类树：

例：程序class\_detail/class\_detail7.py

def class\_tree(cls, indent):  
 print('.' \* indent + cls.\_\_name\_\_)  
 for super\_cls in cls.\_\_bases\_\_: # 查找超类  
 class\_tree(super\_cls, indent + 3)  
  
  
def instance\_tree(inst):  
 print('Tree of {0}'.format(inst))  
 class\_tree(inst.\_\_class\_\_, 3)  
  
  
def test():  
 class A: pass  
  
 class B(A): pass  
  
 class C(A): pass  
  
 class D(B, C): pass  
  
 class E: pass  
  
 class F(D, E): pass  
  
 instance\_tree(B())  
 instance\_tree(F())  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test()

输出为：

Tree of <\_\_main\_\_.test.<locals>.B object at 0x0000024879E0CB70>

...B

......A

.........object

Tree of <\_\_main\_\_.test.<locals>.F object at 0x0000024879E0CB70>

...F

......D

.........B

............A

...............object

.........C

............A

...............object

......E

.........object

类扩展Python内置类型：

例：程序class\_detail/class\_detail13.py

# 扩展内置类型list  
class MyList(list):  
 def \_\_getitem\_\_(self, item):  
 print('indexing {0} at {1}'.format(self, item))  
 return list.\_\_getitem\_\_(self, item - 1)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x = MyList('abcd')  
 print(x)  
 print(x[1])  
 print(x[3])  
 print()  
 x.append('spam')  
 print(x)  
 x.reverse()  
 print(x)

输出为：

['a', 'b', 'c', 'd']

indexing ['a', 'b', 'c', 'd'] at 1

a

indexing ['a', 'b', 'c', 'd'] at 3

c

['a', 'b', 'c', 'd', 'spam']

['spam', 'd', 'c', 'b', 'a']