Python交互模式

在命令行模式下敲命令python进入到Python交互模式，提示符是>>>。

在Python交互模式下输入exit()并回车，退出Python交互模式，回到命令行模式。

也可以直接通过开始菜单选择Python (command line)菜单项，直接进入Python交互模式，输入exit()后窗口会直接关闭，不会回到命令行模式

>>> print('hello, world')

>>>print('feafea', 'nihao', 'shijie')

输出feafea nihao shijie

>>> s

'Hello'

>>> x = len(s)

>>> print("The length of %s is %d" % (s,x))

The length of Hello is 5

* print格式化输出

(1). %

(2). 转换标志：

* -表示左对齐
* +表示在转换值之前要加上正负号
* “”（空白字符）表示正数之前保留空格
* 0表示转换值若位数不够则用0填充

(3). 最小字段宽度：字符串至少应该具有的宽度。如果是\*，则宽度会从值元组中读出。

(4). 点(.)后跟精度值：如果是实数，就表示出现在小数点后的位数。如果转换的是字符串，那么该数字就表示最大字段宽度。如果是\*，那么精度将从元组中读出

(5). 字符串格式化转换类型

d,i 带符号的十进制整数

u 不带符号的十进制

e 科学计数法表示的浮点数（小写）

E 科学计数法表示的浮点数（大写）

f,F 十进制浮点数

C 单字符（接受整数或者单字符字符串）

r 字符串（使用repr转换任意python对象)

s 字符串（使用str转换任意python对象）

pi=3.14159265

print('I am %5.3f' % pi) #打印I am 3.142

print('I am %10.3f' % pi) #打印I am 3.142

print('I am %010.3f' % pi) #打印I am 000003.142

print('I am %+010.3f' % pi) #打印I am +00003.142

>>> print("pi = %.\*f" % (3,pi)) #用\*从后面的元组中读取字段宽度或精度

pi = 3.142

不输出换行

print(pi, end='')

>>> name = input()

之后命令行等待输入一个字符串

name=int(name)

转化为int

float('23.33')

转化为float

hello.py

print(100+200+3000)

命令行模式下进入.py目录，执行hello,py文件E:\tempDirectory>python hello.py

3300

当语句以冒号:结尾时，缩进的语句视为代码块。

按照约定俗成的管理，应该始终坚持使用4个空格的缩进。

十六进制数字加0x前缀。0xa5b4c3d2

字符串双引号里有单引号，单引号不解析不转义，就是引号

"I'm OK'"

就是I'm OK'

转义字符\' \"

r' '表示' '内部的字符串不转义

>>> print(r'\fa\fa\sv\vd\\\feaw\\f\\\\b\t\r\n') （r不能写在里边）

\fa\fa\sv\vd\\\feaw\\f\\\\b\t\r\n

布尔值True False 开头大写

>>> True and False

False

>>> 3>2

True

>>> not True

False

空值None

同一个变量可以反复赋值，而且可以是不同类型的变量

a = 123 # a是整数

a = 'ABC' # a变为字符串

#注释

a = 'ABC'

b = a

a = 'XYZ'

print(b)

ABC

解释器创建字符串ABC和变量a，将a指向ABC——>b指向a——>创建字符串XYZ，将a指向XYZ

——>的意思是下一步……

10//3结果是3。//运算保留整数

10%3=1

ord()函数获取字符的整数表示，chr()函数把编码转换为对应的字符：

>>> ord('A')

65

>>> ord('中')

20013

>>> chr(66)

'B'

>>> chr(25991)

'文'

如果知道字符的整数编码，还可以用十六进制这么写str：

>>> '\u4e2d\u6587'

'中文'

if x: #x非空并且非False就是True

<执行1>

elif <条件判断2>: #注意冒号

<执行2>

elif <条件判断3>:

<执行3>

else:

<执行4>

n=0

sum=0

while n<=100:

sum+=n

n=n+1

for n in list(range(5)): #[0, 1, 2, 3, 4] 0到4

sum=n+sum

for x in range(1, 4): # range(1, 4) x=1, 2, 3

... L.append(x \* x)

break跳出当前循环

continue跳过一次

**字典**

d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}

d['Michael']

d['Adam'] = 67 增加键值对

* 判断key是否存在。两种办法，一是通过in

>>> 'Thomas' in d

False

二是get()方法，不存在返回None，或者自己指定的value；存在返回value

>>> d.get('Thomas')

>>> d.get('Thomas', -1)

-1

删除

d.pop('Michael')

**集合**

不重复，无序

s = set([1, 2, 3, 3]) #自动过滤重复元素

s.add(42)

abs(-20)

max(3,4,60,2,1,32)

**函数**

def fun1(x, y=-4):

if x >= 0:

return x, y

else:

return -x, -y

python可以有多个返回值，实际上返回了一个tuple元组

a, b=fun1(-30, -8)

多个变量接受一个tuple，按位置赋给对应的值

fun1(x, y, z=10, c=39, e=10)

调用的时候可以fun1(23, 1, c=56)

c的值设为56， 其他使用默认值

可变参数

def calc(\*numbers):

sum = 0

for n in numbers:

sum = sum + n \* n

return sum

cal(3,4,5,6,7)

如果参数是一个num=[1, 2, 3]

1.cal(num[0], num[1], num[2])

2.cal(\*num)

关键字参数

def person(name, age, \*\*kw):

print('name:', name, 'age:', age, 'other:', kw)

* 调用方法1

传入任意个数的关键字参数

person('Adam', 45, gender='M', job='Engineer') #这儿属性名没有引号没有大括号

* 调用方法2

>>> extra = {'city': 'Beijing', 'job': 'Engineer'} #这儿属性名有引号

>>> person('Jack', 24, \*\*extra)

kw获得extra的一份拷贝，函数内改变kw不改变extra

命名关键字参数

关键字参数的基础上，限制关键字参数的名字

只接收city和job作为关键字参数

def person(name, age, \*, city, job):

print(name, age, city, job)

\*后面的参数被视为命名关键字参数

>>> person('Jack', 24, city='Beijing', job='Engineer')

* 如果函数定义中已经有了一个可变参数，后面跟着的命名关键字参数就不再需要一个特殊分隔符\*了：

def person(name, age, \*args, city, job):

print(name, age, args, city, job)

* 命名关键字参数可以有缺省值，从而简化调用：

def person(name, age, \*, city='Beijing', job):

print(name, age, city, job)

由于命名关键字参数city具有默认值，调用时，可不传入city参数：

>>> person('Jack', 24, job='Engineer')

参数定义的顺序必须是：必选参数、默认参数、可变参数、命名关键字参数和关键字参数

检查类型

isinstance(x, (int, float))

type('feafa')

type()返回个type类型，这就和JavaScript不一样

* isinstance() 与 type() 区别：

type() 不会认为子类是一种父类类型，不考虑继承关系。

isinstance() 会认为子类是一种父类类型，考虑继承关系。

print(type({'fea': 'feaf'}))

print(type('feafa')==str)

返回

<class 'dict'>

True

>>> import types

>>> def fn():

... pass

>>> type(fn)==types.FunctionType #判断是不是函数

True

>>> type(abs)==types.BuiltinFunctionType

True

>>> type(lambda x: x)==types.LambdaType

True

>>> type((x for x in range(10)))==types.GeneratorType #判断是不是Generator

True

print(type(type('fefefe'))) #返回<class 'type'>

print(type('fefa')) #返回<class 'str'>

raise TypeError('bad operand type')

自定义抛出错误

**切片Slice**

取一个子数组

>>> L[0:3]

['Michael', 'Sarah', 'Tracy']

L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3。即索引0，1，2

如果第一个索引是0，或者最后一个索引是-1，则可以省略：

>>> L[:3] #和L[0:3]效果一样

>>> L[1:3] #取L[1], L[2]

>>> L[-2:] #取L[-2], L[-1]

>>> L[-2:-1] #取L[-2]

倒数第一个元素的索引是-1

前10个数，每两个取一个：

>>> L[:10:2]

[0, 2, 4, 6, 8]

什么都不写，只写[:]就可以原样复制一个list：

>>> L[:]

[0, 1, 2, 3, ..., 99]

* tuple也是一种list，唯一区别是tuple不可变。tuple也可以用切片操作，操作的结果仍是tuple：

>>> (0, 1, 2, 3, 4, 5)[:3]

(0, 1, 2)

* 字符串也可以用切片操作，操作结果仍是字符串：

>>> 'ABCDEFG'[:3]

'ABC'

Python没有针对字符串的截取函数，只需要切片操作

**迭代**

* 遍历字典
* for key in dic:
* for value in d.values():
* for k, v in d.items():
* 遍历字符串
* for ch in 'abcdefg':
* for I, ch in enumrate('abcdefg'):
* 遍历数组list
* for value in [123, 34, 56]:
* for i, value in enumerate(['A', 'B', 'C']):

enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。

判断一个对象是否可遍历

>>>from collections import Iterable

>>> isinstance('abc', Iterable) # str可迭代

>>> isinstance(123, Iterable) #返回False

**列表生成式** List Comprehensions

用于生成一个list

>>> [x \* x for x in range(1, 11)]

返回[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

for循环后面还可以加上if判断

>>> [x \* x for x in range(1, 11) if x % 2 == 0]

[4, 16, 36, 64, 100]

还可以使用两层循环，可以生成全排列：

>>> [m + n for m in 'ABC' for n in 'XYZ']

['AX', 'AY', 'AZ', 'BX', 'BY', 'BZ', 'CX', 'CY', 'CZ']

**生成器generator**

* 创建

g=(x\*x for x in range(10))

print(g)

返回<generator object <genexpr> at 0x0078CF70>

就是列表生成式的中括号变成小括号

创建以后这些值没有被计算出来也没有存储

* 访问
* next(g)返回0

再next(g)返回1

再next(g)返回4

generator保存的是算法，每次调用next(g)，就计算出g的下一个元素的值，直到计算到最后一个元素，没有更多的元素时，抛出StopIteration的错误

* for n in g:

print(n)

* 通过函数创建的generator

def fib(max):

n, a, b = 0, 0, 1

while n < max:

yield b

a, b = b, a + b

n = n + 1

return 'done'

函数中包含yield关键字，则此函数不再是一个普通函数，而是一个generator

generator和函数的执行流程不一样。函数是顺序执行，遇到return语句或者最后一行函数语句就返回。而generator在每次调用next()的时候执行，遇到yield语句返回，再次执行时从上次返回的yield语句处继续执行

用for循环得到generator每个yield的值；Python的for循环本质上就是通过不断调用next()函数实现的

>>> for n in fib(6):

print(n)

* generator函数的“调用”实际返回一个generator对象

>>> g = fib(6)

>>> g

<generator object fib at 0x1022ef948>

* 用for循环调用generator时，拿不到generator的return语句的返回值。如果想要拿到返回值，必须用next()，然后捕获StopIteration错误，返回值包含在StopIteration的value中

赋值语句

a, b = b, a + b

相当于：

t = (b, a + b) # t是一个tuple

a = t[0]

b = t[1]

**迭代器**

* 可以直接作用于for循环的对象统称为可迭代对象：Iterable

可以使用isinstance()判断一个对象是否是Iterable对象：

>>> from collections import Iterable

>>> isinstance([], Iterable)

True

比如list、tuple、dict、set、string、generator

* 可以被next()函数调用并不断返回下一个值的对象称为迭代器：Iterator

可以使用isinstance()判断一个对象是否是Iterator对象：

>>> from collections import Iterator

>>> isinstance((x for x in range(10)), Iterator)

True

比如generator

* 把list、dict、str等Iterable变成Iterator可以使用iter()函数：

>>> isinstance(iter([]), Iterator)

True

>>> isinstance(iter('abc'), Iterator)

True

**高阶函数**Higher-order function

一个函数就接收另一个函数作为参数，这种函数就称之为高阶函数。

def add(x, y, f):

return f(x) + f(y)

print(add(-5, 6, abs))

cmd输入pip install you-get

然后you-get <http://www.bilibili.com/video/av33417359/>

av33417359就被下载到当前目录里

**类**

* 声明类

class Student(object):

pass

类名通常是大写开头的单词，后边是(object)，表示该类是从哪个类继承下来的，通常，如果没有合适的继承类，就使用object类，这是所有类最终都会继承的类

pass占位符，没有pass报错，但是又不需要写什么东西

* 创建实例

bart = Student()

* 绑定属性

bart.name = 'Bart Simpson'

声明类时绑定属性

class Student(object):

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

self.name = name

self.score = score

相当于构造函数应该

\_\_init\_\_一边有两个下划线

self指向实例本身，永远是第一个参数，创建实例时不用填

bart1=Student('Bjergsen', 44)

* 定义方法

class Student1(object):

def \_\_init\_\_(self, name='wo', age=10):

self.name=name;

self.age=age;

def printMsg(self):

print(self.name)

print(self.age)

bart=Student1()

bart.printMsg()

同样，第一个参数传self，调用时不写

* 私有属性

把属性的名称前加上两个下划线\_\_，就变成了私有属性

class Student(object):

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

self.\_\_name = name

self.\_\_score = score

def print\_score(self):

print('%s: %s' % (self.\_\_name, self.\_\_score))

想访问和修改的话写get和set方法

def get\_name(self):

return self.\_\_name

def set\_name(self, name)

self.\_\_name=name

需要注意的是，在Python中，变量名类似\_\_xxx\_\_的，也就是以双下划线开头，并且以双下划线结尾的，是特殊变量，特殊变量可以直接访问，不是private变量，所以，不能用\_\_name\_\_、\_\_score\_\_这样的变量名。

属性名\_name，这样的实例变量外部是可以访问的，这样的变量意思是，“虽然我可以被访问，但是，请把我视为私有变量，不要随意访问”。

* 多态

class Person(object):

def run(self):

print('person is running')

class Student(Person):

def run(self):

print('Student is running')

man1=Person()

man2=Student()

man1.run() #打印person is running

man2.run() #打印Student is running

def whoisrunning(animal):#这个animal只要有一个run方法就行，不需要是Person的子类，区别于Java

animal.run()

class Obj(object):

def run(self):

print('object is running')

whoisrunning(Obj()) #打印object is running

whoisrunning(Student()) #打印Student is running

判断类型

print(isinstance(man1, Person)) #打印True

print(isinstance(man2, Person)) #打印True，isInstance子类也是父类的instance

* 获取对象信息

>>> dir('ABC')

获得一个对象的所有属性名和方法名，返回一个包含字符串的list

返回['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

* 类似\_\_xxx\_\_的属性和方法在Python中都是有特殊用途的，比如\_\_len\_\_方法返回长度

>>> len('ABC')

>>> 'ABC'.\_\_len\_\_()

效果一样

class MyObject(object):

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

self.x = 9

self.\_\_y='niaho'

def power(self):

return self.x \* self.x

obj = MyObject()

hasattr(obj, 'x') #返回True

hasattr(obj, '\_\_y') #返回False

print(dir(obj))

返回['\_MyObject\_\_y', '\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'power', 'x']

灰色都是object原生的方法和属性

* 可以看到，\_\_y被设置为了\_MyObject\_\_y，所以可以在外部用\_MyObject\_\_y访问\_\_y，python没有真正的私有属性。不同版本设置的不一样，不推荐这样访问

setattr(obj, 'z', 19) # 设置一个属性'y'

getattr(obj, 'z') # 获取属性'y'

#可以传入一个default参数，如果属性不存在，就返回默认值：

getattr(obj, 'z', 404) # 获取属性'z'，如果不存在，返回默认值404

f1 = getattr(obj, 'power') # 获取函数'power'并赋值到变量fn

print(f1())

* 类属性

class Student(object):

name = 'Student'

这玩意还不是静态变量，通过实例改变以后，类变量值不变。不对，应该是实例没有权限修改类变量，obj.name='tea'效果是创建了一个新的名为name的实例属性，屏蔽掉了类属性

不过用Student.name='tea'修改以后还是起作用的

* 给实例绑定方法

>>> def set\_age(self, age): # 定义一个函数作为实例方法

... self.age = age

...

>>> from types import MethodType

>>> s.set\_age = MethodType(set\_age, s) # 给实例绑定一个方法

>>> s.set\_age(25) # 调用实例方法

>>> s.age # 测试结果

25

给一个实例绑定的方法，对另一个实例是不起作用的

* 给类（所有实例）绑定方法

为了给所有实例都绑定方法，可以给class绑定方法：

>>> def set\_score(self, score):

... self.score = score

...

>>> Student.set\_score = set\_score

给class绑定方法后，所有实例均可调用：

通常情况下，上面的set\_score方法可以直接定义在class中，但动态绑定允许我们在程序运行的过程中动态给class加上功能，这在静态语言中很难实现。

* 限制属性绑定

class Student(object):

\_\_slots\_\_ = ('name', 'age') # 用tuple定义允许绑定的属性名称

>>> s = Student()

>>> s.age = 25 # 绑定属性'age'OK

>>> s.score = 99 # 绑定属性'score'报错

Traceback (most recent call last):

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

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

\_\_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；子类仍然可以随便定义

>>> 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\_\_。

* 多重继承

class Bat(Mammal, Flyable):

pass

* 定制类

>>> class Student(object):

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

... self.name = name

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

... return 'Student object (name: %s)' % self.name

* 相当于重写toString(),调用print()的时候调用\_\_str\_\_

>>> s

直接显示变量调用的不是\_\_str\_\_()，而是\_\_repr\_\_()，两者的区别是\_\_str\_\_()返回用户看到的字符串，而\_\_repr\_\_()返回程序开发者看到的字符串，也就是说，\_\_repr\_\_()是为调试服务的

解决方法重写\_\_repr\_\_()

或者类声明最后一行

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

\_\_iter\_\_

* 如果一个类想被用于for ... in循环，类似list或tuple那样，就必须实现一个\_\_iter\_\_()方法，该方法返回一个迭代对象，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。

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

if self.a > 100000: # 退出循环的条件

raise StopIteration()

return self.a # 返回下一个值

现在，试试把Fib实例作用于for循环：

\_\_getitem\_\_() 让这个Fib可以f[3]，像这样一样用索引访问元素

* 枚举类

为枚举类型定义一个class类型，然后，每个常量都是class的一个唯一实例

from enum import Enum

Month = Enum('Month', ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'))

这样我们就获得了Month类型的枚举类，可以直接使用Month.Jan来引用一个常量，或者枚举它的所有成员：

for name, member in Month.\_\_members\_\_.items():

print(name, '=>', member, ',', member.value)

打印

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

value属性则是自动赋给成员的int常量，默认从1开始计数。

定义枚举时，成员名称不允许重复

默认情况下，不同的成员值允许相同。但是两个相同值的成员，第二个成员的名称被视作第一个成员的别名

* 从Enum派生出自定义类，更精确地控制枚举类型

from enum import Enum, unique

@unique

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

@unique装饰器保证没有重复值

查不到Month = Enum('Month',……)这样声明时第一个参数什么作用，所以用第二种方法声明吧

* 访问这些枚举类型可以有若干种方法：

>>> print(Weekday.Mon)

Weekday.Mon

>>> print(Weekday['Tue'])

Weekday.Tue

>>> print(Weekday.Tue.value)

2

>>> print(day1 == Weekday.Mon)

True

>>> print(Weekday(1))

Weekday.Mon

>>> for name, member in Weekday.\_\_members\_\_.items():

... print(name, '=>', member)

既可以用成员名称引用枚举常量，又可以直接根据value的值获得枚举常量。