... c = 3

...

>>> c = C()

>>> vars(c)

{}

>>> c.c

3

>>> c.c = 5

>>> c.c

5

>>> C.c

3

>>> c.\_\_class\_\_

<class '\_\_main\_\_.C'>

>>> c.\_\_class\_\_ = B

>>> c

<\_\_main\_\_.B object at 0xb70c2cec>

>>> c.c

5

>>> c.b

2

>>> B.\_\_bases\_\_

(<class '\_\_main\_\_.A'>,)

>>> B.\_\_mro\_\_

(<class '\_\_main\_\_.B'>, <class '\_\_main\_\_.A'>, <type 'object'>)

>>> class D(B, C):

... pass

...

>>> D.\_\_mro\_\_

(<class '\_\_main\_\_.D'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.A'>, <class '\_\_main\_\_.C'>, <type 'object'>)

>>> dir(object)

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_doc\_\_', '\_\_format\_\_', '\_\_getattribute\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_']

>>> class Person(object):

... def \_\_init\_\_(self, name, age):

... self.name = name

... self.age = age

...

>>> p = Person('Bill', 25)

>>> p

<\_\_main\_\_.Person object at 0xb70c2e8c>

>>> class Person(object):

... def \_\_init\_\_(self, name, age):

... self.name = name

... self.age = age

... def \_\_repr\_\_(self):

... return self.name

...

>>> p = Person('Bill', 25)

>>> p

Bill

>>> print p

Bill

>>> class Person(object):

... def \_\_init\_\_(self, name, age):

... self.name = name

... self.age = age

... def \_\_repr\_\_(self):

... return self.name

... def \_\_str\_\_(self):

... return "Person: <{}, {}>".format(self.name, self.age)

...

>>> p = Person('Bill', 25)

>>> p

Bill

>>> print p

Person: <Bill, 25>

>>> str(p)

'Person: <Bill, 25>'

>>> p1 = Person('Bill', 25)

>>>

>>> p == p1

False

>>> id(p)

3071050348L

>>> id(p1)

3071050444L

>>> class Person(object):

... def \_\_init\_\_(self, name, age):

... self.name = name

... self.age = age

... def \_\_repr\_\_(self):

... return self.name

... def \_\_str\_\_(self):

... return "Person: <{}, {}>".format(self.name, self.age)

... def \_\_eq\_\_(self, other):

... return self.name == other.name

...

>>> p = Person('Bill', 25)

>>> p1 = Person('Bill', 25)

>>> p == p1

True

>>> class Person(object):

... def \_\_init\_\_(self, name, age):

... self.name = name

... self.age = age

... def \_\_repr\_\_(self):

... return self.name

... def \_\_str\_\_(self):

... return "Person: <{}, {}>".format(self.name, self.age)

... def \_\_eq\_\_(self, other):

... return self.name == other.name

... def \_\_lt\_\_(self, other):

... return self.name < other.name

...

>>> p = Person('Bill', 25)

>>> p1 = Person('Bob', 25)

>>> p < p1

True

>>> 'Bill' < 'Bob'

True

>>> p != p1

True

>>> p >= p1

True

>>> l = [Person('John', 32), Person('Bill', 23), Person('Bob', 43)]

>>> l.sort()

>>> l

[Bill, Bob, John]

>>> class Person(object):

... def \_\_init\_\_(self, name, age):

... self.name = name

... self.age = age

... def \_\_repr\_\_(self):

... return "Person: <{}, {}>".format(self.name, self.age)

... def \_\_str\_\_(self):

... return self.name

... def \_\_eq\_\_(self, other):

... return self.name == other.name

... def \_\_lt\_\_(self, other):

... return self.name < other.name

... def \_\_add\_\_(self, other):

... return "Not Implemented"

...

>>> p = Person('Bill', 25)

>>> p1 = Person('Bob', 25)

>>> p + p1

'Not Implemented'

>>> import operator

>>> dir(operator)

['\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_concat\_\_', '\_\_contains\_\_', '\_\_delitem\_\_', '\_\_delslice\_\_', '\_\_div\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_floordiv\_\_', '\_\_ge\_\_', '\_\_getitem\_\_', '\_\_getslice\_\_', '\_\_gt\_\_', '\_\_iadd\_\_', '\_\_iand\_\_', '\_\_iconcat\_\_', '\_\_idiv\_\_', '\_\_ifloordiv\_\_', '\_\_ilshift\_\_', '\_\_imod\_\_', '\_\_imul\_\_', '\_\_index\_\_', '\_\_inv\_\_', '\_\_invert\_\_', '\_\_ior\_\_', '\_\_ipow\_\_', '\_\_irepeat\_\_', '\_\_irshift\_\_', '\_\_isub\_\_', '\_\_itruediv\_\_', '\_\_ixor\_\_', '\_\_le\_\_', '\_\_lshift\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_name\_\_', '\_\_ne\_\_', '\_\_neg\_\_', '\_\_not\_\_', '\_\_or\_\_', '\_\_package\_\_', '\_\_pos\_\_', '\_\_pow\_\_', '\_\_repeat\_\_', '\_\_rshift\_\_', '\_\_setitem\_\_', '\_\_setslice\_\_', '\_\_sub\_\_', '\_\_truediv\_\_', '\_\_xor\_\_', 'abs', 'add', 'and\_', 'attrgetter', 'concat', 'contains', 'countOf', 'delitem', 'delslice', 'div', 'eq', 'floordiv', 'ge', 'getitem', 'getslice', 'gt', 'iadd', 'iand', 'iconcat', 'idiv', 'ifloordiv', 'ilshift', 'imod', 'imul', 'index', 'indexOf', 'inv', 'invert', 'ior', 'ipow', 'irepeat', 'irshift', 'isCallable', 'isMappingType', 'isNumberType', 'isSequenceType', 'is\_', 'is\_not', 'isub', 'itemgetter', 'itruediv', 'ixor', 'le', 'lshift', 'lt', 'methodcaller', 'mod', 'mul', 'ne', 'neg', 'not\_', 'or\_', 'pos', 'pow', 'repeat', 'rshift', 'sequenceIncludes', 'setitem', 'setslice', 'sub', 'truediv', 'truth', 'xor']

>>> d = {'q': 1, 'a': 2, 3: 's'}

>>> d[[1,2,3]] = 2

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unhashable type: 'list'

>>> d[(1,2,3)] = 2

>>> d

{'q': 1, 'a': 2, 3: 's', (1, 2, 3): 2}

>>> class Person(object):

... def \_\_init\_\_(self, name, age):

... self.name = name

... self.age = age

... def \_\_repr\_\_(self):

... return "Person: <{}, {}>".format(self.name, self.age)

... def \_\_str\_\_(self):

... return self.name

... def \_\_eq\_\_(self, other):

... return self.name == other.name

... def \_\_lt\_\_(self, other):

... return self.name < other.name

... def \_\_add\_\_(self, other):

... return "Not Implemented"

... def \_\_hash\_\_(self):

... return hash(self.name)

...

>>> d = {Person('Bill',21), Person('Bill', 23),Person('John',23)}

>>> d

set([Person: <John, 23>, Person: <Bill, 23>])

>>> d = {Person('Bill',21):1, Person('Bill', 23):2,Person('John',23):$}

File "<stdin>", line 1

d = {Person('Bill',21):1, Person('Bill', 23):2,Person('John',23):$}

^

SyntaxError: invalid syntax

>>> d = {Person('Bill',21):1, Person('Bill', 23):2,Person('John',23):3}

>>> d

{Person: <Bill, 21>: 2, Person: <John, 23>: 3}

>>> d = [Person('Bill',21), Person('Bill', 23),Person('John',23)]

>>> d

[Person: <Bill, 21>, Person: <Bill, 23>, Person: <John, 23>]

>>> s = {1,2,3,1}

>>> s

set([1, 2, 3])

>>> {1,2} | {2, 3}

set([1, 2, 3])

>>> {1,2} & {2, 3}

set([2])

>>> {1,2} - {2, 3}

set([1])

>>> {1,2} ^ {2, 3}

set([1, 3])

>>> class A(object):

... def \_\_new\_\_(cls):

... print "New"

... return object.\_\_new\_\_(cls)

... def \_\_init\_\_(self):

... print "Init"

...

>>> a = A()

New

Init

>>> class A(object):

... def \_\_new\_\_(cls):

... return 1

...

>>> a = A()

>>> a

1

>>> type(a)

<type 'int'>

>>> class Car(object):

... def run(self):

... print "RRRRR"

...

>>> class Truck(object):

... def run(self):

... print "BZZZZ"

...

>>> class Vehicle(object):

... class \_\_new\_\_(cls, name):

... if name == "Car":

... return object.\_\_new\_\_(Car)

... elif name == "Truck":

... return object.\_\_new\_\_(Truck)

... else:

... raise ValueError

...

File "<stdin>", line 4

SyntaxError: 'return' outside function

>>> class Vehicle(object):

... def \_\_new\_\_(cls, name):

... if name == "Car":

... return object.\_\_new\_\_(Car)

... elif name == "Truck":

... return object.\_\_new\_\_(Truck)

... else:

... raise ValueError

...

>>> Vehicle()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: \_\_new\_\_() takes exactly 2 arguments (1 given)

>>> c = Vehicle("Car")

>>> c.run()

RRRRR

>>> c = Vehicle("Truck")

>>> c.run()

BZZZZ

>>> class Singleton(object):

... def \_\_new\_\_(cls):

... if hasattr(cls, 'instance'):

... return cls.instance

... cls.instance = object.\_\_new\_\_(cls)

... return cls.instance

...

>>> s = Singleton()

>>> s1 = Singleton()

>>> s is s1

True

>>> s.a = 1

>>> s.a1

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'Singleton' object has no attribute 'a1'

>>> s1.a

1

>>> dir(object)

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_doc\_\_', '\_\_format\_\_', '\_\_getattribute\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_']

>>> class V(object):

... def \_\_setattr\_\_(self, name, value):

... print "{} = {}".format(name, value)

... super(V, self).\_\_setattr\_\_(name, value)

... def \_\_delattr\_\_(self, name):

... print "Delete {}".format(name)

... super(V, self).\_\_delattr\_\_(name)

... def \_\_getattribute\_\_(self, name):

... print "Get {}".format(name)

... super(V, self).\_\_getattribute\_\_(name)

... def \_\_getattr\_\_(self, name):

... print "{} doesn't exist".format(name)

...

>>> v = V()

>>>

>>> v.a = 1

a = 1

>>> vars(v)

Get \_\_dict\_\_

>>> v.\_\_dict\_\_

Get \_\_dict\_\_

>>> class V(object):

... def \_\_setattr\_\_(self, name, value):

... print "{} = {}".format(name, value)

... super(V, self).\_\_setattr\_\_(name, value)

... def \_\_delattr\_\_(self, name):

... print "Delete {}".format(name)

... super(V, self).\_\_delattr\_\_(name)

... def \_\_getattribute\_\_(self, name):

... print "Get {}".format(name)

... return super(V, self).\_\_getattribute\_\_(name)

... def \_\_getattr\_\_(self, name):

... print "{} doesn't exist".format(name)

...

>>> v = V()

>>> v.a

Get a

a doesn't exist

>>> v.a = 1

a = 1

>>> vars(a)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: vars() argument must have \_\_dict\_\_ attribute

>>> vars(v)

Get \_\_dict\_\_

{'a': 1}

>>> v.a

Get a

1

>>> del v.a

Delete a

>>> v.a

Get a

a doesn't exist

>>> class V(object):

... def \_\_init\_\_(self, a):

... self.a = 1

... def \_\_setattr\_\_(self, name, value):

... print "{} = {}".format(name, value)

... super(V, self).\_\_setattr\_\_(name, value)

... def \_\_delattr\_\_(self, name):

... print "Delete {}".format(name)

... super(V, self).\_\_delattr\_\_(name)

... def \_\_getattribute\_\_(self, name):

... print "Get {}".format(name)

... return super(V, self).\_\_getattribute\_\_(name)

... def \_\_getattr\_\_(self, name):

... print "{} doesn't exist".format(name)

...

>>> v = V()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: \_\_init\_\_() takes exactly 2 arguments (1 given)

>>> class V(object):

... def \_\_init\_\_(self):

... self.a = 1

... def \_\_setattr\_\_(self, name, value):

... print "{} = {}".format(name, value)

... super(V, self).\_\_setattr\_\_(name, value)

... def \_\_delattr\_\_(self, name):

... print "Delete {}".format(name)

... super(V, self).\_\_delattr\_\_(name)

... def \_\_getattribute\_\_(self, name):

... print "Get {}".format(name)

... return super(V, self).\_\_getattribute\_\_(name)

... def \_\_getattr\_\_(self, name):

... print "{} doesn't exist".format(name)

...

>>> v = V()

a = 1

>>> v.a

Get a

1

>>> v.\_\_dict\_\_

Get \_\_dict\_\_

{'a': 1}

>>> class A(object):

... def m0(self):

... print "m0"

... def m1(self):

... print "m1"

...

>>> class B(object):

... def \_\_init\_\_(self):

... self.a = A()

... def m2(self):

... print "m2"

... def \_\_getattr\_\_(self, name):

... getattr(self.a, name)()

...

>>> b = B()

>>> b.m2()

m2

>>> b.m0()

m0

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'NoneType' object is not callable

>>> class B(object):

... def \_\_init\_\_(self):

... self.a = A()

... def m2(self):

... print "m2"

... def \_\_getattr\_\_(self, name):

... return getattr(self.a, name)

...

>>> b = B()

>>> b.m0()

m0

>>> b.m1()

m1

>>> class B(object):

... def \_\_init\_\_(self):

... self.a = A()

... def m2(self):

... print "m2"

... def \_\_getattr\_\_(self, name):

... if name == 'm1':

... raise AttributeError

... return getattr(self.a, name)

...

>>> b = B()

>>> b.m0()

m0

>>> b.m1()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 8, in \_\_getattr\_\_

AttributeError

>>> class A(object):

... def

File "<stdin>", line 2

def

^

SyntaxError: invalid syntax

>>>

>>> class mult(object):

... def \_\_init\_\_(self, num):

... self.num = num

... def \_\_call\_\_(self, x):

... return self.num \* x

...

>>> double = mult(2)

>>> double(3)

6

>>> def scale(f):

... def wrapper(x):

... print "In"

... res = f(10 \* x)

... print "Out"

... return res

... return wrapper

...

>>> def ident(x):

... return x

...

>>> ident = scale(ident)

>>>

>>> ident(2)

In

Out

20

>>> @scale

... def ident(x):

... return x

...

>>> ident(10)

In

Out

100

>>> def scale(n):

... def decorator(f):

... def wrapper(x):

... return f(n \* x)

... def wrapper

File "<stdin>", line 5

def wrapper

^

SyntaxError: invalid syntax

>>> def scale(n):

... def decorator(f):

... def wrapper(x):

... return f(n \* x)

... return wrapper

... return decorator

...

>>> @scale(5)

... def ident(x):

... return x

...

>>> ident(5)

25

>>> class scale(object):

... def \_\_init\_\_(self, n):

... self.n = n

... def \_\_call\_\_(self, f):

... def wrapper(x):

... return f(self.n \* x)

... return wrapper

...

>>> @scale(5)

... def ident(x):

... return x

...

>>> ident(5)

25

>>>

>>> def connect():

... print "Connect"

... retutn 11

File "<stdin>", line 3

retutn 11

^

SyntaxError: invalid syntax

>>>

>>> def connect():

... print "Connect"

... return 11

...

>>> class DB(object):

... conn = connect()

...

Connect

>>> class MyList(list):

... def sum(self):

... s = 0

... for i in self:

... s += i

... return s

...

>>> l = MyList()

>>> l

[]

>>> l.extend([1,2,3])

>>> l

[1, 2, 3]

>>> l.sum()

6

>>> dir(l)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_delslice\_\_', '\_\_dict\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getslice\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_setslice\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort', 'sum']

>>> class MyList(object):

... def \_\_init\_\_(self):

... self.l = [0 for i in range(10)]

... def \_\_repr\_\_(self):

... return str(self.l)

... def \_\_setitem\_\_(self, index, value):

... self.l[index] = value

... def \_\_getitem\_\_(self, index):

... if not 0 <= index < 10:

... raise IndexError

... return self.l[index]

...

>>> l = MyList()

>>> l[1]

0

>>> l[2] = 5

>>> for i in l:

... print i

...

0

0

5

0

0

0

0

0

0

0

>>> dir({})

['\_\_class\_\_', '\_\_cmp\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'has\_key', 'items', 'iteritems', 'iterkeys', 'itervalues', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values', 'viewitems', 'viewkeys', 'viewvalues']

>>> class A(object):

... class B(object):

... pass

... pass

...

>>> class A(object):

... def \_\_init\_\_(self, x):

... self.x = x

... def get\_double\_x(self):

... return 2 \* self.x

... @staticmethod

... def get\_smth():

... return 42

...

>>> a = A(2)

>>> a.x

2

>>> a.get\_double\_x()

4

>>> A.get\_double\_x()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unbound method get\_double\_x() must be called with A instance as first argument (got nothing instead)

>>> a.get\_smth()

42

>>> A.get\_smth()

42