Python Cookbook

Release 2.0.0

April 01, 2016

Contents:

CHAPTER

ONE

COPYRIGHT

Python Cookbook 3rd Edition
David Beazley, Brian K. Jones

3
O ' Reilly Media, Inc.
2013 5 08

http://oreilly.com/catalog/errata.csp?isbn=9781449340377

CHAPTER

TWO

2.1

https://github.com/yidao620c/python3-cookbook

2.2

Python

Python3 Python
Python3
2.x 3.x

Python Cookbook 3rd Edition Python3
Python3

yidao620@gmail.com

2.3

2008 Python3 Python3
2013 Python
2 Python3

Python3

Python3 Python Cookbook

Python3.3

Python3

Python
ActiveState ' s Python recipes Stack Overflow
Python2
(2.3 2.4)
Python3.3 Python3

Python3
Python

Python
Python
Python 3 Python

( )
Python

2.4

Python

(
C)

Python

Python
Python (
)

2.5

http://github.com/dabeaz/python-cookbook
bug

ISBN Python Cookbook, 3rd edition, by David Beazley and Brian K. Jones (O ' Reilly).

2.6

O ' Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

: http://oreil.ly/python_cookbook_3e

bookquestions@oreilly.com

http://www.oreilly.com

Facebook : http://facebook.com/oreilly

Twitter : http://twitter.com/oreillymedia

YouTube : http://www.youtube.com/oreillymedia

2.7

Jake Vanderplas, Robert Kern Andrea Crotti
Python
Jake Vanderplas, Robert Kern,and Andrea Crotti

CHAPTER

THREE

Python

collections

Contents:

3.1 1.1

3.1.1

N

N

3.1.2

( )

```
>>> p = (4, 5)
>>> x, y = p
>>> x
4
>>> y
5
>>>
>>> data = [ 'ACME', 50, 91.1, (2012, 12, 21) ]
>>> name, shares, price, date = data
>>> name
'ACME'
>>> date
(2012, 12, 21)
>>> name, shares, price, (year, mon, day) = data
```

```
>>> name
'ACME'
>>> year
2012
>>> mon
12
>>> day
21
>>>
```

```
>>> p = (4, 5)
>>> x, y, z = p
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: need more than 2 values to unpack
>>>
```

3.1.3

```
>>> s = 'Hello'
>>> a, b, c, d, e = s
>>> a
'H'
>>> b
'e'
>>> e
'o'
>>>
```

Python

```
>>> data = [ 'ACME', 50, 91.1, (2012, 12, 21) ]
>>> _, shares, price, _ = data
>>> shares
50
>>> price
91.1
>>>
```

3.2 1.2

3.2.1

ValueError

N

3.2.2

Python

24

```
def drop_first_last(grades):
    first, *middle, last = grades
    return avg(middle)
```

```
>>> record = ('Dave', 'dave@example.com', '773-555-1212', '847-555-1212')
>>> name, email, *phone_numbers = record
>>> name
'Dave'
>>> email
'dave@example.com'
>>> phone_numbers
['773-555-1212', '847-555-1212']
>>>
```

phone_numbers

(0) phone_numbers

8

7

```
*trailing_qtrs, current_qtr = sales_record
trailing_avg = sum(trailing_qtrs) / len(trailing_qtrs)
return avg_comparison(trailing_avg, current_qtr)
```

Python

```
>>> *trailing, current = [10, 8, 7, 1, 9, 5, 10, 3]
>>> trailing
[10, 8, 7, 1, 9, 5, 10]
>>> current
3
```

3.2.3

1

```
records = [
    ('foo', 1, 2),
    ('bar', 'hello'),
    ('foo', 3, 4),
]

def do_foo(x, y):
    print('foo', x, y)

def do_bar(s):
    print('bar', s)

for tag, *args in records:
    if tag == 'foo':
        do_foo(*args)
    elif tag == 'bar':
        do_bar(*args)
```

```
>>> line = 'nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false'
>>> uname, *fields, homedir, sh = line.split(':')
>>> uname
'nobody'
>>> homedir
'/var/empty'
>>> sh
'/usr/bin/false'
>>>
```

*

_ ign

```
>>> record = ('ACME', 50, 123.45, (12, 18, 2012))
>>> name, *_, (*_, year) = record
>>> name
'ACME'
>>> year
```

```
2012
>>>
```

```
>>> items = [1, 10, 7, 4, 5, 9]
>>> head, *tail = items
>>> head
1
>>> tail
[10, 7, 4, 5, 9]
>>>
```

```
>>> def sum(items):
... head, *tail = items
... return head + sum(tail) if tail else head
...
>>> sum(items)
36
>>>
```

Python

3.3 1.3 N

3.3.1

3.3.2

collections.deque

N

```
from collections import deque


def search(lines, pattern, history=5):
    previous_lines = deque(maxlen=history)
    for li in lines:
        if pattern in li:
            yield li, previous_lines
        previous_lines.append(li)
```

```
# Example use on a file
if __name__ == '__main__':
    with open(r'../../cookbook/somefile.txt') as f:
        for line, prevlines in search(f, 'python', 5):
            for pline in prevlines:
                print(pline, end='')
            print(line, end='')
            print('-' * 20)
```

3.3.3

yield

4.3

deque(maxlen=N)

```
>>> q = deque(maxlen=3)
>>> q.append(1)
>>> q.append(2)
>>> q.append(3)
>>> q
deque([1, 2, 3], maxlen=3)
>>> q.append(4)
>>> q
deque([2, 3, 4], maxlen=3)
>>> q.append(5)
>>> q
deque([3, 4, 5], maxlen=3)
```

( )

deque

```
>>> q = deque()
>>> q.append(1)
>>> q.append(2)
>>> q.append(3)
>>> q
deque([1, 2, 3])
>>> q.appendleft(4)
>>> q
deque([4, 1, 2, 3])
```

```
>>> q.pop()
3
>>> q
deque([4, 1, 2])
>>> q.popleft()
4
```

O(1)

O(N)

3.4 1.4 N

3.4.1

N

3.4.2

heapq nlargest() nsmallest()

```
import heapq
nums = [1, 8, 2, 23, 7, -4, 18, 23, 42, 37, 2]
print(heapq.nlargest(3, nums)) # Prints [42, 37, 23]
print(heapq.nsmallest(3, nums)) # Prints [-4, 1, 2]
```

```
portfolio = [
    {'name': 'IBM', 'shares': 100, 'price': 91.1},
    {'name': 'AAPL', 'shares': 50, 'price': 543.22},
    {'name': 'FB', 'shares': 200, 'price': 21.09},
    {'name': 'HPQ', 'shares': 35, 'price': 31.75},
    {'name': 'YHOO', 'shares': 45, 'price': 16.35},
    {'name': 'ACME', 'shares': 75, 'price': 115.65}
]
cheap = heapq.nsmallest(3, portfolio, key=lambda s: s['price'])
expensive = heapq.nlargest(3, portfolio, key=lambda s: s['price'])
```

price

3.4.3

N N

```
>>> nums = [1, 8, 2, 23, 7, -4, 18, 23, 42, 37, 2]
>>> import heapq
>>> heapq.heapify(nums)
>>> nums
[-4, 2, 1, 23, 7, 2, 18, 23, 42, 37, 8]
>>>
```

heap[0]
heapq.heappop()
(O(log N) N
) 3

```
>>> heapq.heappop(nums)
-4
>>> heapq.heappop(nums)
1
>>> heapq.heappop(nums)
2
```

nlargest() nsmallest()
(N=1) min()
max() N
(sorted(items)[:N] sorted(items)[-
N:]) nlargest() nsmallest() (
N)

heapq

3.5 1.5

3.5.1

pop

3.5.2

heapq

```
import heapq

class PriorityQueue:
    def __init__(self):
        self._queue = []
        self._index = 0
```

```
    def push(self, item, priority):
        heapq.heappush(self._queue, (-priority, self._index, item))
        self._index += 1

    def pop(self):
        return heapq.heappop(self._queue)[-1]
```

```
>>> class Item:
...     def __init__(self, name):
...         self.name = name
...     def __repr__(self):
...         return 'Item({!r})'.format(self.name)
...
>>> q = PriorityQueue()
>>> q.push(Item('foo'), 1)
>>> q.push(Item('bar'), 5)
>>> q.push(Item('spam'), 4)
>>> q.push(Item('grok'), 1)
>>> q.pop()
Item('bar')
>>> q.pop()
Item('spam')
>>> q.pop()
Item('foo')
>>> q.pop()
Item('grok')
>>>
```

pop()
(foo grok) pop

3.5.3

heapq heapq.heappush()
heapq.heappop() _queue _queue
(1.4) heappop() ”
” pop push
pop O(log N) N N

(-priority, index, item)

index
index index

Item

```
>>> a = Item('foo')
>>> b = Item('bar')
>>> a < b
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unorderable types: Item() < Item()
>>>
```

(priority, item)

```
>>> a = (1, Item('foo'))
>>> b = (5, Item('bar'))
>>> a < b
True
>>> c = (1, Item('grok'))
>>> a < c
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unorderable types: Item() < Item()
>>>
```

index (priority, index, item)
index Python

```
>>> a = (1, 0, Item('foo'))
>>> b = (5, 1, Item('bar'))
>>> c = (1, 2, Item('grok'))
>>> a < b
True
>>> a < c
True
>>>
```

12.3

heapq

3.6 1.6

3.6.1

(multidict)

3.6.2

```
d = {
    'a' : [1, 2, 3],
    'b' : [4, 5]
}
e = {
    'a' : {1, 2, 3},
    'b' : {4, 5}
}
```

collections defaultdict
defaultdict key

```
from collections import defaultdict

d = defaultdict(list)
d['a'].append(1)
d['a'].append(2)
d['b'].append(4)

d = defaultdict(set)
d['a'].add(1)
d['a'].add(2)
d['b'].add(4)
```

defaultdict (
)
setdefault()

```
d = {} # A regular dictionary
d.setdefault('a', []).append(1)
d.setdefault('a', []).append(2)
d.setdefault('b', []).append(4)
```

setdefault()
([])

3.6.3

```
d = {}
for key, value in pairs:
    if key not in d:
        d[key] = []
    d[key].append(value)
```

defaultdict

```
d = defaultdict(list)
for key, value in pairs:
    d[key].append(value)
```

1.15

3.7 1.7

3.7.1

3.7.2

collections

OrderedDict

```
from collections import OrderedDict
def ordered_dict():
    d = OrderedDict()
    d['foo'] = 1
    d['bar'] = 2
    d['spam'] = 3
    d['grok'] = 4
    # Outputs "foo 1", "bar 2", "spam 3", "grok 4"
    for key in d:
        print(key, d[key])
```

OrderedDict

JSON

OrderedDict

```
>>> import json
>>> json.dumps(d)
'{"foo": 1, "bar": 2, "spam": 3, "grok": 4}'
>>>
```

3.7.3

OrderedDict

OrderedDict

OrderedDict

(100,000 CSV OrderedDict)

OrderedDict

3.8 1.8

3.8.1

( )

3.8.2

```
prices = {
    'ACME': 45.23,
    'AAPL': 612.78,
    'IBM': 205.55,
    'HPQ': 37.20,
    'FB': 10.75
}
```

zip()

```
min_price = min(zip(prices.values(), prices.keys()))
# min_price is (10.75, 'FB')
max_price = max(zip(prices.values(), prices.keys()))
# max_price is (612.78, 'AAPL')
```

zip() sorted()

```
prices_sorted = sorted(zip(prices.values(), prices.keys()))
# prices_sorted is [(10.75, 'FB'), (37.2, 'HPQ'),
#                   (45.23, 'ACME'), (205.55, 'IBM'),
#                   (612.78, 'AAPL')]
```

zip()

```
prices_and_names = zip(prices.values(), prices.
```

3.9 1.9

3.9.1

( )

3.9.2

```
a = {
    'x' : 1,
    'y' : 2,
    'z' : 3
}

b = {
    'w' : 10,
    'x' : 11,
    'y' : 2
}
```

keys() items()

```
# Find keys in common
a.keys() & b.keys() # { 'x', 'y' }
# Find keys in a that are not in b
a.keys() - b.keys() # { 'z' }
# Find (key,value) pairs in common
a.items() & b.items() # { ('y', 2) }
```

```
# Make a new dictionary with certain keys removed
c = {key:a[key] for key in a.keys() - {'z', 'w'}}
# c is {'x': 1, 'y': 2}
```

3.9.3

keys()

set

items() ( )

values()

set

3.10 1.10

3.10.1

3.10.2

hashable

```
def dedupe(items):
    seen = set()
    for item in items:
        if item not in seen:
            yield item
            seen.add(item)
```

```
>>> a = [1, 5, 2, 1, 9, 1, 5, 10]
>>> list(dedupe(a))
[1, 5, 2, 9, 10]
>>>
```

hashable

(dict)

```
def dedupe(items, key=None):
    seen = set()
    for item in items:
        val = item if key is None else key(item)
        if val not in seen:
            yield item
            seen.add(val)
```

key hashable

```
>>> a = [ {'x':1, 'y':2}, {'x':1, 'y':3}, {'x':1, 'y':2}, {'x':2, 'y':4}]
>>> list(dedupe(a, key=lambda d: (d['x'],d['y'])))
[{'x': 1, 'y': 2}, {'x': 1, 'y': 3}, {'x': 2, 'y': 4}]
```

```
>>> list(dedupe(a, key=lambda d: d['x']))
[{'x': 1, 'y': 2}, {'x': 2, 'y': 4}]
>>>
```

3.10.3

```
>>> a
[1, 5, 2, 1, 9, 1, 5, 10]
>>> set(a)
{1, 2, 10, 5, 9}
>>>
```

```
with open(somefile,'r') as f:
for line in dedupe(f):
    ...
```

key sorted() , min() max()
1.8 1.13

3.11 1.11

3.11.1

3.11.2

(
)

```
###### 0123456789012345678901234567890123456789012345678901234567890'
record = '.....100 .....513.25 ..........'
cost = int(record[20:23]) * float(record[31:37])
```

```
SHARES = slice(20, 23)
PRICE = slice(31, 37)
cost = int(record[SHARES]) * float(record[PRICE])
```

3.11.3

slice()

```
>>> items = [0, 1, 2, 3, 4, 5, 6]
>>> a = slice(2, 4)
>>> items[2:4]
[2, 3]
>>> items[a]
[2, 3]
>>> items[a] = [10,11]
>>> items
[0, 1, 10, 11, 4, 5, 6]
>>> del items[a]
>>> items
[0, 1, 4, 5, 6]
```

a a.start , a.stop , a.step

```
>>> a = slice(5, 50, 2)
>>> a.start
5
>>> a.stop
50
>>> a.step
2
>>>
```

indices(size)
(start, stop, step)
IndexError

```
>>> s = 'HelloWorld'
>>> a.indices(len(s))
(5, 10, 2)
>>> for i in range(*a.indices(len(s))):
... print(s[i])
...
```

```
W
r
d
>>>
```

3.12 1.12

3.12.1

3.12.2

collections.Counter
most_common()

```
words = [
    'look', 'into', 'my', 'eyes', 'look', 'into', 'my', 'eyes',
    'the', 'eyes', 'the', 'eyes', 'the', 'eyes', 'not', 'around', 'the',
    'eyes', "don't", 'look', 'around', 'the', 'eyes', 'look', 'into',
    'my', 'eyes', "you're", 'under'
]
from collections import Counter
word_counts = Counter(words)
#              3
top_three = word_counts.most_common(3)
print(top_three)
# Outputs [('eyes', 8), ('the', 5), ('look', 4)]
```

3.12.3

Counter hashable
Counter

```
>>> word_counts['not']
1
>>> word_counts['eyes']
8
>>>
```

```
>>> morewords = ['why','are','you','not','looking','in','my','eyes']
>>> for word in morewords:
... word_counts[word] += 1
...
>>> word_counts['eyes']
9
>>>
```

update()

```
>>> word_counts.update(morewords)
>>>
```

Counter

```
>>> a = Counter(words)
>>> b = Counter(morewords)
>>> a
Counter({'eyes': 8, 'the': 5, 'look': 4, 'into': 3, 'my': 3, 'around': 2,
"you're": 1, "don't": 1, 'under': 1, 'not': 1})
>>> b
Counter({'eyes': 1, 'looking': 1, 'are': 1, 'in': 1, 'not': 1, 'you': 1,
'my': 1, 'why': 1})
>>> # Combine counts
>>> c = a + b
>>> c
Counter({'eyes': 9, 'the': 5, 'look': 4, 'my': 4, 'into': 3, 'not': 2,
'around': 2, "you're": 1, "don't": 1, 'in': 1, 'why': 1,
'looking': 1, 'are': 1, 'under': 1, 'you': 1})
>>> # Subtract counts
>>> d = a - b
>>> d
Counter({'eyes': 7, 'the': 5, 'look': 4, 'into': 3, 'my': 2, 'around': 2,
"you're": 1, "don't": 1, 'under': 1})
>>>
```

Counter

3.13 1.13

3.13.1

3.13.2

operator itemgetter

```
rows = [
    {'fname': 'Brian', 'lname': 'Jones', 'uid': 1003},
    {'fname': 'David', 'lname': 'Beazley', 'uid': 1002},
    {'fname': 'John', 'lname': 'Cleese', 'uid': 1001},
    {'fname': 'Big', 'lname': 'Jones', 'uid': 1004}
]
```

```
from operator import itemgetter
rows_by_fname = sorted(rows, key=itemgetter('fname'))
rows_by_uid = sorted(rows, key=itemgetter('uid'))
print(rows_by_fname)
print(rows_by_uid)
```

```
[{'fname': 'Big', 'uid': 1004, 'lname': 'Jones'},
{'fname': 'Brian', 'uid': 1003, 'lname': 'Jones'},
{'fname': 'David', 'uid': 1002, 'lname': 'Beazley'},
{'fname': 'John', 'uid': 1001, 'lname': 'Cleese'}]
[{'fname': 'John', 'uid': 1001, 'lname': 'Cleese'},
{'fname': 'David', 'uid': 1002, 'lname': 'Beazley'},
{'fname': 'Brian', 'uid': 1003, 'lname': 'Jones'},
{'fname': 'Big', 'uid': 1004, 'lname': 'Jones'}]
```

itemgetter() keys

```
rows_by_lfname = sorted(rows, key=itemgetter('lname','fname'))
print(rows_by_lfname)
```

```
[{'fname': 'David', 'uid': 1002, 'lname': 'Beazley'},
{'fname': 'John', 'uid': 1001, 'lname': 'Cleese'},
{'fname': 'Big', 'uid': 1004, 'lname': 'Jones'},
{'fname': 'Brian', 'uid': 1003, 'lname': 'Jones'}]
```

3.13.3

rows sorted()
callable rows
itemgetter() callable

operator.itemgetter() rows
__getitem__()

itemgetter() callable
sorted()
( )

itemgetter() lambda

```
rows_by_fname = sorted(rows, key=lambda r: r['fname'])
rows_by_lfname = sorted(rows, key=lambda r: (r['lname'],r['fname']))
```

itemgetter()
itemgetter()

min() max()

```
>>> min(rows, key=itemgetter('uid'))
{'fname': 'John', 'lname': 'Cleese', 'uid': 1001}
>>> max(rows, key=itemgetter('uid'))
{'fname': 'Big', 'lname': 'Jones', 'uid': 1004}
>>>
```

3.14 1.14

3.14.1

3.14.2

sorted() key callable
callable sorted
User
user_id User user_id
callable

```
class User:
    def __init__(self, user_id):
        self.user_id = user_id

    def __repr__(self):
        return 'User({})'.format(self.user_id)


def sort_notcompare():
    users = [User(23), User(3), User(99)]
    print(users)
    print(sorted(users, key=lambda u: u.user_id))
```

operator.attrgetter() lambda

```
>>> from operator import attrgetter
>>> sorted(users, key=attrgetter('user_id'))
[User(3), User(23), User(99)]
>>>
```

3.14.3

lambda attrgetter()
attrgetter()
operator.itemgetter() (1.13)
User first_name last_name

```
by_name = sorted(users, key=attrgetter('last_name', 'first_name'))
```

min() max()

```
>>> min(users, key=attrgetter('user_id')
User(3)
>>> max(users, key=attrgetter('user_id')
User(99)
>>>
```

3.15 1.15

3.15.1

date

3.15.2

itertools.groupby()

```
rows = [
    {'address': '5412 N CLARK', 'date': '07/01/2012'},
    {'address': '5148 N CLARK', 'date': '07/04/2012'},
    {'address': '5800 E 58TH', 'date': '07/02/2012'},
    {'address': '2122 N CLARK', 'date': '07/03/2012'},
    {'address': '5645 N RAVENSWOOD', 'date': '07/02/2012'},
    {'address': '1060 W ADDISON', 'date': '07/02/2012'},
    {'address': '4801 N BROADWAY', 'date': '07/01/2012'},
```

```
    {'address': '1039 W GRANVILLE', 'date': '07/04/2012'},
]
```

date
(date) itertools.groupby()

```
from operator import itemgetter
from itertools import groupby

# Sort by the desired field first
rows.sort(key=itemgetter('date'))
# Iterate in groups
for date, items in groupby(rows, key=itemgetter('date')):
    print(date)
    for i in items:
        print(' ', i)
```

```
07/01/2012
  {'date': '07/01/2012', 'address': '5412 N CLARK'}
  {'date': '07/01/2012', 'address': '4801 N BROADWAY'}
07/02/2012
  {'date': '07/02/2012', 'address': '5800 E 58TH'}
  {'date': '07/02/2012', 'address': '5645 N RAVENSWOOD'}
  {'date': '07/02/2012', 'address': '1060 W ADDISON'}
07/03/2012
  {'date': '07/03/2012', 'address': '2122 N CLARK'}
07/04/2012
  {'date': '07/04/2012', 'address': '5148 N CLARK'}
  {'date': '07/04/2012', 'address': '1039 W GRANVILLE'}
```

3.15.3

groupby() (key
)

groupby()

date
defaultdict()
1.6

```
from collections import defaultdict
rows_by_date = defaultdict(list)
for row in rows:
    rows_by_date[row['date']].append(row)
```

```
>>> for r in rows_by_date['07/01/2012']:
... print(r)
...
{'date': '07/01/2012', 'address': '5412 N CLARK'}
{'date': '07/01/2012', 'address': '4801 N BROADWAY'}
>>>
```

groupby()

3.16 1.16

3.16.1

3.16.2

```
>>> mylist = [1, 4, -5, 10, -7, 2, 3, -1]
>>> [n for n in mylist if n > 0]
[1, 4, 10, 2, 3]
>>> [n for n in mylist if n < 0]
[-5, -7, -1]
>>>
```

```
>>> pos = (n for n in mylist if n > 0)
>>> pos
<generator object <genexpr> at 0x1006a0eb0>
>>> for x in pos:
... print(x)
...
1
4
10
2
3
>>>
```

filter()

```
values = ['1', '2', '-3', '-', '4', 'N/A', '5']
def is_int(val):
    try:
        x = int(val)
        return True
    except ValueError:
        return False
ivals = list(filter(is_int, values))
print(ivals)
# Outputs ['1', '2', '-3', '4', '5']
```

filter()
list()

3.16.3

```
>>> mylist = [1, 4, -5, 10, -7, 2, 3, -1]
>>> import math
>>> [math.sqrt(n) for n in mylist if n > 0]
[1.0, 2.0, 3.1622776601683795, 1.4142135623730951, 1.7320508075688772]
>>>
```

```
>>> clip_neg = [n if n > 0 else 0 for n in mylist]
>>> clip_neg
[1, 4, 0, 10, 0, 2, 3, 0]
>>> clip_pos = [n if n < 0 else 0 for n in mylist]
>>> clip_pos
[0, 0, -5, 0, -7, 0, 0, -1]
>>>
```

itertools.compress() iterable
Boolean iterable
True

```
addresses = [
    '5412 N CLARK',
    '5148 N CLARK',
    '5800 E 58TH',
    '2122 N CLARK'
    '5645 N RAVENSWOOD',
    '1060 W ADDISON',
    '4801 N BROADWAY',
```

```
    '1039 W GRANVILLE',
]
counts = [ 0, 3, 10, 4, 1, 7, 6, 1]
```

count 5

```
>>> from itertools import compress
>>> more5 = [n > 5 for n in counts]
>>> more5
[False, False, True, False, False, True, True, False]
>>> list(compress(addresses, more5))
['5800 E 58TH', '4801 N BROADWAY', '1039 W GRANVILLE']
>>>
```

Boolean
compress() True

filter() compress()
list()

3.17 1.17

3.17.1

3.17.2

```
prices = {
    'ACME': 45.23,
    'AAPL': 612.78,
    'IBM': 205.55,
    'HPQ': 37.20,
    'FB': 10.75
}
# Make a dictionary of all prices over 200
p1 = {key: value for key, value in prices.items() if value > 200}
# Make a dictionary of tech stocks
tech_names = {'AAPL', 'IBM', 'HPQ', 'MSFT'}
p2 = {key: value for key, value in prices.items() if key in tech_names}
```

3.17.3

dict()

```
p1 = dict((key, value) for key, value in prices.items() if value > 200)
```

(

dcit())

```
# Make a dictionary of tech stocks
tech_names = { 'AAPL', 'IBM', 'HPQ', 'MSFT' }
p2 = { key:prices[key] for key in prices.keys() & tech_names }
```

1.6

14.13

3.18 1.18

3.18.1

3.18.2

collections.namedtuple()

Python

```
>>> from collections import namedtuple
>>> Subscriber = namedtuple('Subscriber', ['addr', 'joined'])
>>> sub = Subscriber('jonesy@example.com', '2012-10-19')
>>> sub
Subscriber(addr='jonesy@example.com', joined='2012-10-19')
>>> sub.addr
'jonesy@example.com'
>>> sub.joined
'2012-10-19'
>>>
```

namedtuple

```
>>> len(sub)
2
>>> addr, joined = sub
>>> addr
'jonesy@example.com'
>>> joined
'2012-10-19'
>>>
```

```
def compute_cost(records):
total = 0.0
for rec in records:
    total += rec[1] * rec[2]
return total
```

```
from collections import namedtuple

Stock = namedtuple('Stock', ['name', 'shares', 'price'])
def compute_cost(records):
    total = 0.0
    for rec in records:
        s = Stock(*rec)
    total += s.shares * s.price
    return total
```

3.18.3

```
>>> s = Stock('ACME', 100, 123.45)
>>> s
Stock(name='ACME', shares=100, price=123.45)
>>> s.shares = 75
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>>
```

_replace()

```
>>> s = s._replace(shares=75)
>>> s
Stock(name='ACME', shares=75, price=123.45)
>>>
```

_replace()

_replace()

```
from collections import namedtuple

Stock = namedtuple('Stock', ['name', 'shares', 'price', 'date', 'time'])

# Create a prototype instance
stock_prototype = Stock('', 0, 0.0, None, None)

# Function to convert a dictionary to a Stock
def dict_to_stock(s):
    return stock_prototype._replace(**s)
```

```
>>> a = {'name': 'ACME', 'shares': 100, 'price': 123.45}
>>> dict_to_stock(a)
Stock(name='ACME', shares=100, price=123.45, date=None, time=None)
>>> b = {'name': 'ACME', 'shares': 100, 'price': 123.45, 'date': '12/17/2012'}
>>> dict_to_stock(b)
Stock(name='ACME', shares=100, price=123.45, date='12/17/2012', time=None)
>>>
```

__slots__

(8.4)

3.19 1.19

3.19.1

(sum() , min() , max())

3.19.2

```
nums = [1, 2, 3, 4, 5]
s = sum(x * x for x in nums)
```

```
# Determine if any .py files exist in a directory
import os
files = os.listdir('dirname')
if any(name.endswith('.py') for name in files):
    print('There be python!')
else:
    print('Sorry, no python.')
# Output a tuple as CSV
s = ('ACME', 50, 123.45)
print(','.join(str(x) for x in s))
# Data reduction across fields of a data structure
portfolio = [
    {'name':'GOOG', 'shares': 50},
    {'name':'YHOO', 'shares': 75},
    {'name':'AOL', 'shares': 20},
    {'name':'SCOX', 'shares': 65}
]
min_shares = min(s['shares'] for s in portfolio)
```

3.19.3

( )

```
s = sum((x * x for x in nums)) #
s = sum(x * x for x in nums) #
```

```
nums = [1, 2, 3, 4, 5]
s = sum([x * x for x in nums])
```

min() max()

key

```
# Original: Returns 20
min_shares = min(s['shares'] for s in portfolio)
# Alternative: Returns {'name': 'AOL', 'shares': 20}
min_shares = min(portfolio, key=lambda s: s['shares'])
```

3.20 1.20

3.20.1

3.20.2

:

```
a = {'x': 1, 'z': 3 }
b = {'y': 2, 'z': 4 }
```

(a b
) collections ChainMap

```
from collections import ChainMap
c = ChainMap(a,b)
print(c['x']) # Outputs 1 (from a)
print(c['y']) # Outputs 2 (from b)
print(c['z']) # Outputs 3 (from a)
```

3.20.3

ChainMap
ChainMap

```
>>> len(c)
3
>>> list(c.keys())
['x', 'y', 'z']
>>> list(c.values())
[1, 2, 3]
>>>
```

c['z']

a b

```
>>> c['z'] = 10
>>> c['w'] = 40
>>> del c['x']
>>> a
{'w': 40, 'z': 10}
>>> del c['y']
Traceback (most recent call last):
...
KeyError: "Key not found in the first mapping: 'y'"
>>>
```

ChainMap (globals , locals)

```
>>> values = ChainMap()
>>> values['x'] = 1
>>> # Add a new mapping
>>> values = values.new_child()
>>> values['x'] = 2
>>> # Add a new mapping
>>> values = values.new_child()
>>> values['x'] = 3
>>> values
ChainMap({'x': 3}, {'x': 2}, {'x': 1})
>>> values['x']
3
>>> # Discard last mapping
>>> values = values.parents
>>> values['x']
2
>>> # Discard last mapping
>>> values = values.parents
>>> values['x']
1
>>> values
ChainMap({'x': 1})
>>>
```

ChainMap update()

```
>>> a = {'x': 1, 'z': 3 }
>>> b = {'y': 2, 'z': 4 }
>>> merged = dict(b)
>>> merged.update(a)
>>> merged['x']
1
>>> merged['y']
2
>>> merged['z']
3
>>>
```

(
)

```
>>> a['x'] = 13
>>> merged['x']
1
```

ChainMap

```
>>> a = {'x': 1, 'z': 3 }
>>> b = {'y': 2, 'z': 4 }
>>> merged = ChainMap(a, b)
>>> merged['x']
1
>>> a['x'] = 42
>>> merged['x'] # Notice change to merged dicts
42
>>>
```

CHAPTER

FOUR

Unicode

Contents:

4.1 2.1

4.1.1

( )

4.1.2

string split()

re.split()

```
>>> line = 'asdf fjdk; afed, fjek,asdf, foo'
>>> import re
>>> re.split(r'[;,\s]\s*', line)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
```

4.1.3

re.split()

str.split()

re.split()

```
>>> fields = re.split(r'(;|,|\s)\s*', line)
>>> fields
['asdf', ' ', 'fjdk', ';', 'afed', ',', 'fjek', ',', 'asdf', ',', 'foo']
>>>
```

```
>>> values = fields[::2]
>>> delimiters = fields[1::2] + ['']
>>> values
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
>>> delimiters
[' ', ';', ',', ',', ',', '']
>>> # Reform the line using the same delimiters
>>> ''.join(v+d for v,d in zip(values, delimiters))
'asdf fjdk;afed,fjek,asdf,foo'
>>>
```

(?:...)

```
>>> re.split(r'(?:,|;|\s)\s*', line)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
>>>
```

4.2 2.2

4.2.1

URL

Scheme

4.2.2

str.startswith()

str.endswith()

```
>>> filename = 'spam.txt'
>>> filename.endswith('.txt')
True
>>> filename.startswith('file:')
False
>>> url = 'http://www.python.org'
```

```
>>> url.startswith('http:')
True
>>>
```

startswith() endswith()

```
>>> import os
>>> filenames = os.listdir('.')
>>> filenames
[ 'Makefile', 'foo.c', 'bar.py', 'spam.c', 'spam.h' ]
>>> [name for name in filenames if name.endswith(('.c', '.h')) ]
['foo.c', 'spam.c', 'spam.h'
>>> any(name.endswith('.py') for name in filenames)
True
>>>
```

```
from urllib.request import urlopen

def read_data(name):
    if name.startswith(('http:', 'https:', 'ftp:')):
        return urlopen(name).read()
    else:
        with open(name) as f:
            return f.read()
```

list

set tuple()

```
>>> choices = ['http:', 'ftp:']
>>> url = 'http://www.python.org'
>>> url.startswith(choices)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: startswith first arg must be str or a tuple of str, not list
>>> url.startswith(tuple(choices))
True
>>>
```

4.2.3

startswith() endswith()

```
>>> filename = 'spam.txt'
>>> filename[-4:] == '.txt'
True
>>> url = 'http://www.python.org'
```

```
>>> url[:5] == 'http:' or url[:6] == 'https:' or url[:4] == 'ftp:'
True
>>>
```

```
>>> import re
>>> url = 'http://www.python.org'
>>> re.match('http:|https:|ftp:', url)
<_sre.SRE_Match object at 0x101253098>
>>>
```

startswith()
endswith()

```
if any(name.endswith(('.c', '.h')) for name in listdir(dirname)):
...
```

4.3 2.3 Shell

4.3.1

Unix Shell (*.py , Dat[0-9]*.csv)

4.3.2

fnmatch —— fnmatch() fnmatchcase()

```
>>> from fnmatch import fnmatch, fnmatchcase
>>> fnmatch('foo.txt', '*.txt')
True
>>> fnmatch('foo.txt', '?oo.txt')
True
>>> fnmatch('Dat45.csv', 'Dat[0-9]*')
True
>>> names = ['Dat1.csv', 'Dat2.csv', 'config.ini', 'foo.py']
>>> [name for name in names if fnmatch(name, 'Dat*.csv')]
['Dat1.csv', 'Dat2.csv']
>>>
```

fnmatch() ( )

```
>>> # On OS X (Mac)
>>> fnmatch('foo.txt', '*.TXT')
False
>>> # On Windows
>>> fnmatch('foo.txt', '*.TXT')
True
>>>
```

fnmatchcase()

```
>>> fnmatchcase('foo.txt', '*.TXT')
False
>>>
```

```
addresses = [
    '5412 N CLARK ST',
    '1060 W ADDISON ST',
    '1039 W GRANVILLE AVE',
    '2122 N CLARK ST',
    '4802 N BROADWAY',
]
```

```
>>> from fnmatch import fnmatchcase
>>> [addr for addr in addresses if fnmatchcase(addr, '* ST')]
['5412 N CLARK ST', '1060 W ADDISON ST', '2122 N CLARK ST']
>>> [addr for addr in addresses if fnmatchcase(addr, '54[0-9][0-9] *CLARK*')]
['5412 N CLARK ST']
>>>
```

4.3.3

fnmatch()

glob 5.13

4.4 2.4

4.4.1

4.4.2

str.find() , str.endswith() , str.startswith()

```
>>> text = 'yeah, but no, but yeah, but no, but yeah'
>>> # Exact match
>>> text == 'yeah'
False
>>> # Match at start or end
>>> text.startswith('yeah')
True
>>> text.endswith('no')
False
>>> # Search for the location of the first occurrence
>>> text.find('no')
10
>>>
```

re
11/27/2012

```
>>> text1 = '11/27/2012'
>>> text2 = 'Nov 27, 2012'
>>>
>>> import re
>>> # Simple matching: \d+ means match one or more digits
>>> if re.match(r'\d+/\d+/\d+', text1):
... print('yes')
... else:
... print('no')
...
yes
>>> if re.match(r'\d+/\d+/\d+', text2):
... print('yes')
... else:
... print('no')
...
no
>>>
```

```
>>> datepat = re.compile(r'\d+/\d+/\d+')
>>> if datepat.match(text1):
... print('yes')
... else:
... print('no')
...
yes
>>> if datepat.match(text2):
```

```
... print('yes')
... else:
... print('no')
...
no
>>>
```

match()
findall()

```
>>> text = 'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> datepat.findall(text)
['11/27/2012', '3/13/2013']
>>>
```

```
>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
>>>
```

```
>>> m = datepat.match('11/27/2012')
>>> m
<_sre.SRE_Match object at 0x1005d2750>
>>> # Extract the contents of each group
>>> m.group(0)
'11/27/2012'
>>> m.group(1)
'11'
>>> m.group(2)
'27'
>>> m.group(3)
'2012'
>>> m.groups()
('11', '27', '2012')
>>> month, day, year = m.groups()
>>>
>>> # Find all matches (notice splitting into tuples)
>>> text
'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> datepat.findall(text)
[('11', '27', '2012'), ('3', '13', '2013')]
>>> for month, day, year in datepat.findall(text):
... print('{}-{}-{}'.format(year, month, day))
...
2012-11-27
2013-3-13
>>>
```

findall()

finditer()

```
>>> for m in datepat.finditer(text):
... print(m.groups())
...
('11', '27', '2012')
('3', '13', '2013')
>>>
```

4.4.3

re

re.compile()

match() , findall() finditer()

r'(\d+)/(\d+)/(\d+)'

'(\\d+)/(\\d+)/(\\d+)'

match()

```
>>> m = datepat.match('11/27/2012abcdef')
>>> m
<_sre.SRE_Match object at 0x1005d27e8>
>>> m.group()
'11/27/2012'
>>>
```

$

```
>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)$')
>>> datepat.match('11/27/2012abcdef')
>>> datepat.match('11/27/2012')
<_sre.SRE_Match object at 0x1005d2750>
>>>
```

/

re

```
>>> re.findall(r'(\d+)/(\d+)/(\d+)', text)
[('11', '27', '2012'), ('3', '13', '2013')]
>>>
```

4.5 2.5

4.5.1

4.5.2

str.repalce()

```
>>> text = 'yeah, but no, but yeah, but no, but yeah'
>>> text.replace('yeah', 'yep')
'yep, but no, but yep, but no, but yep'
>>>
```

re sub()
11/27/2012 2012-11-27

```
>>> text = 'Today is 11/27/2012. PyCon starts 3/13/2013.'
>>> import re
>>> re.sub(r'(\d+)/(\d+)/(\d+)', r'\3-\1-\2', text)
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>>
```

sub()
\3

```
>>> import re
>>> datepat = re.compile(r'(\d+)/(\d+)/(\d+)')
>>> datepat.sub(r'\3-\1-\2', text)
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>>
```

```
>>> from calendar import month_abbr
>>> def change_date(m):
... mon_name = month_abbr[int(m.group(1))]
... return '{} {} {}'.format(m.group(2), mon_name, m.group(3))
...
>>> datepat.sub(change_date, text)
'Today is 27 Nov 2012. PyCon starts 13 Mar 2013.'
>>>
```

match match() find()
group()

re.subn()

```
>>> newtext, n = datepat.subn(r'\3-\1-\2', text)
>>> newtext
'Today is 2012-11-27. PyCon starts 2013-3-13.'
>>> n
2
>>>
```

4.5.3

sub()

4.6 2.6

4.6.1

4.6.2

re

re.IGNORECASE

```
>>> text = 'UPPER PYTHON, lower python, Mixed Python'
>>> re.findall('python', text, flags=re.IGNORECASE)
['PYTHON', 'python', 'Python']
>>> re.sub('python', 'snake', text, flags=re.IGNORECASE)
'UPPER snake, lower snake, Mixed snake'
>>>
```

```
def matchcase(word):
    def replace(m):
        text = m.group()
        if text.isupper():
            return word.upper()
        elif text.islower():
            return word.lower()
        elif text[0].isupper():
            return word.capitalize()
        else:
            return word
    return replace
```

```
>>> re.sub('python', matchcase('snake'), text, flags=re.IGNORECASE)
'UPPER SNAKE, lower snake, Mixed Snake'
>>>
```

matchcase('snake') (match)
sub()

4.6.3

re.IGNORECASE
Unicode

2.10

4.7 2.7

4.7.1

4.7.2

(
)

```
>>> str_pat = re.compile(r'\"(.*)\"')
>>> text1 = 'Computer says "no."'
>>> str_pat.findall(text1)
['no.']
>>> text2 = 'Computer says "no." Phone says "yes."'
>>> str_pat.findall(text2)
['no." Phone says "yes.']
>>>
```

r'\"(.*)\"'
*
text2

* ?

```
>>> str_pat = re.compile(r'\"(.*?)\"')
>>> str_pat.findall(text2)
['no.', 'yes.']
>>>
```

4.7.3

(.)
(.) (.)
( )

* + ?

4.8 2.8

4.8.1

4.8.2

(.) (.)
C

```
>>> comment = re.compile(r'/\*(.*?)\*/')
>>> text1 = '/* this is a comment */'
>>> text2 = '''/* this is a
... multiline comment */
... '''
>>>
>>> comment.findall(text1)
[' this is a comment ']
>>> comment.findall(text2)
[]
>>>
```

```
>>> comment = re.compile(r'/\*((?:.|\n)*?)\*/')
>>> comment.findall(text2)
[' this is a\n multiline comment ']
>>>
```

(?:.|\n) (
)

4.8.3

re.compile() re.DOTALL
(.)

```
>>> comment = re.compile(r'/\*(.*?)\*/', re.DOTALL)
>>> comment.findall(text2)
[' this is a\n multiline comment ']
```

re.DOTALL
(2.18)

4.9 2.9 Unicode

4.9.1

Unicode

4.9.2

Unicode

```
>>> s1 = 'Spicy Jalape\u00f1o'
>>> s2 = 'Spicy Jalapen\u0303o'
>>> s1
'Spicy Jalapeño'
>>> s2
'Spicy Jalapeño'
>>> s1 == s2
False
>>> len(s1)
14
>>> len(s2)
15
>>>
```

”Spicy Jalapeño” ”ñ”(U+00F1) ”n” ”~” (U+0303)

unicodedata

```
>>> import unicodedata
>>> t1 = unicodedata.normalize('NFC', s1)
>>> t2 = unicodedata.normalize('NFC', s2)
>>> t1 == t2
True
>>> print(ascii(t1))
'Spicy Jalape\xf1o'
>>> t3 = unicodedata.normalize('NFD', s1)
```

```
>>> t4 = unicodedata.normalize('NFD', s2)
>>> t3 == t4
True
>>> print(ascii(t3))
'Spicy Jalapen\u0303o'
>>>
```

normalize() NFC
( ) NFD

Python NFKC NFKD

```
>>> s = '\ufb01' # A single character
>>> s
'ﬁ'
>>> unicodedata.normalize('NFD', s)
'ﬁ'
# Notice how the combined letters are broken apart here
>>> unicodedata.normalize('NFKD', s)
'fi'
>>> unicodedata.normalize('NFKC', s)
'fi'
>>>
```

4.9.3

Unicode

( )

```
>>> t1 = unicodedata.normalize('NFD', s1)
>>> ''.join(c for c in t1 if not unicodedata.combining(c))
'Spicy Jalapeno'
>>>
```

unicodedata
combining()

Unicode
Unicode Ned Batchelder Python
Unicode

4.10 2.10 Unicode

4.10.1

Unicode

4.10.2

re Unicode \\d
unicode

```
>>> import re
>>> num = re.compile('\d+')
>>> # ASCII digits
>>> num.match('123')
<_sre.SRE_Match object at 0x1007d9ed0>
>>> # Arabic digits
>>> num.match('\u0661\u0662\u0663')
<_sre.SRE_Match object at 0x101234030>
>>>
```

Unicode Unicode
(\uFFF \UFFFFFFF)

```
>>> arabic = re.compile('[\u0600-\u06ff\u0750-\u077f\u08a0-\u08ff]+')
>>>
```

(2.9)

```
>>> pat = re.compile('stra\u00dfe', re.IGNORECASE)
>>> s = 'straße'
>>> pat.match(s) # Matches
<_sre.SRE_Match object at 0x10069d370>
>>> pat.match(s.upper()) # Doesn't match
>>> s.upper() # Case folds
'STRASSE'
>>>
```

4.10.3

Unicode

Unicode

4.11 2.11

4.11.1

4.11.2

strip() lstrip() rstrip()

```
>>> # Whitespace stripping
>>> s = ' hello world \n'
>>> s.strip()
'hello world'
>>> s.lstrip()
'hello world \n'
>>> s.rstrip()
' hello world'
>>>
>>> # Character stripping
>>> t = '-----hello====='
>>> t.lstrip('-')
'hello====='
>>> t.strip('-=')
'hello'
>>>
```

4.11.3

strip()

```
>>> s = ' hello     world \n'
>>> s = s.strip()
>>> s
'hello     world'
>>>
```

replace()

```
>>> s.replace(' ', '')
'helloworld'
>>> import re
```

```
>>> re.sub('\s+', ' ', s)
'hello world'
>>>
```

strip

```
with open(filename) as f:
    lines = (line.strip() for line in f)
    for line in lines:
        print(line)
```

lines = (line.strip() for line in f)

strip

strip translate()

4.12 2.12

4.12.1

”pýt̂höñ”

4.12.2

(str.upper() str.lower())
str.replace() re.sub()
2.9 unicodedata.normalize() unicode

str.translate()

```
>>> s = 'pýt̂höñ\fis\tawesome\r\n'
>>> s
'pýt̂höñ\x0cis\tawesome\r\n'
>>>
```

translate()

```
>>> remap = {
...     ord('\t') : ' ',
...     ord('\f') : ' ',
...     ord('\r') : None # Deleted
... }
>>> a = s.translate(remap)
>>> a
'pýtĥöñ is awesome\n'
>>>
```

\t \f r

```
>>> import unicodedata
>>> import sys
>>> cmb_chrs = dict.fromkeys(c for c in range(sys.maxunicode)
...                          if unicodedata.combining(chr(c)))
...
>>> b = unicodedata.normalize('NFD', a)
>>> b
'pýtĥöñ is awesome\n'
>>> b.translate(cmb_chrs)
'python is awesome\n'
>>>
```

dict.fromkeys() Unicode
None

unicodedata.normalize()
translate
( )

Unicode ASCII

```
>>> digitmap = { c: ord('0') + unicodedata.digit(chr(c))
...         for c in range(sys.maxunicode)
...         if unicodedata.category(chr(c)) == 'Nd' }
...
>>> len(digitmap)
460
>>> # Arabic digits
>>> x = '\u0661\u0662\u0663'
>>> x.translate(digitmap)
'123'
>>>
```

I/O
encode() decode()

```
>>> a
'pýt̂ôñ is awesome\n'
>>> b = unicodedata.normalize('NFD', a)
>>> b.encode('ascii', 'ignore').decode('ascii')
'python is awesome\n'
>>>
```

ASCII /

ACSII

4.12.3

str.replace()

```
def clean_spaces(s):
    s = s.replace('\r', '')
    s = s.replace('\t', ' ')
    s = s.replace('\f', ' ')
    return s
```

translate()

tanslate()

4.13 2.13

4.13.1

4.13.2

ljust() , rjust() center()

```
>>> text = 'Hello World'
>>> text.ljust(20)
'Hello World         '
>>> text.rjust(20)
'         Hello World'
>>> text.center(20)
'    Hello World     '
>>>
```

```
>>> text.rjust(20,'=')
'=========Hello World'
>>> text.center(20,'*')
'****Hello World*****'
>>>
```

format() <,>
^

```
>>> format(text, '>20')
'         Hello World'
>>> format(text, '<20')
'Hello World         '
>>> format(text, '^20')
'    Hello World     '
>>>
```

```
>>> format(text, '=>20s')
'=========Hello World'
>>> format(text, '*^20s')
'****Hello World*****'
>>>
```

format()

```
>>> '{:>10s} {:>10s}'.format('Hello', 'World')
'     Hello      World'
>>>
```

format()

```
>>> x = 1.2345
>>> format(x, '>10')
'    1.2345'
>>> format(x, '^10.2f')
'   1.23   '
>>>
```

4.13.3

%

```
>>> '%-20s' % text
'Hello World         '
>>> '%20s' % text
'         Hello World'
>>>
```

format() format()
% format() ljust() , rjust() center()

format() Python

4.14 2.14

4.14.1

4.14.2

iterable
join()

```
>>> parts = ['Is', 'Chicago', 'Not', 'Chicago?']
>>> ' '.join(parts)
'Is Chicago Not Chicago?'
>>> ','.join(parts)
'Is,Chicago,Not,Chicago?'
>>> ''.join(parts)
'IsChicagoNotChicago?'
>>>
```

join()
(
) join()
join()

(+)

```
>>> a = 'Is Chicago'
>>> b = 'Not Chicago?'
>>> a + ' ' + b
'Is Chicago Not Chicago?'
>>>
```

(+)

```
>>> print('{} {}'.format(a,b))
Is Chicago Not Chicago?
>>> print(a + ' ' + b)
Is Chicago Not Chicago?
>>>
```

(+)

```
>>> a = 'Hello' 'World'
>>> a
'HelloWorld'
>>>
```

4.14.3

(+)

```
s = ''
for p in parts:
    s += p
```

join() +=

(1.19)

```
>>> data = ['ACME', 50, 91.1]
>>> ','.join(str(d) for d in data)
'ACME,50,91.1'
>>>
```

```
print(a + ':' + b + ':' + c) # Ugly
print(':'.join([a, b, c])) # Still ugly
print(a, b, c, sep=':') # Better
```

I/O

```
# Version 1 (string concatenation)
f.write(chunk1 + chunk2)

# Version 2 (separate I/O operations)
f.write(chunk1)
f.write(chunk2)
```

I/O

yield

```
def sample():
    yield 'Is'
    yield 'Chicago'
    yield 'Not'
    yield 'Chicago?'
```

join()

```
text = ''.join(sample())
```

I/O

```
for part in sample():
    f.write(part)
```

I/O

```
def combine(source, maxsize):
    parts = []
    size = 0
    for part in source:
        parts.append(part)
        size += len(part)
        if size > maxsize:
            yield ''.join(parts)
            parts = []
            size = 0
        yield ''.join(parts)

#
with open('filename', 'w') as f:
    for part in combine(sample(), 32768):
        f.write(part)
```

4.15 2.15

4.15.1

4.15.2

Python
format()

```
>>> s = '{name} has {n} messages.'
>>> s.format(name='Guido', n=37)
'Guido has 37 messages.'
>>>
```

format_map()

vars()

```
>>> name = 'Guido'
>>> n = 37
>>> s.format_map(vars())
'Guido has 37 messages.'
>>>
```

vars()

```
>>> class Info:
...     def __init__(self, name, n):
...         self.name = name
...         self.n = n
...
>>> a = Info('Guido',37)
>>> s.format_map(vars(a))
'Guido has 37 messages.'
>>>
```

format format_map()

```
>>> s.format(name='Guido')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'n'
>>>
```

__missing__()

```
class safesub(dict):
"""      key       """
def __missing__(self, key):
    return '{' + key + '}'
```

format_map()

```
>>> del n # Make sure n is undefined
>>> s.format_map(safesub(vars()))
'Guido has {n} messages.'
>>>
```

```
import sys

def sub(text):
    return text.format_map(safesub(sys._getframe(1).f_locals))
```

```
>>> name = 'Guido'
>>> n = 37
>>> print(sub('Hello {name}'))
Hello Guido
>>> print(sub('You have {n} messages.'))
You have 37 messages.
>>> print(sub('Your favorite color is {color}'))
Your favorite color is {color}
>>>
```

4.15.3

Python

```
>>> name = 'Guido'
>>> n = 37
>>> '%(name) has %(n) messages.' % vars()
'Guido has 37 messages.'
>>>
```

```
>>> import string
>>> s = string.Template('$name has $n messages.')
>>> s.substitute(vars())
'Guido has 37 messages.'
>>>
```

format() format_map()
format()
( )

__missing__()
SafeSub
(
) KeyError

sub() sys._getframe(1)
f_locals

f_locals f_locals

4.16 2.16

4.16.1

4.16.2

textwrap

```
s = "Look into my eyes, look into my eyes, the eyes, the eyes, \
the eyes, not around the eyes, don't look around the eyes, \
look into my eyes, you're under."
```

textwrap

```
>>> import textwrap
>>> print(textwrap.fill(s, 70))
Look into my eyes, look into my eyes, the eyes, the eyes, the eyes,
not around the eyes, don't look around the eyes, look into my eyes,
you're under.

>>> print(textwrap.fill(s, 40))
Look into my eyes, look into my eyes,
the eyes, the eyes, the eyes, not around
the eyes, don't look around the eyes,
look into my eyes, you're under.

>>> print(textwrap.fill(s, 40, initial_indent='    '))
    Look into my eyes, look into my
eyes, the eyes, the eyes, the eyes, not
```

```
around the eyes, don't look around the
eyes, look into my eyes, you're under.

>>> print(textwrap.fill(s, 40, subsequent_indent='    '))
Look into my eyes, look into my eyes,
    the eyes, the eyes, the eyes, not
    around the eyes, don't look around
    the eyes, look into my eyes, you're
    under.
```

4.16.3

textwrap
os.get_terminal_size()

```
>>> import os
>>> os.get_terminal_size().columns
80
>>>
```

fill() tab tex-
twrap.TextWrapper

4.17 2.17 html xml

4.17.1

HTML XML &entity; &#code;
(<, >, &)

4.17.2

‘<’ ‘>’ html.escape()

```
>>> s = 'Elements are written as "<tag>text</tag>".'
>>> import html
>>> print(s)
Elements are written as "<tag>text</tag>".
>>> print(html.escape(s))
Elements are written as &quot;&lt;tag&gt;text&lt;/tag&gt;&quot;.

>>> # Disable escaping of quotes
>>> print(html.escape(s, quote=False))
```

```
Elements are written as "&lt;tag&gt;text&lt;/tag&gt;".
>>>
```

ASCII ASCII
I/O errors='xmlcharrefreplace'

```
>>> s = 'Spicy Jalapeño'
>>> s.encode('ascii', errors='xmlcharrefreplace')
b'Spicy Jalape&#241;o'
>>>
```

HTML
XML HTML XML

HTML XML /

```
>>> s = 'Spicy &quot;Jalape&#241;o&quot.'
>>> from html.parser import HTMLParser
>>> p = HTMLParser()
>>> p.unescape(s)
'Spicy "Jalapeño".'
>>>
>>> t = 'The prompt is &gt;&gt;&gt;'
>>> from xml.sax.saxutils import unescape
>>> unescape(t)
'The prompt is >>>'
>>>
```

4.17.3

HTML XML
print()
html.escape()

xml.sax.saxutils.unescapge()
HTML XML
html.parse xml.etree.ElementTree

4.18 2.18

4.18.1

4.18.2

```
text = 'foo = 23 + 42 * 10'
```

```
tokens = [('NAME', 'foo'), ('EQ','='), ('NUM', '23'), ('PLUS','+'),
    ('NUM', '42'), ('TIMES', '*'), ('NUM', 10')]
```

```
import re
NAME = r'(?P<NAME>[a-zA-Z_][a-zA-Z_0-9]*)'
NUM = r'(?P<NUM>\d+)'
PLUS = r'(?P<PLUS>\+)'
TIMES = r'(?P<TIMES>\*)'
EQ = r'(?P<EQ>=)'
WS = r'(?P<WS>\s+)'

master_pat = re.compile('|'.join([NAME, NUM, PLUS, TIMES, EQ, WS]))
```

?P<TOKENNAME>

scanner()

scanner match()

scanner

```
>>> scanner = master_pat.scanner('foo = 42')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('NAME', 'foo')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('WS', ' ')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('EQ', '=')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('WS', ' ')
>>> scanner.match()
<_sre.SRE_Match object at 0x100677738>
>>> _.lastgroup, _.group()
('NUM', '42')
```

```
>>> scanner.match()
>>>
```

```
def generate_tokens(pat, text):
Token = namedtuple('Token', ['type', 'value'])
scanner = pat.scanner(text)
for m in iter(scanner.match, None):
    yield Token(m.lastgroup, m.group())

# Example use
for tok in generate_tokens(master_pat, 'foo = 42'):
    print(tok)
# Produces output
# Token(type='NAME', value='foo')
# Token(type='WS', value=' ')
# Token(type='EQ', value='=')
# Token(type='WS', value=' ')
# Token(type='NUM', value='42')
```

```
tokens = (tok for tok in generate_tokens(master_pat, text)
            if tok.type != 'WS')
for tok in tokens:
    print(tok)
```

4.18.3

re

```
LT = r'(?P<LT><)'
LE = r'(?P<LE><=)'
EQ = r'(?P<EQ>=)'

master_pat = re.compile('|'.join([LE, LT, EQ])) # Correct
# master_pat = re.compile('|'.join([LT, LE, EQ])) # Incorrect
```

<= LT EQ

LE

```
PRINT = r'(?P<PRINT>print)'
NAME = r'(?P<NAME>[a-zA-Z_][a-zA-Z_0-9]*)'

master_pat = re.compile('|'.join([PRINT, NAME]))

for tok in generate_tokens(master_pat, 'printer'):
    print(tok)

# Outputs :
# Token(type='PRINT', value='print')
# Token(type='NAME', value='er')
```

PyParsing PLY

PLY

4.19 2.19

4.19.1

4.19.2

BNF EBNF

```
expr ::= expr + term
    |   expr - term
    |   term

term ::= term * factor
    |   term / factor
    |   factor

factor ::= ( expr )
    |   NUM
```

EBNF

```
expr ::= term { (+|-) term }*

term ::= factor { (*|/) factor }*

factor ::= ( expr )
    |   NUM
```

EBNF {...}* * 0 (
)

BNF

BNF
3 + 4 * 5

2.18

```
NUM + NUM * NUM
```

```
expr
expr ::= term { (+|-) term }*
expr ::= factor { (*|/) factor }* { (+|-) term }*
expr ::= NUM { (*|/) factor }* { (+|-) term }*
expr ::= NUM { (+|-) term }*
expr ::= NUM + term { (+|-) term }*
expr ::= NUM + factor { (*|/) factor }* { (+|-) term }*
expr ::= NUM + NUM { (*|/) factor}* { (+|-) term }*
expr ::= NUM + NUM * factor { (*|/) factor }* { (+|-) term }*
expr ::= NUM + NUM * NUM { (*|/) factor }* { (+|-) term }*
expr ::= NUM + NUM * NUM { (+|-) term }*
expr ::= NUM + NUM * NUM
```

NUM
+
({ (*/) factor }*)

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-
"""
Topic:
Desc :
"""
import re
import collections

# Token specification
NUM = r'(?P<NUM>\d+)'
PLUS = r'(?P<PLUS>\+)'
MINUS = r'(?P<MINUS>-)'
TIMES = r'(?P<TIMES>\*)'
DIVIDE = r'(?P<DIVIDE>/)'
LPAREN = r'(?P<LPAREN>\()'
RPAREN = r'(?P<RPAREN>\))'
WS = r'(?P<WS>\s+)'
```

```
master_pat = re.compile('|'.join([NUM, PLUS, MINUS, TIMES,
                                  DIVIDE, LPAREN, RPAREN, WS]))
# Tokenizer
Token = collections.namedtuple('Token', ['type', 'value'])


def generate_tokens(text):
    scanner = master_pat.scanner(text)
    for m in iter(scanner.match, None):
        tok = Token(m.lastgroup, m.group())
        if tok.type != 'WS':
            yield tok


# Parser
class ExpressionEvaluator:
    '''
    Implementation of a recursive descent parser. Each method
    implements a single grammar rule. Use the ._accept() method
    to test and accept the current lookahead token. Use the ._expect()
    method to exactly match and discard the next token on on the input
    (or raise a SyntaxError if it doesn't match).
    '''

    def parse(self, text):
        self.tokens = generate_tokens(text)
        self.tok = None  # Last symbol consumed
        self.nexttok = None  # Next symbol tokenized
        self._advance()  # Load first lookahead token
        return self.expr()

    def _advance(self):
        'Advance one token ahead'
        self.tok, self.nexttok = self.nexttok, next(self.tokens, None)

    def _accept(self, toktype):
        'Test and consume the next token if it matches toktype'
        if self.nexttok and self.nexttok.type == toktype:
            self._advance()
            return True
        else:
            return False

    def _expect(self, toktype):
        'Consume next token if it matches toktype or raise SyntaxError'
        if not self._accept(toktype):
            raise SyntaxError('Expected ' + toktype)

    # Grammar rules follow
    def expr(self):
```

```
        "expression ::= term { ('+'|'-') term }*"
        exprval = self.term()
        while self._accept('PLUS') or self._accept('MINUS'):
            op = self.tok.type
            right = self.term()
            if op == 'PLUS':
                exprval += right
            elif op == 'MINUS':
                exprval -= right
        return exprval

    def term(self):
        "term ::= factor { ('*'|'/') factor }*"
        termval = self.factor()
        while self._accept('TIMES') or self._accept('DIVIDE'):
            op = self.tok.type
            right = self.factor()
            if op == 'TIMES':
                termval *= right
            elif op == 'DIVIDE':
                termval /= right
        return termval

    def factor(self):
        "factor ::= NUM | ( expr )"
        if self._accept('NUM'):
            return int(self.tok.value)
        elif self._accept('LPAREN'):
            exprval = self.expr()
            self._expect('RPAREN')
            return exprval
        else:
            raise SyntaxError('Expected NUMBER or LPAREN')


def descent_parser():
    e = ExpressionEvaluator()
    print(e.parse('2'))
    print(e.parse('2 + 3'))
    print(e.parse('2 + 3 * 4'))
    print(e.parse('2 + (3 + 4) * 5'))
    # print(e.parse('2 + (3 + * 4)'))
    # Traceback (most recent call last):
    #   File "<stdin>", line 1, in <module>
    #   File "exprparse.py", line 40, in parse
    #   return self.expr()
    #   File "exprparse.py", line 67, in expr
    #   right = self.term()
    #   File "exprparse.py", line 77, in term
    #   termval = self.factor()
```

```
    #    File "exprparse.py", line 93, in factor
    #    exprval = self.expr()
    #    File "exprparse.py", line 67, in expr
    #    right = self.term()
    #    File "exprparse.py", line 77, in term
    #    termval = self.factor()
    #    File "exprparse.py", line 97, in factor
    #    raise SyntaxError("Expected NUMBER or LPAREN")
    #    SyntaxError: Expected NUMBER or LPAREN


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

4.19.3

```
expr ::= term { ('+'|'-') term }*

term ::= factor { ('*'|'/') factor }*

factor ::= '(' expr ')'
    | NUM
```

```
class ExpressionEvaluator:
    ...
    def expr(self):
    ...
    def term(self):
    ...
    def factor(self):
    ...
```

-

- (term factor)
” ” -
(factor ::= '('expr
')' expr) ” ”

- (()) _expect()
- (+ -) _accept() _expect() ( )
- (::= term { ('+'|'-') term }*) while
-

Python Python Grammar/Grammar

```
items ::= items ',' item
    | item
```

items()

```
def items(self):
    itemsval = self.items()
    if itemsval and self._accept(','):
        itemsval.append(self.item())
    else:
        itemsval = [ self.item() ]
```

```
expr ::= factor { ('+'|'-'|'*'|'/') factor }*

factor ::= '(' expression ')'
    | NUM
```

"3 + 4 * 5" 35 23. "expr" "term"

PyParsing PLY PLY

```
from ply.lex import lex
from ply.yacc import yacc

# Token list
tokens = [ 'NUM', 'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'LPAREN', 'RPAREN' ]
# Ignored characters
t_ignore = ' \t\n'
# Token specifications (as regexs)
t_PLUS = r'\+'
t_MINUS = r'-'
t_TIMES = r'\*'
t_DIVIDE = r'/'
t_LPAREN = r'\('
t_RPAREN = r'\)'

# Token processing functions
def t_NUM(t):
    r'\d+'
    t.value = int(t.value)
    return t

# Error handler
def t_error(t):
    print('Bad character: {!r}'.format(t.value[0]))
    t.skip(1)

# Build the lexer
lexer = lex()

# Grammar rules and handler functions
def p_expr(p):
    '''
    expr : expr PLUS term
         | expr MINUS term
    '''
    if p[2] == '+':
        p[0] = p[1] + p[3]
    elif p[2] == '-':
        p[0] = p[1] - p[3]


def p_expr_term(p):
    '''
    expr : term
    '''
    p[0] = p[1]


def p_term(p):
    '''
    term : term TIMES factor
```

```
    | term DIVIDE factor
    '''
    if p[2] == '*':
        p[0] = p[1] * p[3]
    elif p[2] == '/':
        p[0] = p[1] / p[3]

def p_term_factor(p):
    '''
    term : factor
    '''
    p[0] = p[1]

def p_factor(p):
    '''
    factor : NUM
    '''
    p[0] = p[1]

def p_factor_group(p):
    '''
    factor : LPAREN expr RPAREN
    '''
    p[0] = p[2]

def p_error(p):
    print('Syntax error')

parser = yacc()
```

```
>>> parser.parse('2')
2
>>> parser.parse('2+3')
5
>>> parser.parse('2+(3+4)*5')
37
>>>
```

Python ast

4.20 2.20

4.20.1

( )

4.20.2

```
>>> data = b'Hello World'
>>> data[0:5]
b'Hello'
>>> data.startswith(b'Hello')
True
>>> data.split()
[b'Hello', b'World']
>>> data.replace(b'Hello', b'Hello Cruel')
b'Hello Cruel World'
>>>
```

```
>>> data = bytearray(b'Hello World')
>>> data[0:5]
bytearray(b'Hello')
>>> data.startswith(b'Hello')
True
>>> data.split()
[bytearray(b'Hello'), bytearray(b'World')]
>>> data.replace(b'Hello', b'Hello Cruel')
bytearray(b'Hello Cruel World')
>>>
```

```
>>>
>>> data = b'FOO:BAR,SPAM'
>>> import re
>>> re.split('[:,]',data)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/usr/local/lib/python3.3/re.py", line 191, in split
return _compile(pattern, flags).split(string, maxsplit)
TypeError: can't use a string pattern on a bytes-like object
>>> re.split(b'[:,]',data) # Notice: pattern as bytes
[b'FOO', b'BAR', b'SPAM']
>>>
```

4.20.3

```
>>> a = 'Hello World' # Text string
>>> a[0]
'H'
>>> a[1]
'e'
>>> b = b'Hello World' # Byte string
>>> b[0]
72
>>> b[1]
101
>>>
```

```
>>> s = b'Hello World'
>>> print(s)
b'Hello World' # Observe b'...'
>>> print(s.decode('ascii'))
Hello World
>>>
```

```
>>> b'%10s %10d %10.2f' % (b'ACME', 100, 490.1)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for %: 'bytes' and 'tuple'
>>> b'{} {} {}'.format(b'ACME', 100, 490.1)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: 'bytes' object has no attribute 'format'
>>>
```

```
>>> '{:10s} {:10d} {:10.2f}'.format('ACME', 100, 490.1).encode('ascii')
b'ACME 100 490.10'
>>>
```

/

```
>>> # Write a UTF-8 filename
>>> with open('jalape\xf1o.txt', 'w') as f:
...     f.write('spicy')
...
>>> # Get a directory listing
>>> import os
>>> os.listdir('.') # Text string (names are decoded)
['jalapeño.txt']
>>> os.listdir(b'.') # Byte string (names left as bytes)
[b'jalapen\xcc\x83o.txt']
>>>
```

UTF-8 5.15

(
Unicode)
Python /

CHAPTER

FIVE

Python

Contents:

5.1 3.1

5.1.1

5.1.2

round(value, ndigits)

```
>>> round(1.23, 1)
1.2
>>> round(1.27, 1)
1.3
>>> round(-1.27, 1)
-1.3
>>> round(1.25361,3)
1.254
>>>
```

round

1.5 2.5 2

round() ndigits

```
>>> a = 1627731
>>> round(a, -1)
1627730
>>> round(a, -2)
```

```
1627700
>>> round(a, -3)
1628000
>>>
```

5.1.3

round()

```
>>> x = 1.23456
>>> format(x, '0.2f')
'1.23'
>>> format(x, '0.3f')
'1.235'
>>> 'value is {:0.3f}'.format(x)
'value is 1.235'
>>>
```

” ”

```
>>> a = 2.1
>>> b = 4.2
>>> c = a + b
>>> c
6.300000000000001
>>> c = round(c, 2) # "Fix" result (???)
>>> c
6.3
>>>
```

( ) decimal

5.2 3.2

5.2.1

5.2.2

```
>>> a = 4.2
>>> b = 2.1
>>> a + b
6.300000000000001
>>> (a + b) == 6.3
False
>>>
```

CPU IEEE 754

Python

( ) decimal

```
>>> from decimal import Decimal
>>> a = Decimal('4.2')
>>> b = Decimal('2.1')
>>> a + b
Decimal('6.3')
>>> print(a + b)
6.3
>>> (a + b) == Decimal('6.3')
True
```

Decimal ( )

decimal

```
>>> from decimal import localcontext
>>> a = Decimal('1.3')
>>> b = Decimal('1.7')
>>> print(a / b)
0.76470588235294117647058823 53
>>> with localcontext() as ctx:
... ctx.prec = 3
... print(a / b)
...
0.765
>>> with localcontext() as ctx:
... ctx.prec = 50
... print(a / b)
...
0.76470588235294117647058823529411764705882352941176
>>>
```

5.2.3

decimal IBM ” ”

Python decimal

17

-

```
>>> nums = [1.23e+18, 1, -1.23e+18]
>>> sum(nums) # Notice how 1 disappears
0.0
>>>
```

math.fsum()

```
>>> import math
>>> math.fsum(nums)
1.0
>>>
```

decimal

decimal

Python Decimal

5.3 3.3

5.3.1

5.3.2

format()

```
>>> x = 1234.56789
```

>ormat(x,

```
>>> # Two decimal places of accuracy
>>> format(x, '0.2f')
'1234.57'

>>> # Right justified in 10 chars, one-digit accuracy
>>> format(x, '>10.1f')
'    1234.6'

>>> # Left justified
>>> format(x, '<10.1f')
'1234.6    '

>>> # Centered
>>> format(x, '^10.1f')
'  1234.6  '

>>> # Inclusion of thousands separator
>>> format(x, ',')
'1,234.56789'
>>> format(x, '0,.1f')
'1,234.6'
>>>
```

f e E( )

```
>>> format(x, 'e')
'1.2   ' 1 $ - 1
```

```
>>> format(-x, '0.1f')
'-1234.6'
>>>
```

locale translate()

```
>>> swap_separators = { ord('.'):',', ord(','):'.' }
>>> format(x, ',').translate(swap_separators)
'1.234,56789'
>>>
```

Python %

```
>>> '%0.2f' % x
'1234.57'
>>> '%10.1f' % x
'    1234.6'
>>> '%-10.1f' % x
'1234.6    '
>>>
```

format()

% ( )

5.4 3.4

5.4.1

5.4.2

bin() ,

oct() hex()

```
>>> x = 1234
>>> bin(x)
'0b10011010010'
>>> oct(x)
'0o2322'
>>> hex(x)
'0x4d2'
>>>
```

0b , 0o 0x format()

```
>>> format(x, 'b')
'10011010010'
>>> format(x, 'o')
'2322'
>>> format(x, 'x')
'4d2'
>>>
```

```
>>> x = -1234
>>> format(x, 'b')
'-10011010010'
>>> format(x, 'x')
'-4d2'
>>>
```

32

```
>>> x = -1234
>>> format(2**32 + x, 'b')
'11111111111111111111101100101110'
>>> format(2**32 + x, 'x')
'fffffb2e'
>>>
```

int()

```
>>> int('4d2', 16)
1234
>>> int('10011010010', 2)
1234
>>>
```

5.4.3

Python

```
>>> import os
>>> os.chmod('script.py', 0755)
    File "<stdin>", line 1
        os.chmod('script.py', 0755)
                              ^
SyntaxError: invalid token
>>>
```

0o

```
>>> os.chmod('script.py', 0o755)
>>>
```

5.5 3.5

5.5.1

5.5.2

128 16

```
data = b'\x00\x124V\x00x\x90\xab\x00\xcd\xef\x01\x00#\x004'
```

bytes int.from_bytes()

```
>>> len(data)
16
>>> int.from_bytes(data, 'little')
69120565665751139577663547927094891008
>>> int.from_bytes(data, 'big')
94522842520747284487117727783387188
>>>
```

int.to_bytes()

```
>>> x = 94522842520747284487117727783387188
>>> x.to_bytes(16, 'big')
b'\x00\x124V\x00x\x90\xab\x00\xcd\xef\x01\x00#\x004'
>>> x.to_bytes(16, 'little')
b'4\x00#\x00\x01\xef\xcd\x00\xab\x90x\x00V4\x12\x00'
>>>
```

5.5.3

IPv6 128

6.11 struct

struct

```
>>> data
b'\x00\x124V\x00x\x90\xab\x00\xcd\xef\x01\x00#\x004'
>>> import struct
>>> hi, lo = struct.unpack('>QQ', data)
>>> (hi << 64) + lo
94522842520747284487117727783387188
>>>
```

(little big)
16

```
>>> x = 0x01020304
>>> x.to_bytes(4, 'big')
b'\x01\x02\x03\x04'
>>> x.to_bytes(4, 'little')
b'\x04\x03\x02\x01'
>>>
```

int.bit_length()

```
>>> x = 523 ** 23
>>> x
335381300113661875107536852714019056160355655333978849017944067
>>> x.to_bytes(16, 'little')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
OverflowError: int too big to convert
>>> x.bit_length()
208
>>> nbytes, rem = divmod(x.bit_length(), 8)
>>> if rem:
... nbytes += 1
...
>>>
>>> x.to_bytes(nbytes, 'little')
b'\x03X\xf1\x82iT\x96\xac\xc7c\x16\xf3\xb9\xcf...\xd0'
>>>
```

5.6 3.6

5.6.1

5.6.2

complex(real, imag) j

```
>>> a = complex(2, 4)
>>> b = 3 - 5j
>>> a
(2+4j)
>>> b
(3-5j)
>>>
```

```
>>> a.real
2.0
>>> a.imag
4.0
>>> a.conjugate()
(2-4j)
>>>
```

```
>>> a + b
(5-1j)
>>> a * b
(26+2j)
>>> a / b
(-0.4117647058823529+0.6470588235294118j)
>>> abs(a)
4.47213595499958
>>>
```

cmath

```
>>> import cmath
>>> cmath.sin(a)
(24.83130584894638-11.356612711218174j)
>>> cmath.cos(a)
(-11.36423470640106-24.814651485634187j)
>>> cmath.exp(a)
(-4.829809383269385-5.5920560936409816j)
>>>
```

5.6.3

Python numpy

```
>>> import numpy as np
>>> a = np.array([2+3j, 4+5j, 6-7j, 8+9j])
>>> a
array([ 2.+3.j, 4.+5.j, 6.-7.j, 8.+9.j])
>>> a + 2
array([ 4.+3.j, 6.+5.j, 8.-7.j, 10.+9.j])
>>> np.sin(a)
array([ 9.15449915 -4.16890696j, -56.16227422 -48.50245524j,
        -153.20827755-526.47684926j, 4008.42651446-589.49948373j])
>>>
```

Python

```
>>> import math
>>> math.sqrt(-1)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ValueError: math domain error
>>>
```

cmath

```
>>> import cmath
>>> cmath.sqrt(-1)
1j
>>>
```

5.7 3.7 NaN

5.7.1

NaN( )

5.7.2

Python float()

```
>>> a = float('inf')
>>> b = float('-inf')
>>> c = float('nan')
>>> a
inf
>>> b
-inf
>>> c
```

```
nan
>>>
```

math.isinf() math.isnan()

```
>>> math.isinf(a)
True
>>> math.isnan(c)
True
>>>
```

5.7.3

IEEE 754

```
>>> a = float('inf')
>>> a + 45
inf
>>> a * 10
inf
>>> 10 / a
0.0
>>>
```

NaN

```
>>> a = float('inf')
>>> a/a
nan
>>> b = float('-inf')
>>> a + b
nan
>>>
```

NaN

```
>>> c = float('nan')
>>> c + 23
nan
>>> c / 2
nan
>>> c * 2
nan
>>> math.sqrt(c)
nan
>>>
```

NaN False

```
>>> c = float('nan')
>>> d = float('nan')
>>> c == d
False
>>> c is d
False
>>>
```

NaN math.isnan()

Python NaN
fpectl Python
Python

5.8 3.8

5.8.1

5.8.2

fractions

```
>>> from fractions import Fraction
>>> a = Fraction(5, 4)
>>> b = Fraction(7, 16)
>>> print(a + b)
27/16
>>> print(a * b)
35/64

>>> # Getting numerator/denominator
>>> c = a * b
>>> c.numerator
35
>>> c.denominator
64

>>> # Converting to a float
>>> float(c)
0.546875

>>> # Limiting the denominator of a value
```

```
>>> print(c.limit_denominator(8))
4/7

>>> # Converting a float to a fraction
>>> x = 3.75
>>> y = Fraction(*x.as_integer_ratio())
>>> y
Fraction(15, 4)
>>>
```

5.8.3

5.9 3.9

5.9.1

( )

5.9.2

NumPy NumPy
Python Python
NumPy

```
>>> # Python lists
>>> x = [1, 2, 3, 4]
>>> y = [5, 6, 7, 8]
>>> x * 2
[1, 2, 3, 4, 1, 2, 3, 4]
>>> x + 10
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
>>> x + y
[1, 2, 3, 4, 5, 6, 7, 8]

>>> # Numpy arrays
>>> import numpy as np
>>> ax = np.array([1, 2, 3, 4])
>>> ay = np.array([5, 6, 7, 8])
>>> ax * 2
```

```
array([2, 4, 6, 8])
>>> ax + 10
array([11, 12, 13, 14])
>>> ax + ay
array([ 6, 8, 10, 12])
>>> ax * ay
array([ 5, 12, 21, 32])
>>>
```

NumPy

(ax * 2 ax + 10)

```
>>> def f(x):
... return 3*x**2 - 2*x + 7
...
>>> f(ax)
array([ 8, 15, 28, 47])
>>>
```

NumPy math

```
>>> np.sqrt(ax)
array([ 1. , 1.41421356, 1.73205081, 2. ])
>>> np.cos(ax)
array([ 0.54030231, -0.41614684, -0.9899925 , -0.65364362])
>>>
```

math

NumPy

NumPy C Fortran

Python 10,000*10,000

```
>>> grid = np.zeros(shape=(10000,10000), dtype=float)
>>> grid
    array([[ 0., 0., 0., ..., 0., 0., 0.],
    [ 0., 0., 0., ..., 0., 0., 0.],
    [ 0., 0., 0., ..., 0., 0., 0.],
    ...,
    [ 0., 0., 0., ..., 0., 0., 0.],
    [ 0., 0., 0., ..., 0., 0., 0.],
    [ 0., 0., 0., ..., 0., 0., 0.]])
>>>
```

```
>>> grid += 10
>>> grid
array([[ 10., 10., 10., ..., 10., 10., 10.],
    [ 10., 10., 10., ..., 10., 10., 10.],
    [ 10., 10., 10., ..., 10., 10., 10.],
    ...,
    [ 10., 10., 10., ..., 10., 10., 10.],
    [ 10., 10., 10., ..., 10., 10., 10.],
    [ 10., 10., 10., ..., 10., 10., 10.]])
>>> np.sin(grid)
array([[-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
        -0.54402111, -0.54402111],
    [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
        -0.54402111, -0.54402111],
    [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
        -0.54402111, -0.54402111],
    ...,
    [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
        -0.54402111, -0.54402111],
    [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
        -0.54402111, -0.54402111],
    [-0.54402111, -0.54402111, -0.54402111, ..., -0.54402111,
        -0.54402111, -0.54402111]])
>>>
```

NumPy Python -

```
>>> a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
>>> a
array([[ 1, 2, 3, 4],
[ 5, 6, 7, 8],
[ 9, 10, 11, 12]])

>>> # Select row 1
>>> a[1]
array([5, 6, 7, 8])

>>> # Select column 1
>>> a[:,1]
array([ 2, 6, 10])

>>> # Select a subregion and change it
>>> a[1:3, 1:3]
array([[ 6, 7],
        [10, 11]])
>>> a[1:3, 1:3] += 10
>>> a
array([[ 1, 2, 3, 4],
        [ 5, 16, 17, 8],
        [ 9, 20, 21, 12]])
```

```
>>> # Broadcast a row vector across an operation on all rows
>>> a + [100, 101, 102, 103]
array([[101, 103, 105, 107],
       [105, 117, 119, 111],
       [109, 121, 123, 115]])
>>> a
array([[ 1, 2, 3, 4],
       [ 5, 16, 17, 8],
       [ 9, 20, 21, 12]])

>>> # Conditional assignment on an array
>>> np.where(a < 10, a, 10)
array([[ 1, 2, 3, 4],
       [ 5, 10, 10, 8],
       [ 9, 10, 10, 10]])
>>>
```

5.9.3

NumPy Python

NumPy import numpy as np
numpy np

NumPy http://www.numpy.org

5.10 3.10

5.10.1

5.10.2

NumPy

3.9

```
>>> import numpy as np
>>> m = np.matrix([[1,-2,3],[0,4,5],[7,8,-9]])
>>> m
```

```
matrix([[ 1, -2, 3],
        [ 0, 4, 5],
        [ 7, 8, -9]])

>>> # Return transpose
>>> m.T
matrix([[ 1, 0, 7],
        [-2, 4, 8],
        [ 3, 5, -9]])

>>> # Return inverse
>>> m.I
matrix([[ 0.33043478, -0.02608696, 0.09565217],
        [-0.15217391, 0.13043478, 0.02173913],
        [ 0.12173913, 0.09565217, -0.0173913 ]])

>>> # Create a vector and multiply
>>> v = np.matrix([[2],[3],[4]])
>>> v
matrix([[2],
        [3],
        [4]])
>>> m * v
matrix([[ 8],
        [32],
        [ 2]])
>>>
```

numpy.linalg

```
>>> import numpy.linalg

>>> # Determinant
>>> numpy.linalg.det(m)
-229.99999999999983

>>> # Eigenvalues
>>> numpy.linalg.eigvals(m)
array([-13.11474312, 2.75956154, 6.35518158])

>>> # Solve for x in mx = v
>>> x = numpy.linalg.solve(m, v)
>>> x
matrix([[ 0.96521739],
        [ 0.17391304],
        [ 0.46086957]])
>>> m * x
matrix([[ 2.],
        [ 3.],
        [ 4.]])
>>> v
```

```
matrix([[2],
        [3],
        [4]])
>>>
```

5.10.3

NumPy NumPy http://www.numpy.org

5.11 3.11

5.11.1

5.11.2

random

random.choice()

```
>>> import random
>>> values = [1, 2, 3, 4, 5, 6]
>>> random.choice(values)
2
>>> random.choice(values)
3
>>> random.choice(values)
1
>>> random.choice(values)
4
>>> random.choice(values)
6
>>>
```

N random.sample()

```
>>> random.sample(values, 2)
[6, 2]
>>> random.sample(values, 2)
[4, 3]
>>> random.sample(values, 3)
[4, 3, 1]
>>> random.sample(values, 3)
```

```
[5, 4, 1]
>>>
```

random.shuffle()

```
>>> random.shuffle(values)
>>> values
[2, 4, 6, 5, 3, 1]
>>> random.shuffle(values)
>>> values
[3, 5, 2, 1, 6, 4]
>>>
```

random.randint()

```
>>> random.randint(0,10)
2
>>> random.randint(0,10)
5
>>> random.randint(0,10)
0
>>> random.randint(0,10)
7
>>> random.randint(0,10)
10
>>> random.randint(0,10)
3
>>>
```

0 1 random.random()

```
>>> random.random()
0.9406677561675867
>>> random.random()
0.133129581343897
>>> random.random()
0.4144991136919316
>>>
```

N ( ) random.getrandbits()

```
>>> random.getrandbits(200)
335837000776573622800628485064121869519521710558559406913275
>>>
```

5.11.3

random *Mersenne Twister*
random.seed()

```
random.seed() # Seed based on system time or os.urandom()
random.seed(12345) # Seed based on integer given
random.seed(b'bytedata') # Seed based on byte data
```

random
random.uniform() random.gauss()

random
ssl ssl.RAND_bytes()

5.12 3.12

5.12.1

5.12.2

datetime
timedelta

```
>>> from datetime import timedelta
>>> a = timedelta(days=2, hours=6)
>>> b = timedelta(hours=4.5)
>>> c = a + b
>>> c.days
2
>>> c.seconds
37800
>>> c.seconds / 3600
10.5
>>> c.total_seconds() / 3600
58.5
>>>
```

datetime

```
>>> from datetime import datetime
>>> a = datetime(2012, 9, 23)
>>> print(a + timedelta(days=10))
2012-10-03 00:00:00
>>>
>>> b = datetime(2012, 12, 21)
>>> d = b - a
```

```
>>> d.days
89
>>> now = datetime.today()
>>> print(now)
2012-12-21 14:54:43.094063
>>> print(now + timedelta(minutes=10))
2012-12-21 15:04:43.094063
>>>
```

datetime

```
>>> a = datetime(2012, 3, 1)
>>> b = datetime(2012, 2, 28)
>>> a - b
datetime.timedelta(2)
>>> (a - b).days
2
>>> c = datetime(2013, 3, 1)
>>> d = datetime(2013, 2, 28)
>>> (c - d).days
1
>>>
```

5.12.3

datetime

dateutil

dateutil.relativedelta()
( )

```
>>> a = datetime(2012, 9, 23)
>>> a + timedelta(months=1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'months' is an invalid keyword argument for this function
>>>
>>> from dateutil.relativedelta import relativedelta
>>> a + relativedelta(months=+1)
datetime.datetime(2012, 10, 23, 0, 0)
>>> a + relativedelta(months=+4)
datetime.datetime(2013, 1, 23, 0, 0)
>>>
>>> # Time between two dates
>>> b = datetime(2012, 12, 21)
>>> d = b - a
>>> d
datetime.timedelta(89)
```

```
>>> d = relativedelta(b, a)
>>> d
relativedelta(months=+2, days=+28)
>>> d.months
2
>>> d.days
28
>>>
```

5.13 3.13

5.13.1

5.13.2

Python datetime

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-
"""
Topic:
Desc :
"""
from datetime import datetime, timedelta

weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
            'Friday', 'Saturday', 'Sunday']


def get_previous_byday(dayname, start_date=None):
    if start_date is None:
        start_date = datetime.today()
    day_num = start_date.weekday()
    day_num_target = weekdays.index(dayname)
    days_ago = (7 + day_num - day_num_target) % 7
    if days_ago == 0:
        days_ago = 7
    target_date = start_date - timedelta(days=days_ago)
    return target_date
```

```
>>> datetime.today() # For reference
datetime.datetime(2012, 8, 28, 22, 4, 30, 263076)
```

```
>>> get_previous_byday('Monday')
datetime.datetime(2012, 8, 27, 22, 3, 57, 29045)
>>> get_previous_byday('Tuesday') # Previous week, not today
datetime.datetime(2012, 8, 21, 22, 4, 12, 629771)
>>> get_previous_byday('Friday')
datetime.datetime(2012, 8, 24, 22, 5, 9, 911393)
>>>
```

start_date datetime

```
>>> get_previous_byday('Sunday', datetime(2012, 12, 21))
datetime.datetime(2012, 12, 16, 0, 0)
>>>
```

5.13.3

(
0)

python-dateutil
dateutil relativedelta()

```
>>> from datetime import datetime
>>> from dateutil.relativedelta import relativedelta
>>> from dateutil.rrule import *
>>> d = datetime.now()
>>> print(d)
2012-12-23 16:31:52.718111

>>> # Next Friday
>>> print(d + relativedelta(weekday=FR))
2012-12-28 16:31:52.718111
>>>

>>> # Last Friday
>>> print(d + relativedelta(weekday=FR(-1)))
2012-12-21 16:31:52.718111
>>>
```

5.14 3.14

5.14.1

5.14.2

datetime.timedelta

datetime

```
from datetime import datetime, date, timedelta
import calendar

def get_month_range(start_date=None):
    if start_date is None:
        start_date = date.today().replace(day=1)
    _, days_in_month = calendar.monthrange(start_date.year, start_date.month)
    end_date = start_date + timedelta(days=days_in_month)
    return (start_date, end_date)
```

```
>>> a_day = timedelta(days=1)
>>> first_day, last_day = get_month_range()
>>> while first_day < last_day:
...     print(first_day)
...     first_day += a_day
...
2012-08-01
2012-08-02
2012-08-03
2012-08-04
2012-08-05
2012-08-06
2012-08-07
2012-08-08
2012-08-09
#... and so on...
```

5.14.3

date
datetime replace() days 1 replace()

date date datetime
datetime

calendar.monthrange()
calendar monthrange()

(
) Python slice range

timedelta <

range()

```
def date_range(start, stop, step):
    while start < stop:
        yield start
        start += step
```

```
>>> for d in date_range(datetime(2012, 9, 1), datetime(2012,10,1),
                        timedelta(hours=6)):
...     print(d)
...
2012-09-01 00:00:00
2012-09-01 06:00:00
2012-09-01 12:00:00
2012-09-01 18:00:00
2012-09-02 00:00:00
2012-09-02 06:00:00
...
>>>
```

Python

5.15 3.15

5.15.1

datetime

5.15.2

Python datetime

```
>>> from datetime import datetime
>>> text = '2012-09-20'
>>> y = datetime.strptime(text, '%Y-%m-%d')
>>> z = datetime.now()
>>> diff = z - y
```

```
>>> diff
datetime.timedelta(3, 77824, 177393)
>>>
```

5.15.3

datetime.strptime() %Y 4 %m

datetime

```
>>> z
datetime.datetime(2012, 9, 23, 21, 37, 4, 177393)
>>> nice_z = datetime.strftime(z, '%A %B %d, %Y')
>>> nice_z
'Sunday September 23, 2012'
>>>
```

strptime()

Python

YYYY-MM-DD

```
from datetime import datetime
def parse_ymd(s):
    year_s, mon_s, day_s = s.split('-')
    return datetime(int(year_s), int(mon_s), int(day_s))
```

datetime.strptime() 7

5.16 3.16

5.16.1

2012 12 21 9:30

5.16.2

pytz Olson

pytz datetime

```
>>> from datetime import datetime
>>> from pytz import timezone
>>> d = datetime(2012, 12, 21, 9, 30, 0)
>>> print(d)
2012-12-21 09:30:00
>>>

>>> # Localize the date for Chicago
>>> central = timezone('US/Central')
>>> loc_d = central.localize(d)
>>> print(loc_d)
2012-12-21 09:30:00-06:00
>>>
```

```
>>> # Convert to Bangalore time
>>> bang_d = loc_d.astimezone(timezone('Asia/Kolkata'))
>>> print(bang_d)
2012-12-21 21:00:00+05:30
>>>
```

2013 3 13 2:00(
)

```
>>> d = datetime(2013, 3, 10, 1, 45)
>>> loc_d = central.localize(d)
>>> print(loc_d)
2013-03-10 01:45:00-06:00
>>> later = loc_d + timedelta(minutes=30)
>>> print(later)
2013-03-10 02:15:00-06:00 # WRONG! WRONG!
>>>
```

normalize()

```
>>> from datetime import timedelta
>>> later = central.normalize(loc_d + timedelta(minutes=30))
>>> print(later)
2013-03-10 03:15:00-05:00
>>>
```

5.16.3

UTC

```
>>> print(loc_d)
2013-03-10 01:45:00-06:00
>>> utc_d = loc_d.astimezone(pytz.utc)
>>> print(utc_d)
2013-03-10 07:45:00+00:00
>>>
```

UTC

```
>>> later_utc = utc_d + timedelta(minutes=30)
>>> print(later_utc.astimezone(central))
2013-03-10 03:15:00-05:00
>>>
```

“ Asia/Kolkata ”
ISO 3166 `pytz.country_timezones`

```
>>> pytz.country_timezones['IN']
['Asia/Kolkata']
>>>
```

`pytz` PEP431
(
UTC)

CHAPTER

SIX

Python

itertools

Contents:

6.1 4.1

6.1.1

for

6.1.2

next() StopIteration

```
def manual_iter():
    with open
```

```
            break
        print(line, end='')
```

6.1.3

for

```
>>> items = [1, 2, 3]
>>> # Get the iterator
>>> it = iter(items) # Invokes items.__iter__()
>>> # Run the iterator
>>> next(it) # Invokes it.__next__()
1
>>> next(it)
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
StopIteration
>>>
```

6.2 4.2

6.2.1

6.2.2

__iter__()

```
class Node:
    def __init__(self, value):
        self._value = value
        self._children = []
```

```
    def __repr__(self):
        return 'Node({!r})'.format(self._value)

    def add_child(self, node):
        self._children.append(node)

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

# Example
if __name__ == '__main__':
    root = Node(0)
    child1 = Node(1)
    child2 = Node(2)
    root.add_child(child1)
    root.add_child(child2)
    # Outputs Node(1), Node(2)
    for ch in root:
        print(ch)
```

__iter__() _children

6.2.3

Python __iter__() __next__()

iter() iter(s)
s.__iter__() len(s) s.__len__()

6.3 4.3

6.3.1

range() , reversed()

6.3.2

```
def frange(start, stop, increment):
    x = start
    while x < stop:
        yield x
        x += increment
```

for

(sum() , list())

```
>>> for n in frange(0, 4, 0.5):
...     print(n)
...
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
>>> list(frange(0, 1, 0.125))
[0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875]
>>>
```

6.3.3

yield

```
>>> def countdown(n):
...     print('Starting to count from', n)
...     while n > 0:
...         yield n
...         n -= 1
...     print('Done!')
...

>>> # Create the generator, notice no output appears
>>> c = countdown(3)
>>> c
<generator object countdown at 0x1006a0af0>

>>> # Run to first yield and emit a value
>>> next(c)
Starting to count from 3
3

>>> # Run to the next yield
>>> next(c)
```

```
2

>>> # Run to next yield
>>> next(c)
1

>>> # Run to next yield (iteration stops)
>>> next(c)
Done!
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
StopIteration
>>>
```

next

for

6.4 4.4

6.4.1

6.4.2

4.2

Node

```
class Node:
    def __init__(self, value):
        self._value = value
        self._children = []

    def __repr__(self):
        return 'Node({!r})'.format(self._value)

    def add_child(self, node):
        self._children.append(node)

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

    def depth_first(self):
        yield self
```

```
        for c in self:
            yield from c.depth_first()

# Example
if __name__ == '__main__':
    root = Node(0)
    child1 = Node(1)
    child2 = Node(2)
    root.add_child(child1)
    root.add_child(child2)
    child1.add_child(Node(3))
    child1.add_child(Node(4))
    child2.add_child(Node(5))

    for ch in root.depth_first():
        print(ch)
    # Outputs Node(0), Node(1), Node(3), Node(4), Node(2), Node(5)
```

depth_first()
depth_first() (yield from)

6.4.3

Python __iter__()
__next__() StopIteration

depth_first()

```
class Node2:
    def __init__(self, value):
        self._value = value
        self._children = []

    def __repr__(self):
        return 'Node({!r})'.format(self._value)

    def add_child(self, node):
        self._children.append(node)

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

    def depth_first(self):
        return DepthFirstIterator(self)


class DepthFirstIterator(object):
    '''
```

```
    Depth-first traversal
    '''

    def __init__(self, start_node):
        self._node = start_node
        self._children_iter = None
        self._child_iter = None

    def __iter__(self):
        return self

    def __next__(self):
        # Return myself if just started; create an iterator for children
        if self._children_iter is None:
            self._children_iter = iter(self._node)
            return self._node
        # If processing a child, return its next item
        elif self._child_iter:
            try:
                nextchild = next(self._child_iter)
                return nextchild
            except StopIteration:
                self._child_iter = None
                return next(self)
        # Advance to the next child and start its iteration
        else:
            self._child_iter = next(self._children_iter).depth_first()
            return next(self)
```

DepthFirstIterator

6.5 4.5

6.5.1

6.5.2

reversed()

```
>>> a = [1, 2, 3, 4]
>>> for x in reversed(a):
...     print(x)
...
```

```
4
3
2
1
```

__reversed__()

```
# Print a file backwards
f = open('somefile')
for line in reversed(list(f)):
    print(line, end='')
```

6.5.3

__reversed__()

```
class Countdown:
    def __init__(self, start):
        self.start = start

    # Forward iterator
    def __iter__(self):
        n = self.start
        while n > 0:
            yield n
            n -= 1

    # Reverse iterator
    def __reversed__(self):
        n = 1
        while n <= self.start:
            yield n
            n += 1

for rr in reversed(Countdown(30)):
    print(rr)
for rr in Countdown(30):
    print(rr)
```

6.6 4.6

6.6.1

6.6.2

__iter__()

```
from collections import deque

class linehistory:
    def __init__(self, lines, histlen=3):
        self.lines = lines
        self.history = deque(maxlen=histlen)

    def __iter__(self):
        for lineno, line in enumerate(self.lines, 1):
            self.history.append((lineno, line))
            yield line

    def clear(self):
        self.history.clear()
```

history clear()

```
with open('somefile.txt') as f:
    lines = linehistory(f)
    for line in lines:
        if 'python' in line:
            for lineno, hline in lines.history:
                print('{}:{}'.format(lineno, hline), end='')
```

6.6.3

( )

__iter__()

for

iter()

```
>>> f = open('somefile.txt')
>>> lines = linehistory(f)
>>> next(lines)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: 'linehistory' object is not an iterator

>>> # Call iter() first, then start iterating
>>> it = iter(lines)
>>> next(it)
'hello world\n'
>>> next(it)
'this is a test\n'
>>>
```

6.7 4.7

6.7.1

6.7.2

itertools.islice()

```
>>> def count(n):
...     while True:
...         yield n
...         n += 1
...
>>> c = count(0)
>>> c[10:20]
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: 'generator' object is not subscriptable

>>> # Now using islice()
>>> import itertools
>>> for x in itertools.islice(c, 10, 20):
...     print(x)
...
10
11
12
13
14
15
```

```
16
17
18
19
>>>
```

6.7.3

(

) islice()

islice()

6.8 4.8

6.8.1

6.8.2

itertools
itertools.dropwhile()

True

```
>>> with open('/etc/passwd') as f:
... for line in f:
...     print(line, end='')
...
##
# User Database
#
# Note that this file is consulted directly only when the system is running
# in single-user mode. At other times, this information is provided by
# Open Directory.
...
##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
```

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

```
>>> from itertools import dropwhile
>>> with open('/etc/passwd') as f:
...     for line in dropwhile(lambda line: line.startswith('#'), f):
...         print(line, end='')
...
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
...
>>>
```

itertools.islice()

```
>>> from itertools import islice
>>> items = ['a', 'b', 'c', 1, 4, 10, 15]
>>> for x in islice(items, 3, None):
...     print(x)
...
1
4
10
15
>>>
```

islice() None 3
None 3
([3:] [:3])

6.8.3

dropwhile() islice()

```
with open('/etc/passwd') as f:
    # Skip over initial comments
    while True:
        line = next(f, '')
        if not line.startswith('#'):
            break

    # Process remaining lines
    while line:
        # Replace with useful processing
        print(line, end='')
        line = next(f, None)
```

```
with open('/etc/passwd') as f:
    lines = (line for line in f if not line.startswith('#'))
    for line in lines:
        print(line, end='')
```

6.9 4.9

6.9.1

6.9.2

itertools
itertools.permutations()

```
>>> items = ['a', 'b', 'c']
>>> from itertools import permutations
>>> for p in permutations(items):
...     print(p)
...
('a', 'b', 'c')
('a', 'c', 'b')
('b', 'a', 'c')
('b', 'c', 'a')
('c', 'a', 'b')
('c', 'b', 'a')
>>>
```

```
>>> for p in permutations(items, 2):
...     print(p)
...
('a', 'b')
('a', 'c')
```

```
('b', 'a')
('b', 'c')
('c', 'a')
('c', 'b')
>>>
```

itertools.combinations()

```
>>> from itertools import combinations
>>> for c in combinations(items, 3):
...     print(c)
...
('a', 'b', 'c')

>>> for c in combinations(items, 2):
...     print(c)
...
('a', 'b')
('a', 'c')
('b', 'c')

>>> for c in combinations(items, 1):
...     print(c)
...
('a',)
('b',)
('c',)
>>>
```

combinations() ('a', 'b') ('b', 'a') ( )

('a')

itertools.combinations_with_replacement()

```
>>> for c in combinations_with_replacement(items, 3):
...     print(c)
...
('a', 'a', 'a')
('a', 'a', 'b')
('a', 'a', 'c')
('a', 'b', 'b')
('a', 'b', 'c')
('a', 'c', 'c')
('b', 'b', 'b')
('b', 'b', 'c')
('b', 'c', 'c')
('c', 'c', 'c')
>>>
```

6.9.3

itertools

itertools

6.10 4.10

6.10.1

6.10.2

enumerate()

```
>>> my_list = ['a', 'b', 'c']
>>> for idx, val in enumerate(my_list):
...     print(idx, val)
...
0 a
1 b
2 c
```

(1)

```
>>> my_list = ['a', 'b', 'c']
>>> for idx, val in enumerate(my_list, 1):
...     print(idx, val)
...
1 a
2 b
3 c
```

```
def parse_data(filename):
    with open(filename, 'rt') as f:
        for lineno, line in enumerate(f, 1):
            fields = line.split()
            try:
                count = int(fields[1])
                ...
            except ValueError as e:
                print('Line {}: Parse error: {}'.format(lineno, e))
```

enumerate()

enumerate()

```
word_summary = defaultdict(list)

with open('myfile.txt', 'r') as f:
    lines = f.readlines()

for idx, line in enumerate(lines):
    # Create a list of words in current line
    words = [w.strip().lower() for w in line.split()]
    for word in words:
        word_summary[word].append(idx)
```

word_summary (
defaultdict) key key

6.10.3

enumerate()

```
lineno = 1
for line in f:
    # Process line
    ...
    lineno += 1
```

enumerate()

```
for lineno, line in enumerate(f):
    # Process line
    ...
```

enumerate() enumerate
next()

enumerate()

```
data = [ (1, 2), (3, 4), (5, 6), (7, 8) ]

# Correct!
for n, (x, y) in enumerate(data):
    ...
# Error!
for n, x, y in enumerate(data):
    ...
```

6.11 4.11

6.11.1

6.11.2

zip()

```
>>> xpts = [1, 5, 4, 2, 10, 7]
>>> ypts = [101, 78, 37, 15, 62, 99]
>>> for x, y in zip(xpts, ypts):
...     print(x,y)
...
1 101
5 78
4 37
2 15
10 62
7 99
>>>
```

zip(a, b) (x, y) x a y b

```
>>> a = [1, 2, 3]
>>> b = ['w', 'x', 'y', 'z']
>>> for i in zip(a,b):
...     print(i)
...
(1, 'w')
(2, 'x')
(3, 'y')
>>>
```

itertools.zip_longest()

```
>>> from itertools import zip_longest
>>> for i in zip_longest(a,b):
...     print(i)
...
(1, 'w')
(2, 'x')
(3, 'y')
(None, 'z')

>>> for i in zip_longest(a, b, fillvalue=0):
```

```
...     print(i)
...
(1, 'w')
(2, 'x')
(3, 'y')
(0, 'z')
>>>
```

6.11.3

zip()

```
headers = ['name', 'shares', 'price']
values = ['ACME', 100, 490.1]
```

zip()

```
s = dict(zip(headers,values))
```

```
for name, val in zip(headers, values):
    print(name, '=', val)
```

zip()

;

```
>>> a = [1, 2, 3]
>>> b = [10, 11, 12]
>>> c = ['x','y','z']
>>> for i in zip(a, b, c):
...     print(i)
...
(1, 10, 'x')
(2, 11, 'y')
(3, 12, 'z')
>>>
```

zip()
list()

```
>>> zip(a, b)
<zip object at 0x1007001b8>
>>> list(zip(a, b))
[(1, 10), (2, 11), (3, 12)]
>>>
```

6.12 4.12

6.12.1

6.12.2

itertools.chain()

```
>>> from itertools import chain
>>> a = [1, 2, 3, 4]
>>> b = ['x', 'y', 'z']
>>> for x in chain(a, b):
... print(x)
...
1
2
3
4
x
y
z
>>>
```

chain()

```
# Various working sets of items
active_items = set()
inactive_items = set()

# Iterate over all items
for item in chain(active_items, inactive_items):
    # Process item
```

```
for item in active_items:
    # Process item
    ...

for item in inactive_items:
    # Process item
    ...
```

6.12.3

itertools.chain()

```
# Inefficent
for x in a + b:
    ...

# Better
for x in chain(a, b):
    ...
```

a + b a b
chian()
chain()

6.13 4.13

6.13.1

(Unix)

6.13.2

```
foo/
    access-log-012007.gz
    access-log-022007.gz
    access-log-032007.gz
    ...
    access-log-012008
bar/
    access-log-092007.bz2
    ...
    access-log-022008
```

```
124.115.6.12 - - [10/Jul/2012:00:18:50 -0500] "GET /robots.txt ..." 200 71
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET /ply/ ..." 200 11875
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET /favicon.ico ..." 404 369
61.135.216.105 - - [10/Jul/2012:00:20:04 -0500] "GET /blog/atom.xml ..." 304 -
...
```

```python
import os
import fnmatch
import gzip
import bz2
import re

def gen_find(filepat, top):
    '''
    Find all filenames in a directory tree that match a shell wildcard pattern
    '''
    for path, dirlist, filelist in os.walk(top):
        for name in fnmatch.filter(filelist, filepat):
            yield os.path.join(path,name)

def gen_opener(filenames):
    '''
    Open a sequence of filenames one at a time producing a file object.
    The file is closed immediately when proceeding to the next iteration.
    '''
    for filename in filenames:
        if filename.endswith('.gz'):
            f = gzip.open(filename, 'rt')
        elif filename.endswith('.bz2'):
            f = bz2.open(filename, 'rt')
        else:
            f = open(filename, 'rt')
        yield f
        f.close()

def gen_concatenate(iterators):
    '''
    Chain a sequence of iterators together into a single sequence.
    '''
    for it in iterators:
        yield from it

def gen_grep(pattern, lines):
    '''
    Look for a regex pattern in a sequence of lines
    '''
    pat = re.compile(pattern)
    for line in lines:
        if pat.search(line):
            yield line
```

python

```
lognames = gen_find('access-log*', 'www')
files = gen_opener(lognames)
lines = gen_concatenate(files)
pylines = gen_grep('(?i)python', lines)
for line in pylines:
    print(line)
```

```
lognames = gen_find('access-log*', 'www')
files = gen_opener(lognames)
lines = gen_concatenate(files)
pylines = gen_grep('(?i)python', lines)
bytecolumn = (line.rsplit(None,1)[1] for line in pylines)
bytes = (int(x) for x in bytecolumn if x != '-')
print('Total', sum(bytes))
```

6.13.3

yield for
yield
sum()

gen_concatenate()
itertools.chain()

lines = itertools.chain(*files) gen_opener()
gen_opener()
china()

gen_concatenate() yield from yield
yield from it it
4.14

David Beazley Generator Tricks for Systems Programmers

6.14 4.14

6.14.1

6.14.2

yield from

```
from collections import Iterable

def flatten(items, ignore_types=(str, bytes)):
    for x in items:
        if isinstance(x, Iterable) and not isinstance(x, ignore_types):
            yield from flatten(x)
        else:
            yield x

items = [1, 2, [3, 4, [5, 6], 7], 8]
# Produces 1 2 3 4 5 6 7 8
for x in flatten(items):
    print(x)
```

isinstance(x, Iterable)
yield from

ignore_types isinstance(x, ignore_types)

```
>>> items = ['Dave', 'Paula', ['Thomas', 'Lewis']]
>>> for x in flatten(items):
...     print(x)
...
Dave
Paula
Thomas
Lewis
>>>
```

6.14.3

yield from

for

```
def flatten(items, ignore_types=(str, bytes)):
    for x in items:
        if isinstance(x, Iterable) and not isinstance(x, ignore_types):
            for i in flatten(x):
                yield i
        else:
            yield x
```

yield from

ignore_types

yield from
12.12

6.15 4.15

6.15.1

6.15.2

heapq.merge()

```
>>> import heapq
>>> a = [1, 4, 7, 10]
>>> b = [2, 5, 6, 11]
>>> for c in heapq.merge(a, b):
...     print(c)
...
1
2
4
5
6
7
10
11
```

6.15.3

heapq.merge

```
with open('sorted_file_1', 'rt') as file1, \
    open('sorted_file_2', 'rt') as file2, \
    open('merged_file', 'wt') as outf:

    for line in heapq.merge(file1, file2):
        outf.write(line)
```

heapq.merge()

6.16 4.16 while

6.16.1

while

6.16.2

IO

```
CHUNKSIZE = 8192

def reader(s):
    while True:
        data = s.recv(CHUNKSIZE)
        if data == b'':
            break
        process_data(data)
```

iter()

```
def reader2(s):
    for chunk in iter(lambda: s.recv(CHUNKSIZE), b''):
        pass
        # process_data(data)
```

```
>>> import sys
>>> f = open('/etc/passwd')
>>> for chunk in iter(lambda: f.read(10), ''):
...     n = sys.stdout.write(chunk)
...
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
...
>>>
```

6.16.3

iter callable ( )

callable

I/O

read() recv()

iter()

lambda callable recv read()

size

CHAPTER

SEVEN

IO

Contents:

7.1 5.1

7.1.1

ASCII UTF-8 UTF-16

7.1.2

rt open()

```
# Read the entire file as a single string
with open('somefile.txt', 'rt') as f:
    data = f.read()

# Iterate over the lines of the file
with open('somefile.txt', 'rt') as f:
    for line in f:
        # process line
        ...
```

wt open()

```
# Write chunks of text data
with open('somefile.txt', 'wt') as f:
    f.write(text1)
    f.write(text2)
    ...

# Redirected print statement
```

```
with open('somefile.txt', 'wt') as f:
    print(line1, file=f)
    print(line2, file=f)
    ...
```

at open()

sys.getdefaultencoding()

utf-8

encoding open()

```
with open('somefile.txt', 'rt', encoding='latin-1') as f:
    ...
```

Python ascii, latin-1, utf-8 utf-16

web UTF-8 ascii U+0000 U+007F

7 latin-1 0-255 U+0000 U+00FF Unicode

latin-1

latin-1

7.1.3

with with

with

```
f = open('somefile.txt', 'rt')
data = f.read()
f.close()
```

Unix Windows (

n rn) Python

Python \n

\n

open() newline=''

```
# Read with disabled newline translation
with open('somefile.txt', 'rt', newline='') as f:
    ...
```

Unix Windows

hello world!\r\n

```
>>> # Newline translation enabled (the default)
>>> f = open('hello.txt', 'rt')
>>> f.read()
'hello world!\n'
```

```
>>> # Newline translation disabled
>>> g = open('hello.txt', 'rt', newline='')
>>> g.read()
'hello world!\r\n'
>>>
```

```
>>> f = open('sample.txt', 'rt', encoding='ascii')
>>> f.read()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "/usr/local/lib/python3.3/encodings/ascii.py", line 26, in decode
        return codecs.ascii_decode(input, self.errors)[0]
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position
12: ordinal not in range(128)
>>>
```

(UTF-8 Latin-1)
open() errors

```
>>> # Replace bad chars with Unicode U+fffd replacement char
>>> f = open('sample.txt', 'rt', encoding='ascii', errors='replace')
>>> f.read()
'Spicy Jalape?o!'
>>> # Ignore bad chars entirely
>>> g = open('sample.txt', 'rt', encoding='ascii', errors='ignore')
>>> g.read()
'Spicy Jalapeo!'
>>>
```

errors

(UTF-8)

7.2 5.2

7.2.1

print()

7.2.2

print() file

```
with open('d:/work/test.txt', 'wt') as f:
    print('Hello World!', file=f)
```

7.2.3

7.3 5.3

7.3.1

print()

7.3.2

print() sep end

```
>>> print('ACME', 50, 91.5)
ACME 50 91.5
>>> print('ACME', 50, 91.5, sep=',')
ACME,50,91.5
>>> print('ACME', 50, 91.5, sep=',', end='!!\n')
ACME,50,91.5!!
>>>
```

end

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>> for i in range(5):
...     print(i, end=' ')
...
0 1 2 3 4 >>>
```

7.3.3

print() sep
str.join()

```
>>> print(','.join(('ACME','50','91.5')))
ACME,50,91.5
>>>
```

str.join()

```
>>> row = ('ACME', 50, 91.5)
>>> print(','.join(row))
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: sequence item 1: expected str instance, int found
>>> print(','.join(str(x) for x in row))
ACME,50,91.5
>>>
```

```
>>> print(*row, sep=',')
ACME,50,91.5
>>>
```

7.4 5.4

7.4.1

7.4.2

rb wb open()

```
# Read the entire file as a single byte string
with open('somefile.bin', 'rb') as f:
    data = f.read()

# Write binary data to a file
with open('somefile.bin', 'wb') as f:
    f.write(b'Hello World')
```

( )

7.4.3

```
>>> # Text string
>>> t = 'Hello World'
>>> t[0]
'H'
>>> for c in t:
...     print(c)
...
H
e
l
l
o
...
>>> # Byte string
>>> b = b'Hello World'
>>> b[0]
72
>>> for c in b:
...     print(c)
...
72
101
108
108
111
...
>>>
```

```
with open('somefile.bin', 'rb') as f:
    data = f.read(16)
    text = data.decode('utf-8')

with open('somefile.bin', 'wb') as f:
    text = 'Hello World'
    f.write(text.encode('utf-8'))
```

I/O C

```
import array
nums = array.array('i', [1, 2, 3, 4])
with open('data.bin','wb') as f:
    f.write(nums)
```

” ”

readinto()

```
>>> import array
>>> a = array.array('i', [0, 0, 0, 0, 0, 0, 0, 0])
>>> with open('data.bin', 'rb') as f:
...     f.readinto(a)
...
16
>>> a
array('i', [1, 2, 3, 4, 0, 0, 0, 0])
>>>
```

( ) 5.9

7.5 5.5

7.5.1

7.5.2

open() x w

```
>>> with open('somefile', 'wt') as f:
...     f.write('Hello\n')
...
>>> with open('somefile', 'xt') as f:
...     f.write('Hello\n')
...
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
FileExistsError: [Errno 17] File exists: 'somefile'
>>>
```

xb xt

7.5.3

(
)

```
>>> import os
>>> if not os.path.exists('somefile'):
...     with open('somefile', 'wt') as f:
...         f.write('Hello\n')
... else:
...     print('File already exists!')
...
File already exists!
>>>
```

x x Python3
open() Python Python C

7.6 5.6 I/O

7.6.1

7.6.2

io.StringIO() io.BytesIO()

```
>>> s = io.StringIO()
>>> s.write('Hello World\n')
12
>>> print('This is a test', file=s)
15
>>> # Get all of the data written so far
>>> s.getvalue()
'Hello World\nThis is a test\n'
>>>

>>> # Wrap a file interface around an existing string
>>> s = io.StringIO('Hello\nWorld\n')
>>> s.read(4)
'Hell'
>>> s.read()
'o\nWorld\n'
>>>
```

io.StringIO io.BytesIO

```
>>> s = io.BytesIO()
>>> s.write(b'binary data')
>>> s.getvalue()
b'binary data'
>>>
```

7.6.3

StringIO BytesIO
StringIO

StringIO BytesIO

7.7 5.7

7.7.1

gzip bz2

7.7.2

gzip bz2 open()

```
# gzip compression
import gzip
with gzip.open('somefile.gz', 'rt') as f:
    text = f.read()

# bz2 compression
import bz2
with bz2.open('somefile.bz2', 'rt') as f:
    text = f.read()
```

```
# gzip compression
import gzip
with gzip.open('somefile.gz', 'wt') as f:
    f.write(text)
```

```
# bz2 compression
import bz2
with bz2.open('somefile.bz2', 'wt') as f:
    f.write(text)
```

I/O Unicode /
rb wb

7.7.3

gzip.open() bz2.open()
open() encoding errors newline

compresslevel

```
with gzip.open('somefile.gz', 'wt', compresslevel=5) as f:
    f.write(text)
```

9

gzip.open() bz2.open()

```
import gzip
f = open('somefile.gz', 'rb')
with gzip.open(f, 'rt') as g:
    text = g.read()
```

gzip bz2

7.8 5.8

7.8.1

7.8.2

iter functools.partial()

```
from functools import partial

RECORD_SIZE = 32

with open('somefile.data', 'rb') as f:
    records = iter(partial(f.read, RECORD_SIZE), b'')
    for r in records:
        ...
```

records

7.8.3

iter()

functools.partial
b''

(
)

7.9 5.9

7.9.1

7.9.2

readinto()

```
import os.path

def read_into_buffer(filename):
    buf = bytearray(os.path.getsize(filename))
    with open(filename, 'rb') as f:
        f.readinto(buf)
    return buf
```

```
>>> # Write a sample file
>>> with open('sample.bin', 'wb') as f:
...     f.write(b'Hello World')
...
>>> buf = read_into_buffer('sample.bin')
>>> buf
bytearray(b'Hello World')
>>> buf[0:5] = b'Hallo'
>>> buf
bytearray(b'Hallo World')
>>> with open('newsample.bin', 'wb') as f:
...     f.write(buf)
...
11
>>>
```

7.9.3

readinto()
array numpy read() readinto()

```
record_size = 32 # Size of each record (adjust value)

buf = bytearray(record_size)
with open('somefile', 'rb') as f:
    while True:
        n = f.readinto(buf)
        if n < record_size:
            break
        # Use the contents of buf
        ...
```

memoryview

```
>>> buf
bytearray(b'Hello World')
>>> m1 = memoryview(buf)
>>> m2 = m1[-5:]
>>> m2
<memory at 0x100681390>
>>> m2[:] = b'WORLD'
>>> buf
bytearray(b'Hello WORLD')
>>>
```

f.readinto()

(
)

into (recv_into()
pack_into()) Python I/O

memoryviews 6.12

7.10 5.10

7.10.1

7.10.2

mmap

```
import os
import mmap

def memory_map(filename, access=mmap.ACCESS_WRITE):
    size = os.path.getsize(filename)
    fd = os.open(filename, os.O_RDWR)
    return mmap.mmap(fd, size, access=access)
```

```
>>> size = 1000000
>>> with open('data', 'wb') as f:
...     f.seek(size-1)
...     f.write(b'\x00')
...
>>>
```

memory_map()

```
>>> m = memory_map('data')
>>> len(m)
1000000
>>> m[0:10]
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
>>> m[0]
```

```
0
>>> # Reassign a slice
>>> m[0:11] = b'Hello World'
>>> m.close()

>>> # Verify that changes were made
>>> with open('data', 'rb') as f:
... print(f.read(11))
...
b'Hello World'
>>>
```

mmap() mmap

```
>>> with memory_map('data') as m:
...     print(len(m))
...     print(m[0:10])
...
1000000
b'Hello World'
>>> m.closed
True
>>>
```

memeory_map()

access

mmap.ACCESS_READ

```
m = memory_map(filename, mmap.ACCESS_READ)
```

mmap.ACCESS_COPY

```
m = memory_map(filename, mmap.ACCESS_COPY)
```

7.10.3

mmap

seek() read() write()

mmap()

```
>>> m = memory_map('data')
>>> # Memoryview of unsigned integers
>>> v = memoryview(m).cast('I')
>>> v[0] = 7
>>> m[0:4]
```

```
b'\x07\x00\x00\x00'
>>> m[0:4] = b'\x07\x01\x00\x00'
>>> v[0]
263
>>>
```

Python mmap

Unix Windows

mmap()

Python

7.11 5.11

7.11.1

7.11.2

os.path

```
>>> import os
>>> path = '/Users/beazley/Data/data.csv'

>>> # Get the last component of the path
>>> os.path.basename(path)
'data.csv'

>>> # Get the directory name
>>> os.path.dirname(path)
'/Users/beazley/Data'

>>> # Join path components together
>>> os.path.join('tmp', 'data', os.path.basename(path))
'tmp/data/data.csv'
```

```
>>> # Expand the user's home directory
>>> path = '~/Data/data.csv'
>>> os.path.expanduser(path)
'/Users/beazley/Data/data.csv'

>>> # Split the file extension
>>> os.path.splitext(path)
('~/Data/data', '.csv')
>>>
```

7.11.3

os.path

os.path

Unix Windows Data/data.csv Data\data.csv

os.path

7.12 5.12

7.12.1

7.12.2

os.path

```
>>> import os
>>> os.path.exists('/etc/passwd')
True
>>> os.path.exists('/tmp/spam')
False
>>>
```

False

```
>>> # Is a regular file
>>> os.path.isfile('/etc/passwd')
True
```

```
>>> # Is a directory
>>> os.path.isdir('/etc/passwd')
False

>>> # Is a symbolic link
>>> os.path.islink('/usr/local/bin/python3')
True

>>> # Get the file linked to
>>> os.path.realpath('/usr/local/bin/python3')
'/usr/local/bin/python3.3'
>>>
```

( ) os.path

```
>>> os.path.getsize('/etc/passwd')
3669
>>> os.path.getmtime('/etc/passwd')
1272478234.0
>>> import time
>>> time.ctime(os.path.getmtime('/etc/passwd'))
'Wed Apr 28 13:10:34 2010'
>>>
```

7.12.3

os.path

```
>>> os.path.getsize('/Users/guido/Desktop/foo.txt')
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "/usr/local/lib/python3.3/genericpath.py", line 49, in getsize
        return os.stat(filename).st_size
PermissionError: [Errno 13] Permission denied: '/Users/guido/Desktop/foo.txt'
>>>
```

7.13 5.13

7.13.1

7.13.2

os.listdir()

```
import os
names = os.listdir('somedir')
```

os.path

```
import os.path

# Get all regular files
names = [name for name in os.listdir('somedir')
        if os.path.isfile(os.path.join('somedir', name))]

# Get all dirs
dirnames = [name for name in os.listdir('somedir')
        if os.path.isdir(os.path.join('somedir', name))]
```

startswith() endswith()

```
pyfiles = [name for name in os.listdir('somedir')
            if name.endswith('.py')]
```

glob fnmatch

```
import glob
pyfiles = glob.glob('somedir/*.py')

from fnmatch import fnmatch
pyfiles = [name for name in os.listdir('somedir')
            if fnmatch(name, '*.py')]
```

7.13.3

os.path os.stat()

```
# Example of getting a directory listing

import os
import os.path
import glob

pyfiles = glob.glob('*.py')

# Get file sizes and modification dates
```

```
name_sz_date = [(name, os.path.getsize(name), os.path.getmtime(name))
                for name in pyfiles]
for name, size, mtime in name_sz_date:
    print(name, size, mtime)

# Alternative: Get file metadata
file_metadata = [(name, os.stat(name)) for name in pyfiles]
for name, meta in file_metadata:
    print(name, meta.st_size, meta.st_mtime)
```

os.listdir()

5.14 5.15

7.14 5.14

7.14.1

I/O

7.14.2

sys.getfilesystemencoding()

```
>>> sys.getfilesystemencoding()
'utf-8'
>>>
```

```
>>> # Wrte a file using a unicode filename
>>> with open('jalape\xf1o.txt', 'w') as f:
...     f.write('Spicy!')
...
6
>>> # Directory listing (decoded)
>>> import os
>>> os.listdir('.')
['jalapeño.txt']

>>> # Directory listing (raw)
>>> os.listdir(b'.') # Note: byte string
[b'jalapen\xcc\x83o.txt']
```

```
>>> # Open file with raw filename
>>> with open(b'jalapen\xcc\x830.txt') as f:
...     print(f.read())
...
Spicy!
>>>
```

open() os.listdir()

7.14.3

Python

5.15

7.15 5.15

7.15.1

UnicodeEncodeError —— surrogates not allowed

7.15.2

```
def bad_filename(filename):
    return repr(filename)[1:-1]

try:
    print(filename)
except UnicodeEncodeError:
    print(bad_filename(filename))
```

7.15.3

Python sys.getfilesystemencoding()

(open()
)

os.listdir() Python

Python
\xhh Unicode \udchh ”
” bäd.txt(
Latin-1 UTF-8)

```
>>> import os
>>> files = os.listdir('.')
>>> files
['spam.py', 'b\udce4d.txt', 'foo.txt']
>>>
```

open()
(
)

```
>>> for name in files:
...     print(name)
...
spam.py
Traceback (most recent call last):
    File "<stdin>", line 2, in <module>
UnicodeEncodeError: 'utf-8' codec can't encode character '\udce4' in
position 1: surrogates not allowed
>>>
```

\udce4 Unicode

Unicode

```
>>> for name in files:
... try:
...     print(name)
... except UnicodeEncodeError:
...     print(bad_filename(name))
...
spam.py
b\udce4d.txt
foo.txt
>>>
```

bad_filename()

```
def bad_filename(filename):
    temp = filename.encode(sys.getfilesystemencoding(), errors='surrogateescape')
    return temp.decode('latin-1')
```

:

```
surrogateescape:
      Python              OS    API

                                        Unicode

          OS API
```

```
>>> for name in files:
...     try:
...         print(name)
...     except UnicodeEncodeError:
...         print(bad_filename(name))
...
spam.py
bäd.txt
foo.txt
>>>
```

7.16 5.16

7.16.1

Unicode

7.16.2

Unicode /

io.TextIOWrapper()

```
import urllib.request
import io

u = urllib.request.urlopen('http://www.python.org')
```

```
f = io.TextIOWrapper(u, encoding='utf-8')
text = f.read()
```

detach()

sys.stdout

```
>>> import sys
>>> sys.stdout.encoding
'UTF-8'
>>> sys.stdout = io.TextIOWrapper(sys.stdout.detach(), encoding='latin-1')
>>> sys.stdout.encoding
'latin-1'
>>>
```

7.16.3

I/O

```
>>> f = open('sample.txt','w')
>>> f
<_io.TextIOWrapper name='sample.txt' mode='w' encoding='UTF-8'>
>>> f.buffer
<_io.BufferedWriter name='sample.txt'>
>>> f.buffer.raw
<_io.FileIO name='sample.txt' mode='wb'>
>>>
```

io.TextIOWrapper Unicode
io.BufferedWriter I/O io.FileIO

io.TextIOWrapper

```
>>> f
<_io.TextIOWrapper name='sample.txt' mode='w' encoding='UTF-8'>
>>> f = io.TextIOWrapper(f.buffer, encoding='latin-1')
>>> f
<_io.TextIOWrapper name='sample.txt' encoding='latin-1'>
>>> f.write('Hello')
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ValueError: I/O operation on closed file.
>>>
```

f

detach()

```
>>> f = open('sample.txt', 'w')
>>> f
<_io.TextIOWrapper name='sample.txt' mode='w' encoding='UTF-8'>
>>> b = f.detach()
>>> b
<_io.BufferedWriter name='sample.txt'>
>>> f.write('hello')
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ValueError: underlying buffer has been detached
>>>
```

```
>>> f = io.TextIOWrapper(b, encoding='latin-1')
>>> f
<_io.TextIOWrapper name='sample.txt' encoding='latin-1'>
>>>
```

```
>>> sys.stdout = io.TextIOWrapper(sys.stdout.detach(), encoding='ascii',
...                               errors='xmlcharrefreplace')
>>> print('Jalape\u00f1o')
Jalape&#241;o
>>>
```

ASCII ñ ñ

7.17 5.17

7.17.1

7.17.2

```
>>> import sys
>>> sys.stdout.write(b'Hello\n')
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: must be str, not bytes
>>> sys.stdout.buffer.write(b'Hello\n')
```

```
Hello
5
>>>
```

buffer

7.17.3

I/O

Unicode / buffer

/

sys.stdout sys.stdout

7.18 5.18

7.18.1

I/O ( )

Python

7.18.2

I/O

open() Python

```
# Open a low-level file descriptor
import os
fd = os.open('somefile.txt', os.O_WRONLY | os.O_CREAT)

# Turn into a proper file
f = open(fd, 'wt')
f.write('hello world\n')
f.close()
```

open() colsefd=False

```
# Create a file object, but don't close underlying fd when done
f = open(fd, 'wt', closefd=False)
...
```

7.18.3

Unix

I/O

```
from socket import socket, AF_INET, SOCK_STREAM

def echo_client(client_sock, addr):
    print('Got connection from', addr)

    # Make text-mode file wrappers for socket reading/writing
    client_in = open(client_sock.fileno(), 'rt', encoding='latin-1',
                closefd=False)

    client_out = open(client_sock.fileno(), 'wt', encoding='latin-1',
                closefd=False)

    # Echo lines back to the client using file I/O
    for line in client_in:
        client_out.write(line)
        client_out.flush()

    client_sock.close()

def echo_server(address):
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(address)
    sock.listen(1)
    while True:
        client, addr = sock.accept()
        echo_client(client, addr)
```

open()

Unix

makefile()

makefile()

(

)

```
import sys
# Create a binary-mode file for stdout
bstdout = open(sys.stdout.fileno(), 'wb', closefd=False)
bstdout.write(b'Hello World\n')
bstdout.flush()
```

(

)

Unix

7.19 5.19

7.19.1

7.19.2

tempfile
tempfile.TemporaryFile

```
from tempfile import TemporaryFile

with TemporaryFile('w+t') as f:
    # Read/write to the file
    f.write('Hello World\n')
    f.write('Testing\n')

    # Seek back to beginning and read the data
    f.seek(0)
    data = f.read()

# Temporary file is destroyed
```

```
f = TemporaryFile('w+t')
# Use the temporary file
...
f.close()
# File is destroyed
```

TemporaryFile() w+t
w+b
TemporaryFile()
open()

```
with TemporaryFile('w+t', encoding='utf-8', errors='ignore') as f:
    ...
```

Unix TemporaryFile()
NamedTemporaryFile()

```
from tempfile import NamedTemporaryFile
```

```
with NamedTemporaryFile('w+t') as f:
    print('filename is:', f.name)
    ...

# File automatically destroyed
```

f.name

TemporaryFile()

delte=False

```
with NamedTemporaryFile('w+t', delete=False) as f:
    print('filename is:', f.name)
    ...
```

tempfile.TemporaryDirectory()

```
from tempfile import TemporaryDirectory

with TemporaryDirectory() as dirname:
    print('dirname is:', dirname)
    # Use the directory
    ...
# Directory and all contents destroyed
```

7.19.3

TemporaryFile() NamedTemporaryFile() TemporaryDirectory()

mkstemp() mkdtemp()

```
>>> import tempfile
>>> tempfile.mkstemp()
(3, '/var/folders/7W/7WZl5sfZEF0pljrEB1UMWE+++TI/-Tmp-/tmp7fefhv')
>>> tempfile.mkdtemp()
'/var/folders/7W/7WZl5sfZEF0pljrEB1UMWE+++TI/-Tmp-/tmp5wvcv6'
>>>
```

mkstemp()

OS

/var/tmp

tempfile.gettempdir()

```
>>> tempfile.gettempdir()
'/var/folders/7W/7WZl5sfZEF0pljrEB1UMWE+++TI/-Tmp-'
>>>
```

prefix suffix dir

```
>>> f = NamedTemporaryFile(prefix='mytemp', suffix='.txt', dir='/tmp')
>>> f.name
'/tmp/mytemp8ee899.txt'
>>>
```

tempfile

7.20 5.20

7.20.1

(
)

7.20.2

Python I/O
pySerial pySerial

```
import serial
ser = serial.Serial('/dev/tty.usbmodem641', # Device name varies
                    baudrate=9600,
                    bytesize=8,
                    parity='N',
                    stopbits=1)
```

Windows
0, 1 ”COM0” ”COM1”
read() readline() write()

```
ser.write(b'G1 X50 Y50\r\n')
resp = ser.readline()
```

7.20.3

pySerial (
) RTS-CTS

Serial() rtscts=True

I/O
(/)
struct

7.21 5.21 Python

7.21.1

Python

7.21.2

pickle

```
import pickle

data = ... # Some Python object
f = open('somefile', 'wb')
pickle.dump(data, f)
```

pickle.dumps()

```
s = pickle.dumps(data)
```

picle.load() pickle.loads()

```
# Restore from a file
f = open('somefile', 'rb')
data = pickle.load(f)

# Restore from a string
data = pickle.loads(s)
```

7.21.3

dump() load() pickle
Python
/ Python
pickle

pickle Python

```
>>> import pickle
>>> f = open('somedata', 'wb')
>>> pickle.dump([1, 2, 3, 4], f)
>>> pickle.dump('hello', f)
>>> pickle.dump({'Apple', 'Pear', 'Banana'}, f)
>>> f.close()
>>> f = open('somedata', 'rb')
>>> pickle.load(f)
[1, 2, 3, 4]
>>> pickle.load(f)
'hello'
>>> pickle.load(f)
{'Apple', 'Pear', 'Banana'}
>>>
```

```
>>> import math
>>> import pickle.
>>> pickle.dumps(math.cos)
b'\x80\x03cmath\ncos\nq\x00.'
>>>
```

Python

```
                    pickle.load()
pickle
                pickle
                        Python
            pickle
```

__getstate__() __setstate__()
pickle.dump() __getstate__() __setstate__()

```
# countdown.py
import time
import threading

class Countdown:
    def __init__(self, n):
```

```
        self.n = n
        self.thr = threading.Thread(target=self.run)
        self.thr.daemon = True
        self.thr.start()

    def run(self):
        while self.n > 0:
            print('T-minus', self.n)
            self.n -= 1
            time.sleep(5)

    def __getstate__(self):
        return self.n

    def __setstate__(self, n):
        self.__init__(n)
```

```
>>> import countdown
>>> c = countdown.Countdown(30)
>>> T-minus 30
T-minus 29
T-minus 28
...

>>> # After a few moments
>>> f = open('cstate.p', 'wb')
>>> import pickle
>>> pickle.dump(c, f)
>>> f.close()
```

Python

```
>>> f = open('cstate.p', 'rb')
>>> pickle.load(f)
countdown.Countdown object at 0x10069e2d0>
T-minus 19
T-minus 18
...
```

pickle array numpy

HDF5 (

)

pickle Python

XML CSV JSON

pickle

pickle

CHAPTER

EIGHT

Python CSV JSON XML

Contents:

8.1 6.1 CSV

8.1.1

CSV

8.1.2

CSV csv stocks.csv

```
Symbol,Price,Date,Time,Change,Volume
"AA",39.48,"6/11/2007","9:36am",-0.18,181800
"AIG",71.38,"6/11/2007","9:36am",-0.15,195500
"AXP",62.58,"6/11/2007","9:36am",-0.46,935000
"BA",98.31,"6/11/2007","9:36am",+0.12,104800
"C",53.08,"6/11/2007","9:36am",-0.25,360900
"CAT",78.29,"6/11/2007","9:36am",-0.23,225400
```

```
import csv
with open('stocks.csv') as f:
    f_csv = csv.reader(f)
    headers = next(f_csv)
    for row in f_csv:
        # Process row
        ...
```

row
row[0] Symbol row[4] Change

```
from collections import namedtuple
with open('stock.csv') as f:
    f_csv = csv.reader(f)
    headings = next(f_csv)
    Row = namedtuple('Row', headings)
    for r in f_csv:
        row = Row(*r)
        # Process row
        ...
```

row.Symbol row.Change
Python
( )

```
import csv
with open('stocks.csv') as f:
    f_csv = csv.DictReader(f)
    for row in f_csv:
        # process row
        ...
```

row['Symbol']
row['Change']

CSV csv writer
:

```
headers = ['Symbol','Price','Date','Time','Change','Volume']
rows = [('AA', 39.48, '6/11/2007', '9:36am', -0.18, 181800),
        ('AIG', 71.38, '6/11/2007', '9:36am', -0.15, 195500),
        ('AXP', 62.58, '6/11/2007', '9:36am', -0.46, 935000),
       ]

with open('stocks.csv','w') as f:
    f_csv = csv.writer(f)
    f_csv.writerow(headers)
    f_csv.writerows(rows)
```

```
headers = ['Symbol', 'Price', 'Date', 'Time', 'Change', 'Volume']
rows = [{'Symbol':'AA', 'Price':39.48, 'Date':'6/11/2007',
        'Time':'9:36am', 'Change':-0.18, 'Volume':181800},
        {'Symbol':'AIG', 'Price': 71.38, 'Date':'6/11/2007',
        'Time':'9:36am', 'Change':-0.15, 'Volume': 195500},
        {'Symbol':'AXP', 'Price': 62.58, 'Date':'6/11/2007',
        'Time':'9:36am', 'Change':-0.46, 'Volume': 935000},
```

```
        ]

with open('stocks.csv','w') as f:
    f_csv = csv.DictWriter(f, headers)
    f_csv.writeheader()
    f_csv.writerows(rows)
```

8.1.3

csv CSV

```
with open('stocks.csv') as f:
for line in f:
    row = line.split(',')
    # process row
    ...
```

csv Microsoft Excel CSV

csv

( )

tab

```
# Example of reading tab-separated values
with open('stock.tsv') as f:
    f_tsv = csv.reader(f, delimiter='\t')
    for row in f_tsv:
        # Process row
        ...
```

CSV

CSV

```
Street Address,Num-Premises,Latitude,Longitude 5412 N CLARK,10,41.980262,-87.668452
```

ValueError

```
import re
with open('stock.csv') as f:
    f_csv = csv.reader(f)
    headers = [ re.sub('[^a-zA-Z_]', '_', h) for h in next(f_csv) ]
    Row = namedtuple('Row', headers)
    for r in f_csv:
        row = Row(*r)
```

```
        # Process row
        ...
```

csv

CSV

```
col_types = [str, float, str, str, float, int]
with open('stocks.csv') as f:
    f_csv = csv.reader(f)
    headers = next(f_csv)
    for row in f_csv:
        # Apply conversions to the row items
        row = tuple(convert(value) for convert, value in zip(col_types, row))
        ...
```

```
print('Reading as dicts with type conversion')
field_types = [ ('Price', float),
                ('Change', float),
                ('Volume', int) ]

with open('stocks.csv') as f:
    for row in csv.DictReader(f):
        row.update((key, conversion(row[key]))
                for key, conversion in field_types)
        print(row)
```

CSV

(

)

CSV

Pandas Pandas pandas.read_csv()
CSV DataFrame

6.13

8.2 6.2 JSON

8.2.1

JSON(JavaScript Object Notation)

8.2.2

json JSON
json.dumps() json.loads() pickle
Python JSON

```
import json

data = {
    'name' : 'ACME',
    'shares' : 100,
    'price' : 542.23
}

json_str = json.dumps(data)
```

JSON Python

```
data = json.loads(json_str)
```

json.dump() json.load()
JSON

```
# Writing JSON data
with open('data.json', 'w') as f:
    json.dump(data, f)

# Reading data back
with open('data.json', 'r') as f:
    data = json.load(f)
```

8.2.3

JSON None bool int float str
lists tuples dictionaries dictionaries keys
(key) JSON
Python lists dictionaries web

JSON Python
True true False false None null

```
>>> json.dumps(False)
'false'
>>> d = {'a': True,
...      'b': 'Hello',
...      'c': None}
>>> json.dumps(d)
```

```
'{"b": "Hello", "c": null, "a": true}'
>>>
```

JSON

pprint pprint() print()
key
Twitter

```
>>> from urllib.request import urlopen
>>> import json
>>> u = urlopen('http://search.twitter.com/search.json?q=python&rpp=5')
>>> resp = json.loads(u.read().decode('utf-8'))
>>> from pprint import pprint
>>> pprint(resp)
{'completed_in': 0.074,
'max_id': 264043230692245504,
'max_id_str': '264043230692245504',
'next_page': '?page=2&max_id=264043230692245504&q=python&rpp=5',
'page': 1,
'query': 'python',
'refresh_url': '?since_id=264043230692245504&q=python',
'results': [{'created_at': 'Thu, 01 Nov 2012 16:36:26 +0000',
             'from_user': ...
             },
             {'created_at': 'Thu, 01 Nov 2012 16:36:14 +0000',
             'from_user': ...
             },
             {'created_at': 'Thu, 01 Nov 2012 16:36:13 +0000',
             'from_user': ...
             },
             {'created_at': 'Thu, 01 Nov 2012 16:36:07 +0000',
             'from_user': ...
             }
             {'created_at': 'Thu, 01 Nov 2012 16:36:04 +0000',
             'from_user': ...
             }],
'results_per_page': 5,
'since_id': 0,
'since_id_str': '0'}
>>>
```

JSON dicts lists
json.loads() object_pairs_hook object_hook
JSON OrderedDict

```
>>> s = '{"name": "ACME", "shares": 50, "price": 490.1}'
>>> from collections import OrderedDict
>>> data = json.loads(s, object_pairs_hook=OrderedDict)
>>> data
OrderedDict([('name', 'ACME'), ('shares', 50), ('price', 490.1)])
```

```
>>>
```

JSON Python

```
>>> class JSONObject:
...     def __init__(self, d):
...         self.__dict__ = d
...
>>>
>>> data = json.loads(s, object_hook=JSONObject)
>>> data.name
'ACME'
>>> data.shares
50
>>> data.price
490.1
>>>
```

JSON __init__()

JSON
json.dumps() indent pprint()

```
>>> print(json.dumps(data))
{"price": 542.23, "name": "ACME", "shares": 100}
>>> print(json.dumps(data, indent=4))
{
    "price": 542.23,
    "name": "ACME",
    "shares": 100
}
>>>
```

JSON

```
>>> class Point:
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...
>>> p = Point(2, 3)
>>> json.dumps(p)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "/usr/local/lib/python3.3/json/__init__.py", line 226, in dumps
        return _default_encoder.encode(obj)
    File "/usr/local/lib/python3.3/json/encoder.py", line 187, in encode
        chunks = self.iterencode(o, _one_shot=True)
    File "/usr/local/lib/python3.3/json/encoder.py", line 245, in iterencode
        return _iterencode(o, 0)
    File "/usr/local/lib/python3.3/json/encoder.py", line 169, in default
```

```
        raise TypeError(repr(o) + " is not JSON serializable")
TypeError: <__main__.Point object at 0x1006f2650> is not JSON serializable
>>>
```

```
def serialize_instance(obj):
    d = { '__classname__' : type(obj).__name__ }
    d.update(vars(obj))
    return d
```

```
# Dictionary mapping names to known classes
classes = {
    'Point' : Point
}

def unserialize_object(d):
    clsname = d.pop('__classname__', None)
    if clsname:
        cls = classes[clsname]
        obj = cls.__new__(cls) # Make instance without calling __init__
        for key, value in d.items():
            setattr(obj, key, value)
        return obj
    else:
        return d
```

```
>>> p = Point(2,3)
>>> s = json.dumps(p, default=serialize_instance)
>>> s
'{"__classname__": "Point", "y": 3, "x": 2}'
>>> a = json.loads(s, object_hook=unserialize_object)
>>> a
<__main__.Point object at 0x1017577d0>
>>> a.x
2
>>> a.y
3
>>>
```

json NaN

8.3 6.3 XML

8.3.1

XML

8.3.2

xml.etree.ElementTree XML
Planet Python RSS

```
from urllib.request import urlopen
from xml.etree.ElementTree import parse

# Download the RSS feed and parse it
u = urlopen('http://planet.python.org/rss20.xml')
doc = parse(u)

# Extract and output tags of interest
for item in doc.iterfind('channel/item'):
    title = item.findtext('title')
    date = item.findtext('pubDate')
    link = item.findtext('link')

    print(title)
    print(date)
    print(link)
    print()
```

```
Steve Holden: Python for Data Analysis
Mon, 19 Nov 2012 02:13:51 +0000
http://holdenweb.blogspot.com/2012/11/python-for-data-analysis.html

Vasudev Ram: The Python Data model (for v2 and v3)
Sun, 18 Nov 2012 22:06:47 +0000
http://jugad2.blogspot.com/2012/11/the-python-data-model.html

Python Diary: Been playing around with Object Databases
Sun, 18 Nov 2012 20:40:29 +0000
http://www.pythondiary.com/blog/Nov.18,2012/been-...-object-databases.html

Vasudev Ram: Wakari, Scientific Python in the cloud
Sun, 18 Nov 2012 20:19:41 +0000
http://jugad2.blogspot.com/2012/11/wakari-scientific-python-in-cloud.html

Jesse Jiryu Davis: Toro: synchronization primitives for Tornado coroutines
```

```
Sun, 18 Nov 2012 20:17:49 +0000
http://feedproxy.google.com/~r/EmptysquarePython/~3/_DOZT2Kd0hQ/
```

print()

8.3.3

XML XML
Internet
( ) XML

XML
RSS

```
<?xml version="1.0"?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
    <channel>
        <title>Planet Python</title>
        <link>http://planet.python.org/</link>
        <language>en</language>
        <description>Planet Python - http://planet.python.org/</description>
        <item>
            <title>Steve Holden: Python for Data Analysis</title>
            <guid>http://holdenweb.blogspot.com/...-data-analysis.html</guid>
            <link>http://holdenweb.blogspot.com/...-data-analysis.html</link>
            <description>...</description>
            <pubDate>Mon, 19 Nov 2012 02:13:51 +0000</pubDate>
        </item>
        <item>
            <title>Vasudev Ram: The Python Data model (for v2 and v3)</title>
            <guid>http://jugad2.blogspot.com/...-data-model.html</guid>
            <link>http://jugad2.blogspot.com/...-data-model.html</link>
            <description>...</description>
            <pubDate>Sun, 18 Nov 2012 22:06:47 +0000</pubDate>
        </item>
        <item>
            <title>Python Diary: Been playing around with Object Databases</title>
            <guid>http://www.pythondiary.com/...-object-databases.html</guid>
            <link>http://www.pythondiary.com/...-object-databases.html</link>
            <description>...</description>
            <pubDate>Sun, 18 Nov 2012 20:40:29 +0000</pubDate>
        </item>
        ...
    </channel>
</rss>
```

xml.etree.ElementTree.parse() XML
find() iterfind() findtext()

XML channel/item title

doc.iterfind('channel/item') channel item doc
(rss) item.findtext()
item

ElementTree
tag text get()

```
>>> doc
<xml.etree.ElementTree.ElementTree object at 0x101339510>
>>> e = doc.find('channel/title')
>>> e
<Element 'title' at 0x10135b310>
>>> e.tag
'title'
>>> e.text
'Planet Python'
>>> e.get('some_attribute')
>>>
```

xml.etree.ElementTree XML
lxml ElementTree
lxml import from
lxml.etree import parse lxml XML
XSLT XPath

8.4 6.4 XML

8.4.1

XML

8.4.2

XML

```
from xml.etree.ElementTree import iterparse

def parse_and_remove(filename, path):
    path_parts = path.split('/')
    doc = iterparse(filename, ('start', 'end'))
    # Skip the root element
```

```
    next(doc)

    tag_stack = []
    elem_stack = []
    for event, elem in doc:
        if event == 'start':
            tag_stack.append(elem.tag)
            elem_stack.append(elem)
        elif event == 'end':
            if tag_stack == path_parts:
                yield elem
                elem_stack[-2].remove(elem)
            try:
                tag_stack.pop()
                elem_stack.pop()
            except IndexError:
                pass
```

XML

XML

100,000

```
<response>
    <row>
        <row ...>
            <creation_date>2012-11-18T00:00:00</creation_date>
            <status>Completed</status>
            <completion_date>2012-11-18T00:00:00</completion_date>
            <service_request_number>12-01906549</service_request_number>
            <type_of_service_request>Pot Hole in Street</type_of_service_request>
            <current_activity>Final Outcome</current_activity>
            <most_recent_action>CDOT Street Cut ... Outcome</most_recent_action>
            <street_address>4714 S TALMAN AVE</street_address>
            <zip>60632</zip>
            <x_coordinate>1159494.68618856</x_coordinate>
            <y_coordinate>1873313.83503384</y_coordinate>
            <ward>14</ward>
            <police_district>9</police_district>
            <community_area>58</community_area>
            <latitude>41.808090232127896</latitude>
            <longitude>-87.69053684711305</longitude>
            <location latitude="41.808090232127896"
            longitude="-87.69053684711305" />
        </row>
        <row ...>
            <creation_date>2012-11-18T00:00:00</creation_date>
            <status>Completed</status>
            <completion_date>2012-11-18T00:00:00</completion_date>
            <service_request_number>12-01906695</service_request_number>
            <type_of_service_request>Pot Hole in Street</type_of_service_request>
```

```
            <current_activity>Final Outcome</current_activity>
            <most_recent_action>CDOT Street Cut ... Outcome</most_recent_action>
            <street_address>3510 W NORTH AVE</street_address>
            <zip>60647</zip>
            <x_coordinate>1152732.14127696</x_coordinate>
            <y_coordinate>1910409.38979075</y_coordinate>
            <ward>26</ward>
            <police_district>14</police_district>
            <community_area>23</community_area>
            <latitude>41.91002084292946</latitude>
            <longitude>-87.71435952353961</longitude>
            <location latitude="41.91002084292946"
            longitude="-87.71435952353961" />
        </row>
    </row>
</response>
```

```
from xml.etree.ElementTree import parse
from collections import Counter

potholes_by_zip = Counter()

doc = parse('potholes.xml')
for pothole in doc.iterfind('row/row'):
    potholes_by_zip[pothole.findtext('zip')] += 1
for zipcode, num in potholes_by_zip.most_common():
    print(zipcode, num)
```

XML
450MB

```
from collections import Counter

potholes_by_zip = Counter()

data = parse_and_remove('potholes.xml', 'row/row')
for pothole in data:
    potholes_by_zip[pothole.findtext('zip')] += 1
for zipcode, num in potholes_by_zip.most_common():
    print(zipcode, num)
```

7MB –

8.4.3

ElementTree iterparse()
XML

start , end, start-ns end-ns iterparse()
(event, elem) event
elem XML

```
>>> data = iterparse('potholes.xml',('start','end'))
>>> next(data)
('start', <Element 'response' at 0x100771d60>)
>>> next(data)
('start', <Element 'row' at 0x100771e68>)
>>> next(data)
('start', <Element 'row' at 0x100771fc8>)
>>> next(data)
('start', <Element 'creation_date' at 0x100771f18>)
>>> next(data)
('end', <Element 'creation_date' at 0x100771f18>)
>>> next(data)
('start', <Element 'status' at 0x1006a7f18>)
>>> next(data)
('end', <Element 'status' at 0x1006a7f18>)
>>>
```

start ( )
end start-ns
end-ns XML

start end

parse_and_remove()

yield

yield ElementTree

```
elem_stack[-2].remove(elem)
```

yield

XML

8.5 6.5 XML

8.5.1 L

8.5.2

xml.etree.ElementTree XML

```
from xml.etree.ElementTree import Element

def dict_to_xml(tag, d):
'''
Turn a simple dict of key/value pairs into XML
'''
elem = Element(tag)
for key, val in d.items():
    child = Element(key)
    child.text = str(val)
    elem.append(child)
return elem
```

```
>>> s = { 'name': 'GOOG', 'shares': 100, 'price':490.1 }
>>> e = dict_to_xml('stock', s)
>>> e
<Element 'stock' at 0x1004b64c8>
>>>
```

Element I/O xml.etree.ElementTree
tostring()

```
>>> from xml.etree.ElementTree import tostring
>>> tostring(e)
b'<stock><price>490.1</price><shares>100</shares><name>GOOG</name></stock>'
>>>
```

set()

```
>>> e.set('_id','1234')
>>> tostring(e)
b'<stock _id="1234"><price>490.1</price><shares>100</shares><name>GOOG</name>
</stock>'
>>>
```

OrderedDict
1.7

8.5.3

XML

```
def dict_to_xml_str(tag, d):
    '''
    Turn a simple dict of key/value pairs into XML
```

```
    '''
    parts = ['<{}>'.format(tag)]
    for key, val in d.items():
        parts.append('<{0}>{1}</{0}>'.format(key,val))
    parts.append('</{}>'.format(tag))
    return ''.join(parts)
```

```
>>> d = { 'name' : '<spam>' }

>>> # String creation
>>> dict_to_xml_str('item',d)
'<item><name><spam></name></item>'

>>> # Proper XML creation
>>> e = dict_to_xml('item',d)
>>> tostring(e)
b'<item><name>&lt;spam&gt;</name></item>'
>>>
```

‘<’ ‘>’ < >

xml.sax.saxutils

escape() unescape()

```
>>> from xml.sax.saxutils import escape, unescape
>>> escape('<spam>')
'&lt;spam&gt;'
>>> unescape(_)
'<spam>'
>>>
```

Element

Element

XML

8.6 6.6 XML

8.6.1

XML XML

8.6.2

xml.etree.ElementTree

pred.xml

```
<?xml version="1.0"?>
<stop>
    <id>14791</id>
    <nm>Clark &amp;amp; Balmoral</nm>
    <sri>
        <rt>22</rt>
        <d>North Bound</d>
        <dd>North Bound</dd>
    </sri>
    <cr>22</cr>
    <pre>
        <pt>5 MIN</pt>
        <fd>Howard</fd>
        <v>1378</v>
        <rn>22</rn>
    </pre>
    <pre>
        <pt>15 MIN</pt>
        <fd>Howard</fd>
        <v>1867</v>
        <rn>22</rn>
    </pre>
</stop>
```

ElementTree

```
>>> from xml.etree.ElementTree import parse, Element
>>> doc = parse('pred.xml')
>>> root = doc.getroot()
>>> root
<Element 'stop' at 0x100770cb0>

>>> # Remove a few elements
>>> root.remove(root.find('sri'))
>>> root.remove(root.find('cr'))
>>> # Insert a new element after <nm>...</nm>
>>> root.getchildren().index(root.find('nm'))
1
>>> e = Element('spam')
>>> e.text = 'This is a test'
>>> root.insert(2, e)

>>> # Write back to a file
>>> doc.write('newpred.xml', xml_declaration=True)
>>>
```

XML

```
<?xml version='1.0' encoding='us-ascii'?>
<stop>
    <id>14791</id>
    <nm>Clark &amp; Balmoral</nm>
    <spam>This is a test</spam>
    <pre>
        <pt>5 MIN</pt>
        <fd>Howard</fd>
        <v>1378</v>
        <rn>22</rn>
    </pre>
    <pre>
        <pt>15 MIN</pt>
        <fd>Howard</fd>
        <v>1867</v>
        <rn>22</rn>
    </pre>
</stop>
```

8.6.3

XML

remove()
insert() append()
element[i] element[i:j]

Element 6.5

8.7 6.7 XML

8.7.1

XML XML

8.7.2

```
<?xml version="1.0" encoding="utf-8"?>
<top>
    <author>David Beazley</author>
    <content>
        <html xmlns="http://www.w3.org/1999/xhtml">
```

```
            <head>
                <title>Hello World</title>
            </head>
            <body>
                <h1>Hello World!</h1>
            </body>
        </html>
    </content>
</top>
```

```
>>> # Some queries that work
>>> doc.findtext('author')
'David Beazley'
>>> doc.find('content')
<Element 'content' at 0x100776ec0>
>>> # A query involving a namespace (doesn't work)
>>> doc.find('content/html')
>>> # Works if fully qualified
>>> doc.find('content/{http://www.w3.org/1999/xhtml}html')
<Element '{http://www.w3.org/1999/xhtml}html' at 0x1007767e0>
>>> # Doesn't work
>>> doc.findtext('content/{http://www.w3.org/1999/xhtml}html/head/title')
>>> # Fully qualified
>>> doc.findtext('content/{http://www.w3.org/1999/xhtml}html/'
... '{http://www.w3.org/1999/xhtml}head/{http://www.w3.org/1999/xhtml}title')
'Hello World'
>>>
```

```
class XMLNamespaces:
    def __init__(self, **kwargs):
        self.namespaces = {}
        for name, uri in kwargs.items():
            self.register(name, uri)
    def register(self, name, uri):
        self.namespaces[name] = '{'+uri+'}'
    def __call__(self, path):
        return path.format_map(self.namespaces)
```

```
>>> ns = XMLNamespaces(html='http://www.w3.org/1999/xhtml')
>>> doc.find(ns('content/{html}html'))
<Element '{http://www.w3.org/1999/xhtml}html' at 0x1007767e0>
>>> doc.findtext(ns('content/{html}html/{html}head/{html}title'))
'Hello World'
>>>
```

8.7.3

XML XMLNamespaces
URI

ElementTree
iterparse()

```
>>> from xml.etree.ElementTree import iterparse
>>> for evt, elem in iterparse('ns2.xml', ('end', 'start-ns', 'end-ns')):
... print(evt, elem)
...
end <Element 'author' at 0x10110de10>
start-ns ('', 'http://www.w3.org/1999/xhtml')
end <Element '{http://www.w3.org/1999/xhtml}title' at 0x1011131b0>
end <Element '{http://www.w3.org/1999/xhtml}head' at 0x1011130a8>
end <Element '{http://www.w3.org/1999/xhtml}h1' at 0x101113310>
end <Element '{http://www.w3.org/1999/xhtml}body' at 0x101113260>
end <Element '{http://www.w3.org/1999/xhtml}html' at 0x10110df70>
end-ns None
end <Element 'content' at 0x10110de68>
end <Element 'top' at 0x10110dd60>
>>> elem # This is the topmost element
<Element 'top' at 0x10110dd60>
>>>
```

XML XML
lxml ElementTree lxml
DTD XPath XML
XML

8.8 6.8

8.8.1

8.8.2

Python

```
stocks = [
    ('GOOG', 100, 490.1),
    ('AAPL', 50, 545.75),
    ('FB', 150, 7.45),
    ('HPQ', 75, 33.2),
]
```

PEP249 Python
API SQL

Python sqlite3
(MySql Postgresql ODBC)

connect()

```
>>> import sqlite3
>>> db = sqlite3.connect('database.db')
>>>
```

SQL

```
>>> c = db.cursor()
>>> c.execute('create table portfolio (symbol text, shares integer, price real)')
<sqlite3.Cursor object at 0x10067a730>
>>> db.commit()
>>>
```

```
>>> c.executemany('insert into portfolio values (?,?,?)', stocks)
<sqlite3.Cursor object at 0x10067a730>
>>> db.commit()
>>>
```

```
>>> for row in db.execute('select * from portfolio'):
...     print(row)
...
('GOOG', 100, 490.1)
('AAPL', 50, 545.75)
('FB', 150, 7.45)
('HPQ', 75, 33.2)
>>>
```

“?“

```
>>> min_price = 100
>>> for row in db.execute('select * from portfolio where price >= ?',
                          (min_price,)):
...     print(row)
...
('GOOG', 100, 490.1)
('AAPL', 50, 545.75)
>>>
```

8.8.3

SQL

Python
`datetime` `datetime` `time`
`decimal` `Decimal`

SQL Python
(%) `.format()`
SQL
(http://xkcd.com/327) `?`

`?`
`%s` :0 :1
`paramstyle`

API
ORM
`SQLAlchemy` Python
SQL

8.9 6.9

8.9.1

8.9.2

`binascii`

```
>>> # Initial byte string
>>> s = b'hello'
>>> # Encode as hex
>>> import binascii
>>> h = binascii.b2a_hex(s)
>>> h
b'68656c6c6f'
```

```
>>> # Decode back to bytes
>>> binascii.a2b_hex(h)
b'hello'
>>>
```

base64

```
>>> import base64
>>> h = base64.b16encode(s)
>>> h
b'68656C6C6F'
>>> base64.b16decode(h)
b'hello'
>>>
```

8.9.3

base64.b16decode() base64.b16encode()
binascii

Unicode

```
>>> h = base64.b16encode(s)
>>> print(h)
b'68656C6C6F'
>>> print(h.decode('ascii'))
68656C6C6F
>>>
```

b16decode() a2b_hex() unicode
unicode ASCII

8.10 6.10 Base64

8.10.1

Base64

8.10.2

base64 b64encode() and b64decode()
;

```
>>> # Some byte data
>>> s = b'hello'
>>> import base64

>>> # Encode as Base64
>>> a = base64.b64encode(s)
>>> a
b'aGVsbG8='

>>> # Decode from Base64
>>> base64.b64decode(a)
b'hello'
>>>
```

8.10.3

Base64

Base64 Unicode

```
>>> a = base64.b64encode(s).decode('ascii')
>>> a
'aGVsbG8='
>>>
```

Base64 Unicode Unicode ASCII

8.11 6.11

8.11.1

Python

8.11.2

struct Python

struct

```
from struct import Struct
def write_records(records, format, f):
    '''
    Write a sequence of tuples to a binary file of structures.
    '''
    record_struct = Struct(format)
    for r in records:
```

```
        f.write(record_struct.pack(*r))

# Example
if __name__ == '__main__':
    records = [ (1, 2.3, 4.5),
                (6, 7.8, 9.0),
                (12, 13.4, 56.7) ]
    with open('data.b', 'wb') as f:
        write_records(records, '<idd', f)
```

```
from struct import Struct

def read_records(format, f):
    record_struct = Struct(format)
    chunks = iter(lambda: f.read(record_struct.size), b'')
    return (record_struct.unpack(chunk) for chunk in chunks)

# Example
if __name__ == '__main__':
    with open('data.b','rb') as f:
        for rec in read_records('<idd', f):
            # Process rec
            ...
```

```
from struct import Struct

def unpack_records(format, data):
    record_struct = Struct(format)
    return (record_struct.unpack_from(data, offset)
            for offset in range(0, len(data), record_struct.size))

# Example
if __name__ == '__main__':
    with open('data.b', 'rb') as f:
        data = f.read()
    for rec in unpack_records('<idd', data):
        # Process rec
        ...
```

8.11.3

struct

Struct

```
# Little endian 32-bit integer, two double precision floats
record_struct = Struct('<idd')
```

i, d, f [Python]
32 64 32
< ” ” >
!

Struct size
I/O pack() unpack()

```
>>> from struct import Struct
>>> record_struct = Struct('<idd')
>>> record_struct.size
20
>>> record_struct.pack(1, 2.0, 3.0)
b'\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00@\x00\x00\x00\x00\x00\x00\x08@'
>>> record_struct.unpack(_)
(1, 2.0, 3.0)
>>>
```

pack() unpack()

```
>>> import struct
>>> struct.pack('<idd', 1, 2.0, 3.0)
b'\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00@\x00\x00\x00\x00\x00\x00\x08@'
>>> struct.unpack('<idd', _)
(1, 2.0, 3.0)
>>>
```

Struct
(
)

read_records iter()
5.8 (lambda:
f.read(record_struct.size)) (b‘)

```
>>> f = open('data.b', 'rb')
>>> chunks = iter(lambda: f.read(20), b'')
>>> chunks
<callable_iterator object at 0x10069e6d0>
>>> for chk in chunks:
... print(chk)
...
b'\x01\x00\x00\x00ffffff\x02@\x00\x00\x00\x00\x00\x00\x12@'
b'\x06\x00\x00\x00333333\x1f@\x00\x00\x00\x00\x00\x00"@'
```

```
b'\x0c\x00\x00\x00\xcd\xcc\xcc\xcc\xcc\xcc*@\x9a\x99\x99\x99\x99YL@'
>>>
```

```
def read_records(format, f):
    record_struct = Struct(format)
    while True:
        chk = f.read(record_struct.size)
        if chk == b'':
            break
        yield record_struct.unpack(chk)
```

unpack_records() unpack_from()
unpack_from()
(
)

unpack() unpack_from()

```
def unpack_records(format, data):
    record_struct = Struct(format)
    return (record_struct.unpack(data[offset:offset + record_struct.size])
            for offset in range(0, len(data), record_struct.size))
```

unpack_from()

collections

```
from collections import namedtuple

Record = namedtuple('Record', ['kind','x','y'])

with open('data.p', 'rb') as f:
    records = (Record(*r) for r in read_records('<idd', f))

for r in records:
    print(r.kind, r.x, r.y)
```

numpy

```
>>> import numpy as np
>>> f = open('data.b', 'rb')
>>> records = np.fromfile(f, dtype='<i,<d,<d')
>>> records
array([(1, 2.3, 4.5), (6, 7.8, 9.0), (12, 13.4, 56.7)],
```

```
dtype=[('f0', '<i4'), ('f1', '<f8'), ('f2', '<f8')])
>>> records[0]
(1, 2.3, 4.5)
>>> records[1]
(6, 7.8, 9.0)
>>>
```

(HDF5)
Python

8.12 6.12

8.12.1

8.12.2

struct /
Python

```
polys = [
    [ (1.0, 2.5), (3.5, 4.0), (2.5, 1.5) ],
    [ (7.0, 1.2), (5.1, 3.0), (0.5, 7.5), (0.8, 9.0) ],
    [ (3.4, 6.3), (1.2, 0.5), (4.6, 9.2) ],
]
```

```
+------+--------+-----------------------------------+
|Byte  | Type   |  Description                      |
+======+========+===================================+
|0     | int    |          0x1234                   |
+------+--------+-----------------------------------+
|4     | double |  x                                |
+------+--------+-----------------------------------+
|12    | double |  y                                |
+------+--------+-----------------------------------+
|20    | double |  x                                |
+------+--------+-----------------------------------+
|28    | double |  y                                |
+------+--------+-----------------------------------+
|36    | int    |                                   |
+------+--------+-----------------------------------+
```

```
+------+--------+-----------------------------------------+
|Byte  | Type   |  Description                            |
+======+========+=========================================+
|0     | int    |          N                              |
+------+--------+-----------------------------------------+
|4-N   | Points |  (X,Y)                                  |
+------+--------+-----------------------------------------+
```

Python

```
import struct
import itertools

def write_polys(filename, polys):
    # Determine bounding box
    flattened = list(itertools.chain(*polys))
    min_x = min(x for x, y in flattened)
    max_x = max(x for x, y in flattened)
    min_y = min(y for x, y in flattened)
    max_y = max(y for x, y in flattened)
    with open(filename, 'wb') as f:
        f.write(struct.pack('<iddddi', 0x1234,
                            min_x, min_y,
                            max_x, max_y,
                            len(polys)))
        for poly in polys:
            size = len(poly) * struct.calcsize('<dd')
            f.write(struct.pack('<i', size + 4))
            for pt in poly:
                f.write(struct.pack('<dd', *pt))
```

struct.unpack()

```
def read_polys(filename):
    with open(filename, 'rb') as f:
        # Read the header
        header = f.read(40)
        file_code, min_x, min_y, max_x, max_y, num_polys = \
            struct.unpack('<iddddi', header)
        polys = []
        for n in range(num_polys):
            pbytes, = struct.unpack('<i', f.read(4))
            poly = []
            for m in range(pbytes // 16):
                pt = struct.unpack('<dd', f.read(16))
                poly.append(pt)
            polys.append(poly)
    return polys
```

struct

```
import struct

class StructField:
    '''
    Descriptor representing a simple structure field
    '''
    def __init__(self, format, offset):
        self.format = format
        self.offset = offset
    def __get__(self, instance, cls):
        if instance is None:
            return self
        else:
            r = struct.unpack_from(self.format, instance._buffer, self.offset)
            return r[0] if len(r) == 1 else r

class Structure:
    def __init__(self, bytedata):
        self._buffer = memoryview(bytedata)
```

__get__()
struct.unpack_from()

Structure
StructField memoryview()

```
class PolyHeader(Structure):
    file_code = StructField('<i', 0)
    min_x = StructField('<d', 4)
    min_y = StructField('<d', 12)
    max_x = StructField('<d', 20)
    max_y = StructField('<d', 28)
    num_polys = StructField('<i', 36)
```

```
>>> f = open('polys.bin', 'rb')
>>> phead = PolyHeader(f.read(40))
>>> phead.file_code == 0x1234
True
>>> phead.min_x
0.5
>>> phead.min_y
0.5
>>> phead.max_x
7.0
>>> phead.max_y
9.2
>>> phead.num_polys
3
>>>
```

(

StructField)

Structure

```
class StructureMeta(type):
    '''
    Metaclass that automatically creates StructField descriptors
    '''
    def __init__(self, clsname, bases, clsdict):
        fields = getattr(self, '_fields_', [])
        byte_order = ''
        offset = 0
        for format, fieldname in fields:
            if format.startswith(('<','>','!','@')):
                byte_order = format[0]
                format = format[1:]
            format = byte_order + format
            setattr(self, fieldname, StructField(format, offset))
            offset += struct.calcsize(format)
        setattr(self, 'struct_size', offset)

class Structure(metaclass=StructureMeta):
    def __init__(self, bytedata):
        self._buffer = bytedata

    @classmethod
    def from_file(cls, f):
        return cls(f.read(cls.struct_size))
```

Structure

```
class PolyHeader(Structure):
    _fields_ = [
        ('<i', 'file_code'),
        ('d', 'min_x'),
        ('d', 'min_y'),
        ('d', 'max_x'),
        ('d', 'max_y'),
        ('i', 'num_polys')
    ]
```

from_file()

```
>>> f = open('polys.bin', 'rb')
>>> phead = PolyHeader.from_file(f)
>>> phead.file_code == 0x1234
True
>>> phead.min_x
0.5
>>> phead.min_y
0.5
>>> phead.max_x
7.0
>>> phead.max_y
9.2
>>> phead.num_polys
3
>>>
```

```
class NestedStruct:
    '''
    Descriptor representing a nested structure
    '''
    def __init__(self, name, struct_type, offset):
        self.name = name
        self.struct_type = struct_type
        self.offset = offset

    def __get__(self, instance, cls):
        if instance is None:
            return self
        else:
            data = instance._buffer[self.offset:
                                    self.offset+self.struct_type.struct_size]
            result = self.struct_type(data)
            # Save resulting structure back on instance to avoid
```

```
            # further recomputation of this step
            setattr(instance, self.name, result)
            return result

class StructureMeta(type):
    '''
    Metaclass that automatically creates StructField descriptors
    '''
    def __init__(self, clsname, bases, clsdict):
        fields = getattr(self, '_fields_', [])
        byte_order = ''
        offset = 0
        for format, fieldname in fields:
            if isinstance(format, StructureMeta):
                setattr(self, fieldname,
                        NestedStruct(fieldname, format, offset))
                offset += format.struct_size
            else:
                if format.startswith(('<','>','!','@')):
                    byte_order = format[0]
                    format = format[1:]
                format = byte_order + format
                setattr(self, fieldname, StructField(format, offset))
                offset += struct.calcsize(format)
        setattr(self, 'struct_size', offset)
```

NestedStruct

8.10

```
class Point(Structure):
    _fields_ = [
        ('<d', 'x'),
        ('d', 'y')
    ]

class PolyHeader(Structure):
    _fields_ = [
        ('<i', 'file_code'),
        (Point, 'min'), # nested struct
        (Point, 'max'), # nested struct
        ('i', 'num_polys')
    ]
```

```
>>> f = open('polys.bin', 'rb')
>>> phead = PolyHeader.from_file(f)
>>> phead.file_code == 0x1234
True
>>> phead.min # Nested structure
<__main__.Point object at 0x1006a48d0>
>>> phead.min.x
0.5
>>> phead.min.y
0.5
>>> phead.max.x
7.0
>>> phead.max.y
9.2
>>> phead.num_polys
3
>>>
```

6.11

```
class SizedRecord:
    def __init__(self, bytedata):
        self._buffer = memoryview(bytedata)

    @classmethod
    def from_file(cls, f, size_fmt, includes_size=True):
        sz_nbytes = struct.calcsize(size_fmt)
        sz_bytes = f.read(sz_nbytes)
        sz, = struct.unpack(size_fmt, sz_bytes)
        buf = f.read(sz - includes_size * sz_nbytes)
        return cls(buf)

    def iter_as(self, code):
        if isinstance(code, str):
            s = struct.Struct(code)
            for off in range(0, len(self._buffer), s.size):
                yield s.unpack_from(self._buffer, off)
        elif isinstance(code, StructureMeta):
            size = code.struct_size
            for off in range(0, len(self._buffer), size):
                data = self._buffer[off:off+size]
                yield code(data)
```

SizedRecord.from_file()

includes_size

```
>>> f = open('polys.bin', 'rb')
>>> phead = PolyHeader.from_file(f)
>>> phead.num_polys
3
>>> polydata = [ SizedRecord.from_file(f, '<i')
...              for n in range(phead.num_polys) ]
>>> polydata
[<__main__.SizedRecord object at 0x1006a4d50>,
<__main__.SizedRecord object at 0x1006a4f50>,
<__main__.SizedRecord object at 0x10070da90>]
>>>
```

SizedRecord iter_as()
Structure

```
>>> for n, poly in enumerate(polydata):
...     print('Polygon', n)
...     for p in poly.iter_as('<dd'):
...         print(p)
...
Polygon 0
(1.0, 2.5)
(3.5, 4.0)
(2.5, 1.5)
Polygon 1
(7.0, 1.2)
(5.1, 3.0)
(0.5, 7.5)
(0.8, 9.0)
Polygon 2
(3.4, 6.3)
(1.2, 0.5)
(4.6, 9.2)
>>>

>>> for n, poly in enumerate(polydata):
...     print('Polygon', n)
...     for p in poly.iter_as(Point):
...         print(p.x, p.y)
...
Polygon 0
1.0 2.5
3.5 4.0
2.5 1.5
Polygon 1
7.0 1.2
5.1 3.0
0.5 7.5
0.8 9.0
Polygon 2
```

```
3.4 6.3
1.2 0.5
4.6 9.2
>>>
```

read_polys()

```
class Point(Structure):
    _fields_ = [
        ('<d', 'x'),
        ('d', 'y')
    ]

class PolyHeader(Structure):
    _fields_ = [
        ('<i', 'file_code'),
        (Point, 'min'),
        (Point, 'max'),
        ('i', 'num_polys')
    ]

def read_polys(filename):
    polys = []
    with open(filename, 'rb') as f:
        phead = PolyHeader.from_file(f)
        for n in range(phead.num_polys):
            rec = SizedRecord.from_file(f, '<i')
            poly = [ (p.x, p.y) for p in rec.iter_as(Point) ]
            polys.append(poly)
    return polys
```

8.12.3

Structure

__init__()

StructField _fields_

StructField

StructureMeta

StructureMeta

(< >)

```
class ShapeFile(Structure):
    _fields_ = [ ('>i', 'file_code'), # Big endian
        ('20s', 'unused'),
        ('i', 'file_length'),
        ('<i', 'version'), # Little endian
        ('i', 'shape_type'),
        ('d', 'min_x'),
        ('d', 'min_y'),
        ('d', 'max_x'),
        ('d', 'max_y'),
        ('d', 'min_z'),
        ('d', 'max_z'),
        ('d', 'min_m'),
        ('d', 'max_m') ]
```

memoryview()
memoryviews

8.13

8.10

NestedStruct 9.19
StructureMeta Python ctypes

8.13 6.13

8.13.1

8.13.2

Pandas

Pandas

74,000

CSV

```
>>> import pandas

>>> # Read a CSV file, skipping last line
>>> rats = pandas.read_csv('rats.csv', skip_footer=1)
>>> rats
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74055 entries, 0 to 74054
Data columns:
Creation Date 74055 non-null values
Status 74055 non-null values
Completion Date 72154 non-null values
Service Request Number 74055 non-null values
Type of Service Request 74055 non-null values
Number of Premises Baited 65804 non-null values
Number of Premises with Garbage 65600 non-null values
Number of Premises with Rats 65752 non-null values
Current Activity 66041 non-null values
Most Recent Action 66023 non-null values
Street Address 74055 non-null values
ZIP Code 73584 non-null values
X Coordinate 74043 non-null values
Y Coordinate 74043 non-null values
Ward 74044 non-null values
Police District 74044 non-null values
Community Area 74044 non-null values
Latitude 74043 non-null values
Longitude 74043 non-null values
Location 74043 non-null values
dtypes: float64(11), object(9)

>>> # Investigate range of values for a certain field
>>> rats['Current Activity'].unique()
array([nan, Dispatch Crew, Request Sanitation Inspector], dtype=object)
>>> # Filter the data
>>> crew_dispatched = rats[rats['Current Activity'] == 'Dispatch Crew']
>>> len(crew_dispatched)
65676
>>>

>>> # Find 10 most rat-infested ZIP codes in Chicago
>>> crew_dispatched['ZIP Code'].value_counts()[:10]
60647 3837
60618 3530
60614 3284
60629 3251
60636 2801
60657 2465
60641 2238
60609 2206
60651 2152
60632 2071
```

```
>>>

>>> # Group by completion date
>>> dates = crew_dispatched.groupby('Completion Date')
<pandas.core.groupby.DataFrameGroupBy object at 0x10d0a2a10>
>>> len(dates)
472
>>>

>>> # Determine counts on each day
>>> date_counts = dates.size()
>>> date_counts[0:10]
Completion Date
01/03/2011 4
01/03/2012 125
01/04/2011 54
01/04/2012 38
01/05/2011 78
01/05/2012 100
01/06/2011 100
01/06/2012 58
01/07/2011 1
01/09/2012 12
>>>

>>> # Sort the counts
>>> date_counts.sort()
>>> date_counts[-10:]
Completion Date
10/12/2012 313
10/21/2011 314
09/20/2011 316
10/26/2011 319
02/22/2011 325
10/26/2012 333
03/17/2011 336
10/13/2011 378
10/14/2011 391
10/07/2011 457
>>>
```

2011 10 7 ^_^

8.13.3

Pandas

CHAPTER

NINE

def

Contents:

9.1 7.1

9.1.1

9.1.2

*

```
def avg(first, *rest):
    return (first + sum(rest)) / (1 + len(rest))

# Sample use
avg(1, 2) # 1.5
avg(1, 2, 3, 4) # 2.5
```

rest

**

```
import html

def make_element(name, value, **attrs):
    keyvals = [' %s="%s"' % item for item in attrs.items()]
    attr_str = ''.join(keyvals)
    element = '<{name}{attrs}>{value}</{name}>'.format(
```

```
                name=name,
                attrs=attr_str,
                value=html.escape(value))
    return element

# Example
# Creates '<item size="large" quantity="6">Albatross</item>'
make_element('item', 'Albatross', size='large', quantity=6)

# Creates '<p>&lt;spam&gt;</p>'
make_element('p', '<spam>')
```

attrs

* **

```
def anyargs(*args, **kwargs):
    print(args) # A tuple
    print(kwargs) # A dict
```

args

kwargs

9.1.3

* **

*

```
def a(x, *args, y):
    pass

def b(x, *args, y, **kwargs):
    pass
```

7.2

9.2 7.2

9.2.1

9.2.2

* *

```
def recv(maxsize, *, block):
    'Receives a message'
    pass

recv(1024, True) # TypeError
recv(1024, block=True) # Ok
```

```
def mininum(*values, clip=None):
    m = min(values)
    if clip is not None:
        m = clip if clip > m else m
    return m

minimum(1, 5, 2, -5, 10) # Returns -5
minimum(1, 5, 2, -5, 10, clip=0) # Returns 0
```

9.2.3

```
msg = recv(1024, False)
```

recv False

```
msg = recv(1024, block=False)
```

**kwargs help

```
>>> help(recv)
Help on function recv in module __main__:
recv(maxsize, *, block)
    Receives a message
```

*args **kwargs 9.11

9.3 7.3

9.3.1

9.3.2

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

python

```
>>> help(add)
Help on function add in module __main__:
add(x: int, y: int) -> int
>>>
```

(

)

9.3.3

__annotations__

```
>>> add.__annotations__
{'y': <class 'int'>, 'return': <class 'int'>, 'x': <class 'int'>}
```

python

9.20 (

)

9.4 7.4

9.4.1

9.4.2

return

```
>>> def myfun():
... return 1, 2, 3
...
```

```
>>> a, b, c = myfun()
>>> a
1
>>> b
2
>>> c
3
```

9.4.3

myfun()

```
>>> a = (1, 2) # With parentheses
>>> a
(1, 2)
>>> b = 1, 2 # Without parentheses
>>> b
(1, 2)
>>>
```

1.1

```
>>> x = myfun()
>>> x
(1, 2, 3)
>>>
```

9.5 7.5

9.5.1

9.5.2

```
def spam(a, b=42):
    print(a, b)
```

```
spam(1) # Ok. a=1, b=42
spam(1, 2) # Ok. a=1, b=2
```

None

```
# Using a list as a default value
def spam(a, b=None):
    if b is None:
        b = []
    ...
```

```
_no_value = object()

def spam(a, b=_no_value):
    if b is _no_value:
        print('No b value supplied')
    ...
```

```
>>> spam(1)
No b value supplied
>>> spam(1, 2) # b = 2
>>> spam(1, None) # b = None
>>>
```

None

9.5.3

```
>>> x = 42
>>> def spam(a, b=x):
...     print(a, b)
...
>>> spam(1)
1 42
>>> x = 23 # Has no effect
>>> spam(1)
1 42
>>>
```

x

None True False

```
def spam(a, b=[]): # NO!
    ...
```

```
>>> def spam(a, b=[]):
...     print(b)
...     return b
...
>>> x = spam(1)
>>> x
[]
>>> x.append(99)
>>> x.append('Yow!')
>>> x
[99, 'Yow!']
>>> spam(1) # Modified list gets returned!
[99, 'Yow!']
>>>
```

None

None is

```
def spam(a, b=None):
    if not b: # NO! Use 'b is None' instead
        b = []
    ...
```

None False (
0) False

```
>>> spam(1) # OK
>>> x = []
>>> spam(1, x) # Silent error. x value overwritten by default
>>> spam(1, 0) # Silent error. 0 ignored
>>> spam(1, '') # Silent error. '' ignored
>>>
```

None 0 False
( )

_no_value

_no_value

object() object python

( )

9.6 7.6

9.6.1

sort() def

9.6.2

lambda

```
>>> add = lambda x, y: x + y
>>> add(2,3)
5
>>> add('hello', 'world')
'helloworld'
>>>
```

lambda

```
>>> def add(x, y):
...     return x + y
...
>>> add(2,3)
5
>>>
```

lambda reduce

```
>>> names = ['David Beazley', 'Brian Jones',
...          'Raymond Hettinger', 'Ned Batchelder']
>>> sorted(names, key=lambda name: name.split()[-1].lower())
['Ned Batchelder', 'David Beazley', 'Raymond Hettinger', 'Brian Jones']
>>>
```

9.6.3

lambda

lambda python

lambda

9.7 7.7

9.7.1

lambda

9.7.2

```
>>> x = 10
>>> a = lambda y: x + y
>>> x = 20
>>> b = lambda y: x + y
>>>
```

a(10) b(10) 20 30

```
>>> a(10)
30
>>> b(10)
30
>>>
```

lambda x

lambda

x

```
>>> x = 15
>>> a(10)
25
>>> x = 3
>>> a(10)
13
>>>
```

```
>>> x = 10
>>> a = lambda y, x=x: x + y
>>> x = 20
>>> b = lambda y, x=x: x + y
>>> a(10)
20
>>> b(10)
30
>>>
```

9.7.3

lambda lambda

```
>>> funcs = [lambda x: x+n for n in range(5)]
>>> for f in funcs:
... print(f(0))
...
4
4
4
4
4
>>>
```

n

```
>>> funcs = [lambda x, n=n: x+n for n in range(5)]
>>> for f in funcs:
... print(f(0))
...
0
1
2
3
4
>>>
```

lambda

9.8 7.8

9.8.1

python callable

9.8.2

functools.partial()

partial()

```
def spam(a, b, c, d):
    print(a, b, c, d)
```

partial()

```
>>> from functools import partial
>>> s1 = partial(spam, 1) # a = 1
>>> s1(2, 3, 4)
1 2 3 4
>>> s1(4, 5, 6)
1 4 5 6
>>> s2 = partial(spam, d=42) # d = 42
>>> s2(1, 2, 3)
1 2 3 42
>>> s2(4, 5, 5)
4 5 5 42
>>> s3 = partial(spam, 1, 2, d=42) # a = 1, b = 2, d = 42
>>> s3(3)
1 2 3 42
>>> s3(4)
1 2 4 42
>>> s3(5)
1 2 5 42
>>>
```

partial() callable callable

9.8.3

(x,y)

```
points = [ (1, 2), (3, 4), (5, 6), (7, 8) ]

import math
def distance(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return math.hypot(x2 - x1, y2 - y1)
```

sort()
(distance()) partial()

```
>>> pt = (4, 3)
>>> points.sort(key=partial(distance,pt))
>>> points
[(3, 4), (1, 2), (5, 6), (7, 8)]
>>>
```

partial()
multiprocessing
result logging

```
def output_result(result, log=None):
    if log is not None:
        log.debug('Got: %r', result)

# A sample function
def add(x, y):
    return x + y

if __name__ == '__main__':
    import logging
    from multiprocessing import Pool
    from functools import partial

    logging.basicConfig(level=logging.DEBUG)
    log = logging.getLogger('test')

    p = Pool()
    p.apply_async(add, (3, 4), callback=partial(output_result, log=log))
    p.close()
    p.join()
```

apply_async() partial() logging
multiprocessing ——

socketserver

echo

```
from socketserver import StreamRequestHandler, TCPServer

class EchoHandler(StreamRequestHandler):
    def handle(self):
        for line in self.rfile:
            self.wfile.write(b'GOT:' + line)

serv = TCPServer(('', 15000), EchoHandler)
serv.serve_forever()
```

EchoHandler __init__

```
class EchoHandler(StreamRequestHandler):
    # ack is added keyword-only argument. *args, **kwargs are
    # any normal parameters supplied (which are passed on)
    def __init__(self, *args, ack, **kwargs):
        self.ack = ack
        super().__init__(*args, **kwargs)

    def handle(self):
        for line in self.rfile:
            self.wfile.write(self.ack + line)
```

TCPServer

```
Exception happened during processing of request from ('127.0.0.1', 59834)
Traceback (most recent call last):
...
TypeError: __init__() missing 1 required keyword-only argument: 'ack'
```

socketserver
partial() —— ack

```
from functools import partial
serv = TCPServer(('', 15000), partial(EchoHandler, ack=b'RECEIVED:'))
serv.serve_forever()
```

__init__() ack
ack 7.2

partial() lambda

```
points.sort(key=lambda p: distance(pt, p))
p.apply_async(add, (3, 4), callback=lambda result: output_result(result,log))
serv = TCPServer(('', 15000),
        lambda *args, **kwargs: EchoHandler(*args, ack=b'RECEIVED:', **kwargs))
```

partial() (
)

9.9 7.9

9.9.1

__init__()

9.9.2

URL

```
from urllib.request import urlopen

class UrlTemplate:
    def __init__(self, template):
        self.template = template

    def open(self, **kwargs):
        return urlopen(self.template.format_map(kwargs))

# Example use. Download stock data from yahoo
yahoo = UrlTemplate('http://finance.yahoo.com/d/quotes.csv?s={names}&f={fields}')
for line in yahoo.open(names='IBM,AAPL,FB', fields='sl1c1v'):
    print(line.decode('utf-8'))
```

```
def urltemplate(template):
    def opener(**kwargs):
        return urlopen(template.format_map(kwargs))
    return opener

# Example use
yahoo = urltemplate('http://finance.yahoo.com/d/quotes.csv?s={names}&f={fields}')
for line in yahoo(names='IBM,AAPL,FB', fields='sl1c1v'):
    print(line.decode('utf-8'))
```

9.9.3

UrlTemplate

open()

opener() template

9.10 7.10

9.10.1

(

)

9.10.2

___

```
def apply_async(func, args, *, callback):
    # Compute the result
    result = func(*args)

    # Invoke the callback with the result
    callback(result)
```

```
>>> def print_result(result):
...     print('Got:', result)
...
>>> def add(x, y):
...     return x + y
...
>>> apply_async(add, (2, 3), callback=print_result)
Got: 5
>>> apply_async(add, ('hello', 'world'), callback=print_result)
Got: helloworld
>>>
```

print_result() result

result

1

```
class ResultHandler:

    def __init__(self):
        self.sequence = 0

    def handler(self, result):
        self.sequence += 1
        print('[{}] Got: {}'.format(self.sequence, result))
```

handler()

```
>>> r = ResultHandler()
>>> apply_async(add, (2, 3), callback=r.handler)
[1] Got: 5
>>> apply_async(add, ('hello', 'world'), callback=r.handler)
[2] Got: helloworld
>>>
```

```
def make_handler():
    sequence = 0
    def handler(result):
        nonlocal sequence
        sequence += 1
        print('[{}] Got: {}'.format(sequence, result))
    return handler
```

```
>>> handler = make_handler()
>>> apply_async(add, (2, 3), callback=handler)
[1] Got: 5
>>> apply_async(add, ('hello', 'world'), callback=handler)
[2] Got: helloworld
>>>
```

```
def make_handler():
    sequence = 0
    while True:
        result = yield
        sequence += 1
        print('[{}] Got: {}'.format(sequence, result))
```

send()

```
>>> handler = make_handler()
>>> next(handler) # Advance to the yield
>>> apply_async(add, (2, 3), callback=handler.send)
[1] Got: 5
>>> apply_async(add, ('hello', 'world'), callback=handler.send)
[2] Got: helloworld
>>>
```

9.10.3

(
)

( )

nonlocal

nonlocal Python
next()

( )

partial()
partial() lambda

```
>>> apply_async(add, (2, 3), callback=lambda r: handler(r, seq))
[1] Got: 5
>>>
```

7.8 partial()

9.11 7.11

9.11.1

9.11.2

(7.10)

```
def apply_async(func, args, *, callback):
    # Compute the result
    result = func(*args)

    # Invoke the callback with the result
    callback(result)
```

Async inlined_async

```
from queue import Queue
from functools import wraps

class Async:
    def __init__(self, func, args):
        self.func = func
        self.args = args

def inlined_async(func):
    @wraps(func)
    def wrapper(*args):
        f = func(*args)
        result_queue = Queue()
        result_queue.put(None)
        while True:
            result = result_queue.get()
            try:
                a = f.send(result)
                apply_async(a.func, a.args, callback=result_queue.put)
            except StopIteration:
                break
    return wrapper
```

yield

```
def add(x, y):
    return x + y

@inlined_async
def test():
    r = yield Async(add, (2, 3))
    print(r)
    r = yield Async(add, ('hello', 'world'))
    print(r)
    for n in range(10):
        r = yield Async(add, (n, n))
```

```
        print(r)
    print('Goodbye')
```

test()

```
5
helloworld
0
2
4
6
8
10
12
14
16
18
Goodbye
```

yield
( )

9.11.3

( )
apply_async()
( )

yield
__next__() send()

inline_async()
yield
result None
yield

Async
apply_async()
put()

get() put()
apply_async()
multiprocessing

```
if __name__ == '__main__':
    import multiprocessing
    pool = multiprocessing.Pool()
    apply_async = pool.apply_async

    # Run the test function
    test()
```

contextlib @contextmanager
yield Twisted

9.12 7.12

9.12.1

9.12.2

```
def sample():
    n = 0
    # Closure function
    def func():
        print('n=', n)

    # Accessor methods for n
    def get_n():
        return n

    def set_n(value):
        nonlocal n
        n = value

    # Attach as function attributes
    func.get_n = get_n
    func.set_n = set_n
    return func
```

:

```
>>> f = sample()
>>> f()
n= 0
>>> f.set_n(10)
>>> f()
n= 10
>>> f.get_n()
10
>>>
```

9.12.3

nonlocal

( )

```
import sys
class ClosureInstance:
    def __init__(self, locals=None):
        if locals is None:
            locals = sys._getframe(1).f_locals

        # Update instance dictionary with callables
        self.__dict__.update((key,value) for key, value in locals.items()
                             if callable(value) )
    # Redirect special methods
    def __len__(self):
        return self.__dict__['__len__']()

# Example use
def Stack():
    items = []
    def push(item):
        items.append(item)

    def pop():
        return items.pop()

    def __len__():
        return len(items)

    return ClosureInstance()
```

```
>>> s = Stack()
>>> s
```

```
<__main__.ClosureInstance object at 0x10069ed10>
>>> s.push(10)
>>> s.push(20)
>>> s.push('Hello')
>>> len(s)
3
>>> s.pop()
'Hello'
>>> s.pop()
20
>>> s.pop()
10
>>>
```

```
class Stack2:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

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

    def __len__(self):
        return len(self.items)
```

```
>>> from timeit import timeit
>>> # Test involving closures
>>> s = Stack()
>>> timeit('s.push(1);s.pop()', 'from __main__ import s')
0.9874754269840196
>>> # Test involving a class
>>> s = Stack2()
>>> timeit('s.push(1);s.pop()', 'from __main__ import s')
1.0707052160287276
>>>
```

8%
self

Raymond Hettinger

(ClosureInstance __len__()
)

( )

ض ع r ő

CHAPTER

TEN

Python

Contents:

10.1 8.1

10.1.1

10.1.2

__str__() __repr__()

```
class Pair:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Pair({0.x!r}, {0.y!r})'.format(self)

    def __str__(self):
        return '({0.x!s}, {0.y!s})'.format(self)
```

__repr__()
repr()
__str__() str() print()

```
>>> p = Pair(3, 4)
>>> p
Pair(3, 4) # __repr__() output
>>> print(p)
(3, 4) # __str__() output
>>>
```

!r __repr__() __str__()

```
>>> p = Pair(3, 4)
>>> print('p is {0!r}'.format(p))
p is Pair(3, 4)
>>> print('p is {0}'.format(p))
p is (3, 4)
>>>
```

10.1.3

__repr__() __str__()

__repr__() eval(repr(x)) == x
< >

```
>>> f = open('file.dat')
>>> f
<_io.TextIOWrapper name='file.dat' mode='r' encoding='UTF-8'>
>>>
```

__str__() __repr__()

format() {0.x} 1
x 0 self

```
def __repr__(self):
    return 'Pair({0.x!r}, {0.y!r})'.format(self)
```

%

```
def __repr__(self):
    return 'Pair(%r, %r)' % (self.x, self.y)
```

10.2 8.2

10.2.1

format()

10.2.2

__format__()

```
_formats = {
    'ymd' : '{d.year}-{d.month}-{d.day}',
    'mdy' : '{d.month}/{d.day}/{d.year}',
    'dmy' : '{d.day}/{d.month}/{d.year}'
    }

class Date:
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day

    def __format__(self, code):
        if code == '':
            code = 'ymd'
        fmt = _formats[code]
        return fmt.format(d=self)
```

Date

```
>>> d = Date(2012, 12, 21)
>>> format(d)
'2012-12-21'
>>> format(d, 'mdy')
'12/21/2012'
>>> 'The date is {:ymd}'.format(d)
'The date is 2012-12-21'
>>> 'The date is {:mdy}'.format(d)
'The date is 12/21/2012'
>>>
```

10.2.3

__format__() Python

datetime

```
>>> from datetime import date
>>> d = date(2012, 12, 21)
>>> format(d)
'2012-12-21'
>>> format(d,'%A, %B %d, %Y')
'Friday, December 21, 2012'
>>> 'The end is {:%d %b %Y}. Goodbye'.format(d)
'The end is 21 Dec 2012. Goodbye'
>>>
```

string

10.3 8.3

10.3.1

(with)

10.3.2

with __enter__() __exit__()

```
from socket import socket, AF_INET, SOCK_STREAM

class LazyConnection:
    def __init__(self, address, family=AF_INET, type=SOCK_STREAM):
        self.address = address
        self.family = family
        self.type = type
        self.sock = None

    def __enter__(self):
        if self.sock is not None:
            raise RuntimeError('Already connected')
        self.sock = socket(self.family, self.type)
        self.sock.connect(self.address)
        return self.sock

    def __exit__(self, exc_ty, exc_val, tb):
        self.sock.close()
        self.sock = None
```

( ) with

```
from functools import partial

conn = LazyConnection(('www.python.org', 80))
# Connection closed
with conn as s:
    # conn.__enter__() executes: connection open
    s.send(b'GET /index.html HTTP/1.0\r\n')
    s.send(b'Host: www.python.org\r\n')
    s.send(b'\r\n')
    resp = b''.join(iter(partial(s.recv, 8192), b''))
    # conn.__exit__() executes: connection closed
```

10.3.3

with with
__enter__() ( )
as with __exit__()

with
__exit__()
( ) __exit__()
None __exit__() True
with

LazyConnection with
socket socket
with

```
from socket import socket, AF_INET, SOCK_STREAM

class LazyConnection:
    def __init__(self, address, family=AF_INET, type=SOCK_STREAM):
        self.address = address
        self.family = family
        self.type = type
        self.connections = []

    def __enter__(self):
        sock = socket(self.family, self.type)
        sock.connect(self.address)
        self.connections.append(sock)
        return sock

    def __exit__(self, exc_ty, exc_val, tb):
        self.connections.pop().close()

# Example use
```

```
from functools import partial

conn = LazyConnection(('www.python.org', 80))
with conn as s1:
    pass
    with conn as s2:
        pass
        # s1 and s2 are independent sockets
```

LazyConnection
__enter__()
__exit__()
with

__enter__() __exit__() with
__exit__()

contextmanager 9.22
12.6

10.4 8.4

10.4.1

( )

10.4.2

__slots__

```
class Date:
    __slots__ = ['year', 'month', 'day']
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day
```

__slots__ Python

__slots__
slots __slots__

10.4.3

slots

slots Date 64 Python 428
slots 156

slots
Python slots

slots ( )

__slots__
slots __slots__

10.5 8.5

10.5.1

“ ” Python

10.5.2

Python

_

```
class A:
    def __init__(self):
        self._internal = 0 # An internal attribute
        self.public = 1 # A public attribute

    def public_method(self):
        '''
        A public method
        '''
        pass

    def _internal_method(self):
        pass
```

Python

(_socket)

sys._getframe()

(__)

```
class B:
    def __init__(self):
        self.__private = 0

    def __private_method(self):
        pass

    def public_method(self):
        pass
        self.__private_method()
```

B

_B__private _B__private_method

____

```
class C(B):
    def __init__(self):
        super().__init__()
        self.__private = 1 # Does not override B.__private

    # Does not override B.__private_method()
    def __private_method(self):
        pass
```

__private __private_method _C__private

_C__private_method B

10.5.3

( )

```
lambda_ = 2.0 # Trailing _ to avoid clash with lambda keyword
```

(

)

10.6 8.6

10.6.1

attribute

10.6.2

property

property

```
class Person:
    def __init__(self, first_name):
        self.first_name = first_name

    # Getter function
    @property
    def first_name(self):
        return self._first_name

    # Setter function
    @first_name.setter
    def first_name(self, value):
        if not isinstance(value, str):
            raise TypeError('Expected a string')
        self._first_name = value

    # Deleter function (optional)
    @first_name.deleter
    def first_name(self):
        raise AttributeError("Can't delete attribute")
```

getter first_name first_name
setter deleter first_name
@first_name.setter @first_name.deleter

property attribute
getter setter deleter

```
>>> a = Person('Guido')
>>> a.first_name # Calls the getter
'Guido'
>>> a.first_name = 42 # Calls the setter
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "prop.py", line 14, in first_name
        raise TypeError('Expected a string')
```

```
TypeError: Expected a string
>>> del a.first_name
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: can`t delete attribute
>>>
```

```
<function Person.first_name at 0x1006a62e0>
>>>
```

fget fset property

attribute property
(Java)
getter setter

```
class Person:
    def __init__(self, first_name):
        self.first_name = first_name

    @property
    def first_name(self):
        return self._first_name

    @first_name.setter
    def first_name(self, value):
        self._first_name = value
```

property

attribute
property
attribute

Properties attribute attributes

```
import math
class Circle:
    def __init__(self, radius):
        self.radius = radius

    @property
    def area(self):
        return math.pi * self.radius ** 2

    @property
    def diameter(self):
        return self.radius ** 2

    @property
    def perimeter(self):
        return 2 * math.pi * self.radius
```

properties
attribute

```
>>> c = Circle(4.0)
>>> c.radius
4.0
>>> c.area  # Notice lack of ()
50.26548245743669
>>> c.perimeter  # Notice lack of ()
25.132741228718345
>>>
```

properties getter setter

```
>>> p = Person('Guido')
>>> p.get_first_name()
'Guido'
>>> p.set_first_name('Larry')
>>>
```

Python Python get/set ( ) property

property

```
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def first_name(self):
        return self._first_name

    @first_name.setter
    def first_name(self, value):
        if not isinstance(value, str):
            raise TypeError('Expected a string')
        self._first_name = value

    # Repeated property code, but for a different name (bad!)
    @property
    def last_name(self):
        return self._last_name

    @last_name.setter
    def last_name(self, value):
        if not isinstance(value, str):
            raise TypeError('Expected a string')
        self._last_name = value
```

8.9 9.21

10.7 8.7

10.7.1

10.7.2

( ) super()

```
class A:
    def spam(self):
        print('A.spam')

class B(A):
    def spam(self):
        print('B.spam')
        super().spam()  # Call parent spam()
```

super() __init__()

```
class A:
    def __init__(self):
        self.x = 0

class B(A):
    def __init__(self):
        super().__init__()
        self.y = 1
```

super() Python

```
class Proxy:
    def __init__(self, obj):
        self._obj = obj

    # Delegate attribute lookup to internal obj
    def __getattr__(self, name):
        return getattr(self._obj, name)

    # Delegate attribute assignment
    def __setattr__(self, name, value):
        if name.startswith('_'):
            super().__setattr__(name, value) # Call original __setattr__
        else:
            setattr(self._obj, name, value)
```

__setattr__()
(_) super() __setattr__()
self._obj
super()

10.7.3

Python super()

```
class Base:
    def __init__(self):
        print('Base.__init__')

class A(Base):
    def __init__(self):
        Base.__init__(self)
        print('A.__init__')
```

```
class Base:
    def __init__(self):
        print('Base.__init__')

class A(Base):
    def __init__(self):
        Base.__init__(self)
        print('A.__init__')

class B(Base):
    def __init__(self):
        Base.__init__(self)
        print('B.__init__')

class C(A,B):
    def __init__(self):
        A.__init__(self)
        B.__init__(self)
        print('C.__init__')
```

Base.__init__()

```
>>> c = C()
Base.__init__
A.__init__
Base.__init__
B.__init__
C.__init__
>>>
```

Base.__init__()
super()

```
class Base:
    def __init__(self):
        print('Base.__init__')

class A(Base):
    def __init__(self):
        super().__init__()
        print('A.__init__')

class B(Base):
    def __init__(self):
        super().__init__()
        print('B.__init__')

class C(A,B):
    def __init__(self):
        super().__init__()  # Only one call to super() here
        print('C.__init__')
```

__init__()

```
>>> c = C()
Base.__init__
B.__init__
A.__init__
C.__init__
>>>
```

Python

Python (MRO) MRO

```
>>> C.__mro__
(<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>,
<class '__main__.Base'>, <class 'object'>)
>>>
```

Python MRO

MRO C3

MRO

-
-
-

MRO

super() Python MRO
super()
MRO
Base.__init__()
super() MRO

```
class A:
    def spam(self):
        print('A.spam')
        super().spam()
```

```
>>> a = A()
>>> a.spam()
A.spam
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "<stdin>", line 4, in spam
AttributeError: 'super' object has no attribute 'spam'
>>>
```

```
>>> class B:
...     def spam(self):
...         print('B.spam')
...
>>> class C(A,B):
...     pass
...
>>> c = C()
>>> c.spam()
A.spam
B.spam
>>>
```

A super().spam() A
B spam() C MRO

```
>>> C.__mro__
(<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>,
<class 'object'>)
>>>
```

super() 8.13 8.18

super()

(
) super()
MRO

Python super()

Raymond Hettinger

“ Python ’ s super() Considered Super! ”

super()

10.8 8.8 property

10.8.1

property

10.8.2

property

```
class Person:
    def __init__(self, name):
        self.name = name

    # Getter function
    @property
    def name(self):
        return self._name

    # Setter function
    @name.setter
    def name(self, value):
        if not isinstance(value, str):
            raise TypeError('Expected a string')
        self._name = value

    # Deleter function
    @name.deleter
    def name(self):
        raise AttributeError("Can't delete attribute")
```

Person name

```
class SubPerson(Person):
    @property
    def name(self):
        print('Getting name')
        return super().name

    @name.setter
    def name(self, value):
        print('Setting name to', value)
        super(SubPerson, SubPerson).name.__set__(self, value)
```

```
    @name.deleter
    def name(self):
        print('Deleting name')
        super(SubPerson, SubPerson).name.__delete__(self)
```

```
>>> s = SubPerson('Guido')
Setting name to Guido
>>> s.name
Getting name
'Guido'
>>> s.name = 'Larry'
Setting name to Larry
>>> s.name = 42
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example.py", line 16, in name
        raise TypeError('Expected a string')
TypeError: Expected a string
>>>
```

property

```
class SubPerson(Person):
    @Person.name.getter
    def name(self):
        print('Getting name')
        return super().name
```

setter

```
class SubPerson(Person):
    @Person.name.setter
    def name(self, value):
        print('Setting name to', value)
        super(SubPerson, SubPerson).name.__set__(self, value)
```

10.8.3

property property
getter setter deleter
property

property
super() setter super(SubPerson, SubPerson).name.__set__(self, value)
setter name __set__()

super(SubPerson, SubPerson)

@property

```
class SubPerson(Person):
    @property  # Doesn't work
    def name(self):
        print('Getting name')
        return super().name
```

setter

```
>>> s = SubPerson('Guido')
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example.py", line 5, in __init__
        self.name = name
AttributeError: can't set attribute
>>>
```

```
class SubPerson(Person):
    @Person.getter
    def name(self):
        print('Getting name')
        return super().name
```

property getter

```
>>> s = SubPerson('Guido')
>>> s.name
Getting name
'Guido'
>>> s.name = 'Larry'
>>> s.name
Getting name
'Larry'
>>> s.name = 42
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example.py", line 16, in name
        raise TypeError('Expected a string')
TypeError: Expected a string
>>>
```

Person property

property super()

(8.9
)

```
# A descriptor
class String:
    def __init__(self, name):
        self.name = name

    def __get__(self, instance, cls):
        if instance is None:
            return self
        return instance.__dict__[self.name]

    def __set__(self, instance, value):
        if not isinstance(value, str):
            raise TypeError('Expected a string')
        instance.__dict__[self.name] = value

# A class with a descriptor
class Person:
    name = String('name')

    def __init__(self, name):
        self.name = name

# Extending a descriptor with a property
class SubPerson(Person):
    @property
    def name(self):
        print('Getting name')
        return super().name

    @name.setter
    def name(self, value):
        print('Setting name to', value)
        super(SubPerson, SubPerson).name.__set__(self, value)

    @name.deleter
    def name(self):
        print('Deleting name')
        super(SubPerson, SubPerson).name.__delete__(self)
```

setter deleter
Python issue
bug Python

10.9 8.9

10.9.1

10.9.2

```
# Descriptor attribute for an integer type-checked attribute
class Integer:
    def __init__(self, name):
        self.name = name

    def __get__(self, instance, cls):
        if instance is None:
            return self
        else:
            return instance.__dict__[self.name]

    def __set__(self, instance, value):
        if not isinstance(value, int):
            raise TypeError('Expected an int')
        instance.__dict__[self.name] = value

    def __delete__(self, instance):
        del instance.__dict__[self.name]
```

(get, set, delete)

__get__() __set__() __delete__()

```
class Point:
    x = Integer('x')
    y = Integer('y')

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

(x y) __get__() __set__()
__delete__()

```
>>> p = Point(2, 3)
>>> p.x # Calls Point.x.__get__(p,Point)
2
>>> p.y = 5 # Calls Point.y.__set__(p, 5)
>>> p.x = 2.3 # Calls Point.x.__set__(p, 2.3)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "descrip.py", line 12, in __set__
        raise TypeError('Expected an int')
TypeError: Expected an int
>>>
```

(__dict__) self.name
key

10.9.3

Python @classmethod
@staticmethod @property __slots__

(get, set, delete)

```
# Does NOT work
class Point:
    def __init__(self, x, y):
        self.x = Integer('x') # No! Must be a class variable
        self.y = Integer('y')
        self.x = x
        self.y = y
```

__get__()

```
# Descriptor attribute for an integer type-checked attribute
class Integer:

    def __get__(self, instance, cls):
        if instance is None:
            return self
        else:
            return instance.__dict__[self.name]
```

__get__()
instance None
( )

```
>>> p = Point(2,3)
>>> p.x # Calls Point.x.__get__(p, Point)
2
>>> Point.x # Calls Point.x.__get__(None, Point)
<__main__.Integer object at 0x100671890>
>>>
```

```
# Descriptor for a type-checked attribute
class Typed:
    def __init__(self, name, expected_type):
        self.name = name
        self.expected_type = expected_type
    def __get__(self, instance, cls):
        if instance is None:
            return self
        else:
            return instance.__dict__[self.name]

    def __set__(self, instance, value):
        if not isinstance(value, self.expected_type):
            raise TypeError('Expected ' + str(self.expected_type))
        instance.__dict__[self.name] = value
    def __delete__(self, instance):
        del instance.__dict__[self.name]

# Class decorator that applies it to selected attributes
def typeassert(**kwargs):
    def decorate(cls):
        for name, expected_type in kwargs.items():
            # Attach a Typed descriptor to the class
            setattr(cls, name, Typed(name, expected_type))
        return cls
    return decorate

# Example use
@typeassert(name=str, shares=int, price=float)
class Stock:
    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price
```

8.6 property
(
)

10.10 8.10

10.10.1

property

10.10.2

```
class lazyproperty:
    def __init__(self, func):
        self.func = func

    def __get__(self, instance, cls):
        if instance is None:
            return self
        else:
            value = self.func(instance)
            setattr(instance, self.func.__name__, value)
            return value
```

```
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    @lazyproperty
    def area(self):
        print('Computing area')
        return math.pi * self.radius ** 2

    @lazyproperty
    def perimeter(self):
        print('Computing perimeter')
        return 2 * math.pi * self.radius
```

```
>>> c = Circle(4.0)
>>> c.radius
4.0
>>> c.area
Computing area
50.26548245743669
```

```
>>> c.area
50.26548245743669
>>> c.perimeter
Computing perimeter
25.132741228718345
>>> c.perimeter
25.132741228718345
>>>
```

Computing area Computing perimeter

10.10.3

(8.9)
__get__() __set__() __delete__()
__get__()
__get__()

lazyproperty __get__()
property
property

```
>>> c = Circle(4.0)
>>> # Get instance variables
>>> vars(c)
{'radius': 4.0}

>>> # Compute area and observe variables afterward
>>> c.area
Computing area
50.26548245743669
>>> vars(c)
{'area': 50.26548245743669, 'radius': 4.0}

>>> # Notice access doesn't invoke property anymore
>>> c.area
50.26548245743669

>>> # Delete the variable and see property trigger again
>>> del c.area
>>> vars(c)
{'radius': 4.0}
>>> c.area
Computing area
50.26548245743669
>>>
```

```
>>> c.area
Computing area
50.26548245743669
>>> c.area = 25
>>> c.area
25
>>>
```

```
def lazyproperty(func):
    name = '_lazy_' + func.__name__
    @property
    def lazy(self):
        if hasattr(self, name):
            return getattr(self, name)
        else:
            value = func(self)
            setattr(self, name, value)
            return value
    return lazy
```

```
>>> c = Circle(4.0)
>>> c.area
Computing area
50.26548245743669
>>> c.area
50.26548245743669
>>> c.area = 25
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>>
```

get getter

property 8.6
8.9

10.11 8.11

10.11.1

__init__()

10.11.2

__init__()

```
import math

class Structure1:
    # Class variable that specifies expected fields
    _fields = []

    def __init__(self, *args):
        if len(args) != len(self._fields):
            raise TypeError('Expected {} arguments'.format(len(self._fields)))
        # Set the arguments
        for name, value in zip(self._fields, args):
            setattr(self, name, value)
```

:

```
# Example class definitions
class Stock(Structure1):
    _fields = ['name', 'shares', 'price']

class Point(Structure1):
    _fields = ['x', 'y']

class Circle(Structure1):
    _fields = ['radius']

    def area(self):
        return math.pi * self.radius ** 2
```

```
>>> s = Stock('ACME', 50, 91.1)
>>> p = Point(2, 3)
>>> c = Circle(4.5)
>>> s2 = Stock('ACME', 50)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "structure.py", line 6, in __init__
        raise TypeError('Expected {} arguments'.format(len(self._fields)))
TypeError: Expected 3 arguments
```

```
class Structure2:
    _fields = []

    def __init__(self, *args, **kwargs):
        if len(args) > len(self._fields):
            raise TypeError('Expected {} arguments'.format(len(self._fields)))
```

```
        # Set all of the positional arguments
        for name, value in zip(self._fields, args):
            setattr(self, name, value)

        # Set the remaining keyword arguments
        for name in self._fields[len(args):]:
            setattr(self, name, kwargs.pop(name))

        # Check for any remaining unknown arguments
        if kwargs:
            raise TypeError('Invalid argument(s): {}'.format(','.join(kwargs)))
# Example use
if __name__ == '__main__':
    class Stock(Structure2):
        _fields = ['name', 'shares', 'price']

    s1 = Stock('ACME', 50, 91.1)
    s2 = Stock('ACME', 50, price=91.1)
    s3 = Stock('ACME', shares=50, price=91.1)
    # s3 = Stock('ACME', shares=50, price=91.1, aa=1)
```

_fields

```
class Structure3:
    # Class variable that specifies expected fields
    _fields = []

    def __init__(self, *args, **kwargs):
        if len(args) != len(self._fields):
            raise TypeError('Expected {} arguments'.format(len(self._fields)))

        # Set the arguments
        for name, value in zip(self._fields, args):
            setattr(self, name, value)

        # Set the additional arguments (if any)
        extra_args = kwargs.keys() - self._fields
        for name in extra_args:
            setattr(self, name, kwargs.pop(name))

        if kwargs:
            raise TypeError('Duplicate values for {}'.format(','.join(kwargs)))

# Example use
if __name__ == '__main__':
    class Stock(Structure3):
        _fields = ['name', 'shares', 'price']

    s1 = Stock('ACME', 50, 91.1)
    s2 = Stock('ACME', 50, 91.1, date='8/2/2012')
```

10.11.3

__init__()

setattr()

```
class Structure:
    # Class variable that specifies expected fields
    _fields= []
    def __init__(self, *args):
        if len(args) != len(self._fields):
            raise TypeError('Expected {} arguments'.format(len(self._fields)))

        # Set the arguments (alternate)
        self.__dict__.update(zip(self._fields,args))
```

__slots__ property( )
setattr()

IDE

```
>>> help(Stock)
Help on class Stock in module __main__:
class Stock(Structure)
...
| Methods inherited from Structure:
|
| __init__(self, *args, **kwargs)
|
...
>>>
```

9.16 __init__()

10.12 8.12

10.12.1

10.12.2

abc

```
from abc import ABCMeta, abstractmethod

class IStream(metaclass=ABCMeta):
    @abstractmethod
    def read(self, maxbytes=-1):
        pass

    @abstractmethod
    def write(self, data):
        pass
```

```
a = IStream() # TypeError: Can't instantiate abstract class
              # IStream with abstract methods read, write
```

```
class SocketStream(IStream):
    def read(self, maxbytes=-1):
        pass

    def write(self, data):
        pass
```

```
def serialize(obj, stream):
    if not isinstance(stream, IStream):
        raise TypeError('Expected an IStream')
    pass
```

```
import io

# Register the built-in I/O classes as supporting our interface
IStream.register(io.IOBase)

# Open a normal file and type check
f = open('foo.txt')
isinstance(f, IStream) # Returns True
```

    @abstractmethod   properties

```
class A(metaclass=ABCMeta):
    @property
    @abstractmethod
    def name(self):
        pass
```

```
    @name.setter
    @abstractmethod
    def name(self, value):
        pass

    @classmethod
    @abstractmethod
    def method1(cls):
        pass

    @staticmethod
    @abstractmethod
    def method2():
        pass
```

10.12.3

collections
( ) numbers (
) io I/O

```
import collections

# Check if x is a sequence
if isinstance(x, collections.Sequence):
...

# Check if x is iterable
if isinstance(x, collections.Iterable):
...

# Check if x has a size
if isinstance(x, collections.Sized):
...

# Check if x is a mapping
if isinstance(x, collections.Mapping):
```

ABCs
Python

10.13 8.13

10.13.1

10.13.2

```
# Base class. Uses a descriptor to set a value
class Descriptor:
    def __init__(self, name=None, **opts):
        self.name = name
        for key, value in opts.items():
            setattr(self, key, value)

    def __set__(self, instance, value):
        instance.__dict__[self.name] = value


# Descriptor for enforcing types
class Typed(Descriptor):
    expected_type = type(None)

    def __set__(self, instance, value):
        if not isinstance(value, self.expected_type):
            raise TypeError('expected ' + str(self.expected_type))
        super().__set__(instance, value)


# Descriptor for enforcing values
class Unsigned(Descriptor):
    def __set__(self, instance, value):
        if value < 0:
            raise ValueError('Expected >= 0')
        super().__set__(instance, value)


class MaxSized(Descriptor):
    def __init__(self, name=None, **opts):
        if 'size' not in opts:
            raise TypeError('missing size option')
        super().__init__(name, **opts)

    def __set__(self, instance, value):
```

```
        if len(value) >= self.size:
            raise ValueError('size must be < ' + str(self.size))
        super().__set__(instance, value)
```

```
class Integer(Typed):
    expected_type = int

class UnsignedInteger(Integer, Unsigned):
    pass

class Float(Typed):
    expected_type = float

class UnsignedFloat(Float, Unsigned):
    pass

class String(Typed):
    expected_type = str

class SizedString(String, MaxSized):
    pass
```

```
class Stock:
    # Specify constraints
    name = SizedString('name', size=8)
    shares = UnsignedInteger('shares')
    price = UnsignedFloat('price')

    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price
```

```
>>> s.name
'ACME'
>>> s.shares = 75
>>> s.shares = -10
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example.py", line 17, in __set__
        super().__set__(instance, value)
    File "example.py", line 23, in __set__
        raise ValueError('Expected >= 0')
ValueError: Expected >= 0
```

```
>>> s.price = 'a lot'
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example.py", line 16, in __set__
        raise TypeError('expected ' + str(self.expected_type))
TypeError: expected <class 'float'>
>>> s.name = 'ABRACADABRA'
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example.py", line 17, in __set__
        super().__set__(instance, value)
    File "example.py", line 35, in __set__
        raise ValueError('size must be < ' + str(self.size))
ValueError: size must be < 8
>>>
```

```
# Class decorator to apply constraints
def check_attributes(**kwargs):
    def decorate(cls):
        for key, value in kwargs.items():
            if isinstance(value, Descriptor):
                value.name = key
                setattr(cls, key, value)
            else:
                setattr(cls, key, value(key))
        return cls

    return decorate

# Example
@check_attributes(name=SizedString(size=8),
                  shares=UnsignedInteger,
                  price=UnsignedFloat)
class Stock:
    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price
```

```
# A metaclass that applies checking
class checkedmeta(type):
    def __new__(cls, clsname, bases, methods):
        # Attach attribute names to the descriptors
        for key, value in methods.items():
            if isinstance(value, Descriptor):
                value.name = key
        return type.__new__(cls, clsname, bases, methods)
```

```
# Example  ss
class Stock2(metaclass=checkedmeta):      /      .
    name = SizedString(size=8)
    shares = UnsignedInteger()
    price = UnsignedFloat()

    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price
```

10.13.3

super()
8.9 8.18 9.19

Descriptor __set__() __get__()

__get__()

Unsigned MaxSized

Typed

super() *

```
    def __set__(self, instance, value):
        if not isinstance(value, expected_type):
            raise TypeError('expected ' + str(expected_type))
        super_set(self, instance, value)

    cls.__set__ = __set__
    return cls


# Decorator for unsigned values
def Unsigned(cls):
    super_set = cls.__set__

    def __set__(self, instance, value):
        if value < 0:
            raise ValueError('Expected >= 0')
        super_set(self, instance, value)

    cls.__set__ = __set__
    return cls


# Decorator for allowing sized values
def MaxSized(cls):
    super_init = cls.__init__

    def __init__(self, name=None, **opts):
        if 'size' not in opts:
            raise TypeError('missing size option')
        super_init(self, name, **opts)

    cls.__init__ = __init__

    super_set = cls.__set__

    def __set__(self, instance, value):
        if len(value) >= self.size:
            raise ValueError('size must be < ' + str(self.size))
        super_set(self, instance, value)

    cls.__set__ = __set__
    return cls


# Specialized descriptors
@Typed(int)
class Integer(Descriptor):
    pass
```

```
@Unsigned
class UnsignedInteger(Integer):
    pass


@Typed(float)
class Float(Descriptor):
    pass


@Unsigned
class UnsignedFloat(Float):
    pass


@Typed(str)
class String(Descriptor):
    pass


@MaxSized
class SizedString(String):
    pass
```

100%

^_^

10.14 8.14

10.14.1

10.14.2

collections

collections.Iterable

```
import collections
class A(collections.Iterable):
    pass
```

collections.Iterable :

```
>>> a = A()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class A with abstract methods __iter__
>>>
```

__iter__() (4.2 4.7)

```
>>> import collections
>>> collections.Sequence()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class Sequence with abstract methods \
__getitem__, __len__
>>>
```

Sequence

```
class SortedItems(collections.Sequence):
    def __init__(self, initial=None):
        self._items = sorted(initial) if initial is not None else []

    # Required sequence methods
    def __getitem__(self, index):
        return self._items[index]

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

    # Method for adding an item in the right location
    def add(self, item):
        bisect.insort(self._items, item)


items = SortedItems([5, 1, 3])
print(list(items))
print(items[0], items[-1])
items.add(2)
print(list(items))
```

SortedItems

bisect

10.14.3

collections

```
>>> items = SortedItems()
>>> import collections
>>> isinstance(items, collections.Iterable)
True
>>> isinstance(items, collections.Sequence)
True
>>> isinstance(items, collections.Container)
True
>>> isinstance(items, collections.Sized)
True
>>> isinstance(items, collections.Mapping)
False
>>>
```

collections

collections.MutableSequence

```
class Items(collections.MutableSequence):
    def __init__(self, initial=None):
        self._items = list(initial) if initial is not None else []

    # Required sequence methods
    def __getitem__(self, index):
        print('Getting:', index)
        return self._items[index]

    def __setitem__(self, index, value):
        print('Setting:', index, value)
        self._items[index] = value

    def __delitem__(self, index):
        print('Deleting:', index)
        del self._items[index]

    def insert(self, index, value):
        print('Inserting:', index, value)
        self._items.insert(index, value)

    def __len__(self):
        print('Len')
        return len(self._items)
```

Items (append() remove() count())

```
>>> a = Items([1, 2, 3])
>>> len(a)
Len
3
>>> a.append(4)
Len
Inserting: 3 4
>>> a.append(2)
Len
Inserting: 4 2
>>> a.count(2)
Getting: 0
Getting: 1
Getting: 2
Getting: 3
Getting: 4
Getting: 5
2
>>> a.remove(3)
Getting: 0
Getting: 1
Getting: 2
Deleting: 2
>>>
```

Python numbers
8.12

10.15 8.15

10.15.1

10.15.2

```
class A:
    def spam(self, x):
        pass

    def foo(self):
        pass
```

```
class B1:
    """          """

    def __init__(self):
        self._a = A()

    def spam(self, x):
        # Delegate to the internal self._a instance
        return self._a.spam(x)

    def foo(self):
        # Delegate to the internal self._a instance
        return self._a.foo()

    def bar(self):
        pass
```

__getattr__()

```
class B2:
    """     __getattr__                          """

    def __init__(self):
        self._a = A()

    def bar(self):
        pass

    # Expose all of the methods defined on class A
    def __getattr__(self, name):
        """                attribute
        the __getattr__() method is actually a fallback method
        that only gets called when an attribute is not found"""
        return getattr(self._a, name)
```

__getattr__ attribute

```
b = B()
b.bar() # Calls B.bar() (exists on B)
b.spam(42) # Calls B.__getattr__('spam') and delegates to A.spam
```

```
# A proxy class that wraps around another object, but
# exposes its public attributes
class Proxy:
    def __init__(self, obj):
        self._obj = obj

    # Delegate attribute lookup to internal obj
```

```
    def __getattr__(self, name):
        print('getattr:', name)
        return getattr(self._obj, name)

    # Delegate attribute assignment
    def __setattr__(self, name, value):
        if name.startswith('_'):
            super().__setattr__(name, value)
        else:
            print('setattr:', name, value)
            setattr(self._obj, name, value)

    # Delegate attribute deletion
    def __delattr__(self, name):
        if name.startswith('_'):
            super().__delattr__(name)
        else:
            print('delattr:', name)
            delattr(self._obj, name)
```

```
class Spam:
    def __init__(self, x):
        self.x = x

    def bar(self, y):
        print('Spam.bar:', self.x, y)

# Create an instance
s = Spam(2)
# Create a proxy around it
p = Proxy(s)
# Access the proxy
print(p.x)  # Outputs 2
p.bar(3)  # Outputs "Spam.bar: 2 3"
p.x = 37  # Changes s.x to 37
```

( )

10.15.3

```
class A:
    def spam(self, x):
        print('A.spam', x)
    def foo(self):
        print('A.foo')
```

```
class B(A):
    def spam(self, x):
        print('B.spam')
        super().spam(x)
    def bar(self):
        print('B.bar')
```

```
class A:
    def spam(self, x):
        print('A.spam', x)
    def foo(self):
        print('A.foo')

class B:
    def __init__(self):
        self._a = A()
    def spam(self, x):
        print('B.spam', x)
        self._a.spam(x)
    def bar(self):
        print('B.bar')
    def __getattr__(self, name):
        return getattr(self._a, name)
```

__getattr__()

__setattr__() __delattr__()

_obj _

( )

__getattr__() (__)

```
class ListLike:
    """__getattr__ """

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

    def __getattr__(self, name):
        return getattr(self._items, name)
```

ListLike append()
insert() len()

```
>>> a = ListLike()
>>> a.append(2)
>>> a.insert(0, 1)
>>> a.sort()
```

```
>>> len(a)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: object of type 'ListLike' has no len()
>>> a[0]
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: 'ListLike' object does not support indexing
>>>
```

```
class ListLike:
    """__getattr__                                                    """

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

    def __getattr__(self, name):
        return getattr(self._items, name)

    # Added special methods to support certain list operations
    def __len__(self):
        return len(self._items)

    def __getitem__(self, index):
        return self._items[index]

    def __setitem__(self, index, value):
        self._items[index] = value

    def __delitem__(self, index):
        del self._items[index]
```

11.8

10.16 8.16

10.16.1

__init__()

10.16.2

```
import time

class Date:
    """                  """
    # Primary constructor
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day

    # Alternate constructor
    @classmethod
    def today(cls):
        t = time.localtime()
        return cls(t.tm_year, t.tm_mon, t.tm_mday)
```

```
a = Date(2012, 12, 21) # Primary
b = Date.today() # Alternate
```

10.16.3

class

(cls)

```
class NewDate(Date):
    pass

c = Date.today() # Creates an instance of Date (cls=Date)
d = NewDate.today() # Creates an instance of NewDate (cls=NewDate)
```

10.17 8.17 init

10.17.1

__init__()

10.17.2

__new__()

```
class Date:
    def __init__(self, year, month, day):
```

```
        self.year = year
        self.month = month
        self.day = day
```

__init__() Date

```
>>> d = Date.__new__(Date)
>>> d
<__main__.Date object at 0x1006716d0>
>>> d.year
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: 'Date' object has no attribute 'year'
>>>
```

Date year

```
>>> data = {'year':2012, 'month':8, 'day':29}
>>> for key, value in data.items():
...     setattr(d, key, value)
...
>>> d.year
2012
>>> d.month
8
>>>
```

10.17.3

__init__()

Date

today()

```
from time import localtime

class Date:
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day

    @classmethod
    def today(cls):
        d = cls.__new__(cls)
        t = localtime()
        d.year = t.tm_year
        d.month = t.tm_mon
        d.day = t.tm_mday
        return d
```

JSON

```
data = { 'year': 2012, 'month': 8, 'day': 29 }
```

Date

__slots__ properties
descriptors setattr()

10.18 8.18 Mixins

10.18.1

10.18.2

```
class LoggedMappingMixin:
    """
    Add logging to get/set/delete operations for debugging.
    """
    __slots__ = ()  #

    def __getitem__(self, key):
        print('Getting ' + str(key))
        return super().__getitem__(key)

    def __setitem__(self, key, value):
        print('Setting {} = {!r}'.format(key, value))
        return super().__setitem__(key, value)

    def __delitem__(self, key):
        print('Deleting ' + str(key))
        return super().__delitem__(key)


class SetOnceMappingMixin:
    '''
    Only allow a key to be set once.
    '''
```

```
    __slots__ = ()

    def __setitem__(self, key, value):
        if key in self:
            raise KeyError(str(key) + ' already set')
        return super().__setitem__(key, value)


class StringKeysMappingMixin:
    '''
    Restrict keys to strings only
    '''
    __slots__ = ()

    def __setitem__(self, key, value):
        if not isinstance(key, str):
            raise TypeError('keys must be strings')
        return super().__setitem__(key, value)
```

```
class LoggedDict(LoggedMappingMixin, dict):
    pass

d = LoggedDict()
d['x'] = 23
print(d['x'])
del d['x']

from collections import defaultdict

class SetOnceDefaultDict(SetOnceMappingMixin, defaultdict):
    pass


d = SetOnceDefaultDict(list)
d['x'].append(2)
d['x'].append(3)
# d['x'] = 23  # KeyError: 'x already set'
```

(dict defaultdict OrderedDict)

10.18.3

socketserver ThreadingMixIn

XML-RPC

```
from xmlrpc.server import SimpleXMLRPCServer
from socketserver import ThreadingMixIn
class ThreadedXMLRPCServer(ThreadingMixIn, SimpleXMLRPCServer):
    pass
```

__init__()
__slots__ = ()

```
def LoggedMapping(cls):
    """                         """
    cls_getitem = cls.__getitem__
    cls_setitem = cls.__setitem__
    cls_delitem = cls.__delitem__

    def __getitem__(self, key):
        print('Getting ' + str(key))
        return cls_getitem(self, key)

    def __setitem__(self, key, value):
        print('Setting {} = {!r}'.format(key, value))
        return cls_setitem(self, key, value)

    def __delitem__(self, key):
        print('Deleting ' + str(key))
        return cls_delitem(self, key)

    cls.__getitem__ = __getitem__
    cls.__setitem__ = __setitem__
    cls.__delitem__ = __delitem__
    return cls


@LoggedMapping
class LoggedDict(dict):
    pass
```

9.12

8.13

10.19 8.19

10.19.1

10.19.2

```
class Connection:
    """                                  ~~"""

    def __init__(self):
        self.state = 'CLOSED'

    def read(self):
        if self.state != 'OPEN':
            raise RuntimeError('Not open')
        print('reading')

    def write(self, data):
        if self.state != 'OPEN':
            raise RuntimeError('Not open')
        print('writing')

    def open(self):
        if self.state == 'OPEN':
            raise RuntimeError('Already open')
        self.state = 'OPEN'

    def close(self):
        if self.state == 'CLOSED':
            raise RuntimeError('Already closed')
        self.state = 'CLOSED'
```

read() write()

```
class Connection1:
    """                              """

    def __init__(self):
        self.new_state(ClosedConnectionState)
```

```
    def new_state(self, newstate):
        self._state = newstate
        # Delegate to the state class

    def read(self):
        return self._state.read(self)

    def write(self, data):
        return self._state.write(self, data)

    def open(self):
        return self._state.open(self)

    def close(self):
        return self._state.close(self)


# Connection state base class
class ConnectionState:
    @staticmethod
    def read(conn):
        raise NotImplementedError()

    @staticmethod
    def write(conn, data):
        raise NotImplementedError()

    @staticmethod
    def open(conn):
        raise NotImplementedError()

    @staticmethod
    def close(conn):
        raise NotImplementedError()


# Implementation of different states
class ClosedConnectionState(ConnectionState):
    @staticmethod
    def read(conn):
        raise RuntimeError('Not open')

    @staticmethod
    def write(conn, data):
        raise RuntimeError('Not open')

    @staticmethod
    def open(conn):
        conn.new_state(OpenConnectionState)

    @staticmethod
```

```
    def close(conn):
        raise RuntimeError('Already closed')


class OpenConnectionState(ConnectionState):
    @staticmethod
    def read(conn):
        print('reading')

    @staticmethod
    def write(conn, data):
        print('writing')

    @staticmethod
    def open(conn):
        raise RuntimeError('Already open')

    @staticmethod
    def close(conn):
        conn.new_state(ClosedConnectionState)
```

```
>>> c = Connection()
>>> c._state
<class '__main__.ClosedConnectionState'>
>>> c.read()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example.py", line 10, in read
        return self._state.read(self)
    File "example.py", line 43, in read
        raise RuntimeError('Not open')
RuntimeError: Not open
>>> c.open()
>>> c._state
<class '__main__.OpenConnectionState'>
>>> c.read()
reading
>>> c.write('hello')
writing
>>> c.close()
>>> c._state
<class '__main__.ClosedConnectionState'>
>>>
```

10.19.3

Connection

NotImplementedError 8.12

10.20 8.20

10.20.1

10.20.2

getattr()

```
import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Point({!r:},{!r:})'.format(self.x, self.y)

    def distance(self, x, y):
        return math.hypot(self.x - x, self.y - y)


p = Point(2, 3)
d = getattr(p, 'distance')(0, 0)  # Calls p.distance(0, 0)
```

operator.methodcaller()

```
import operator
operator.methodcaller('distance', 0, 0)(p)
```

operator.methodcaller

```
points = [
    Point(1, 2),
    Point(3, 0),
    Point(10, -3),
    Point(-5, -7),
    Point(-1, 8),
```

```
    Point(3, 2)
]
# Sort by distance from origin (0, 0)
points.sort(key=operator.methodcaller('distance', 0, 0))
```

10.20.3

getattr()

operator.methodcaller()

```
>>> p = Point(3, 4)
>>> d = operator.methodcaller('distance', 0, 0)
>>> d(p)
5.0
>>>
```

case

10.21 8.21

10.21.1

10.21.2

```
class Node:
    pass

class UnaryOperator(Node):
    def __init__(self, operand):
        self.operand = operand

class BinaryOperator(Node):
    def __init__(self, left, right):
        self.left = left
```

```
        self.right = right

class Add(BinaryOperator):
    pass

class Sub(BinaryOperator):
    pass

class Mul(BinaryOperator):
    pass

class Div(BinaryOperator):
    pass

class Negate(UnaryOperator):
    pass

class Number(Node):
    def __init__(self, value):
        self.value = value
```

```
# Representation of 1 + 2 * (3 - 4) / 5
t1 = Sub(Number(3), Number(4))
t2 = Mul(Number(2), t1)
t3 = Div(t2, Number(5))
t4 = Add(Number(1), t3)
```

```
class NodeVisitor:
    def visit(self, node):
        methname = 'visit_' + type(node).__name__
        meth = getattr(self, methname, None)
        if meth is None:
            meth = self.generic_visit
        return meth(node)

    def generic_visit(self, node):
        raise RuntimeError('No {} method'.format('visit_' + type(node).__name__))
```

visit_Name()

Name node

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

    def visit_Add(self, node):
        return self.visit(node.left) + self.visit(node.right)
```

```
    def visit_Sub(self, node):
        return self.visit(node.left) - self.visit(node.right)

    def visit_Mul(self, node):
        return self.visit(node.left) * self.visit(node.right)

    def visit_Div(self, node):
        return self.visit(node.left) / self.visit(node.right)

    def visit_Negate(self, node):
        return -node.operand
```

```
>>> e = Evaluator()
>>> e.visit(t4)
0.6
>>>
```

```
class StackCode(NodeVisitor):
    def generate_code(self, node):
        self.instructions = []
        self.visit(node)
        return self.instructions

    def visit_Number(self, node):
        self.instructions.append(('PUSH', node.value))

    def binop(self, node, instruction):
        self.visit(node.left)
        self.visit(node.right)
        self.instructions.append((instruction,))

    def visit_Add(self, node):
        self.binop(node, 'ADD')

    def visit_Sub(self, node):
        self.binop(node, 'SUB')

    def visit_Mul(self, node):
        self.binop(node, 'MUL')

    def visit_Div(self, node):
        self.binop(node, 'DIV')

    def unaryop(self, node, instruction):
        self.visit(node.operand)
        self.instructions.append((instruction,))
```

```
    def visit_Negate(self, node):
        self.unaryop(node, 'NEG')
```

```
>>> s = StackCode()
>>> s.generate_code(t4)
[('PUSH', 1), ('PUSH', 2), ('PUSH', 3), ('PUSH', 4), ('SUB',),
('MUL',), ('PUSH', 5), ('DIV',), ('ADD',)]
>>>
```

10.21.3

if/else

getattr()

```
def binop(self, node, instruction):
    self.visit(node.left)
    self.visit(node.right)
    self.instructions.append((instruction,))
```

switch case

HTTP

```
class HTTPHandler:
    def handle(self, request):
        methname = 'do_' + request.request_method
        getattr(self, methname)(request)
    def do_GET(self, request):
        pass
    def do_POST(self, request):
        pass
    def do_HEAD(self, request):
        pass
```

Python (sys.getrecursionlimit())

8.22

Python ast

9.24 ast Python

10.22 8.22

10.22.1

10.22.2

8.21

```
import types

class Node:
    pass

class NodeVisitor:
    def visit(self, node):
        stack = [node]
        last_result = None
        while stack:
            try:
                last = stack[-1]
                if isinstance(last, types.GeneratorType):
                    stack.append(last.send(last_result))
                    last_result = None
                elif isinstance(last, Node):
                    stack.append(self._visit(stack.pop()))
                else:
                    last_result = stack.pop()
            except StopIteration:
                stack.pop()

        return last_result

    def _visit(self, node):
        methname = 'visit_' + type(node).__name__
        meth = getattr(self, methname, None)
        if meth is None:
            meth = self.generic_visit
        return meth(node)

    def generic_visit(self, node):
        raise RuntimeError('No {} method'.format('visit_' + type(node).__name__))
```

```
class UnaryOperator(Node):
    def __init__(self, operand):
        self.operand = operand

class BinaryOperator(Node):
    def __init__(self, left, right):
        self.left = left
        self.right = right

class Add(BinaryOperator):
    pass

class Sub(BinaryOperator):
    pass

class Mul(BinaryOperator):
    pass

class Div(BinaryOperator):
    pass

class Negate(UnaryOperator):
    pass

class Number(Node):
    def __init__(self, value):
        self.value = value

# A sample visitor class that evaluates expressions
class Evaluator(NodeVisitor):
    def visit_Number(self, node):
        return node.value

    def visit_Add(self, node):
        return self.visit(node.left) + self.visit(node.right)

    def visit_Sub(self, node):
        return self.visit(node.left) - self.visit(node.right)

    def visit_Mul(self, node):
        return self.visit(node.left) * self.visit(node.right)

    def visit_Div(self, node):
        return self.visit(node.left) / self.visit(node.right)

    def visit_Negate(self, node):
        return -self.visit(node.operand)

if __name__ == '__main__':
    # 1 + 2*(3-4) / 5
    t1 = Sub(Number(3), Number(4))
```

```
    t2 = Mul(Number(2), t1)
    t3 = Div(t2, Number(5))
    t4 = Add(Number(1), t3)
    # Evaluate it
    e = Evaluator()
    print(e.visit(t4))  # Outputs 0.6
```

Evaluator

```
>>> a = Number(0)
>>> for n in range(1, 100000):
... a = Add(a, Number(n))
...
>>> e = Evaluator()
>>> e.visit(a)
Traceback (most recent call last):
...
    File "visitor.py", line 29, in _visit
return meth(node)
    File "visitor.py", line 67, in visit_Add
return self.visit(node.left) + self.visit(node.right)
RuntimeError: maximum recursion depth exceeded
>>>
```

Evaluator

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

    def visit_Add(self, node):
        yield (yield node.left) + (yield node.right)

    def visit_Sub(self, node):
        yield (yield node.left) - (yield node.right)

    def visit_Mul(self, node):
        yield (yield node.left) * (yield node.right)

    def visit_Div(self, node):
        yield (yield node.left) / (yield node.right)

    def visit_Negate(self, node):
        yield - (yield node.operand)
```

```
>>> a = Number(0)
>>> for n in range(1,100000):
...     a = Add(a, Number(n))
...
>>> e = Evaluator()
```

```
>>> e.visit(a)
4999950000
>>>
```

```
class Evaluator(NodeVisitor):
    ...
    def visit_Add(self, node):
        print('Add:', node)
        lhs = yield node.left
        print('left=', lhs)
        rhs = yield node.right
        print('right=', rhs)
        yield lhs + rhs
    ...
```

```
>>> e = Evaluator()
>>> e.visit(t4)
Add: <__main__.Add object at 0x1006a8d90>
left= 1
right= -0.4
0.6
>>>
```

10.22.3

`visit()`

yield yield

```
value = self.visit(node.left)
```

yield

```
value = yield node.left
```

`node.left` `visit()` `visit()` `visit_Name()` yield value

yield

yield

10.23 8.23

10.23.1

( )

10.23.2

weakref

```
import weakref

class Node:
    def __init__(self, value):
        self.value = value
        self._parent = None
        self.children = []

    def __repr__(self):
        return 'Node({!r:})'.format(self.value)

    # property that manages the parent as a weak-reference
    @property
    def parent(self):
        return None if self._parent is None else self._parent()

    @parent.setter
    def parent(self, node):
        self._parent = weakref.ref(node)

    def add_child(self, child):
        self.children.append(child)
        child.parent = self
```

parent

```
>>> root = Node('parent')
>>> c1 = Node('child')
>>> root.add_child(c1)
>>> print(c1.parent)
Node('parent')
```

```
>>> del root
>>> print(c1.parent)
None
>>>
```

10.23.3

Python

```
# Class just to illustrate when deletion occurs
class Data:
    def __del__(self):
        print('Data.__del__')

# Node class involving a cycle
class Node:
    def __init__(self):
        self.data = Data()
        self.parent = None
        self.children = []

    def add_child(self, child):
        self.children.append(child)
        child.parent = self
```

```
>>> a = Data()
>>> del a # Immediately deleted
Data.__del__
>>> a = Node()
>>> del a # Immediately deleted
Data.__del__
>>> a = Node()
>>> a.add_child(Node())
>>> del a # Not deleted (no message)
>>>
```

Python

0

0

Python

```
>>> import gc
>>> gc.collect() # Force collection
Data.__del__
```

```
Data.__del__
>>>
```

__del__()

Node __del__()

```
# Node class involving a cycle
class Node:
    def __init__(self):
        self.data = Data()
        self.parent = None
        self.children = []

    def add_child(self, child):
        self.children.append(child)
        child.parent = self

    # NEVER DEFINE LIKE THIS.
    # Only here to illustrate pathological behavior
    def __del__(self):
        del self.data
        del.parent
        del.children
```

Data.__del__ ,

```
>>> a = Node()
>>> a.add_child(Node()
>>> del a # No message (not collected)
>>> import gc
>>> gc.collect() # No message (not collected)
>>>
```

weakref

```
>>> import weakref
>>> a = Node()
>>> a_ref = weakref.ref(a)
>>> a_ref
<weakref at 0x100581f70; to 'Node' at 0x1005c5410>
>>>
```

None

;

```
>>> print(a_ref())
<__main__.Node object at 0x1005c5410>
>>> del a
Data.__del__
>>> print(a_ref())
```

```
None
>>>
```

8.25

10.24 8.24

10.24.1

(>=,!=,<=,<)

10.24.2

Python >=
__ge__()

functools.total_ordering
__eq__() (__lt__, __le__, __gt__, or __ge__)

```
from functools import total_ordering

class Room:
    def __init__(self, name, length, width):
        self.name = name
        self.length = length
        self.width = width
        self.square_feet = self.length * self.width

@total_ordering
class House:
    def __init__(self, name, style):
        self.name = name
        self.style = style
        self.rooms = list()

    @property
    def living_space_footage(self):
        return sum(r.square_feet for r in self.rooms)

    def add_room(self, room):
```

```
        self.rooms.append(room)

    def __str__(self):
        return '{}: {} square foot {}'.format(self.name,
                self.living_space_footage,
                self.style)

    def __eq__(self, other):
        return self.living_space_footage == other.living_space_footage

    def __lt__(self, other):
        return self.living_space_footage < other.living_space_footage
```

House __eq__() __lt__()

```
# Build a few houses, and add rooms to them
h1 = House('h1', 'Cape')
h1.add_room(Room('Master Bedroom', 14, 21))
h1.add_room(Room('Living Room', 18, 20))
h1.add_room(Room('Kitchen', 12, 16))
h1.add_room(Room('Office', 12, 12))
h2 = House('h2', 'Ranch')
h2.add_room(Room('Master Bedroom', 14, 21))
h2.add_room(Room('Living Room', 18, 20))
h2.add_room(Room('Kitchen', 12, 16))
h3 = House('h3', 'Split')
h3.add_room(Room('Master Bedroom', 14, 21))
h3.add_room(Room('Living Room', 18, 20))
h3.add_room(Room('Office', 12, 16))
h3.add_room(Room('Kitchen', 15, 17))
houses = [h1, h2, h3]
print('Is h1 bigger than h2?', h1 > h2) # prints True
print('Is h2 smaller than h3?', h2 < h3) # prints True
print('Is h2 greater than or equal to h1?', h2 >= h1) # Prints False
print('Which one is biggest?', max(houses)) # Prints 'h3: 1101-square-foot Split'
print('Which is smallest?', min(houses)) # Prints 'h2: 846-square-foot Ranch'
```

10.24.3

total_ordering

__le__()

```
class House:
    def __eq__(self, other):
        pass
    def __lt__(self, other):
```

```
        pass
    # Methods created by @total_ordering
    __le__ = lambda self, other: self < other or self == other
    __gt__ = lambda self, other: not (self < other or self == other)
    __ge__ = lambda self, other: not (self < other)
    __ne__ = lambda self, other: not self == other
```

@total_ordering

10.25 8.25

10.25.1

10.25.2

logging logger

```
>>> import logging
>>> a = logging.getLogger('foo')
>>> b = logging.getLogger('bar')
>>> a is b
False
>>> c = logging.getLogger('foo')
>>> a is c
True
>>>
```

```
# The class in question
class Spam:
    def __init__(self, name):
        self.name = name

# Caching support
import weakref
_spam_cache = weakref.WeakValueDictionary()
def get_spam(name):
    if name not in _spam_cache:
        s = Spam(name)
        _spam_cache[name] = s
    else:
```

```
        s = _spam_cache[name]
    return s
```

```
>>> a = get_spam('foo')
>>> b = get_spam('bar')
>>> a is b
False
>>> c = get_spam('foo')
>>> a is c
True
>>>
```

10.25.3

__new__()

```
# Note: This code doesn't quite work
import weakref

class Spam:
    _spam_cache = weakref.WeakValueDictionary()
    def __new__(cls, name):
        if name in cls._spam_cache:
            return cls._spam_cache[name]
        else:
            self = super().__new__(cls)
            cls._spam_cache[name] = self
            return self
    def __init__(self, name):
        print('Initializing Spam')
        self.name = name
```

__init__()

```
>>> s = Spam('Dave')
Initializing Spam
>>> t = Spam('Dave')
Initializing Spam
>>> s is t
True
>>>
```

8.23

WeakValueDictionary

```
>>> a = get_spam('foo')
>>> b = get_spam('bar')
>>> c = get_spam('foo')
>>> list(_spam_cache)
['foo', 'bar']
>>> del a
>>> del c
>>> list(_spam_cache)
['bar']
>>> del b
>>> list(_spam_cache)
[]
>>>
```

```
import weakref

class CachedSpamManager:
    def __init__(self):
        self._cache = weakref.WeakValueDictionary()

    def get_spam(self, name):
        if name not in self._cache:
            s = Spam(name)
            self._cache[name] = s
        else:
            s = self._cache[name]
        return s

    def clear(self):
            self._cache.clear()

class Spam:
    manager = CachedSpamManager()
    def __init__(self, name):
        self.name = name

    def get_spam(name):
        return Spam.manager.get_spam(name)
```

manager

```
>>> a = Spam('foo')
>>> b = Spam('foo')
>>> a is b
False
>>>
```

(_)

__init__()

```
class Spam:
    def __init__(self, *args, **kwargs):
        raise RuntimeError("Can't instantiate directly")

    # Alternate constructor
    @classmethod
    def _new(cls, name):
        self = cls.__new__(cls)
        self.name = name
```

Spam._new() Spam()

```
# ------------------------              ------------------------
class CachedSpamManager2:
    def __init__(self):
        self._cache = weakref.WeakValueDictionary()

    def get_spam(self, name):
        if name not in self._cache:
            temp = Spam3._new(name)  # Modified creation
            self._cache[name] = temp
        else:
            temp = self._cache[name]
        return temp

    def clear(self):
            self._cache.clear()

class Spam3:
    def __init__(self, *args, **kwargs):
        raise RuntimeError("Can't instantiate directly")

    # Alternate constructor
    @classmethod
    def _new(cls, name):
        self = cls.__new__(cls)
        self.name = name
        return self
```

9.13

( )

CHAPTER

ELEVEN

“ don ’ t repeat yourself ”
( )
Python
( )

`exec()`

Contents:

11.1 9.1

11.1.1

( )

11.1.2

```
import time
from functools import wraps

def timethis(func):
    '''
    Decorator that reports the execution time.
    '''
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(func.__name__, end-start)
        return result
    return wrapper
```

```
>>> @timethis
... def countdown(n):
...     '''
...     Counts down
...     '''
...     while n > 0:
...         n -= 1
...
>>> countdown(100000)
countdown 0.008917808532714844
>>> countdown(10000000)
countdown 0.87188299392912
>>>
```

11.1.3

```
@timethis
def countdown(n):
    pass
```

```
def countdown(n):
    pass
countdown = timethis(countdown)
```

@staticmethod, @classmethod,@property

```
class A:
    @classmethod
    def method(cls):
        pass

class B:
    # Equivalent definition of a class method
    def method(cls):
        pass
    method = classmethod(method)
```

wrapper() *args **kwargs

( )

*args
**kwargs
func(*args, **kwargs) func

@wraps(func)
( )

11.2 9.2

11.2.1

11.2.2

functools @wraps

```
import time
from functools import wraps
def timethis(func):
    '''
    Decorator that reports the execution time.
    '''
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(func.__name__, end-start)
        return result
    return wrapper
```

```
>>> @timethis
... def countdown(n:int):
...     '''
...     Counts down
...     '''
...     while n > 0:
...         n -= 1
...
>>> countdown(100000)
```

```
countdown 0.008917808532714844
>>> countdown.__name__
'countdown'
>>> countdown.__doc__
'\n\tCounts down\n\t'
>>> countdown.__annotations__
{'n': <class 'int'>}
>>>
```

11.2.3

@wraps @wraps

```
>>> countdown.__name__
'wrapper'
>>> countdown.__doc__
>>> countdown.__annotations__
{}
>>>
```

@wraps __wrapped__
:

```
>>> countdown.__wrapped__(100000)
>>>
```

__wrapped__

```
>>> from inspect import signature
>>> print(signature(countdown))
(n:int)
>>>
```

__wrapped__
__wrapped__ 9.16

11.3 9.3

11.3.1

11.3.2

@wraps (9.2)
__wrapped__

```
>>> @somedecorator
>>> def add(x, y):
...     return x + y
...
>>> orig_add = add.__wrapped__
>>> orig_add(3, 4)
7
>>>
```

11.3.3

@wraps __wrapped__

__wrapped__
Python3.3

```
from functools import wraps

def decorator1(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        print('Decorator 1')
        return func(*args, **kwargs)
    return wrapper

def decorator2(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        print('Decorator 2')
        return func(*args, **kwargs)
    return wrapper

@decorator1
@decorator2
def add(x, y):
    return x + y
```

Python3.3

```
>>> add(2, 3)
Decorator 1
Decorator 2
5
```

```
>>> add.__wrapped__(2, 3)
5
>>>
```

Python3.4

```
>>> add(2, 3)
Decorator 1
Decorator 2
5
>>> add.__wrapped__(2, 3)
Decorator 2
5
>>>
```

@wraps
@staticmethod @classmethod
(__func__)

11.4 9.4

11.4.1

11.4.2

```
from functools import wraps
import logging

def logged(level, name=None, message=None):
    """
    Add logging to a function. level is the logging
    level, name is the logger name, and message is the
    log message. If name and message aren't specified,
    they default to the function's module and name.
    """
    def decorate(func):
        logname = name if name else func.__module__
        log = logging.getLogger(logname)
        logmsg = message if message else func.__name__

        @wraps(func)
```

```
        def wrapper(*args, **kwargs):
            log.log(level, logmsg)
            return func(*args, **kwargs)
        return wrapper
    return decorate

# Example use
@logged(logging.DEBUG)
def add(x, y):
    return x + y

@logged(logging.CRITICAL, 'example')
def spam():
    print('Spam!')
```

logged()

decorate()

logged()

11.4.3

```
@decorator(x, y, z)
def func(a, b):
    pass
```

;

```
def func(a, b):
    pass
func = decorator(x, y, z)(func)
```

decorator(x, y, z)

9.7

11.5 9.5

11.5.1

11.5.2

nolocal

```
from functools import wraps, partial
import logging
# Utility decorator to attach a function as an attribute of obj
def attach_wrapper(obj, func=None):
    if func is None:
        return partial(attach_wrapper, obj)
    setattr(obj, func.__name__, func)
    return func

def logged(level, name=None, message=None):
    '''
    Add logging to a function. level is the logging
    level, name is the logger name, and message is the
    log message. If name and message aren't specified,
    they default to the function's module and name.
    '''
    def decorate(func):
        logname = name if name else func.__module__
        log = logging.getLogger(logname)
        logmsg = message if message else func.__name__

        @wraps(func)
        def wrapper(*args, **kwargs):
            log.log(level, logmsg)
            return func(*args, **kwargs)

        # Attach setter functions
        @attach_wrapper(wrapper)
        def set_level(newlevel):
            nonlocal level
            level = newlevel

        @attach_wrapper(wrapper)
        def set_message(newmsg):
            nonlocal logmsg
            logmsg = newmsg

        return wrapper

    return decorate

# Example use
@logged(logging.DEBUG)
def add(x, y):
    return x + y
```

```
@logged(logging.CRITICAL, 'example')
def spam():
    print('Spam!')
```

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG)
>>> add(2, 3)
DEBUG:__main__:add
5
>>> # Change the log message
>>> add.set_message('Add called')
>>> add(2, 3)
DEBUG:__main__:Add called
5
>>> # Change the log level
>>> add.set_level(logging.WARNING)
>>> add(2, 3)
WARNING:__main__:Add called
5
>>>
```

11.5.3

(set_message() set_level())
nonlocal

(
@functools.wraps) 9.2
@timethis

```
@timethis
@logged(logging.DEBUG)
def countdown(n):
    while n > 0:
        n -= 1
```

```
>>> countdown(10000000)
DEBUG:__main__:countdown
countdown 0.8198461532592773
>>> countdown.set_level(logging.WARNING)
>>> countdown.set_message("Counting down to zero")
>>> countdown(10000000)
WARNING:__main__:Counting down to zero
countdown 0.8225970268249512
>>>
```

```
@logged(logging.DEBUG)
@timethis
def countdown(n):
    while n > 0:
        n -= 1
```

lambda

```
@attach_wrapper(wrapper)
def get_level():
    return level

# Alternative
wrapper.get_level = lambda: level
```

```
@wraps(func)
def wrapper(*args, **kwargs):
    wrapper.log.log(wrapper.level, wrapper.logmsg)
    return func(*args, **kwargs)

# Attach adjustable attributes
wrapper.level = level
wrapper.logmsg = logmsg
wrapper.log = log
```

(@timethis)

9.9

11.6 9.6

11.6.1

@decorator

@decorator(x,y,z)

11.6.2

9.5

```
from functools import wraps, partial
import logging
```

```
def logged(func=None, *, level=logging.DEBUG, name=None, message=None):
    if func is None:
        return partial(logged, level=level, name=name, message=message)

    logname = name if name else func.__module__
    log = logging.getLogger(logname)
    logmsg = message if message else func.__name__

    @wraps(func)
    def wrapper(*args, **kwargs):
        log.log(level, logmsg)
        return func(*args, **kwargs)

    return wrapper

# Example use
@logged
def add(x, y):
    return x + y

@logged(level=logging.CRITICAL, name='example')
def spam():
    print('Spam!')
```

@logged

11.6.3

```
@logged()
def add(x, y):
    return x+y
```

```
# Example use
@logged
def add(x, y):
    return x + y
```

```
def add(x, y):
    return x + y

add = logged(add)
```

logged

logged()

```
@logged(level=logging.CRITICAL, name='example')
def spam():
    print('Spam!')
```

```
def spam():
    print('Spam!')
spam = logged(level=logging.CRITICAL, name='example')(spam)
```

logged()

(9.5)

functools.partial

7.8

partial()

11.7 9.7

11.7.1

11.7.2

```
>>> @typeassert(int, int)
... def add(x, y):
...     return x + y
...
>>>
>>> add(2, 3)
5
>>> add(2, 'hello')
Traceback (most recent call last):
```

```
    File "<stdin>", line 1, in <module>
    File "contract.py", line 33, in wrapper
TypeError: Argument y must be <class 'int'>
>>>
```

@typeassert

```
from inspect import signature
from functools import wraps

def typeassert(*ty_args, **ty_kwargs):
    def decorate(func):
        # If in optimized mode, disable type checking
        if not __debug__:
            return func

        # Map function argument names to supplied types
        sig = signature(func)
        bound_types = sig.bind_partial(*ty_args, **ty_kwargs).arguments

        @wraps(func)
        def wrapper(*args, **kwargs):
            bound_values = sig.bind(*args, **kwargs)
            # Enforce type assertions across supplied arguments
            for name, value in bound_values.arguments.items():
                if name in bound_types:
                    if not isinstance(value, bound_types[name]):
                        raise TypeError(
                            'Argument {} must be {}'.format(name, bound_types[name])
                            )
            return func(*args, **kwargs)
        return wrapper
    return decorate
```

```
>>> @typeassert(int, z=int)
... def spam(x, y, z=42):
...     print(x, y, z)
...
>>> spam(1, 2, 3)
1 2 3
>>> spam(1, 'hello', 3)
1 hello 3
>>> spam(1, 'hello', 'world')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "contract.py", line 33, in wrapper
TypeError: Argument z must be <class 'int'>
>>>
```

11.7.3

__debug__
False(-O -OO)

```
def decorate(func):
    # If in optimized mode, disable type checking
    if not __debug__:
        return func
```

inspect.signature()

```
>>> from inspect import signature
>>> def spam(x, y, z=42):
...     pass
...
>>> sig = signature(spam)
>>> print(sig)
(x, y, z=42)
>>> sig.parameters
mappingproxy(OrderedDict([('x', <Parameter at 0x10077a050 'x'>),
('y', <Parameter at 0x10077a158 'y'>), ('z', <Parameter at 0x10077a1b0 'z'>)]))
>>> sig.parameters['z'].name
'z'
>>> sig.parameters['z'].default
42
>>> sig.parameters['z'].kind
<_ParameterKind: 'POSITIONAL_OR_KEYWORD'>
>>>
```

bind_partial()

```
>>> bound_types = sig.bind_partial(int,z=int)
>>> bound_types
<inspect.BoundArguments object at 0x10069bb50>
>>> bound_types.arguments
OrderedDict([('x', <class 'int'>), ('z', <class 'int'>)])
>>>
```

(y
) bound_types.arguments

sig.bind() bind()
bind_partial()

```
>>> bound_values = sig.bind(1, 2, 3)
>>> bound_values.arguments
OrderedDict([('x', 1), ('y', 2), ('z', 3)])
>>>
```

```
>>> for name, value in bound_values.arguments.items():
...     if name in bound_types.arguments:
...         if not isinstance(value, bound_types.arguments[name]):
...             raise TypeError()
...
>>>
```

items

```
>>> @typeassert(int, list)
... def bar(x, items=None):
...     if items is None:
...         items = []
...     items.append(x)
...     return items
>>> bar(2)
[2]
>>> bar(2,3)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "contract.py", line 33, in wrapper
TypeError: Argument items must be <class 'list'>
>>> bar(4, [1, 2, 3])
[1, 2, 3, 4]
>>>
```

```
@typeassert
def spam(x:int, y, z:int = 42):
    print(x,y,z)
```

@typeassert

PEP 362 inspect 9.16

11.8 9.8

11.8.1

11.8.2

```
from functools import wraps

class A:
    # Decorator as an instance method
    def decorator1(self, func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            print('Decorator 1')
            return func(*args, **kwargs)
        return wrapper

    # Decorator as a class method
    @classmethod
    def decorator2(cls, func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            print('Decorator 2')
            return func(*args, **kwargs)
        return wrapper
```

```
# As an instance method
a = A()
@a.decorator1
def spam():
    pass
# As a class method
@A.decorator2
def grok():
    pass
```

11.8.3

@property getter(), setter(), deleter() ,

```
class Person:
    # Create a property instance
    first_name = property()

    # Apply decorator methods
    @first_name.getter
    def first_name(self):
        return self._first_name

    @first_name.setter
    def first_name(self, value):
        if not isinstance(value, str):
            raise TypeError('Expected a string')
        self._first_name = value
```

property

self cls

decorator1() decorator2() self

cls wrapper() self

A B

```
class B(A):
    @A.decorator2
    def bar(self):
        pass
```

@B.decorator2 B

11.9 9.9

11.9.1

11.9.2

__call__() __get__()

```
import types
from functools import wraps

class Profiled:
    def __init__(self, func):
        wraps(func)(self)
        self.ncalls = 0

    def __call__(self, *args, **kwargs):
        self.ncalls += 1
        return self.__wrapped__(*args, **kwargs)

    def __get__(self, instance, cls):
        if instance is None:
            return self
        else:
            return types.MethodType(self, instance)
```

```
@Profiled
def add(x, y):
    return x + y

class Spam:
    @Profiled
    def bar(self, x):
        print(self, x)
```

```
>>> add(2, 3)
5
>>> add(4, 5)
9
>>> add.ncalls
2
>>> s = Spam()
>>> s.bar(1)
<__main__.Spam object at 0x10069e9d0> 1
>>> s.bar(2)
<__main__.Spam object at 0x10069e9d0> 2
>>> s.bar(3)
<__main__.Spam object at 0x10069e9d0> 3
>>> Spam.bar.ncalls
3
```

11.9.3

functools.wraps()

__get__()

```
>>> s = Spam()
>>> s.bar(3)
Traceback (most recent call last):
...
TypeError: bar() missing 1 required positional argument: 'x'
```

__get__()
8.9 __get__()
(self)

```
>>> s = Spam()
>>> def grok(self, x):
...     pass
...
>>> grok.__get__(s, Spam)
<bound method Spam.grok of <__main__.Spam object at 0x100671e90>>
>>>
```

__get__() type.MethodType()

__get__() instance None
Profiled ncalls

nonlocal

9.5

```
import types
from functools import wraps

def profiled(func):
    ncalls = 0
    @wraps(func)
    def wrapper(*args, **kwargs):
        nonlocal ncalls
        ncalls += 1
        return func(*args, **kwargs)
    wrapper.ncalls = lambda: ncalls
    return wrapper

# Example
@profiled
```

```
def add(x, y):
    return x + y
```

ncalls

```
>>> add(2, 3)
5
>>> add(4, 5)
9
>>> add.ncalls()
2
>>>
```

11.10 9.10

11.10.1

11.10.2

@classmethod

@staticmethod

```
import time
from functools import wraps

# A simple decorator
def timethis(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.time()
        r = func(*args, **kwargs)
        end = time.time()
        print(end-start)
        return r
    return wrapper

# Class illustrating application of the decorator to different kinds of methods
class Spam:
    @timethis
    def instance_method(self, n):
        print(self, n)
        while n > 0:
            n -= 1
```

```
    @classmethod
    @timethis
    def class_method(cls, n):
        print(cls, n)
        while n > 0:
            n -= 1

    @staticmethod
    @timethis
    def static_method(n):
        print(n)
        while n > 0:
            n -= 1
```

```
>>> s = Spam()
>>> s.instance_method(1000000)
<__main__.Spam object at 0x1006a6050> 1000000
0.11817407608032227
>>> Spam.class_method(1000000)
<class '__main__.Spam'> 1000000
0.11334395408630371
>>> Spam.static_method(1000000)
1000000
0.11740279197692871
>>>
```

11.10.3

```
class Spam:
    @timethis
    @staticmethod
    def static_method(n):
        print(n)
        while n > 0:
            n -= 1
```

```
>>> Spam.static_method(1000000)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "timethis.py", line 6, in wrapper
start = time.time()
TypeError: 'staticmethod' object is not callable
>>>
```

@classmethod @staticmethod

(8.9)

(8.12)

```
from abc import ABCMeta, abstractmethod
class A(metaclass=ABCMeta):
    @classmethod
    @abstractmethod
    def method(cls):
        pass
```

@classmethod @abstractmethod

11.11 9.11

11.11.1

11.11.2

```
from functools import wraps

def optional_debug(func):
    @wraps(func)
    def wrapper(*args, debug=False, **kwargs):
        if debug:
            print('Calling', func.__name__)
        return func(*args, **kwargs)

    return wrapper
```

```
>>> @optional_debug
... def spam(a,b,c):
... print(a,b,c)
...
>>> spam(1,2,3)
1 2 3
>>> spam(1,2,3, debug=True)
Calling spam
```

```
1 2 3
>>>
```

11.11.3

```
def a(x, debug=False):
    if debug:
        print('Calling a')

def b(x, y, z, debug=False):
    if debug:
        print('Calling b')

def c(x, y, debug=False):
    if debug:
        print('Calling c')
```

```
from functools import wraps
import inspect

def optional_debug(func):
    if 'debug' in inspect.getargspec(func).args:
        raise TypeError('debug argument already defined')

    @wraps(func)
    def wrapper(*args, debug=False, **kwargs):
        if debug:
            print('Calling', func.__name__)
        return func(*args, **kwargs)
    return wrapper

@optional_debug
def a(x):
    pass

@optional_debug
def b(x, y, z):
    pass

@optional_debug
def c(x, y):
    pass
```

*args

**kwargs

**kwargs

@optional_debug debug

```
>>> @optional_debug
... def add(x,y):
...     return x+y
...
>>> import inspect
>>> print(inspect.signature(add))
(x, y)
>>>
```

```
from functools import wraps
import inspect

def optional_debug(func):
    if 'debug' in inspect.getargspec(func).args:
        raise TypeError('debug argument already defined')

    @wraps(func)
    def wrapper(*args, debug=False, **kwargs):
        if debug:
            print('Calling', func.__name__)
        return func(*args, **kwargs)

    sig = inspect.signature(func)
    parms = list(sig.parameters.values())
    parms.append(inspect.Parameter('debug',
                inspect.Parameter.KEYWORD_ONLY,
                default=False))
    wrapper.__signature__ = sig.replace(parameters=parms)
    return wrapper
```

debug

```
>>> @optional_debug
... def add(x,y):
...     return x+y
...
>>> print(inspect.signature(add))
(x, y, *, debug=False)
>>> add(2,3)
5
>>>
```

9.16

11.12 9.12

11.12.1

11.12.2

__getattribute__

```
def log_getattribute(cls):
    # Get the original implementation
    orig_getattribute = cls.__getattribute__

    # Make a new definition
    def new_getattribute(self, name):
        print('getting:', name)
        return orig_getattribute(self, name)

    # Attach to the class and return
    cls.__getattribute__ = new_getattribute
    return cls

# Example use
@log_getattribute
class A:
    def __init__(self,x):
        self.x = x
    def spam(self):
        pass
```

```
>>> a = A(42)
>>> a.x
getting: x
42
>>> a.spam()
getting: spam
>>>
```

11.12.3

```
class LoggedGetattribute:
    def __getattribute__(self, name):
        print('getting:', name)
        return super().__getattribute__(name)

# Example:
class A(LoggedGetattribute):
    def __init__(self,x):
        self.x = x
    def spam(self):
        pass
```

super()

8.7

super()

A B

A B

8.13

11.13 9.13

11.13.1

11.13.2

Python

```
class Spam:
    def __init__(self, name):
        self.name = name

a = Spam('Guido')
b = Spam('Diana')
```

__call__()

```
class NoInstances(type):
    def __call__(self, *args, **kwargs):
        raise TypeError("Can't instantiate directly")

# Example
class Spam(metaclass=NoInstances):
    @staticmethod
    def grok(x):
        print('Spam.grok')
```

```
>>> Spam.grok(42)
Spam.grok
>>> s = Spam()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "example1.py", line 7, in __call__
        raise TypeError("Can't instantiate directly")
TypeError: Can't instantiate directly
>>>
```

```
class Singleton(type):
    def __init__(self, *args, **kwargs):
        self.__instance = None
        super().__init__(*args, **kwargs)

    def __call__(self, *args, **kwargs):
        if self.__instance is None:
            self.__instance = super().__call__(*args, **kwargs)
            return self.__instance
        else:
            return self.__instance

# Example
class Spam(metaclass=Singleton):
    def __init__(self):
        print('Creating Spam')
```

Spam

```
>>> a = Spam()
Creating Spam
>>> b = Spam()
>>> a is b
True
```

```
>>> c = Spam()
>>> a is c
True
>>>
```

8.25

```
import weakref

class Cached(type):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.__cache = weakref.WeakValueDictionary()

    def __call__(self, *args):
        if args in self.__cache:
            return self.__cache[args]
        else:
            obj = super().__call__(*args)
            self.__cache[args] = obj
            return obj

# Example
class Spam(metaclass=Cached):
    def __init__(self, name):
        print('Creating Spam({!r})'.format(name))
        self.name = name
```

```
>>> a = Spam('Guido')
Creating Spam('Guido')
>>> b = Spam('Diana')
Creating Spam('Diana')
>>> c = Spam('Guido') # Cached
>>> a is b
False
>>> a is c # Cached value returned
True
>>>
```

11.13.3

```
class _Spam:
    def __init__(self):
        print('Creating Spam')
```

```
_spam_instance = None

def Spam():
    global _spam_instance

    if _spam_instance is not None:
        return _spam_instance
    else:
        _spam_instance = _Spam()
        return _spam_instance
```

8.25

11.14 9.14

11.14.1

11.14.2

Ordered-

Dict

```
from collections import OrderedDict

# A set of descriptors for various types
class Typed:
    _expected_type = type(None)
    def __init__(self, name=None):
        self._name = name

    def __set__(self, instance, value):
        if not isinstance(value, self._expected_type):
            raise TypeError('Expected ' + str(self._expected_type))
        instance.__dict__[self._name] = value

class Integer(Typed):
    _expected_type = int

class Float(Typed):
    _expected_type = float
```

```
class String(Typed):
    _expected_type = str

# Metaclass that uses an OrderedDict for class body
class OrderedMeta(type):
    def __new__(cls, clsname, bases, clsdict):
        d = dict(clsdict)
        order = []
        for name, value in clsdict.items():
            if isinstance(value, Typed):
                value._name = name
                order.append(name)
        d['_order'] = order
        return type.__new__(cls, clsname, bases, d)

    @classmethod
    def __prepare__(cls, clsname, bases):
        return OrderedDict()
```

OrderedDict``

``_order

CSV

```
class Structure(metaclass=OrderedMeta):
    def as_csv(self):
        return ','.join(str(getattr(self,name)) for name in self._order)

# Example use
class Stock(Structure):
    name = String()
    shares = Integer()
    price = Float()

    def __init__(self, name, shares, price):
        self.name = name
        self.shares = shares
        self.price = price
```

Stock

```
>>> s = Stock('GOOG',100,490.1)
>>> s.name
'GOOG'
>>> s.as_csv()
'GOOG,100,490.1'
>>> t = Stock('AAPL','a lot', 610.23)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "dupmethod.py", line 34, in __init__
TypeError: shares expects <class 'int'>
```

```
>>>
```

11.14.3

OrderedMeta “ __prepare__()“

OrderedDict

```
from collections import OrderedDict

class NoDupOrderedDict(OrderedDict):
    def __init__(self, clsname):
        self.clsname = clsname
        super().__init__()
    def __setitem__(self, name, value):
        if name in self:
            raise TypeError('{} already defined in {}'.format(name, self.clsname))
        super().__setitem__(name, value)

class OrderedMeta(type):
    def __new__(cls, clsname, bases, clsdict):
        d = dict(clsdict)
        d['_order'] = [name for name in clsdict if name[0] != '_']
        return type.__new__(cls, clsname, bases, d)

    @classmethod
    def __prepare__(cls, clsname, bases):
        return NoDupOrderedDict(clsname)
```

```
>>> class A(metaclass=OrderedMeta):
... def spam(self):
... pass
... def spam(self):
... pass
...
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "<stdin>", line 4, in A
    File "dupmethod2.py", line 25, in __setitem__
        (name, self.clsname))
TypeError: spam already defined in A
>>>
```

__new__()

class
dict d = dict(clsdict)

```
class Stock(Model):
    name = String()
    shares = Integer()
    price = Float()
```

as_csv()

11.15 9.15

11.15.1

11.15.2

Python “metaclass“

```
from abc import ABCMeta, abstractmethod
class IStream(metaclass=ABCMeta):
    @abstractmethod
    def read(self, maxsize=None):
        pass

    @abstractmethod
    def write(self, data):
        pass
```

```
class Spam(metaclass=MyMeta, debug=True, synchronize=True):
    pass
```

__prepare__() , __new__()
__init__()

```
class MyMeta(type):
    # Optional
```

```
    @classmethod
    def __prepare__(cls, name, bases, *, debug=False, synchronize=False):
        # Custom processing
        pass
        return super().__prepare__(name, bases)

    # Required
    def __new__(cls, name, bases, ns, *, debug=False, synchronize=False):
        # Custom processing
        pass
        return super().__new__(cls, name, bases, ns)

    # Required
    def __init__(self, name, bases, ns, *, debug=False, synchronize=False):
        # Custom processing
        pass
        super().__init__(name, bases, ns)
```

11.15.3

__prepare__()

__new__()
__init__()

__new__() __init__()

__prepare__()

__prepare__()

```
class Spam(metaclass=MyMeta):
    debug = True
    synchronize = True
    pass
```

__prepare__()

__new__() __init__()

11.16 9.16 *args **kwargs

11.16.1

*args **kwargs

11.16.2

inspect
Signature Parameter

```
>>> from inspect import Signature, Parameter
>>> # Make a signature for a func(x, y=42, *, z=None)
>>> parms = [ Parameter('x', Parameter.POSITIONAL_OR_KEYWORD),
...         Parameter('y', Parameter.POSITIONAL_OR_KEYWORD, default=42),
...         Parameter('z', Parameter.KEYWORD_ONLY, default=None) ]
>>> sig = Signature(parms)
>>> print(sig)
(x, y=42, *, z=None)
>>>
```

bind()
*args **kwargs

```
>>> def func(*args, **kwargs):
...     bound_values = sig.bind(*args, **kwargs)
...     for name, value in bound_values.arguments.items():
...         print(name,value)
...
>>> # Try various examples
>>> func(1, 2, z=3)
x 1
y 2
z 3
>>> func(1)
x 1
>>> func(1, z=3)
x 1
z 3
>>> func(y=2, x=1)
x 1
y 2
>>> func(1, 2, 3, 4)
Traceback (most recent call last):
...
    File "/usr/local/lib/python3.3/inspect.py", line 1972, in _bind
        raise TypeError('too many positional arguments')
```

```
TypeError: too many positional arguments
>>> func(y=2)
Traceback (most recent call last):
...
    File "/usr/local/lib/python3.3/inspect.py", line 1961, in _bind
        raise TypeError(msg) from None
TypeError: 'x' parameter lacking default value
>>> func(1, y=2, x=3)
Traceback (most recent call last):
...
    File "/usr/local/lib/python3.3/inspect.py", line 1985, in _bind
        '{arg!r}'.format(arg=param.name))
TypeError: multiple values for argument 'x'
>>>
```

__init__()

```
from inspect import Signature, Parameter

def make_sig(*names):
    parms = [Parameter(name, Parameter.POSITIONAL_OR_KEYWORD)
             for name in names]
    return Signature(parms)

class Structure:
    __signature__ = make_sig()
    def __init__(self, *args, **kwargs):
        bound_values = self.__signature__.bind(*args, **kwargs)
        for name, value in bound_values.arguments.items():
            setattr(self, name, value)

# Example use
class Stock(Structure):
    __signature__ = make_sig('name', 'shares', 'price')

class Point(Structure):
    __signature__ = make_sig('x', 'y')
```

Stock

```
>>> import inspect
>>> print(inspect.signature(Stock))
(name, shares, price)
>>> s1 = Stock('ACME', 100, 490.1)
>>> s2 = Stock('ACME', 100)
Traceback (most recent call last):
...
TypeError: 'price' parameter lacking default value
```

```
>>> s3 = Stock('ACME', 100, 490.1, shares=50)
Traceback (most recent call last):
...
TypeError: multiple values for argument 'shares'
>>>
```

11.16.3

*args

**kwargs

8.11

```
from inspect import Signature, Parameter

def make_sig(*names):
    parms = [Parameter(name, Parameter.POSITIONAL_OR_KEYWORD)
             for name in names]
    return Signature(parms)

class StructureMeta(type):
    def __new__(cls, clsname, bases, clsdict):
        clsdict['__signature__'] = make_sig(*clsdict.get('_fields',[]))
        return super().__new__(cls, clsname, bases, clsdict)

class Structure(metaclass=StructureMeta):
    _fields = []
    def __init__(self, *args, **kwargs):
        bound_values = self.__signature__.bind(*args, **kwargs)
        for name, value in bound_values.arguments.items():
            setattr(self, name, value)

# Example
class Stock(Structure):
    _fields = ['name', 'shares', 'price']

class Point(Structure):
    _fields = ['x', 'y']
```

__signature__

inspect

```
>>> import inspect
>>> print(inspect.signature(Stock))
(name, shares, price)
>>> print(inspect.signature(Point))
```

```
(x, y)
>>>
```

11.17 9.17

11.17.1

11.17.2

type __new__() __init__()

```
class MyMeta(type):
    def __new__(self, clsname, bases, clsdict):
        # clsname is name of class being defined
        # bases is tuple of base classes
        # clsdict is class dictionary
        return super().__new__(cls, clsname, bases, clsdict)
```

__init__()

```
class MyMeta(type):
    def __init__(self, clsname, bases, clsdict):
        super().__init__(clsname, bases, clsdict)
        # clsname is name of class being defined
        # bases is tuple of base classes
        # clsdict is class dictionary
```

```
class Root(metaclass=MyMeta):
    pass

class A(Root):
    pass

class B(Root):
    pass
```

__init__()

Java ^_^

```
class NoMixedCaseMeta(type):
    def __new__(cls, clsname, bases, clsdict):
        for name in clsdict:
            if name.lower() != name:
                raise TypeError('Bad attribute name: ' + name)
        return super().__new__(cls, clsname, bases, clsdict)

class Root(metaclass=NoMixedCaseMeta):
    pass

class A(Root):
    def foo_bar(self): # Ok
        pass

class B(Root):
    def fooBar(self): # TypeError
        pass
```

```
from inspect import signature
import logging

class MatchSignaturesMeta(type):

    def __init__(self, clsname, bases, clsdict):
        super().__init__(clsname, bases, clsdict)
        sup = super(self, self)
        for name, value in clsdict.items():
            if name.startswith('_') or not callable(value):
                continue
            # Get the previous definition (if any) and compare the signatures
            prev_dfn = getattr(sup,name,None)
            if prev_dfn:
                prev_sig = signature(prev_dfn)
                val_sig = signature(value)
                if prev_sig != val_sig:
                    logging.warning('Signature mismatch in %s. %s != %s',
                                    value.__qualname__, prev_sig, val_sig)

# Example
class Root(metaclass=MatchSignaturesMeta):
    pass

class A(Root):
    def foo(self, x, y):
        pass
```

```
    def spam(self, x, *, z):
        pass

# Class with redefined methods, but slightly different signatures
class B(A):
    def foo(self, a, b):
        pass

    def spam(self,x,z):
        pass
```

```
WARNING:root:Signature mismatch in B.spam. (self, x, *, z) != (self, x, z)
WARNING:root:Signature mismatch in B.foo. (self, x, y) != (self, a, b)
```

bug

11.17.3

IDE

__new__() __init__()
__new__()
__init__()

super()

Python

inspect.signature()

super(self, self)
self
self

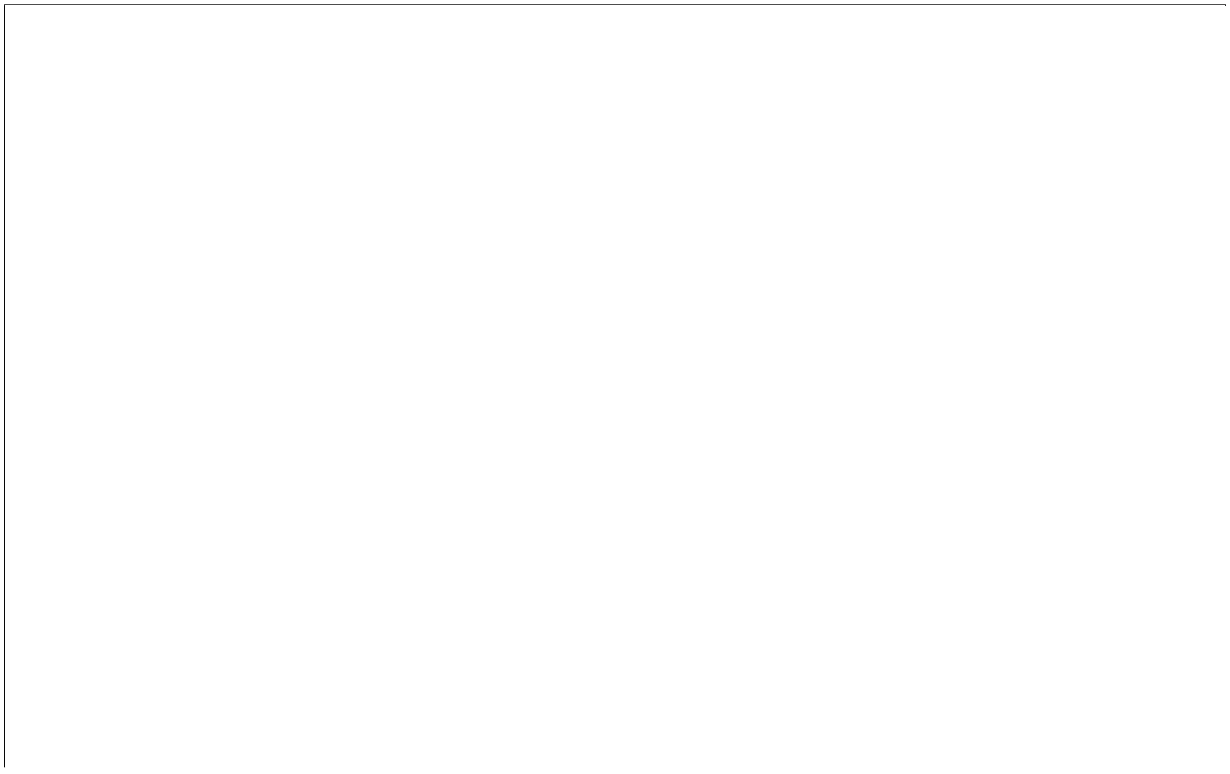
11.18 9.18

11.18.1

exec()

11.18.2

types.new_class()



O J21™Q/þ

pickle “ ”

types.new_class()

```
>>> import abc
>>> Stock = types.new_class('Stock', (), {'metaclass': abc.ABCMeta},
...                         lambda ns: ns.update(cls_dict))
...
>>> Stock.__module__ = __name__
>>> Stock
<class '__main__.Stock'>
>>> type(Stock)
<class 'abc.ABCMeta'>
>>>
```

```
class Spam(Base, debug=True, typecheck=False):
    pass
```

new_class()

```
Spam = types.new_class('Spam', (Base,),
                       {'debug': True, 'typecheck': False},
                       lambda ns: ns.update(cls_dict))
```

new_class()

__prepare__()

9.14 update()

11.18.3

collections.namedtuple()

```
>>> Stock = collections.namedtuple('Stock', ['name', 'shares', 'price'])
>>> Stock
<class '__main__.Stock'>
>>>
```

namedtuple() exec()

```
import operator
import types
import sys

def named_tuple(classname, fieldnames):
    # Populate a dictionary of field property accessors
```

```
    cls_dict = { name: property(operator.itemgetter(n))
                 for n, name in enumerate(fieldnames) }

    # Make a __new__ function and add to the class dict
    def __new__(cls, *args):
        if len(args) != len(fieldnames):
            raise TypeError('Expected {} arguments'.format(len(fieldnames)))
        return tuple.__new__(cls, args)

    cls_dict['__new__'] = __new__

    # Make the class
    cls = types.new_class(classname, (tuple,), {},
                          lambda ns: ns.update(cls_dict))

    # Set the module to that of the caller
    cls.__module__ = sys._getframe(1).f_globals['__name__']
    return cls
```

” ” sys._getframe()

2.15

```
>>> Point = named_tuple('Point', ['x', 'y'])
>>> Point
<class '__main__.Point'>
>>> p = Point(4, 5)
>>> len(p)
2
>>> p.x
4
>>> p.y
5
>>> p.x = 2
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>> print('%s %s' % p)
4 5
>>>
```

```
Stock = type('Stock', (), cls_dict)
```

__prepare__()

types.new_class()

types.new_class() __prepare__()

types.prepare_class()

```
import types
metaclass, kwargs, ns = types.prepare_class('Stock', (), {'metaclass': type})
```

__prepare__()

, PEP 3115 , Python documentation .

11.19 9.19

11.19.1

11.19.2

collections

```
import operator

class StructTupleMeta(type):
    def __init__(cls, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for n, name in enumerate(cls._fields):
            setattr(cls, name, property(operator.itemgetter(n)))

class StructTuple(tuple, metaclass=StructTupleMeta):
    _fields = []
    def __new__(cls, *args):
        if len(args) != len(cls._fields):
            raise ValueError('{} arguments required'.format(len(cls._fields)))
        return super().__new__(cls,args)
```

```
class Stock(StructTuple):
    _fields = ['name', 'shares', 'price']

class Point(StructTuple):
    _fields = ['x', 'y']
```

```
>>> s = Stock('ACME', 50, 91.1)
>>> s
('ACME', 50, 91.1)
>>> s[0]
'ACME'
>>> s.name
'ACME'
>>> s.shares * s.price
4555.0
>>> s.shares = 23
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>>
```

11.19.3

StructTupleMeta _fields
operator.itemgetter()
property()
StructTupleMeta
__init__() cls
_fields

StructTuple __new__()
__new__()

```
s = Stock('ACME', 50, 91.1) # OK
s = Stock(('ACME', 50, 91.1)) # Error
```

__init__() __new__()
__init__()
__new__()

Python

PEP 422

Python 3.3

11.20 9.20

11.20.1

11.20.2

Python

```
class Spam:
    def bar(self, x:int, y:int):
        print('Bar 1:', x, y)

    def bar(self, s:str, n:int = 0):
        print('Bar 2:', s, n)

s = Spam()
s.bar(2, 3) # Prints Bar 1: 2 3
s.bar('hello') # Prints Bar 2: hello 0
```

```
# multiple.py
import inspect
import types

class MultiMethod:
    '''
    Represents a single multimethod.
    '''
    def __init__(self, name):
        self._methods = {}
        self.__name__ = name

    def register(self, meth):
        '''
        Register a new method as a multimethod
        '''
        sig = inspect.signature(meth)

        # Build a type signature from the method's annotations
        types = []
        for name, parm in sig.parameters.items():
            if name == 'self':
                continue
            if parm.annotation is inspect.Parameter.empty:
```

```
                raise TypeError(
                    'Argument {} must be annotated with a type'.format(name)
                )
            if not isinstance(parm.annotation, type):
                raise TypeError(
                    'Argument {} annotation must be a type'.format(name)
                )
            if parm.default is not inspect.Parameter.empty:
                self._methods[tuple(types)] = meth
            types.append(parm.annotation)

        self._methods[tuple(types)] = meth

    def __call__(self, *args):
        '''
        Call a method based on type signature of the arguments
        '''
        types = tuple(type(arg) for arg in args[1:])
        meth = self._methods.get(types, None)
        if meth:
            return meth(*args)
        else:
            raise TypeError('No matching method for types {}'.format(types))

    def __get__(self, instance, cls):
        '''
        Descriptor method needed to make calls work in a class
        '''
        if instance is not None:
            return types.MethodType(self, instance)
        else:
            return self

class MultiDict(dict):
    '''
    Special dictionary to build multimethods in a metaclass
    '''
    def __setitem__(self, key, value):
        if key in self:
            # If key already exists, it must be a multimethod or callable
            current_value = self[key]
            if isinstance(current_value, MultiMethod):
                current_value.register(value)
            else:
                mvalue = MultiMethod(key)
                mvalue.register(current_value)
                mvalue.register(value)
                super().__setitem__(key, mvalue)
        else:
            super().__setitem__(key, value)
```

```
class MultipleMeta(type):
    '''
    Metaclass that allows multiple dispatch of methods
    '''
    def __new__(cls, clsname, bases, clsdict):
        return type.__new__(cls, clsname, bases, dict(clsdict))

    @classmethod
    def __prepare__(cls, clsname, bases):
        return MultiDict()
```

```
class Spam(metaclass=MultipleMeta):
    def bar(self, x:int, y:int):
        print('Bar 1:', x, y)

    def bar(self, s:str, n:int = 0):
        print('Bar 2:', s, n)

# Example: overloaded __init__
import time

class Date(metaclass=MultipleMeta):
    def __init__(self, year: int, month:int, day:int):
        self.year = year
        self.month = month
        self.day = day

    def __init__(self):
        t = time.localtime()
        self.__init__(t.tm_year, t.tm_mon, t.tm_mday)
```

```
>>> s = Spam()
>>> s.bar(2, 3)
Bar 1: 2 3
>>> s.bar('hello')
Bar 2: hello 0
>>> s.bar('hello', 5)
Bar 2: hello 5
>>> s.bar(2, 'hello')
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "multiple.py", line 42, in __call__
        raise TypeError('No matching method for types {}'.format(types))
TypeError: No matching method for types (<class 'int'>, <class 'str'>)
>>> # Overloaded __init__
>>> d = Date(2012, 12, 21)
>>> # Get today's date
```

```
>>> e = Date()
>>> e.year
2012
>>> e.month
12
>>> e.day
3
>>>
```

11.20.3

MutipleMeta
__prepare__() MultiDict
MultiDict
MultiMethod

MultiMethod

MultiMethod.register()

MultiMethod __call__()
slef map
MultiMethod __get__()

```
>>> b = s.bar
>>> b
<bound method Spam.bar of <__main__.Spam object at 0x1006a46d0>>
>>> b.__self__
<__main__.Spam object at 0x1006a46d0>
>>> b.__func__
<__main__.MultiMethod object at 0x1006a4d50>
>>> b(2, 3)
Bar 1: 2 3
>>> b('hello')
Bar 2: hello 0
>>>
```

```
>>> s.bar(x=2, y=3)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: __call__() got an unexpected keyword argument 'y'
```

```
>>> s.bar(s='hello')
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: __call__() got an unexpected keyword argument 's'
>>>
```

__call__()

```
class A:
    pass

class B(A):
    pass

class C:
    pass

class Spam(metaclass=MultipleMeta):
    def foo(self, x:A):
        print('Foo 1:', x)

    def foo(self, x:C):
        print('Foo 2:', x)
```

x:A B

```
>>> s = Spam()
>>> a = A()
>>> s.foo(a)
Foo 1: <__main__.A object at 0x1006a5310>
>>> c = C()
>>> s.foo(c)
Foo 2: <__main__.C object at 0x1007a1910>
>>> b = B()
>>> s.foo(b)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "multiple.py", line 44, in __call__
        raise TypeError('No matching method for types {}'.format(types))
TypeError: No matching method for types (<class '__main__.B'>,)
>>>
```

```
import types

class multimethod:
    def __init__(self, func):
```

```
        self._methods = {}
        self.__name__ = func.__name__
        self._default = func

    def match(self, *types):
        def register(func):
            ndefaults = len(func.__defaults__) if func.__defaults__ else 0
            for n in range(ndefaults+1):
                self._methods[types[:len(types) - n]] = func
            return self
        return register

    def __call__(self, *args):
        types = tuple(type(arg) for arg in args[1:])
        meth = self._methods.get(types, None)
        if meth:
            return meth(*args)
        else:
            return self._default(*args)

    def __get__(self, instance, cls):
        if instance is not None:
            return types.MethodType(self, instance)
        else:
            return self
```

```
class Spam:
    @multimethod
    def bar(self, *args):
        # Default method called if no match
        raise TypeError('No matching method for bar')

    @bar.match(int, int)
    def bar(self, x, y):
        print('Bar 1:', x, y)

    @bar.match(str, int)
    def bar(self, s, n = 0):
        print('Bar 2:', s, n)
```

8.21

Python

Guido van Rossum Five-Minute Multimethods in Python

11.21 9.21

11.21.1

11.21.2

```
class Person:
    def __init__(self, name ,age):
        self.name = name
        self.age = age

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        if not isinstance(value, str):
            raise TypeError('name must be a string')
        self._name = value

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if not isinstance(value, int):
            raise TypeError('age must be an int')
        self._age = value
```

```
def typed_property(name, expected_type):
    storage_name = '_' + name

    @property
    def prop(self):
        return getattr(self, storage_name)

    @prop.setter
    def prop(self, value):
```

```
        if not isinstance(value, expected_type):
            raise TypeError('{} must be a {}'.format(name, expected_type))
        setattr(self, storage_name, value)

    return prop

# Example use
class Person:
    name = typed_property('name', str)
    age = typed_property('age', int)

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

11.21.3

typed_property()

getter setter name , expected_type storate_name

functools.partial()

```
from functools import partial

String = partial(typed_property, expected_type=str)
Integer = partial(typed_property, expected_type=int)

# Example:
class Person:
    name = String('name')
    age = Integer('age')

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

8.13

11.22 9.22

11.22.1

with

11.22.2

contexlib

@contextmanager

```
import time
from contextlib import contextmanager

@contextmanager
def timethis(label):
    start = time.time()
    try:
        yield
    finally:
        end = time.time()
        print('{}: {}'.format(label, end - start))

# Example use
with timethis('counting'):
    n = 10000000
    while n > 0:
        n -= 1
```

timethis() yield __enter__()
yield __exit__()
yield

```
@contextmanager
def list_transaction(orig_list):
    working = list(orig_list)
    yield working
    orig_list[:] = working
```

```
>>> items = [1, 2, 3]
>>> with list_transaction(items) as working:
...     working.append(4)
...     working.append(5)
...
>>> items
[1, 2, 3, 4, 5]
>>> with list_transaction(items) as working:
...     working.append(6)
...     working.append(7)
...     raise RuntimeError('oops')
...
Traceback (most recent call last):
    File "<stdin>", line 4, in <module>
```

```
RuntimeError: oops
>>> items
[1, 2, 3, 4, 5]
>>>
```

11.22.3

__enter__() __exit__()

```
import time

class timethis:
    def __init__(self, label):
        self.label = label

    def __enter__(self):
        self.start = time.time()

    def __exit__(self, exc_ty, exc_val, exc_tb):
        end = time.time()
        print('{}: {}'.format(self.label, end - self.start))
```

@contextmanager

@contextmanager
( ) with
__enter__() __exit__()

11.23 9.23

11.23.1

11.23.2

```
>>> a = 13
>>> exec('b = a + 1')
>>> print(b)
14
>>>
```

```
>>> def test():
...     a = 13
...     exec('b = a + 1')
...     print(b)
...
>>> test()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    File "<stdin>", line 4, in test
NameError: global name 'b' is not defined
>>>
```

NameError exec() exec()

exec() locals()

```
>>> def test():
...     a = 13
...     loc = locals()
...     exec('b = a + 1')
...     b = loc['b']
...     print(b)
...
>>> test()
14
>>>
```

11.23.3

exec() exec()

exec()

exec()

exec() exec()

```
>>> def test1():
...     x = 0
...     exec('x += 1')
...     print(x)
...
>>> test1()
0
>>>
```

locals() exec()

```
>>> def test2():
...     x = 0
...     loc = locals()
...     print('before:', loc)
...     exec('x += 1')
...     print('after:', loc)
...     print('x =', x)
...
>>> test2()
before: {'x': 0}
after: {'loc': {...}, 'x': 1}
x = 0
>>>
```

loc x x

locals() locals()

```
>>> def test3():
...     x = 0
...     loc = locals()
...     print(loc)
...     exec('x += 1')
...     print(loc)
...     locals()
...     print(loc)
...
>>> test3()
{'x': 0}
{'loc': {...}, 'x': 1}
{'loc': {...}, 'x': 0}
>>>
```

locals() x

locals() exec()

```
>>> def test4():
...     a = 13
...     loc = { 'a' : a }
...     glb = { }
...     exec('b = a + 1', glb, loc)
...     b = loc['b']
...     print(b)
...
```

```
>>> test4()
14
>>>
```

exec()

exec()

exec()

11.24 9.24 Python

11.24.1

Python

11.24.2

Python

```
>>> x = 42
>>> eval('2 + 3*4 + x')
56
>>> exec('for i in range(10): print(i)')
0
1
2
3
4
5
6
7
8
9
>>>
```

ast Python
AST

```
>>> import ast
>>> ex = ast.parse('2 + 3*4 + x', mode='eval')
>>> ex
<_ast.Expression object at 0x1007473d0>
>>> ast.dump(ex)
"Expression(body=BinOp(left=BinOp(left=Num(n=2), op=Add(),
right=BinOp(left=Num(n=3), op=Mult(), right=Num(n=4))), op=Add(),
right=Name(id='x', ctx=Load())))"
```

dump(top)

```
>>> top = ast.parse('for i in range(10): print(i)', mode='exec')
>>> top
<_ast.Module object at 0x100747390>
>>> ast.dump(top)
"Module(body=[For(target=Name(id='i', ctx=Store()),
iter=Call(func=Name(id='range', ctx=Load()), args=[Num(n=10)],
keywords=[], starargs=None, kwargs=None),
body=[Expr(value=Call(func=Name(id='print', ctx=Load()),
args=[Name(id='i', ctx=Load())], keywords=[], starargs=None,
kwargs=None))], orelse=[])])"
>>>
```

AST

visit_NodeName()

NodeName() =

```
    print('Deleted:', c.deleted)
```

```
Loaded: {'i', 'range', 'print'}
Stored: {'i'}
Deleted: {'i'}
```

AST compile()

```
>>> exec(compile(top,'<stdin>', 'exec'))
0
1
2
3
4
5
6
7
8
9
>>>
```

11.24.3

exec()

AST

AST
AST

```
# namelower.py
import ast
import inspect

# Node visitor that lowers globally accessed names into
# the function body as local variables.
class NameLower(ast.NodeVisitor):
    def __init__(self, lowered_names):
        self.lowered_names = lowered_names

    def visit_FunctionDef(self, node):
        # Compile some assignments to lower the constants
        code = '__globals = globals()\n'
        code += '\n'.join("{0} = __globals['{0}']".format(name)
                          for name in self.lowered_names)
        code_ast = ast.parse(code, mode='exec')
```

```
        # Inject new statements into the function body
        node.body[:0] = code_ast.body

        # Save the function object
        self.func = node

# Decorator that turns global names into locals
def lower_names(*namelist):
    def lower(func):
        srclines = inspect.getsource(func).splitlines()
        # Skip source lines prior to the @lower_names decorator
        for n, line in enumerate(srclines):
            if '@lower_names' in line:
                break

        src = '\n'.join(srclines[n+1:])
        # Hack to deal with indented code
        if src.startswith((' ','\t')):
            src = 'if 1:\n' + src
        top = ast.parse(src, mode='exec')

        # Transform the AST
        cl = NameLower(namelist)
        cl.visit(top)

        # Execute the modified AST
        temp = {}
        exec(compile(top,'','exec'), temp, temp)

        # Pull out the modified code object
        func.__code__ = temp[func.__name__].__code__
        return func
    return lower
```

```
INCR = 1
@lower_names('INCR')
def countdown(n):
    while n > 0:
        n -= INCR
```

```
countdown()
```

```
def countdown(n):
    __globals = globals()
    INCR = __globals['INCR']
    while n > 0:
        n -= INCR
```

20%

AST

ActiveState Python AST

11.25 9.25 Python

11.25.1

11.25.2

dis Python

```
>>> def countdown(n):
... while n > 0:
...     print('T-minus', n)
...     n -= 1
... print('Blastoff!')
...
>>> import dis
>>> dis.dis(countdown)
...
>>>
```

11.25.3

dis

dis()

```
>>> countdown.__code__.co_code
b"x'\x00|\x00\x00d\x01\x00k\x04\x00r)\x00t\x00\x00d\x02\x00|\x00\x00\x83
\x02\x00\x01|\x00\x00d\x03\x008}\x00\x00q\x03\x00Wt\x00\x00d\x04\x00\x83
\x01\x00\x01d\x00\x00S"
>>>
```

opcode

```
>>> c = countdown.__code__.co_code
>>> import opcode
>>> opcode.opname[c[0]]
>>> opcode.opname[c[0]]
'SETUP_LOOP'
>>> opcode.opname[c[3]]
```

```
'LOAD_FAST'
>>>
```

dis

opcodes

```
import opcode

def generate_opcodes(codebytes):
    extended_arg = 0
    i = 0
    n = len(codebytes)
    while i < n:
        op = codebytes[i]
        i += 1
        if op >= opcode.HAVE_ARGUMENT:
            oparg = codebytes[i] + codebytes[i+1]*256 + extended_arg
            extended_arg = 0
            i += 2
            if op == opcode.EXTENDED_ARG:
                extended_arg = oparg * 65536
                continue
        else:
            oparg = None
        yield (op, oparg)
```

```
>>> for op, oparg in generate_opcodes(countdown.__code__.co_code):
...     print(op, opcode.opname[op], oparg)
```

```
>>> def add(x, y):
...     return x + y
...
>>> c = add.__code__
>>> c
<code object add at 0x1007beed0, file "<stdin>", line 1>
>>> c.co_code
b'|\x00\x00|\x01\x00\x17S'
>>>
>>> # Make a completely new code object with bogus byte code
>>> import types
>>> newbytecode = b'xxxxxxx'
>>> nc = types.CodeType(c.co_argcount, c.co_kwonlyargcount,
...     c.co_nlocals, c.co_stacksize, c.co_flags, newbytecode, c.co_consts,
...     c.co_names, c.co_varnames, c.co_filename, c.co_name,
...     c.co_firstlineno, c.co_lnotab)
>>> nc
<code object add at 0x10069fe40, file "<stdin>", line 1>
```

```
>>> add.__code__ = nc
>>> add(2,3)
Segmentation fault
```

this code on ActiveState

CHAPTER

TWELVE

Python

Contents:

12.1 10.1

12.1.1

12.1.2

__init__.py

```
graphics/
    __init__.py
    primitive/
        __init__.py
        line.py
        fill.py
        text.py
    formats/
        __init__.py
        png.py
        jpg.py
```

import

```
import graphics.primitive.line
from graphics.primitive import line
import graphics.formats.jpg as jpg
```

12.1.3

__init__.py

import graphics graphics/__init__.py , graphics import graphics.format.jpg graphics/__init__.py graphics/ graphics/formats/__init__.py graphics/formats/jpg.py

__init__.py
__init__.py :

```
# graphics/formats/__init__.py
from . import jpg
from . import png
```

, import grahpics.formats import graphics.formats.jpg import graphics.formats.png

__init__.py 10.4

__init__.py python
__init__.py “ ” 10.5
__init__.py

12.2 10.2

12.2.1

’from module import *‘

12.2.2

__all__

:

```
# somemodule.py
def spam():
    pass

def grok():
    pass

blah = 42
# Only export 'spam' and 'grok'
__all__ = ['spam', 'grok']
```

12.2.3

‘from module import *‘,
, ,
__all__ ,
__all__ , __all__
, AttributeError

12.3 10.3

12.3.1

, import

12.3.2

mypackage

```
mypackage/
    __init__.py
    A/
        __init__.py
        spam.py
        grok.py
    B/
        __init__.py
        bar.py
```

mypackage.A.spam grok import

```
# mypackage/A/spam.py
from . import grok
```

mypackage.A.spam B.bar im-
port

```
# mypackage/A/spam.py
from ..B import bar
```

import spam.py

12.3.3

```
# mypackage/A/spam.py
from mypackage.A import grok # OK
from . import grok # OK
import grok # Error (not found)
```

mypackage.A

ok

import . ".." , . ..B ../B import

```
from . import grok # OK
import .grok # ERROR
```

```
% python3 mypackage/A/spam.py # Relative imports fail
```

Python -m

```
% python3 -m mypackage.A.spam # Relative imports work
```

, PEP 328 .

12.4 10.4

12.4.1

12.4.2

```
# mymodule.py
class A:
    def spam(self):
        print('A.spam')

class B(A):
    def bar(self):
        print('B.bar')
```

mymodule.py
mymodule mymodule.py

```
mymodule/
    __init__.py
    a.py
    b.py
```

a.py

```
# a.py
class A:
    def spam(self):
        print('A.spam')
```

b.py

```
# b.py
from .a import A
class B(A):
    def bar(self):
        print('B.bar')
```

__init__.py 2

```
# __init__.py
from .a import A
from .b import B
```

MyModule

```
>>> import mymodule
>>> a = mymodule.A()
>>> a.spam()
A.spam
>>> b = mymodule.B()
>>> b.bar()
B.bar
>>>
```

12.4.3

import

```
from mymodule.a import A
from mymodule.b import B
...
```

import

```
from mymodule import A, B
```

mymodule

__init__.py

B A from .a import A

10.3

__init__.py

__init__.py

```
# __init__.py
def A():
    from .a import A
    return A()

def B():
    from .b import B
    return B()
```

A B

```
>>> import mymodule
>>> a = mymodule.A()
>>> a.spam()
A.spam
>>>
```

:

```
if isinstance(x, mymodule.A): # Error
...
```

```
if isinstance(x, mymodule.a.A): # Ok
...
```

, multiprocessing/__init__.py .

12.5 10.5

12.5.1

12.5.2

Python

__init__.py Python

```
foo-package/
    spam/
        blah.py

bar-package/
    spam/
        grok.py
```

2 spam
__init__.py

foo-package bar-package python

```
>>> import sys
>>> sys.path.extend(['foo-package', 'bar-package'])
>>> import spam.blah
>>> import spam.grok
>>>
```

spam.blah spam.grok

12.5.3

“ ”

__init__.py

__init__.py

__path__

```
>>> import spam
>>> spam.__path__
_NamespacePath(['foo-package/spam', 'bar-package/spam'])
>>>
```

__path__ (, spam.grok spam.blah).

```
my-package/
    spam/
        custom.py
```

sys.path

spam

```
>>> import spam.custom
>>> import spam.grok
>>> import spam.blah
>>>
```

__file__

“ namespace ”

```
>>> spam.__file__
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: 'module' object has no attribute '__file__'
>>> spam
<module 'spam' (namespace)>
>>>
```

PEP 420.

12.6 10.6

12.6.1

12.6.2

imp.reload()

```
>>> import spam
>>> import imp
>>> imp.reload(spam)
<module 'spam' from './spam.py'>
>>>
```

12.6.3

reload()

reload() "from module import name" import

```
# spam.py
def bar():
    print('bar')

def grok():
    print('grok')
```

```
>>> import spam
>>> from spam import grok
>>> spam.bar()
bar
>>> grok()
grok
>>>
```

Python spam.py grok()

```
def grok():
    print('New grok')
```

```
>>> import imp
>>> imp.reload(spam)
<module 'spam' from './spam.py'>
>>> spam.bar()
bar
>>> grok() # Notice old output
grok
>>> spam.grok() # Notice new output
New grok
>>>
```

2 grok()

12.7 10.7

12.7.1

12.7.2

__main__.py

```
myapplication/
    spam.py
    bar.py
    grok.py
    __main__.py
```

__main__.py Python

```
bash % python3 myapplication
```

__main__.py

zip

```
bash % ls
spam.py bar.py grok.py __main__.py
bash % zip -r myapp.zip *.py
```

```
bash % python3 myapp.zip
... output from __main__.py ...
```

12.7.3

zip __main__.py Python
Python

zip shell
myapp.zip

```
#!/usr/bin/env python3 /usr/local/bin/myapp.zip
```

12.8 10.8

12.8.1

12.8.2

```
mypackage/
    __init__.py
    somedata.dat
    spam.py
```

spam.py somedata.dat

```
# spam.py
import pkgutil
data = pkgutil.get_data(__package__, 'somedata.dat')
```

12.8.3

I/ O open()

I/O

__file__

.zip .egg

open()

pkgutil.get_data()

get_data()

__package__

Unix

12.9 10.9 sys.path

12.9.1

Python sys.path

Python

12.9.2

sys.path PYTHONPATH

```
bash % env PYTHONPATH=/some/dir:/other/dir python3
Python 3.3.0 (default, Oct 4 2012, 10:17:33)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', '/some/dir', '/other/dir', ...]
>>>
```

shell

.pth

```
# myapplication.pth
/some/dir
/other/dir
```

.pth Python site-packages /usr/local/lib/python3.3/site-packages ~/.local/lib/python3.3/sitepackages

.pth sys.path .pth

Python

12.9.3

sys.path :

```
import sys
sys.path.insert(0, '/some/dir')
sys.path.insert(0, '/other/dir')
```

“ ”

path

__file__

```
import sys
from os.path import abspath, join, dirname
sys.path.insert(0, join(abspath(dirname('__file__')), 'src'))
```

src path

site-packages

site-packages path .pth site-packages

.pth

12.10 10.10

12.10.1

12.10.2

importlib.import_module()

```
>>> import importlib
>>> math = importlib.import_module('math')
>>> math.sin(2)
0.9092974268256817
>>> mod = importlib.import_module('urllib.request')
>>> u = mod.urlopen('http://www.python.org')
>>>
```

import_module import

import_module()

```
import importlib
# Same as 'from . import b'
b = importlib.import_module('.b', __package__)
```

12.10.3

import_module()

__import__()
importlib.import_module()

10.11

12.11 10.11

12.11.1

Python import

12.11.2

Python import
import

Python

```
testcode/
    spam.py
    fib.py
    grok/
        __init__.py
        blah.py
```

```
# spam.py
print("I'm spam")

def hello(name):
    print('Hello %s' % name)
```

```
# fib.py
print("I'm fib")

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

# grok/__init__.py
print("I'm grok.__init__")

# grok/blah.py
print("I'm grok.blah")
```

web testcode Python

```
bash % cd testcode
bash % python3 -m http.server 15000
Serving HTTP on 0.0.0.0 port 15000 ...
```

Python `urllib`

```
>>> from urllib.request import urlopen
>>> u = urlopen('http://localhost:15000/fib.py')
>>> data = u.read().decode('utf-8')
>>> print(data)
# fib.py
print("I'm fib")

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

`urlopen()`

import

```
import imp
import urllib.request
import sys

def load_module(url):
    u = urllib.request.urlopen(url)
    source = u.read().decode('utf-8')
    mod = sys.modules.setdefault(url, imp.new_module(url))
```

```
    code = compile(source, url, 'exec')
    mod.__file__ = url
    mod.__package__ = ''
    exec(code, mod.__dict__)
    return mod
```

compile()

```
>>> fib = load_module('http://localhost:15000/fib.py')
I'm fib
>>> fib.fib(10)
89
>>> spam = load_module('http://localhost:15000/spam.py')
I'm spam
>>> spam.hello('Guido')
Hello Guido
>>> fib
<module 'http://localhost:15000/fib.py' from 'http://localhost:15000/fib.py'>
>>> spam
<module 'http://localhost:15000/spam.py' from 'http://localhost:15000/spam.py'>
>>>
```

import

```
# urlimport.py
import sys
import importlib.abc
import imp
from urllib.request import urlopen
from urllib.error import HTTPError, URLError
from html.parser import HTMLParser

# Debugging
import logging
log = logging.getLogger(__name__)

# Get links from a given URL
def _get_links(url):
    class LinkParser(HTMLParser):
        def handle_starttag(self, tag, attrs):
            if tag == 'a':
                attrs = dict(attrs)
                links.add(attrs.get('href').rstrip('/'))
    links = set()
    try:
        log.debug('Getting links from %s' % url)
        u = urlopen(url)
```

```
        parser = LinkParser()
        parser.feed(u.read().decode('utf-8'))
    except Exception as e:
        log.debug('Could not get links. %s', e)
    log.debug('links: %r', links)
    return links

class UrlMetaFinder(importlib.abc.MetaPathFinder):
    def __init__(self, baseurl):
        self._baseurl = baseurl
        self._links = { }
        self._loaders = { baseurl : UrlModuleLoader(baseurl) }

    def find_module(self, fullname, path=None):
        log.debug('find_module: fullname=%r, path=%r', fullname, path)
        if path is None:
            baseurl = self._baseurl
        else:
            if not path[0].startswith(self._baseurl):
                return None
            baseurl = path[0]
        parts = fullname.split('.')
        basename = parts[-1]
        log.debug('find_module: baseurl=%r, basename=%r', baseurl, basename)

        # Check link cache
        if basename not in self._links:
            self._links[baseurl] = _get_links(baseurl)

        # Check if it's a package
        if basename in self._links[baseurl]:
            log.debug('find_module: trying package %r', fullname)
            fullurl = self._baseurl + '/' + basename
            # Attempt to load the package (which accesses __init__.py)
            loader = UrlPackageLoader(fullurl)
            try:
                loader.load_module(fullname)
                self._links[fullurl] = _get_links(fullurl)
                self._loaders[fullurl] = UrlModuleLoader(fullurl)
                log.debug('find_module: package %r loaded', fullname)
            except ImportError as e:
                log.debug('find_module: package failed. %s', e)
                loader = None
            return loader
        # A normal module
        filename = basename + '.py'
        if filename in self._links[baseurl]:
            log.debug('find_module: module %r found', fullname)
            return self._loaders[baseurl]
        else:
            log.debug('find_module: module %r not found', fullname)
```

```
            return None

    def invalidate_caches(self):
        log.debug('invalidating link cache')
        self._links.clear()

# Module Loader for a URL
class UrlModuleLoader(importlib.abc.SourceLoader):
    def __init__(self, baseurl):
        self._baseurl = baseurl
        self._source_cache = {}

    def module_repr(self, module):
        return '<urlmodule %r from %r>' % (module.__name__, module.__file__)

    # Required method
    def load_module(self, fullname):
        code = self.get_code(fullname)
        mod = sys.modules.setdefault(fullname, imp.new_module(fullname))
        mod.__file__ = self.get_filename(fullname)
        mod.__loader__ = self
        mod.__package__ = fullname.rpartition('.')[0]
        exec(code, mod.__dict__)
        return mod

    # Optional extensions
    def get_code(self, fullname):
        src = self.get_source(fullname)
        return compile(src, self.get_filename(fullname), 'exec')

    def get_data(self, path):
        pass

    def get_filename(self, fullname):
        return self._baseurl + '/' + fullname.split('.')[-1] + '.py'

    def get_source(self, fullname):
        filename = self.get_filename(fullname)
        log.debug('loader: reading %r', filename)
        if filename in self._source_cache:
            log.debug('loader: cached %r', filename)
            return self._source_cache[filename]
        try:
            u = urlopen(filename)
            source = u.read().decode('utf-8')
            log.debug('loader: %r loaded', filename)
            self._source_cache[filename] = source
            return source
        except (HTTPError, URLError) as e:
            log.debug('loader: %r failed. %s', filename, e)
            raise ImportError("Can't load %s" % filename)
```

```
    def is_package(self, fullname):
        return False

# Package loader for a URL
class UrlPackageLoader(UrlModuleLoader):
    def load_module(self, fullname):
        mod = super().load_module(fullname)
        mod.__path__ = [ self._baseurl ]
        mod.__package__ = fullname

    def get_filename(self, fullname):
        return self._baseurl + '/' + '__init__.py'

    def is_package(self, fullname):
        return True

# Utility functions for installing/uninstalling the loader
_installed_meta_cache = { }
def install_meta(address):
    if address not in _installed_meta_cache:
        finder = UrlMetaFinder(address)
        _installed_meta_cache[address] = finder
        sys.meta_path.append(finder)
        log.debug('%r installed on sys.meta_path', finder)

def remove_meta(address):
    if address in _installed_meta_cache:
        finder = _installed_meta_cache.pop(address)
        sys.meta_path.remove(finder)
        log.debug('%r removed from sys.meta_path', finder)
```

```
>>> # importing currently fails
>>> import fib
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'
>>> # Load the importer and retry (it works)
>>> import urlimport
>>> urlimport.install_meta('http://localhost:15000')
>>> import fib
I'm fib
>>> import spam
I'm spam
>>> import grok.blah
I'm grok.__init__
I'm grok.blah
>>> grok.blah.__file__
'http://localhost:15000/grok/blah.py'
```

```
>>>
```

UrlMetaFinder
sys.meta_path sys.meta_path
UrlMetaFinder

UrlMetaFinder URL
URL
UrlModuleLoader

HTTP

sys.path
urlimport.py

```
# urlimport.py
# ... include previous code above ...
# Path finder class for a URL
class UrlPathFinder(importlib.abc.PathEntryFinder):
    def __init__(self, baseurl):
        self._links = None
        self._loader = UrlModuleLoader(baseurl)
        self._baseurl = baseurl

    def find_loader(self, fullname):
        log.debug('find_loader: %r', fullname)
        parts = fullname.split('.')
        basename = parts[-1]
        # Check link cache
        if self._links is None:
            self._links = [] # See discussion
            self._links = _get_links(self._baseurl)

        # Check if it's a package
        if basename in self._links:
            log.debug('find_loader: trying package %r', fullname)
            fullurl = self._baseurl + '/' + basename
            # Attempt to load the package (which accesses __init__.py)
            loader = UrlPackageLoader(fullurl)
            try:
                loader.load_module(fullname)
                log.debug('find_loader: package %r loaded', fullname)
            except ImportError as e:
                log.debug('find_loader: %r is a namespace package', fullname)
                loader = None
            return (loader, [fullurl])

        # A normal module
        filename = basename + '.py'
        if filename in self._links:
            log.debug('find_loader: module %r found', fullname)
```

```
            return (self._loader, [])
        else:
            log.debug('find_loader: module %r not found', fullname)
            return (None, [])

    def invalidate_caches(self):
        log.debug('invalidating link cache')
        self._links = None

# Check path to see if it looks like a URL
_url_path_cache = {}
def handle_url(path):
    if path.startswith(('http://', 'https://')):
        log.debug('Handle path? %s. [Yes]', path)
        if path in _url_path_cache:
            finder = _url_path_cache[path]
        else:
            finder = UrlPathFinder(path)
            _url_path_cache[path] = finder
        return finder
    else:
        log.debug('Handle path? %s. [No]', path)

def install_path_hook():
    sys.path_hooks.append(handle_url)
    sys.path_importer_cache.clear()
    log.debug('Installing handle_url')

def remove_path_hook():
    sys.path_hooks.remove(handle_url)
    sys.path_importer_cache.clear()
    log.debug('Removing handle_url')
```

sys.path URL

```
>>> # Initial import fails
>>> import fib
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'

>>> # Install the path hook
>>> import urlimport
>>> urlimport.install_path_hook()

>>> # Imports still fail (not on path)
>>> import fib
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'
```

```
>>> # Add an entry to sys.path and watch it work
>>> import sys
>>> sys.path.append('http://localhost:15000')
>>> import fib
I'm fib
>>> import grok.blah
I'm grok.__init__
I'm grok.blah
>>> grok.blah.__file__
'http://localhost:15000/grok/blah.py'
>>>
```

handle_url() sys.path_hooks sys.path sys.path_hooks sys.path

```
>>> fib
<urlmodule 'fib' from 'http://localhost:15000/fib.py'>
>>> fib.__name__
'fib'
>>> fib.__file__
'http://localhost:15000/fib.py'
>>> import inspect
>>> print(inspect.getsource(fib))
# fib.py
print("I'm fib")

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

12.11.3

Python Python importlib module PEP 302.

imp.new_module()

```
>>> import imp
>>> m = imp.new_module('spam')
>>> m
<module 'spam'>
>>> m.__name__
```

```
'spam'
>>>
```

__file__
__package__ ( )

sys.modules

```
>>> import sys
>>> import imp
>>> m = sys.modules.setdefault('spam', imp.new_module('spam'))
>>> m
<module 'spam'>
>>>
```

```
>>> import math
>>> m = sys.modules.setdefault('math', imp.new_module('math'))
>>> m
<module 'math' from '/usr/local/lib/python3.3/lib-dynload/math.so'>
>>> m.sin(2)
0.9092974268256817
>>> m.cos(2)
-0.4161468365471424
>>>
```

load_module()

import __init__.py
import

import
sys.meta_path “ ”

```
>>> from pprint import pprint
>>> pprint(sys.meta_path)
[<class '_frozen_importlib.BuiltinImporter'>,
<class '_frozen_importlib.FrozenImporter'>,
<class '_frozen_importlib.PathFinder'>]
>>>
```

import fib sys.mata_path
find_module()

```
>>> class Finder:
...     def find_module(self, fullname, path):
...         print('Looking for', fullname, path)
...         return None
...
```

```
>>> import sys
>>> sys.meta_path.insert(0, Finder()) # Insert as first entry
>>> import math
Looking for math None
>>> import types
Looking for types None
>>> import threading
Looking for threading None
Looking for time None
Looking for traceback None
Looking for linecache None
Looking for tokenize None
Looking for token None
>>>
```

find_module()
path __path__
xml.etree
xml.etree.ElementTree

```
>>> import xml.etree.ElementTree
Looking for xml None
Looking for xml.etree ['/usr/local/lib/python3.3/xml']
Looking for xml.etree.ElementTree ['/usr/local/lib/python3.3/xml/etree']
Looking for warnings None
Looking for contextlib None
Looking for xml.etree.ElementPath ['/usr/local/lib/python3.3/xml/etree']
Looking for _elementtree None
Looking for copy None
Looking for org None
Looking for pyexpat None
Looking for ElementC14N None
>>>
```

sys.meta_path

```
>>> del sys.meta_path[0]
>>> sys.meta_path.append(Finder())
>>> import urllib.request
>>> import datetime
```

sys.meta_path

```
>>> import fib
Looking for fib None
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'
>>> import xml.superfast
Looking for xml.superfast ['/usr/local/lib/python3.3/xml']
```

```
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ImportError: No module named 'xml.superfast'
>>>
```

UrlMetaFinder

UrlMetaFinder sys.meta_path

path URL

UrlPackageLoader
__init__.py __path__
find_module()
sys.path

Python

```
>>> from pprint import pprint
>>> import sys
>>> pprint(sys.path)
['',
'/usr/local/lib/python33.zip',
'/usr/local/lib/python3.3',
'/usr/local/lib/python3.3/plat-darwin',
'/usr/local/lib/python3.3/lib-dynload',
'/usr/local/lib/...3.3/site-packages']
>>>
```

sys.path
sys.path_importer_cache

```
>>> pprint(sys.path_importer_cache)
{'.': FileFinder('.'),
'/usr/local/lib/python3.3': FileFinder('/usr/local/lib/python3.3'),
'/usr/local/lib/python3.3/': FileFinder('/usr/local/lib/python3.3/'),
'/usr/local/lib/python3.3/collections': FileFinder('...python3.3/collections'),
'/usr/local/lib/python3.3/encodings': FileFinder('...python3.3/encodings'),
'/usr/local/lib/python3.3/lib-dynload': FileFinder('...python3.3/lib-dynload'),
'/usr/local/lib/python3.3/plat-darwin': FileFinder('...python3.3/plat-darwin'),
'/usr/local/lib/python3.3/site-packages': FileFinder('...python3.3/site-packages'),
'/usr/local/lib/python33.zip': None}
>>>
```

sys.path_importer_cache sys.path
sys.path

import fib sys.path “ fib ”
sys.path_importer_cache

```
>>> class Finder:
... def find_loader(self, name):
```

```
...     print('Looking for', name)
...     return (None, [])
...
>>> import sys
>>> # Add a "debug" entry to the importer cache
>>> sys.path_importer_cache['debug'] = Finder()
>>> # Add a "debug" directory to sys.path
>>> sys.path.insert(0, 'debug')
>>> import threading
Looking for threading
Looking for time
Looking for traceback
Looking for linecache
Looking for tokenize
Looking for token
>>>
```

“ debug ” sys.path

(None, [])

sys.path_importer_cache sys.path_hooks
sys.path_hooks

```
>>> sys.path_importer_cache.clear()
>>> def check_path(path):
...     print('Checking', path)
...     raise ImportError()
...
>>> sys.path_hooks.insert(0, check_path)
>>> import fib
Checked debug
Checking .
Checking /usr/local/lib/python33.zip
Checking /usr/local/lib/python3.3
Checking /usr/local/lib/python3.3/plat-darwin
Checking /usr/local/lib/python3.3/lib-dynload
Checking /Users/beazley/.local/lib/python3.3/site-packages
Checking /usr/local/lib/python3.3/site-packages
Looking for fib
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'
>>>
```

check_path() sys.path
ImportError sys.path_hooks

sys.path
URL

```
>>> def check_url(path):
...     if path.startswith('http://'):
...         return Finder()
...     else:
...         raise ImportError()
...
>>> sys.path.append('http://localhost:15000')
>>> sys.path_hooks[0] = check_url
>>> import fib
Looking for fib # Finder output!
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'

>>> # Notice installation of Finder in sys.path_importer_cache
>>> sys.path_importer_cache['http://localhost:15000']
<__main__.Finder object at 0x10064c850>
>>>
```

sys.path URL
UrlPathFinder
sys.path_importer_cache. sys.path

find_loader()
find_loader() (loader, None) loader

find_loader() (loader, path) loader
__init__.py path
__path__ URL http://localhost:15000
import grok , find_loader() path ['http://localhost:15000/grok']

find_loader()
__init__.py find_loader()
(None, path) path __init__.py
__path__ sys.path

10.5

__path__

```
>>> import xml.etree.ElementTree
>>> xml.__path__
['/usr/local/lib/python3.3/xml']
>>> xml.etree.__path__
['/usr/local/lib/python3.3/xml/etree']
>>>
```

__path__ find_loader()
__path__ sys.path_hooks

__path__ handle_url() UrlPathFinder
sys.path_importer_cache

handle_url() _get_links()
urllib.request
handle_url()

URL

```
# Check link cache
if self._links is None:
    self._links = [] # See discussion
    self._links = _get_links(self._baseurl)
```

invalidate_caches()
importlib.invalidate_caches()
URL

sys.meta_path sys.meta_path

/
sys.path

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG)
>>> import urlimport
>>> urlimport.install_path_hook()
DEBUG:urlimport:Installing handle_url
>>> import fib
DEBUG:urlimport:Handle path? /usr/local/lib/python33.zip. [No]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ImportError: No module named 'fib'
>>> import sys
>>> sys.path.append('http://localhost:15000')
>>> import fib
DEBUG:urlimport:Handle path? http://localhost:15000. [Yes]
DEBUG:urlimport:Getting links from http://localhost:15000
DEBUG:urlimport:links: {'spam.py', 'fib.py', 'grok'}
DEBUG:urlimport:find_loader: 'fib'
DEBUG:urlimport:find_loader: module 'fib' found
DEBUG:urlimport:loader: reading 'http://localhost:15000/fib.py'
DEBUG:urlimport:loader: 'http://localhost:15000/fib.py' loaded
I'm fib
>>>
```

PEP 302 importlib

12.12 10.12

12.12.1

12.12.2

10.11

```
# postimport.py
import importlib
import sys
from collections import defaultdict

_post_import_hooks = defaultdict(list)

class PostImportFinder:
    def __init__(self):
        self._skip = set()

    def find_module(self, fullname, path=None):
        if fullname in self._skip:
            return None
        self._skip.add(fullname)
        return PostImportLoader(self)

class PostImportLoader:
    def __init__(self, finder):
        self._finder = finder

    def load_module(self, fullname):
        importlib.import_module(fullname)
        module = sys.modules[fullname]
        for func in _post_import_hooks[fullname]:
            func(module)
        self._finder._skip.remove(fullname)
        return module

def when_imported(fullname):
    def decorate(func):
        if fullname in sys.modules:
            func(sys.modules[fullname])
        else:
```

```
            _post_import_hooks[fullname].append(func)
        return func
    return decorate

sys.meta_path.insert(0, PostImportFinder())
```

when_imported()

```
>>> from postimport import when_imported
>>> @when_imported('threading')
... def warn_threads(mod):
...     print('Threads? Are you crazy?')
...
>>>
>>> import threading
Threads? Are you crazy?
>>>
```

```
from functools import wraps
from postimport import when_imported

def logged(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        print('Calling', func.__name__, args, kwargs)
        return func(*args, **kwargs)
    return wrapper

# Example
@when_imported('math')
def add_logging(mod):
    mod.cos = logged(mod.cos)
    mod.sin = logged(mod.sin)
```

12.12.3

10.11

@when_imported
sys.modules
_post_import_hooks
_post_import_hooks

PostImportFinder sys.meta_path

PostImportFinder
sys.meta_path

PostImportLoader imp.import_module()
PostImportFinder

imp.import_module() _post_import_hooks

imp.reload()

sys.modules

PEP 369.

12.13 10.13

12.13.1

Python

12.13.2

Python ”~/.local/lib/python3.3/site-packages”
“ –user ”

```
python3 setup.py install --user
```

```
pip install --user packagename
```

sys.path “ site-packages ” “ site-packages ”

distribute pip

12.13.3

site-packages “ /usr/local/lib/python3.3/site-packages ” sudo

sudo

12.14 10.14 Python

12.14.1

Python
Python Python

12.14.2

pyvenv “ ” Python
Windows Scripts

```
bash % pyvenv Spam
bash %
```

pyvenv Span

```
bash % cd Spam
bash % ls
bin include lib pyvenv.cfg
bash %
```

bin Python

```
bash % Spam/bin/python3
Python 3.3.0 (default, Oct 6 2012, 15:45:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from pprint import pprint
>>> import sys
>>> pprint(sys.path)
['',
'/usr/local/lib/python33.zip',
'/usr/local/lib/python3.3',
'/usr/local/lib/python3.3/plat-darwin',
'/usr/local/lib/python3.3/lib-dynload',
'/Users/beazley/Spam/lib/python3.3/site-packages']
>>>
```

site-packages
site-packages

12.14.3

sys.path Python site-packages

distribute pip

site-packages

Python

Python

“ –system-site-packages ”

```
bash % pyvenv --system-site-packages Spam
bash %
```

pyvenv PEP 405.

12.15 10.15

12.15.1

12.15.2

```
projectname/
    README.txt
    Doc/
        documentation.txt
    projectname/
        __init__.py
        foo.py
        bar.py
        utils/
            __init__.py
            spam.py
            grok.py
    examples/
        helloworld.py
        ...
```

setup.py

```
# setup.py
from distutils.core import setup

```

'

```
setup(name='projectname',
    version='1.0',
    author='Your Name',
    author_email='you@youraddress.com',
```

CHAPTER

THIRTEEN

WEB

Python

Python

Contents:

13.1 11.1 HTTP

13.1.1

HTTP
REST API

13.1.2

urllib.request

HTTP GET

```
from urllib import request, parse

# Base URL being accessed
url = 'http://httpbin.org/get'

# Dictionary of query parameters (if any)
parms = {
   'name1' : 'value1',
   'name2' : 'value2'
}

# Encode the query string
querystring = parse.urlencode(parms)

# Make a GET request and read the response
u = request.urlopen(url+'?' + querystring)
resp = u.read()
```

POST
urlopen()

```
from urllib import request, parse

# Base URL being accessed
url = 'http://httpbin.org/post'

# Dictionary of query parameters (if any)
parms = {
   'name1' : 'value1',
   'name2' : 'value2'
}

# Encode the query string
querystring = parse.urlencode(parms)

# Make a POST request and read the response
u = request.urlopen(url, querystring.encode('ascii'))
resp = u.read()
```

HTTP user-agent
, Request
urlopen()

```
from urllib import request, parse
...

# Extra headers
headers = {
    'User-agent' : 'none/ofyourbusiness',
    'Spam' : 'Eggs'
}

req = request.Request(url, querystring.encode('ascii'), headers=headers)

# Make a request and read the response
u = request.urlopen(req)
resp = u.read()
```

requests
https://pypi.python.org/pypi/requests requests

```
import requests

# Base URL being accessed
url = 'http://httpbin.org/post'

# Dictionary of query parameters (if any)
parms = {
   'name1' : 'value1',
```

```
    'name2' : 'value2'
}

# Extra headers
headers = {
    'User-agent' : 'none/ofyourbusiness',
    'Spam' : 'Eggs'
}

resp = requests.post(url, data=parms, headers=headers)

# Decoded text returned by the request
text = resp.text
```

requests

resp.text Unicode

resp.content

resp.json JSON

requests HEAD

HTTP

```
import requests

resp = requests.head('http://www.python.org/index.html')

status = resp.status_code
last_modified = resp.headers['last-modified']
content_type = resp.headers['content-type']
content_length = resp.headers['content-length']
```

requests Pypi

```
import requests

resp = requests.get('http://pypi.python.org/pypi?:action=login',
                    auth=('user','password'))
```

requests HTTP cookies

```
import requests

# First request
resp1 = requests.get(url)
...

# Second requests with cookies received on first requests
resp2 = requests.get(url, cookies=resp1.cookies)
```

requests

```
import requests
url = 'http://httpbin.org/post'
files = { 'file': ('data.csv', open('data.csv', 'rb')) }

r = requests.post(url, files=files)
```

13.1.3

HTTP urllib
GET POST
requests

requests
http.client
HEAD

```
from http.client import HTTPConnection
from urllib import parse

c = HTTPConnection('www.python.org', 80)
c.request('HEAD', '/index.html')
resp = c.getresponse()

print('Status', resp.status)
for name, value in resp.getheaders():
    print(name, value)
```

cookies
urllib Python

```
import urllib.request

auth = urllib.request.HTTPBasicAuthHandler()
auth.add_password('pypi','http://pypi.python.org','username','password')
opener = urllib.request.build_opener(auth)

r = urllib.request.Request('http://pypi.python.org/pypi?:action=login')
u = opener.open(r)
resp = u.read()

# From here. You can access more pages using opener
...
```

requests

HTTP
cookies HTTP
httpbin http://httpbin.org
JSON

```
>>> import requests
>>> r = requests.get('http://httpbin.org/get?name=Dave&n=37',
...     headers = { 'User-agent': 'goaway/1.0' })
>>> resp = r.json
>>> resp['headers']
{'User-Agent': 'goaway/1.0', 'Content-Length': '', 'Content-Type': '',
'Accept-Encoding': 'gzip, deflate, compress', 'Connection':
'keep-alive', 'Host': 'httpbin.org', 'Accept': '*/*'}
>>> resp['args']
{'name': 'Dave', 'n': '37'}
>>>
```

httpbin.org

3

HTTP

request HTTP

OAuth requests http://docs.python-requests.org)

13.2 11.2 TCP

13.2.1

TCP

13.2.2

TCP socketserver

```
from socketserver import BaseRequestHandler, TCPServer

class EchoHandler(BaseRequestHandler):
    def handle(self):
        print('Got connection from', self.client_address)
        while True:

            msg = self.request.recv(8192)
            if not msg:
                break
            self.request.send(msg)

if __name__ == '__main__':
    serv = TCPServer(('', 20000), EchoHandler)
    serv.serve_forever()
```

handle()

request socket client_address

Python

```
>>> from socket import socket, AF_INET, SOCK_STREAM
>>> s = socket(AF_INET, SOCK_STREAM)
>>> s.connect(('localhost', 20000))
>>> s.send(b'Hello')
5
>>> s.recv(8192)
b'Hello'
>>>
```

StreamRequestHandler socket

```
from socketserver import StreamRequestHandler, TCPServer

class EchoHandler(StreamRequestHandler):
    def handle(self):
        print('Got connection from', self.client_address)
        # self.rfile is a file-like object for reading
        for line in self.rfile:
            # self.wfile is a file-like object for writing
            self.wfile.write(line)

if __name__ == '__main__':
    serv = TCPServer(('', 20000), EchoHandler)
    serv.serve_forever()
```

13.2.3

socketserver TCP

ForkingTCPServer ThreadingTCPServer

```
from socketserver import ThreadingTCPServer


if __name__ == '__main__':
    serv = ThreadingTCPServer(('', 20000), EchoHandler)
    serv.serve_forever()
```

fork

serve_forever()

```
if __name__ == '__main__':
    from threading import Thread
    NWORKERS = 16
    serv = TCPServer(('', 20000), EchoHandler)
    for n in range(NWORKERS):
        t = Thread(target=serv.serve_forever)
        t.daemon = True
        t.start()
    serv.serve_forever()
```

TCPServer socket
socket‘
bind_and_activate=False

```
if __name__ == '__main__':
    serv = TCPServer(('', 20000), EchoHandler, bind_and_activate=False)
    # Set up various socket options
    serv.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    # Bind and activate
    serv.server_bind()
    serv.server_activate()
    serv.serve_forever()
```

socket
TCPServer

```
if __name__ == '__main__':
    TCPServer.allow_reuse_address = True
    serv = TCPServer(('', 20000), EchoHandler)
    serv.serve_forever()
```

BaseRequestHandler
StreamRequestHandler StreamRequestHandler

```
import socket

class EchoHandler(StreamRequestHandler):
    # Optional settings (defaults shown)
    timeout = 5                    # Timeout on all socket operations
    rbufsize = -1                  # Read buffer size
    wbufsize = 0                   # Write buffer size
    disable_nagle_algorithm = False  # Sets TCP_NODELAY socket option
    def handle(self):
        print('Got connection from', self.client_address)
        try:
            for line in self.rfile:
                # self.wfile is a file-like object for writing
                self.wfile.write(line)
        except socket.timeout:
            print('Timed out!')
```

Python HTTP XML-RPC socketserver socket socket

```
from socket import socket, AF_INET, SOCK_STREAM

def echo_handler(address, client_sock):
    print('Got connection from {}'.format(address))
    while True:
        msg = client_sock.recv(8192)
        if not msg:
            break
        client_sock.sendall(msg)
    client_sock.close()

def echo_server(address, backlog=5):
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(address)
    sock.listen(backlog)
    while True:
        client_sock, client_addr = sock.accept()
        echo_handler(client_addr, client_sock)

if __name__ == '__main__':
    echo_server(('', 20000))
```

13.3 11.3 UDP

13.3.1

UDP

13.3.2

TCP UDP socketserver

```
from socketserver import BaseRequestHandler, UDPServer
import time

class TimeHandler(BaseRequestHandler):
    def handle(self):
        print('Got connection from', self.client_address)
        # Get message and client socket
        msg, sock = self.request
```

```
        resp = time.ctime()
        sock.sendto(resp.encode('ascii'), self.client_address)

if __name__ == '__main__':
    serv = UDPServer(('', 20000), TimeHandler)
    serv.serve_forever()
```

handle()

request socket client_address

Python

```
>>> from socket import socket, AF_INET, SOCK_DGRAM
>>> s = socket(AF_INET, SOCK_DGRAM)
>>> s.sendto(b'', ('localhost', 20000))
0
>>> s.recvfrom(8192)
(b'Wed Aug 15 20:35:08 2012', ('127.0.0.1', 20000))
>>>
```

13.3.3

UPD ( )

socket

sendto() recvfrom() send() recv()

UDP

UPD TCP

UDP

UDP

UDPServer

UDP TCP

ForkingUDPServer ThreadingUDPServer

```
from socketserver import ThreadingUDPServer

   if __name__ == '__main__':
    serv = ThreadingUDPServer(('',20000), TimeHandler)
    serv.serve_forever()
```

socket UDP

```
from socket import socket, AF_INET, SOCK_DGRAM
import time

def time_server(address):
    sock = socket(AF_INET, SOCK_DGRAM)
    sock.bind(address)
    while True:
        msg, addr = sock.recvfrom(8192)
        print('Got message from', addr)
        resp = time.ctime()
        sock.sendto(resp.encode('ascii'), addr)

if __name__ == '__main__':
    time_server(('', 20000))
```

13.4 11.4 CIDR IP

13.4.1

CIDR “123.45.67.89/27” IP “123.45.67.64”, “123.45.67.65”, ..., “123.45.67.95”)

13.4.2

ipaddress

```
>>> import ipaddress
>>> net = ipaddress.ip_network('123.45.67.64/27')
>>> net
IPv4Network('123.45.67.64/27')
>>> for a in net:
...     print(a)
...
123.45.67.64
123.45.67.65
123.45.67.66
123.45.67.67
123.45.67.68
...
123.45.67.95
>>>

>>> net6 = ipaddress.ip_network('12:3456:78:90ab:cd:ef01:23:30/125')
>>> net6
IPv6Network('12:3456:78:90ab:cd:ef01:23:30/125')
>>> for a in net6:
...     print(a)
```

```
...
12:3456:78:90ab:cd:ef01:23:30
12:3456:78:90ab:cd:ef01:23:31
12:3456:78:90ab:cd:ef01:23:32
12:3456:78:90ab:cd:ef01:23:33
12:3456:78:90ab:cd:ef01:23:34
12:3456:78:90ab:cd:ef01:23:35
12:3456:78:90ab:cd:ef01:23:36
12:3456:78:90ab:cd:ef01:23:37
>>>
```

Network

```
>>> net.num_addresses
32
>>> net[0]

IPv4Address('123.45.67.64')
>>> net[1]
IPv4Address('123.45.67.65')
>>> net[-1]
IPv4Address('123.45.67.95')
>>> net[-2]
IPv4Address('123.45.67.94')
>>>
```

```
>>> a = ipaddress.ip_address('123.45.67.69')
>>> a in net
True
>>> b = ipaddress.ip_address('123.45.67.123')
>>> b in net
False
>>>
```

IP IP

```
>>> inet = ipaddress.ip_interface('123.45.67.73/27')
>>> inet.network
IPv4Network('123.45.67.64/27')
>>> inet.ip
IPv4Address('123.45.67.73')
>>>
```

13.4.3

ipaddress IP

ipaddress socket
IPv4Address

str()

```
>>> a = ipaddress.ip_address('127.0.0.1')
>>> from socket import socket, AF_INET, SOCK_STREAM
>>> s = socket(AF_INET, SOCK_STREAM)
>>> s.connect((a, 8080))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'IPv4Address' object to str implicitly
>>> s.connect((str(a), 8080))
>>>
```

An Introduction to the ipaddress Module

13.5 11.5

```
        self.pathmap[method.lower(), path] = function
        return function
```

```
import time

_hello_resp = '''\
<html>
  <head>
     <title>Hello {name}</title>
   </head>
   <body>
     <h1>Hello {name}!</h1>
   </body>
</html>'''

def hello_world(environ, start_response):
    start_response('200 OK', [ ('Content-type','text/html')])
    params = environ['params']
    resp = _hello_resp.format(name=params.get('name'))
    yield resp.encode('utf-8')

_localtime_resp = '''\
<?xml version="1.0"?>
<time>
  <year>{t.tm_year}</year>
  <month>{t.tm_mon}</month>
  <day>{t.tm_mday}</day>
  <hour>{t.tm_hour}</hour>
  <minute>{t.tm_min}</minute>
  <second>{t.tm_sec}</second>
</time>'''

def localtime(environ, start_response):
    start_response('200 OK', [ ('Content-type', 'application/xml') ])
    resp = _localtime_resp.format(t=time.localtime())
    yield resp.encode('utf-8')

if __name__ == '__main__':
    from resty import PathDispatcher
    from wsgiref.simple_server import make_server

    # Create the dispatcher and register functions
    dispatcher = PathDispatcher()
    dispatcher.register('GET', '/hello', hello_world)
    dispatcher.register('GET', '/localtime', localtime)

    # Launch a basic server
    httpd = make_server('', 8080, dispatcher)
    print('Serving on port 8080...')
```

```
    httpd.serve_forever()
```

urllib

```
>>> u = urlopen('http://localhost:8080/hello?name=Guido')
>>> print(u.read().decode('utf-8'))
<html>
  <head>
     <title>Hello Guido</title>
   </head>
   <body>
     <h1>Hello Guido!</h1>
   </body>
</html>

>>> u = urlopen('http://localhost:8080/localtime')
>>> print(u.read().decode('utf-8'))
<?xml version="1.0"?>
<time>
  <year>2012</year>
  <month>11</month>
  <day>24</day>
  <hour>14</hour>
  <minute>49</minute>
  <second>17</second>
</time>
>>>
```

13.5.3

REST HTTP

XML

JSON CSV API

REST API

REST REST

REST API

Javascript, Android, iOS

REST Python WSGI

WSGI web

WSGI

```
import cgi
```

```
def wsgi_app(environ, start_response):
    pass
```

environ web Apache[Internet RFC 3875]
CGI

```
def wsgi_app(environ, start_response):
    method = environ['REQUEST_METHOD']
    path = environ['PATH_INFO']
    # Parse the query parameters
    params = cgi.FieldStorage(environ['wsgi.input'], environ=environ)
```

environ['REQUEST_METHOD']
GET POST HEAD environ['PATH_INFO']
cgi.FieldStorage()

start_response
HTTP (,)
HTTP

```
def wsgi_app(environ, start_response):
    pass
    start_response('200 OK', [('Content-type', 'text/plain')])
```

WSGI

```
def wsgi_app(environ, start_response):
    pass
    start_response('200 OK', [('Content-type', 'text/plain')])
    resp = []
    resp.append(b'Hello World\n')
    resp.append(b'Goodbye!\n')
    return resp
```

yield

```
def wsgi_app(environ, start_response):
    pass
    start_response('200 OK', [('Content-type', 'text/plain')])
    yield b'Hello World\n'
    yield b'Goodbye!\n'
```

WSGI
__call__()

```
class WSGIApplication:
    def __init__(self):
        ...
    def __call__(self, environ, start_response)
       ...
```

PathDispatcher

(,)

environ['params']

WSGI

start_response()

print() XML

( )

WSGI web

```
if __name__ == '__main__':
    from wsgiref.simple_server import make_server

    # Create the dispatcher and register functions
    dispatcher = PathDispatcher()
    pass

    # Launch a basic server
    httpd = make_server('', 8080, dispatcher)
    print('Serving on port 8080...')
    httpd.serve_forever()
```

WSGI
cookies

WebOb Paste

13.6 11.6 XML-RPC

13.6.1

Python

13.6.2

XML-RPC

-

```
from xmlrpc.server import SimpleXMLRPCServer

class KeyValueServer:
    _rpc_methods_ = ['get', 'set', 'delete', 'exists', 'keys']
    def __init__(self, address):
        self._data = {}
        self._serv = SimpleXMLRPCServer(address, allow_none=True)
        for name in self._rpc_methods_:
            self._serv.register_function(getattr(self, name))

    def get(self, name):
        return self._data[name]

    def set(self, name, value):
        self._data[name] = value

    def delete(self, name):
        del self._data[name]

    def exists(self, name):
        return name in self._data

    def keys(self):
        return list(self._data)

    def serve_forever(self):
        self._serv.serve_forever()

# Example
if __name__ == '__main__':
    kvserv = KeyValueServer(('', 15000))
    kvserv.serve_forever()
```

```
>>> from xmlrpc.client import ServerProxy
>>> s = ServerProxy('http://localhost:15000', allow_none=True)
```

```
>>> s.set('foo', 'bar')
>>> s.set('spam', [1, 2, 3])
>>> s.keys()
['spam', 'foo']
>>> s.get('foo')
'bar'
>>> s.get('spam')
[1, 2, 3]
>>> s.delete('spam')
>>> s.exists('spam')
False
>>>
```

13.6.3

XML-RPC

register_function()

serve_forever()

```
from xmlrpc.server import SimpleXMLRPCServer
def add(x,y):
    return x+y

serv = SimpleXMLRPCServer(('', 15000))
serv.register_function(add)
serv.serve_forever()
```

XML-RPC

XML-RPC

```
>>> class Point:
...     def __init__(self, x, y):
...             self.x = x
...             self.y = y
...
>>> p = Point(2, 3)
>>> s.set('foo', p)
>>> s.get('foo')
{'x': 2, 'y': 3}
>>>
```

```
>>> s.set('foo', b'Hello World')
>>> s.get('foo')
<xmlrpc.client.Binary object at 0x10131d410>

>>> _.data
```

```
b'Hello World'
>>>
```

XML-RPC API

XML-RPC SimpleXMLRPCServer
11.2
XML-RPC XML

XML-RPC

13.7 11.7 Python

13.7.1

Python

13.7.2

multiprocessing.connection

```
from multiprocessing.connection import Listener
import traceback

def echo_client(conn):
    try:
        while True:
            msg = conn.recv()
            conn.send(msg)
    except EOFError:
        print('Connection closed')

def echo_server(address, authkey):
    serv = Listener(address, authkey=authkey)
    while True:
        try:
            client = serv.accept()

            echo_client(client)
        except Exception:
            traceback.print_exc()
```

```
echo_server(('', 25000), authkey=b'peekaboo')
```

```
>>> from multiprocessing.connection import Client
>>> c = Client(('localhost', 25000), authkey=b'peekaboo')
>>> c.send('hello')
>>> c.recv()
'hello'
>>> c.send(42)
>>> c.recv()
42
>>> c.send([1, 2, 3, 4, 5])
>>> c.recv()
[1, 2, 3, 4, 5]
>>>
```

socket send() recv() pickle pickle

13.7.3

ZeroMQ Celery socket multiprocessing.connection

Unix Windows UNIX

```
s = Listener('/tmp/myconn', authkey=b'peekaboo')
```

Windows

```
s = Listener(r'\\.\pipe\myconn', authkey=b'peekaboo')
```

multiprocessing Client() Listener() authkey

I/O socket

13.8 11.8

13.8.1

sockets multiprocessing connections ZeroMQ
RPC

13.8.2

pickle pickle
RPC PRC

```
# rpcserver.py

import pickle
class RPCHandler:
    def __init__(self):
        self._functions = { }

    def register_function(self, func):
        self._functions[func.__name__] = func

    def handle_connection(self, connection):
        try:
            while True:
                # Receive a message
                func_name, args, kwargs = pickle.loads(connection.recv())
                # Run the RPC and send a response
                try:
                    r = self._functions[func_name](*args,**kwargs)
                    connection.send(pickle.dumps(r))
                except Exception as e:
                    connection.send(pickle.dumps(e))
        except EOFError:
             pass
```

multiprocessing RPC

```
from multiprocessing.connection import Listener
from threading import Thread

def rpc_server(handler, address, authkey):
    sock = Listener(address, authkey=authkey)
    while True:
        client = sock.accept()
        t = Thread(target=handler.handle_connection, args=(client,))
        t.daemon = True
```

```
        t.start()

# Some remote functions
def add(x, y):
    return x + y

def sub(x, y):
    return x - y

# Register with a handler
handler = RPCHandler()
handler.register_function(add)
handler.register_function(sub)

# Run the server
rpc_server(handler, ('localhost', 17000), authkey=b'peekaboo')
```

RPC

```
import pickle

class RPCProxy:
    def __init__(self, connection):
        self._connection = connection
    def __getattr__(self, name):
        def do_rpc(*args, **kwargs):
            self._connection.send(pickle.dumps((name, args, kwargs)))
            result = pickle.loads(self._connection.recv())
            if isinstance(result, Exception):
                raise result
            return result
        return do_rpc
```

```
>>> from multiprocessing.connection import Client
>>> c = Client(('localhost', 17000), authkey=b'peekaboo')
>>> proxy = RPCProxy(c)
>>> proxy.add(2, 3)

5
>>> proxy.sub(2, 3)
-1
>>> proxy.sub([1, 2], 4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "rpcserver.py", line 37, in do_rpc
    raise result
TypeError: unsupported operand type(s) for -: 'list' and 'int'
>>>
```

multiprocessing pickle
pickle.dumps() pickle.loads()

13.8.3

RPCHandler RPCProxy
foo(1, 2, z=3) ,
('foo', (1, 2), {'z': 3}) pickle
RPCProxy __getattr__() do_rpc()
pickle
( ) pickle
multiprocessing
ZeroMQ RPC ZeroMQ
socket

pickle
pickle
RPC Internet

pickle JSON XML
JSON
pickle.loads() pickle.dumps() json.loads() json.dumps()

```
# jsonrpcserver.py
import json

class RPCHandler:
    def __init__(self):
        self._functions = { }

    def register_function(self, func):
        self._functions[func.__name__] = func

    def handle_connection(self, connection):
        try:
            while True:
                # Receive a message
                func_name, args, kwargs = json.loads(connection.recv())
                # Run the RPC and send a response
                try:
                    r = self._functions[func_name](*args,**kwargs)
                    connection.send(json.dumps(r))
                except Exception as e:
                    connection.send(json.dumps(str(e)))
        except EOFError:
             pass

# jsonrpcclient.py
import json
```

```
class RPCProxy:
    def __init__(self, connection):
        self._connection = connection
    def __getattr__(self, name):
        def do_rpc(*args, **kwargs):
            self._connection.send(json.dumps((name, args, kwargs)))
            result = json.loads(self._connection.recv())
            return result
        return do_rpc
```

RPC

pickle

JSON

RPC XML-RPC
SimpleXMLRPCServer ServerProxy 11.6

13.9 11.9

13.9.1

SSL

13.9.2

hmac

```
import hmac
import os

def client_authenticate(connection, secret_key):
    '''
    Authenticate client to a remote service.
    connection represents a network connection.
    secret_key is a key known only to both client/server.
    '''
    message = connection.recv(32)
    hash = hmac.new(secret_key, message)
    digest = hash.digest()
    connection.send(digest)

def server_authenticate(connection, secret_key):
```

```
    '''
    Request client authentication.
    '''
    message = os.urandom(32)
    connection.send(message)
    hash = hmac.new(secret_key, message)
    digest = hash.digest()
    response = connection.recv(len(digest))
    return hmac.compare_digest(digest,response)
```

os.urandom() hmac

hmac.compare_digest()
==
sockets

```
from socket import socket, AF_INET, SOCK_STREAM

secret_key = b'peekaboo'
def echo_handler(client_sock):
    if not server_authenticate(client_sock, secret_key):
        client_sock.close()
        return
    while True:

        msg = client_sock.recv(8192)
        if not msg:
            break
        client_sock.sendall(msg)

def echo_server(address):
    s = socket(AF_INET, SOCK_STREAM)
    s.bind(address)
    s.listen(5)
    while True:
        c,a = s.accept()
        echo_handler(c)

echo_server(('', 18000))

Within a client, you would do this:

from socket import socket, AF_INET, SOCK_STREAM

secret_key = b'peekaboo'

s = socket(AF_INET, SOCK_STREAM)
s.connect(('localhost', 18000))
client_authenticate(s, secret_key)
```

```
s.send(b'Hello World')
resp = s.recv(1024)
```

13.9.3

hmac

hmac multiprocessing

hmac MD5 SHA-1 IETF RFC 2104

13.10 11.10 SSL

13.10.1

sockets SSL

13.10.2

ssl socket SSL ssl.wrap_socket()
socket SSL

```
from socket import socket, AF_INET, SOCK_STREAM
import ssl

KEYFILE = 'server_key.pem'   # Private key of the server
CERTFILE = 'server_cert.pem' # Server certificate (given to client)

def echo_client(s):
    while True:
        data = s.recv(8192)
        if data == b'':
            break
        s.send(data)
    s.close()
    print('Connection closed')
```

```
def echo_server(address):
    s = socket(AF_INET, SOCK_STREAM)
    s.bind(address)
    s.listen(1)

    # Wrap with an SSL layer requiring client certs
    s_ssl = ssl.wrap_socket(s,
                            keyfile=KEYFILE,
                            certfile=CERTFILE,
                            server_side=True
                            )
    # Wait for connections
    while True:
        try:
            c,a = s_ssl.accept()
            print('Got connection', c, a)
            echo_client(c)
        except Exception as e:
            print('{}: {}'.format(e.__class__.__name__, e))

echo_server(('', 20000))
```

```
>>> from socket import socket, AF_INET, SOCK_STREAM
>>> import ssl
>>> s = socket(AF_INET, SOCK_STREAM)
>>> s_ssl = ssl.wrap_socket(s,
                cert_reqs=ssl.CERT_REQUIRED,
                ca_certs = 'server_cert.pem')
>>> s_ssl.connect(('localhost', 20000))
>>> s_ssl.send(b'Hello World?')
12
>>> s_ssl.recv(8192)
b'Hello World?'
>>>
```

socket

HTTP XML-RPC

socketserver
SSL

mixin SSL

```
import ssl

class SSLMixin:
'''
Mixin class that adds support for SSL to existing servers based
on the socketserver module.
'''
```

```
def __init__(self, *args,
             keyfile=None, certfile=None, ca_certs=None,
             cert_reqs=ssl.NONE,
             **kwargs):
    self._keyfile = keyfile
    self._certfile = certfile
    self._ca_certs = ca_certs
    self._cert_reqs = cert_reqs
    super().__init__(*args, **kwargs)

def get_request(self):
    client, addr = super().get_request()
    client_ssl = ssl.wrap_socket(client,
                                 keyfile = self._keyfile,
                                 certfile = self._certfile,
                                 ca_certs = self._ca_certs,
                                 cert_reqs = self._cert_reqs,
                                 server_side = True)
    return client_ssl, addr
```

mixin
SSL XML-RPC

```
# XML-RPC server with SSL

from xmlrpc.server import SimpleXMLRPCServer

class SSLSimpleXMLRPCServer(SSLMixin, SimpleXMLRPCServer):
    pass

Here's the XML-RPC server from Recipe 11.6 modified only slightly to use SSL:

import ssl
from xmlrpc.server import SimpleXMLRPCServer
from sslmixin import SSLMixin

class SSLSimpleXMLRPCServer(SSLMixin, SimpleXMLRPCServer):
    pass

class KeyValueServer:
    _rpc_methods_ = ['get', 'set', 'delete', 'exists', 'keys']
    def __init__(self, *args, **kwargs):
        self._data = {}
        self._serv = SSLSimpleXMLRPCServer(*args, allow_none=True, **kwargs)
        for name in self._rpc_methods_:
            self._serv.register_function(getattr(self, name))

    def get(self, name):
        return self._data[name]

    def set(self, name, value):
```

```
        self._data[name] = value

    def delete(self, name):
        del self._data[name]

    def exists(self, name):
        return name in self._data

    def keys(self):
        return list(self._data)

    def serve_forever(self):
        self._serv.serve_forever()

if __name__ == '__main__':
    KEYFILE='server_key.pem'    # Private key of the server
    CERTFILE='server_cert.pem'  # Server certificate
    kvserv = KeyValueServer(('', 15000),
                            keyfile=KEYFILE,
                            certfile=CERTFILE),
    kvserv.serve_forever()
```

xmlrpc.client

URL https:

```
>>> from xmlrpc.client import ServerProxy
>>> s = ServerProxy('https://localhost:15000', allow_none=True)
>>> s.set('foo','bar')
>>> s.set('spam', [1, 2, 3])
>>> s.keys()
['spam', 'foo']
>>> s.get('foo')
'bar'
>>> s.get('spam')
[1, 2, 3]
>>> s.delete('spam')
>>> s.exists('spam')
False
>>>
```

SSL

XML-RPC

```
from xmlrpc.client import SafeTransport, ServerProxy
import ssl

class VerifyCertSafeTransport(SafeTransport):
    def __init__(self, cafile, certfile=None, keyfile=None):
        SafeTransport.__init__(self)
```

```
        self._ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLSv1)
        self._ssl_context.load_verify_locations(cafile)
        if cert:
            self._ssl_context.load_cert_chain(certfile, keyfile)
        self._ssl_context.verify_mode = ssl.CERT_REQUIRED

    def make_connection(self, host):
        # Items in the passed dictionary are passed as keyword
        # arguments to the http.client.HTTPSConnection() constructor.
        # The context argument allows an ssl.SSLContext instance to
        # be passed with information about the SSL configuration
        s = super().make_connection((host, {'context': self._ssl_context}))

        return s

# Create the client proxy
s = ServerProxy('https://localhost:15000',
                transport=VerifyCertSafeTransport('server_cert.pem'),
                allow_none=True)
```

```
if __name__ == '__main__':
    KEYFILE='server_key.pem'   # Private key of the server
    CERTFILE='server_cert.pem' # Server certificate
    CA_CERTS='client_cert.pem' # Certificates of accepted clients

    kvserv = KeyValueServer(('', 15000),
                            keyfile=KEYFILE,
                            certfile=CERTFILE,
                            ca_certs=CA_CERTS,
                            cert_reqs=ssl.CERT_REQUIRED,
                            )
    kvserv.serve_forever()
```

XML-RPC ServerProxy

```
# Create the client proxy
s = ServerProxy('https://localhost:15000',
                transport=VerifyCertSafeTransport('server_cert.pem',
                                                  'client_cert.pem',
                                                  'client_key.pem'),
                allow_none=True)
```

13.10.3

SSL

key

SSL

Verisign Equifax

web

HTTPS

bash % openssl req -new -x509 -days 365 -nodes -out server_cert.pem -keyout server_key.pem

Generating a 1024 bit RSA private key .....+++++++ ...+++++++

writing new private key to 'server_key.pem'

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]:US State or Province Name (full name) [Some-State]:Illinois Locality Name (eg, city) []:Chicago Organization Name (eg, company) [Internet Widgits Pty Ltd]:Dabeaz, LLC Organizational Unit Name (eg, section) []: Common Name (eg, YOUR name) []:localhost Email Address []: bash %

" Common Name "

DNS " localhost "

— BEGIN RSA PRIVATE KEY — MIICXQIBAAKBgQCZrCN-
LoEyAKF +f9UNcFaz5Osa6jf7qkbUl8si5xQrY3ZYC7juu nL1dZLn/ VbE-
FIITaUOgvBtPv1qUWTJGwga62VSG1oFE0ODIx3g2Nh4sRf +rySsx2
L4442nx0z4O5vJQ7k6eRNHAZUUnCL50+YvjyLyt7ryLSjSuKhCcJsbZgPwIDAQAB
AoGAB5evrr7eyL4160tM5rHTeATlaLY3UBOe5Z8XN8Z6gLiB/
ucSX9AysviVD/6F 3oD6z2aL8jbeJc1vHqjt0dC2dwwm32vVl8mRdyoAsQpWmiqXrkvP4Bsl04Vpl
Qt8xNSW9SFhceL3LEvw9M8i9MV39viih1ILyH8OuHdvJyFECQQDLEjl2d2ppxND9
PoLqVFAirDfX2JnLTdWbc+M11a9Jdn3hKF8TcxfEnFVs5Gav1MusicY5KB0ylYPb
YbTvqKc7AkEAwbnRBO2VYEZsJZp2X0IZqP9ovWokkpYx
+PE4+c6MySDgaMcigL7v WDIHJG1CHudD09GbqENasDzyb2HAIW4CzQJBAKDdkv
+xoW6gJx42Auc2WzTcUHCA eXR/ +BLpPrhKykzb-
vOQ8YvS5W764SUO1u1LWs3G +wnRMvrRvlM-
CZKgggBjkCQQCG Jewto2+a +WkOKQXrNNSc-
CDE5aPTmZQc5waCYq4UmCZQcOjkUOiN3ST1U5iuxRqfb V/ yX6fw0qh
+fLWtkOs/ JAkA +okMSxZwqRtfgOFGBfwQ8/ iKrnizeanTQ3L6scFXI
CHZXdJ3XQ6qUmNxNn7iJ7S/ LDawo1QfWkCfD9FYoxBlg — END RSA

PRIVATE KEY——

server_cert.pem

—— BEGIN CERTIFICATE —— MIIC +DCCAmGgAwIBAgIJAPMd
+vi45js3MA0GCSqGSIb3DQEBBQUAMFwxCzAJBgNV BAYTAlVTM-
REwDwYDVQQIEwhJbGxpbm9pczEQMA4GA1UEBxMHQ2hpY2FnbzEUMBIG
A1UEChMLRGFiZWF6LCBMTEMxEjAQBgNVBAMTCWxvY2FsaG9zdDAeFw0xMzAxMTEx
ODQyMjdaFw0xNDAxMTExODQyMjdaMFwxCzAJBgNVBAYTAlVTMREwDwYDVQQIEwh.
bGxpbm9pczEQMA4GA1UEBxMHQ2hpY2FnbzEUMBIGA1UEChMLRGFiZWF6LCBMTEMx
EjAQBgNVBAMTCWxvY2FsaG9zdDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA
mawjS6BMgChfn/VDXBWs+TrGuo3+6pG1JfLIucUK2N2WAu47rpy9XWS5/1WxBSCE
2lDoLwbT79alFkyRsIGutlUhtaBRNDgyMd4NjYeLEX/ q8krMdi
+OONp8dM+DubyU

O5OnkTRwGVFJwi+dPmL48i8re68i0o0rioQnCbG2YD8CAwEAAaOBwTCBvjAdBgNV
HQ4EFgQUrtoLHHgXiDZTr26NMmgKJLJLFtIwgY4GA1UdIwSBhjCBg4AUrtoLHHgX
iDZTr26NMmgKJLJLFtKhYKReMFwxCzAJBgNVBAYTAlVTMREwDwYDVQQIEwhJbGxp
bm9pczEQMA4GA1UEBxMHQ2hpY2FnbzEUMBIGA1UEChMLRGFiZWF6LCBMTEMxEjAQ
BgNVBAMTCWxvY2FsaG9zdIIJAPMd+vi45js3MAwGA1UdEwQFMAMBAf8wDQYJKoZI
hvcNAQEFBQADgYEAFci+dqvMG4xF8UTnbGVvZJPIzJDRee6Nbt6AHQo9pOdAIMAu
WsGCplSOaDNdKKzl +b2UT2Zp3AIW4Qd51bouSNnR4M/
gnr9ZD1ZctFd3jS+C5XRp D3vvcW5lAnCCC80P6rXy7d7hTeFu5EYKtRGXNvVNd/
06NALGDflrrOwxF3Y= ——END CERTIFICATE——

SSL

SSL

~ ^ _ ^

13.11 11.11 Socket

13.11.1

Python

13.11.2

Unix

Unix windows
multiprocessing

multiprocessing.reduction
send_handle() recv_handle()

```
import multiprocessing
from multiprocessing.reduction import recv_handle, send_handle
import socket

def worker(in_p, out_p):
    out_p.close()
    while True:
        fd = recv_handle(in_p)
        print('CHILD: GOT FD', fd)
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM, fileno=fd) as s:
            while True:
                msg = s.recv(1024)
                if not msg:
                    break
                print('CHILD: RECV {!r}'.format(msg))
                s.send(msg)

def server(address, in_p, out_p, worker_pid):
    in_p.close()
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind(address)
    s.listen(1)
    while True:
        client, addr = s.accept()
        print('SERVER: Got connection from', addr)
        send_handle(out_p, client.fileno(), worker_pid)
        client.close()

if __name__ == '__main__':
    c1, c2 = multiprocessing.Pipe()
    worker_p = multiprocessing.Process(target=worker, args=(c1,c2))
    worker_p.start()

    server_p = multiprocessing.Process(target=server,
                    args=(('', 15000), c1, c2, worker_p.pid))
    server_p.start()

    c1.close()
    c2.close()
```

multiprocessing

socket recv_handle()
socket

send_handle() socket

Telnet

bash % python3 passfd.py SERVER: Got connection from ('127.0.0.1', 55543)
CHILD: GOT FD 7 CHILD: RECV b'Hellorn' CHILD: RECV b'Worldrn'

socket

13.11.3

Python

send_handle() recv_handle() multiprocessing
11.7 Unix Windows

```
# servermp.py
from multiprocessing.connection import Listener
from multiprocessing.reduction import send_handle
import socket

def server(work_address, port):
    # Wait for the worker to connect
    work_serv = Listener(work_address, authkey=b'peekaboo')
    worker = work_serv.accept()
    worker_pid = worker.recv()

    # Now run a TCP/IP server and send clients to worker
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind(('', port))
    s.listen(1)
    while True:
        client, addr = s.accept()
        print('SERVER: Got connection from', addr)

        send_handle(worker, client.fileno(), worker_pid)
        client.close()

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 3:
```

```
        print('Usage: server.py server_address port', file=sys.stderr)
        raise SystemExit(1)

    server(sys.argv[1], int(sys.argv[2]))
```

python3 servermp.py /tmp/servconn 15000

```
# workermp.py

from multiprocessing.connection import Client
from multiprocessing.reduction import recv_handle
import os
from socket import socket, AF_INET, SOCK_STREAM

def worker(server_address):
    serv = Client(server_address, authkey=b'peekaboo')
    serv.send(os.getpid())
    while True:
        fd = recv_handle(serv)
        print('WORKER: GOT FD', fd)
        with socket(AF_INET, SOCK_STREAM, fileno=fd) as client:
            while True:
                msg = client.recv(1024)
                if not msg:
                    break
                print('WORKER: RECV {!r}'.format(msg))
                client.send(msg)

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 2:
        print('Usage: worker.py server_address', file=sys.stderr)
        raise SystemExit(1)

    worker(sys.argv[1])
```

python3 workermp.py /tmp/servconn .

Pipe() UNIX

sendmsg()

```
# server.py
import socket

import struct

def send_fd(sock, fd):
    '''
    Send a single file descriptor.
    '''
```

```
    sock.sendmsg([b'x'],
                 [(socket.SOL_SOCKET, socket.SCM_RIGHTS, struct.pack('i', fd))])
    ack = sock.recv(2)
    assert ack == b'OK'

def server(work_address, port):
    # Wait for the worker to connect
    work_serv = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
    work_serv.bind(work_address)
    work_serv.listen(1)
    worker, addr = work_serv.accept()

    # Now run a TCP/IP server and send clients to worker
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
    s.bind(('',port))
    s.listen(1)
    while True:
        client, addr = s.accept()
        print('SERVER: Got connection from', addr)
        send_fd(worker, client.fileno())
        client.close()

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 3:
        print('Usage: server.py server_address port', file=sys.stderr)
        raise SystemExit(1)

    server(sys.argv[1], int(sys.argv[2]))
```

```
# worker.py
import socket
import struct

def recv_fd(sock):
    '''
    Receive a single file descriptor
    '''
    msg, ancdata, flags, addr = sock.recvmsg(1,
                                     socket.CMSG_LEN(struct.calcsize('i')))

    cmsg_level, cmsg_type, cmsg_data = ancdata[0]
    assert cmsg_level == socket.SOL_SOCKET and cmsg_type == socket.SCM_RIGHTS
    sock.sendall(b'OK')

    return struct.unpack('i', cmsg_data)[0]

def worker(server_address):
```

```
    serv = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
    serv.connect(server_address)
    while True:
        fd = recv_fd(serv)
        print('WORKER: GOT FD', fd)
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM, fileno=fd) as client:
            while True:
                msg = client.recv(1024)
                if not msg:
                    break
                print('WORKER: RECV {!r}'.format(msg))
                client.send(msg)

if __name__ == '__main__':
    import sys
    if len(sys.argv) != 2:
        print('Usage: worker.py server_address', file=sys.stderr)
        raise SystemExit(1)

    worker(sys.argv[1])
```

Unix Network Programming by W. Richard Stevens (Prentice Hall, 1990) . Windows Unix multiprocessing.reduction

13.12 11.12 IO

13.12.1

I/O

13.12.2

I/O I/O socket receive

```
class EventHandler:
    def fileno(self):
        'Return the associated file descriptor'
        raise NotImplemented('must implement')

    def wants_to_receive(self):
```

```
        'Return True if receiving is allowed'
        return False

    def handle_receive(self):
        'Perform the receive operation'
        pass

    def wants_to_send(self):
        'Return True if sending is requested'
        return False

    def handle_send(self):
        'Send outgoing data'
        pass
```

```
import select

def event_loop(handlers):
    while True:
        wants_recv = [h for h in handlers if h.wants_to_receive()]
        wants_send = [h for h in handlers if h.wants_to_send()]
        can_recv, can_send, _ = select.select(wants_recv, wants_send, [])
        for h in can_recv:
            h.handle_receive()
        for h in can_send:
            h.handle_send()
```

select() ``
``select()
select() select()
handle_receive() handle_send()

EventHandler
UDP

```
import socket
import time

class UDPServer(EventHandler):
    def __init__(self, address):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind(address)

    def fileno(self):
        return self.sock.fileno()

    def wants_to_receive(self):
        return True

class UDPTimeServer(UDPServer):
```

```
    def handle_receive(self):
        msg, addr = self.sock.recvfrom(1)
        self.sock.sendto(time.ctime().encode('ascii'), addr)

class UDPEchoServer(UDPServer):
    def handle_receive(self):
        msg, addr = self.sock.recvfrom(8192)
        self.sock.sendto(msg, addr)

if __name__ == '__main__':
    handlers = [ UDPTimeServer(('',14000)), UDPEchoServer(('',15000))  ]
    event_loop(handlers)
```

Python

```
>>> from socket import *
>>> s = socket(AF_INET, SOCK_DGRAM)
>>> s.sendto(b'',('localhost',14000))
0
>>> s.recvfrom(128)
(b'Tue Sep 18 14:29:23 2012', ('127.0.0.1', 14000))
>>> s.sendto(b'Hello',('localhost',15000))
5
>>> s.recvfrom(128)
(b'Hello', ('127.0.0.1', 15000))
>>>
```

TCP

TCP

```
class TCPServer(EventHandler):
    def __init__(self, address, client_handler, handler_list):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
        self.sock.bind(address)
        self.sock.listen(1)
        self.client_handler = client_handler
        self.handler_list = handler_list

    def fileno(self):
        return self.sock.fileno()

    def wants_to_receive(self):
        return True

    def handle_receive(self):
        client, addr = self.sock.accept()
        # Add the client to the event loop's handler list
        self.handler_list.append(self.client_handler(client, self.handler_list))

class TCPClient(EventHandler):
```

```
    def __init__(self, sock, handler_list):
        self.sock = sock
        self.handler_list = handler_list
        self.outgoing = bytearray()

    def fileno(self):
        return self.sock.fileno()

    def close(self):
        self.sock.close()
        # Remove myself from the event loop's handler list
        self.handler_list.remove(self)

    def wants_to_send(self):
        return True if self.outgoing else False

    def handle_send(self):
        nsent = self.sock.send(self.outgoing)
        self.outgoing = self.outgoing[nsent:]

class TCPEchoClient(TCPClient):
    def wants_to_receive(self):
        return True

    def handle_receive(self):
        data = self.sock.recv(8192)
        if not data:
            self.close()
        else:
            self.outgoing.extend(data)

if __name__ == '__main__':
   handlers = []
   handlers.append(TCPServer(('',16000), TCPEchoClient, handlers))
   event_loop(handlers)
```

TCP

Telnet

13.12.3

socket

I/O

select() socket

I/O

concurrent.futures

```
from concurrent.futures import ThreadPoolExecutor
import os

class ThreadPoolHandler(EventHandler):
    def __init__(self, nworkers):
        if os.name == 'posix':
            self.signal_done_sock, self.done_sock = socket.socketpair()
        else:
            server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            server.bind(('127.0.0.1', 0))
            server.listen(1)
            self.signal_done_sock = socket.socket(socket.AF_INET,
                                                  socket.SOCK_STREAM)
            self.signal_done_sock.connect(server.getsockname())
            self.done_sock, _ = server.accept()
            server.close()

        self.pending = []
        self.pool = ThreadPoolExecutor(nworkers)

    def fileno(self):
        return self.done_sock.fileno()

    # Callback that executes when the thread is done
    def _complete(self, callback, r):

        self.pending.append((callback, r.result()))
        self.signal_done_sock.send(b'x')

    # Run a function in a thread pool
    def run(self, func, args=(), kwargs={},*,callback):
        r = self.pool.submit(func, *args, **kwargs)
        r.add_done_callback(lambda r: self._complete(callback, r))

    def wants_to_receive(self):
        return True

    # Run callback functions of completed work
    def handle_receive(self):
        # Invoke all pending callback functions
        for callback, result in self.pending:
            callback(result)
```

```
            self.done_sock.recv(1)
        self.pending = []
```

run()
ThreadPoolExecutor
socket
_complete() socket
fileno() socket
handle_receive()

```
# A really bad Fibonacci implementation
def fib(n):
    if n < 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

class UDPFibServer(UDPServer):
    def handle_receive(self):
        msg, addr = self.sock.recvfrom(128)
        n = int(msg)
        pool.run(fib, (n,), callback=lambda r: self.respond(r, addr))

    def respond(self, result, addr):
        self.sock.sendto(str(result).encode('ascii'), addr)

if __name__ == '__main__':
    pool = ThreadPoolHandler(16)
    handlers = [ pool, UDPFibServer(('',16000))]
    event_loop(handlers)
```

Python

```
from socket import *
sock = socket(AF_INET, SOCK_DGRAM)
for x in range(40):
    sock.sendto(str(x).encode('ascii'), ('localhost', 16000))
    resp = sock.recvfrom(8192)
    print(resp[0])
```

12.12

13.13 11.13

13.13.1

13.13.2

memoryviews

```
# zerocopy.py

def send_from(arr, dest):
    view = memoryview(arr).cast('B')
    while len(view):
        nsent = dest.send(view)
        view = view[nsent:]

def recv_into(arr, source):
    view = memoryview(arr).cast('B')
    while len(view):
        nrecv = source.recv_into(view)
        view = view[nrecv:]
```

socket

```
>>> from socket import *
>>> s = socket(AF_INET, SOCK_STREAM)
>>> s.bind(('', 25000))
>>> s.listen(1)
>>> c,a = s.accept()
>>>
```

```
>>> from socket import *
>>> c = socket(AF_INET, SOCK_STREAM)
>>> c.connect(('localhost', 25000))
>>>
```

array

numpy

```
# Server
>>> import numpy
>>> a = numpy.arange(0.0, 50000000.0)
>>> send_from(a, c)
>>>

# Client
```

```
>>> import numpy
>>> a = numpy.zeros(shape=50000000, dtype=float)
>>> a[0:10]
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
>>> recv_into(a, c)
>>> a[0:10]
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
>>>
```

13.13.3

/

——

```
view = memoryview(arr).cast('B')
```

arr

socket socket.send() send.recv_into()

sock.send()

send.recv_into()

socket

send() recv_into()

CHAPTER

FOURTEEN

, Python , , ,
. ,
.
, . ,
.

Contents:

14.1 12.1

14.1.1

/

14.1.2

`threading` Python
`Thread` target

```
# Code to execute in an independent thread
import time
def countdown(n):
    while n > 0:
        print('T-minus', n)
        n -= 1
        time.sleep(5)

# Create and launch a thread
from threading import Thread
t = Thread(target=countdown, args=(10,))
t.start()
```

start()

start()
Python
POSIX Windows

```
if t.is_alive():
    print('Still running')
else:
    print('Completed')
```

```
t.join()
```

Python

```
t = Thread(target=countdown, args=(10,), daemon=True)
t.start()
```

```
class CountdownTask:
    def __init__(self):
        self._running = True

    def terminate(self):
        self._running = False

    def run(self, n):
        while self._running and n > 0:
            print('T-minus', n)
            n -= 1
            time.sleep(5)

    c = CountdownTask()
    t = Thread(target=c.run, args=(10,))
    t.start()
    c.terminate() # Signal termination
    t.join()      # Wait for actual termination (if needed)
```

I/O

I/O

```
class IOTask:
    def terminate(self):
        self._running = False

    def run(self, sock):
        # sock is a socket
        sock.settimeout(5)        # Set timeout period
        while self._running:
            # Perform a blocking I/O operation w/ timeout
            try:
                data = sock.recv(8192)
                break
            except socket.timeout:
                continue
            # Continued processing
            ...
        # Terminated
        return
```

14.1.3

GIL Python
Python I/O
I/O

Thread

```
from threading import Thread

class CountdownThread(Thread):
    def __init__(self, n):
        super().__init__()
        self.n = 0
    def run(self):
        while self.n > 0:

            print('T-minus', self.n)
            self.n -= 1
            time.sleep(5)

c = CountdownThread(5)
c.start()
```

threading
threading

multiprocessing

```
import multiprocessing
c = CountdownTask(5)
p = multiprocessing.Process(target=c.run)
p.start()
```

```
# Wait for the thread to start
started_evt.wait()
print('countdown is running')
```

“ countdown is running ” “ countdown starting ”
event `countdown()`

14.2.3

event event
`clear()`
event event
event
event event
`Condition` `Condition`

```
import threading
import time

class PeriodicTimer:
    def __init__(self, interval):
        self._interval = interval
        self._flag = 0
        self._cv = threading.Condition()

    def start(self):
        t = threading.Thread(target=self.run)
        t.daemon = True

        t.start()

    def run(self):
        '''
        Run the timer and notify waiting threads after each interval
        '''
        while True:
            time.sleep(self._interval)
            with self._cv:
                self._flag ^= 1
                self._cv.notify_all()

    def wait_for_tick(self):
        '''
        Wait for the next tick of the timer
        '''
        with self._cv:
            last_flag = self._flag
```

```
            while last_flag == self._flag:
                self._cv.wait()

# Example use of the timer
ptimer = PeriodicTimer(5)
ptimer.start()

# Two threads that synchronize on the timer
def countdown(nticks):
    while nticks > 0:
        ptimer.wait_for_tick()
        print('T-minus', nticks)
        nticks -= 1

def countup(last):
    n = 0
    while n < last:
        ptimer.wait_for_tick()
        print('Counting', n)
        n += 1

threading.Thread(target=countdown, args=(10,)).start()
threading.Thread(target=countup, args=(5,)).start()
```

event

Condition

```
# Worker thread
def worker(n, sema):
    # Wait to be signaled
    sema.acquire()

    # Do some work
    print('Working', n)

# Create some threads
sema = threading.Semaphore(0)
nworkers = 10
for n in range(nworkers):
    t = threading.Thread(target=worker, args=(n, sema,))
    t.start()
```

```
>>> sema.release()
Working 0
>>> sema.release()
Working 1
```

```
>>>
```

Actor Actor
Actor 12.10

14.3 12.3

14.3.1

14.3.2

queue
Queue put() get()

```
from queue import Queue
from threading import Thread

# A thread that produces data
def producer(out_q):
    while True:
        # Produce some data
        ...
        out_q.put(data)

# A thread that consumes data
def consumer(in_q):
    while True:
# Get some data
        data = in_q.get()
        # Process the data
        ...

# Create the shared queue and launch both threads
q = Queue()
t1 = Thread(target=consumer, args=(q,))
t2 = Thread(target=producer, args=(q,))
t1.start()
t2.start()
```

Queue

```
from queue import Queue
from threading import Thread

# Object that signals shutdown
_sentinel = object()

# A thread that produces data
def producer(out_q):
    while running:
        # Produce some data
        ...
        out_q.put(data)

    # Put the sentinel on the queue to indicate completion
    out_q.put(_sentinel)

# A thread that consumes data
def consumer(in_q):
    while True:
        # Get some data
        data = in_q.get()

        # Check for termination
        if data is _sentinel:
            in_q.put(_sentinel)
            break

        # Process the data
        ...
```

Condition

1.5

```
import heapq
import threading

class PriorityQueue:
    def __init__(self):
        self._queue = []
        self._count = 0
        self._cv = threading.Condition()
    def put(self, item, priority):
        with self._cv:
            heapq.heappush(self._queue, (-priority, self._count, item))
            self._count += 1
            self._cv.notify()
```

```
    def get(self):
        with self._cv:
            while len(self._queue) == 0:
                self._cv.wait()
            return heapq.heappop(self._queue)[-1]
```

task_done() join()

```
from queue import Queue
from threading import Thread

# A thread that produces data
def producer(out_q):
    while running:
        # Produce some data
        ...
        out_q.put(data)

# A thread that consumes data
def consumer(in_q):
    while True:
        # Get some data
        data = in_q.get()

        # Process the data
        ...
        # Indicate completion
        in_q.task_done()

# Create the shared queue and launch both threads
q = Queue()
t1 = Thread(target=consumer, args=(q,))
t2 = Thread(target=producer, args=(q,))
t1.start()
t2.start()

# Wait for all produced items to be consumed
q.join()
```

“ ”

Event “ ”

Event

```
from queue import Queue
from threading import Thread, Event

# A thread that produces data
def producer(out_q):
    while running:
```

```
        # Produce some data
        ...
        # Make an (data, event) pair and hand it to the consumer
        evt = Event()
        out_q.put((data, evt))
        ...
        # Wait for the consumer to process the item
        evt.wait()

# A thread that consumes data
def consumer(in_q):
    while True:
        # Get some data
        data, evt = in_q.get()
        # Process the data
        ...
        # Indicate completion
        evt.set()
```

14.3.3

```
from queue import Queue
from threading import Thread
import copy

# A thread that produces data
def producer(out_q):
    while True:
        # Produce some data
        ...
        out_q.put(copy.deepcopy(data))

# A thread that consumes data
def consumer(in_q):
    while True:
        # Get some data
        data = in_q.get()
        # Process the data
        ...
```

Queue Queue
size “ ” “
” “
” “ ” “ ”

“ ”

get() put()

```
import queue
q = queue.Queue()

try:
    data = q.get(block=False)
except queue.Empty:
    ...

try:
    q.put(item, block=False)
except queue.Full:
    ...

try:
    data = q.get(timeout=5.0)
except queue.Empty:
    ...
```

put()

```
def producer(q):
    ...
    try:
        q.put(item, block=False)
    except queue.Full:
        log.warning('queued item %r discarded!', item)
```

q.get()
q.get() timeout

```
_running = True

def consumer(q):
    while _running:
        try:
            item = q.get(timeout=5.0)
            # Process item
            ...
        except queue.Empty:
            pass
```

q.qsize() q.full() q.empty()

empty()

14.4 12.4

14.4.1

14.4.2

threading Lock

```
import threading

class SharedCounter:
    '''
    A counter object that can be shared by multiple threads.
    '''
    def __init__(self, initial_value = 0):
        self._value = initial_value
        self._value_lock = threading.Lock()

    def incr(self,delta=1):
        '''
        Increment the counter with locking
        '''
        with self._value_lock:
             self._value += delta

    def decr(self,delta=1):
        '''
        Decrement the counter with locking
        '''
        with self._value_lock:
             self._value -= delta
```

Lock with
with with

14.4.3

“ ” Python

```
import threading

class SharedCounter:
    '''
    A counter object that can be shared by multiple threads.
    '''
    def __init__(self, initial_value = 0):
        self._value = initial_value
        self._value_lock = threading.Lock()

    def incr(self,delta=1):
        '''
        Increment the counter with locking
        '''
        self._value_lock.acquire()
        self._value += delta
        self._value_lock.release()

    def decr(self,delta=1):
        '''
        Decrement the counter with locking
        '''
        self._value_lock.acquire()
        self._value -= delta
        self._value_lock.release()
```

with

release() with

12.5 threading

RLoct Semaphore

RLock

SharedCounter

```
import threading

class SharedCounter:
    '''
    A counter object that can be shared by multiple threads.
    '''
    _lock = threading.RLock()
```

```
    def __init__(self, initial_value = 0):
        self._value = initial_value

    def incr(self,delta=1):
        '''
        Increment the counter with locking
        '''
        with SharedCounter._lock:
            self._value += delta

    def decr(self,delta=1):
        '''
        Decrement the counter with locking
        '''
        with SharedCounter._lock:
             self.incr(-delta)
```

decr

0 with 1 with
0
1

```
from threading import Semaphore
import urllib.request

# At most, five threads allowed to run at once
_fetch_url_sema = Semaphore(5)

def fetch_url(url):
    with _fetch_url_sema:
        return urllib.request.urlopen(url)
```

14.5 12.5

14.5.1

14.5.2

id

```
import threading
from contextlib import contextmanager

# Thread-local state to stored information on locks already acquired
_local = threading.local()

@contextmanager
def acquire(*locks):
    # Sort locks by object identifier
    locks = sorted(locks, key=lambda x: id(x))

    # Make sure lock order of previously acquired locks is not violated
    acquired = getattr(_local,'acquired',[])
    if acquired and max(id(lock) for lock in acquired) >= id(locks[0]):
        raise RuntimeError('Lock Order Violation')

    # Acquire all of the locks
    acquired.extend(locks)
    _local.acquired = acquired

    try:
        for lock in locks:
            lock.acquire()
        yield
    finally:
        # Release locks in reverse order of acquisition
        for lock in reversed(locks):
            lock.release()
        del acquired[-len(locks):]
```

acquire()

```
import threading
x_lock = threading.Lock()
y_lock = threading.Lock()

def thread_1():
    while True:
        with acquire(x_lock, y_lock):
            print('Thread-1')

def thread_2():
    while True:
        with acquire(y_lock, x_lock):
            print('Thread-2')

t1 = threading.Thread(target=thread_1)
t1.daemon = True
t1.start()

t2 = threading.Thread(target=thread_2)
t2.daemon = True
t2.start()
```

acquire() TLS

```
import threading
x_lock = threading.Lock()
y_lock = threading.Lock()

def thread_1():

    while True:
        with acquire(x_lock):
            with acquire(y_lock):
                print('Thread-1')

def thread_2():
    while True:
        with acquire(y_lock):
            with acquire(x_lock):
                print('Thread-2')

t1 = threading.Thread(target=thread_1)
t1.daemon = True
t1.start()

t2 = threading.Thread(target=thread_2)
t2.daemon = True
```

```
t2.start()
```

```
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/local/lib/python3.3/threading.py", line 639, in _bootstrap_inner
    self.run()
  File "/usr/local/lib/python3.3/threading.py", line 596, in run
    self._target(*self._args, **self._kwargs)
  File "deadlock.py", line 49, in thread_1
    with acquire(y_lock):
  File "/usr/local/lib/python3.3/contextlib.py", line 48, in __enter__
    return next(self.gen)
  File "deadlock.py", line 15, in acquire
    raise RuntimeError("Lock Order Violation")
RuntimeError: Lock Order Violation
>>>
```

acquire()

id

14.5.3

id

id

“ ”

“ ”

```
import threading

# The philosopher thread
```

```
def philosopher(left, right):
    while True:
        with acquire(left,right):
             print(threading.currentThread(), 'eating')

# The chopsticks (represented by locks)
NSTICKS = 5
chopsticks = [threading.Lock() for n in range(NSTICKS)]

# Create all of the philosophers
for n in range(NSTICKS):
    t = threading.Thread(target=philosopher,
                         args=(chopsticks[n],chopsticks[(n+1) % NSTICKS]))
    t.start()
```

acquire()

acquire

14.6 12.6

14.6.1

14.6.2

thread.local()

8.3

LazyConnection

```
from socket import socket, AF_INET, SOCK_STREAM
import threading

class LazyConnection:
    def __init__(self, address, family=AF_INET, type=SOCK_STREAM):
        self.address = address
        self.family = AF_INET
        self.type = SOCK_STREAM
        self.local = threading.local()

    def __enter__(self):
        if hasattr(self.local, 'sock'):
```

```
            raise RuntimeError('Already connected')
        self.local.sock = socket(self.family, self.type)
        self.local.sock.connect(self.address)
        return self.local.sock

    def __exit__(self, exc_ty, exc_val, tb):
        self.local.sock.close()
        del self.local.sock
```

self.local
threading.local() self.local.sock
LazyConnection

```
from functools import partial
def test(conn):
    with conn as s:
        s.send(b'GET /index.html HTTP/1.0\r\n')
        s.send(b'Host: www.python.org\r\n')

        s.send(b'\r\n')
        resp = b''.join(iter(partial(s.recv, 8192), b''))

    print('Got {} bytes'.format(len(resp)))

if __name__ == '__main__':
    conn = LazyConnection(('www.python.org', 80))

    t1 = threading.Thread(target=test, args=(conn,))
    t2 = threading.Thread(target=test, args=(conn,))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

self.local.sock

14.6.3

thread.local() LazyConnection

threading.local()

14.7 12.7

14.7.1

14.7.2

concurrent.futures ThreadPoolExecutor
TCP

```
from socket import AF_INET, SOCK_STREAM, socket
from concurrent.futures import ThreadPoolExecutor

def echo_client(sock, client_addr):
    '''
    Handle a client connection
    '''
    print('Got connection from', client_addr)
    while True:
        msg = sock.recv(65536)
        if not msg:
            break
        sock.sendall(msg)
    print('Client closed connection')
    sock.close()

def echo_server(addr):
    pool = ThreadPoolExecutor(128)
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(addr)
    sock.listen(5)
    while True:
        client_sock, client_addr = sock.accept()
        pool.submit(echo_client, client_sock, client_addr)

echo_server(('',15000))
```

Queue

```
from socket import socket, AF_INET, SOCK_STREAM
from threading import Thread
from queue import Queue
```

```
def echo_client(q):
    '''
    Handle a client connection
    '''
    sock, client_addr = q.get()
    print('Got connection from', client_addr)
    while True:
        msg = sock.recv(65536)
        if not msg:
            break
        sock.sendall(msg)
    print('Client closed connection')

    sock.close()

def echo_server(addr, nworkers):
    # Launch the client workers
    q = Queue()
    for n in range(nworkers):
        t = Thread(target=echo_client, args=(q,))
        t.daemon = True
        t.start()

    # Run the server
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(addr)
    sock.listen(5)
    while True:
        client_sock, client_addr = sock.accept()
        q.put((client_sock, client_addr))

echo_server(('',15000), 128)
```

ThreadPoolExecutor

```
from concurrent.futures import ThreadPoolExecutor
import urllib.request

def fetch_url(url):
    u = urllib.request.urlopen(url)
    data = u.read()
    return data

pool = ThreadPoolExecutor(10)
# Submit work to the pool
a = pool.submit(fetch_url, 'http://www.python.org')
b = pool.submit(fetch_url, 'http://www.pypy.org')

# Get the results back
x = a.result()
```

```
y = b.result()
```

handle
a.result()

14.7.3

```
from threading import Thread
from socket import socket, AF_INET, SOCK_STREAM

def echo_client(sock, client_addr):
    '''
    Handle a client connection
    '''
    print('Got connection from', client_addr)
    while True:
        msg = sock.recv(65536)
        if not msg:
            break
        sock.sendall(msg)
    print('Client closed connection')
    sock.close()

def echo_server(addr, nworkers):
    # Run the server
    sock = socket(AF_INET, SOCK_STREAM)
    sock.bind(addr)
    sock.listen(5)
    while True:
        client_sock, client_addr = sock.accept()
        t = Thread(target=echo_client, args=(client_sock, client_addr))
        t.daemon = True
        t.start()

echo_server(('',15000))
```

CPU ——
GIL I/O

OS X

2000 Python 9GB

8MB
Python 2000
70MB 9GB
`threading.stack_size()`

```
import threading
threading.stack_size(65536)
```

2000 Python
210MB
32768 4096 8192

14.8 12.8

14.8.1

CPU CPU

14.8.2

`concurrent.futures` `ProcessPoolExecutor`
Python
Apache web
gzip

```
logs/
   20120701.log.gz
   20120702.log.gz
   20120703.log.gz
   20120704.log.gz
   20120705.log.gz
   20120706.log.gz
   ...
```

```
124.115.6.12 - - [10/Jul/2012:00:18:50 -0500] "GET /robots.txt ..." 200 71
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET /ply/ ..." 200 11875
210.212.209.67 - - [10/Jul/2012:00:18:51 -0500] "GET /favicon.ico ..." 404 369
61.135.216.105 - - [10/Jul/2012:00:20:04 -0500] "GET /blog/atom.xml ..." 304 -
...
```

robots.txt

```
# findrobots.py

import gzip
import io
import glob

def find_robots(filename):
    '''
    Find all of the hosts that access robots.txt in a single log file
    '''
    robots = set()
    with gzip.open(filename) as f:
        for line in io.TextIOWrapper(f,encoding='ascii'):
            fields = line.split()
            if fields[6] == '/robots.txt':
                robots.add(fields[0])
    return robots

def find_all_robots(logdir):
    '''
    Find all hosts across and entire sequence of files
    '''
    files = glob.glob(logdir+'/*.log.gz')
    all_robots = set()
    for robots in map(find_robots, files):
        all_robots.update(robots)
    return all_robots

if __name__ == '__main__':
    robots = find_all_robots('logs')
    for ipaddr in robots:
        print(ipaddr)
```

map-reduce find_robots()
map find_all_robots()
all_robots CPU
—— map() concurrent.futures

```
# findrobots.py

import gzip
import io
import glob
from concurrent import futures

def find_robots(filename):
    '''
    Find all of the hosts that access robots.txt in a single log file

    '''
```

```
    robots = set()
    with gzip.open(filename) as f:
        for line in io.TextIOWrapper(f,encoding='ascii'):
            fields = line.split()
            if fields[6] == '/robots.txt':
                robots.add(fields[0])
    return robots

def find_all_robots(logdir):
    '''
    Find all hosts across and entire sequence of files
    '''
    files = glob.glob(logdir+'/*.log.gz')
    all_robots = set()
    with futures.ProcessPoolExecutor() as pool:
        for robots in pool.map(find_robots, files):
            all_robots.update(robots)
    return all_robots

if __name__ == '__main__':
    robots = find_all_robots('logs')
    for ipaddr in robots:
        print(ipaddr)
```

3.5 CPU

14.8.3

ProcessPoolExecutor

```
from concurrent.futures import ProcessPoolExecutor

with ProcessPoolExecutor() as pool:
    ...
    do work in parallel using pool
    ...
```

ProcessPoolExecutor N Python N CPU ProcessPoolExecutor(N) with

map() pool.map() :

```
# A function that performs a lot of work
def work(x):
    ...
    return result
```

```
# Nonparallel code
results = map(work, data)

# Parallel implementation
with ProcessPoolExecutor() as pool:
    results = pool.map(work, data)
```

pool.submit()

```
# Some function
def work(x):
    ...
    return result

with ProcessPoolExecutor() as pool:
    ...
    # Example of submitting work to the pool
    future_result = pool.submit(work, arg)

    # Obtaining the result (blocks until done)
    r = future_result.result()
    ...
```

Future

result()

```
def when_done(r):
    print('Got:', r.result())

with ProcessPoolExecutor() as pool:
     future_result = pool.submit(work, arg)
     future_result.add_done_callback(when_done)
```

Future

result()

-
-
- pickle
-

——

- Unix fork()

Python fork Windows
fork `pool.map()` `pool.submit()`

-

main

14.9 12.9 Python

14.9.1

GIL

14.9.2

Python C

Python GIL Python
CPU CPU

GIL GIL CPU
I/O

Python

CPU

Python

C
NumPy
PyPy JIT
Python 3

CPU
GIL
GIL CPU
C

GIL
Python `multiprocessing`

```
# Performs a large calculation (CPU bound)
def some_work(args):
    ...
```

```
    return result

# A thread that calls the above function
def some_thread():
    while True:
        ...
        r = some_work(args)
    ...
```

```
# Processing pool (see below for initiazation)
pool = None

# Performs a large calculation (CPU bound)
def some_work(args):
    ...
    return result

# A thread that calls the above function
def some_thread():
    while True:
        ...
        r = pool.apply(some_work, (args))
        ...

# Initiaze the pool
if __name__ == '__main__':
    import multiprocessing
    pool = multiprocessing.Pool()
```

GIL CPU

Python GIL
GIL
CPU

GIL C
C Python C GIL C

```
#include "Python.h"
...

PyObject *pyfunc(PyObject *self, PyObject *args) {
   ...
   Py_BEGIN_ALLOW_THREADS
   // Threaded C code
   ...
   Py_END_ALLOW_THREADS
   ...
}
```

C Cython ctypes
ctypes C GIL

14.9.3

GIL

`stalls` DNS GIL
GIL GIL
CPU I/O.

Python def Python
lambda pickle

C Python
Python C C Python
Python Python C API
C C

GIL

C

PyPy

C GIL 15.7 15.10

14.10 12.10 Actor

14.10.1

actor “ actors ”

14.10.2

actore

actor

actor actor

actor

```
from queue import Queue
from threading import Thread, Event

# Sentinel used for shutdown
class ActorExit(Exception):
    pass

class Actor:
    def __init__(self):
        self._mailbox = Queue()

    def send(self, msg):
        '''
        Send a message to the actor
        '''
        self._mailbox.put(msg)

    def recv(self):
        '''
        Receive an incoming message
        '''
        msg = self._mailbox.get()
        if msg is ActorExit:
            raise ActorExit()
        return msg

    def close(self):
        '''
        Close the actor, thus shutting it down
        '''
        self.send(ActorExit)

    def start(self):
        '''
        Start concurrent execution
        '''
        self._terminated = Event()
        t = Thread(target=self._bootstrap)

        t.daemon = True
        t.start()

    def _bootstrap(self):
        try:
            self.run()
        except ActorExit:
            pass
```

```
        finally:
            self._terminated.set()

    def join(self):
        self._terminated.wait()

    def run(self):
        '''
        Run method to be implemented by the user
        '''
        while True:
            msg = self.recv()

# Sample ActorTask
class PrintActor(Actor):
    def run(self):
        while True:
            msg = self.recv()
            print('Got:', msg)

# Sample use
p = PrintActor()
p.start()
p.send('Hello')
p.send('World')
p.close()
p.join()
```

actor send()

close() ActorExit actor
Actor run() actor
ActorExit
get()

actor

```
def print_actor():
    while True:

        try:
            msg = yield      # Get a message
            print('Got:', msg)
        except GeneratorExit:
            print('Actor terminating')

# Sample use
p = print_actor()
next(p)      # Advance to the yield (ready to receive)
p.send('Hello')
```

```
p.send('World')
p.close()
```

14.10.3

actor send()
. actor “ ”
actor

```
class TaggedActor(Actor):
    def run(self):
        while True:
            tag, *payload = self.recv()
            getattr(self,'do_'+tag)(*payload)

    # Methods correponding to different message tags
    def do_A(self, x):
        print('Running A', x)

    def do_B(self, x, y):
        print('Running B', x, y)

# Example
a = TaggedActor()
a.start()
a.send(('A', 1))      # Invokes do_A(1)
a.send(('B', 2, 3))   # Invokes do_B(2,3)
```

actor
Result

```
from threading import Event
class Result:
    def __init__(self):
        self._evt = Event()
        self._result = None

    def set_result(self, value):
        self._result = value

        self._evt.set()

    def result(self):
        self._evt.wait()
        return self._result

class Worker(Actor):
    def submit(self, func, *args, **kwargs):
        r = Result()
```

```
        self.send((func, args, kwargs, r))
        return r

    def run(self):
        while True:
            func, args, kwargs, r = self.recv()
            r.set_result(func(*args, **kwargs))

# Example use
worker = Worker()
worker.start()
r = worker.submit(pow, 2, 3)
print(r.result())
```

“ ”

actor send()

AMQP ZMQ

14.11 12.11 /

14.11.1

/

14.11.2

/ “ ” “ ”

```
from collections import defaultdict

class Exchange:
    def __init__(self):
        self._subscribers = set()

    def attach(self, task):
        self._subscribers.add(task)

    def detach(self, task):
        self._subscribers.remove(task)

    def send(self, msg):
        for subscriber in self._subscribers:
            subscriber.send(msg)
```

```
# Dictionary of all created exchanges
_exchanges = defaultdict(Exchange)

# Return the Exchange instance associated with a given name
def get_exchange(name):
    return _exchanges[name]
```

get_exchange()

Exchange

```
# Example of a task.  Any object with a send() method

class Task:
    ...
    def send(self, msg):
        ...

task_a = Task()
task_b = Task()

# Example of getting an exchange
exc = get_exchange('name')

# Examples of subscribing tasks to it
exc.attach(task_a)
exc.attach(task_b)

# Example of sending messages
exc.send('msg1')
exc.send('msg2')

# Example of unsubscribing
exc.detach(task_a)
exc.detach(task_b)
```

14.11.3

/

```
class DisplayMessages:
    def __init__(self):
        self.count = 0
    def send(self, msg):
        self.count += 1
        print('msg[{}]: {!r}'.format(self.count, msg))

exc = get_exchange('name')
d = DisplayMessages()
exc.attach(d)
```

“ task-like ”

actor 12.10 send()

```
exc = get_exchange('name')
exc.attach(some_task)
try:
    ...
finally:
    exc.detach(some_task)
```

detach()

subscribe()

```
from contextlib import contextmanager
from collections import defaultdict

class Exchange:
    def __init__(self):
        self._subscribers = set()

    def attach(self, task):
        self._subscribers.add(task)

    def detach(self, task):
        self._subscribers.remove(task)

    @contextmanager
    def subscribe(self, *tasks):
        for task in tasks:
            self.attach(task)
        try:
            yield
        finally:
```

```
            for task in tasks:
                self.detach(task)

    def send(self, msg):
        for subscriber in self._subscribers:
            subscriber.send(msg)

# Dictionary of all created exchanges
_exchanges = defaultdict(Exchange)

# Return the Exchange instance associated with a given name
def get_exchange(name):
    return _exchanges[name]

# Example of using the subscribe() method
exc = get_exchange('name')
with exc.subscribe(task_a, task_b):
     ...
     exc.send('msg1')
     exc.send('msg2')
     ...

# task_a and task_b detached here
```

14.12 12.12

14.12.1

14.12.2

yield

yield

“ ”

yield

```
# Two simple generator functions
def countdown(n):
    while n > 0:
        print('T-minus', n)
        yield
```

```
        n -= 1
    print('Blastoff!')

def countup(n):
    x = 0
    while x < n:
        print('Counting up', x)
        yield
        x += 1
```

yield

```
from collections import deque

class TaskScheduler:
    def __init__(self):
        self._task_queue = deque()

    def new_task(self, task):
        '''
        Admit a newly started task to the scheduler

        '''
        self._task_queue.append(task)

    def run(self):
        '''
        Run until there are no more tasks
        '''
        while self._task_queue:
            task = self._task_queue.popleft()
            try:
                # Run until the next yield statement
                next(task)
                self._task_queue.append(task)
            except StopIteration:
                # Generator is no longer executing
                pass

# Example use
sched = TaskScheduler()
sched.new_task(countdown(10))
sched.new_task(countdown(5))
sched.new_task(countup(15))
sched.run()
```

TaskScheduler —— yield

```
T-minus 10
T-minus 5
Counting up 0
```

```
T-minus 9
T-minus 4
Counting up 1
T-minus 8
T-minus 3
Counting up 2
T-minus 7
T-minus 2
...
```

“ ”

yield

actor

actor

```
from collections import deque

class ActorScheduler:
    def __init__(self):
        self._actors = { }          # Mapping of names to actors
        self._msg_queue = deque()   # Message queue

    def new_actor(self, name, actor):
        '''
        Admit a newly started actor to the scheduler and give it a name
        '''
        self._msg_queue.append((actor,None))
        self._actors[name] = actor

    def send(self, name, msg):
        '''
        Send a message to a named actor
        '''
        actor = self._actors.get(name)
        if actor:
            self._msg_queue.append((actor,msg))

    def run(self):
        '''
        Run as long as there are pending messages.
        '''
        while self._msg_queue:
            actor, msg = self._msg_queue.popleft()
            try:
                 actor.send(msg)
            except StopIteration:
                 pass
```

```
# Example use
if __name__ == '__main__':
    def printer():
        while True:
            msg = yield
            print('Got:', msg)

    def counter(sched):
        while True:
            # Receive the current count
            n = yield
            if n == 0:
                break
            # Send to the printer task
            sched.send('printer', n)
            # Send the next count to the counter task (recursive)

            sched.send('counter', n-1)

    sched = ActorScheduler()
    # Create the initial actors
    sched.new_actor('printer', printer())
    sched.new_actor('counter', counter(sched))

    # Send an initial message to the counter to initiate
    sched.send('counter', 10000)
    sched.run()
```

```
from collections import deque
from select import select

# This class represents a generic yield event in the scheduler
class YieldEvent:
    def handle_yield(self, sched, task):
        pass
    def handle_resume(self, sched, task):
        pass

# Task Scheduler
class Scheduler:
    def __init__(self):
        self._numtasks = 0       # Total num of tasks
        self._ready = deque()    # Tasks ready to run
        self._read_waiting = {}  # Tasks waiting to read
        self._write_waiting = {} # Tasks waiting to write
```

```
    # Poll for I/O events and restart waiting tasks
    def _iopoll(self):
        rset,wset,eset = select(self._read_waiting,
                                self._write_waiting,[])
        for r in rset:
            evt, task = self._read_waiting.pop(r)
            evt.handle_resume(self, task)
        for w in wset:
            evt, task = self._write_waiting.pop(w)
            evt.handle_resume(self, task)

    def new(self,task):
        '''
        Add a newly started task to the scheduler
        '''

        self._ready.append((task, None))
        self._numtasks += 1

    def add_ready(self, task, msg=None):
        '''
        Append an already started task to the ready queue.
        msg is what to send into the task when it resumes.
        '''
        self._ready.append((task, msg))

    # Add a task to the reading set
    def _read_wait(self, fileno, evt, task):
        self._read_waiting[fileno] = (evt, task)

    # Add a task to the write set
    def _write_wait(self, fileno, evt, task):
        self._write_waiting[fileno] = (evt, task)

    def run(self):
        '''
        Run the task scheduler until there are no tasks
        '''
        while self._numtasks:
            if not self._ready:
                self._iopoll()
            task, msg = self._ready.popleft()
            try:
                # Run the coroutine to the next yield
                r = task.send(msg)
                if isinstance(r, YieldEvent):
                    r.handle_yield(self, task)
                else:
                    raise RuntimeError('unrecognized yield event')
            except StopIteration:
```

```
                self._numtasks -= 1

# Example implementation of coroutine-based socket I/O
class ReadSocket(YieldEvent):
    def __init__(self, sock, nbytes):
        self.sock = sock
        self.nbytes = nbytes
    def handle_yield(self, sched, task):
        sched._read_wait(self.sock.fileno(), self, task)
    def handle_resume(self, sched, task):
        data = self.sock.recv(self.nbytes)
        sched.add_ready(task, data)

class WriteSocket(YieldEvent):
    def __init__(self, sock, data):
        self.sock = sock
        self.data = data
    def handle_yield(self, sched, task):

        sched._write_wait(self.sock.fileno(), self, task)
    def handle_resume(self, sched, task):
        nsent = self.sock.send(self.data)
        sched.add_ready(task, nsent)

class AcceptSocket(YieldEvent):
    def __init__(self, sock):
        self.sock = sock
    def handle_yield(self, sched, task):
        sched._read_wait(self.sock.fileno(), self, task)
    def handle_resume(self, sched, task):
        r*    (
              .add_ready(task, r)

# Wrapper around a socket object for use with yield
class Socket object):
    def __init__(self, sock):
        self._sock = sock
    def recv(self, maxbytes):
        return ReadSocket(self._sock, maxbytes)
    def send(self, data):
        return WriteSocket(self._sock, data)
    def accept(self):
        return AcceptSocket(self._sock)
    def __getattr__(self, name):
        return getattr(self._sock, name)

if __name__ == '__main__':
    from socket import socket, AF_INET, SOCK_STREAM
    import time

    # Example of a function involving generators.  This should
```

```
    # be called using line = yield from readline(sock)
    def readline(sock):
        chars = []
        while True:
            c = yield sock.recv(1)
            if not c:
                break
            chars.append(c)
            if c == b'\n':
                break
        return b''.join(chars)

    # Echo server using generators
    class EchoServer:
        def __init__(self,addr,sched):
            self.sched = sched
            sched.new(self.server_loop(addr))

        def server_loop(self,addr):
            s = Socket(socket(AF_INET,SOCK_STREAM))

            s.bind(addr)
            s.listen(5)
            while True:
                c,a = yield s.accept()
                print('Got connection from ', a)
                self.sched.new(self.client_handler(Socket(c)))

        def client_handler(self,client):
            while True:
                line = yield from readline(client)
                if not line:
                    break
                line = b'GOT:' + line
                while line:
                    nsent = yield client.send(line)
                    line = line[nsent:]
            client.close()
            print('Client closed')

    sched = Scheduler()
    EchoServer(('',16000),sched)
    sched.run()
```

I/O I/O

14.12.3

yield

```
def some_generator():
    ...
    result = yield data
    ...
```

yield “ ” yield

```
f = some_generator()

# Initial result. Is None to start since nothing has been computed
result = None
while True:
    try:
        data = f.send(result)
        result = ... do some calculation ...
    except StopIteration:
        break
```

send() yield
yield yield
send() None
yield

close()
yield GeneratorExit
throw()
yield

yield from

yield from yield from
yield from PEP 380

CPU I/O

Python

PEP 342 “
”

PEP 3156 I/O
gevent,
greenlet, Stackless Python

14.13 12.13

14.13.1

14.13.2

select()

```
import queue
import socket
import os

class PollableQueue(queue.Queue):
    def __init__(self):
        super().__init__()
        # Create a pair of connected sockets
        if os.name == 'posix':
            self._putsocket, self._getsocket = socket.socketpair()
        else:
            # Compatibility on non-POSIX systems
            server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            server.bind(('127.0.0.1', 0))
            server.listen(1)
            self._putsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self._putsocket.connect(server.getsockname())
            self._getsocket, _ = server.accept()
            server.close()

    def fileno(self):
        return self._getsocket.fileno()

    def put(self, item):
        super().put(item)
        self._putsocket.send(b'x')

    def get(self):
        self._getsocket.recv(1)
        return super().get()
```

Queue

Unix socketpair() Windows

get() put()

I/O put() get()

fileno() select() get() socket

```
import select
import threading

def consumer(queues):
    '''
    Consumer that reads data on multiple queues simultaneously
    '''
    while True:
        can_read, _, _ = select.select(queues,[],[])
        for r in can_read:
            item = r.get()
            print('Got:', item)

q1 = PollableQueue()
q2 = PollableQueue()
q3 = PollableQueue()
t = threading.Thread(target=consumer, args=([q1,q2,q3],))
t.daemon = True
t.start()

# Feed data to the queues
q1.put(1)
q2.put(10)
q3.put('hello')
q2.put(15)
...
```

14.13.3

```
import time
def consumer(queues):
    while True:
        for q in queues:
            if not q.empty():
                item = q.get()
```

```
                print('Got:', item)

        # Sleep briefly to avoid 100% CPU
        time.sleep(0.01)
```

10

```
import select

def event_loop(sockets, queues):
    while True:
        # polling with a timeout
        can_read, _, _ = select.select(sockets, [], [], 0.01)
        for r in can_read:
            handle_read(r)
        for q in queues:
            if not q.empty():
                item = q.get()
                print('Got:', item)
```

select()

I/O

14.14 12.14 Unix

14.14.1

Unix Unix

14.14.2

```
#!/usr/bin/env python3
# daemon.py

import os
import sys

import atexit
import signal
```

```
def daemonize(pidfile, *, stdin='/dev/null',
                          stdout='/dev/null',
                          stderr='/dev/null'):

    if os.path.exists(pidfile):
        raise RuntimeError('Already running')

    # First fork (detaches from parent)
    try:
        if os.fork() > 0:
            raise SystemExit(0)   # Parent exit
    except OSError as e:
        raise RuntimeError('fork #1 failed.')

    os.chdir('/')
    os.umask(0)
    os.setsid()
    # Second fork (relinquish session leadership)
    try:
        if os.fork() > 0:
            raise SystemExit(0)
    except OSError as e:
        raise RuntimeError('fork #2 failed.')

    # Flush I/O buffers
    sys.stdout.flush()
    sys.stderr.flush()

    # Replace file descriptors for stdin, stdout, and stderr
    with open(stdin, 'rb', 0) as f:
        os.dup2(f.fileno(), sys.stdin.fileno())
    with open(stdout, 'ab', 0) as f:
        os.dup2(f.fileno(), sys.stdout.fileno())
    with open(stderr, 'ab', 0) as f:
        os.dup2(f.fileno(), sys.stderr.fileno())

    # Write the PID file
    with open(pidfile,'w') as f:
        print(os.getpid(),file=f)

    # Arrange to have the PID file removed on exit/signal
    atexit.register(lambda: os.remove(pidfile))

    # Signal handler for termination (required)
    def sigterm_handler(signo, frame):
        raise SystemExit(1)

    signal.signal(signal.SIGTERM, sigterm_handler)

def main():
```

```
    import time
    sys.stdout.write('Daemon started with pid {}\n'.format(os.getpid()))
    while True:
        sys.stdout.write('Daemon Alive! {}\n'.format(time.ctime()))
        time.sleep(10)

if __name__ == '__main__':
    PIDFILE = '/tmp/daemon.pid'

    if len(sys.argv) != 2:
        print('Usage: {} [start|stop]'.format(sys.argv[0]), file=sys.stderr)
        raise SystemExit(1)

    if sys.argv[1] == 'start':
        try:
            daemonize(PIDFILE,
                      stdout='/tmp/daemon.log',
                      stderr='/tmp/dameon.log')
        except RuntimeError as e:
            print(e, file=sys.stderr)
            raise SystemExit(1)

        main()

    elif sys.argv[1] == 'stop':
        if os.path.exists(PIDFILE):
            with open(PIDFILE) as f:
                os.kill(int(f.read()), signal.SIGTERM)
        else:
            print('Not running', file=sys.stderr)
            raise SystemExit(1)

    else:
        print('Unknown command {!r}'.format(sys.argv[1]), file=sys.stderr)
        raise SystemExit(1)
```

```
bash % daemon.py start
bash % cat /tmp/daemon.pid
2882
bash % tail -f /tmp/daemon.log
Daemon started with pid 2882
Daemon Alive! Fri Oct 12 13:45:37 2012
Daemon Alive! Fri Oct 12 13:45:47 2012
...
```

pid

```
bash % daemon.py stop
bash %
```

14.14.3

daemonize()
daemonize()

```
daemonize('daemon.pid',
          stdin='/dev/null,
          stdout='/tmp/daemon.log',
          stderr='/tmp/daemon.log')
```

```
# Illegal. Must use keyword arguments
daemonize('daemon.pid',
          '/dev/null', '/tmp/daemon.log','/tmp/daemon.log')
```

os.fork()

os.setsid()

os.chdir() os.umask(0)

os.fork()

daemon

I/O

I/O

sys.stdout, sys.__stdout__ sys.stdout

sys.stdout

os.dup2() sys.stdout

sys.stdout

I/O

ID

daemonize()
atexit.register() Python
SIGTERM
SystemExit()
atexit.register()

stop

UNIX , by W. Richard Stevens and Stephen A. Rago (Addison-Wesley, 2005) C Python POSIX

CHAPTER

FIFTEEN

Python shell

5

Contents:

15.1 13.1 / /

15.1.1

15.1.2

Python `fileinput`

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

with fileinput.input() as f_input:
    for line in f_input:
        print(line, end='')
```

`filein.py`

```
$ ls | ./filein.py          # Prints a directory listing to stdout.
$ ./filein.py /etc/passwd   # Reads /etc/passwd to stdout.
$ ./filein.py < /etc/passwd # Reads /etc/passwd to stdout.
```

15.1.3

fileinput.input() FileInput

```
>>> import fileinput
>>> with fileinput.input('/etc/passwd') as f:
>>>     for line in f:
...         print(f.filename(), f.lineno(), line, end='')
...
/etc/passwd 1 ##
/etc/passwd 2 # User Database
/etc/passwd 3 #

<other output omitted>
```

FileInput

15.2 13.2

15.2.1

15.2.2

SystemExit

```
raise SystemExit('It failed!')
```

sys.stderr 1

15.2.3

```
import sys
sys.stderr.write('It failed!\n')
raise SystemExit(1)
```

SystemExit()
import sys.stderr

15.3 13.3

15.3.1

sys.argv

15.3.2

argparse

```
# search.py
'''
Hypothetical command-line tool for searching a collection of
files for one or more text patterns.
'''
import argparse
parser = argparse.ArgumentParser(description='Search some files')

parser.add_argument(dest='filenames',metavar='filename', nargs='*')

parser.add_argument('-p', '--pat',metavar='pattern', required=True,
                    dest='patterns', action='append',
                    help='text pattern to search for')

parser.add_argument('-v', dest='verbose', action='store_true',
                    help='verbose mode')

parser.add_argument('-o', dest='outfile', action='store',
                    help='output file')

parser.add_argument('--speed', dest='speed', action='store',
                    choices={'slow','fast'}, default='slow',
                    help='search speed')

args = parser.parse_args()

# Output the collected arguments
print(args.filenames)
print(args.patterns)
print(args.verbose)
print(args.outfile)
print(args.speed)
```

```
bash % python3 search.py -h
usage: search.py [-h] [-p pattern] [-v] [-o OUTFILE] [--speed {slow,fast}]
                 [filename [filename ...]]

Search some files

positional arguments:
  filename

optional arguments:
  -h, --help            show this help message and exit
  -p pattern, --pat pattern
                        text pattern to search for
  -v                    verbose mode
  -o OUTFILE            output file
  --speed {slow,fast}   search speed
```

print()

```
bash % python3 search.py foo.txt bar.txt
usage: search.py [-h] -p pattern [-v] [-o OUTFILE] [--speed {fast,slow}]
                 [filename [filename ...]]
search.py: error: the following arguments are required: -p/--pat

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt
filenames = ['foo.txt', 'bar.txt']
patterns  = ['spam', 'eggs']
verbose   = True
outfile   = None
speed     = slow

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt -o results
filenames = ['foo.txt', 'bar.txt']
patterns  = ['spam', 'eggs']
verbose   = True
outfile   = results
speed     = slow

bash % python3 search.py -v -p spam --pat=eggs foo.txt bar.txt -o results \
             --speed=fast
filenames = ['foo.txt', 'bar.txt']
patterns  = ['spam', 'eggs']
verbose   = True
outfile   = results
speed     = fast
```

print()

15.3.3

argparse

ArgumentParser
add_argument() add-argument() dest
metavar action
store ,

```
parser.add_argument(dest='filenames',metavar='filename', nargs='*')
```

Boolean

```
parser.add_argument('-v', dest='verbose', action='store_true',
                    help='verbose mode')
```

```
parser.add_argument('-o', dest='outfile', action='store',
                    help='output file')
```

required -p --pat

```
parser.add_argument('-p', '--pat',metavar='pattern', required=True,
                    dest='patterns', action='append',
                    help='text pattern to search for')
```

```
parser.add_argument('--speed', dest='speed', action='store',
                    choices={'slow','fast'}, default='slow',
                    help='search speed')
```

parser.parse() sys.argv
“add_argument()“
“dest“

sys.argv

getopt
argparse optparse
optparse argparse

15.4 13.4

15.4.1

15.4.2

Python getpass

```
import getpass

user = getpass.getuser()
passwd = getpass.getpass()

if svc_login(user, passwd):    # You must write svc_login()
   print('Yay!')
else:
   print('Boo!')
```

svc_login()

15.4.3

getpass.getuser()

shell *pwd*

input

```
user = input('Enter your username: ')
```

getpass()

Python

15.5 13.5

15.5.1

15.5.2

os.get_terminal_size()

```
>>> import os
>>> sz = os.get_terminal_size()
>>> sz
os.terminal_size(columns=80, lines=24)
>>> sz.columns
80
>>> sz.lines
24
>>>
```

15.5.3

ioctl()

15.6 13.6

15.6.1

Python

15.6.2

subprocess.check_output()

```
import subprocess
out_bytes = subprocess.check_output(['netstat','-a'])
```

```
out_text = out_bytes.decode('utf-8')
```

```
try:
    out_bytes = subprocess.check_output(['cmd','arg1','arg2'])
except subprocess.CalledProcessError as e:
    out_bytes = e.output       # Output generated before error
    code      = e.returncode   # Return code
```

check_output()
stderr

```
out_bytes = subprocess.check_output(['cmd','arg1','arg2'],
                                    stderr=subprocess.STDOUT)
```

timeout

```
try:
    out_bytes = subprocess.check_output(['cmd','arg1','arg2'], timeout=5)
except subprocess.TimeoutExpired as e:
    ...
```

shell sh bash
os.execve()
shell shell=True . Python
shell I/O

```
out_bytes = subprocess.check_output('grep python | wc > out', shell=True)
```

shell
shlex.quote()

15.6.3

check_output()

subprocess.Popen

```
import subprocess

# Some text to send
text = b'''
hello world
this is a test
goodbye
'''

# Launch a command with pipes
p = subprocess.Popen(['wc'],
          stdout = subprocess.PIPE,
          stdin = subprocess.PIPE)

# Send the data and get the output
stdout, stderr = p.communicate(text)

# To interpret as text, decode
out = stdout.decode('utf-8')
err = stderr.decode('utf-8')
```

subprocess TTY
ssh
expect pexpect

15.7 13.7

15.7.1

shell

15.7.2

shutil

```
import shutil

# Copy src to dst. (cp src dst)
shutil.copy(src, dst)

# Copy files, but preserve metadata (cp -p src dst)
shutil.copy2(src, dst)

# Copy directory tree (cp -R src dst)
shutil.copytree(src, dst)

# Move src to dst (mv src dst)
shutil.move(src, dst)
```

Unix

follow_symlinks ,

```
shutil.copytree(src, dst, symlinks=True)
```

copytree()

```
def ignore_pyc_files(dirname, filenames):
    return [name in filenames if name.endswith('.pyc')]

shutil.copytree(src, dst, ignore=ignore_pyc_files)
```

Since ignoring filename patterns is common, a utility function ignore_patterns() has already been provided to do it. For example:

shutil.copytree(src, dst, ignore=shutil.ignore_patterns(‘ ~’, ’.pyc’))

15.7.3

shutil
copy2()
ACLs fork

shutil.copytree()
os.path Unix
Windows

```
>>> filename = '/Users/guido/programs/spam.py'
>>> import os.path
>>> os.path.basename(filename)
'spam.py'
>>> os.path.dirname(filename)
'/Users/guido/programs'
>>> os.path.split(filename)
('/Users/guido/programs', 'spam.py')
>>> os.path.join('/new/dir', os.path.basename(filename))
'/new/dir/spam.py'
>>> os.path.expanduser('~/guido/programs/spam.py')
'/Users/guido/programs/spam.py'
>>>
```

copytree()

```
try:
    shutil.copytree(src, dst)
except shutil.Error as e:
    for src, dst, msg in e.args[0]:
        # src is source name
        # dst is destination name
        # msg is error message from exception
        print(dst, src, msg)
```

ignore_dangling_symlinks=True copytree()

shutil
Python documentation

>>>

15.8 13.8

15.8.1

.tar, .tgz .zip

15.8.2

shutil —— make_archive() unpack_archive()

```
>>> import
```

15.9.2

os.walk()

```
#!/usr/bin/env python3.3
import os

def findfile(start, name):
    for relpath, dirs, files in os.walk(start):
        if name in files:
            full_path = os.path.join(start, relpath, name)
            print(os.path.normpath(os.path.abspath(full_path)))

if __name__ == '__main__':
    findfile(sys.argv[1], sys.argv[2])
```

findfile.py

15.9.3

os.walk()

os.path.join() ./../foo//bar
os.path.abspath() ,
os.path.normpath()

UNIX

```
#!/usr/bin/env python3.3

import os
import time

def
.       = time time()
    for path, dirs, files in os.walk(top):
        for name in files:
            fullpath = os.path.join(path, name)
            if os.path.exists(fullpath):
                mtime = os.path.getmtime(fullpath)
                if mtime > (now - seAon0
                    print(fullpath)
```

```
if __name__ == '__main__':
    import sys
    if len(sys.argv) != 3:
        print('Usage: {} dir seconds'.format(sys.argv[0]))
        raise SystemExit(1)

    modified_within(sys.argv[1], float(sys.argv[2]))
```

os,os.path,glob

5.11 5.13

15.10 13.10

15.10.1

.ini

15.10.2

configparser

```
; config.ini
; Sample configuration file

[installation]
library=%(prefix)s/lib
include=%(prefix)s/include
bin=%(prefix)s/bin
prefix=/usr/local

# Setting related to debug configuration
[debug]
log_errors=true
show_warnings=False

[server]
port: 8080
nworkers: 32
pid-file=/tmp/spam.pid
root=/www/root
signature:
    =================================
    Brought to you by the Python Cookbook
    =================================
```

```
>>> from configparser import ConfigParser
>>> cfg = ConfigParser()
>>> cfg.read('config.ini')
['config.ini']
>>> cfg.sections()
['installation', 'debug', 'server']
>>> cfg.get('installation','library')
'/usr/local/lib'
>>> cfg.getboolean('debug','log_errors')

True
>>> cfg.getint('server','port')
8080
>>> cfg.getint('server','nworkers')
32
>>> print(cfg.get('server','signature'))

\=================================
Brought to you by the Python Cookbook
\=================================
>>>
```

cfg.write()

```
>>> cfg.set('server','port','9000')
>>> cfg.set('debug','log_errors','False')
>>> import sys
>>> cfg.write(sys.stdout)
```

```
[installation]
library = %(prefix)s/lib
include = %(prefix)s/include
bin = %(prefix)s/bin
prefix = /usr/local

[debug]
log_errors = False
show_warnings = False

[server]
port = 9000
nworkers = 32
pid-file = /tmp/spam.pid
root = /www/root
signature =
          =================================
          Brought to you by the Python Cookbook
          =================================
>>>
```

15.10.3

“ installation ” “ debug ” “ server ”

Python

```
prefix=/usr/local
prefix: /usr/local
```

```
>>> cfg.get('installation','PREFIX')
'/usr/local'
>>> cfg.get('installation','prefix')
'/usr/local'
>>>
```

getboolean()

```
log_errors = true
log_errors = TRUE
log_errors = Yes
log_errors = 1
```

Python

prefix

```
[installation]
library=%(prefix)s/lib
include=%(prefix)s/include
bin=%(prefix)s/bin
prefix=/usr/local
```

ConfigParser

```
; ~/.config.ini
[installation]
prefix=/Users/beazley/test

[debug]
log_errors=False
```

```
>>> # Previously read configuration
>>> cfg.get('installation', 'prefix')
'/usr/local'
```

```
>>> # Merge in user-specific configuration
>>> import os
>>> cfg.read(os.path.expanduser('~/.config.ini'))
['/Users/beazley/.config.ini']

>>> cfg.get('installation', 'prefix')
'/Users/beazley/test'
>>> cfg.get('installation', 'library')
'/Users/beazley/test/lib'
>>> cfg.getboolean('debug', 'log_errors')
False
>>>
```

prefix library

```
>>> cfg.get('installation','library')
'/Users/beazley/test/lib'
>>> cfg.set('installation','prefix','/tmp/dir')
>>> cfg.get('installation','library')
'/tmp/dir/lib'
>>>
```

Python .ini
windows configparser

15.11 13.11

15.11.1

15.11.2

The easiest way to add logging to simple programs is to use the logging module. For example: logging

```
import logging

def main():
    # Configure the logging system
    logging.basicConfig(
        filename='app.log',
        level=logging.ERROR
    )
```

```
    # Variables (to make the calls that follow work)
    hostname = 'www.python.org'
    item = 'spam'
    filename = 'data.csv'
    mode = 'r'

    # Example logging calls (insert into your program)
    logging.critical('Host %s unknown', hostname)
    logging.error("Couldn't find %r", item)
    logging.warning('Feature is deprecated')
    logging.info('Opening file %r, mode=%r', filename, mode)
    logging.debug('Got here')

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

critical(), error(), warning(), info(), debug()
basicConfig() level
logging
%

app.log

```
CRITICAL:root:Host www.python.org unknown
ERROR:root:Could not find 'spam'
```

If you want to change the output or level of output, you can change the parameters to the basicConfig() call. For example:
basicConfig()

```
logging.basicConfig(
     filename='app.log',
     level=logging.WARNING,
     format='%(levelname)s:%(asctime)s:%(message)s')
```

```
CRITICAL:2012-11-20 12:27:13,595:Host www.python.org unknown
ERROR:2012-11-20 12:27:13,595:Could not find 'spam'
WARNING:2012-11-20 12:27:13,595:Feature is deprecated
```

basicConfig()

```
import logging
import logging.config

def main():
    # Configure the logging system
    logging.config.fileConfig('logconfig.ini')
    ...
```

logconfig.ini

```
[loggers]
keys=root

[handlers]
keys=defaultHandler

[formatters]
keys=defaultFormatter

[logger_root]
level=INFO
handlers=defaultHandler
qualname=root

[handler_defaultHandler]
class=FileHandler
formatter=defaultFormatter
args=('app.log', 'a')

[formatter_defaultFormatter]
format=%(levelname)s:%(name)s:%(message)s
```

logconfig.ini

15.11.3

logging

basicConfig()

basicConfig()

```
logging.basicConfig(level=logging.INFO)
```

basicConfig()
root logger

```
logging.getLogger().level = logging.DEBUG
```

logging

Logging Cookbook

15.12 13.12

15.12.1

15.12.2

logger

```
# somelib.py

import logging
log = logging.getLogger(__name__)
log.addHandler(logging.NullHandler())

# Example function (for testing)
def func():
    log.critical('A Critical Error!')
    log.debug('A debug message')
```

```
>>> import somelib
>>> somelib.func()
>>>
```

```
>>> import logging
>>> logging.basicConfig()
>>> somelib.func()
CRITICAL:somelib:A Critical Error!
>>>
```

15.12.3

getLogger(__name__) logger
logger

log.addHandler(logging.NullHandler())
logger

```
>>> import logging
>>> logging.basicConfig(level=logging.ERROR)

>>> import somelib
>>> somelib.func()
CRITICAL:somelib:A Critical Error!

>>> # Change the logging level for 'somelib' only
>>> logging.getLogger('somelib').level=logging.DEBUG
>>> somelib.func()
CRITICAL:somelib:A Critical Error!
DEBUG:somelib:A debug message
>>>
```

ERROR somelib
debug

——

Logging HOWTO

15.13 13.13

15.13.1

15.13.2

time

```
import time

class Timer:
    def __init__(self, func=time.perf_counter):
        self.elapsed = 0.0
        self._func = func
        self._start = None

    def start(self):
        if self._start is not None:
            raise RuntimeError('Already started')
        self._start = self._func()
```

```
    def stop(self):
        if self._start is None:
            raise RuntimeError('Not started')
        end = self._func()
        self.elapsed += end - self._start
        self._start = None

    def reset(self):
        self.elapsed = 0.0

    @property
    def running(self):
        return self._start is not None

    def __enter__(self):
        self.start()
        return self

    def __exit__(self, *args):
        self.stop()
```

elapsed

```
def countdown(n):
    while n > 0:
        n -= 1

# Use 1: Explicit start/stop
t = Timer()
t.start()
countdown(1000000)
t.stop()
print(t.elapsed)

# Use 2: As a context manager
with t:
    countdown(1000000)

print(t.elapsed)

with Timer() as t2:
    countdown(1000000)
print(t2.elapsed)
```

15.13.3

with

time.time()

time.clock()
time.perf_counter()

Timer
CPU time.process_time()

```
t = Timer(time.process_time)
with t:
    countdown(1000000)
print(t.elapsed)
```

time.perf_counter() time.process_time()

14.13

15.14 13.14 CPU

15.14.1

Unix CPU

15.14.2

resource CPU

```
import signal
import resource
import os

def time_exceeded(signo, frame):
    print("Time's up!")
    raise SystemExit(1)

def set_max_runtime(seconds):
    # Install the signal handler and set a resource limit
    soft, hard = resource.getrlimit(resource.RLIMIT_CPU)
    resource.setrlimit(resource.RLIMIT_CPU, (seconds, hard))
    signal.signal(signal.SIGXCPU, time_exceeded)

if __name__ == '__main__':
    set_max_runtime(15)
    while True:
        pass
```

SIGXCPU

```
import resource

def limit_memory(maxsize):
    soft, hard = resource.getrlimit(resource.RLIMIT_AS)
    resource.setrlimit(resource.RLIMIT_AS, (maxsize, hard))
```

MemoryError

15.14.3

setrlimit()

setrlimit()

resource

Unix

Linux OS X

15.15 13.15 WEB

15.15.1

URL

15.15.2

webbrowser

```
>>> import webbrowser
>>> webbrowser.open('http://www.python.org')
True
>>>
```

```
>>> # Open the page in a new browser window
>>> webbrowser.open_new('http://www.python.org')
True
>>>
```

```
>>> # Open the page in a new browser tab
>>> webbrowser.open_new_tab('http://www.python.org')
True
>>>
```

webbrowser.get()

```
>>> c = webbrowser.get('firefox')
>>> c.open('http://www.python.org')
True
>>> c.open_new_tab('http://docs.python.org')
True
>>>
```

‘Python <http://docs.python.org/3/library/webbrowser.html>‘_

15.15.3

HTML
webbrowser

CHAPTER

SIXTEEN

Python

Contents:

16.1 14.1 stdout

16.1.1

sys.stdout

16.1.2

unittest.mock patch()
sys.stdout

mymodule

```
# mymodule.py

def urlprint(protocol, host, domain):
    url = '{}://{}.{}'.format(protocol, host, domain)
    print(url)
```

print sys.stdout

unittest.mock patch()

mymodule

```
from io import StringIO
from unittest import TestCase
from unittest.mock import patch
import mymodule

class TestURLPrint(TestCase):
    def test_url_gets_to_stdout(self):
        protocol = 'http'
        host = 'www'
        domain = 'example.com'
        expected_url = '{}://{}.{}\n'.format(protocol, host, domain)

        with patch('sys.stdout', new=StringIO()) as fake_out:
            mymodule.urlprint(protocol, host, domain)
            self.assertEqual(fake_out.getvalue(), expected_url)
```

16.1.3

urlprint()
expected_url

unittest.mock.patch() StringIO
sys.stdout . fake_out with
with patch
Python C sys.stdout

Python C I/O
I/O StringIO
5.6

16.2 14.2

16.2.1

16.2.2

unittest.mock.patch() patch()

```
from unittest.mock import patch
import example

@patch('example.func')
def test1(x, mock_func):
    example.func(x)       # Uses patched example.func
    mock_func.assert_called_with(x)
```

```
with patch('example.func') as mock_func:
    example.func(x)      # Uses patched example.func
    mock_func.assert_called_with(x)
```

```
p = patch('example.func')
mock_func = p.start()
example.func(x)
mock_func.assert_called_with(x)
p.stop()
```

```
@patch('example.func1')
@patch('example.func2')
@patch('example.func3')
def test1(mock1, mock2, mock3):
    ...

def test2():
    with patch('example.patch1') as mock1, \
         patch('example.patch2') as mock2, \
         patch('example.patch3') as mock3:
    ...
```

16.2.3

patch()

MagicMock

```
>>> x = 42
>>> with patch('__main__.x'):
...     print(x)
...
<MagicMock name='x' id='4314230032'>
>>> x
42
>>>
```

patch()

```
>>> x
42
>>> with patch('__main__.x', 'patched_value'):
...     print(x)
...
patched_value
>>> x
42
>>>
```

MagicMock

```
>>> from unittest.mock import MagicMock
>>> m = MagicMock(return_value = 10)
>>> m(1, 2, debug=True)
10
>>> m.assert_called_with(1, 2, debug=True)
>>> m.assert_called_with(1, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File ".../unittest/mock.py", line 726, in assert_called_with
    raise AssertionError(msg)
AssertionError: Expected call: mock(1, 2)
Actual call: mock(1, 2, debug=True)
>>>

>>> m.upper.return_value = 'HELLO'
>>> m.upper('hello')
'HELLO'
>>> assert m.upper.called

>>> m.split.return_value = ['hello', 'world']
>>> m.split('hello world')
['hello', 'world']
>>> m.split.assert_called_with('hello world')
>>>

>>> m['blah']
<MagicMock name='mock.__getitem__()' id='4314412048'>
>>> m.__getitem__.called
True
>>> m.__getitem__.assert_called_with('blah')
>>>
```

```
# example.py
from urllib.request import urlopen
```

```
import csv

def dowprices():
    u = urlopen('http://finance.yahoo.com/d/quotes.csv?s=@^DJI&f=sl1')
    lines = (line.decode('utf-8') for line in u)
    rows = (row for row in csv.reader(lines) if len(row) == 2)
    prices = { name:float(price) for name, price in rows }
    return prices
```

urlopen() Web

:

```
import unittest
from unittest.mock import patch
import io
import example

sample_data = io.BytesIO(b'''\
"IBM",91.1\r
"AA",13.25\r
"MSFT",27.72\r
\r
''')

class Tests(unittest.TestCase):
    @patch('example.urlopen', return_value=sample_data)
    def test_dowprices(self, mock_urlopen):
        p = example.dowprices()
        self.assertTrue(mock_urlopen.called)
        self.assertEqual(p,
                         {'IBM': 91.1,
                          'AA': 13.25,
                          'MSFT' : 27.72})

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

example urlopen()
ByteIO().

example.urlopen
urllib.request.urlopen
from urllib.request import urlopen ,
dowprices() urlopen() example

unittest.mock

16.3 14.3

16.3.1

16.3.2

assertRaises()

ValueError

```
import unittest

# A simple function to illustrate
def parse_int(s):
    return int(s)

class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        self.assertRaises(ValueError, parse_int, 'N/A')
```

```
import errno

class TestIO(unittest.TestCase):
    def test_file_not_found(self):
        try:
            f = open('/file/not/found')
        except IOError as e:
            self.assertEqual(e.errno, errno.ENOENT)

        else:
            self.fail('IOError not raised')
```

16.3.3

assertRaises()

```
class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        try:
            r = parse_int('N/A')
        except ValueError as e:
            self.assertEqual(type(e), ValueError)
```

```
class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        try:
            r = parse_int('N/A')
        except ValueError as e:
            self.assertEqual(type(e), ValueError)
        else:
            self.fail('ValueError not raised')
```

assertRaises()

assertRaises()
assertRaisesRegex()

```
class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        self.assertRaisesRegex(ValueError, 'invalid literal .*',
                               parse_int, 'N/A')
```

assertRaises() assertRaisesRegex()

```
class TestConversion(unittest.TestCase):
    def test_bad_int(self):
        with self.assertRaisesRegex(ValueError, 'invalid literal .*'):
            r = parse_int('N/A')
```

16.4 14.4

16.4.1

16.4.2

```
import unittest

class MyTest(unittest.TestCase):
    pass
```

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

main()

```
import sys

def main(out=sys.stderr, verbosity=2):
    loader = unittest.TestLoader()
    suite = loader.loadTestsFromModule(sys.modules[__name__])
    unittest.TextTestRunner(out,verbosity=verbosity).run(suite)

if __name__ == '__main__':
    with open('testing.out', 'w') as f:
        main(f)
```

16.4.3

unittest

unittest

unittest.TestLoader
loadTestsFromModule() TestCase

loadTestsFromTestCase() TestCase
TextTestRunner
unittest.main()

unittest
TestLoader
TextTestRunner
unittest

16.5 14.5

16.5.1

16.5.2

unittest

```
import unittest
import os
import platform

class Tests(unittest.TestCase):
    def test_0(self):
        self.assertTrue(True)

    @unittest.skip('skipped test')
    def test_1(self):
        self.fail('should have failed!')

    @unittest.skipIf(os.name=='posix', 'Not supported on Unix')
    def test_2(self):
        import winreg

    @unittest.skipUnless(platform.system() == 'Darwin', 'Mac specific test')
    def test_3(self):
        self.assertTrue(True)

    @unittest.expectedFailure
    def test_4(self):
        self.assertEqual(2+2, 5)

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

Mac

```
bash % python3 testsample.py -v
test_0 (__main__.Tests) ... ok
test_1 (__main__.Tests) ... skipped 'skipped test'
test_2 (__main__.Tests) ... skipped 'Not supported on Unix'
test_3 (__main__.Tests) ... ok
test_4 (__main__.Tests) ... expected failure

----------------------------------------------------------------------
Ran 5 tests in 0.002s

OK (skipped=2, expected failures=1)
```

16.5.3

skip() skipIf() skipUnless()

Python

@expected

```
@unittest.skipUnless(platform.system() == 'Darwin', 'Mac specific tests')
class DarwinTests(unittest.TestCase):
    pass
```

16.6 14.6

16.6.1

16.6.2

```
try:
    client_obj.get_url(url)
except (URLError, ValueError, SocketTimeout):
    client_obj.remove_url(url)
```

remove_url()
except

```
try:
    client_obj.get_url(url)
except (URLError, ValueError):
    client_obj.remove_url(url)
except SocketTimeout:
    client_obj.handle_url_timeout(url)
```

```
try:
    f = open(filename)
except (FileNotFoundError, PermissionError):
    pass
```

```
try:
    f = open(filename)
```

```
except OSError:
    pass
```

OSError FileNotFoundError PermissionError

16.6.3

as

```
try:
    f = open(filename)
except OSError as e:
    if e.errno == errno.ENOENT:
        logger.error('File not found')
    elif e.errno == errno.EACCES:
        logger.error('Permission denied')
    else:
        logger.error('Unexpected error: %d', e.errno)
```

e OSError

except
except

```
>>> f = open('missing')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'missing'
>>> try:
...     f = open('missing')
... except OSError:
...     print('It failed')
... except FileNotFoundError:
...     print('File not found')
...
It failed
>>>
```

FileNotFoundError OSError
FileNotFoundError

__mro__

```
>>> FileNotFoundError.__mro__
(<class 'FileNotFoundError'>, <class 'OSError'>, <class 'Exception'>,
 <class 'BaseException'>, <class 'object'>)
>>>
```

BaseException except

2 . b Q M @ D ;ø ePð •m i ` v

14.7

X

16.7.1

16.7.2

Exception

```
try:
   ...
except Exception as e:
   ...
   log('Reason:', e)       # Important!
```

SystemExit KeyboardInterrupt GeneratorExit
Exception BaseException

16.7.3

```
def 7
```

```
        print("Couldn't parse")
        print('Reason:', e)
```

```
>>> parse_int('42')
Couldn't parse
Reason: global name 'v' is not defined
>>>
```

16.8 14.8

16.8.1

16.8.2

——— Exception

```
class NetworkError(Exception):
    pass

class HostnameError(NetworkError):
    pass

class TimeoutError(NetworkError):
    pass

class ProtocolError(NetworkError):
    pass
```

```
try:
    msg = s.recv()
except TimeoutError as e:
    ...
except ProtocolError as e:
    ...
```

16.8.3

Exception
Exception BaseException
BaseException
KeyboardInterrupt SystemExit
BaseException

```
try:
    s.send(msg)
except ProtocolError:
    ...
```

```
try:
    s.send(msg)
except NetworkError:
    ...
```

__init__()
Exception.__init__()

```
class CustomError(Exception):
    def __init__(self, message, status):
        super().__init__(message, status)
        self.message = message
        self.status = status
```

Exception
.args . Python
.args
.args *RuntimeError'*
raise

```
>>> try:
...     raise RuntimeError('It failed')
... except RuntimeError as e:
...     print(e.args)
...
('It failed',)
>>> try:
...     raise RuntimeError('It failed', 42, 'spam')
... except RuntimeError as e:

...     print(e.args)
```

```
...
('It failed', 42, 'spam')
>>>
```

‘Python <https://docs.python.org/3/tutorial/errors.html>‘_

16.9 14.9

16.9.1

16.9.2

raise from raise

```
>>> def example():
...     try:
...             int('N/A')
...     except ValueError as e:
...             raise RuntimeError('A parsing error occurred') from e
>>>
example()
Traceback (most recent call last):
  File "<stdin>", line 3, in example
ValueError: invalid literal for int() with base 10: 'N/A'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in example
RuntimeError: A parsing error occurred
>>>
```

except __cause__

```
try:
    example()
except RuntimeError as e:
    print("It didn't work:", e)
```

```
    if e.__cause__:
        print('Cause:', e.__cause__)
```

except

```
>>> def example2():
...     try:
...             int('N/A')
...     except ValueError as e:
...             print("Couldn't parse:", err)
...
>>>
>>> example2()
Traceback (most recent call last):
  File "<stdin>", line 3, in example2
ValueError: invalid literal for int() with base 10: 'N/A'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in example2
NameError: global name 'err' is not defined
>>>
```

NameError

raise from None :

```
>>> def example3():
...     try:
...             int('N/A')
...     except ValueError:
...             raise RuntimeError('A parsing error occurred') from None...
>>>
example3()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in example3
RuntimeError: A parsing error occurred
>>>
```

16.9.3

except raise
raise raise from

```
try:
   ...
except SomeException as e:
   raise DifferentException() from e
```

DifferentException

SomeException

```
try:
   ...
except SomeException:
   raise DifferentException()
```

raise from

16.10 14.10

16.10.1

except

16.10.2

rasie

```
>>> def example():
...     try:
...             int('N/A')
...     except ValueError:
...             print("Didn't work")
...             raise
...

>>> example()
Didn't work
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in example
ValueError: invalid literal for int() with base 10: 'N/A'
>>>
```

16.10.3

```
try:
    ...
except Exception as e:
    # Process exception information in some way
    ...

    # Propagate the exception
    raise
```

16.11 14.11

16.11.1

16.11.2

warning.warn()

```
import warnings

def func(x, y, logfile=None, debug=False):
    if logfile is not None:
         warnings.warn('logfile argument deprecated', DeprecationWarning)
    ...
```

warn() UserWarning, DeprecationWarning, SyntaxWarning, RuntimeWarning, ResourceWarning, FutureWarning.

-W

all Python

```
bash % python3 -W all example.py
example.py:5: DeprecationWarning: logfile argument is deprecated
  warnings.warn('logfile argument is deprecated', DeprecationWarning)
```

-W

error

```
bash % python3 -W error example.py
Traceback (most recent call last):
  File "example.py", line 10, in <module>
```

```
    func(2, 3, logfile='log.txt')
  File "example.py", line 5, in func
    warnings.warn('logfile argument is deprecated', DeprecationWarning)
DeprecationWarning: logfile argument is deprecated
bash %
```

16.11.3

```
>>> import warnings
>>> warnings.simplefilter('always')
>>> f = open('/etc/passwd')
>>> del f
__main__:1: ResourceWarning: unclosed file <_io.TextIOWrapper name='/etc/passwd'
 mode='r' encoding='UTF-8'>
>>>
```

-W -W
all -W ignore -W error
warnings.simplefilter() always
`ignore error

warnings
Python

16.12 14.12

16.12.1

16.12.2

python3 -i someprogram.py
-i shell

```
# sample.py

def func(n):
    return n + 10

func('Hello')
```

python3 -i sample.py

```
bash % python3 -i sample.py
Traceback (most recent call last):
  File "sample.py", line 6, in <module>
    func('Hello')
  File "sample.py", line 4, in func
    return n + 10
TypeError: Can't convert 'int' object to str implicitly
>>> func(10)
20
>>>
```

Python

```
>>> import pdb
>>> pdb.pm()
> sample.py(4)func()
-> return n + 10
(Pdb) w
  sample.py(6)<module>()
-> func('Hello')
> sample.py(4)func()
-> return n + 10
(Pdb) print n
'Hello'
(Pdb) q
>>>
```

shell

```
import traceback
import sys

try:
    func(arg)
except:
    print('**** AN ERROR OCCURRED ****')
    traceback.print_exc(file=sys.stderr)
```

print()

traceback.print_stack()

```
>>> def sample(n):
...     if n > 0:
...             sample(n-1)
...     else:
...             traceback.print_stack(file=sys.stderr)
...
>>> sample(5)
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in sample
  File "<stdin>", line 3, in sample
  File "<stdin>", line 3, in sample
  File "<stdin>", line 3, in sample
  File "<stdin>", line 3, in sample
  File "<stdin>", line 5, in sample
>>>
```

pdb.set_trace()

```
import pdb

def func(arg):
    ...
    pdb.set_trace()
    ...
```

print w

16.12.3

print()

pdb.set_trace()

set_trace()

IDE Python IDE pdb
IDE

16.13 14.13

16.13.1

16.13.2

Unix

```
bash % time python3 someprogram.py
real 0m13.937s
user 0m12.162s
sys  0m0.098s
bash %
```

cProfile

```
bash % python3 -m cProfile someprogram.py
         859647 function calls in 16.016 CPU seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   263169    0.080    0.000    0.080    0.000 someprogram.py:16(frange)
      513    0.001    0.000    0.002    0.000 someprogram.py:30(generate_mandel)
   262656    0.194    0.000   15.295    0.000 someprogram.py:32(<genexpr>)
        1    0.036    0.036   16.077   16.077 someprogram.py:4(<module>)
   262144   15.021    0.000   15.021    0.000 someprogram.py:4(in_mandelbrot)
        1    0.000    0.000    0.000    0.000 os.py:746(urandom)
        1    0.000    0.000    0.000    0.000 png.py:1056(_readable)
        1    0.000    0.000    0.000    0.000 png.py:1073(Reader)
        1    0.227    0.227    0.438    0.438 png.py:163(<module>)
      512    0.010    0.000    0.010    0.000 png.py:200(group)
    ...
bash %
```

```
# timethis.py

import time
from functools import wraps

def timethis(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
```

```
        start = time.perf_counter()
        r = func(*args, **kwargs)
        end = time.perf_counter()
        print('{}.{} : {}'.format(func.__module__, func.__name__, end - start))
        return r
    return wrapper
```

```
>>> @timethis
... def countdown(n):
...     while n > 0:
...             n -= 1
...
>>> countdown(10000000)
__main__.countdown : 0.803001880645752
>>>
```

```
from contextlib import contextmanager

@contextmanager
def timeblock(label):
    start = time.perf_counter()
    try:
        yield
    finally:
        end = time.perf_counter()
        print('{} : {}'.format(label, end - start))
```

```
>>> with timeblock('counting'):
...     n = 10000000
...     while n > 0:
...             n -= 1
...
counting : 1.5551159381866455
>>>
```

timeit

```
>>> from timeit import timeit
>>> timeit('math.sqrt(2)', 'import math')
0.1432319980012835
>>> timeit('sqrt(2)', 'from math import sqrt')
0.10836604500218527
>>>
```

timeit 100

number

```
>>> timeit('math.sqrt(2)', 'import math', number=10000000)
1.434852126003534
>>> timeit('sqrt(2)', 'from math import sqrt', number=10000000)
1.0270336690009572
>>>
```

16.13.3

time.perf_counter()

time.process_time()

```
from functools import wraps
def timethis(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.process_time()
        r = func(*args, **kwargs)
        end = time.process_time()
        print('{}.{} : {}'.format(func.__module__, func.__name__, end - start))
        return r
    return wrapper
```

time timeit

13.13

16.14 14.14

16.14.1

C JIT

16.14.2

“ ” “

” 14.13

Python

```
# somescript.py

import sys
import csv

with open(sys.argv[1]) as f:
     for row in csv.reader(f):

         # Some kind of processing
         pass
```

```
# somescript.py
import sys
import csv

def main(filename):
    with open(filename) as f:
         for row in csv.reader(f):
             # Some kind of processing
             pass

main(sys.argv[1])
```

15-30%

(.)
__getattribute__() __getattr__()

from module import name

```
import math

def compute_roots(nums):
    result = []
    for n in nums:
        result.append(math.sqrt(n))
    return result

# Test
nums = range(1000000)
for n in range(100):
    r = compute_roots(nums)
```

40

compute_roots()

```
from math import sqrt

def compute_roots(nums):

    result = []
    result_append = result.append
    for n in nums:
        result_append(sqrt(n))
    return result
```

29

sqrt() math.sqrt() The result.append() result_append

compute_roots()

```
import math

def compute_roots(nums):
    sqrt = math.sqrt
    result = []
    result_append = result.append
    for n in nums:
        result_append(sqrt(n))
    return result
```

sqrt match
25 29
sqrt sqrt

self.name

```
# Slower
class SomeClass:
    ...
    def method(self):
         for x in s:
             op(self.value)

# Faster
class SomeClass:
```

```
    ...
    def method(self):
         value = self.value
         for x in s:
             op(value)
```

```
class A:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    @property
    def y(self):
        return self._y
    @y.setter
    def y(self, value):
        self._y = value
```

```
>>> from timeit import timeit
>>> a = A(1,2)
>>> timeit('a.x', 'from __main__ import a')
0.07817923510447145
>>> timeit('a.y', 'from __main__ import a')
0.35766440676525235
>>>
```

y x 4.5

y

getter/ setter

C

```
values = [x for x in sequence]
squares = [x*x for x in values]
```

```
squares = [x*x for x in sequence]
```

Python
Python copy.deepcopy()

16.14.3

O(n log n)
O(n**2)

,

```
a = {
    'name' : 'AAPL',
    'shares' : 100,
    'price' : 534.22
}

b = dict(name='AAPL', shares=100, price=534.22)
```

dict() 3
dict()

JIT PyPy Python

C
PyPy Python3.
Numba Numba Python
LLVM
PyPy Python 3

John Ousterhout “
”

.

CHAPTER

SEVENTEEN

C

Python C Python C
C Python
Python 2 Python 3 Python API
C
C++

```
/* sample.c */_method
#include <math.h>

/* Compute the greatest common divisor */
int gcd(int x, int y) {
    int g = y;
    while (x > 0) {
        g = x;
        x = y % x;
        y = g;
    }
    return g;
}

/* Test if (x0,y0) is in the Mandelbrot set or not */
int in_mandel(double x0, double y0, int n) {
    double x=0,y=0,xtemp;
    while (n > 0) {
        xtemp = x*x - y*y + x0;
        y = 2*x*y + y0;
        x = xtemp;
        n -= 1;
        if (x*x + y*y > 4) return 0;
    }
    return 1;
}

/* Divide two numbers */
```

```
int divide(int a, int b, int *remainder) {
    int quot = a / b;
    *remainder = a % b;
    return quot;
}

/* Average values in an array */
double avg(double *a, int n) {
    int i;
    double total = 0.0;
    for (i = 0; i < n; i++) {
        total += a[i];
    }
    return total / n;
}

/* A C data structure */
typedef struct Point {
    double x,y;
} Point;

/* Function involving a C data structure */
double distance(Point *p1, Point *p2) {
    return hypot(p1->x - p2->x, p1->y - p2->y);
}
```

C gcd() is_mandel() divide() C avg() C Point distance() C

“ sample.c ” “ sample.h ” “ libsample ” C C

Contents:

17.1 15.1 ctypes C

17.1.1

C DLL Python C

17.1.2

C Python ctypes
ctypes C Python

libsample.so 15
libsample.so sample.py

Python

```
# sample.py
import ctypes
import os

# Try to locate the .so file in the same directory as this file
_file = 'libsample.so'
_path = os.path.join(*(os.path.split(__file__)[:-1] + (_file,)))
_mod = ctypes.cdll.LoadLibrary(_path)

# int gcd(int, int)
gcd = _mod.gcd
gcd.argtypes = (ctypes.c_int, ctypes.c_int)
gcd.restype = ctypes.c_int

# int in_mandel(double, double, int)
in_mandel = _mod.in_mandel
in_mandel.argtypes = (ctypes.c_double, ctypes.c_double, ctypes.c_int)
in_mandel.restype = ctypes.c_int

# int divide(int, int, int *)
_divide = _mod.divide
_divide.argtypes = (ctypes.c_int, ctypes.c_int, ctypes.POINTER(ctypes.c_int))
_divide.restype = ctypes.c_int

def divide(x, y):
    rem = ctypes.c_int()
    quot = _divide(x, y, rem)

    return quot,rem.value

# void avg(double *, int n)
# Define a special type for the 'double *' argument
class DoubleArrayType:
    def from_param(self, param):
        typename = type(param).__name__
        if hasattr(self, 'from_' + typename):
            return getattr(self, 'from_' + typename)(param)
        elif isinstance(param, ctypes.Array):
            return param
        else:
```

```
            raise TypeError("Can't convert %s" % typename)

    # Cast from array.array objects
    def from_array(self, param):
        if param.typecode != 'd':
            raise TypeError('must be an array of doubles')
        ptr, _ = param.buffer_info()
        return ctypes.cast(ptr, ctypes.POINTER(ctypes.c_double))

    # Cast from lists/tuples
    def from_list(self, param):
        val = ((ctypes.c_double)*len(param))(*param)
        return val

    from_tuple = from_list

    # Cast from a numpy array
    def from_ndarray(self, param):
        return param.ctypes.data_as(ctypes.POINTER(ctypes.c_double))

DoubleArray = DoubleArrayType()
_avg = _mod.avg
_avg.argtypes = (DoubleArray, ctypes.c_int)
_avg.restype = ctypes.c_double

def avg(values):
    return _avg(values, len(values))

# struct Point { }
class Point(ctypes.Structure):
    _fields_ = [('x', ctypes.c_double),
                ('y', ctypes.c_double)]

# double distance(Point *, Point *)
distance = _mod.distance
distance.argtypes = (ctypes.POINTER(Point), ctypes.POINTER(Point))
distance.restype = ctypes.c_double
```

C

```
>>> import sample
>>> sample.gcd(35,42)
7
>>> sample.in_mandel(0,0,500)
1
>>> sample.in_mandel(2.0,1.0,500)
0
>>> sample.divide(42,8)
(5, 2)
>>> sample.avg([1,2,3])
2.0
```

```
>>> p1 = sample.Point(1,2)
>>> p2 = sample.Point(4,5)
>>> sample.distance(p1,p2)
4.242640687119285
>>>
```

17.1.3

C Python

ctypes C

sample.py .so Python

recipe sample.py __file__

libsample.so

C C

ctypes.util.find_library()

```
>>> from ctypes.util import find_library
>>> find_library('m')
'/usr/lib/libm.dylib'
>>> find_library('pthread')
'/usr/lib/libpthread.dylib'
>>> find_library('sample')
'/usr/local/lib/libsample.so'
>>>
```

C

ctypes.cdll.LoadLibrary() _path

```
_mod = ctypes.cdll.LoadLibrary(_path)
```

```
# int in_mandel(double, double, int)
in_mandel = _mod.in_mandel
in_mandel.argtypes = (ctypes.c_double, ctypes.c_double, ctypes.c_int)
in_mandel.restype = ctypes.c_int
```

.argtypes

.restype ctypes c_double, c_int, c_short, c_float C Python

ctypes C Python

divide()

C Python

```
>>> divide = _mod.divide
>>> divide.argtypes = (ctypes.c_int, ctypes.c_int, ctypes.POINTER(ctypes.c_int))
>>> x = 0
>>> divide(10, 3, x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ctypes.ArgumentError: argument 3: <class 'TypeError'>: expected LP_c_int
instance instead of int
>>>
```

Python

ctypes

```
>>> x = ctypes.c_int()
>>> divide(10, 3, x)
3
>>> x.value
1
>>>
```

ctypes.c_int Python
c_int .value

Python C
divide()

```
# int divide(int, int, int *)
_divide = _mod.divide
_divide.argtypes = (ctypes.c_int, ctypes.c_int, ctypes.POINTER(ctypes.c_int))
_divide.restype = ctypes.c_int

def divide(x, y):
    rem = ctypes.c_int()
    quot = _divide(x,y,rem)
    return quot, rem.value
```

avg() C
Python
array numpy
Python “ ”

DoubleArrayType
from_param()
ctypes ctypes.c_double from_param()

typename list from_list

from_list ctypes
ctypes

```
>>> nums = [1, 2, 3]
>>> a = (ctypes.c_double * len(nums))(*nums)
>>> a
<__main__.c_double_Array_3 object at 0x10069cd40>
>>> a[0]
1.0
>>> a[1]
2.0
>>> a[2]
3.0
>>>
```

from_array() ctypes

```
>>> import array
>>> a = array.array('d',[1,2,3])
>>> a
array('d', [1.0, 2.0, 3.0])
>>> ptr_ = a.buffer_info()
>>> ptr
4298687200
>>> ctypes.cast(ptr, ctypes.POINTER(ctypes.c_double))
<__main__.LP_c_double object at 0x10069cd40>
>>>
```

from_ndarray() numpy DoubleArrayType avg()

```
>>> import sample
>>> sample.avg([1,2,3])
2.0
>>> sample.avg((1,2,3))
2.0
>>> import array
>>> sample.avg(array.array('d',[1,2,3]))
2.0
>>> import numpy
>>> sample.avg(numpy.array([1.0,2.0,3.0]))
2.0
>>>
```

C

```
class Point(ctypes.Structure):
    _fields_ = [('x', ctypes.c_double),
                ('y', ctypes.c_double)]
```

```
>>> p1 = sample.Point(1,2)
>>> p2 = sample.Point(4,5)
>>> p1.x
1.0
>>> p1.y
2.0
>>> sample.distance(p1,p2)
4.242640687119285
>>>
```

Python C ctypes

Swig (15.9) Cython 15.10

ctypes

C

Python

ctypes CFFI CFFI
C C CFFI
Python

17.2 15.2 C

17.2.1

Python API C

17.2.2

C
C

```
/* sample.h */

#include <math.h>

extern int gcd(int, int);
extern int in_mandel(double x0, double y0, int n);
extern int divide(int a, int b, int *remainder);
extern double avg(double *a, int n);

typedef struct Point {
```

```
    double x,y;
} Point;

extern double distance(Point *p1, Point *p2);
```

```
#include "Python.h"
#include "sample.h"

/* int gcd(int, int) */
static PyObject *py_gcd(PyObject *self, PyObject *args) {
  int x, y, result;

  if (!PyArg_ParseTuple(args,"ii", &x, &y)) {
    return NULL;
  }
  result = gcd(x,y);
  return Py_BuildValue("i", result);
}

/* int in_mandel(double, double, int) */
static PyObject *py_in_mandel(PyObject *self, PyObject *args) {
  double x0, y0;
  int n;
  int result;

  if (!PyArg_ParseTuple(args, "ddi", &x0, &y0, &n)) {
    return NULL;
  }
  result = in_mandel(x0,y0,n);
  return Py_BuildValue("i", result);
}

/* int divide(int, int, int *) */
static PyObject *py_divide(PyObject *self, PyObject *args) {
  int a, b, quotient, remainder;
  if (!PyArg_ParseTuple(args, "ii", &a, &b)) {
    return NULL;
  }
  quotient = divide(a,b, &remainder);
  return Py_BuildValue("(ii)", quotient, remainder);
}

/* Module method table */
static PyMethodDef SampleMethods[] = {
  {"gcd",  py_gcd, METH_VARARGS, "Greatest common divisor"},
  {"in_mandel", py_in_mandel, METH_VARARGS, "Mandelbrot test"},
  {"divide", py_divide, METH_VARARGS, "Integer division"},
  { NULL, NULL, 0, NULL}
```

```
};

/* Module structure */
static struct PyModuleDef samplemodule = {
  PyModuleDef_HEAD_INIT,

  "sample",           /* name of module */
  "A sample module",  /* Doc string (may be NULL) */
  -1,                 /* Size of per-interpreter state or -1 */
  SampleMethods       /* Method table */
};

/* Module initialization function */
PyMODINIT_FUNC
PyInit_sample(void) {
  return PyModule_Create(&samplemodule);
}
```

setup.py

```
# setup.py
from distutils.core import setup, Extension

setup(name='sample',
      ext_modules=[
        Extension('sample',
                  ['pysample.c'],
                  include_dirs = ['/some/dir'],
                  define_macros = [('FOO','1')],
                  undef_macros = ['BAR'],
                  library_dirs = ['/usr/local/lib'],
                  libraries = ['sample']
                  )
        ]
)
```

python3 buildlib.py build_ext --inplace

```
bash % python3 setup.py build_ext --inplace
running build_ext
building 'sample' extension
gcc -fno-strict-aliasing -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
 -I/usr/local/include/python3.3m -c pysample.c
 -o build/temp.macosx-10.6-x86_64-3.3/pysample.o
gcc -bundle -undefined dynamic_lookup
build/temp.macosx-10.6-x86_64-3.3/pysample.o \
 -L/usr/local/lib -lsample -o sample.so
bash %
```

sample.so

```
>>> import sample
>>> sample.gcd(35, 42)
7
>>> sample.in_mandel(0, 0, 500)
1
>>> sample.in_mandel(2.0, 1.0, 500)

0
>>> sample.divide(42, 8)
(5, 2)
>>>
```

Windows

Python Microsoft Visual Studio

Python

17.2.3

Python Python

. Python C API

```
static PyObject *py_func(PyObject *self, PyObject *args) {
  ...
}
```

`PyObject` Python C

Python PyObject *args

Python C `self`

C

`self`

`PyArg_ParseTuple()` Python C

“ i ” “ d ”

C

NULL NULL

`Py_BuildValue()` C Python

Python

`Py_BuildValue()`

`py_divide()`

```
return Py_BuildValue("i", 34);      // Return an integer
return Py_BuildValue("d", 3.4);     // Return a double
return Py_BuildValue("s", "Hello"); // Null-terminated UTF-8 string
return Py_BuildValue("(ii)", 3, 4); // Tuple (3, 4)
```

SampleMethods

C Python

PyInit_sample()

C Python
C API 500
PyArg_ParseTuple() Py_BuildValue()

17.3 15.3

17.3.1

C array Numpy

17.3.2

Buffer Protocol .
C avg(double *buf, int len)

```
/* Call double avg(double *, int) */
static PyObject *py_avg(PyObject *self, PyObject *args) {
  PyObject *bufobj;
  Py_buffer view;
  double result;
  /* Get the passed Python object */
  if (!PyArg_ParseTuple(args, "O", &bufobj)) {
    return NULL;
  }

  /* Attempt to extract buffer information from it */

  if (PyObject_GetBuffer(bufobj, &view,
      PyBUF_ANY_CONTIGUOUS | PyBUF_FORMAT) == -1) {
    return NULL;
  }

  if (view.ndim != 1) {
    PyErr_SetString(PyExc_TypeError, "Expected a 1-dimensional array");
    PyBuffer_Release(&view);
    return NULL;
  }
```

```
  /* Check the type of items in the array */
  if (strcmp(view.format,"d") != 0) {
    PyErr_SetString(PyExc_TypeError, "Expected an array of doubles");
    PyBuffer_Release(&view);
    return NULL;
  }

  /* Pass the raw buffer and size to the C function */
  result = avg(view.buf, view.shape[0]);

  /* Indicate we're done working with the buffer */
  PyBuffer_Release(&view);
  return Py_BuildValue("d", result);
}
```

```
>>> import array
>>> avg(array.array('d',[1,2,3]))
2.0
>>> import numpy
>>> avg(numpy.array([1.0,2.0,3.0]))
2.0
>>> avg([1,2,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'list' does not support the buffer interface
>>> avg(b'Hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Expected an array of doubles
>>> a = numpy.array([[1.,2.,3.],[4.,5.,6.]])
>>> avg(a[:,2])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: ndarray is not contiguous
>>> sample.avg(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Expected a 1-dimensional array
>>> sample.avg(a[0])

2.0
>>>
```

17.3.3

C Python

PyBuffer_GetBuffer() Python
-1.

PyBuffer_GetBuffer()
PyBUF_ANY_CONTIGUOUS

Py_buffer

```
typedef struct bufferinfo {
    void *buf;              /* Pointer to buffer memory */
    PyObject *obj;          /* Python object that is the owner */
    Py_ssize_t len;         /* Total size in bytes */
    Py_ssize_t itemsize;    /* Size in bytes of a single item */
    int readonly;           /* Read-only access flag */
    int ndim;               /* Number of dimensions */
    char *format;           /* struct code of a single item */
    Py_ssize_t *shape;      /* Array containing dimensions */
    Py_ssize_t *strides;    /* Array containing strides */
    Py_ssize_t *suboffsets; /* Array containing suboffsets */
} Py_buffer;
```

format ”d”. struct
format struct
C
C C

array numpy

PyBuffer_Release()

Cython
15.11

17.4 15.4 C

17.4.1

C
Python

17.4.2

C

```
typedef struct Point {
    double x,y;
} Point;

extern double distance(Point *p1, Point *p2);
```

Point distance()

```
/* Destructor function for points */
static void del_Point(PyObject *obj) {
  free(PyCapsule_GetPointer(obj,"Point"));
}

/* Utility functions */
static Point *PyPoint_AsPoint(PyObject *obj) {
  return (Point *) PyCapsule_GetPointer(obj, "Point");
}

static PyObject *PyPoint_FromPoint(Point *p, int must_free) {
  return PyCapsule_New(p, "Point", must_free ? del_Point : NULL);
}

/* Create a new Point object */
static PyObject *py_Point(PyObject *self, PyObject *args) {

  Point *p;
  double x,y;
  if (!PyArg_ParseTuple(args,"dd",&x,&y)) {
    return NULL;
  }
  p = (Point *) malloc(sizeof(Point));
  p->x = x;
  p->y = y;
  return PyPoint_FromPoint(p, 1);
}

static PyObject *py_distance(PyObject *self, PyObject *args) {
  Point *p1, *p2;
  PyObject *py_p1, *py_p2;
  double result;

  if (!PyArg_ParseTuple(args,"OO",&py_p1, &py_p2)) {
    return NULL;
  }
  if (!(p1 = PyPoint_AsPoint(py_p1))) {
    return NULL;
```

```
  }
  if (!(p2 = PyPoint_AsPoint(py_p2))) {
    return NULL;
  }
  result = distance(p1,p2);
  return Py_BuildValue("d", result);
}
```

Python

```
>>> import sample
>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> p1
<capsule object "Point" at 0x1004ea330>
>>> p2
<capsule object "Point" at 0x1005d1db0>
>>> sample.distance(p1,p2)
2.8284271247461903
>>>
```

17.4.3

C
`PyCapsule_New()`

`PyCapsule_GetPointer()`
NULL

—— `PyPoint_FromPoint()` `PyPoint_AsPoint()`
Point

Point

`PyPoint_FromPoint()`
`must_free` Point *
C Point
`extra`
`PyCapsule_SetDestructor()`

C

17.5 15.5 C API

17.5.1

C
C API
C /

17.5.2

15.4 Point C

```
/* Destructor function for points */
static void del_Point(PyObject *obj) {

  free(PyCapsule_GetPointer(obj,"Point"));
}

/* Utility functions */
static Point *PyPoint_AsPoint(PyObject *obj) {
  return (Point *) PyCapsule_GetPointer(obj, "Point");
}

static PyObject *PyPoint_FromPoint(Point *p, int must_free) {
  return PyCapsule_New(p, "Point", must_free ? del_Point : NULL);
}
```

PyPoint_AsPoint() Point_FromPoint() API

Point

sample pysample.h

```
/* pysample.h */
#include "Python.h"
#include "sample.h"
#ifdef __cplusplus
extern "C" {
#endif

/* Public API Table */
typedef struct {
  Point *(*aspoint)(PyObject *);
  PyObject *(*frompoint)(Point *, int);
} _PointAPIMethods;

#ifndef PYSAMPLE_MODULE
```

```
/* Method table in external module */
static _PointAPIMethods *_point_api = 0;

/* Import the API table from sample */
static int import_sample(void) {
  _point_api = (_PointAPIMethods *) PyCapsule_Import("sample._point_api",0);
  return (_point_api != NULL) ? 1 : 0;
}

/* Macros to implement the programming interface */
#define PyPoint_AsPoint(obj) (_point_api->aspoint)(obj)
#define PyPoint_FromPoint(obj) (_point_api->frompoint)(obj)
#endif

#ifdef __cplusplus
}
#endif
```

_PointAPIMethods .

```
/* pysample.c */

#include "Python.h"
#define PYSAMPLE_MODULE
#include "pysample.h"

...
/* Destructor function for points */
static void del_Point(PyObject *obj) {
  printf("Deleting point\n");
  free(PyCapsule_GetPointer(obj,"Point"));
}

/* Utility functions */
static Point *PyPoint_AsPoint(PyObject *obj) {
  return (Point *) PyCapsule_GetPointer(obj, "Point");
}

static PyObject *PyPoint_FromPoint(Point *p, int free) {
  return PyCapsule_New(p, "Point", free ? del_Point : NULL);
}

static _PointAPIMethods _point_api = {
  PyPoint_AsPoint,
  PyPoint_FromPoint
};
...

/* Module initialization function */
PyMODINIT_FUNC
```

```
PyInit_sample(void) {
  PyObject *m;
  PyObject *py_point_api;

  m = PyModule_Create(&samplemodule);
  if (m == NULL)
    return NULL;

  /* Add the Point C API functions */
  py_point_api = PyCapsule_New((void *) &_point_api, "sample._point_api", NULL);
  if (py_point_api) {
    PyModule_AddObject(m, "_point_api", py_point_api);
  }
  return m;
}
```

API

```
/* ptexample.c */

/* Include the header associated with the other module */
#include "pysample.h"

/* An extension function that uses the exported API */
static PyObject *print_point(PyObject *self, PyObject *args) {
  PyObject *obj;
  Point *p;
  if (!PyArg_ParseTuple(args,"O", &obj)) {
    return NULL;
  }

  /* Note: This is defined in a different module */
  p = PyPoint_AsPoint(obj);
  if (!p) {
    return NULL;
  }
  printf("%f %f\n", p->x, p->y);
  return Py_BuildValue("");
}

static PyMethodDef PtExampleMethods[] = {
  {"print_point", print_point, METH_VARARGS, "output a point"},
  { NULL, NULL, 0, NULL}
};

static struct PyModuleDef ptexamplemodule = {
  PyModuleDef_HEAD_INIT,
  "ptexample",           /* name of module */
  "A module that imports an API",  /* Doc string (may be NULL) */
  -1,                 /* Size of per-interpreter state or -1 */
  PtExampleMethods       /* Method table */
```

```
};

/* Module initialization function */
PyMODINIT_FUNC
PyInit_ptexample(void) {
  PyObject *m;

  m = PyModule_Create(&ptexamplemodule);
  if (m == NULL)
    return NULL;

  /* Import sample, loading its API functions */
  if (!import_sample()) {
    return NULL;
  }

  return m;
}
```

setup.py

```
# setup.py
from distutils.core import setup, Extension

setup(name='ptexample',
      ext_modules=[
        Extension('ptexample',
                  ['ptexample.c'],
                  include_dirs = [],  # May need pysample.h directory
                  )
        ]
)
```

C API

```
>>> import sample
>>> p1 = sample.Point(2,3)
>>> p1
<capsule object "Point *" at 0x1004ea330>
>>> import ptexample
>>> ptexample.print_point(p1)
2.000000 3.000000
>>>
```

17.5.3

sample._point_api .

Python

PyCapsule_Import()
sample._point_api

C

pysample.h _point_api
import_sample()

C API

——

API

Python

C API Python

17.6 15.6 C Python

17.6.1

C Python C C
Python

17.6.2

C Python C

```
#include <Python.h>

/* Execute func(x,y) in the Python interpreter.  The
   arguments and return result of the function must
   be Python floats */

double call_func(PyObject *func, double x, double y) {
  PyObject *args;
  PyObject *kwargs;
  PyObject *result = 0;
  double retval;

  /* Make sure we own the GIL */
```

```
  PyGILState_STATE state = PyGILState_Ensure();

  /* Verify that func is a proper callable */
  if (!PyCallable_Check(func)) {
    fprintf(stderr,"call_func: expected a callable\n");
    goto fail;
  }
  /* Build arguments */
  args = Py_BuildValue("(dd)", x, y);
  kwargs = NULL;

  /* Call the function */
  result = PyObject_Call(func, args, kwargs);
  Py_DECREF(args);
  Py_XDECREF(kwargs);

  /* Check for Python exceptions (if any) */
  if (PyErr_Occurred()) {
    PyErr_Print();
    goto fail;
  }

  /* Verify the result is a float object */
  if (!PyFloat_Check(result)) {
    fprintf(stderr,"call_func: callable didn't return a float\n");
    goto fail;
  }

  /* Create the return value */
  retval = PyFloat_AsDouble(result);
  Py_DECREF(result);

  /* Restore previous GIL state and return */
  PyGILState_Release(state);
  return retval;

fail:
  Py_XDECREF(result);
  PyGILState_Release(state);
  abort();   // Change to something more appropriate
}
```

Python

C

Python

```
#include <Python.h>

/* Definition of call_func() same as above */
```

```
...

/* Load a symbol from a module */
PyObject *import_name(const char *modname, const char *symbol) {
  PyObject *u_name, *module;
  u_name = PyUnicode_FromString(modname);
  module = PyImport_Import(u_name);
  Py_DECREF(u_name);
  return PyObject_GetAttrString(module, symbol);
}

/* Simple embedding example */
int main() {
  PyObject *pow_func;
  double x;

  Py_Initialize();
  /* Get a reference to the math.pow function */
  pow_func = import_name("math","pow");

  /* Call it using our call_func() code */
  for (x = 0.0; x < 10.0; x += 0.1) {
    printf("%0.2f %0.2f\n", x, call_func(pow_func,x,2.0));
  }
  /* Done */
  Py_DECREF(pow_func);
  Py_Finalize();
  return 0;
}
```

C Python Makefile

```
all::
        cc -g embed.c -I/usr/local/include/python3.3m \
          -L/usr/local/lib/python3.3/config-3.3m -lpython3.3m
```

```
0.00 0.00
0.10 0.01
0.20 0.04
0.30 0.09
0.40 0.16
...
```

call_func()

```
/* Extension function for testing the C-Python callback */
PyObject *py_call_func(PyObject *self, PyObject *args) {
  PyObject *func;
```

```
  double x, y, result;
  if (!PyArg_ParseTuple(args,"Odd", &func,&x,&y)) {
    return NULL;
  }
  result = call_func(func, x, y);
  return Py_BuildValue("d", result);
}
```

```
>>> import sample
>>> def add(x,y):
...     return x+y
...
>>> sample.call_func(add,3,4)
7.0
>>>
```

17.6.3

C Python C
C Python

Python
__call__()
PyCallable_Check()

```
double call_func(PyObject *func, double x, double y) {
  ...
  /* Verify that func is a proper callable */
  if (!PyCallable_Check(func)) {
    fprintf(stderr,"call_func: expected a callable\n");
    goto fail;
  }
  ...
```

C Python
C
abort()
C

——— PyObject_Call()

Py_BuildValue() ,

```
double call_func(PyObject *func, double x, double y) {
  PyObject *args;
  PyObject *kwargs;
```

```
  ...
  /* Build arguments */
  args = Py_BuildValue("(dd)", x, y);
  kwargs = NULL;

  /* Call the function */
  result = PyObject_Call(func, args, kwargs);
  Py_DECREF(args);
  Py_XDECREF(kwargs);
  ...
```

NULL

Py_DECREF() Py_XDECREF()
NULL

Python PyErr_Occurred()
C

Python

abort() C

```
...
/* Check for Python exceptions (if any) */
if (PyErr_Occurred()) {
  PyErr_Print();
  goto fail;
}
...
fail:
  PyGILState_Release(state);
  abort();
```

Python
Python PyFloat_Check()
PyFloat_AsDouble() Python

Python C Python
GIL
PyGILState_Ensure() PyGILState_Release()

```
double call_func(PyObject *func, double x, double y) {
  ...
  double retval;

  /* Make sure we own the GIL */
  PyGILState_STATE state = PyGILState_Ensure();
  ...
  /* Code that uses Python C API functions */
  ...
  /* Restore previous GIL state and return */
  PyGILState_Release(state);
```

```
  return retval;

fail:
  PyGILState_Release(state);
  abort();
}
```

PyGILState_Ensure() Python C
C
Python C-API PyGILState_Release()

PyGILState_Ensure()
PyGILState_Release() —— goto
exit
fail: Python fianl:

C GIL
C Python ——

17.7 15.7 C

17.7.1

C Python
GIL

17.7.2

C GIL

```
#include "Python.h"
...

PyObject *pyfunc(PyObject *self, PyObject *args) {
   ...
   Py_BEGIN_ALLOW_THREADS
   // Threaded C code.  Must not use Python API functions
   ...
   Py_END_ALLOW_THREADS
   ...
   return result;
}
```

17.7.3

Python C API C GIL
GIL C
numpy I/O

GIL Python
Py_END_ALLOW_THREADS GIL

17.8 15.8 C Python

17.8.1

C Python C
Python Python C API

17.8.2

C Python
Python GIL C

```
#include <Python.h>
  ...
  if (!PyEval_ThreadsInitialized()) {
    PyEval_InitThreads();
  }
  ...
```

Python Python C API C
GIL PyGILState_Ensure() PyGILState_Release()

```
...
/* Make sure we own the GIL */
PyGILState_STATE state = PyGILState_Ensure();

/* Use functions in the interpreter */
...
/* Restore previous GIL state and return */
PyGILState_Release(state);
...
```

PyGILState_Ensure() PyGILState_Release() .

17.8.3

C Python ——
C Python C Python
C GIL

PyGILState_Ensure()

GIL

17.9 15.9 WSIG C

17.9.1

C C Swig

17.9.2

Swig C C

```
/* sample.h */

#include <math.h>
extern int gcd(int, int);
extern int in_mandel(double x0, double y0, int n);
extern int divide(int a, int b, int *remainder);
extern double avg(double *a, int n);

typedef struct Point {
    double x,y;
} Point;

extern double distance(Point *p1, Point *p2);
```

Swig” ”
”.i”

```
// sample.i - Swig interface
%module sample
%{
#include "sample.h"
%}

/* Customizations */
```

```
%extend Point {
    /* Constructor for Point objects */
    Point(double x, double y) {
        Point *p = (Point *) malloc(sizeof(Point));
        p->x = x;
        p->y = y;
        return p;
   };
};

/* Map int *remainder as an output argument */
%include typemaps.i
%apply int *OUTPUT { int * remainder };

/* Map the argument pattern (double *a, int n) to arrays */
%typemap(in) (double *a, int n)(Py_buffer view) {
  view.obj = NULL;
  if (PyObject_GetBuffer($input, &view, PyBUF_ANY_CONTIGUOUS | PyBUF_FORMAT) == -1) {
    SWIG_fail;
  }
  if (strcmp(view.format,"d") != 0) {
    PyErr_SetString(PyExc_TypeError, "Expected an array of doubles");
    SWIG_fail;
  }
  $1 = (double *) view.buf;
  $2 = view.len / sizeof(double);
}

%typemap(freearg) (double *a, int n) {
  if (view$argnum.obj) {
    PyBuffer_Release(&view$argnum);
  }
}

/* C declarations to be included in the extension module */

extern int gcd(int, int);
extern int in_mandel(double x0, double y0, int n);
extern int divide(int a, int b, int *remainder);
extern double avg(double *a, int n);

typedef struct Point {
    double x,y;
} Point;

extern double distance(Point *p1, Point *p2);
```

Swig

```
bash % swig -python -py3 sample.i
bash %
```

swig sample_wrap.c sample.py
sample_wrap.c _sample C

setup.py

```
# setup.py
from distutils.core import setup, Extension

setup(name='sample',
      py_modules=['sample.py'],
      ext_modules=[
        Extension('_sample',
                  ['sample_wrap.c'],
                  include_dirs = [],
                  define_macros = [],

                  undef_macros = [],
                  library_dirs = [],
                  libraries = ['sample']
                  )
        ]
)
```

setup.py python3

```
bash % python3 setup.py build_ext --inplace
running build_ext
building '_sample' extension
gcc -fno-strict-aliasing -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
-I/usr/local/include/python3.3m -c sample_wrap.c
 -o build/temp.macosx-10.6-x86_64-3.3/sample_wrap.o
sample_wrap.c: In function  SWIG_InitializeModule :
sample_wrap.c:3589: warning: statement with no effect
gcc -bundle -undefined dynamic_lookup build/temp.macosx-10.6-x86_64-3.3/sample.o
 build/temp.macosx-10.6-x86_64-3.3/sample_wrap.o -o _sample.so -lsample
bash %
```

C

```
>>> import sample
>>> sample.gcd(42,8)
2
>>> sample.divide(42,8)
[5, 2]
>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> sample.distance(p1,p2)
2.8284271247461903
>>> p1.x
2.0
>>> p1.y
3.0
```

```
>>> import array
>>> a = array.array('d',[1,2,3])
>>> sample.avg(a)
2.0
>>>
```

17.9.3

Swig Python Swig

Swig

```
%module sample
%{
#include "sample.h"
%}
```

C

%{ %}

Swig C

```
%module sample
%{
#include "sample.h"
%}
...
extern int gcd(int, int);
extern int in_mandel(double x0, double y0, int n);
extern int divide(int a, int b, int *remainder);
extern double avg(double *a, int n);

typedef struct Point {
    double x,y;
} Point;

extern double distance(Point *p1, Point *p2);
```

Swig Python

Swig C

%extend
Point

```
>>> p1 = sample.Point(2,3)
>>>
```

Point

```
>>> # Usage if %extend Point is omitted
>>> p1 = sample.Point()
>>> p1.x = 2.0
>>> p1.y = 3
```

typemaps.i %apply Swig
int *remainder
int *remainder
divide()

```
>>> sample.divide(42,8)
[5, 2]
>>>
```

%typemap
typemap typemap
(double *a, int n) . typemap C
Swig Python C Python
NumPy array
15.3

typemap $1 $2 typemap C
$1 double *a $input PyObject *
$argnum

typemaps Swig
Python C API Swig Swig

C Swig
Swig C
Swig
Python

17.10 15.10 Cython C

17.10.1

Cython Python C

17.10.2

Cython

C ——

Python

libsample C

csample.pxd

```
# csample.pxd
#
# Declarations of "external" C functions and structures

cdef extern from "sample.h":
    int gcd(int, int)
    bint in_mandel(double, double, int)
    int divide(int, int, int *)
    double avg(double *, int) nogil

    ctypedef struct Point:
         double x
         double y

    double distance(Point *, Point *)
```

Cython C cdef extern from "sample.h" C csample.pxd sample.pxd ——

sample.pyx Python csample.pxd C

```
# sample.pyx

# Import the low-level C declarations
cimport csample

# Import some functionality from Python and the C stdlib
from cpython.pycapsule cimport *

from libc.stdlib cimport malloc, free

# Wrappers
def gcd(unsigned int x, unsigned int y):
    return csample.gcd(x, y)

def in_mandel(x, y, unsigned int n):
    return csample.in_mandel(x, y, n)

def divide(x, y):
    cdef int rem
    quot = csample.divide(x, y, &rem)
```

```
    return quot, rem

def avg(double[:] a):
    cdef:
        int sz
        double result

    sz = a.size
    with nogil:
        result = csample.avg(<double *> &a[0], sz)
    return result

# Destructor for cleaning up Point objects
cdef del_Point(object obj):
    pt = <csample.Point *> PyCapsule_GetPointer(obj,"Point")
    free(<void *> pt)

# Create a Point object and return as a capsule
def Point(double x,double y):
    cdef csample.Point *p
    p = <csample.Point *> malloc(sizeof(csample.Point))
    if p == NULL:
        raise MemoryError("No memory to make a Point")
    p.x = x
    p.y = y
    return PyCapsule_New(<void *>p,"Point",<PyCapsule_Destructor>del_Point)

def distance(p1, p2):
    pt1 = <csample.Point *> PyCapsule_GetPointer(p1,"Point")
    pt2 = <csample.Point *> PyCapsule_GetPointer(p2,"Point")
    return csample.distance(pt1,pt2)
```

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [
    Extension('sample',

              ['sample.pyx'],
              libraries=['sample'],
              library_dirs=['.'])]
setup(
  name = 'Sample extension module',
  cmdclass = {'build_ext': build_ext},
  ext_modules = ext_modules
)
```

```
bash % python3 setup.py build_ext --inplace
running build_ext
cythoning sample.pyx to sample.c
building 'sample' extension
gcc -fno-strict-aliasing -DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes
 -I/usr/local/include/python3.3m -c sample.c
 -o build/temp.macosx-10.6-x86_64-3.3/sample.o
gcc -bundle -undefined dynamic_lookup build/temp.macosx-10.6-x86_64-3.3/sample.o
  -L. -lsample -o sample.so
bash %
```

sample.so

```
>>> import sample
>>> sample.gcd(42,10)
2
>>> sample.in_mandel(1,1,400)
False
>>> sample.in_mandel(0,0,400)
True
>>> sample.divide(42,10)
(4, 2)
>>> import array
>>> a = array.array('d',[1,2,3])
>>> sample.avg(a)
2.0
>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> p1
<capsule object "Point" at 0x1005d1e70>
>>> p2
<capsule object "Point" at 0x1005d1ea0>
>>> sample.distance(p1,p2)
2.8284271247461903
>>>
```

17.10.3

GIL

Cython C .pxd C .h .pyx .c cimport Cython .pxd Python

.pxd

csample.pxd int gcd(int, int) sample.pyx

```
cimport csample

def gcd(unsigned int x, unsigned int y):
    return csample.gcd(x,y)
```

Cython
C

```
>>> sample.gcd(-10,2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "sample.pyx", line 7, in sample.gcd (sample.c:1284)
    def gcd(unsigned int x,unsigned int y):
OverflowError: can't convert negative value to unsigned int
>>>
```

```
def gcd(unsigned int x, unsigned int y):
    if x <= 0:
        raise ValueError("x must be > 0")
    if y <= 0:
        raise ValueError("y must be > 0")
    return csample.gcd(x,y)
```

csample.pxd "in_mandel()"
bint int
Boolean 0 False 1 True

Cython C Python
divide()

```
def divide(x,y):
    cdef int rem
    quot = csample.divide(x,y,&rem)
    return quot, rem
```

rem C divide()
&rem C avg() Cython
def avg(double[:] a) avg()
numpy

```
>>> import array
>>> a = array.array('d',[1,2,3])
>>> import numpy
>>> b = numpy.array([1., 2., 3.])
>>> import sample
>>> sample.avg(a)
```

```
2.0
>>> sample.avg(b)
2.0
>>>
```

a.size0 &a[0]
<double *> &a[0] C avg()
Cython

avg()
with nogil: GIL
Python —— cdef
GIL csample.pxd avg()
double avg(double *, int) nogil .

Point Point
15.4 Cython
C Python C API

```
from cpython.pycapsule cimport *
from libc.stdlib cimport malloc, free
```

del_Point() Point()
Point * cdef del_Point() del_Point()
Cython Python ——
PyCapsule_New()
PyCapsule_GetPointer() Python C API

distance Point()
PyCapsule_GetPointer()
Cython distance()

Point

```
# sample.pyx

cimport csample
from libc.stdlib cimport malloc, free
...

cdef class Point:
    cdef csample.Point *_c_point
    def __cinit__(self, double x, double y):
        self._c_point = <csample.Point *> malloc(sizeof(csample.Point))
        self._c_point.x = x
        self._c_point.y = y

    def __dealloc__(self):
        free(self._c_point)

    property x:
```

```
        def __get__(self):
            return self._c_point.x
        def __set__(self, value):
            self._c_point.x = value

    property y:
        def __get__(self):
            return self._c_point.y
        def __set__(self, value):
            self._c_point.y = value

def distance(Point p1, Point p2):
    return csample.distance(p1._c_point, p2._c_point)
```

cdif Point Point cdef csample.Point *_c_point Point __cinit__() __dealloc__() malloc() free() C x y distance() Point C

Point

```
>>> import sample
>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> p1
<sample.Point object at 0x100447288>
>>> p2
<sample.Point object at 0x1004472a0>
>>> p1.x
2.0
>>> p1.y
3.0
>>> sample.distance(p1,p2)
2.8284271247461903
>>>
```

Cython

Cython

17.11 15.11 Cython

17.11.1

NumPy Cython

17.11.2

Cython

```
# sample.pyx (Cython)

cimport cython

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:] out):
    '''
    Clip the valu                                                    n out
    '''
    if min > max:
        raise ValueError("min must be <= max")
    if a.shape[0] != out.shape[0]:
        raise ValueError("input and output arrays must be the same size")
    for i in range(a.shape[0]):
        if a[i] < min:
            out[i] = min
        elif a[i] > max:
            out[i] = max
        else:
            out[i
```

setup.py python3

```
setup.py build_ext --inplace
```

B K 7 ` Q K

```
>>> a

array('d', [1.0, -3.0, 4.0, 7.0, 2.0, 0.0])
>>> sample.clip(a,1,4,a)
>>> a
array('d', [1.0, 1.0, 4.0, 4.0, 2.0, 1.0])

>>> # numpy example
>>> import numpy
>>> b = numpy.random.uniform(-10,10,size=1000000)
>>> b
array([-9.55546017,  7.45599334,  0.69248932, ...,  0.69583148,
       -3.86290931,  2.37266888])
>>> c = numpy.zeros_like(b)
>>> c
array([ 0.,  0.,  0., ...,  0.,  0.,  0.])
>>> sample.clip(b,-5,5,c)
>>> c
array([-5.        ,  5.        ,  0.69248932, ...,  0.69583148,
       -3.86290931,  2.37266888])
>>> min(c)
-5.0
>>> max(c)
5.0
>>>
```

numpy

`clip()`

```
>>> timeit('numpy.clip(b,-5,5,c)','from __main__ import b,c,numpy',number=1000)
8.093049556000551
>>> timeit('sample.clip(b,-5,5,c)','from __main__ import b,c,sample',
...         number=1000)
3.760528204000366
>>>
```

—— NumPy

C

17.11.3

Cython `cpdef clip()`

`clip()` C Python Cython

Cython

Cython clip()

`double[:] a` `double[:] out`

PEP 3118

NumPy array

NumPy numpy.zeros() numpy.zeros_like() numpy.empty() numpy.empty_like() .

a[i],out[i]

Cython

clip() @cython.boundscheck(False)

@cython.wraparound(False) Python 2.5

clip()

```
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:] out):
    if min > max:
        raise ValueError("min must be <= max")
    if a.shape[0] != out.shape[0]:
        raise ValueError("input and output arrays must be the same size")
    for i in range(a.shape[0]):
        out[i] = (a[i] if a[i] < max else max) if a[i] > min else min
```

50% 2.44 timeit() 3.76

C PK C

```
void clip(double *a, int n, double min, double max, double *out) {
  double x;
  for (; n >= 0; n--, a++, out++) {
    x = *a;

    *out = x > max ? max : (x < min ? min : x);
  }
}
```

C Cython 10%

GIL with nogil:

```
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:] out):
    if min > max:
```

f l r à

```
        raise ValueError("min must be <= max")
    if a.shape[0] != out.shape[0]:
        raise ValueError("input and output arrays must be the same size")
    with nogil:
        for i in range(a.shape[0]):
            out[i] = (a[i] if a[i] < max else max) if a[i] > min else min
```

```
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip2d(double[:,:] a, double min, double max, double[:,:] out):
    if min > max:
        raise ValueError("min must be <= max")
    for n in range(a.ndim):
        if a.shape[n] != out.shape[n]:
            raise TypeError("a and out have different shapes")
    for i in range(a.shape[0]):
        for j in range(a.shape[1]):
            if a[i,j] < min:
                out[i,j] = min
            elif a[i,j] > max:
                out[i,j] = max
            else:
                out[i,j] = a[i,j]
```

NumPy

PEP 3118 Cython “ ”

17.12 15.12

17.12.1

Python

17.12.2

ctypes p ´ § \ n j

ṅ

```
>>> addr
140735505915760

>>> # Turn the address into a callable function
>>> functype = ctypes.CFUNCTYPE(ctypes.c_double, ctypes.c_double)
>>> func = functype(addr)
>>> func
<CFunctionType object at 0x1006816d0>

>>> # Call the resulting function
>>> func(2)
0.9092974268256817
>>> func(0)
0.0
>>>
```

17.12.3

CFUNCTYPE CFUNCTYPE()

ctypes

LLVM

llvmpy

Python

```
>>> from llvm.core import Module, Function, Type, Builder
>>> mod = Module.new('example')
>>> f = Function.new(mod,Type.function(Type.double(), \
                    [Type.double(), Type.double()], False), 'foo')
>>> block = f.append_basic_block('entry')
>>> builder = Builder.new(block)
>>> x2 = builder.fmul(f.args[0],f.args[0])
>>> y2 = builder.fmul(f.args[1],f.args[1])
>>> r = builder.fadd(x2,y2)
>>> builder.ret(r)
<llvm.core.Instruction object at 0x10078e990>
>>> from llvm.ee import ExecutionEngine
>>> engine = ExecutionEngine.new(mod)
>>> ptr = engine.get_pointer_to_function(f)
>>> ptr
4325863440
>>> foo = ctypes.CFUNCTYPE(ctypes.c_double, ctypes.c_double, ctypes.c_double)(ptr)

>>> # Call the resulting function
>>> foo(2,3)
13.0
```

```
>>> foo(4,5)
41.0
>>> foo(1,2)
5.0
>>>
```

Python

Python

17.13 15.13 NULL C

17.13.1

NULL C

Python Unicode

17.13.2

C NULL char * .

C

```
void print_chars(char *s) {
    while (*s) {
        printf("%2x ", (unsigned char) *s);

        s++;
    }
    printf("\n");
}
```

```
print_chars("Hello");   // Outputs: 48 65 6c 6c 6f
```

Python C

PyArg_ParseTuple() ” y “

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  char *s;

  if (!PyArg_ParseTuple(args, "y", &s)) {
    return NULL;
  }
  print_chars(s);
  Py_RETURN_NONE;
}
```

NULL Unicode

```
>>> print_chars(b'Hello World')
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>> print_chars(b'Hello\x00World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be bytes without null bytes, not bytes
>>> print_chars('Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' does not support the buffer interface
>>>
```

Unicode PyArg_ParseTuple() " s "

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  char *s;

  if (!PyArg_ParseTuple(args, "s", &s)) {
    return NULL;
  }
  print_chars(s);
  Py_RETURN_NONE;
}
```

NULL UTF-8

```
>>> print_chars('Hello World')
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>> print_chars('Spicy Jalape\u00f1o')  # Note: UTF-8 encoding
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> print_chars('Hello\x00World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str without null characters, not str
>>> print_chars(b'Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not bytes
>>>
```

PyObject * PyArg_ParseTuple()
char *

```
/* Some Python Object (obtained somehow) */
PyObject *obj;

/* Conversion from bytes */
```

```
{
   char *s;
   s = PyBytes_AsString(o);
   if (!s) {
      return NULL;   /* TypeError already raised */
   }
   print_chars(s);
}

/* Conversion to UTF-8 bytes from a string */
{
   PyObject *bytes;
   char *s;
   if (!PyUnicode_Check(obj)) {
       PyErr_SetString(PyExc_TypeError, "Expected string");
       return NULL;
   }
   bytes = PyUnicode_AsUTF8String(obj);
   s = PyBytes_AsString(bytes);
   print_chars(s);
   Py_DECREF(bytes);
}
```

NULL

NULL

17.13.3

NULL Python

C

PyArg_ParseTuple()

“ s ”

UTF-8

ASCII

```
>>> import sys
>>> s = 'Spicy Jalape\u00f1o'
>>> sys.getsizeof(s)
87
>>> print_chars(s)     # Passing string
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> sys.getsizeof(s)   # Notice increased size
103
>>>
```

C

PyUnicode_AsUTF8String()

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  PyObject *o, *bytes;
  char *s;

  if (!PyArg_ParseTuple(args, "U", &o)) {
    return NULL;
  }
  bytes = PyUnicode_AsUTF8String(o);
  s = PyBytes_AsString(bytes);
  print_chars(s);
  Py_DECREF(bytes);
  Py_RETURN_NONE;
}
```

UTF-8

```
>>> import sys
>>> s = 'Spicy Jalape\u00f1o'
>>> sys.getsizeof(s)
87
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> sys.getsizeof(s)
87
>>>
```

NULL ctypes ctypes
NULL

```
>>> import ctypes
>>> lib = ctypes.cdll.LoadLibrary("./libsample.so")
>>> print_chars = lib.print_chars
>>> print_chars.argtypes = (ctypes.c_char_p,)
>>> print_chars(b'Hello World')
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>> print_chars(b'Hello\x00World')
48 65 6c 6c 6f
>>> print_chars('Hello World')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ctypes.ArgumentError: argument 1: <class 'TypeError'>: wrong type
>>>
```

UTF-8

```
>>> print_chars('Hello World'.encode('utf-8'))
48 65 6c 6c 6f 20 57 6f 72 6c 64
>>>
```

Swig Cython C

17.14 15.14 Unicode C

17.14.1

Python C
Unicode

17.14.2

C
Python Unicode Python C

C
char *, int wchar_t *, int

```
void print_chars(char *s, int len) {
  int n = 0;

  while (n < len) {
    printf("%2x ", (unsigned char) s[n]);
    n++;
  }
  printf("\n");
}

void print_wchars(wchar_t *s, int len) {
  int n = 0;
  while (n < len) {
    printf("%x ", s[n]);
    n++;
  }
  printf("\n");
}
```

print_chars() Python
UTF-8.

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  char *s;
  Py_ssize_t  len;

  if (!PyArg_ParseTuple(args, "s#", &s, &len)) {
    return NULL;
  }
  print_chars(s, len);
  Py_RETURN_NONE;
}
```

wchar_t

```
static PyObject *py_print_wchars(PyObject *self, PyObject *args) {
  wchar_t *s;
  Py_ssize_t  len;

  if (!PyArg_ParseTuple(args, "u#", &s, &len)) {
    return NULL;
  }
  print_wchars(s,len);
  Py_RETURN_NONE;
}
```

```
>>> s = 'Spicy Jalape\u00f1o'
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> print_wchars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 f1 6f
>>>
```

print_chars() UTF-8
print_wchars() Unicode

17.14.3

C C

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  char *s;
  Py_ssize_t  len;

  /* accepts bytes, bytearray, or other byte-like object */
  if (!PyArg_ParseTuple(args, "y#", &s, &len)) {
    return NULL;
  }
  print_chars(s, len);
  Py_RETURN_NONE;
}
```

Python 3
char * wchar_t * PEP
393 C C
PyArg_ParseTuple() ”s#” ”u#”

```
>>> import sys
>>> s = 'Spicy Jalape\u00f1o'
>>> sys.getsizeof(s)
87
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f
>>> sys.getsizeof(s)
103
>>> print_wchars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 f1 6f
>>> sys.getsizeof(s)
163
>>>
```

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  PyObject *obj, *bytes;
  char *s;
  Py_ssize_t   len;

  if (!PyArg_ParseTuple(args, "U", &obj)) {
    return NULL;
  }
  bytes = PyUnicode_AsUTF8String(obj);
  PyBytes_AsStringAndSize(bytes, &s, &len);
  print_chars(s, len);
  Py_DECREF(bytes);
  Py_RETURN_NONE;
}
```

wchar_t Python
ASCII
U+0000 U+FFFF
wchar_t *
wchar_t PyArg_ParseTuple() ”u#”

Unicode
C

```
static PyObject *py_print_wchars(PyObject *self, PyObject *args) {
  PyObject *obj;
  wchar_t *s;
  Py_ssize_t len;

  if (!PyArg_ParseTuple(args, "U", &obj)) {
    return NULL;
  }
  if ((s = PyUnicode_AsWideCharString(obj, &len)) == NULL) {
    return NULL;
```

```
  }
  print_wchars(s, len);
  PyMem_Free(s);
  Py_RETURN_NONE;
}
```

PyUnicode_AsWideCharString() wchar_t
C
bug Python
C UTF-8 Python

```
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  char *s = 0;
  int   len;
  if (!PyArg_ParseTuple(args, "es#", "encoding-name", &s, &len)) {
    return NULL;
  }
  print_chars(s, len);
  PyMem_Free(s);
  Py_RETURN_NONE;
}
```

Unicode

```
static PyObject *py_print_wchars(PyObject *self, PyObject *args) {
  PyObject *obj;
  int n, len;
  int kind;
  void *data;

  if (!PyArg_ParseTuple(args, "U", &obj)) {
    return NULL;
  }
  if (PyUnicode_READY(obj) < 0) {
    return NULL;
  }

  len = PyUnicode_GET_LENGTH(obj);
  kind = PyUnicode_KIND(obj);
  data = PyUnicode_DATA(obj);

  for (n = 0; n < len; n++) {
    Py_UCS4 ch = PyUnicode_READ(kind, data, n);
    printf("%x ", ch);
  }
  printf("\n");
  Py_RETURN_NONE;
}
```

PyUnicode_KIND() PyUnicode_DATA() Unicode

PEP 393 kind 8 16 32

PyUnicode_READ()

Python Unicode C UTF-8 UTF-8. UTF-8

Unicode

17.15 15.15 C Python

17.15.1

C Python

17.15.2

C char * int Unicode Py_BuildValue()

```
char *s;     /* Pointer to C string data */
int   len;   /* Length of data */

/* Make a bytes object */
PyObject *obj = Py_BuildValue("y#", s, len);
```

Unicode s UTF-8

```
PyObject *obj = Py_BuildValue("s#", s, len);
```

s PyUnicode_Decode()

```
PyObject *obj = PyUnicode_Decode(s, len, "encoding", "errors");

/* Examples /*
obj = PyUnicode_Decode(s, len, "latin-1", "strict");
obj = PyUnicode_Decode(s, len, "ascii", "ignore");
```

wchar_t *, len Py_BuildValue()

```
wchar_t *w;    /* Wide character string */
int len;       /* Length */

PyObject *obj = Py_BuildValue("u#", w, len);
```

PyUnicode_FromWideChar() :

```
PyObject *obj = PyUnicode_FromWideChar(w, len);
```

—— Unicode
Python

17.15.3

C Python I/O C
ASCII
Latin-1 UTF-8.
Python
C
NULL

17.16 15.16 C

17.16.1

C Python C
C UTF-8
Python

17.16.2

C

```
/* Some dubious string data (malformed UTF-8) */
const char *sdata = "Spicy Jalape\xc3\xb1o\xae";
int slen = 16;

/* Output character data */
void print_chars(char *s, int len) {
  int n = 0;
  while (n < len) {
    printf("%2x ", (unsigned char) s[n]);
    n++;
  }
  printf("\n");
}
```

sdata UTF-8 C
print_chars(sdata, slen) sdata

Python
print_chars()

```
/* Return the C string back to Python */
static PyObject *py_retstr(PyObject *self, PyObject *args) {
  if (!PyArg_ParseTuple(args, "")) {
    return NULL;
  }
  return PyUnicode_Decode(sdata, slen, "utf-8", "surrogateescape");
}

/* Wrapper for the print_chars() function */
static PyObject *py_print_chars(PyObject *self, PyObject *args) {
  PyObject *obj, *bytes;
  char *s = 0;
  Py_ssize_t   len;

  if (!PyArg_ParseTuple(args, "U", &obj)) {
    return NULL;
  }

  if ((bytes = PyUnicode_AsEncodedString(obj,"utf-8","surrogateescape"))
        == NULL) {
    return NULL;
  }
  PyBytes_AsStringAndSize(bytes, &s, &len);
  print_chars(s, len);
  Py_DECREF(bytes);
  Py_RETURN_NONE;
}
```

Python

```
>>> s = retstr()
>>> s
'Spicy Jalapeño\udcae'
>>> print_chars(s)
53 70 69 63 79 20 4a 61 6c 61 70 65 c3 b1 6f ae
>>>
```

Python
C C

17.16.3

C Python Unicode /
C Python

Unicode

```
>>> raw = b'Spicy Jalape\xc3\xb1o\xae'
>>> raw.decode('utf-8','ignore')
'Spicy Jalapeño'
>>> raw.decode('utf-8','replace')
'Spicy Jalapeño?'
>>>
```

surrogateeescape
udcXX XX

```
>>> raw.decode('utf-8','surrogateescape')
'Spicy Jalapeño\udcae'
>>>
```

\udcae Unicode

```
>>> s = raw.decode('utf-8', 'surrogateescape')
>>> print(s)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'utf-8' codec can't encode character '\udcae'
in position 14: surrogates not allowed
>>>
```

C Python C
surrogateeescape

```
>>> s
'Spicy Jalapeño\udcae'
>>> s.encode('utf-8','surrogateeescape')
b'Spicy Jalape\xc3\xb1o\xae'
>>>
```

——

Python

os.listdir()

5.15

PEP 383 surrogateeescape

17.17 15.17 C

17.17.1

C

17.17.2

```
static PyObject *py_get_filename(PyObject *self, PyObject *args) {
  PyObject *bytes;
  char *filename;
  Py_ssize_t len;
  if (!PyArg_ParseTuple(args,"O&", PyUnicode_FSConverter, &bytes)) {
    return NULL;
  }
  PyBytes_AsStringAndSize(bytes, &filename, &len);
  /* Use filename */
  ...

  /* Cleanup and return */
  Py_DECREF(bytes)
  Py_RETURN_NONE;
}
```

PyObject *

```
PyObject *obj;    /* Object with the filename */
PyObject *bytes;
char *filename;
Py_ssize_t len;

bytes = PyUnicode_EncodeFSDefault(obj);
PyBytes_AsStringAndSize(bytes, &filename, &len);
/* Use filename */
...

/* Cleanup */
Py_DECREF(bytes);

If you need to return a filename back to Python, use the following code:

/* Turn a filename into a Python object */

char *filename;       /* Already set */
int   filename_len;   /* Already set */
```

```
PyObject *obj = PyUnicode_DecodeFSDefaultAndSize(filename, filename_len);
```

17.17.3

Python
Python

/

17.18 15.18 C

17.18.1

Python C

17.18.2

PyFile_FromFd()

```
PyObject *fobj;     /* File object (already obtained somehow) */
int fd = PyObject_AsFileDescriptor(fobj);
if (fd < 0) {
   return NULL;
}
```

fobj fileno()

C

Python

PyFile_FromFd() :

```
int fd;     /* Existing file descriptor (already open) */
PyObject *fobj = PyFile_FromFd(fd, "filename","r",-1,NULL,NULL,NULL,1);
```

PyFile_FromFd() open() NULL

17.18.3

Python C Python io
I/O C
I/O

C Python C
Python
`PyFile_FromFd()` 1 Python

C I/O `fdopen()`
`FILE *` I/O
I/O Python `io` C `stdio` C
`fclose()` Python
<stdio.h>

17.19 15.19 C

17.19.1

C Python
StringIO

17.19.2

`read()`

C

```
#define CHUNK_SIZE 8192

/* Consume a "file-like" object and write bytes to stdout */
static PyObject *py_consume_file(PyObject *self, PyObject *args) {
  PyObject *obj;
  PyObject *read_meth;
  PyObject *result = NULL;
  PyObject *read_args;

  if (!PyArg_ParseTuple(args,"O", &obj)) {
    return NULL;
  }

  /* Get the read method of the passed object */
  if ((read_meth = PyObject_GetAttrString(obj, "read")) == NULL) {
    return NULL;
  }

  /* Build the argument list to read() */
  read_args = Py_BuildValue("(i)", CHUNK_SIZE);
```

```
  while (1) {
    PyObject *data;
    PyObject *enc_data;
    char *buf;
    Py_ssize_t len;

    /* Call read() */
    if ((data = PyObject_Call(read_meth, read_args, NULL)) == NULL) {
      goto final;
    }

    /* Check for EOF */
    if (PySequence_Length(data) == 0) {
      Py_DECREF(data);
      break;
    }

    /* Encode Unicode as Bytes for C */
    if ((enc_data=PyUnicode_AsEncodedString(data,"utf-8","strict"))==NULL) {
      Py_DECREF(data);
      goto final;
    }

    /* Extract underlying buffer data */
    PyBytes_AsStringAndSize(enc_data, &buf, &len);

    /* Write to stdout (replace with something more useful) */
    write(1, buf, len);

    /* Cleanup */
    Py_DECREF(enc_data);
    Py_DECREF(data);
  }
  result = Py_BuildValue("");

 final:
  /* Cleanup */
  Py_DECREF(read_meth);
  Py_DECREF(read_args);
  return result;
}
```

StringIO

```
>>> import io
>>> f = io.StringIO('Hello\nWorld\n')
>>> import sample
>>> sample.consume_file(f)
Hello
World
>>>
```

17.19.3

C Python C API

read()
PyObject_Call() EOF
PySequence_Length() 0.

I/O Unicode

C

```
...
/* Call read() */
if ((data = PyObject_Call(read_meth, read_args, NULL)) == NULL) {
  goto final;
}

/* Check for EOF */
if (PySequence_Length(data) == 0) {
  Py_DECREF(data);
  break;
}
if (!PyBytes_Check(data)) {
  Py_DECREF(data);
  PyErr_SetString(PyExc_IOError, "File must be in binary mode");
  goto final;
}

/* Extract underlying buffer data */
PyBytes_AsStringAndSize(data, &buf, &len);
...
```

PyObject * ``
``Py_DECREF()

write()
Python Unicode

readline(), read_info()
read() write() C

17.20 15.20 C

17.20.1

C

17.20.2

C

```
static PyObject *py_consume_iterable(PyObject *self, PyObject *args) {
  PyObject *obj;
  PyObject *iter;
  PyObject *item;

  if (!PyArg_ParseTuple(args, "O", &obj)) {
    return NULL;
  }
  if ((iter = PyObject_GetIter(obj)) == NULL) {
    return NULL;
  }
  while ((item = PyIter_Next(iter)) != NULL) {
    /* Use item */
    ...
    Py_DECREF(item);
  }

  Py_DECREF(iter);
  return Py_BuildValue("");
}
```

17.20.3

Python PyObject_GetIter()
iter() PyIter_Next() next
NULL( ) —— Py_DECREF()

17.21 15.21

17.21.1

Python

17.21.2

faulthandler

```
import faulthandler
faulthandler.enable()
```

-Xfaulthandler Python

```
bash % python3 -Xfaulthandler program.py
```

PYTHONFAULTHANDLER faulthandler C
Python

```
Fatal Python error: Segmentation fault

Current thread 0x00007fff71106cc0:
  File "example.py", line 6 in foo
  File "example.py", line 10 in bar
  File "example.py", line 14 in spam
  File "example.py", line 19 in <module>
Segmentation fault
```

C Python

17.21.3

faulthandler Python
pdb Python

faulthandler C C
gdb faulthandler
C C
faulthandler

CHAPTER

EIGHTEEN

A

18.1

http://docs.python.org

Python
python 3

http://www.python.org/dev/peps

python PEPs
Python Enhancement Proposals—-Python
PEPS

http://pyvideo.org

PyCon
python Python
Python 3

http://code.activestate.com/recipes/langs/python

ActiveState Python
300 Python3

http://stackoverflow.com/questions/tagged/python

Stack Overflow 175,000 Python
5000 Python 3

18.2 Python

Python Python 3

Beginning Python: From Novice to Professional, 2nd Edition, by Magnus Lie Het - land, Apress (2008). Programming in Python 3, 2nd Edition, by Mark Summerfield, Addison-Wesley (2010).

- *Learning Python* Mark Lutz O ' Reilly & Associates (2009)
- *The Quick Python Book* Vernon Ceder Manning (2010)
- *Python Programming for the Absolute Beginner* Michael Dawson Course Technology PTR (2010).
- *Beginning Python: From Novice to Professional* Magnus Lie Het - land Apress (2008).
- *Programming in Python 3* Mark Summerfield Addison-Wesley (2010).

18.3

Python 3

- *Programming Python* , by Mark Lutz, O ' Reilly & Associates (2010).
- *Python Essential Reference* David Beazley, Addison-Wesley (2009).
- *Core Python Applications Programming* Wesley Chun, Prentice Hall (2012).
- *The Python Standard Library by Example* Doug Hellmann Addison-Wesley (2011).
- *Python 3 Object Oriented Programming* Dusty Phillips, Packt Publishing (2010).
- *Porting to Python 3* Lennart Regebro CreateSpace (2011), http://python3porting.com.

CHAPTER

NINETEEN

-
- yidao620
- Email yidao620@gmail.com
- http://yidao620c.github.io/
- GitHub https://github.com/yidao620c

CHAPTER

TWENTY

ROADMAP

2014/08/10 - 2014/08/31:

```
|   github          readthedocs
|
```

2014/09/01 - 2014/10/31:

```
|      4
```

2014/11/01 - 2015/01/31:

```
|      8
```

2015/02/01 - 2015/03/31:

```
|      9
```

2015/04/01 - 2015/05/31:

```
|   10
```

2015/06/01 - 2015/06/30:

```
|   11
```

2015/07/01 - 2015/07/31:

```
|   12
```

2015/08/01 - 2015/08/31:

```
|   13
```

2015/09/01 - 2015/11/30:

```
|   14
```

2015/12/01 - 2015/12/20:

```
|   15
```

2015/12/21 - 2015/12/31:

```
|
```

2016/01/01 - 2016/01/10:

```
|                    1.0                PDF
```