python-cheatsheet Documentation

Release 0.1.0

crazyguitar

Nov 30, 2018

Contents

Welcome to pysheeet. This project aims at collecting useful Python snippets in order to enhance pythoneers' coding experiences. Please feel free to contribute if you have any awesome ideas for improvements to code snippets, explanations, etc.

Any snippets are welcome. If you'd like to contribute, fork pysheeet on GitHub. If there is any question or suggestion, please create an issue on GitHub Issues.

CHAPTER 1

What's New In Python 3

The official document, What's New In Python, displays all of the most important changes. However, if you're too busy to read the whole changes, this part provides a brief glance of new features in Python 3.

1.1 New in Python3

Table of Contents

1.1.1 `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
```

1.1.2 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\xc3\xa9'
>>> type(s)
<type 'str'>
>>> u = u'Café' # unicode string
>>> u
u'Caf\xe9'
>>> type(u)
<type 'unicode'>
>>> len([_c for _c in 'Café'])
5
```

Python 3

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

1.1.3 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
```

1.1.4 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'}
```

1.1.5 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
```

1.1.6 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
```

1.1.7 Remove <>

New in Python 3.0

Python 2

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

# equal to !=
>>> a != "Python3"
True
```

Python 3

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

1.1.8 BDFL retirement

New in Python 3.1

- PEP 401 - BDFL Retirement

```
>>> from __future__ import barry_as_FLUFL
>>> 1 != 2
  File "<stdin>", line 1
    1 != 2
       ^
SyntaxError: with Barry as BDFL, use '<>' instead of '!='
>>> 1 <> 2
True
```

1.1.9 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
```

1.1.10 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
```

1.1.11 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)]:
...     print(a, b)
...
1 [2, 3]
4 [5, 6, 7]
```

1.1.12 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'}
```

1.1.13 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
...         b, a = a + b, b
...
>>> [f for f in fib(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

1.1.14 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'}
```

1.1.15 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
```

1.1.16 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'
```

1.1.17 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'
```

1.1.18 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
```

1.1.19 Generator delegation

New in Python 3.3

- PEP 380 - Syntax for Delegating to a Subgenerator

```
>>> def fib(n: int):
...     a, b = 0, 1
```

(continues on next page)

(continued from previous page)

```
...     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]
```

1.1.20 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)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(coro(3))
```

(continues on next page)

(continued from previous page)

```
0
1
1
```

1.1.21 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
```

1.1.22 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):
...         print(f, end=" ")
...     else:
...         print()
...
>>> loop = asyncio.get_event_loop()
```

(continues on next page)

(continued from previous page)

```
>>> 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']
```

1.1.23 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)
...     def __str__(self):
...         return self.__repr__()
...     def __repr__(self):
...         return "Arr({})".format(repr(self._arr))
...
>>> a = Arr(9, 5, 2, 7)
```

(continues on next page)

(continued from previous page)

```
>>> b = Arr(5, 5, 6, 6)
>>> a @ b  # __matmul__
124
>>> a @= b  # __imatmul__
>>> a
Arr([124])
```

1.1.24 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)
...
cannot assign to field 'z'
```

1.1.25 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
```

CHAPTER 2

Cheat Sheet

This part mainly focuses on common snippets in Python code. The cheat sheet not only includes basic Python features but also data structures and algorithms.

2.1 Style

Table of Contents

2.1.1 Naming

Class

Bad

```
class fooClass: ...
class foo_class: ...
```

Good

```
class FooClass: ...
```

Function

Bad

```
def CapCamelCase(*a): ...
def mixCamelCase(*a): ...
```

Good

```
def func_separated_by_underscores(*a): ...
```

Variable

Bad

```
FooVar = "CapWords"
fooVar = "mixedCase"
Foo_Var = "CapWords_With_Underscore"
```

Good

```
# local variable
var = "lowercase"

# internal use
_var = "_single_leading_underscore"

# avoid conflicts with Python keyword
var_ = "single_trailing_underscore_"

# a class attribute (private use in class)
var = " __double_leading_underscore"

# "magic" objects or attributes, ex: __init__
__name__

# throwaway variable, ex: _, v = (1, 2)
_ = "throwaway"
```

2.2 From Scratch

The main goal of this cheat sheet is to collect some common and basic semantics or snippets. The cheat sheet includes some syntax, which we have already known but still ambiguous in our mind, or some snippets, which we google them again and again. In addition, because **the end Of life date for Python 2** is coming. Most of the snippets are mainly based on **Python 3**'s syntax.

Table of Contents

2.2.1 Hello world!

When we start to learn a new language, we usually learn from printing **Hello world!**. In Python, we can use another way to print the message by importing `__hello__` module. The source code can be found on frozen.c.

```
>>> print("Hello world!")
Hello world!
>>> import __hello__
Hello world!
>>> import __phello__
Hello world!
>>> import __phello__.spam
Hello world!
```

2.2.2 Python Version

It is important for a programmer to know current Python version because not every syntax will work in the current version. In this case, we can get the Python version by `python -V` or using the module, `sys`.

```
>>> import sys
>>> print(sys.version)
3.7.1 (default, Nov  6 2018, 18:46:03)
[Clang 10.0.0 (clang-1000.11.45.5)]
```

We can also use `platform.python_version` to get Python version.

```
>>> import platform
>>> platform.python_version()
'3.7.1'
```

Sometimes, checking the current Python version is important because we may want to enable some features in some specific versions. `sys.version_info` provides more detail information about the interpreter. We can use it to compare with the version we want.

```
>>> import sys
>>> sys.version_info >= (3, 6)
True
>>> sys.version_info >= (3, 7)
False
```

2.2.3 Ellipsis

Ellipsis is a built-in constant. After Python 3.0, we case use `...` as `Ellipsis`. It may be the most enigmatic constant in Python. Based on the official document, we can use it to extend slicing syntax. Nevertheless, there are some other conventions in type hinting, stub files, or function expressions.

```
>>> ...
Ellipsis
>>> ... == Ellipsis
True
>>> type(...)
<class 'ellipsis'>
```

The following snippet shows that we can use the ellipsis to represent a function or a class which has not implemented yet.

```
>>> class Foo: ...
...
>>> def foo(): ...
...
```

2.2.4 if ... elif ... else

The **if statements** are used to control the code flow. Instead of using `switch` or `case` statements control the logic of the code, Python uses `if ... elif ... else` sequence. Although someone proposes we can use `dict` to achieve `switch` statements, this solution may introduce unnecessary overhead such as creating disposable dictionaries and undermine a readable code. Thus, the solution is not recommended.

```
>>> import random
>>> num = random.randint(0, 10)
>>> if num < 3:
...     print("less than 3")
... elif num < 5:
...     print("less than 5")
... else:
...     print(num)
...
less than 3
```

2.2.5 for Loop

In Python, we can access iterable object's items directly through the **for statement**. If we need to get indexes and items of an iterable object such as list or tuple at the same time, using `enumerate` is better than `range(len(iterable))`. Further information can be found on Looping Techniques.

```
>>> for val in ["foo", "bar"]:
...     print(val)
...
foo
bar
>>> for idx, val in enumerate(["foo", "bar", "baz"]):
...     print(idx, val)
...
(0, 'foo')
(1, 'bar')
(2, 'baz')
```

2.2.6 for ... else ...

It may be a little weired when we see the `else` belongs to a `for` loop at the first time. The `else` clause can assist us to avoid using flag variables in loops. A loop's `else` clause runs when no break occurs.

```
>>> for _ in range(5):
...     pass
... else:
...     print("no break")
...
no break
```

The following snippet shows the difference between using a flag variable and the `else` clause to control the loop. We can see that the `else` does not run when the `break` occurs in the loop.

```
>>> is_break = False
>>> for x in range(5):
...     if x % 2 == 0:
...         is_break = True
...         break
...
>>> if is_break:
...     print("break")
...
break

>>> for x in range(5):
...     if x % 2 == 0:
...         print("break")
...         break
... else:
...     print("no break")
...
break
```

2.2.7 Using `range`

The problem of `range` in Python 2 is that `range` may take up a lot of memory if we need to iterate a loop many times. Consequently, using `xrange` is recommended in Python 2.

```
>>> import platform
>>> import sys
>>> platform.python_version()
'2.7.15'
>>> sys.getsizeof(range(100000000))
800000072
>>> sys.getsizeof(xrange(100000000))
40
```

In Python 3, the built-in function `range` returns an iterable **range object** instead of a list. The behavior of `range` is the same as the `xrange` in Python 2. Therefore, using `range` do not take up huge memory anymore if we want to run a code block many times within a loop. Further information can be found on PEP 3100.

```
>>> import platform
>>> import sys
>>> platform.python_version()
'3.7.1'
>>> sys.getsizeof(range(100000000))
48
```

2.2.8 while ... else ...

The `else` clause belongs to a while loop serves the same purpose as the `else` clause in a for loop. We can observe that the `else` does not run when the `break` occurs in the while loop.

```
>>> n = 0
>>> while n < 5:
...     if n == 3:
...         break
...     n += 1
... else:
...     print("no break")
...
```

2.2.9 The `do while` Statement

There are many programming languages such as C/C++, Ruby, or Javascript, provide the `do while` statement. In Python, there is no `do while` statement. However, we can place the condition and the `break` at the end of a `while` loop to achieve the same thing.

```
>>> n = 0
>>> while True:
...     n += 1
...     if n == 5:
...         break
...
>>> n
5
```

2.2.10 try . . . except . . . else . . .

Most of the time, we handle errors in `except` clause and clean up resources in `finally` clause. Interestingly, the `try` statement also provides an `else` clause for us to avoid catching an exception which was raised by the code that should not be protected by `try ... except`. The `else` clause runs when no exception occurs between `try` and `except`.

```
>>> try:
...     print("No exception")
... except:
...     pass
... else:
...     print("Success")
...
No exception
Success
```

2.2.11 List

Lists are versatile containers. Python provides a lot of ways such as **negative index**, **slicing statement**, or **list comprehension** to manipulate lists. The following snippet shows some common operations of lists.

```
>>> a = [1, 2, 3, 4, 5]
>>> a[-1]                     # negative index
5
>>> a[1:]                     # slicing
[2, 3, 4, 5]
>>> a[1:-1]
[2, 3, 4]
>>> a[1:-1:2]
[2, 4]
>>> a[::-1]                   # reverse
[5, 4, 3, 2, 1]
>>> a[0] = 0                  # set an item
>>> a
[0, 2, 3, 4, 5]
>>> a.append(6)               # append an item
>>> a
[0, 2, 3, 4, 5, 6]
>>> del a[-1]                 # del an item
>>> a
[0, 2, 3, 4, 5]
>>> b = [x for x in range(3)] # list comprehension
>>> b
[0, 1, 2]
>>> a + b                     # add two lists
[0, 2, 3, 4, 5, 0, 1, 2]
```

2.2.12 Dict

Dictionaries are key-value pairs containers. Like lists, Python supports many ways such as **dict comprehensions** to manipulate dictionaries. After Python 3.6, dictionaries preserve the insertion order of keys. The Following snippet shows some common operations of dictionaries.

```
>>> d = {'timmy': 'red', 'barry': 'green', 'guido': 'blue'}
>>> d
{'timmy': 'red', 'barry': 'green', 'guido': 'blue'}
>>> d['timmy'] = "yellow"        # set data
>>> d
{'timmy': 'yellow', 'barry': 'green', 'guido': 'blue'}
>>> del d['guido']               # del data
>>> d
>>> 'guido' in d                 # contain data
False
{'timmy': 'yellow', 'barry': 'green'}
>>> {k: v for k ,v in d.items()} # dict comprehension
{'timmy': 'yellow', 'barry': 'green'}
>>> d.keys()                     # list all keys
dict_keys(['timmy', 'barry'])
>>> d.values()                   # list all values
dict_values(['yellow', 'green'])
```

2.2.13 Function

Defining a function in Python is flexible. We can define a function with **function documents**, **default values**, **arbitrary arguments**, **keyword arguments**, **keyword-only arguments**, and so on. The Following snippet shows some common expressions to define functions.

```
def foo_with_doc():
    """Documentation String."""

def foo_with_arg(arg): ...
def foo_with_args(*arg): ...
def foo_with_kwarg(a, b="foo"): ...
def foo_with_args_kwargs(*args, **kwargs): ...
def foo_with_kwonly(a, b, *, k): ...            # python3
def foo_with_annotations(a: int) -> int: ...    # python3
```

2.2.14 Function Annotations

Instead of writing string documents in functions to hint the type of parameters and return values, we can denote types by **function annotations**. Function annotations which the details can be found on PEP 3017 and PEP 484 were introduced in Python 3.0. They are an **optional** feature in **Python 3**. Using function annotations will lose compatibility in **Python 2**. We can solve this issue by stub files. In addition, we can do static type checking through mypy.

```
>>> def fib(n: int) -> int:
...     a, b = 0, 1
...     for _ in range(n):
...         b, a = a + b, b
...     return a
...
>>> fib(10)
55
```

2.2.15 Generators

Python uses the `yield` statement to define a **generator function**. In other words, when we call a generator function, the generator function will return a **generator** instead of return values for creating an **iterator**.

```
>>> def fib(n):
...     a, b = 0, 1
...     for _ in range(n):
...         yield a
...         b, a = a + b, b
...
>>> g = fib(10)
>>> g
<generator object fib at 0x10b240c78>
>>> for f in fib(5):
...     print(f)
...
0
1
1
2
3
```

2.2.16 Generator Delegation

Python 3.3 introduced `yield from` expression. It allows a generator to delegate parts of operations to another generator. In other words, we can **yield** a sequence **from** other **generators** in the current **generator function**. Further information can be found on PEP 380.

```
>>> def fib(n):
...     a, b = 0, 1
...     for _ in range(n):
...         yield a
...         b, a = a + b, b
...
>>> def fibonacci(n):
...     yield from fib(n)
...
>>> [f for f in fibonacci(5)]
[0, 1, 1, 2, 3]
```

2.2.17 Class

Python supports many common features such as **class documents**, **multiple inheritance**, **class variables**, **instance variables**, **static method**, **class method**, and so on. Furthermore, Python provides some special methods for programmers to implement **iterators**, **context manager**, etc. The following snippet displays common definition of a class.

```
class A: ...
class B: ...
class Foo(A, B):
    """A class document."""

    foo = "class variable"
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, v):
        self.attr = v
        self.__private = "private var"

    @staticmethod
    def bar_static_method(): ...

    @classmethod
    def bar_class_method(cls): ...

    def bar(self):
        """A method document."""

    def bar_with_arg(self, arg): ...
    def bar_with_args(self, *args): ...
    def bar_with_kwarg(self, kwarg="bar"): ...
    def bar_with_args_kwargs(self, *args, **kwargs): ...
    def bar_with_kwonly(self, *, k): ...
    def bar_with_annotations(self, a: int): ...
```

2.2.18 async / await

`async` and `await` syntax was introduced from Python 3.5. They were designed to be used with an event loop. Some other features such as the **asynchronous generator** were implemented in later versions.

A **coroutine function** (`async def`) are used to create a **coroutine** for an event loop. Python provides a built-in module, **asyncio**, to write a concurrent code through `async`/`await` syntax. The following snippet shows a simple example of using **asyncio**. The code must be run on Python 3.7 or above.

```
import asyncio

async def http_ok(r, w):
    head = b"HTTP/1.1 200 OK\r\n"
    head += b"Content-Type: text/html\r\n"
    head += b"\r\n"

    body = b"<html>"
    body += b"<body><h1>Hello world!</h1></body>"
    body += b"</html>"

    _ = await r.read(1024)
    w.write(head + body)
    await w.drain()
    w.close()

async def main():
    server = await asyncio.start_server(
        http_ok, "127.0.0.1", 8888
    )

    async with server:
        await server.serve_forever()

asyncio.run(main())
```

2.2.19 Avoid exec and eval

The following snippet shows how to use the built-in function exec. Yet, using exec and eval are not recommended because of some security issues and unreadable code for a human. Further reading can be found on Be careful with exec and eval in Python and Eval really is dangerous

```
>>> py = '''
... def fib(n):
...     a, b = 0, 1
...     for _ in range(n):
...         b, a = b + a, b
...     return a
... print(fib(10))
... '''
>>> exec(py, globals(), locals())
55
```

2.3 Future

Future statements tell the interpreter to compile some semantics as the semantics which will be available in the future Python version. In other words, Python uses from __future__ import feature to backport features from other higher Python versions to the current interpreter. In Python 3, many features such as print_function are already enabled, but we still leave these future statements for backward compatibility.

Future statements are **NOT** import statements. Future statements change how Python interprets the code. They **MUST** be at the top of the file. Otherwise, Python interpreter will raise SyntaxError.

If you're interested in future statements and want to acquire more explanation, further information can be found on PEP 236 - Back to the __future__

Table of Contents

2.3.1 List All New Features

__future__ is a Python module. We can use it to check what kind of future features can import to current Python interpreter. The fun is import __future__ is **NOT** a future statement, it is a import statement.

```
>>> from pprint import pprint
>>> import __future__
>>> pprint(__future__.all_feature_names)
['nested_scopes',
 'generators',
 'division',
 'absolute_import',
 'with_statement',
 'print_function',
 'unicode_literals',
 'barry_as_FLUFL',
 'generator_stop',
 'annotations']
```

Future statements not only change the behavior of the Python interpreter but also import `__future__._Feature` into the current program.

```
>>> from __future__ import print_function
>>> print_function
_Feature((2, 6, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 65536)
```

2.3.2 Print Function

Replacing **print statement** to **print function** is one of the most notorious decision in Python history. However, this change brings some flexibilities to extend the ability of `print`. Further information can be found on PEP 3105.

```
>>> 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
```

2.3.3 Unicode

As **print function**, making text become Unicode is another infamous decision. Nevertheless, many modern programming languages' text is Unicode. This change compels us to decode texts early in order to prevent runtime error after we run programs for a while. Further information can be found on PEP 3112.

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

2.3.4 Division

Sometimes, it is counterintuitive when the division result is int or long. In this case, Python 3 enables the **true division** by default. However, in Python 2, we have to backport `division` to the current interpreter. Further information can be found on PEP 238.

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

2.3.5 Annotations

Before Python 3.7, we cannot assign annotations in a class or a function if it is not available in the current scope. A common situation is the definition of a container class.

```
class Tree(object):

    def insert(self, tree: Tree): ...
```

Example

```
$ python3 foo.py
Traceback (most recent call last):
  File "foo.py", line 1, in <module>
    class Tree(object):
  File "foo.py", line 3, in Tree
    def insert(self, tree: Tree): ...
NameError: name 'Tree' is not defined
```

In this case, the definition of the class is not available yet. Python interpreter cannot parse the annotation during their definition time. To solve this issue, Python uses string literals to replace the class.

```
class Tree(object):

    def insert(self, tree: 'Tree'): ...
```

After version 3.7, Python introduces the future statement, `annotations`, to perform postponed evaluation. It will become the default feature in Python 4. For further information please refer to PEP 563.

```
from __future__ import annotations

class Tree(object):

    def insert(self, tree: Tree): ...
```

2.3.6 BDFL Retirement

New in Python 3.1

PEP 401 is just an Easter egg. This feature brings the current interpreter back to the past. It enables the diamond operator <> in Python 3.

```
>>> 1 != 2
True
```

(continues on next page)

(continued from previous page)

```
>>> from __future__ import barry_as_FLUFL
>>> 1 != 2
  File "<stdin>", line 1
    1 != 2
       ^
SyntaxError: with Barry as BDFL, use '<>' instead of '!='
>>> 1 <> 2
True
```

2.3.7 Braces

`braces` is an Easter egg. The source code can be found on future.c.

```
>>> from __future__ import braces
  File "<stdin>", line 1
SyntaxError: not a chance
```

2.4 Unicode

Table of Contents

2.4.1 Encode: unicode code point to bytes

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

2.4.2 Decode: bytes to unicode code point

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

2.4.3 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
```

2.4.4 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'>
```

2.4.5 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'
```

2.4.6 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
```

2.4.7 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
```

2.4.8 unicode normalization

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

2.4.9 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")
''
```

2.4.10 Long String

Original 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'
```

2.5 List

The list is a common data structure which we use to store objects. Most of the time, programmers concern about getting, setting, searching, filtering, and sorting. Furthermore, sometimes, we waltz ourself into common pitfalls of the memory management. Thus, the main goal of this cheat sheet is to collect some common operations and pitfalls.

Table of Contents

2.5.1 From Scratch

There are so many ways that we can manipulate lists in Python. Before we start to learn those versatile manipulations, the following snippet shows the most common operations of lists.

```
>>> a = [1, 2, 3, 4, 5]
>>> # contains
```

(continues on next page)

(continued from previous page)

```
>>> 2 in a
True
>>> # positive index
>>> a[0]
1
>>> # negative index
>>> a[-1]
5
>>> # slicing list[start:end:step]
>>> a[1:]
[2, 3, 4, 5]
>>> a[1:-1]
[2, 3, 4]
>>> a[1:-1:2]
[2, 4]
>>> # reverse
>>> a[::-1]
[5, 4, 3, 2, 1]
>>> a[:0:-1]
[5, 4, 3, 2]
>>> # set an item
>>> a[0] = 0
>>> a
[0, 2, 3, 4, 5]
>>> # append items to list
>>> a.append(6)
>>> a
[0, 2, 3, 4, 5, 6]
>>> a.extend([7, 8, 9])
>>> a
[0, 2, 3, 4, 5, 6, 7, 8, 9]
>>> # delete an item
>>> del a[-1]
>>> a
[0, 2, 3, 4, 5, 6, 7, 8]
>>> # list comprehension
>>> b = [x for x in range(3)]
>>> b
[0, 1, 2]
>>> # add two lists
>>> a + b
[0, 2, 3, 4, 5, 6, 7, 8, 0, 1, 2]
```

2.5.2 Initialize

Generally speaking, we can create a list through `*` operator if the item in the list expression is an immutable object.

```
>>> a = [None] * 3
>>> a
[None, None, None]
>>> a[0] = "foo"
>>> a
['foo', None, None]
```

However, if the item in the list expression is a mutable object, the `*` operator will copy the reference of the item N times. In order to avoid this pitfall, we should use a list comprehension to initialize a list.

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

2.5.3 Copy

Assigning a list to a variable is a common pitfall. This assignment does not copy the list to the variable. The variable only refers to the list and increase the reference count of the list.

```
import sys
>>> a = [1, 2, 3]
>>> sys.getrefcount(a)
2
>>> b = a
>>> sys.getrefcount(a)
3
>>> b[2] = 123456  # a[2] = 123456
>>> b
[1, 2, 123456]
>>> a
[1, 2, 123456]
```

There are two types of copy. The first one is called *shallow copy* (non-recursive copy) and the second one is called *deep copy* (recursive copy). Most of the time, it is sufficient for us to copy a list by shallow copy. However, if a list is nested, we have to use a deep copy.

```
>>> # shallow copy
>>> a = [1, 2]
>>> b = list(a)
>>> b[0] = 123
>>> a
[1, 2]
>>> b
[123, 2]
>>> a = [[1], [2]]
>>> b = list(a)
>>> b[0][0] = 123
>>> a
[[123], [2]]
>>> b
[[123], [2]]
>>> # deep copy
>>> import copy
>>> a = [[1], [2]]
>>> b = copy.deepcopy(a)
>>> b[0][0] = 123
>>> a
[[1], [2]]
>>> b
[[123], [2]]
```

2.5.4 Using `slice`

Sometimes, our data may concatenate as a large segment such as packets. In this case, we will represent the range of data by using `slice` objects as explaining variables instead of using *slicing expressions*.

```
>>> icmp = (
...     b"080062988e2100005bff49c20005767c"
...     b"08090a0b0c0d0e0f1011121314151617"
...     b"18191a1b1c1d1e1f2021222324252627"
...     b"28292a2b2c2d2e2f3031323334353637"
... )
>>> head = slice(0, 32)
>>> data = slice(32, len(icmp))
>>> icmp[head]
b'080062988e2100005bff49c20005767c'
```

2.5.5 List Comprehensions

List comprehensions which was proposed in PEP 202 provides a graceful way to create a new list based on another list, sequence, or some object which is iterable. In addition, we can use this expression to substitute `map` and `filter` sometimes.

```
>>> [x for x in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [(lambda x: x**2)(i) for i in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [x for x in range(10) if x > 5]
[6, 7, 8, 9]
>>> [x if x > 5 else 0 for x in range(10)]
[0, 0, 0, 0, 0, 0, 6, 7, 8, 9]
>>> [(x, y) for x in range(3) for y in range(2)]
[(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1)]
```

2.5.6 Unpacking

Sometimes, we want to unpack our list to variables in order to make our code become more readable. In this case, we assign N elements to N variables as following example.

```
>>> arr = [1, 2, 3]
>>> a, b, c = arr
>>> a, b, c
(1, 2, 3)
```

Based on PEP 3132, we can use a single asterisk to unpack N elements to the number of variables which is less than N in Python 3.

```
>>> arr = [1, 2, 3, 4, 5]
>>> a, b, *c, d = arr
>>> a, b, d
(1, 2, 5)
>>> c
[3, 4]
```

2.5.7 Using `enumerate`

`enumerate` is a built-in function. It helps us to acquire indexes (or a count) and elements at the same time without using `range(len(list))`. Further information can be found on Looping Techniques.

```
>>> for i, v in enumerate(range(3)):
...     print(i, v)
...
0 0
1 1
2 2
>>> for i, v in enumerate(range(3), 1): # start = 1
...     print(i, v)
...
1 0
2 1
3 2
```

2.5.8 Zip Lists

zip enables us to iterate over items contained in multiple lists at a time. Iteration stops whenever one of the lists is exhausted. As a result, the length of the iteration is the same as the shortest list. If this behavior is not desired, we can use `itertools.zip_longest` in **Python 3** or `itertools.izip_longest` in **Python 2**.

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> list(zip(a, b))
[(1, 4), (2, 5), (3, 6)]
>>> c = [1]
>>> list(zip(a, b, c))
[(1, 4, 1)]
>>> from itertools import zip_longest
>>> list(zip_longest(a, b, c))
[(1, 4, 1), (2, 5, None), (3, 6, None)]
```

2.5.9 Filter Items

filter is a built-in function to assist us to remove unnecessary items. In **Python 2**, `filter` returns a list. However, in **Python 3**, `filter` returns an *iterable object*. Note that *list comprehension* or *generator expression* provides a more concise way to remove items.

```
>>> [x for x in range(5) if x > 1]
[2, 3, 4]
>>> l = ['1', '2', 3, 'Hello', 4]
>>> f = lambda x: isinstance(x, int)
>>> filter(f, l)
<filter object at 0x10bee2198>
>>> list(filter(f, l))
[3, 4]
>>> list((i for i in l if f(i)))
[3, 4]
```

2.5.10 Stacks

There is no need an additional data structure, stack, in Python because the `list` provides `append` and `pop` methods which enable us use a list as a stack.

```
>>> stack = []
>>> stack.append(1)
>>> stack.append(2)
>>> stack.append(3)
>>> stack
[1, 2, 3]
>>> stack.pop()
3
>>> stack.pop()
2
>>> stack
[1]
```

2.5.11 `in` Operation

We can implement the `__contains__` method to make a class do `in` operations. It is a common way for a programmer to emulate a membership test operations for custom classes.

```
class Stack:

    def __init__(self):
        self.__list = []

    def push(self, val):
        self.__list.append(val)

    def pop(self):
        return self.__list.pop()

    def __contains__(self, item):
        return True if item in self.__list else False

stack = Stack()
stack.push(1)
print(1 in stack)
print(0 in stack)
```

Example

```
python stack.py
True
False
```

2.5.12 Accessing Items

Making custom classes perform get and set operations like lists is simple. We can implement a `__getitem__` method and a `__setitem__` method to enable a class to retrieve and overwrite data by index. In addition, if we want to use the function, `len`, to calculate the number of elements, we can implement a `__len__` method.

```
class Stack:

    def __init__(self):
        self.__list = []

    def push(self, val):
        self.__list.append(val)

    def pop(self):
        return self.__list.pop()

    def __repr__(self):
        return "{}".format(self.__list)

    def __len__(self):
        return len(self.__list)

    def __getitem__(self, idx):
        return self.__list[idx]

    def __setitem__(self, idx, val):
        self.__list[idx] = val


stack = Stack()
stack.push(1)
stack.push(2)
print("stack:", stack)

stack[0] = 3
print("stack:", stack)
print("num items:", len(stack))
```

Example

```
$ python stack.py
stack: [1, 2]
stack: [3, 2]
num items: 2
```

2.5.13 Delegating Iterations

If a custom container class holds a list and we want iterations to work on the container, we can implement a `__iter__` method to delegate iterations to the list. Note that the method, `__iter__`, should return an *iterator object*, so we cannot return the list directly; otherwise, Python raises a `TypeError`.

```
class Stack:

    def __init__(self):
        self.__list = []

    def push(self, val):
        self.__list.append(val)

    def pop(self):
```

(continues on next page)

(continued from previous page)

```
        return self.__list.pop()

    def __iter__(self):
        return iter(self.__list)

stack = Stack()
stack.push(1)
stack.push(2)
for s in stack:
    print(s)
```

Example

```
$ python stack.py
1
2
```

2.6 Set

2.6.1 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])
```

2.6.2 Uniquify a 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]
```

2.6.3 Union Two Sets

```
>>> 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]
```

(continues on next page)

(continued from previous page)

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

2.6.4 Append Items to a Set

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

2.6.5 Intersection Two Sets

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

2.6.6 Common Items from Sets

```
>>> a = [1, 1, 2, 3]
>>> b = [1, 3, 5, 5, 6, 6]
>>> com = list(set(a) & set(b))
>>> com
[1, 3]
```

2.6.7 Contain

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
```

2.6.8 Set Diff

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

2.6.9 Symmetric diff

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

2.7 Dictionary

2.7.1 Get All Keys

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

2.7.2 Get Key and Value

```
>>> a = {"1":1, "2":2, "3":3}
>>> a.items()
```

2.7.3 Find Same Keys

```
>>> a = {"1":1, "2":2, "3":3}
>>> [_ 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']
[('1', 1), ('3', 3), ('2', 2)]
```

2.7.4 Update Dictionary

```
>>> a = {"1":1, "2":2, "3":3}
>>> b = {"2":2, "3":3, "4":4}
>>> a.update(b)
>>> a
{'1': 1, '3': 3, '2': 2, '4': 4}
```

2.7.5 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'}
```

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'}
```

2.7.6 Emulating a Dictionary

```
>>> class EmuDict(object):
...   def __init__(self, dict_):
...     self._dict = dict_
...   def __repr__(self):
...     return "EmuDict: " + repr(self._dict)
...   def __getitem__(self, key):
...     return self._dict[key]
...   def __setitem__(self, key, val):
...     self._dict[key] = val
...   def __delitem__(self, key):
...     del self._dict[key]
...   def __contains__(self, key):
...     return key in self._dict
...   def __iter__(self):
...     return iter(self._dict.keys())
...
>>> _ = {"1":1, "2":2, "3":3}
>>> emud = EmuDict(_)
>>> emud  # __repr__
EmuDict: {'1': 1, '2': 2, '3': 3}
>>> emud['1']  # __getitem__
1
>>> emud['5'] = 5  # __setitem__
>>> emud
EmuDict: {'1': 1, '2': 2, '3': 3, '5': 5}
>>> del emud['2']  # __delitem__
>>> emud
```

(continues on next page)

(continued from previous page)

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

2.8 Function

Table of Contents

2.8.1 Document Functions

Define a function document

```
>>> def example():
...   """This is an example function."""
...   print("Example function")
...
>>> example.__doc__
'This is an example function.'
>>> help(example)
```

2.8.2 Default Arguments

```
>>> def add(a, b=0):
...     return a + b
...
>>> add(1)
1
>>> add(1, 2)
3
>>> add(1, b=2)
3
```

2.8.3 Option Arguments

```
>>> 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'}
```

2.8.4 Unpack Arguments

```
>>> def foo(a, b, c='BAZ'):
...     print(a, b, c)
...
>>> foo(*("FOO", "BAR"), **{"c": "baz"})
FOO BAR baz
```

2.8.5 Keyword-Only Arguments

New in Python 3.0

```
>>> def f(a, b, *, kw):
...     print(a, b, kw)
...
>>> f(1, 2, kw=3)
1 2 3
>>> 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
```

2.8.6 Annotations

New in Python 3.0

```
>>> def fib(n: int) -> int:
...     a, b = 0, 1
...     for _ in range(n):
...         b, a = a + b, b
...     return a
...
>>> fib(10)
55
>>> fib.__annotations__
{'n': <class 'int'>, 'return': <class 'int'>}
```

2.8.7 Callable

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

2.8.8 Get Function Name

```
>>> def example_function():
...   pass
...
>>> example_function.__name__
'example_function'
```

2.8.9 Lambda

```
>>> 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
```

2.8.10 Generator

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

(continues on next page)

(continued from previous page)

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

2.8.11 Decorator

New in Python 2.4

- PEP 318 - Decorators for Functions and Methods

```
>>> 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.
```

Equals to

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

2.8.12 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)
```

(continues on next page)

(continued from previous page)

```
... def example():
...     print("This is example function.")
...
>>> example()
Val is 10
This is example function.
```

Equals to

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

2.8.13 Cache

New in Python 3.2

Without Cache

```
>>> import time
>>> def fib(n):
...     if n < 2:
...         return n
...     return fib(n - 1) + fib(n - 2)
...
>>> s = time.time(); _ = fib(32); e = time.time(); e - s
1.1562161445617676
```

With Cache (dynamic programming)

```
>>> from functools import lru_cache
>>> @lru_cache(maxsize=None)
... def fib(n):
...     if n < 2:
...         return n
...     return fib(n - 1) + fib(n - 2)
...
>>> s = time.time(); _ = fib(32); e = time.time(); e - s
2.9087066650390625e-05
>>> fib.cache_info()
CacheInfo(hits=30, misses=33, maxsize=None, currsize=33)
```

2.9 Classes and Objects

2.9.1 List Attributes

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

2.9.2 Get Instance Type

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

2.9.3 Declare a Class

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

Equals 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
```

2.9.4 Has / Get / Set Attributes

```
>>> class Example(object):
...   def __init__(self):
...     self.name = "ex"
...   def printex(self):
...     print("This is an example")
...
>>> ex = Example()
>>> hasattr(ex,"name")
True
>>> hasattr(ex,"printex")
True
>>> hasattr(ex,"print")
False
>>> getattr(ex,'name')
'ex'
```

(continues on next page)

(continued from previous page)

```
>>> setattr(ex,'name','example')
>>> ex.name
'example'
```

2.9.5 Check Inheritance

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

2.9.6 Get Class Name

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

2.9.7 New and Init

`__init__` will be invoked

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

`__init__` won't be 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
```

2.9.8 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
```

2.9.9 Representation of a Class

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

2.9.10 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!
```

2.9.11 Context Manager

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

import socket

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

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

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

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

2.9.12 Using contextlib

```
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()
```

2.9.13 Property

```
>>> 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
```

Equals 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)
...
```

2.9.14 Computed Attributes

`@property` computes a value of a attribute only when we need. Not store in memory previously.

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

(continues on next page)

(continued from previous page)

```
>>> ex.square3
8
```

2.9.15 Descriptor

```
>>> 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
```

2.9.16 Static and Class Methond

`@classmethod` is bound to a class. `@staticmethod` is similar to a python function but define in a class.

```
>>> class example(object):
...   @classmethod
...   def clsmethod(cls):
...     print("I am classmethod")
...   @staticmethod
...   def stmethod():
...     print("I am staticmethod")
...   def instmethod(self):
```

(continues on next page)

(continued from previous page)

```
...     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() ...
```

2.9.17 Abstract Method

`abc` is used to 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 common way is to `raise NotImplementedError`

```
>>> class base(object):
...   def absmethod(self):
...     raise NotImplementedError
...
>>> class example(base):
...   def absmethod(self):
...     print("abstract")
...
>>> ex = example()
>>> ex.absmethod()
abstract
```

2.9.18 Using slot to Save Memory

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

import resource
import platform
import functools


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

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


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

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


class D(object):

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


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


alloc(S)
alloc(D)
```

output:

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

2.9.19 Common Magic

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

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

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

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

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

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

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

# arithmetical operation related
__add__(self, other)
__sub__(self, other)
__mul__(self, other)
__div__(self, other)
```

(continues on next page)

(continued from previous page)

```
__mod__(self, other)
__and__(self, other)
__or__(self, other)
__xor__(self, other)
```

2.10 Generator

Table of Contents

2.10.1 Glossary of Generator

```
# generator function

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

# generator
#
# calling the generator function returns a generator

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

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

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

2.10.2 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
```

(continues on next page)

(continued from previous page)

```
...         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 >>>
```

2.10.3 Unpacking Generators

```
# PEP 448

# unpacking inside a list

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

# unpacking inside a set

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

# unpacking to variables

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

(continues on next page)

(continued from previous page)

```
# unpacking inside a function

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

2.10.4 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 >>>
```

2.10.5 Send message to generator

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

2.10.6 `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'
```

2.10.7 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
```

(continues on next page)

(continued from previous page)

```
...     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
```

2.10.8 Generate sequences

```
# get a list via generator

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

# equivalent to

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

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

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

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

2.10.10 `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
```

(continues on next page)

(continued from previous page)

```
# equal to

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

2.10.11 Check generator type

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

2.10.12 Check Generator State

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

2.10.13 Simple compiler

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

import re
import types
from collections import namedtuple

tokens = [
```

(continues on next page)

(continued from previous page)

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

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

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


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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
    return expr()


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

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

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

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

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


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

output:

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

2.10.14 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
```

2.10.15 What `@contextmanager` actually doing?

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

class GeneratorCM(object):

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
        print("exit scope")

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

output:

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

2.10.16 profile code block

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

2.10.17 `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
```

(continues on next page)

(continued from previous page)

```
>>> 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
```

2.10.18 `yield from == await` expression

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

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

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

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

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

async def handler(conn):
```

(continues on next page)

(continued from previous page)

```
    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
```

2.10.19 Closure in Python - using generator

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

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

(continues on next page)

(continued from previous page)

```
>>> 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
```

2.10.20 Implement a simple scheduler

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

2.10.21 Simple round-robin with blocking

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

from collections import deque
from select import select
import socket

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

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

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

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

(continues on next page)

(continued from previous page)

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

2.10.22 simple round-robin with blocking and non-blocking

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

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

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

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

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

tasks.append(g_fib(15))

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

(continues on next page)

(continued from previous page)

```
    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()
```

2.10.23 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
```

2.10.24 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")
```

(continues on next page)

(continued from previous page)

```
...
>>> 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',)
```

2.10.25 send value and throw exception into async generator

```
# Need python-3.6 or above

>>> import asyncio
>>> async def agen(n, t=0.1):
...     try:
...         for x in range(n):
...             await asyncio.sleep(t)
...             val = yield x
...             print(f'get val: {val}')
...     except RuntimeError as e:
...         await asyncio.sleep(t)
...         yield repr(e)
...
>>> async def main(n):
...     g = agen(n)
...     ret = await g.asend(None) + await g.asend('foo')
...     print(ret)
...     ret = await g.athrow(RuntimeError('Get RuntimeError'))
...     print(ret)
...
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(main(5))
get val: foo
1
RuntimeError('Get RuntimeError',)
```

2.10.26 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
...
```

(continues on next page)

(continued from previous page)

```
>>> 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
```

2.10.27 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}")
...
```

(continues on next page)

(continued from previous page)

```
>>> 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
```

2.10.28 Asynchronous Comprehensions

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

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

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

# await in Comprehensions

>>> async def foo(t):
...     await asyncio.sleep(t)
...     return "foo"
...
>>> async def bar(t):
...     await asyncio.sleep(t)
...     return "bar"
...
>>> async def baz(t):
...     await asyncio.sleep(t)
...     return "baz"
...
>>> async def gen(*f, t=0.1):
...     for x in f:
...         await asyncio.sleep(t)
...         yield x
...
>>> async def await_simple_task():
...     ret = [await f(0.1) for f in [foo, bar]]
```

(continues on next page)

(continued from previous page)

```
...     print(ret)
...     ret = {await f(0.1) for f in [foo, bar]}
...     print(ret)
...     ret = {f.__name__: await f(0.1) for f in [foo, bar]}
...     print(ret)
...
>>> async def await_other_task():
...     ret = [await f(0.1) for f in [foo, bar] if await baz(1)]
...     print(ret)
...     ret = {await f(0.1) for f in [foo, bar] if await baz(1)}
...     print(ret)
...     ret = {f.__name__: await f(0.1) for f in [foo, bar] if await baz(1)}
...     print(ret)
...
>>> async def await_aiter_task():
...     ret = [await f(0.1) async for f in gen(foo, bar)]
...     print(ret)
...     ret = {await f(0.1) async for f in gen(foo, bar)}
...     print(ret)
...     ret = {f.__name__: await f(0.1) async for f in gen(foo, bar)}
...     print(ret)
...     ret = [await f(0.1) async for f in gen(foo, bar) if await baz(1)]
...     print(ret)
...     ret = {await f(0.1) async for f in gen(foo, bar) if await baz(1)}
...     print(ret)
...     ret = {f.__name__: await f(0.1) async for f in gen(foo, bar) if await baz(1)}
...
>>> import asyncio
>>> asyncio.get_event_loop()
>>> loop.run_until_complete(await_simple_task())
['foo', 'bar']
{'bar', 'foo'}
{'foo': 'foo', 'bar': 'bar'}
>>> loop.run_until_complete(await_other_task())
['foo', 'bar']
{'bar', 'foo'}
{'foo': 'foo', 'bar': 'bar'}
>>> loop.run_until_complete(await_gen_task())
['foo', 'bar']
{'bar', 'foo'}
{'foo': 'foo', 'bar': 'bar'}
['foo', 'bar']
{'bar', 'foo'}
{'foo': 'foo', 'bar': 'bar'}
```

2.11 Typing

Table of Contents

2.11.1 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):
```

(continues on next page)

(continued from previous page)

```
  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
```

2.11.2 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"
```

2.11.3 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,
```

(continues on next page)

(continued from previous page)

```
)

# 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'

# cound 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)
var_deque: Deque = deque([1, 2, 3])
var_nametuple: 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"}
```

2.11.4 Functions

```
from typing import Generator, Callable

# function
```

(continues on next page)

(continued from previous page)

```
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
```

2.11.5 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]

    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"))
```

2.11.6 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))
```

2.11.7 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))

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

2.11.8 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())
```

2.11.9 Asynchronous Context 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
```

(continues on next page)

(continued from previous page)

```
    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())
```

2.11.10 Avoid None access

```
import re

from typing import Pattern, Dict, Optional

# 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"
```

2.11.11 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"
```

2.11.12 Multiple return values

```
from typing import Tuple, Iterable, Union

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

# 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
```

2.11.13 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])
```

2.11.14 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
```

(continues on next page)

(continued from previous page)

```
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"
```

2.11.15 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
```

2.11.16 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"]
    ) -> 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
```

2.11.17 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.

2.11.18 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.]
```

2.11.19 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

# 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

2.11.20 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`

```python
from typing import TypeVar

T = TypeVar("T")

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

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

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

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

```cpp
#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

```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")
```

(continues on next page)

(continued from previous page)

2.11.22 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.

```python
from typing import TypeVar, Generic

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

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.

```python
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"
```

2.11.23 Restricting to a fixed set of possible types

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

```python
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
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"
```

2.11.24 `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"
```

2.11.25 @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.

```python
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`.

```python
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
```

2.11.26 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"
```

2.12 Files and I/O

Table of Contents

2.12.1 Read a File

In Python 2, the content of the file which read from file system does not decode. That is, the content of the file is a byte string, not a Unicode string.

```
>>> with open("/etc/passwd") as f:
...     content = f.read()
>>> print(type(content))
<type 'str'>
>>> print(type(content.decode("utf-8")))
<type 'unicode'>
```

In Python 3, open provides `encoding` option. If files do not open in binary mode, the encoding will be determined by `locale.getpreferredencoding(False)` or user's input.

```
>>> with open("/etc/hosts", encoding="utf-8") as f:
...     content = f.read()
...
>>> print(type(content))
<class 'str'>
```

Binary mode

```
>>> with open("/etc/hosts", "rb") as f:
...     content = f.read()
```

(continues on next page)

(continued from previous page)

```
...
>>> print(type(content))
<class 'bytes'>
```

2.12.2 Readline

```
>>> with open("/etc/hosts") as f:
...     for line in f:
...         print(line, end='')
...
127.0.0.1       localhost
255.255.255.255     broadcasthost
::1             localhost
```

2.12.3 Reading File Chunks

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

2.12.4 Write a File

```
>>> content = "Awesome Python!"
>>> with open("foo.txt", "w") as f:
...     f.write(content)
```

2.12.5 Create a Symbolic Link

```
>>> import os
>>> os.symlink("foo", "bar")
>>> os.readlink("bar")
'foo'
```

2.12.6 Copy a File

```
>>> from distutils.file_util import copy_file
>>> copy_file("foo", "bar")
('bar', 1)
```

2.12.7 Move a File

```
>>> from distutils.file_util import move_file
>>> move_file("./foo", "./bar")
'./bar'
```

2.12.8 List a Directory

```
>>> >>> import os
>>> dirs = os.listdir(".")
```

After Python 3.6, we can use `os.scandir` to list a directory. It is more convenient because `os.scandir` return an iterator of `os.DirEntry` objects. In this case, we can get file information through access the attributes of `os.DirEntry`. Further information can be found on the document.

```
>>> with os.scandir("foo") as it:
...     for entry in it:
...         st = entry.stat()
...
```

2.12.9 Create Directories

Similar to `mkdir -p /path/to/dest`

```
>>> from distutils.dir_util import mkpath
>>> mkpath("foo/bar/baz")
['foo', 'foo/bar', 'foo/bar/baz']
```

2.12.10 Copy a Directory

```
>>> from distutils.dir_util import copy_tree
>>> copy_tree("foo", "bar")
['bar/baz']
```

2.12.11 Remove a Directory

```
>>> from distutils.dir_util import remove_tree
>>> remove_tree("dir")
```

2.12.12 Path Join

```
>>> from pathlib import Path
>>> p = Path("/Users")
>>> p = p / "Guido" / "pysheeet"
>>> p
PosixPath('/Users/Guido/pysheeet')
```

2.12.13 Get Absolute Path

```
>>> from pathlib import Path
>>> p = Path("README.rst")
PosixPath('/Users/Guido/pysheeet/README.rst')
```

2.12.14 Get Home Directory

```
>>> from pathlib import Path
>>> Path.home()
PosixPath('/Users/Guido')
```

2.12.15 Get Current Directory

```
>>> from pathlib import Path
>>> p = Path("README.rst")
>>> p.cwd()
PosixPath('/Users/Guido/pysheeet')
```

2.12.16 Get Path Properties

```
>>> from pathlib import Path
>>> p = Path("README.rst").absolute()
>>> p.root
'/'
>>> p.anchor
'/'
>>> p.parent
PosixPath('/Users/Guido/pysheeet')
>>> p.parent.parent
PosixPath('/Users/Guido')
>>> p.name
'README.rst'
>>> p.suffix
'.rst'
>>> p.stem
'README'
>>> p.as_uri()
'file:///Users/Guido/pysheeet/README.rst'
```

CHAPTER 3

Advanced Cheat Sheet

The goal of this part is to give common snippets including built-in and 3rd party modules usages.

3.1 Regular Expression

Table of Contents

3.1.1 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('<[^/>][^>]*>', '<img src="/img">') != None
True
>>> re.search('<[^/>][^>]*>', '</table>') != None
False

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

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

3.1.2 `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")
```

3.1.3 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'
```

3.1.4 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', '/')
```

3.1.5 Back Reference

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

(continues on next page)

(continued from previous page)

```
# 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
```

3.1.6 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>
```

3.1.7 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'
```

3.1.8 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'
```

3.1.9 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
```

3.1.10 Match hex color value

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

3.1.11 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>
```

3.1.12 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')
```

3.1.13 Match IP address

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

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

3.1.14 Match Mac address

```
>>> import random
>>> mac = [random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f)]
>>> mac = ':'.join(map(lambda x: "%02x" % x, mac))
>>> mac
'3c:38:51:05:03:1e'
```

(continues on next page)

(continued from previous page)

```
>>> 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
```

3.1.15 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')
```

3.2 Socket

Table of Contents

3.2.1 Get Hostname

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

(continues on next page)

(continued from previous page)

```
'127.0.0.1'
```

3.2.2 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)
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)
```

3.2.3 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
```

(continues on next page)

(continued from previous page)

```
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
```

3.2.4 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'
```

3.2.5 Mac address & byte format convert

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

3.2.6 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):
```

(continues on next page)

(continued from previous page)

```
        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
```

3.2.7 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
```

3.2.8 Disable IPv6 Only

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

import contextlib
import socket

host = "::"
port = 5566

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


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

            if msg:
                conn.send(msg)

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

output:

```
$ python3 ipv6.py
[1] 23914
$ nc -4 127.0.0.1 5566
('::ffff:127.0.0.1', 42604, 0, 0)
Hello IPv4
Hello IPv4
$ nc -6 ::1 5566
('::1', 50882, 0, 0)
```

(continues on next page)

(continued from previous page)

```
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
```

3.2.9 Simple TCP Echo Server Via SocketServer

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

output:

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

3.2.10 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
```

3.2.11 Set ciphers on TLS/SSL TCP Echo Server

```python
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",
```

(continues on next page)

(continued from previous page)

```
    "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
```

3.2.12 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
```

3.2.13 Simple UDP Echo Server Via SocketServer

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

(continues on next page)

(continued from previous page)

```
...     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
```

3.2.14 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
```

3.2.15 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
```

3.2.16 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)
        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
```

3.2.17 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)
```

(continues on next page)

(continued from previous page)

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

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

output:

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

3.2.18 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
```

3.2.19 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:
    while True:
        r, w, _ = select(rl,wl,[])
        # process ready to ready
        for _ in r:
            if _ == sock:
                conn, addr = sock.accept()
                rl.append(conn)
            else:
                msg = _.recv(1024)
                ml[_.fileno()] = msg
                wl.append(_)
        # process ready to write
        for _ in w:
            msg = ml[_.fileno()]
            _.send(msg)
            wl.remove(_)
            del ml[_.fileno()]
except:
    sock.close()
```

output: (bash 1)

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

output: (bash 2)

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

3.2.20 Simple Asynchronous TCP Server - poll

```
from __future__ import print_function, unicode_literals

import socket
import select
import contextlib

host = 'localhost'
```

(continues on next page)

(continued from previous page)

```
port = 5566

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

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


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


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


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

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

    del req[fd]
```

(continues on next page)

(continued from previous page)

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

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

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

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

        poll.register(server.fileno())

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

output: (bash 1)

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

output: (bash 2)

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

3.2.21 Simple Asynchronous TCP Server - epoll

```
from __future__ import print_function, unicode_literals

import socket
import select
import contextlib


host = 'localhost'
port = 5566

con = {}
req = {}
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)
```

(continues on next page)

(continued from previous page)

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

    del req[fd]


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

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

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


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

        epoll.register(server.fileno())

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

output: (bash 1)

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

output: (bash 2)

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

3.2.22 Simple Asynchronous TCP Server - kqueue

```python
from __future__ import print_function, unicode_literals

import socket
import select
import contextlib

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


host = 'localhost'
port = 5566

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

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


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


def accept(server, kq):
    conn, addr = server.accept()
    conn.setblocking(False)
```

(continues on next page)

(continued from previous page)

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


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

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

    del req[fd]


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

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

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

(continues on next page)

(continued from previous page)

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


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

        max_events = 1024
        timeout = 1

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

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

output: (bash 1)

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

output: (bash 2)

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

3.2.23 High-Level API - selectors

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

(continues on next page)

(continued from previous page)

```
import contextlib

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

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

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

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

output: (bash 1)

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

output: (bash 1)

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

3.2.24 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:
        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)
```

(continues on next page)

(continued from previous page)

```
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
```

3.2.25 “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)
```

(continues on next page)

(continued from previous page)

```
    os.wait()
else:
    # child process
    p_s.close()
    while True:
        msg = c_s.recv(1024)
        print(msg)
        c_s.sendall("Hi! Parent!")
```

output:

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

3.2.26 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
```

(continues on next page)

(continued from previous page)

```
$ python3 sendfile.py dd.in dd.out
$ md5sum dd.in
e79afdd6aba71b7174142c0bbc289674  dd.in
$ md5sum dd.out
e79afdd6aba71b7174142c0bbc289674  dd.out
```

3.2.27 Sending a file through sendfile

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

import os
import sys
import time
import socket
import contextlib

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


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


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


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

        if not data: break

        fout.write(data)
```

(continues on next page)

(continued from previous page)

```
host = 'localhost'
port = 5566

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

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

pid = os.fork()

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

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

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

output:

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

3.2.28 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:
```

(continues on next page)

(continued from previous page)

```
        s.bind((typ, name))
        yield s
    finally:
        s.close()

msg = b'Python is awesome!'

with create_alg('hash', 'sha256') as algo:
    op, _ = algo.accept()
    with op:
        op.sendall(msg)
        data = op.recv(512)
        print(data.hex())

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

output:

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

3.2.29 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)
```

(continues on next page)

(continued from previous page)

```
            op.sendmsg_afalg([plaintext],
                             op=socket.ALG_OP_ENCRYPT,
                             iv=iv)
            ciphertext = op.recv(len(plaintext))

    return ciphertext


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

    return upad(plaintext)


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

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

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

output:

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

3.2.30 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()
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

    return ciphertext, tag


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

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

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

(continues on next page)

(continued from previous page)

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

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

    return plaintext

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

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

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

output:

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

3.2.31 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)
```

(continues on next page)

(continued from previous page)

```
    totalbytes = st.st_size

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

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

            op.sendall(assoc, socket.MSG_MORE)

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

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

    return ciphertext, tag


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

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

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

    return plaintext

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

(continues on next page)

(continued from previous page)

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

plain = sys.argv[1]

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

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

output:

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

3.2.32 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)
```

(continues on next page)

(continued from previous page)

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

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

    return ciphertext, tag


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

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

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

    return plaintext


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

count = 1000000
plain = "tmp.rand"

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


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

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

    dec_algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_KEY, key)
    dec_algo.setsockopt(socket.SOL_ALG,
                        socket.ALG_SET_AEAD_AUTHSIZE,
```

(continues on next page)

(continued from previous page)

```
                        None,
                        assoclen)

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

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

    with enc_op, dec_op:

        s = time.time()

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

        cost = time.time() - s

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


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

    aesgcm = AESGCM(key)

    s = time.time()

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

    cost = time.time() - s

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

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

output:

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

3.2.33 Sniffer IP packets

```
from ctypes import *
import socket
import struct

# ref: IP protocol numbers
```

(continues on next page)

(continued from previous page)

```
PROTO_MAP = {
        1 : "ICMP",
        2 : "IGMP",
        6 : "TCP",
        17: "UDP",
        27: "RDP"}

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

    In linux api, it define as below:

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

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

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

(continues on next page)

(continued from previous page)

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

output: (bash 1)

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

output: (bash 2)

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

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

3.2.34 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|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
```

(continues on next page)

(continued from previous page)

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

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

struct tcphdr {
    __be16  source;
    __be16  dest;
    __be32  seq;
    __be32  ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u16   res1:4,
            doff:4,
            fin:1,
            syn:1,
            rst:1,
            psh:1,
            ack:1,
            urg:1,
            ece:1,
            cwr:1;
#elif defined(__BIG_ENDIAN_BITFIELD)
    __u16   doff:4,
            res1:4,
            cwr:1,
            ece:1,
            urg:1,
            ack:1,
            psh:1,
            rst:1,
            syn:1,
            fin:1;
#else
#error      "Adjust your <asm/byteorder.h> defines"
#endif
    __be16  window;
    __sum16 check;
    __be16  urg_ptr;
};
"""
import sys
import socket
import platform

from struct import unpack
from contextlib import contextmanager

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

@contextmanager
def create_socket():
    ''' Create a TCP raw socket '''
    s = socket.socket(socket.AF_INET,
```

(continues on next page)

(continued from previous page)

```
                          socket.SOCK_RAW,
                          socket.IPPROTO_TCP)
    try:
        yield s
    finally:
        s.close()


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

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

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

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

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

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

            if not data:
                continue

            print("------------ TCP_HEADER --------------")
            print(f"Source Port:             {source}")
            print(f"Destination Port:        {dest}")
            print(f"Sequence Number:         {seq}")
            print(f"Acknowledgment Number: {ack_seq}")
            print(f"Data offset:             {doff}")
            print(f"FIN:                     {fin}")
            print(f"SYN:                     {syn}")
            print(f"RST:                     {rst}")
```

(continues on next page)

(continued from previous page)

```
            print(f"PSH:                    {psh}")
            print(f"ACK:                    {ack}")
            print(f"URG:                    {urg}")
            print(f"ECE:                    {ece}")
            print(f"CWR:                    {cwr}")
            print(f"Window:                 {window}")
            print(f"Checksum:               {check}")
            print(f"Urgent Point:           {urg_ptr}")
            print("--------------- DATA -----------------")
            print(data)

except KeyboardInterrupt:
    pass
```

output:

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

3.2.35 Sniffer ARP packet

```
"""
Ehternet Packet Header

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

ARP Packet Header

struct arphdr {
    uint16_t htype;    /* Hardware Type           */
    uint16_t ptype;    /* Protocol Type           */
    u_char   hlen;     /* Hardware Address Length */
```

(continues on next page)

(continued from previous page)

```
    u_char   plen;     /* Protocol Address Length */
    uint16_t opcode;   /* Operation Code          */
    u_char   sha[6];   /* Sender hardware address */
    u_char   spa[4];   /* Sender IP address       */
    u_char   tha[6];   /* Target hardware address */
    u_char   tpa[4];   /* Target IP address       */
};
"""

import socket
import struct
import binascii

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

while True:

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

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

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

output:

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

(continues on next page)

(continued from previous page)

```
Opcode:           0001
Source MAC:       f0257252f5ca
Source IP:        140.112.91.254
Dest MAC:         000000000000
Dest IP:          140.112.91.20
--------------------------------------------
```

3.3 Asyncio

Table of Contents

3.3.1 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'))
```

3.3.2 Future like object

```
>>> import sys
>>> PY_35 = sys.version_info >= (3, 5)
>>> import asyncio
>>> loop = asyncio.get_event_loop()
>>> class SlowObj:
...     def __init__(self, n):
...         print("__init__")
...         self._n = n
...     if PY_35:
...         def __await__(self):
...             print("__await__ sleep({})".format(self._n))
...             yield from asyncio.sleep(self._n)
...             print("ok")
...             return self
...
>>> async def main():
...     obj = await SlowObj(3)
...
>>> loop.run_until_complete(main())
__init__
__await__ sleep(3)
ok
```

3.3.3 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):
```

(continues on next page)

(continued from previous page)

```
...             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
```

3.3.4 Patch loop runner `_run_once`

```
>>> import asyncio
>>> def _run_once(self):
...     num_tasks = len(self._scheduled)
...     print("num tasks in queue: {}".format(num_tasks))
...     super(asyncio.SelectorEventLoop, self)._run_once()
...
>>> EventLoop = asyncio.SelectorEventLoop
>>> EventLoop._run_once = _run_once
>>> loop = EventLoop()
>>> asyncio.set_event_loop(loop)
>>> async def task(n):
...     await asyncio.sleep(n)
...     print("sleep: {} sec".format(n))
...
>>> coro = loop.create_task(task(3))
>>> loop.run_until_complete(coro)
num tasks in queue: 0
num tasks in queue: 1
num tasks in queue: 0
sleep: 3 sec
num tasks in queue: 0
>>> loop.close()
```

3.3.5 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)
```

3.3.6 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
```

3.3.7 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):
```

(continues on next page)

(continued from previous page)

```
    yield 'read_wait', s

@asyncio.coroutine
def write_wait(s):
    yield 'write_wait', s

class Loop:
    """Simple loop prototype"""

    def __init__(self):
        self.ready = deque()
        self.selector = selectors.DefaultSelector()

    @asyncio.coroutine
    def sock_accept(self, s):
        yield from read_wait(s)
        return s.accept()

    @asyncio.coroutine
    def sock_recv(self, c, mb):
        yield from read_wait(c)
        return c.recv(mb)

    @asyncio.coroutine
    def sock_sendall(self, c, m):
        while m:
            yield from write_wait(c)
            nsent = c.send(m)
            m = m[nsent:]

    def create_task(self, coro):
        self.ready.append(coro)

    def run_forever(self):
        while True:
            self._run_once()

    def _run_once(self):
        while not self.ready:
            events = self.selector.select()
            for k, _ in events:
                self.ready.append(k.data)
                self.selector.unregister(k.fileobj)

        while self.ready:
            self.cur_t = ready.popleft()
            try:
                op, *a = self.cur_t.send(None)
                getattr(self, op)(*a)
            except StopIteration:
                pass

    def read_wait(self, s):
        self.selector.register(s, selectors.EVENT_READ, self.cur_t)

    def write_wait(self, s):
        self.selector.register(s, selectors.EVENT_WRITE, self.cur_t)
```

(continues on next page)

(continued from previous page)

```
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()
```

3.3.8 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:
```

(continues on next page)

(continued from previous page)

```
    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
```

3.3.9 Transport and Protocol with SSL

```
import asyncio
import ssl


def make_header():
    head = b"HTTP/1.1 200 OK\r\n"
    head += b"Content-Type: text/html\r\n"
    head += b"\r\n"
    return head


def make_body():
    resp = b"<html>"
    resp += b"<h1>Hello SSL</h1>"
    resp += b"</html>"
    return resp


sslctx = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
sslctx.load_cert_chain(
    certfile="./root-ca.crt", keyfile="./root-ca.key"
)


class Service(asyncio.Protocol):
    def connection_made(self, tr):
        self.tr = tr
        self.total = 0

    def data_received(self, data):
        if data:
            resp = make_header()
            resp += make_body()
            self.tr.write(resp)
        self.tr.close()
```

(continues on next page)

(continued from previous page)

```
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
```

3.3.10 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
```

3.3.11 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
```

3.3.12 Asynchronous context manager

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

>>> class AsyncCtxMgr:
...     async def __aenter__(self):
...         await asyncio.sleep(3)
...         print("__anter__")
...         return self
...     async def __aexit__(self, *exc):
...         await asyncio.sleep(1)
...         print("__aexit__")
...
>>> async def hello():
...     async with AsyncCtxMgr() as m:
...         print("hello block")
...
>>> async def world():
...     print("world block")
...
>>> t = loop.create_task(world())
```

(continues on next page)

(continued from previous page)

```
>>> loop.run_until_complete(hello())
world block
__anter__
hello block
__aexit__
```

3.3.13 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__
```

3.3.14 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()
>>> loop.run_until_complete(main())
Hello
done
```

3.3.15 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'
```

3.3.16 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)
```

3.3.17 Gather Results

```
import asyncio
import ssl


path = ssl.get_default_verify_paths()
sslctx = ssl.SSLContext()
sslctx.verify_mode = ssl.CERT_REQUIRED
sslctx.check_hostname = True
sslctx.load_verify_locations(path.cafile)
```

(continues on next page)

(continued from previous page)

```
async def fetch(host, port):
    r, w = await asyncio.open_connection(host, port, ssl=sslctx)
    req = "GET / HTTP/1.1\r\n"
    req += f"Host: {host}\r\n"
    req += "Connection: close\r\n"
    req += "\r\n"

    # send request
    w.write(req.encode())

    # recv response
    resp = ""
    while True:
        line = await r.readline()
        if not line:
            break
        line = line.decode("utf-8")
        resp += line

    # close writer
    w.close()
    await w.wait_closed()
    return resp


async def main():
    loop = asyncio.get_running_loop()
    url = ["python.org", "github.com", "google.com"]
    fut = [fetch(u, 443) for u in url]
    resps = await asyncio.gather(*fut)
    for r in resps:
        print(r.split("\r\n")[0])


asyncio.run(main())
```

output:

```
$ python fetch.py
HTTP/1.1 301 Moved Permanently
HTTP/1.1 200 OK
HTTP/1.1 301 Moved Permanently
```

3.3.18 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, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = loop.create_future()
    if registed:
        loop.remove_reader(fd)

    try:
        data, addr = sock.recvfrom(n_bytes)
    except (BlockingIOError, InterruptedError):
        loop.add_reader(fd, recvfrom, loop, sock, n_bytes, fut, True)
    else:
        fut.set_result((data, addr))
    return fut

def sendto(loop, sock, data, addr, fut=None, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = loop.create_future()
    if registed:
        loop.remove_writer(fd)
    if not data:
        return

    try:
        n = sock.sendto(data, addr)
    except (BlockingIOError, InterruptedError):
        loop.add_writer(fd, sendto, loop, sock, data, addr, fut, True)
    else:
        fut.set_result(n)
    return fut

async def udp_server(loop, sock):
    while True:
        data, addr = await recvfrom(loop, sock, 1024)
        n_bytes = await sendto(loop, sock, data, addr)

try:
    loop.run_until_complete(udp_server(loop, sock))
finally:
    loop.close()
```

output:

```
$ python3 udp_server.py
$ nc -u localhost 3553
Hello UDP
Hello UDP
```

3.3.19 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
```

3.3.20 Simple HTTPS Web Server

```
import asyncio
import ssl

ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
```

(continues on next page)

(continued from previous page)

```
ctx.load_cert_chain('crt.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))
```

3.3.21 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')


def do_handshake(loop, sock, waiter):
    sock_fd = sock.fileno()
```

(continues on next page)

(continued from previous page)

```
    try:
        sock.do_handshake()
    except ssl.SSLWantReadError:
        loop.remove_reader(sock_fd)
        loop.add_reader(sock_fd, do_handshake,
                        loop, sock, waiter)
        return
    except ssl.SSLWantWriteError:
        loop.remove_writer(sock_fd)
        loop.add_writer(sock_fd, do_handshake,
                        loop, sock, waiter)
        return

    loop.remove_reader(sock_fd)
    loop.remove_writer(sock_fd)
    waiter.set_result(None)


def handle_read(loop, conn, waiter):
    try:
        req = conn.recv(1024)
    except ssl.SSLWantReadError:
        loop.remove_reader(conn.fileno())
        loop.add_reader(conn.fileno(), handle_read,
                        loop, conn, waiter)
        return
    loop.remove_reader(conn.fileno())
    waiter.set_result(req)


def handle_write(loop, conn, msg, waiter):
    try:
        resp = make_header()
        resp += make_body()
        ret = conn.send(resp)
    except ssl.SSLWantReadError:
        loop.remove_writer(conn.fileno())
        loop.add_writer(conn.fileno(), handle_write,
                        loop, conn, waiter)
        return
    loop.remove_writer(conn.fileno())
    conn.close()
    waiter.set_result(None)


async def server(loop):
    while True:
        conn, addr = await loop.sock_accept(sock)
        conn.setblocking(False)
        sslconn = sslctx.wrap_socket(conn,
                                     server_side=True,
                                     do_handshake_on_connect=False)
        # wait SSL handshake
        waiter = loop.create_future()
        do_handshake(loop, sslconn, waiter)
        await waiter
```

(continues on next page)

(continued from previous page)

```
        # 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
```

3.3.22 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)


async def main():
    paths = ssl.get_default_verify_paths()
    sslctx = ssl.SSLContext()
```

(continues on next page)

(continued from previous page)

```
    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
```

3.3.23 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)
        writer.close()

async def main(host, port):
    # run a simplle http server
```

(continues on next page)

(continued from previous page)

```
    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>
```

3.3.24 Simple asyncio WSGI web server

```
# ref: PEP333

import asyncio
import socket
import io
import sys

from flask import Flask, Response

host = 'localhost'
port = 9527
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.setblocking(False)
s.bind((host, port))
s.listen(10)

loop = asyncio.get_event_loop()

class WSGIServer(object):

    def __init__(self, sock, app):
        self._sock = sock
        self._app = app
        self._header = []

    def parse_request(self, req):
        """ HTTP Request Format:

        GET /hello.htm HTTP/1.1\r\n
        Accept-Language: en-us\r\n
        ...
        Connection: Keep-Alive\r\n
        """
        # bytes to string
        req_info = req.decode('utf-8')
        first_line = req_info.splitlines()[0]
        method, path, ver = first_line.split()
        return method, path, ver
```

(continues on next page)

(continued from previous page)

```
    def get_environ(self, req, method, path):
        env = {}

        # Required WSGI variables
        env['wsgi.version']      = (1, 0)
        env['wsgi.url_scheme']   = 'http'
        env['wsgi.input']        = req
        env['wsgi.errors']       = sys.stderr
        env['wsgi.multithread']  = False
        env['wsgi.multiprocess'] = False
        env['wsgi.run_once']     = False

        # Required CGI variables
        env['REQUEST_METHOD']    = method    # GET
        env['PATH_INFO']         = path      # /hello
        env['SERVER_NAME']       = host      # localhost
        env['SERVER_PORT']       = str(port) # 9527
        return env

    def start_response(self, status, resp_header, exc_info=None):
        header = [('Server', 'WSGIServer 0.2')]
        self.headers_set = [status, resp_header + header]

    async def finish_response(self, conn, data, headers):
        status, resp_header = headers

        # make header
        resp = 'HTTP/1.1 {0}\r\n'.format(status)
        for header in resp_header:
            resp += '{0}: {1}\r\n'.format(*header)
        resp += '\r\n'

        # make body
        resp += '{0}'.format(data)
        try:
            await loop.sock_sendall(conn, str.encode(resp))
        finally:
            conn.close()

    async def run_server(self):
        while True:
            conn, addr = await loop.sock_accept(self._sock)
            loop.create_task(self.handle_request(conn))

    async def handle_request(self, conn):
        # get request data
        req = await loop.sock_recv(conn, 1024)
        if req:
            method, path, ver = self.parse_request(req)
            # get environment
            env = self.get_environ(req, method, path)
            # get application execute result
            res = self._app(env, self.start_response)
            res = [_.decode('utf-8') for _ in list(res)]
            res = ''.join(res)
            loop.create_task(
```

(continues on next page)

(continued from previous page)

```
                    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
```

3.4 Concurrency

Table of Contents

3.4.1 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
```

3.4.2 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
```

3.4.3 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
```

3.4.4 Consumer and Producer

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

(continues on next page)

(continued from previous page)

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

3.4.5 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()
```

3.4.6 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
```

3.4.7 Mutex lock

Simplest synchronization primitive lock

```
>>> from threading import Thread
>>> from threading import Lock
>>> lock = Lock()
>>> def getlock(id):
...   lock.acquire()
...   print("task{0} get".format(id))
```

(continues on next page)

(continued from previous page)

```
...     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
```

3.4.8 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
```

3.4.9 Implement "Monitor"

Using RLock

```python
# 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
```

3.4.10 Control primitive resources

Using Semaphore

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

# limit resource to 3
sema = Semaphore(3)
def foo(tid):
    with sema:
        print("%d acquire sema" % tid)
        wt = random()*5
        time.sleep(wt)
    print("%d release sema" % tid)

threads = []
```

(continues on next page)

(continued from previous page)

```
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
```

3.4.11 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
```

(continues on next page)

(continued from previous page)

```
 3 get event set
1 get event set
```

3.4.12 Thread-safe priority queue

Using 'condition'

```
import threading
import heapq
import time
import random

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

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

    def __repr__(self):
        return self._q

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

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

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

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

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

t1=threading.Thread(target=producer)
```

(continues on next page)

(continued from previous page)

```
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...
```

3.4.13 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
```

3.4.14 Custom multiprocessing map

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

(continues on next page)

(continued from previous page)

```
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)))
```

3.4.15 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)
```

3.4.16 Simple round-robin scheduler

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

(continues on next page)

(continued from previous page)

```
...
>>> 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
```

3.4.17 Scheduler with blocking function

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

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

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

def run():
    while any([tasks,r_wait,s_wait]):
        while not tasks:
            # polling
            rr, sr, _ = select(r_wait, s_wait, {})
            for _ in rr:
                tasks.append(r_wait.pop(_))
            for _ in sr:
                tasks.append(s_wait.pop(_))
```

(continues on next page)

(continued from previous page)

```
        try:
            task = tasks.popleft()
            why,what = task.next()
            if why == 'recv':
                r_wait[what] = task
            elif why == 'send':
                s_wait[what] = task
            else:
                raise RuntimeError
        except StopIteration:
            pass

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

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

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

output: (bash 1)

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

output: (bash 2)

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

3.4.18 PoolExecutor

```
# python2.x is module futures on PyPI
# new in Python3.2
>>> from concurrent.futures import \
...     ThreadPoolExecutor
```

(continues on next page)

(continued from previous page)

```
>>> def fib(n):
...     if n<=2:
...         return 1
...     return fib(n-1) + fib(n-2)
...
>>> with ThreadPoolExecutor(3) as e:
...     res= e.map(fib,[1,2,3,4,5])
...     for _ in res:
...         print(_, end=' ')
...
1 1 2 3 5 >>>
# result is generator?!
>>> with ThreadPoolExecutor(3) as e:
...   res = e.map(fib, [1,2,3])
...   inspect.isgenerator(res)
...
True

# demo GIL
from concurrent import futures
import time

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

def thread():
    s = time.time()
    with futures.ThreadPoolExecutor(2) as e:
        res = e.map(fib, [35]*2)
        for _ in res:
            print(_)
    e = time.time()
    print("thread cost: {}".format(e-s))

def process():
    s = time.time()
    with futures.ProcessPoolExecutor(2) as e:
        res = e.map(fib, [35]*2)
        for _ in res:
            print(_)
    e = time.time()
    print("pocess cost: {}".format(e-s))


# bash> python3 -i test.py
>>> thread()
9227465
9227465
thread cost: 12.550225019454956
>>> process()
9227465
9227465
pocess cost: 5.538189888000488
```

3.4.19 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]
```

3.4.20 What does "with ThreadPoolExecutor" work?

```
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
```

3.4.21 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
```

3.4.22 Future error handling

```
from concurrent import futures

def spam():
    raise RuntimeError

def handler(future):
```

(continues on next page)

(continued from previous page)

```
    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
```

3.5 SQLAlchemy

Table of Contents

3.5.1 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
```

3.5.2 Sqlalchemy Support DBAPI - PEP249

```
from sqlalchemy import create_engine

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

# DBAPI - PEP249
```

(continues on next page)

(continued from previous page)

```
# create table
engine.execute('CREATE TABLE "EX1" ('
               'id INTEGER NOT NULL,'
               'name VARCHAR, '
               'PRIMARY KEY (id));')
# insert a raw
engine.execute('INSERT INTO "EX1" '
               '(id, name) '
               'VALUES (1,"raw1")')

# select *
result = engine.execute('SELECT * FROM '
                        '"EX1"')
for _r in result:
   print(_r)

# delete *
engine.execute('DELETE from "EX1" where id=1;')
result = engine.execute('SELECT * FROM "EX1"')
print(result.fetchall())
```

3.5.3 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()
```

3.5.4 Metadata - Generating Database Schema

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# Create a metadata instance
metadata = MetaData(engine)
# Declare a table
```

(continues on next page)

(continued from previous page)

```
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)
```

3.5.5 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'))
```

3.5.6 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)
```

3.5.7 Get Table from MetaData

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

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

(continues on next page)

(continued from previous page)

```
# Create MetaData instance
metadata = MetaData(engine, reflect=True)
print(metadata.tables)

# Get Table
ex_table = metadata.tables['Example']
print(ex_table)
```

3.5.8 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()
```

3.5.9 Create Specific Table

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer, String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

meta = MetaData(engine)
t1 = Table('Table_1', meta,
           Column('id', Integer, primary_key=True),
           Column('name',String))
t2 = Table('Table_2', meta,
           Column('id', Integer, primary_key=True),
           Column('val',Integer))
t1.create()
```

3.5.10 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)
```

3.5.11 Drop a Table

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import inspect
from sqlalchemy import Table
from sqlalchemy import Column, Integer, String
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))
m = MetaData()
table = Table('Test', m,
              Column('id', Integer, primary_key=True),
              Column('key', String, nullable=True),
              Column('val', String))

```

(continues on next page)

(continued from previous page)

```
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
```

3.5.12 Some Table Object Operation

```python
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)
```

3.5.13 SQL Expression Language

```python
# 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_

```

(continues on next page)

(continued from previous page)

```
meta = MetaData()
table = Table('example', meta,
              Column('id', Integer, primary_key=True),
              Column('l_name', String),
              Column('f_name', String))
# sql expression binary object
print(repr(table.c.l_name == 'ed'))
# exhbit sql expression
print(str(table.c.l_name == 'ed'))

print(repr(table.c.f_name != 'ed'))

# comparison operator
print(repr(table.c.id > 3))

# or expression
print((table.c.id > 5) | (table.c.id < 2))
# Equal to
print(or_(table.c.id > 5, table.c.id < 2))

# compare to None produce IS NULL
print(table.c.l_name == None)
# Equal to
print(table.c.l_name.is_(None))

# + means "addition"
print(table.c.id + 5)
# or means "string concatenation"
print(table.c.l_name + "some name")

# in expression
print(table.c.l_name.in_(['a','b']))
```

3.5.14 insert() - Create an "INSERT" Statement

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import Column
from sqlalchemy import Integer
from sqlalchemy import String

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)

# create table
meta = MetaData(engine)
table = Table('user', meta,
   Column('id', Integer, primary_key=True),
   Column('l_name', String),
   Column('f_name', String))
meta.create_all()

# insert data via insert() construct
ins = table.insert().values(
```

(continues on next page)

(continued from previous page)

```
      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'}])
```

3.5.15 select() - Create a "SELECT" Statement

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Table
from sqlalchemy import select
from sqlalchemy import or_

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
conn = engine.connect()

meta = MetaData(engine,reflect=True)
table = meta.tables['user']

# select * from 'user'
select_st = select([table]).where(
   table.c.l_name == 'Hello')
res = conn.execute(select_st)
for _row in res:
    print(_row)

# or equal to
select_st = table.select().where(
   table.c.l_name == 'Hello')
res = conn.execute(select_st)
for _row in res:
    print(_row)

# combine with "OR"
select_st = select([
   table.c.l_name,
   table.c.f_name]).where(or_(
      table.c.l_name == 'Hello',
      table.c.l_name == 'Hi'))
res = conn.execute(select_st)
for _row in res:
    print(_row)

# combine with "ORDER_BY"
select_st = select([table]).where(or_(
      table.c.l_name == 'Hello',
      table.c.l_name == 'Hi')).order_by(table.c.f_name)
res = conn.execute(select_st)
for _row in res:
```

(continues on next page)

(continued from previous page)

```
    print(_row)
```

3.5.16 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)
```

3.5.17 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']
```

(continues on next page)

(continued from previous page)

```
# 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)
```

3.5.18 Check Table Existing

```
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import Column
from sqlalchemy import Integer, String
from sqlalchemy import inspect
from sqlalchemy.ext.declarative import declarative_base

Modal = declarative_base()
class Example(Modal):
   __tablename__ = "ex_t"
   id = Column(Integer, primary_key=True)
   name = Column(String(20))

db_uri = 'sqlite:///db.sqlite'
engine = create_engine(db_uri)
Modal.metadata.create_all(engine)

# check register table exist to Modal
for _t in Modal.metadata.tables:
    print(_t)

# check all table in database
meta = MetaData(engine, reflect=True)
for _t in meta.tables:
    print(_t)

# check table names exists via inspect
ins = inspect(engine)
for _t in ins.get_table_names():
    print(_t)
```

3.5.19 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']
```

3.5.20 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}
```

(continues on next page)

(continued from previous page)

```
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
```

3.5.21 Object Relational add data

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))

Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)
```

(continues on next page)

(continued from previous page)

```
# 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()
```

3.5.22 Object Relational update data

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))
Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

try:
```

(continues on next page)

(continued from previous page)

```
    # 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
```

3.5.23 Object Relational delete row

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, DateTime
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL


db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}
engine = create_engine(URL(**db_url))
Base = declarative_base()

class TestTable(Base):
    __tablename__ = 'Test Table'
    id   = Column(Integer, primary_key=True)
    key  = Column(String, nullable=False)
    val  = Column(String)
    date = Column(DateTime, default=datetime.utcnow)
```

(continues on next page)

(continued from previous page)

```
# 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>
[]
```

3.5.24 Object Relational relationship

```
from sqlalchemy import Column, String, Integer, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class User(Base):
    __tablename__ = 'user'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    addresses = relationship("Address", backref="user")

class Address(Base):
    __tablename__ = 'address'
    id = Column(Integer, primary_key=True)
    email = Column(String)
    user_id = Column(Integer, ForeignKey('user.id'))

u1 = User()
a1 = Address()
print(u1.addresses)
print(a1.user)

u1.addresses.append(a1)
print(u1.addresses)
print(a1.user)
```

output:

```
$ python sqlalchemy_relationship.py
[]
None
[<__main__.Address object at 0x10c4edb50>]
<__main__.User object at 0x10c4ed810>
```

3.5.25 Object Relational self association

```python
import json

from sqlalchemy import (
    Column,
    Integer,
    String,
    ForeignKey,
    Table)

from sqlalchemy.orm import (
    sessionmaker,
    relationship)

from sqlalchemy.ext.declarative import declarative_base

base = declarative_base()

association = Table("Association", base.metadata,
    Column('left', Integer, ForeignKey('node.id'), primary_key=True),
    Column('right', Integer, ForeignKey('node.id'), primary_key=True))

class Node(base):
    __tablename__ = 'node'
    id = Column(Integer, primary_key=True)
    label = Column(String)
    friends = relationship('Node',
                           secondary=association,
                           primaryjoin=id==association.c.left,
                           secondaryjoin=id==association.c.right,
                           backref='left')
    def to_json(self):
        return dict(id=self.id,
                    friends=[_.label for _ in self.friends])

nodes = [Node(label='node_{}'.format(_)) for _ in range(0, 3)]
nodes[0].friends.extend([nodes[1], nodes[2]])
nodes[1].friends.append(nodes[2])

print('----> right')
print(json.dumps([_.to_json() for _ in nodes], indent=2))

print('----> left')
print(json.dumps([_n.to_json() for _n in nodes[1].left], indent=2))
```

output:

```
----> right
```

(continues on next page)

(continued from previous page)

```
[
  {
    "friends": [
      "node_1",
      "node_2"
    ],
    "id": null
  },
  {
    "friends": [
      "node_2"
    ],
    "id": null
  },
  {
    "friends": [],
    "id": null
  }
]
----> left
[
  {
    "friends": [
      "node_1",
      "node_2"
    ],
    "id": null
  }
]
```

3.5.26 Object Relational basic query

```
from datetime import datetime

from sqlalchemy import create_engine
from sqlalchemy import Column, String, Integer, DateTime
from sqlalchemy import or_
from sqlalchemy import desc
from sqlalchemy.orm import sessionmaker
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.engine.url import URL

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

Base = declarative_base()

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

(continues on next page)

(continued from previous page)

```
    fullname = Column(String, nullable=False)
    birth    = Column(DateTime)

# create tables
engine = create_engine(URL(**db_url))
Base.metadata.create_all(bind=engine)

users = [
    User(name='ed',
         fullname='Ed Jones',
         birth=datetime(1989,7,1)),
    User(name='wendy',
         fullname='Wendy Williams',
         birth=datetime(1983,4,1)),
    User(name='mary',
         fullname='Mary Contrary',
         birth=datetime(1990,1,30)),
    User(name='fred',
         fullname='Fred Flinstone',
         birth=datetime(1977,3,12)),
    User(name='justin',
         fullname="Justin Bieber")]

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

# add_all
session.add_all(users)
session.commit()

print("----> order_by(id):")
query = session.query(User).order_by(User.id)
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> order_by(desc(id)):")
query = session.query(User).order_by(desc(User.id))
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> order_by(date):")
query = session.query(User).order_by(User.birth)
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)

print("\n----> EQUAL:")
query = session.query(User).filter(User.id == 2)
_row = query.first()
print(_row.name, _row.fullname, _row.birth)

print("\n----> NOT EQUAL:")
query = session.query(User).filter(User.id != 2)
for _row in query.all():
    print(_row.name, _row.fullname, _row.birth)
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
mary Mary Contrary 1990-01-30 00:00:00
justin Justin Bieber None

----> EQUAL:
wendy Wendy Williams 1983-04-01 00:00:00

----> NOT EQUAL:
ed Ed Jones 1989-07-01 00:00:00
mary Mary Contrary 1990-01-30 00:00:00
fred Fred Flinstone 1977-03-12 00:00:00
justin Justin Bieber None

----> IN:
ed Ed Jones 1989-07-01 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00

----> NOT IN:
mary Mary Contrary 1990-01-30 00:00:00
fred Fred Flinstone 1977-03-12 00:00:00
justin Justin Bieber None

----> AND:
ed Ed Jones 1989-07-01 00:00:00

----> OR:
ed Ed Jones 1989-07-01 00:00:00
wendy Wendy Williams 1983-04-01 00:00:00

----> NULL:
justin Justin Bieber

----> NOT NULL:
ed Ed Jones
wendy Wendy Williams
mary Mary Contrary
fred Fred Flinstone

----> LIKE
ed Ed Jones
fred Fred Flinstone
```

3.5.27 mapper: Map `Table` to `class`

```
from sqlalchemy import (
    create_engine,
    Table,
    MetaData,
    Column,
    Integer,
    String,
    ForeignKey)

from sqlalchemy.orm import (
    mapper,
    relationship,
```

(continues on next page)

(continued from previous page)

```
    sessionmaker)

# classical mapping: map "table" to "class"
db_url = 'sqlite://'
engine = create_engine(db_url)

meta = MetaData(bind=engine)

user = Table('User', meta,
             Column('id', Integer, primary_key=True),
             Column('name', String),
             Column('fullname', String),
             Column('password', String))

addr = Table('Address', meta,
             Column('id', Integer, primary_key=True),
             Column('email', String),
             Column('user_id', Integer, ForeignKey('User.id')))

# map table to class
class User(object):
    def __init__(self, name, fullname, password):
        self.name = name
        self.fullname = fullname
        self.password = password

class Address(object):
    def __init__(self, email):
        self.email = email

mapper(User, user, properties={
       'addresses': relationship(Address, backref='user')})
mapper(Address, addr)

# create table
meta.create_all()

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

u = User(name='Hello', fullname='HelloWorld', password='ker')
a = Address(email='hello@hello.com')
u.addresses.append(a)
try:
    session.add(u)
    session.commit()

    # query result
    u = session.query(User).filter(User.name == 'Hello').first()
    print(u.name, u.fullname, u.password)

finally:
    session.close()
```

output:

```
$ python map_table_class.py
Hello HelloWorld ker
```

3.5.28 Get table dynamically

```
from sqlalchemy import (
    create_engine,
    MetaData,
    Table,
    inspect,
    Column,
    String,
    Integer)

from sqlalchemy.orm import (
    mapper,
    scoped_session,
    sessionmaker)

db_url = "sqlite://"
engine = create_engine(db_url)
metadata = MetaData(engine)

class TableTemp(object):
    def __init__(self, name):
        self.name = name

def get_table(name):
    if name in metadata.tables:
        table = metadata.tables[name]
    else:
        table = Table(name, metadata,
                Column('id', Integer, primary_key=True),
                Column('name', String))
        table.create(engine)

    cls = type(name.title(), (TableTemp,), {})
    mapper(cls, table)
    return cls

# get table first times
t = get_table('Hello')

# get table secone times
t = get_table('Hello')

Session = scoped_session(sessionmaker(bind=engine))
try:
    Session.add(t(name='foo'))
    Session.add(t(name='bar'))
    for _ in Session.query(t).all():
        print(_.name)
except Exception as e:
    Session.rollback()
finally:
    Session.close()
```

output:

```
$ python get_table.py
foo
bar
```

3.5.29 Object Relational join two tables

```python
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.engine.url import URL
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class User(Base):
    __tablename__ = 'user'
    id    = Column(Integer, primary_key=True)
    name  = Column(String)
    addresses = relationship("Address", backref="user")

class Address(Base):
    __tablename__ = 'address'
    id = Column(Integer, primary_key=True)
    email = Column(String)
    user_id = Column(Integer, ForeignKey('user.id'))

db_url = {'drivername': 'postgres',
          'username': 'postgres',
          'password': 'postgres',
          'host': '192.168.99.100',
          'port': 5432}

# create engine
engine = create_engine(URL(**db_url))

# create tables
Base.metadata.create_all(bind=engine)

# create session
Session = sessionmaker()
Session.configure(bind=engine)
session = Session()

user = User(name='user1')
mail1 = Address(email='user1@foo.com')
mail2 = Address(email='user1@bar.com')
user.addresses.extend([mail1, mail2])

session.add(user)
session.add_all([mail1, mail2])
session.commit()

query = session.query(Address, User).join(User)
```

(continues on next page)

(continued from previous page)

```
for _a, _u in query.all():
    print(_u.name, _a.email)
```

output:

```
$ python sqlalchemy_join.py
user1 user1@foo.com
user1 user1@bar.com
```

3.5.30 join on relationship and group_by count

```
from sqlalchemy import (
    create_engine,
    Column,
    String,
    Integer,
    ForeignKey,
    func)

from sqlalchemy.orm import (
    relationship,
    sessionmaker,
    scoped_session)

from sqlalchemy.ext.declarative import declarative_base

db_url = 'sqlite://'
engine = create_engine(db_url)

Base = declarative_base()

class Parent(Base):
    __tablename__ = 'parent'
    id       = Column(Integer, primary_key=True)
    name     = Column(String)
    children = relationship('Child', back_populates='parent')

class Child(Base):
    __tablename__ = 'child'
    id        = Column(Integer, primary_key=True)
    name      = Column(String)
    parent_id = Column(Integer, ForeignKey('parent.id'))
    parent    = relationship('Parent', back_populates='children')

Base.metadata.create_all(bind=engine)
Session = scoped_session(sessionmaker(bind=engine))

p1 = Parent(name="Alice")
p2 = Parent(name="Bob")

c1 = Child(name="foo")
c2 = Child(name="bar")
c3 = Child(name="ker")
c4 = Child(name="cat")
```

(continues on next page)

(continued from previous page)

```
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
```

3.5.31 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', {
```

(continues on next page)

(continued from previous page)

```
                '__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
```

3.5.32 Close database connection

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

from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

engine = create_engine('sqlite://')
base = declarative_base()

@event.listens_for(engine, 'engine_disposed')
def receive_engine_disposed(engine):
    print("engine dispose")

class Table(base):
    __tablename__ = 'example table'
    id = Column(Integer, primary_key=True)

base.metadata.create_all(bind=engine)
session = sessionmaker(bind=engine)()

try:
    try:
        row = Table()
        session.add(row)
    except Exception as e:
        session.rollback()
        raise
    finally:
        session.close()
finally:
    engine.dispose()
```

output:

```
$ python db_dispose.py
engine dispose
```

Warning: Be careful. Close *session* does not mean close database connection. SQLAlchemy *session* generally represents the *transactions*, not connections.

3.5.33 Cannot use the object after close the session

```python
from __future__ import print_function

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

from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base


url = 'sqlite://'
engine = create_engine(url)
base = declarative_base()

class Table(base):
    __tablename__ = 'table'
    id  = Column(Integer, primary_key=True)
    key = Column(String)
    val = Column(String)

base.metadata.create_all(bind=engine)
session = sessionmaker(bind=engine)()

try:
    t = Table(key="key", val="val")
    try:
        print(t.key, t.val)
        session.add(t)
        session.commit()
    except Exception as e:
        print(e)
        session.rollback()
    finally:
        session.close()

    print(t.key, t.val) # exception raise from here
except Exception as e:
    print("Cannot use the object after close the session")
finally:
    engine.dispose()
```

output:

```
$ python sql.py
key val
Cannot use the object after close the session
```

3.6 Security

Table of Contents

3.6.1 Simple https server

```
# python2

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

# python3

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

3.6.2 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,
    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)
```

3.6.3 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 \
>       -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
```

(continues on next page)

(continued from previous page)

```
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
```

3.6.4 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  = "pysheeet"
cert.get_subject().OU = "cheat sheet"
cert.get_subject().CN = "pythonsheets.com"
cert.set_serial_number(1000)
cert.set_notBefore(now.strftime("%Y%m%d%H%M%SZ").encode())
cert.set_notAfter(expire.strftime("%Y%m%d%H%M%SZ").encode())
cert.set_issuer(cert.get_subject())
cert.set_pubkey(k)
cert.sign(k, 'sha1')

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

output:

```
$ openssl genrsa -out key.pem 2048
Generating RSA private key, 2048 bit long modulus
.............+++
```

(continues on next page)

(continued from previous page)

```
...................................+++
e is 65537 (0x10001)
$ python3 x509.py
$ openssl x509 -subject -issuer -noout -in cert.pem
subject= /C=TW/ST=Taiwan/L=Taipei/O=pysheeet/OU=cheat sheet/CN=pythonsheets.com
issuer= /C=TW/ST=Taiwan/L=Taipei/O=pysheeet/OU=cheat sheet/CN=pythonsheets.com
```

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

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

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

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

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

output:

```
# create a root ca
$ openssl genrsa -out ca-key.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.......................................+++
e is 65537 (0x10001)
$ openssl req -x509 -new -nodes -key ca-key.pem \
> -days 10000 -out ca.pem -subj "/CN=root-ca"

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

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

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

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

3.6.6 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())
```

(continues on next page)

(continued from previous page)

```
...
>>> with open('cert.key', 'wb') as f:
...     f.write(key.private_bytes(
...     encoding=serialization.Encoding.PEM,
...     format=serialization.PrivateFormat.TraditionalOpenSSL,
...     encryption_algorithm=serialization.NoEncryption()))
```

3.6.7 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:

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

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

3.6.8 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)
```

(continues on next page)

(continued from previous page)

```
    signer = PKCS1_v1_5.new(rsakey)
    digest = SHA256.new()

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


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

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

output:

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

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

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

3.6.9 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
```

(continues on next page)

(continued from previous page)

```
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!
```

3.6.10 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)

# 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:
```

(continues on next page)

(continued from previous page)

```
    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!
```

3.6.11 Simple RSA decrypt via pem file

```python
from __future__ import print_function, unicode_literals

import base64
import sys

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

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

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

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

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

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

output:

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

3.6.12 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          |\
> python3 rsa.py          |\
> openssl base64 -d -A    |\
> openssl rsautl -decrypt -oaep -inkey private.key
Hello RSA OAEP!
```

3.6.13 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)
```

(continues on next page)

(continued from previous page)

```
# 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
```

3.6.14 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()

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)
```

(continues on next page)

(continued from previous page)

```
# 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!
```

3.6.15 Using AES CBC mode encrypt a file

```
from __future__ import print_function, unicode_literals

import struct
import sys
import os

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

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

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

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

    return ctext

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

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

(continues on next page)

(continued from previous page)

```
# 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
```

3.6.16 Using AES CBC mode decrypt a file

```
from __future__ import print_function, unicode_literals

import struct
import sys
import os

from binascii import unhexlify

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

backend = default_backend()

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

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

    return ptext

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

(continues on next page)

(continued from previous page)

```
# 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
```

3.6.17 AES CBC mode encrypt via password (using cryptography)

```
from __future__ import print_function, unicode_literals

import base64
import struct
import sys
import os

from hashlib import md5, sha1

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

backend = default_backend()

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


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

    # generate salt
    salt = os.urandom(8)

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

(continues on next page)

(continued from previous page)

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

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

    # encrypt plain text
    ctext = encryptor.update(ptext) + encryptor.finalize()
    ctext = b'Salted__' + salt + ctext

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


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

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

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

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

output:

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

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

3.6.18 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
```

(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()

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


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

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

    # get salt
    salt = ctext[8:16]

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

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

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

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

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

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

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

output:

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

(continues on next page)

(continued from previous page)

```
> 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
```

3.6.19 AES CBC mode encrypt via password (using pycrypto)

```python
from __future__ import print_function, unicode_literals

import struct
import base64
import sys

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

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

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

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


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

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

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

    # pad plaintext
    plaintext = pad(plaintext)

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

output:

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

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

3.6.20 AES CBC mode decrypt via password (using pycrytpo)

```
from __future__ import print_function, unicode_literals

import struct
import base64
import sys

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

    # get salt
    salt = ciphertext[8:16]

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

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


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

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

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

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

output:

```bash
# 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
```

3.6.21 Ephemeral Diffie Hellman Key Exchange via cryptography

```python
>>> 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)
```

(continues on next page)

(continued from previous page)

```
>>> a_shared_key == b_shared_key
True
```

3.6.22 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
```

3.6.23 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
```

3.7 Boto3

3.8 Test

Table of Contents

3.8.1 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()
```

(continues on next page)

(continued from previous page)

```
.
----------------------------------------------------------------------
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)
```

3.8.2 Python unittest setup & teardown hierarchy

```
from __future__ import print_function

import unittest

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

def setUpModule():
        print("setup module")
def tearDownModule():
        print("teardown module")

class TestFib(unittest.TestCase):

    def setUp(self):
        print("setUp")
        self.n = 10
    def tearDown(self):
        print("tearDown")
        del self.n
    @classmethod
    def setUpClass(cls):
        print("setUpClass")
    @classmethod
    def tearDownClass(cls):
        print("tearDownClass")
    def test_fib_assert_equal(self):
        self.assertEqual(fib(self.n), 55)
    def test_fib_assert_true(self):
```

(continues on next page)

(continued from previous page)

```
        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
```

3.8.3 Different module of setUp & tearDown hierarchy

```
# test_module.py
from __future__ import print_function

import unittest

class TestFoo(unittest.TestCase):
    @classmethod
    def setUpClass(self):
        print("foo setUpClass")
    @classmethod
    def tearDownClass(self):
        print("foo tearDownClass")
    def setUp(self):
        print("foo setUp")
    def tearDown(self):
        print("foo tearDown")
    def test_foo(self):
        self.assertTrue(True)

class TestBar(unittest.TestCase):
    def setUp(self):
        print("bar setUp")
    def tearDown(self):
        print("bar tearDown")
    def test_bar(self):
        self.assertTrue(True)

# test.py
from __future__ import print_function

from test_module import TestFoo
```

(continues on next page)

(continued from previous page)

```
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
```

3.8.4 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
```

3.8.5 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)
```

3.8.6 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
```

(continues on next page)

(continued from previous page)

```
ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

3.8.7 Group multiple testcases into a suite

```
>>> import unittest
>>> class TestFooBar(unittest.TestCase):
...     def test_foo(self):
...         self.assertTrue(True)
...     def test_bar(self):
...         self.assertTrue(True)
...
>>> class TestHelloWorld(unittest.TestCase):
...     def test_hello(self):
...         self.assertEqual("Hello", "Hello")
...     def test_world(self):
...         self.assertEqual("World", "World")
...
>>> suite_loader = unittest.TestLoader()
>>> suite1 = suite_loader.loadTestsFromTestCase(TestFooBar)
>>> suite2 = suite_loader.loadTestsFromTestCase(TestHelloWorld)
>>> suite = unittest.TestSuite([suite1, suite2])
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_bar (__main__.TestFooBar) ... ok
test_foo (__main__.TestFooBar) ... ok
test_hello (__main__.TestHelloWorld) ... ok
test_world (__main__.TestHelloWorld) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

3.8.8 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)
```

(continues on next page)

(continued from previous page)

```
test_foo (__main__.TestFoo) ... ok
test_bar (__main__.TestBar) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

3.8.9 Skip some tests in the TestCase

```
>>> import unittest
>>> RUN_FOO = False
>>> DONT_RUN_BAR = False
>>> class TestSkip(unittest.TestCase):
...     def test_always_run(self):
...         self.assertTrue(True)
...     @unittest.skip("always skip this test")
...     def test_always_skip(self):
...         raise RuntimeError
...     @unittest.skipIf(RUN_FOO == False, "demo skipIf")
...     def test_skipif(self):
...         raise RuntimeError
...     @unittest.skipUnless(DONT_RUN_BAR == True, "demo skipUnless")
...     def test_skipunless(self):
...         raise RuntimeError
...
>>> suite = unittest.TestLoader().loadTestsFromTestCase(TestSkip)
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_always_run (__main__.TestSkip) ... ok
test_always_skip (__main__.TestSkip) ... skipped 'always skip this test'
test_skipif (__main__.TestSkip) ... skipped 'demo skipIf'
test_skipunless (__main__.TestSkip) ... skipped 'demo skipUnless'

----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK (skipped=3)
```

3.8.10 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'):
```

(continues on next page)

(continued from previous page)

```
...                 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>
```

3.8.11 Cross-module variables to Test files

test_foo.py

```
from __future__ import print_function

import unittest

print(conf)

class TestFoo(unittest.TestCase):
    def test_foo(self):
        print(conf)

    @unittest.skipIf(conf.isskip==True, "skip test")
    def test_skip(self):
        raise RuntimeError
```

test_bar.py

```
from __future__ import print_function

import unittest
import __builtin__

if __name__ == "__main__":
    conf = type('TestConf', (object,), {})
    conf.isskip = True

    # make a cross-module variable
    __builtin__.conf = conf
    module = __import__('test_foo')
    loader = unittest.TestLoader()
```

(continues on next page)

(continued from previous page)

```
    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)
```

3.8.12 skip setup & teardown when the test is skipped

```
>>> from __future__ import print_function
>>> import unittest
>>> class TestSkip(unittest.TestCase):
...     def setUp(self):
...         print("setUp")
...     def tearDown(self):
...         print("tearDown")
...     @unittest.skip("skip this test")
...     def test_skip(self):
...         raise RuntimeError
...     def test_not_skip(self):
...         self.assertTrue(True)
...
>>> suite = unittest.TestLoader().loadTestsFromTestCase(TestSkip)
>>> unittest.TextTestRunner(verbosity=2).run(suite)
test_not_skip (__main__.TestSkip) ... setUp
tearDown
ok
test_skip (__main__.TestSkip) ... skipped 'skip this test'

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK (skipped=1)
```

3.8.13 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")
```

(continues on next page)

(continued from previous page)

```
...
>>> 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>
```

3.8.14 Testing your document is right

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

>>> fib(10)
55
"""

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

Example:

    >>> fib(10)
    55
    >>> fib(-1)
    Traceback (most recent call last):
    ...
    ValueError
"""
if n < 0:
    raise ValueError('')
return 1 if n<=2 else fib(n-1) + fib(n-2)

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

output:

```
$ python demo_doctest.py -v
Trying:
fib(10)
Expecting:
55
ok
```

(continues on next page)

(continued from previous page)

```
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.
```

3.8.15 Re-using doctest to unittest

```python
import unittest
import doctest

"""
This is an example of doctest

>>> fib(10)
55
"""

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

    Example:

        >>> fib(10)
        55
        >>> fib(-1)
        Traceback (most recent call last):
            ...
        ValueError
    """
    if n < 0:
        raise ValueError('')
    return 1 if n<=2 else fib(n-1) + fib(n-2)

if __name__ == "__main__":
    finder = doctest.DocTestFinder()
    suite = doctest.DocTestSuite(test_finder=finder)
    unittest.TextTestRunner(verbosity=2).run(suite)
```

output:

```
fib (__main__)
Doctest: __main__.fib ... ok

----------------------------------------------------------------------
Ran 1 test in 0.023s

OK
```

3.8.16 Customize test report

```
from unittest import (
        TestCase,
        TestLoader,
        TextTestResult,
        TextTestRunner)

from pprint import pprint
import unittest
import os

OK = 'ok'
FAIL = 'fail'
ERROR = 'error'
SKIP = 'skip'

class JsonTestResult(TextTestResult):

    def __init__(self, stream, descriptions, verbosity):
        super_class = super(JsonTestResult, self)
        super_class.__init__(stream, descriptions, verbosity)

        # TextTestResult has no successes attr
        self.successes = []

    def addSuccess(self, test):
        # addSuccess do nothing, so we need to overwrite it.
        super(JsonTestResult, self).addSuccess(test)
        self.successes.append(test)

    def json_append(self, test, result, out):
        suite = test.__class__.__name__
        if suite not in out:
            out[suite] = {OK: [], FAIL: [], ERROR:[], SKIP: []}
        if result is OK:
            out[suite][OK].append(test._testMethodName)
        elif result is FAIL:
            out[suite][FAIL].append(test._testMethodName)
        elif result is ERROR:
            out[suite][ERROR].append(test._testMethodName)
        elif result is SKIP:
            out[suite][SKIP].append(test._testMethodName)
        else:
            raise KeyError("No such result: {}".format(result))
        return out
```

(continues on next page)

(continued from previous page)

```
    def jsonify(self):
        json_out = dict()
        for t in self.successes:
            json_out = self.json_append(t, OK, json_out)

        for t, _ in self.failures:
            json_out = self.json_append(t, FAIL, json_out)

        for t, _ in self.errors:
            json_out = self.json_append(t, ERROR, json_out)

        for t, _ in self.skipped:
            json_out = self.json_append(t, SKIP, json_out)

        return json_out

class TestSimple(TestCase):

    def test_ok_1(self):
        foo = True
        self.assertTrue(foo)

    def test_ok_2(self):
        bar = True
        self.assertTrue(bar)

    def test_fail(self):
        baz = False
        self.assertTrue(baz)

    def test_raise(self):
        raise RuntimeError

    @unittest.skip("Test skip")
    def test_skip(self):
        raise NotImplementedError

if __name__ == '__main__':
    # redirector default output of unittest to /dev/null
    with open(os.devnull, 'w') as null_stream:
        # new a runner and overwrite resultclass of runner
        runner = TextTestRunner(stream=null_stream)
        runner.resultclass = JsonTestResult

        # create a testsuite
        suite = TestLoader().loadTestsFromTestCase(TestSimple)

        # run the testsuite
        result = runner.run(suite)

        # print json output
        pprint(result.jsonify())
```

output:

```
$ python test.py
{'TestSimple': {'error': ['test_raise'],
```

(continues on next page)

(continued from previous page)

```
                 'fail': ['test_fail'],
                 'ok': ['test_ok_1', 'test_ok_2'],
                 'skip': ['test_skip']}}
```

3.8.17 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('%$!?&*')
...     except OSError as e:
...         print(e)
...     else:
...         print('test success')
...
>>> test()
[Errno 2] No such file or directory: '%$!?&*'
```

3.8.18 What `with unittest.mock.patch` do?

```
from unittest.mock import patch
import os

PATH = '$@!%?&'

def fake_remove(path):
    print("Fake remove")

```

(continues on next page)

(continued from previous page)

```
class SimplePatch:

    def __init__(self, target, new):
        self._target = target
        self._new = new

    def get_target(self, target):
        target, attr = target.rsplit('.', 1)
        getter = __import__(target)
        return getter, attr

    def __enter__(self):
        orig, attr = self.get_target(self._target)
        self.orig, self.attr = orig, attr
        self.orig_attr = getattr(orig, attr)
        setattr(orig, attr, self._new)
        return self._new

    def __exit__(self, *exc_info):
        setattr(self.orig, self.attr, self.orig_attr)
        del self.orig_attr


print('---> inside unittest.mock.patch scope')
with patch('os.remove', fake_remove):
    os.remove(PATH)

print('---> inside simple patch scope')
with SimplePatch('os.remove', fake_remove):
    os.remove(PATH)

print('---> outside patch scope')
try:
    os.remove(PATH)
except OSError as e:
    print(e)
```

output:

```
$ python3 simple_patch.py
---> inside unittest.mock.patch scope
Fake remove
---> inside simple patch scope
Fake remove
---> outside patch scope
[Errno 2] No such file or directory: '$@!%?&'
```

3.8.19 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:
```

(continues on next page)

(continued from previous page)

```
...             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
```

3.9 C Extensions

Table of Contents

3.9.1 Simple setup.py

```
from distutils.core import setup, Extension

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

3.9.2 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])
```

3.9.3 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
};
```

3.9.4 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'
```

3.9.5 Release the GIL

```
#include <Python.h>

static PyObject* foo(PyObject* self)
{
    Py_BEGIN_ALLOW_THREADS
    sleep(3);
    Py_END_ALLOW_THREADS
    Py_RETURN_NONE;
}

static PyMethodDef methods[] = {
    {"foo", (PyCFunction)foo, METH_NOARGS, 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 threading
> import foo
> from datetime import datetime
> def f(n):
>     now = datetime.now()
>     print(f'{now}: thread {n}')
>     foo.foo()
> ts = [threading.Thread(target=f, args=(n,)) for n in range(3)]
> [t.start() for t in ts]
> [t.join() for t in ts]"
2018-11-04 20:15:34.860454: thread 0
2018-11-04 20:15:34.860592: thread 1
2018-11-04 20:15:34.860705: thread 2
```

In C extension, blocking I/O should be inserted into a block which is wrapped by `Py_BEGIN_ALLOW_THREADS` and `Py_END_ALLOW_THREADS` for releasing the GIL temporarily; Otherwise, a blocking I/O operation has to wait until previous operation finish. For example

```
#include <Python.h>

static PyObject* foo(PyObject* self)
{
    sleep(3);
    Py_RETURN_NONE;
}

```

(continues on next page)

(continued from previous page)

```
static PyMethodDef methods[] = {
    {"foo", (PyCFunction)foo, METH_NOARGS, 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 -c "
> import threading
> import foo
> from datetime import datetime
> def f(n):
>     now = datetime.now()
>     print(f'{now}: thread {n}')
>     foo.foo()
> ts = [threading.Thread(target=f, args=(n,)) for n in range(3)]
> [t.start() for t in ts]
> [t.join() for t in ts]"
2018-11-04 20:16:44.055932: thread 0
2018-11-04 20:16:47.059718: thread 1
2018-11-04 20:16:50.063579: thread 2
```

Warning: The GIL can only be safely released when there is **NO** Python C API functions between `Py_BEGIN_ALLOW_THREADS` and `Py_END_ALLOW_THREADS`.

3.9.6 Acquire the GIL

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

typedef struct {
    PyObject *sec;
    PyObject *py_callback;
} foo_args;

void *
foo_thread(void *args)
{
    long n = -1;
    PyObject *rv = NULL, *sec = NULL,* py_callback = NULL;
    foo_args *a = NULL;

    if (!args)
        return NULL;
```

(continues on next page)

(continued from previous page)

```
    a = (foo_args *)args;
    sec = a->sec;
    py_callback = a->py_callback;

    n = PyLong_AsLong(sec);
    if ((n == -1) && PyErr_Occurred()) {
        return NULL;
    }

    sleep(n);  // slow task

    // acquire the GIL
    PyGILState_STATE state = PyGILState_Ensure();
    rv = PyObject_CallFunction(py_callback, "s", "Awesome Python!");
    // release the GIL
    PyGILState_Release(state);
    Py_XDECREF(rv);
    return NULL;
}

static PyObject *
foo(PyObject *self, PyObject *args)
{
    long i = 0, n = 0;
    pthread_t *arr = NULL;
    PyObject *py_callback = NULL;
    PyObject *sec = NULL, *num = NULL;
    PyObject *rv = NULL;
    foo_args a = {};

    if (!PyArg_ParseTuple(args, "OOO:callback", &num, &sec, &py_callback))
        return NULL;

    // allow releasing GIL
    Py_BEGIN_ALLOW_THREADS

    if (!PyLong_Check(sec) || !PyLong_Check(num)) {
        PyErr_SetString(PyExc_TypeError, "should be int");
        goto error;
    }

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

    n = PyLong_AsLong(num);
    if (n == -1 && PyErr_Occurred())
        goto error;

    arr = (pthread_t *)PyMem_RawCalloc(n, sizeof(pthread_t));
    if (!arr)
        goto error;

    a.sec = sec;
    a.py_callback = py_callback;
```

(continues on next page)

(continued from previous page)

```
    for (i = 0; i < n; i++) {
        if (pthread_create(&arr[i], NULL, foo_thread, &a)) {
            PyErr_SetString(PyExc_TypeError, "create a thread failed");
            goto error;
        }
    }

    for (i = 0; i < n; i++) {
        if (pthread_join(arr[i], NULL)) {
            PyErr_SetString(PyExc_TypeError, "thread join failed");
            goto error;
        }
    }
    Py_XINCREF(Py_None);
    rv = Py_None;
error:
    PyMem_RawFree(arr);
    Py_XDECREF(sec);
    Py_XDECREF(num);
    Py_XDECREF(py_callback);
    // restore GIL
    Py_END_ALLOW_THREADS
    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
$ pyton -q
>>> import foo
>>> from datetime import datetime
>>> def cb(s):
...     now = datetime.now()
...     print(f'{now}: {s}')
...
>>> foo.foo(3, 1, cb)
2018-11-05 09:33:50.642543: Awesome Python!
2018-11-05 09:33:50.642634: Awesome Python!
2018-11-05 09:33:50.642672: Awesome Python!
```

If threads are created from C/C++, those threads do not hold the GIL. Without acquiring the GIL, the interpreter cannot access Python functions safely. For example

```c
void *
foo_thread(void *args)
{
    ...
    // without acquiring the GIL
    rv = PyObject_CallFunction(py_callback, "s", "Awesome Python!");
    Py_XDECREF(rv);
    return NULL;
}
```

output:

```
>>> import foo
>>> from datetime import datetime
>>> def cb(s):
...     now = datetime.now()
...     print(f"{now}: {s}")
...
>>> foo.foo(1, 1, cb)
[2]    8590 segmentation fault  python -q
```

Warning: In order to call python function safely, we can simply warp **Python Functions** between `PyGILState_Ensure` and `PyGILState_Release` in C extension code.

```c
PyGILState_STATE state = PyGILState_Ensure();
// Perform Python actions
result = PyObject_CallFunction(callback)
// Error handling
PyGILState_Release(state);
```

3.9.7 Get Reference Count

```c
#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
};

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
```

3.9.8 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)
{
    static char *keywords[] = {"x", "y", NULL};
    PyObject *x = NULL, *y = NULL;
    if (!PyArg_ParseTupleAndKeywords(args, kwargs,
                                     "O|O", keywords,
                                     &x, &y))
    {
        return NULL;
    }
    if (!y) {
```

(continues on next page)

(continued from previous page)

```
        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)
```

3.9.9 Calling Python Functions

```
#include <Python.h>

static PyObject *
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");
```

(continues on next page)

(continued from previous page)

```
        return NULL;
    }

    // Make sure we own the GIL
    PyGILState_STATE state = PyGILState_Ensure();
    // similar to py_callback("Awesome Python!")
    rv = PyObject_CallFunction(py_callback, "s", "Awesome Python!");
    // Restore previous GIL state
    PyGILState_Release(state);
    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!
```

3.9.10 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;
}

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
```

(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 -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
```

3.9.11 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)
{
    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);
```

(continues on next page)

(continued from previous page)

```
    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
```

3.9.12 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_XINCREF(Py_None);
    result = Py_None;
error:
    Py_XDECREF(iter);
    return result;
}

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

static struct PyModuleDef module = {
```

(continues on next page)

(continued from previous page)

```
    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
```

3.9.13 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;
error:
    return result;
}

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

static struct PyModuleDef module = {
```

(continues on next page)

(continued from previous page)

```
    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)'
```

3.9.14 Simple Class

```
#include <Python.h>

typedef struct {
    PyObject_HEAD
} FooObject;

/* calss 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)
{
    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'>
```

3.9.15 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 {
    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);
}
```

(continues on next page)

(continued from previous page)

```c
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_XINCREF(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;
        Py_XDECREF(ptr);
    }

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

(continues on next page)

(continued from previous page)

```
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;
    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
};

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
```

3.9.16 Simplie 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 {
    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;
```

(continues on next page)

(continued from previous page)

```
    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;
}

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)) {
```

(continues on next page)

(continued from previous page)

```
        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);
    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'
```

(continues on next page)

(continued from previous page)

```
>>> o.foo = None
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: value should be unicode
```

3.9.17 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);
}

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:
```

(continues on next page)

(continued from previous page)

```
    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;
    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 PyTypeObject FooType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "foo.Foo",
    .tp_doc = "Foo objects",
```

(continues on next page)

(continued from previous page)

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

static PyObject *
Bar_gcd(BarObject *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}
};
```

(continues on next page)

(continued from previous page)

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

3.9.18 Run a Python Command

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

(continues on next page)

(continued from previous page)

```
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
```

3.9.19 Run a Python File

```
#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;

    // 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:
```

(continues on next page)

(continued from previous page)

```
    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']
```

3.9.20 Import a Python 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 *json_module = NULL, *json_dict = 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;
    }

    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);
```

(continues on next page)

(continued from previous page)

```
    // 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}'
```

3.9.21 Import everything 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;
```

(continues on next page)

(continued from previous page)

```
    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);

    // 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'
```

3.9.22 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;
    }

    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:
```

(continues on next page)

(continued from previous page)

```
    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}'
```

3.9.23 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;
    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
```

(continues on next page)

(continued from previous page)

```
>>> 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
```

3.9.24 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)
...
>>> 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
```

3.9.25 ctypes Error handling

```
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:
```

(continues on next page)

(continued from previous page)

```
    raise RuntimeError("Not support platform: {}".format(platform))

stat = libc.stat

class Stat(Structure):
    """
    From /usr/include/sys/stat.h

    struct stat {
        dev_t          st_dev;
        ino_t          st_ino;
        mode_t         st_mode;
        nlink_t        st_nlink;
        uid_t          st_uid;
        gid_t          st_gid;
        dev_t          st_rdev;
    #ifndef _POSIX_SOURCE
        struct      timespec st_atimespec;
        struct      timespec st_mtimespec;
        struct      timespec st_ctimespec;
    #else
        time_t         st_atime;
        long           st_atimensec;
        time_t         st_mtime;
        long           st_mtimensec;
        time_t         st_ctime;
        long           st_ctimensec;
    #endif
        off_t          st_size;
        int64_t        st_blocks;
        u_int32_t      st_blksize;
        u_int32_t      st_flags;
        u_int32_t      st_gen;
        int32_t        st_lspare;
        int64_t        st_qspare[2];
    };
    """
    _fields_ = [
        ("st_dev", c_ulong),
        ("st_ino", c_ulong),
        ("st_mode", c_ushort),
        ("st_nlink", c_uint),
        ("st_uid", c_uint),
        ("st_gid", c_uint),
        ("st_rdev", c_ulong),
        ("st_atime", c_longlong),
        ("st_atimendesc", c_long),
        ("st_mtime", c_longlong),
        ("st_mtimendesc", c_long),
        ("st_ctime", c_longlong),
        ("st_ctimendesc", c_long),
        ("st_size", c_ulonglong),
        ("st_blocks", c_int64),
        ("st_blksize", c_uint32),
        ("st_flags", c_uint32),
        ("st_gen", c_uint32),
        ("st_lspare", c_int32),
```

(continues on next page)

(continued from previous page)

```
        ("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
```

CHAPTER 4

Appendix

The appendix mainly focuses on some critical concepts missing in cheat sheets.

4.1 Why does Decorator Need @wraps

`@wraps` preserve attributes of the original function, otherwise attributes of the decorated function will be replaced by **wrapper function**. For example

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
...
```

(continues on next page)

(continued from previous page)

```
>>> @decorator
... def example(*a, **kw):
...     pass
...
>>> example.__name__  # attr of function preserve
'example'
```

4.2 Yet Another Introduction to Asyncio

Table of Contents

4.2.1 What is 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")
```

(continues on next page)

(continued from previous page)

```
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
```

4.2.2 What is 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")
```

(continues on next page)

(continued from previous page)

```
@asyncio.coroutine
def bar():
    yield from asyncio.sleep(1)
    print("Hello Bar")

loop = asyncio.get_event_loop()
tasks = [Task(foo(), loop=loop),
         loop.create_task(bar())]
loop.run_until_complete(
        asyncio.wait(tasks))
loop.close()
```

output:

```
$ python test.py
Hello Bar
hello Foo
```

4.2.3 How does event loop work?

```
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:
```

(continues on next page)

(continued from previous page)

```
                self._run_once()
                if self._stopping:
                    break
        finally:
            self._stopping = False

    def call_soon(self, cb, *args):
        """Append task to ready queue"""
        self._ready.append((cb, args))
    def call_exception_handler(self, c):
        pass

    def _run_once(self):
        """Run task at once"""
        ntodo = len(self._ready)
        for i in range(ntodo):
            t, a = self._ready.popleft()
            t(*a)

    def stop(self):
        self._stopping = True

    def close(self):
        self._ready.clear()

    def get_debug(self):
        return False

@asyncio.coroutine
def foo():
    print("Foo")

@asyncio.coroutine
def bar():
    print("Bar")

loop = Loop()
tasks = [loop.create_task(foo()),
         loop.create_task(bar())]
loop.run_until_complete(
        asyncio.wait(tasks))
loop.close()
```

output:

```
$ python test.py
Foo
Bar
```

4.2.4 How does `asyncio.wait` work?

```
import asyncio

async def wait(fs, loop=None):
    fs = {asyncio.ensure_future(_) for _ in set(fs)}
```

(continues on next page)

(continued from previous page)

```python
    if loop is None:
        loop = asyncio.get_event_loop()

    waiter = loop.create_future()
    counter = len(fs)

    def _on_complete(f):
        nonlocal counter
        counter -= 1
        if counter <= 0 and not waiter.done():
             waiter.set_result(None)

    for f in fs:
        f.add_done_callback(_on_complete)

    # wait all tasks done
    await waiter

    done, pending = set(), set()
    for f in fs:
        f.remove_done_callback(_on_complete)
        if f.done():
            done.add(f)
        else:
            pending.add(f)
    return done, pending

async def slow_task(n):
    await asyncio.sleep(n)
    print('sleep "{}" sec'.format(n))

loop = asyncio.get_event_loop()

try:
    print("---> wait")
    loop.run_until_complete(
            wait([slow_task(_) for _ in range(1,3)]))
    print("---> asyncio.wait")
    loop.run_until_complete(
            asyncio.wait([slow_task(_) for _ in range(1,3)]))
finally:
    loop.close()
```

output:

```
---> wait
sleep "1" sec
sleep "2" sec
---> asyncio.wait
sleep "1" sec
sleep "2" sec
```

4.2.5 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))
```

4.2.6 How does `loop.sock_*` work?

```
import asyncio
import socket

def sock_accept(self, sock, fut=None, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = self.create_future()
    if registed:
        self.remove_reader(fd)
    try:
        conn, addr = sock.accept()
        conn.setblocking(False)
    except (BlockingIOError, InterruptedError):
        self.add_reader(fd, self.sock_accept, sock, fut, True)
    except Exception as e:
        fut.set_exception(e)
    else:
        fut.set_result((conn, addr))
    return fut

def sock_recv(self, sock, n , fut=None, registed=False):
    fd = sock.fileno()
    if fut is None:
        fut = self.create_future()
    if registed:
        self.remove_reader(fd)
    try:
        data = sock.recv(n)
    except (BlockingIOError, InterruptedError):
        self.add_reader(fd, self.sock_recv, sock, n ,fut, True)
    except Exception as e:
        fut.set_exception(e)
    else:
        fut.set_result(data)
    return fut

def sock_sendall(self, sock, data, fut=None, registed=False):
    fd = sock.fileno()
```

(continues on next page)

(continued from previous page)

```
    if fut is None:
        fut = self.create_future()
    if registed:
        self.remove_writer(fd)
    try:
        n = sock.send(data)
    except (BlockingIOError, InterruptedError):
        n = 0
    except Exception as e:
        fut.set_exception(e)
        return
    if n == len(data):
        fut.set_result(None)
    else:
        if n:
            data = data[n:]
        self.add_writer(fd, sock, data, fut, True)
    return fut

async def handler(loop, conn):
    while True:
        msg = await loop.sock_recv(conn, 1024)
        if msg: await loop.sock_sendall(conn, msg)
        else: break
    conn.close()

async def server(loop):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.setblocking(False)
    sock.bind(('localhost', 9527))
    sock.listen(10)

    while True:
        conn, addr = await loop.sock_accept(sock)
        loop.create_task(handler(loop, conn))

EventLoop = asyncio.SelectorEventLoop
EventLoop.sock_accept = sock_accept
EventLoop.sock_recv = sock_recv
EventLoop.sock_sendall = sock_sendall
loop = EventLoop()

try:
    loop.run_until_complete(server(loop))
except KeyboardInterrupt:
    pass
finally:
    loop.close()
```

output:

```
# console 1
$ python3 async_sock.py &
$ nc localhost 9527
Hello
Hello
```

(continues on next page)

(continued from previous page)

```
# console 2
$ nc localhost 9527
asyncio
asyncio
```

4.2.7 How does `loop.create_server` work?

```python
import asyncio
import socket

loop = asyncio.get_event_loop()

async def create_server(loop, protocol_factory, host,
                        port, *args, **kwargs):
   sock = socket.socket(socket.AF_INET,
                        socket.SOCK_STREAM, 0)
   sock.setsockopt(socket.SOL_SOCKET,
                   socket.SO_REUSEADDR, 1)
   sock.setblocking(False)
   sock.bind((host, port))
   sock.listen(10)
   sockets = [sock]
   server = asyncio.base_events.Server(loop, sockets)
   loop._start_serving(protocol_factory, sock, None, server)

   return server


class EchoProtocol(asyncio.Protocol):
    def connection_made(self, transport):
        peername = transport.get_extra_info('peername')
        print('Connection from {}'.format(peername))
        self.transport = transport

    def data_received(self, data):
        message = data.decode()
        self.transport.write(data)

# Equal to: loop.create_server(EchoProtocol,
#                              'localhost', 5566)
coro = create_server(loop, EchoProtocol, 'localhost', 5566)
server = loop.run_until_complete(coro)

try:
    loop.run_forever()
finally:
    server.close()
    loop.run_until_complete(server.wait_closed())
    loop.close()
```

output:

```
# console1
$ nc localhost 5566
```

(continues on next page)

(continued from previous page)

```
Hello
Hello

# console2
$ nc localhost 5566
asyncio
asyncio
```