BlockingIOError, InterruptedError):
        n = 0
    except Exception as e:
        fut.set_exception(e)
```

(continues on next page)

(continued from previous page)

```

        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

```

11.23 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):
```

(continues on next page)

(continued from previous page)

```

    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()

```

(continues on next page)

(continued from previous page)

```
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'
```

11.24 Get domain name

```
>>> import asyncio
>>> async def getaddrinfo(host, port):
...     loop = asyncio.get_event_loop()
...     return (await loop.getaddrinfo(host, port))
...
>>> addrs = asyncio.run(getaddrinfo('github.com', 443))
>>> for a in addrs:
...     family, typ, proto, name, sockaddr = a
...     print(sockaddr)
...
('192.30.253.113', 443)
('192.30.253.113', 443)
('192.30.253.112', 443)
('192.30.253.112', 443)
```

11.25 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'
```

(continues on next page)

(continued from previous page)

```

port = 3553

sock.bind((host, port))

def recvfrom(loop, sock, n_bytes, fut=None, registered=False):
    fd = sock.fileno()
    if fut is None:
        fut = loop.create_future()
    if registered:
        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, registered=False):
    fd = sock.fileno()
    if fut is None:
        fut = loop.create_future()
    if registered:
        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

```

11.26 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
```

11.27 Simple HTTPS Web Server

```
import asyncio
import ssl
```

(continues on next page)

(continued from previous page)

```

ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
ctx.load_cert_chain('cert.pem', 'key.pem')

async def conn(reader, writer):
    _ = await reader.read(1024)
    head = b"HTTP/1.1 200 OK\r\n"
    head += b"Content-Type: text/html\r\n"
    head += b"\r\n"

    body = b"<!doctype html>"
    body += b"<html>"
    body += b"<body><h1>Awesome Python</h1></body>"
    body += b"</html>"

    writer.write(head + body)
    writer.close()

async def main(host, port):
    srv = await asyncio.start_server(conn, host, port, ssl=ctx)
    async with srv:
        await srv.serve_forever()

asyncio.run(main('0.0.0.0', 8000))

```

11.28 Simple HTTPS Web server (low-level api)

```

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')

```

(continues on next page)

(continued from previous page)

```

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)

```

(continues on next page)

(continued from previous page)

```

        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

```

11.29 TLS Upgrade

New in Python 3.7

```

import asyncio
import ssl

class HttpClient(asyncio.Protocol):
    def __init__(self, on_con_lost):
        self.on_con_lost = on_con_lost
        self.resp = b""

    def data_received(self, data):
        self.resp += data

    def connection_lost(self, exc):
        resp = self.resp.decode()
        print(resp.split("\r\n")[0])
        self.on_con_lost.set_result(True)

```

(continues on next page)

(continued from previous page)

```

async def main():
    paths = ssl.get_default_verify_paths()
    sslctx = ssl.SSLContext()
    sslctx.verify_mode = ssl.CERT_REQUIRED
    sslctx.check_hostname = True
    sslctx.load_verify_locations(paths.cafile)

    loop = asyncio.get_running_loop()
    on_con_lost = loop.create_future()

    tr, proto = await loop.create_connection(
        lambda: HttpClient(on_con_lost), "github.com", 443
    )
    new_tr = await loop.start_tls(tr, proto, sslctx)
    req = f"GET / HTTP/1.1\r\n"
    req += "Host: github.com\r\n"
    req += "Connection: close\r\n"
    req += "\r\n"
    new_tr.write(req.encode())

    await on_con_lost
    new_tr.close()

asyncio.run(main())

```

output:

```

$ python3 --version
Python 3.7.0
$ python3 https.py
HTTP/1.1 200 OK

```

11.30 Using sendfile

New in Python 3.7

```

import asyncio

path = "index.html"

async def conn(reader, writer):

    loop = asyncio.get_event_loop()
    _ = await reader.read(1024)

    with open(path, "rb") as f:
        tr = writer.transport
        head = b"HTTP/1.1 200 OK\r\n"
        head += b"Content-Type: text/html\r\n"
        head += b"\r\n"

        tr.write(head)
        await loop.sendfile(tr, f)

```

(continues on next page)

(continued from previous page)

```

        writer.close()

async def main(host, port):
    # run a simple http server
    srv = await asyncio.start_server(conn, host, port)
    async with srv:
        await srv.serve_forever()

asyncio.run(main("0.0.0.0", 8000))

```

output:

```

$ echo '<!doctype html><h1>Awesome Python</h1>' > index.html
$ python http.py &
[2] 60506
$ curl http://localhost:8000
<!doctype html><h1>Awesome Python</h1>

```

11.31 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
        """

```

(continues on next page)

(continued from previous page)

```

    # 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)

```

(continues on next page)

(continued from previous page)

```
        # 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
```


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*

12.1 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)
```

12.2 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")
```

(continues on next page)

(continued from previous page)

```

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

```

12.3 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")

```

(continues on next page)

(continued from previous page)

```

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

```

12.4 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
```

12.5 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)
```

12.6 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
```

12.7 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
```

(continues on next page)

(continued from previous page)

OK

12.8 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

12.9 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'
```

(continues on next page)

(continued from previous page)

```
Ran 4 tests in 0.000s
```

```
OK (skipped=3)
```

12.10 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>
```

12.11 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):
```

(continues on next page)

(continued from previous page)

```

    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)

```

12.12 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)

```

(continues on next page)

(continued from previous page)

```
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)
```

12.13 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>
```

12.14 Testing your document is right

```
"""
This is an example of doctest

>>> fib(10)
55
"""

def fib(n):
    """ This function calculate fib number.

Example:
```

(continues on next page)

(continued from previous page)

```

>>> 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.

```

12.15 Re-using doctest to unittest

```

import unittest
import doctest

"""
This is an example of doctest

>>> fib(10)
55
"""

```

(continues on next page)

(continued from previous page)

```
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
```

12.16 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)
```

(continues on next page)

(continued from previous page)

```

    # 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

```

(continues on next page)

(continued from previous page)

```
@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']}}
```

12.17 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('%$!?!&*')
...     except OSError as e:
...         print(e)
...     else:
...         print('test success')
...
>>> test()
[Errno 2] No such file or directory: '%$!?!&*'

```

12.18 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')

```

(continues on next page)

(continued from previous page)

```
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: '$@!%?&'
```

12.19 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
```

Table of Contents

- *Python C Extensions cheatsheet*
 - *Simple setup.py for c extension*
 - *Customize CFLAGS*
 - *Doc string*
 - *Simple C Extension*
 - *Get Reference Count*
 - *Parse Arguments*
 - *Calling Python Functions from C*
 - *Raise Exception*
 - *Customize Exception*
 - *Iterate a List*
 - *Iterate a Dictionary*
 - *Simple Class*
 - *Simple Class with Members and Methods*
 - *Simple Class with Getter and Setter*
 - *Inherit from Other Class*
 - *Run a Python command from C*
 - *Run a Python file from C*
 - *Import a Python Module from C*

- *Import All Attributes of a Module*
- *Access Attributes*
- *Performance of C Extension*
- *Performance of ctypes*
- *Error handling when using ctypes*

13.1 Simple setup.py for c extension

```
from distutils.core import setup, Extension

ext = Extension('foo', sources=['foo.c'])
setup(name="Foo", version="1.0", ext_modules=[ext])
```

13.2 Customize CFLAGS

```
import sysconfig
from distutils.core import setup, Extension

cflags = sysconfig.get_config_var("CFLAGS")

extra_compile_args = cflags.split()
extra_compile_args += ["-Wextra"]

ext = Extension(
    "foo", ["foo.c"],
    extra_compile_args=extra_compile_args
)

setup(name="foo", version="1.0", ext_modules=[ext])
```

13.3 Doc string

```
PyDoc_STRVAR(doc_mod, "Module document\n");
PyDoc_STRVAR(doc_foo, "foo() -> None\n\nFoo doc");

static PyMethodDef methods[] = {
    {"foo", (PyCFunction)foo, METH_NOARGS, doc_foo},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    .m_base = PyModuleDef_HEAD_INIT,
    .m_name = "Foo",
    .m_doc = doc_mod,
    .m_size = -1,
    .m_methods = methods
};
```


13.4 Simple C Extension

foo.c

```
#include <Python.h>

PyDoc_STRVAR(doc_mod, "Module document\n");
PyDoc_STRVAR(doc_foo, "foo() -> None\n\nFoo doc");

static PyObject* foo(PyObject* self)
{
    PyObject* s = PyUnicode_FromString("foo");
    PyObject_Print(s, stdout, 0);
    Py_RETURN_NONE;
}

static PyMethodDef methods[] = {
    {"foo", (PyCFunction)foo, METH_NOARGS, doc_foo},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "Foo", doc_mod, -1, methods
};

PyMODINIT_FUNC PyInit_foo(void)
{
    return PyModule_Create(&module);
}
```

output:

```
$ python setup.py -q build
$ python setup.py -q install
$ python -c "import foo; foo.foo()"
'foo'
```

13.5 Get Reference Count

```
#include <Python.h>

static PyObject *
getrefcount(PyObject *self, PyObject *a)
{
    return PyLong_FromSsize_t(Py_REFCNT(a));
}

static PyMethodDef methods[] = {
    {"getrefcount", (PyCFunction)getrefcount, METH_O, NULL},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, methods
}
```

(continues on next page)

(continued from previous page)

```
};

PyMODINIT_FUNC PyInit_foo(void)
{
    return PyModule_Create(&module);
}
```

output:

```
$ python setup.py -q build
$ python setup.py -q install
$ python -q
>>> import sys
>>> import foo
>>> l = [1, 2, 3]
>>> sys.getrefcount(l[0])
104
>>> foo.getrefcount(l[0])
104
>>> i = l[0]
>>> sys.getrefcount(l[0])
105
>>> foo.getrefcount(l[0])
105
```

13.6 Parse Arguments

```
#include <Python.h>

static PyObject *
foo(PyObject *self)
{
    Py_RETURN_NONE;
}

static PyObject *
bar(PyObject *self, PyObject *arg)
{
    return Py_BuildValue("O", arg);
}

static PyObject *
baz(PyObject *self, PyObject *args)
{
    PyObject *x = NULL, *y = NULL;
    if (!PyArg_ParseTuple(args, "OO", &x, &y)) {
        return NULL;
    }
    return Py_BuildValue("OO", x, y);
}

static PyObject *
qux(PyObject *self, PyObject *args, PyObject *kwargs)
{

```

(continues on next page)

(continued from previous page)

```

static char *keywords[] = {"x", "y", NULL};
PyObject *x = NULL, *y = NULL;
if (!PyArg_ParseTupleAndKeywords(args, kwargs,
                                "OO", keywords,
                                &x, &y))
{
    return NULL;
}
if (!y) {
    y = Py_None;
}
return Py_BuildValue("OO", x, y);
}

static PyMethodDef methods[] = {
    {"foo", (PyCFunction)foo, METH_NOARGS, NULL},
    {"bar", (PyCFunction)bar, METH_O, NULL},
    {"baz", (PyCFunction)baz, METH_VARARGS, NULL},
    {"qux", (PyCFunction)qux, METH_VARARGS | METH_KEYWORDS, NULL},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, methods
};

PyMODINIT_FUNC PyInit_foo(void)
{
    return PyModule_Create(&module);
}

```

output:

```

$ python setup.py -q build
$ python setup.py -q install
$ python -q
>>> import foo
>>> foo.foo()
>>> foo.bar(3.7)
3.7
>>> foo.baz(3, 7)
(3, 7)
>>> foo.qux(3, y=7)
(3, 7)
>>> foo.qux(x=3, y=7)
(3, 7)
>>> foo.qux(x=3)
(3, None)

```

13.7 Calling Python Functions from C

```

#include <Python.h>

static PyObject *

```

(continues on next page)

(continued from previous page)

```

foo(PyObject *self, PyObject *args)
{
    PyObject *py_callback = NULL;
    PyObject *rv = NULL;

    if (!PyArg_ParseTuple(args, "O:callback", &py_callback))
        return NULL;

    if (!PyCallable_Check(py_callback)) {
        PyErr_SetString(PyExc_TypeError, "should be callable");
        return NULL;
    }

    // similar to py_callback("Awesome Python!")
    rv = PyObject_CallFunction(py_callback, "s", "Awesome Python!");
    return rv;
}

static PyMethodDef methods[] = {
    {"foo", (PyCFunction)foo, METH_VARARGS, NULL},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, methods
};

PyMODINIT_FUNC PyInit_foo(void)
{
    return PyModule_Create(&module);
}

```

output:

```

$ python setup.py -q build
$ python setup.py -q install
$ python -c "import foo; foo.foo(print)"
Awesome Python!

```

13.8 Raise Exception

```

#include <Python.h>

PyDoc_STRVAR(doc_mod, "Module document\n");
PyDoc_STRVAR(doc_foo, "foo() -> None\n\nFoo doc");

static PyObject*
foo(PyObject* self)
{
    // raise NotImplementedError
    PyErr_SetString(PyExc_NotImplementedError, "Not implemented");
    return NULL;
}

```

(continues on next page)

(continued from previous page)

```
static PyMethodDef methods[] = {
    {"foo", (PyCFunction)foo, METH_NOARGS, doc_foo},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "Foo", doc_mod, -1, methods
};

PyMODINIT_FUNC PyInit_foo(void)
{
    return PyModule_Create(&module);
}
```

output:

```
$ python setup.py -q build
$ python setup.py -q install
$ python -c "import foo; foo.foo(print)"
$ python -c "import foo; foo.foo()"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
NotImplementedError: Not implemented
```

13.9 Customize Exception

```
#include <stdio.h>
#include <Python.h>

static PyObject *FooError;

PyDoc_STRVAR(doc_foo, "foo() -> void\n\n"
    "Equal to the following example:\n\n"
    "def foo():\n"
    "    raise FooError(\"Raise exception in C\")"
);

static PyObject *
foo(PyObject *self __attribute__((unused)))
{
    PyErr_SetString(FooError, "Raise exception in C");
    return NULL;
}

static PyMethodDef methods[] = {
    {"foo", (PyCFunction)foo, METH_NOARGS, doc_foo},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", "doc", -1, methods
};

PyMODINIT_FUNC PyInit_foo(void)
```

(continues on next page)

(continued from previous page)

```
{
    PyObject *m = NULL;
    m = PyModule_Create(&module);
    if (!m) return NULL;

    FooError = PyErr_NewException("foo.FooError", NULL, NULL);
    Py_INCREF(FooError);
    PyModule_AddObject(m, "FooError", FooError);
    return m;
}
```

output:

```
$ python setup.py -q build
$ python setup.py -q install
$ python -c "import foo; foo.foo()"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
foo.FooError: Raise exception in C
```

13.10 Iterate a List

```
#include <Python.h>

#define PY_PRINTF(o) \
    PyObject_Print(o, stdout, 0); printf("\n");

static PyObject *
iter_list(PyObject *self, PyObject *args)
{
    PyObject *list = NULL, *item = NULL, *iter = NULL;
    PyObject *result = NULL;

    if (!PyArg_ParseTuple(args, "O", &list))
        goto error;

    if (!PyList_Check(list))
        goto error;

    // Get iterator
    iter = PyObject_GetIter(list);
    if (!iter)
        goto error;

    // for i in arr: print(i)
    while ((item = PyIter_Next(iter)) != NULL) {
        PY_PRINTF(item);
        Py_XDECREF(item);
    }

    Py_XINCRREF(Py_None);
    result = Py_None;
error:
    Py_XDECREF(iter);
```

(continues on next page)

(continued from previous page)

```

    return result;
}

static PyMethodDef methods[] = {
    {"iter_list", (PyCFunction)iter_list, METH_VARARGS, NULL},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, methods
};

PyMODINIT_FUNC PyInit_foo(void)
{
    return PyModule_Create(&module);
}

```

output:

```

$ python setup.py -q build
$ python setup.py -q install
$ python -c "import foo; foo.iter_list([1,2,3])"
1
2
3

```

13.11 Iterate a Dictionary

```

#include <Python.h>

#define PY_PRINTF(o) \
    PyObject_Print(o, stdout, 0); printf("\n");

static PyObject *
iter_dict(PyObject *self, PyObject *args)
{
    PyObject *dict = NULL;
    PyObject *key = NULL, *val = NULL;
    PyObject *o = NULL, *result = NULL;
    Py_ssize_t pos = 0;

    if (!PyArg_ParseTuple(args, "O", &dict))
        goto error;

    // for k, v in d.items(): print(f"{k}, {v}")
    while (PyDict_Next(dict, &pos, &key, &val)) {
        o = PyUnicode_FromFormat("(%S, %S)", key, val);
        if (!o) continue;
        PY_PRINTF(o);
        Py_XDECREF(o);
    }

    Py_INCREF(Py_None);
    result = Py_None;
}

```

(continues on next page)

(continued from previous page)

```

error:
    return result;
}

static PyMethodDef methods[] = {
    {"iter_dict", (PyCFunction)iter_dict, METH_VARARGS, NULL},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, methods
};

PyMODINIT_FUNC PyInit_foo(void)
{
    return PyModule_Create(&module);
}

```

output:

```

$ python setup.py -q build
$ python setup.py -q install
$ python -c "import foo; foo.iter_dict({'k': 'v'})"
' (k, v) '

```

13.12 Simple Class

```

#include <Python.h>

typedef struct {
    PyObject_HEAD
} FooObject;

/* class Foo(object): pass */

static PyTypeObject FooType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "foo.Foo",
    .tp_doc = "Foo objects",
    .tp_basicsize = sizeof(FooObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT,
    .tp_new = PyType_GenericNew
};

static PyModuleDef module = {
    PyModuleDef_HEAD_INIT,
    .m_name = "foo",
    .m_doc = "module foo",
    .m_size = -1
};

PyMODINIT_FUNC
PyInit_foo(void)

```

(continues on next page)

(continued from previous page)

```
{
    PyObject *m = NULL;
    if (PyType_Ready(&FooType) < 0)
        return NULL;
    if ((m = PyModule_Create(&module)) == NULL)
        return NULL;
    Py_XINCREF(&FooType);
    PyModule_AddObject(m, "Foo", (PyObject *) &FooType);
    return m;
}
```

output:

```
$ python setup.py -q build
$ python setup.py -q install
$ python -q
>>> import foo
>>> print(type(foo.Foo))
<class 'type'>
>>> o = foo.Foo()
>>> print(type(o))
<class 'foo.Foo'>
>>> class Foo(object): ...
...
>>> print(type(Foo))
<class 'type'>
>>> o = Foo()
>>> print(type(o))
<class '__main__.Foo'>
```

13.13 Simple Class with Members and Methods

```
#include <Python.h>
#include <structmember.h>

/*
 * class Foo:
 *     def __new__(cls, *a, **kw):
 *         foo_obj = object.__new__(cls)
 *         foo_obj.foo = ""
 *         foo_obj.bar = ""
 *         return foo_obj
 *
 *     def __init__(self, foo, bar):
 *         self.foo = foo
 *         self.bar = bar
 *
 *     def fib(self, n):
 *         if n < 2:
 *             return n
 *         return self.fib(n - 1) + self.fib(n - 2)
 */

typedef struct {
```

(continues on next page)

(continued from previous page)

```

    PyObject_HEAD
    PyObject *foo;
    PyObject *bar;
} FooObject;

static void
Foo_dealloc(FooObject *self)
{
    Py_XDECREF(self->foo);
    Py_XDECREF(self->bar);
    Py_TYPE(self)->tp_free((PyObject *) self);
}

static PyObject *
Foo_new(PyTypeObject *type, PyObject *args, PyObject *kw)
{
    int rc = -1;
    FooObject *self = NULL;
    self = (FooObject *) type->tp_alloc(type, 0);

    if (!self) goto error;

    /* allocate attributes */
    self->foo = PyUnicode_FromString("");
    if (self->foo == NULL) goto error;

    self->bar = PyUnicode_FromString("");
    if (self->bar == NULL) goto error;

    rc = 0;
error:
    if (rc < 0) {
        Py_XDECREF(self->foo);
        Py_XDECREF(self->bar);
        Py_XDECREF(self);
    }
    return (PyObject *) self;
}

static int
Foo_init(FooObject *self, PyObject *args, PyObject *kw)
{
    int rc = -1;
    static char *keywords[] = {"foo", "bar", NULL};
    PyObject *foo = NULL, *bar = NULL, *ptr = NULL;

    if (!PyArg_ParseTupleAndKeywords(args, kw,
                                     "|OO", keywords,
                                     &foo, &bar))
    {
        goto error;
    }

    if (foo) {
        ptr = self->foo;
        Py_INCREF(foo);
        self->foo = foo;
    }
}

```

(continues on next page)

(continued from previous page)

```

        Py_XDECREF(ptr);
    }

    if (bar) {
        ptr = self->bar;
        Py_INCREF(bar);
        self->bar = bar;
        Py_XDECREF(ptr);
    }
    rc = 0;
error:
    return rc;
}

static unsigned long
fib(unsigned long n)
{
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

static PyObject *
Foo_fib(PyObject *self, PyObject *args)
{
    unsigned long n = 0;
    if (!PyArg_ParseTuple(args, "k", &n)) return NULL;
    return PyLong_FromUnsignedLong(fib(n));
}

static PyMemberDef Foo_members[] = {
    {"foo", T_OBJECT_EX, offsetof(FooObject, foo), 0, NULL},
    {"bar", T_OBJECT_EX, offsetof(FooObject, bar), 0, NULL}
};

static PyMethodDef Foo_methods[] = {
    {"fib", (PyCFunction)Foo_fib, METH_VARARGS | METH_KEYWORDS, NULL},
    {NULL, NULL, 0, NULL}
};

static PyTypeObject FooType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "foo.Foo",
    .tp_doc = "Foo objects",
    .tp_basicsize = sizeof(FooObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE,
    .tp_new = Foo_new,
    .tp_init = (initproc) Foo_init,
    .tp_dealloc = (destructor) Foo_dealloc,
    .tp_members = Foo_members,
    .tp_methods = Foo_methods
};

static PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, NULL
};

```

(continues on next page)

(continued from previous page)

```
PyMODINIT_FUNC
PyInit_foo(void)
{
    PyObject *m = NULL;
    if (PyType_Ready(&FooType) < 0)
        return NULL;
    if ((m = PyModule_Create(&module)) == NULL)
        return NULL;
    Py_XINCREF(&FooType);
    PyModule_AddObject(m, "Foo", (PyObject *) &FooType);
    return m;
}
```

output:

```
$ python setup.py -q build
$ python setup.py -q install
$ python -q
>>> import foo
>>> o = foo.Foo('foo', 'bar')
>>> o.foo
'foo'
>>> o.bar
'bar'
>>> o.fib(10)
55
```

13.14 Simple Class with Getter and Setter

```
#include <Python.h>

/*
 * class Foo:
 *     def __new__(cls, *a, **kw):
 *         foo_obj = object.__new__(cls)
 *         foo_obj._foo = ""
 *         return foo_obj
 *
 *     def __init__(self, foo=None):
 *         if foo and isinstance(foo, 'str'):
 *             self._foo = foo
 *
 *     @property
 *     def foo(self):
 *         return self._foo
 *
 *     @foo.setter
 *     def foo(self, value):
 *         if not value or not isinstance(value, str):
 *             raise TypeError("value should be unicode")
 *         self._foo = value
 */

typedef struct {
```

(continues on next page)

(continued from previous page)

```

    PyObject_HEAD
    PyObject *foo;
} FooObject;

static void
Foo_dealloc(FooObject *self)
{
    Py_XDECREF(self->foo);
    Py_TYPE(self)->tp_free((PyObject *) self);
}

static PyObject *
Foo_new(PyTypeObject *type, PyObject *args, PyObject *kw)
{
    int rc = -1;
    FooObject *self = NULL;
    self = (FooObject *) type->tp_alloc(type, 0);

    if (!self) goto error;

    /* allocate attributes */
    self->foo = PyUnicode_FromString("");
    if (self->foo == NULL) goto error;

    rc = 0;
error:
    if (rc < 0) {
        Py_XDECREF(self->foo);
        Py_XDECREF(self);
    }
    return (PyObject *) self;
}

static int
Foo_init(FooObject *self, PyObject *args, PyObject *kw)
{
    int rc = -1;
    static char *keywords[] = {"foo", NULL};
    PyObject *foo = NULL, *ptr = NULL;

    if (!PyArg_ParseTupleAndKeywords(args, kw,
                                     "|O", keywords,
                                     &foo))
    {
        goto error;
    }

    if (foo && PyUnicode_Check(foo)) {
        ptr = self->foo;
        Py_INCREF(foo);
        self->foo = foo;
        Py_XDECREF(ptr);
    }

    rc = 0;
error:
    return rc;
}

```

(continues on next page)

(continued from previous page)

```

}

static PyObject *
Foo_getfoo(FooObject *self, void *closure)
{
    Py_INCREF(self->foo);
    return self->foo;
}

static int
Foo_setfoo(FooObject *self, PyObject *value, void *closure)
{
    int rc = -1;

    if (!value || !PyUnicode_Check(value)) {
        PyErr_SetString(PyExc_TypeError, "value should be unicode");
        goto error;
    }
    Py_INCREF(value);
    Py_XDECREF(self->foo);
    self->foo = value;
    rc = 0;
error:
    return rc;
}

static PyGetSetDef Foo_getsetters[] = {
    {"foo", (getter)Foo_getfoo, (setter)Foo_setfoo}
};

static PyTypeObject FooType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "foo.Foo",
    .tp_doc = "Foo objects",
    .tp_basicsize = sizeof(FooObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE,
    .tp_new = Foo_new,
    .tp_init = (initproc) Foo_init,
    .tp_dealloc = (destructor) Foo_dealloc,
    .tp_getset = Foo_getsetters,
};

static PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, NULL
};

PyMODINIT_FUNC
PyInit_foo(void)
{
    PyObject *m = NULL;
    if (PyType_Ready(&FooType) < 0)
        return NULL;
    if ((m = PyModule_Create(&module)) == NULL)
        return NULL;
    Py_XINCREF(&FooType);
    PyModule_AddObject(m, "Foo", (PyObject *) &FooType);
}

```

(continues on next page)

(continued from previous page)

```

    return m;
}

```

output:

```

$ python setup.py -q build
$ python setup.py -q install
$ python -q
>>> import foo
>>> o = foo.Foo()
>>> o.foo
''
>>> o.foo = "foo"
>>> o.foo
'foo'
>>> o.foo = None
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: value should be unicode

```

13.15 Inherit from Other Class

```

#include <Python.h>
#include <structmember.h>

/*
 * class Foo:
 *     def __new__(cls, *a, **kw):
 *         foo_obj = object.__new__(cls)
 *         foo_obj.foo = ""
 *         return foo_obj
 *
 *     def __init__(self, foo):
 *         self.foo = foo
 *
 *     def fib(self, n):
 *         if n < 2:
 *             return n
 *         return self.fib(n - 1) + self.fib(n - 2)
 */

/* FooObject */

typedef struct {
    PyObject_HEAD
    PyObject *foo;
} FooObject;

static void
Foo_dealloc(FooObject *self)
{
    Py_XDECREF(self->foo);
    Py_TYPE(self)->tp_free((PyObject *) self);
}

```

(continues on next page)

(continued from previous page)

```

static PyObject *
Foo_new(PyTypeObject *type, PyObject *args, PyObject *kw)
{
    int rc = -1;
    FooObject *self = NULL;
    self = (FooObject *) type->tp_alloc(type, 0);

    if (!self) goto error;

    /* allocate attributes */
    self->foo = PyUnicode_FromString("");
    if (self->foo == NULL) goto error;

    rc = 0;
error:
    if (rc < 0) {
        Py_XDECREF(self->foo);
        Py_XDECREF(self);
    }
    return (PyObject *) self;
}

static int
Foo_init(FooObject *self, PyObject *args, PyObject *kw)
{
    int rc = -1;
    static char *keywords[] = {"foo", NULL};
    PyObject *foo = NULL, *ptr = NULL;

    if (!PyArg_ParseTupleAndKeywords(args, kw, "|O", keywords, &foo)) {
        goto error;
    }

    if (foo) {
        ptr = self->foo;
        Py_INCREF(foo);
        self->foo = foo;
        Py_XDECREF(ptr);
    }
    rc = 0;
error:
    return rc;
}

static unsigned long
fib(unsigned long n)
{
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

static PyObject *
Foo_fib(FooObject *self, PyObject *args)
{
    unsigned long n = 0;
    if (!PyArg_ParseTuple(args, "k", &n)) return NULL;

```

(continues on next page)

(continued from previous page)

```

    return PyLong_FromUnsignedLong(fib(n));
}

static PyMemberDef Foo_members[] = {
    {"foo", T_OBJECT_EX, offsetof(FooObject, foo), 0, NULL}
};

static PyMethodDef Foo_methods[] = {
    {"fib", (PyCFunction)Foo_fib, METH_VARARGS | METH_KEYWORDS, NULL},
    {NULL, NULL, 0, NULL}
};

static PyObject FooType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "foo.Foo",
    .tp_doc = "Foo objects",
    .tp_basicsize = sizeof(FooObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE,
    .tp_new = Foo_new,
    .tp_init = (initproc) Foo_init,
    .tp_dealloc = (destructor) Foo_dealloc,
    .tp_members = Foo_members,
    .tp_methods = Foo_methods
};

/*
 * class Bar(Foo):
 *     def __init__(self, bar):
 *         super().__init__(bar)
 *
 *     def gcd(self, a, b):
 *         while b:
 *             a, b = b, a % b
 *         return a
 */

/* BarObject */

typedef struct {
    FooObject super;
} BarObject;

static unsigned long
gcd(unsigned long a, unsigned long b)
{
    unsigned long t = 0;
    while (b) {
        t = b;
        b = a % b;
        a = t;
    }
    return a;
}

static int
Bar_init(FooObject *self, PyObject *args, PyObject *kw)

```

(continues on next page)

(continued from previous page)

```

{
    return FooType.tp_init((PyObject *) self, args, kw);
}

static PyObject *
Bar_gcd(PyObject *self, PyObject *args)
{
    unsigned long a = 0, b = 0;
    if (!PyArg_ParseTuple(args, "kk", &a, &b)) return NULL;
    return PyLong_FromUnsignedLong(gcd(a, b));
}

static PyMethodDef Bar_methods[] = {
    {"gcd", (PyCFunction)Bar_gcd, METH_VARARGS, NULL},
    {NULL, NULL, 0, NULL}
};

static PyTypeObject BarType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "foo.Bar",
    .tp_doc = "Bar objects",
    .tp_basicsize = sizeof(BarObject),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE,
    .tp_base = &FooType,
    .tp_init = (initproc) Bar_init,
    .tp_methods = Bar_methods
};

/* Module */

static PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, NULL
};

PyMODINIT_FUNC
PyInit_foo(void)
{
    PyObject *m = NULL;
    if (PyType_Ready(&FooType) < 0)
        return NULL;
    if (PyType_Ready(&BarType) < 0)
        return NULL;
    if ((m = PyModule_Create(&module)) == NULL)
        return NULL;

    Py_XINCREF(&FooType);
    Py_XINCREF(&BarType);
    PyModule_AddObject(m, "Foo", (PyObject *) &FooType);
    PyModule_AddObject(m, "Bar", (PyObject *) &BarType);
    return m;
}

```

output:

```

$ python setup.py -q build
$ python setup.py -q install

```

(continues on next page)

(continued from previous page)

```
$ python -q
>>> import foo
>>> bar = foo.Bar('bar')
>>> bar.foo
'bar'
>>> bar.fib(10)
55
>>> bar.gcd(3, 7)
1
```

13.16 Run a Python command from C

```
#include <stdio.h>
#include <Python.h>

int
main(int argc, char *argv[])
{
    int rc = -1;
    Py_Initialize();
    rc = PyRun_SimpleString(argv[1]);
    Py_Finalize();
    return rc;
}
```

output:

```
$ clang `python3-config --cflags` -c foo.c -o foo.o
$ clang `python3-config --ldflags` foo.o -o foo
$ ./foo "print('Hello Python')"
Hello Python
```

13.17 Run a Python file from C

```
#include <stdio.h>
#include <Python.h>

int
main(int argc, char *argv[])
{
    int rc = -1, i = 0;
    wchar_t **argv_copy = NULL;
    const char *filename = NULL;
    FILE *fp = NULL;
    PyCompilerFlags cf = {.cf_flags = 0};

    filename = argv[1];
    fp = fopen(filename, "r");
    if (!fp)
        goto error;
}
```

(continues on next page)

(continued from previous page)

```

// copy argv
argv_copy = PyMem_RawMalloc(sizeof(wchar_t*) * argc);
if (!argv_copy)
    goto error;

for (i = 0; i < argc; i++) {
    argv_copy[i] = Py_DecodeLocale(argv[i], NULL);
    if (argv_copy[i]) continue;
    fprintf(stderr, "Unable to decode the argument");
    goto error;
}

Py_Initialize();
Py_SetProgramName(argv_copy[0]);
PySys_SetArgv(argc, argv_copy);
rc = PyRun_AnyFileExFlags(fp, filename, 0, &cf);

error:
    if (argv_copy) {
        for (i = 0; i < argc; i++)
            PyMem_RawFree(argv_copy[i]);
        PyMem_RawFree(argv_copy);
    }
    if (fp) fclose(fp);
    Py_Finalize();
    return rc;
}

```

output:

```

$ clang `python3-config --cflags` -c foo.c -o foo.o
$ clang `python3-config --ldflags` foo.o -o foo
$ echo "import sys; print(sys.argv)" > foo.py
$ ./foo foo.py arg1 arg2 arg3
['./foo', 'foo.py', 'arg1', 'arg2', 'arg3']

```

13.18 Import a Python Module from C

```

#include <stdio.h>
#include <Python.h>

#define PYOBJECT_CHECK(obj, label) \
    if (!obj) { \
        PyErr_Print(); \
        goto label; \
    }

int
main(int argc, char *argv[])
{
    int rc = -1;
    wchar_t *program = NULL;
    PyObject *json_module = NULL, *json_dict = NULL;
    PyObject *json_dumps = NULL;

```

(continues on next page)

(continued from previous page)

```

PyObject *dict = NULL;
PyObject *result = NULL;

program = Py_DecodeLocale(argv[0], NULL);
if (!program) {
    fprintf(stderr, "unable to decode the program name");
    goto error;
}

Py_SetProgramName(program);
Py_Initialize();

// import json
json_module = PyImport_ImportModule("json");
PYOBJECT_CHECK(json_module, error);

// json_dict = json.__dict__
json_dict = PyModule_GetDict(json_module);
PYOBJECT_CHECK(json_dict, error);

// json_dumps = json.__dict__['dumps']
json_dumps = PyDict_GetItemString(json_dict, "dumps");
PYOBJECT_CHECK(json_dumps, error);

// dict = {'foo': 'Foo', 'bar': 123}
dict = Py_BuildValue("({sssi})", "foo", "Foo", "bar", 123);
PYOBJECT_CHECK(dict, error);

// result = json.dumps(dict)
result = PyObject_CallObject(json_dumps, dict);
PYOBJECT_CHECK(result, error);
PyObject_Print(result, stdout, 0);
printf("\n");
rc = 0;

error:
    Py_XDECREF(result);
    Py_XDECREF(dict);
    Py_XDECREF(json_dumps);
    Py_XDECREF(json_dict);
    Py_XDECREF(json_module);

    PyMem_RawFree(program);
    Py_Finalize();
    return rc;
}

```

output:

```

$ clang `python3-config --cflags` -c foo.c -o foo.o
$ clang `python3-config --ldflags` foo.o -o foo
$ ./foo
{"foo": "Foo", "bar": 123}

```

13.19 Import All Attributes of a Module

```
#include <stdio.h>
#include <Python.h>

#define PYOBJECT_CHECK(obj, label) \
    if (!obj) { \
        PyErr_Print(); \
        goto label; \
    }

int
main(int argc, char *argv[])
{
    int rc = -1;
    wchar_t *program = NULL;
    PyObject *main_module = NULL, *main_dict = NULL;
    PyObject *uname = NULL;
    PyObject *sysname = NULL;
    PyObject *result = NULL;

    program = Py_DecodeLocale(argv[0], NULL);
    if (!program) {
        fprintf(stderr, "unable to decode the program name");
        goto error;
    }

    Py_SetProgramName(program);
    Py_Initialize();

    // import __main__
    main_module = PyImport_ImportModule("__main__");
    PYOBJECT_CHECK(main_module, error);

    // main_dict = __main__.__dict__
    main_dict = PyModule_GetDict(main_module);
    PYOBJECT_CHECK(main_dict, error);

    // from os import *
    result = PyRun_String("from os import *",
                          Py_file_input,
                          main_dict,
                          main_dict);
    PYOBJECT_CHECK(result, error);
    Py_XDECREF(result);
    Py_XDECREF(main_dict);

    // uname = __main__.__dict__['uname']
    main_dict = PyModule_GetDict(main_module);
    PYOBJECT_CHECK(main_dict, error);

    // result = uname()
    uname = PyDict_GetItemString(main_dict, "uname");
    PYOBJECT_CHECK(uname, error);
    result = PyObject_CallObject(uname, NULL);
    PYOBJECT_CHECK(result, error);
}
```

(continues on next page)

(continued from previous page)

```

    // sysname = result.sysname
    sysname = PyObject_GetAttrString(result, "sysname");
    PYOBJECT_CHECK(sysname, error);
    PyObject_Print(sysname, stdout, 0);
    printf("\n");

    rc = 0;
error:
    Py_XDECREF(sysname);
    Py_XDECREF(result);
    Py_XDECREF(uname);
    Py_XDECREF(main_dict);
    Py_XDECREF(main_module);

    PyMem_RawFree(program);
    Py_Finalize();
    return rc;
}

```

output:

```

$ clang `python3-config --cflags` -c foo.c -o foo.o
$ clang `python3-config --ldflags` foo.o -o foo
$ ./foo
'Darwin'

```

13.20 Access Attributes

```

#include <stdio.h>
#include <Python.h>

#define PYOBJECT_CHECK(obj, label) \
    if (!obj) { \
        PyErr_Print(); \
        goto label; \
    }

int
main(int argc, char *argv[])
{
    int rc = -1;
    wchar_t *program = NULL;
    PyObject *json_module = NULL;
    PyObject *json_dumps = NULL;
    PyObject *dict = NULL;
    PyObject *result = NULL;

    program = Py_DecodeLocale(argv[0], NULL);
    if (!program) {
        fprintf(stderr, "unable to decode the program name");
        goto error;
    }
}

```

(continues on next page)

(continued from previous page)

```

Py_SetProgramName(program);
Py_Initialize();

// import json
json_module = PyImport_ImportModule("json");
PYOBJECT_CHECK(json_module, error);

// json_dumps = json.dumps
json_dumps = PyObject_GetAttrString(json_module, "dumps");
PYOBJECT_CHECK(json_dumps, error);

// dict = {'foo': 'Foo', 'bar': 123}
dict = Py_BuildValue("{sssi}", "foo", "Foo", "bar", 123);
PYOBJECT_CHECK(dict, error);

// result = json.dumps(dict)
result = PyObject_CallObject(json_dumps, dict);
PYOBJECT_CHECK(result, error);
PyObject_Print(result, stdout, 0);
printf("\n");
rc = 0;
error:
Py_XDECREF(result);
Py_XDECREF(dict);
Py_XDECREF(json_dumps);
Py_XDECREF(json_module);

PyMem_RawFree(program);
Py_Finalize();
return rc;
}

```

output:

```

$ clang `python3-config --cflags` -c foo.c -o foo.o
$ clang `python3-config --ldflags` foo.o -o foo
$ ./foo
'{"foo": "Foo", "bar": 123}'

```

13.21 Performance of C Extension

```

#include <Python.h>

static unsigned long
fib(unsigned long n)
{
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

static PyObject *
fibonacci(PyObject *self, PyObject *args)
{
    unsigned long n = 0;

```

(continues on next page)

(continued from previous page)

```

    if (!PyArg_ParseTuple(args, "k", &n)) return NULL;
    return PyLong_FromUnsignedLong(fib(n));
}

static PyMethodDef methods[] = {
    {"fib", (PyCFunction)fibonacci, METH_VARARGS, NULL},
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef module = {
    PyModuleDef_HEAD_INIT, "foo", NULL, -1, methods
};

PyMODINIT_FUNC PyInit_foo(void)
{
    return PyModule_Create(&module);
}

```

Compare the performance with pure Python

```

>>> from time import time
>>> import foo
>>> def fib(n):
...     if n < 2: return n
...     return fib(n - 1) + fib(n - 2)
...
>>> s = time(); _ = fib(35); e = time(); e - s
4.953313112258911
>>> s = time(); _ = foo.fib(35); e = time(); e - s
0.04628586769104004

```

13.22 Performance of ctypes

```

// Compile (Mac)
// -----
//
// $ clang -Wall -Werror -shared -fPIC -o libfib.dylib fib.c
//
unsigned int fib(unsigned int n)
{
    if ( n < 2) {
        return n;
    }
    return fib(n-1) + fib(n-2);
}

```

Compare the performance with pure Python

```

>>> from time import time
>>> from ctypes import CDLL
>>> def fib(n):
...     if n < 2: return n
...     return fib(n - 1) + fib(n - 2)
...

```

(continues on next page)

(continued from previous page)

```
>>> cfib = CDLL("./libfib.dylib").fib
>>> s = time(); _ = fib(35); e = time(); e - s
4.918856859207153
>>> s = time(); _ = cfib(35); e = time(); e - s
0.07283687591552734
```

13.23 Error handling when using ctypes

```
from __future__ import print_function

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;
#ifdef _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;
```

(continues on next page)

(continued from previous page)

```

        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

```