Python Learning

date: Feb. 11, 2017
author: Xiang

This document is sorted out form http://www.liaoxuefeng.com/ python tutorial.

Function

Cast

```
int('123')
float('12.34')
str(1.23)
bool(1)   # True
bool('')  # False
```

函数别名

```
a = abs
a(-1)     # 1
a = ddd   # NameError
```

Define a Funciton

Notice `return`, if a function have not `return`, it will run `return none`.

```
def my_abs(x):
    if x >= 0:
        return x
    else:
        return -x
```

空函数 `pass #啥也不做`

```
if age >= 18:
  pass
```

Paras Checking

CAUSION： isinstance()

类型检查用build_in函数`isinstance()`实现：

```
def my_abs(x):
    if not isinstance(x,(int,float)):
      raise TypeError('bad operand type')
    if x >= 0:
        return x
    else:
        return -x
```

Return Multiple Value

```
import math

def move(x,y,step,angle=0):
   nx = x+step*math.cos(angle)
   ny = y-step*math.sin(angle)
   return nx,ny
```

CAUSION： return a tuple ！

```
x, y = move(100, 100, 60, math.pi/6)
print(x,y)    #return 151.96152422706632 70.0

r = move(100, 100, 60, math.pi/6)
print(r)    #return a tuple: (151.96152422706632, 70.0)
```

位置参数

```
def power(x):
  return x*x
```

默认参数

***CAUSION*： *a* 坑**

```
def power(x, n=2):     # n那就是默认参数
    s = 1
    while n > 0:
        n = n - 1
        s = s * x
    return s
```

输入`>>>power(3,4)`时，可算出3^4.
当输入`>>>power(5)`时，仍然可以算出5^2.
注意几点：

- 必选参数在前，默认参数在后
- 变化大的参数放前面，变化小的参数放后面
- 不按顺序提供部分默认参数时，需要把参数名写上
  如：输入`>>>power(n=3, x=2)`
- 默认参数的一个坑, 如下：

```
# 坑代码展示如下，以后code时小心为妙：

def add_end(L=[]):
    L.append('END')
    return L

print(add_end())
print(add_end())
print(add_end())

#上面将输出：
#['END']
#['END', 'END']
#['END', 'END', 'END']


print(add_end([]))
print(add_end([]))
print(add_end([]))

#这个将输出：
#['END']
#['END']
#['END']

# We change the code like this:
def add_end7(L = None):
    if L is None:
        L = []
    L.append('END')
    return L

print(add_end7())
print(add_end7())
print(add_end7())

#这个将输出：
#['END']
#['END']
#['END']

# 自己体会这个坑吧
```

可变参数 *var list

有时候，由于参数个数无法事先确定，我们就把参数们作为一个list或tuple传进来，如下：

CAUSION: 用 `*variable` 表示可变参数

```
# CAUSION: the *
def calc(*numbers):
    sum = 0
    for n in numbers:
        sum = sum + n*n
    return sum

# def calc(numbers):
# print(calc([1,2,3]))        # list
# print(calc((1,2,3,4)))      # tuple

# def calc(*numbers):
print(calc(1,2,3))
print(calc(1,2,3,4))
print(calc())   # 无参数也可
```

CAUSION: `*nums` 表示把 `nums` 这个***list***的所有元素作为可变参数传进去

```
nums = [11, 2, 3]
result=calc(*nums)
# *nums表示把nums这个list的所有元素作为可变参数传进去。
# 这种写法相当有用，而且很常见。
print(result)
```

关键字参数 **kw dict

CAUSION: 本节全部

- 可变参数允许你传入0个或任意个参数，这些可变参数在函数调用时自动组装为一个**tuple**;
- 关键字参数允许你传入0个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个**dict**。
  用途： 试想你正在做一个用户注册的功能，除了用户名和年龄是必填项外，其他都是可选项，利用关键字参数来定义这个函数就能满足注册的需求。

```
def person(name, age, **kw):
    print('name: ', name, '  age: ', age, '  other: ', kw)
```

```
person('Xiang', 37)     # name:  Xiang      age:  37      other:  {}
person('Bob', 30, city = 'Beijing', job = 'CS Engineer', gender = 'M')
# name:  Bob      age:  30      other:  {'gender': 'M', 'job': 'CS Engine
er', 'city': 'Beijing'}
```

下面好繁琐

```
extra_info = {'gender': 'F', 'city': 'Chaihe'}
person('xiaoqiang', 22, city = extra_info['city'], gender = extra_info['g
ender'])
# name:  xiaoqiang      age:  22      other:  {'gender': 'F', 'city': 'Ch
aihe'}
```

简化操作

```
person('xiaoqiang', 22, **extra_info)
```

命名关键字参数

CAUSION: 我本节不是很清楚，具体参考***http://www.liaoxuefeng.com/***的***python3***教程部分

> `person(name, age, *, city, job):`
> `person(name, age, *args, city, job):`
> `person(name, age, city, job): #位置参数`

与关键字参数`**kw`不同，命名关键字参数需要一个特殊分隔符`*`，`*`后面的参数被视为命名关键字参数。

```
def person(name, age, *, city, job):
    print(name, age, city, job)

# 已经有*args（可变参数）了，后面再定义命名关键字参数时，就不再需要一个*了
def person(name, age, *args, city, job):
    print(name, age, args, city, job)

def person(name, age, city, job):
    # 缺少 *，city和job被视为位置参数
    pass
```

递归函数

过深的调用会导致栈溢出, 试一试`fact(5555)`.

```
def fact(n):
    if n==1:
        return 1
    return n * fact(n - 1)

print(fact(5))
```

高级特性

OOP

CAUSION: 继承的方式， 连构造器都继承！
直接看代码：

```
class Student(object):
    def __init__(self, name):
        self.name = name
    def set_age(self, age):
        self.age = age
    def set_major(self, major):
        self.major = major

anna = Student('anna')
anna.set_age(21)
anna.set_major('CS')
print(anna.major)

# 继承！ MasterStudent继承Student类的成员方法，成员变量，还有构造函数！！！

class MasterStudent(Student):
    internship = 'hehehe'

    # 自己有构造函数就调自己的，若无，调父类的
    def __init__(self, name):
        self.name = name + "aa"

james = MasterStudent('james')
print(james.internship)
print(james.name)
james.set_age(18)
print(james.age)
```

```
class C(object):
```

访问限制

CAUSION: `self.__name = name` 这时，外部代码无法用`对象名.__name`访问， 实现了***private***的赶脚。 但是，外部仍可用`对象名._Student__name`的方式访问，既，***python***无法实现***private.***

e.g.:

```
class Student():
    def __init__(self,name,score):
        self.__name = name
        self.__score = score

    def print_score(self):
        print('%s : %s' %(self.__name, self.__score))

s = Student("Peter", 98)
s.print_score()
# s.__name          #AttributeError
print(s._Student__name)     # right, no any private
```

继承和多态

获取对象信息

CAUSION: `type()`, `isinstance()` ***and*** `dir()`

- type() 基本数据类型获取

```
type(123)    # <class 'int'>
type('abc')==type(123)   # True
```

- isinstance() 表继承关系，也可基本类型

```
class Animal():
    pass

class Dog(Animal):
    pass

class Husky(Dog):
    pass

a = Animal()
d = Dog()
h = Husky()

j = isinstance(h, Husky)    # True
print(j)
j = isinstance(h, Animal)   # True
print(j)
j = isinstance(a, Dog)  # False
print(j)
```

isinstance() 判断基本类型

```
# 判断是否是list或者tuple
j = isinstance([1, 2, 3], (list, tuple))
print(j)
```

- dir()
  如果要获得一个对象的所有属性和方法，可以使用`dir()`函数，它返回一个包含字符串的list.

```
dir('abc')
```

返回

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc
__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem_
_', '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le
__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__r
educe__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr
__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefol
d', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'forma
t', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigi
t', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'is
title', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'part
ition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'r
strip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'titl
e', 'translate', 'upper', 'zfill']
```

- `__xxx__`的属性和方法在Python中都是有特殊用途
  如`__len__`

```
>>> len('ABC')
3
>>> 'ABC'.__len__()
3
```

自己的类，也想`len(自己类)`的话，就自己写一个`__len__()`方法：

```
>>> class MyDog(object):
...     def __len__(self):
...         return 100
...
>>> dog = MyDog()
>>> len(dog)
100
```

还有一些普通的属性和方法：
`lower()`、`getattr()`、`setattr()`以及`hasattr()`。

实例属性和类属性

CAUSION: Python是动态语言，根据类创建的实例可以任意绑定属性

- 属性属于对象， 不属于类，如下：

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

s = Student('Bob')
s.score = 90
# Student.name  无此属性
```

- 类属性

```
class Student(object):
    name = 'Student'

>>> s = Student() # 创建实例s
>>> print(s.name) # 打印name属性，因为实例并没有name属性，所以会继续查找class的name属性
Student
>>> print(Student.name) # 打印类的name属性
Student
>>> s.name = 'Michael' # 给实例绑定name属性
>>> print(s.name) # 由于实例属性优先级比类属性高，因此，它会屏蔽掉类的name属性
Michael
>>> print(Student.name) # 但是类属性并未消失，用Student.name仍然可以访问
Student
>>> del s.name # 如果删除实例的name属性
>>> print(s.name) # 再次调用s.name，由于实例的name属性没有找到，类的name属性就显示出来了
Student
```

CAUSION: 看上面的 `del s.name`
对象先访问自己的属性， 删掉后再访问类的同名属性， 若类无之， 则报错

使用slots

如果我们想要限制实例的属性怎么办？比如，只允许对Student实例添加name和age属性。

```
class Student(object):
    __slots__ = ('name', 'age') # 用tuple定义允许绑定的属性名称
```

再试一试(绑定`score`时，报异常)：

```
>>> s = Student() # 创建新的实例
>>> s.name = 'Michael' # 绑定属性'name'
>>> s.age = 25 # 绑定属性'age'
>>> s.score = 99 # 绑定属性'score'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Student' object has no attribute 'score'
```

CAUSION: slots定义的属性仅对当前类实例起作用，对继承的子类是不起作用的
e.g.:

```
>>> class GraduateStudent(Student):
...     pass
...
>>> g = GraduateStudent()
>>> g.score = 9999
```

除非在子类中也定义`__slots__`，这样，子类实例允许定义的属性就是自身的`__slots__`加上父类的`__slots__`。

使用@property

使用枚举类

每个常量都是class的一个唯一实例。Python提供了`Enum`类来实现这个功能：

```
from enum import Enum
# 获得Month类型的枚举类
Month = Enum('Month', ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Ju
l', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'))
```

```
for name, member in Month.__members__.items():
    print(name, '=>', member, ',', member.value)
```

结果：

```
Jan => Month.Jan , 1
Feb => Month.Feb , 2
Mar => Month.Mar , 3
Apr => Month.Apr , 4
May => Month.May , 5
Jun => Month.Jun , 6
Jul => Month.Jul , 7
Aug => Month.Aug , 8
Sep => Month.Sep , 9
Oct => Month.Oct , 10
Nov => Month.Nov , 11
Dec => Month.Dec , 12
```

若想精确控制枚举类型，可从`Enum`中派生：

```
from enum import Enum, unique

@unique     # @unique装饰类用来保证枚举值唯一
class Weekday(Enum):
    Sun = 0 # Sun的value被设定为0
    Mon = 1
    Tue = 2
    Wed = 3
    Thu = 4
    Fri = 5
    Sat = 6
```

调用枚举：

```
Weekday.Mon   # 返回 Weekday.Mon
Weekday.Mon.value   # 返回 1
Weekday(1)    # 返回 Weekday.Mon
print(Weekday['Tue'])   #返回 Weekday.Tue
for name, member in Weekday.__members__.items():
  print(name,'=>',member)
```

virtualev

> 整理自
> http://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000/001432712108300322c61f256c74803b43bfd65c6f8d0d0000

在开发Python应用程序的时候，系统安装的Python3只有一个版本：3.4。所有第三方的包都会被`pip`安装到Python3的`site-packages`目录下。
virtualev可以让每一个project拥有各自独立**Python**运行环境的隔离器。
假定我们要开发一个新的项目，需要一套独立的Python运行环境，可以这么做：
Xiang已在ubuntu14.04环境下安装

```
sudo pip install virtualenv
```

step1, 创建目录:

```
mkdir myproject
cd myproject
```

step2, 创建一个独立的Python运行环境,命名为`venv`：

```
 virtualenv --no-site-packages venv
```

参数`--no-site-packages`使得安装在系统Python环境中的所有第三方包都不会复制过来，这样，我们就在`venv`目录下得到了一个不带任何第三方包的“干净”的Python运行环境。
step3, 用`source`进入该环境：

```
source venv/bin/activate
```

在`venv`环境下，用`pip`安装的包都被安装到`venv`这个环境下，系统Python环境不受任何影响。也就是说，`venv`环境是专门针对myproject这个应用创建的。

final step, 退出当前的`venv`环境，使用`deactivate`命令：

```
deactivate
```