**整理Python（定制类&特殊方法）**

**1.\_\_str\_\_和\_\_repr\_\_**

如果把一个类的实例变成str，就需要实现特殊方法\_\_str\_\_( ):

class Person(object):

def \_\_init\_\_(self,name,gender):

self.name=name

self.gender=gender

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

return '(Person: %s ,%s)'%(self.name,self.gender)

\_\_str\_\_:用于显示给用户   \_\_repr\_\_:用于显示给开发人员

\_\_repr\_\_=\_\_str\_\_

**2.\_\_slots\_\_**

限制实例的属性

class Student(object):

\_\_slots\_\_=('name','age)

#用tuple定义 允许绑定的属性名称

\_\_slots\_\_定义的属性仅对当前实例起作用，对继承的子类不起作用

**3.@property**

Python内置的@property 装饰器 负责把一个方法变成属性调用

class Student(object):

@property *#把一个getter方法变成属性，只需要加上@property*

def score(self): *#此时，又创建了另外一个装饰器 @score.setter*

return self.\_score

@score.setter *#@score.setter 负责把一个setter方法变成属性赋值*

def score(self,value):

if not isinstance(value,int):

raise ValueError('score must be an integer')

if value <0 or value >100:

raise ValueError('score must between 0~100')

self.\_score=value

**4.\_\_len\_\_**

如果一个类表现的像一个list，要获取有多少个元素，就得用len（）函数

要让len（）函数正常运行，类必须提供一个特殊方法\_\_len\_\_( )，它返回元素的个数

class Studernt(object):

def \_\_init\_\_(self,\*args):

self.names=args

deg \_\_len\_\_(self):

return len(self.names)

**5.\_\_iter\_\_   和  \_\_next\_\_**

如果一个类想被用于for...in 循环，类似list 和tuple 那样，就要用\_\_iter\_\_( ) 方法，它返回一个迭代对象

然后，python的for循环就会不断调用该迭代对象的\_\_next\_\_( )方法拿到循环的下一个值，直到遇到StopIteration 错误时退出循环

*#斐波那契数列为例*

class Fib(object):

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

self.a,self.b=0,1 *#初始化两个计数器 a，b*

def \_\_iter\_\_(self):

return self *#实例本身就是迭代对象，故返回自己*

def \_\_next\_\_(self):

self.a,self.b = self.b,self.a+self.b *#计算下一个值*

if self.a>100000: *#退出循环的条件*

raise StopTteration()

return self.a *#返回下一个值*

**6. \_\_getitem\_\_**

要表现的像list那样按照下标取元素，需要\_\_getitem\_\_（）方法：

class Fib(object):

def \_\_getitem\_\_(self,n):

a,b=0,1

for x in range(n):

a,b=b,a+b

return

**7.\_\_getatter\_\_**

在自定义类中，如果调用一个属性不存在时，可以调用\_\_getatter\_\_( ) ，动态返回一个属性

class Student(object):

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

self.name='li'

def \_\_getatter\_\_(self,atter)

if atter='score'

return 99

比如不存在score时，\_\_getatter\_\_(self,'score')来尝试获得属性，返回定义的值

**8.\_\_call\_\_  和 callable( )**

一个对象实例可以有自己的属性和方法，在调用实例方法时，用instance.method（）调用

也可以直接作用在实例本身上    （***对象中调用实例方法***，类比，***类中调用实例***）

任何一个类，只需要定义\_\_call\_\_( ) 方法，就可以直接对实例进行调用

class Student(object):

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

self.name=name

def \_\_call\_\_(self):

print('My name is %s.'%self.name)

怎么判断一个变量是对象还是函数

需要判断一个对象是否被调用，能被调用的对象就是一个**callable**对象

callable(Student()) *#返回 True*

callable(max) *#返回 True*

callable([1,2,3,]) *#返回False*

python中 \_\_str\_\_和\_\_repr\_\_

\_\_str\_\_()用于显示给用户，而\_\_repr\_\_()用于显示给开发人员。

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

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

return '(Person: %s, %s)' % (self.name, self.gender)

\_\_repr\_\_ = \_\_str\_\_ #直接让repr和str相同

定义了repr函数后，在调试时直接向命令行输入p即可输出person的信息，否则会输出

<\_\_main\_\_.Person object at 0x0000000002E66C88>

\_\_str\_\_和\_\_repr\_\_函数会被子类继承。

### python中 \_\_cmp\_\_

为了对对象进行排序，可以使用函数sorted函数，前提是设置了类的\_\_cmp\_\_方法。

class Student(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.score = score

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

return '(%s: %s)' % (self.name, self.score)

\_\_repr\_\_ = \_\_str\_\_

def \_\_cmp\_\_(self, s):

if self.name < s.name:

return -1

elif self.name > s.name:

return 1

else:

return 0

>>> L = [Student('Tim', 99), Student('Bob', 88), Student('Alice', 77)]

>>> print sorted(L)

[(Alice: 77), (Bob: 88), (Tim: 99)]

1

### python中 \_\_len\_\_

调用len()返回实例的长度.

class Students(object):

def \_\_init\_\_(self, \*args):

self.names = args

def \_\_len\_\_(self):

return len(self.names)

>>> ss = Students('Bob', 'Alice', 'Tim')

>>> print len(ss)

#### 任务：

斐波那契数列是由 0, 1, 1, 2, 3, 5, 8...构成。

请编写一个Fib类，Fib(10)表示数列的前10个元素，print Fib(10) 可以打印出数列的前 10 个元素，len(Fib(10))可以正确返回数列的个数10。

代码1：（参考代码）

可以使用str()将整型列表转化为字符串。

class Fib(object):

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

a, b, L = 0, 1, []

for n in range(num):

L.append(a)

a, b = b, a + b

self.numbers = L

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

return str(self.numbers)

\_\_repr\_\_ = \_\_str\_\_

def \_\_len\_\_(self):

return len(self.numbers)

f = Fib(10)

print(f )

print (len(f))

代码2：（自己写的）

class Fib(object):

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

self.num = num

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

if(self.num==1):

return "[0]"

elif(self.num==2):

return "[0,1]"

else:

fib\_str = "[0,"

fib=[0,1]

for k in range(2,self.num+1):

fib\_str = fib\_str+ str(fib[k-1]) + ","

fib.append(fib[k-1]+fib[k-2])

return fib\_str[:-1]+']'

def \_\_len\_\_(self):

return self.num

f = Fib(10)

print(f)

print(len(f))

### python中数学运算（运算符重载）

例：有理数可以用p/q表示，其中p和q都是整数。

调用求最大公约数的gcd函数（递归）：

欧几里得定理：若a=b\*r+q，则 GCD（a,b）=GCD（b,q）

def gcd(a,b):

if(b==0):

return a

return gcd(b,a%b)

class Rational(object):

def \_\_init\_\_(self, p, q):

self.p = p

self.q = q

def \_\_add\_\_(self, r):

return Rational(self.p \* r.q + self.q \* r.p, self.q \* r.q)

def \_\_sub\_\_(self, r):

return Rational(self.p\*r.q-r.p\*self.q,self.q\*r.q)

def \_\_mul\_\_(self, r):

return Rational(self.p\*r.p,self.q\*r.q)

def \_\_div\_\_(self, r):

return Rational(self.p\*r.q,self.q\*r.p)

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

g = gcd(self.p,self.q)

return '%s/%s' % (self.p/g, self.q/g)

\_\_repr\_\_ = \_\_str\_\_

r1 = Rational(1, 2)

r2 = Rational(1, 4)

print (r1 + r2)

print( r1 - r2)

print (r1 \* r2)

print( r1 / r2)

### python中类型转换

\_\_int\_\_ 和 \_\_float\_\_方法实现类型转换，

\_\_str\_\_ 方法也可以看做是一种类型转换。

class Rational(object):

def \_\_init\_\_(self, p, q):

self.p = p

self.q = q

def \_\_int\_\_(self):

return self.p // self.q

def \_\_float\_\_(self):

return float(self.p)/float(self.q)

print(float(Rational(7, 2)))

print(float(Rational(1, 3)))

### python中 @property（装饰器函改写set和get方法）

@property 修饰方法后即是get方法。

“@+方法名字+点+setter”是个固定格式与@property搭配使用，set方法。

如果没有定义set方法，就不能对“属性”赋值，这时，就可以创建一个只读“属性”。

class Student(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.\_\_score = score

@property

def score(self):

return self.\_\_score

@score.setter

def score(self, score):

if score < 0 or score > 100:

raise ValueError('invalid score')

self.\_\_score = score

### python中 \_\_slots\_\_ 属性限制

\_\_slots\_\_规定了一个类允许的属性列表。不允许添加列表外的属性。

class Student(object):

\_\_slots\_\_ = ('name', 'gender', 'score')

def \_\_init\_\_(self, name, gender, score):

self.name = name

self.gender = gender

self.score = score

1 强行添加grade属性会报错。

>>> s = Student('Bob', 'male', 59)

>>> s.name = 'Tim' # OK

>>> s.score = 99 # OK

>>> s.grade = 'A'

Traceback (most recent call last):

...

AttributeError: 'Student' object has no attribute 'grade'

### python中\_\_call\_\_ 把类实例变成可调用对象

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

def \_\_call\_\_(self, friend):

print( 'My name is %s...' % self.name)

print( 'My friend is %s...' % friend)

>>> p = Person('Bob', 'male')

>>> p('Tim')

My name is Bob...

My friend is Tim...