PYTHON
常见 & 易错知识点

BY

杨向南

yxnan@pm.me

浙江大学

Java&Python 春学期朋辈辅学群　首发

2020 年 4 月 24 日

目录（除第 0 章外，其余章节均按浙大 Python 课本划分）

0 前言：如何寻求帮助 **1**

0.1 使用 dir 函数查看可用的信息 1

0.2 查看“文档字符串” 2

1 Python 语言概述 **3**

1.1 数制及编码 3

1.2 输入输出函数 3

2 用 Python 编写程序 **4**

2.1 数字类型 4

2.2 赋值语句 5

2.3 格式化输出 5

3 使用字符串、列表和元组 **7**

3.1 序列 7

3.2 字符串 8

3.3 列表与元组 8

4 条件、循环 **9**

5 集合与字典 **10**

5.1 集合 10

5.2 字典 12

6 函数 **13**

6.1 值传递与引用传递 13

6.2 参数绑定 14

6.3 常用的内置函数 16

6.4 模块 18

0 前言：如何寻求帮助

0.1 使用 dir 函数查看可用的信息

dir 函数是 Python 的一个内建函数，它可用于列出任意一个对象的所有成员。这样做有什么用呢？也许有时候，你花很多代码去实现的功能，说不定是 Python 自带的呢！

例如,我们想知道可以对字符串进行哪些操作,可以在 IDLE 中输入 `dir(str)`或任意一个字符串实例，`dir("py")`

```
>>> dir("py")
['__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']
```

先不管前面带下划线的晦涩内容，我们可以在后面看到一些熟悉的函数，比如 split、strip、join 等等，其实这些就是 Python 为字符串提供的所有功能，其中很多都可以通过名字猜出大致作用，方便进一步查找。

dir 函数也可以用于查看模块。比如，在导入 math 模块后，通过`dir(math)`就能知道数学模块中所有可用的函数。`print(__builtins__)`则可以查看所有的内置函数。

0.2 查看“文档字符串”

文档字符串 (*docstring*)，是指放在函数定义最开头的一个字符串，往往用于说明这个函数的**功能**、**参数**、**返回值**等信息。Python 自带的库函数几乎都有文档字符串，直接查看这些函数的 docstring 就能获得很多信息，而不必再到网上查找。对任意一个函数，我们可以通过打印`__doc__`来获得它的 docstring. 这是文档字符串的定义例子：

```
def func(args):
    """This is docstring.
    It can cross multi-lines"""
    # 函数体
```

如上定义的一个函数，我们可以通过`func.__doc__`来获取它的文档字符串。

在之前，我们通过 dir 获取了字符串的成员函数，我们可以通过打印出其中感兴趣的函数的文档字符串来获取简单的帮助：

```
>>> print("py".split.__doc__)
Return a list of the words in the string, using sep
as the delimiter string.

  sep
    The delimiter according which to split the string.
    None (the default value) means split according to any
    whitespace and discard empty strings from the result.
  maxsplit
    Maximum number of splits to do.
    -1 (the default value) means no limit.
```

读完这几行信息，我们就可以得知 split 函数的功能，还有它的两个参数 sep 和 maxsplit。类似地，模块也有文档字符串，查看方法和函数相同，此处不再赘述。

1 Python 语言概述

1.1 数制及编码

使用 bin、hex 函数将整数转化为二进制、十六进制字符串：

```
>>> bin(214)
'0b11010110'
>>> hex(214)
'0×d6'
```

掌握 ASCII 码的转换以及输出：

```
>>> ord('y')
121
>>> chr(78)
'N'
>>> '{:c}'.format(78)
N
```

1.2 输入输出函数

使用 input 函数进行输入，读到回车时结束，返回的字符串不包含换行符。得到的字符串可以用 split 等方法进行初步处理：

```
# 列表解构
>>> idNum,name,score = input().split()
319000 LiHua 90
>>> idNum,name,score
('319000', 'LiHua', '90')

# 对于不定长的输入，可以限制分割的最大次数
>>> idNum,name,scoreList = input().split(maxsplit=2)
319000 LiHua 90 89 94 100 87
>>> idNum,name,scoreList
('319000', 'LiHua', '90 89 94 100 87')
```

当然，输入只是第一步，重要的是掌握字符串、列表等数据结构的应用，便于游刃有余地处理输入数据。

输出使用 print 函数：

```
# print 有 4 个关键字参数：sep,end,file,flush
# 打印 print 的文档字符串，并取第一行
>>> print(print.__doc__.split(sep='\n')[0])
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

# 以 + 作为间隔，不换行
>>> print(1,2,3,4, sep='+', end='')
1+2+3+4
```

2 用 Python 编写程序

2.1 数字类型

Python 有 4 种内置数字类型：int,float,bool,complex。浮点类型 float 在计算机内部是以二进制形式储存的，因为十进制小数不一定都能转换为有限位数的二进制小数，所以可能存在误差，但在一般情况下不会影响结果。如果要求百分百的精确，可以使用 decimal 模块处理浮点数。

布尔值 bool 在算术运算中会被视作**整数**看待，其中 True 为 1，False 为 0。利用这一点，我们可以用来计算表达式中正确的个数：

```
>>> sum([1==2, 'white'.islower(), 'himself'.endswith('self')])
2
```

math 模块中有很多实用的数学函数，例如处理浮点舍入的 floor,ceil,round，数学常数 e,pi，常用函数 sqrt,log10,exp 等等。例如，计算 a,b 两数的最大公因数 gcd 和最小公倍数 lcm：

```
# math.gcd
>>> gcd = math.gcd(78,21)
>>> gcd
3
# 根据数论相关知识，有 a*b = gcd*lcm
```

```
>>> lcm = 78*21/gcd
>>> lcm
546.0
```

2.2 赋值语句

除了简单的“变量 = 值”这种赋值语句外，前面我们给出的例子中使用了形如 `a,b,c = ...` 的语句，这其实是**序列的封装与解构** (*packing and unpacking*)。所谓序列，就是指诸如字符串、列表、元组这样由“一连串有顺序的值”组成的结构。当等号两边都是序列时，右边的对象被解构（拆分），各自赋给左边对应的变量。`a,b,c = ...` 其实是一种省略的写法，完整的形式是`(a,b,c) = ...`

```
# idNum=319000, name='LiHua', score=95
>>> [idNum,name,score] = 319000,'LiHua',95

# 星号代表解构，可以收集不定数量的值
>>> idNum,name,*scoreList = [319000,'LiHua',90,95,85,79]
>>> print(idNum,name,scoreList)
319000 LiHua [90, 95, 85, 79]

# 解构的另一个例子
>>> first,second,*mid,last = [1,2,3,4,5,6,7]
>>> print(mid)
[3, 4, 5, 6]

# 优先保证不带星号的位置
>>> a,*b,c = 1,2
>>> print(b)
[]
```

2.3 格式化输出

格式化输出有两种方式，通过字符串的 format 方法，以及% 运算符。一些使用 format 的例子如下：

```
# 索引
>>> '{0} {0[1]} {1[1]}'.format("hello", ['my', 'friend'])
'hello e friend'
# 关键字
>>> '{a} {b}'.format(a="hello", b="world")
'hello world'
# 取得对象属性
>>> '{.imag}'.format(1+2j)
'2.0'
```

关于数字的 format，需要记住这些用法：**^**, **<**, **>** 分别是居中、左对齐、右对齐，后面带宽度，**:**号后面带填充的字符，只能是一个字符，不指定则默认是用空格填充。**+**表示在正数前显示 +；（空格）表示在正数前加空格。

```
>>> '{:02d}:{:02d}:{:02d}'.format(15,2,3)
'15:02:03'
>>> '{:/^4d}'.format(12)
'/12/'
>>> '{:>4d}{:>4d}{:>4d}{:>4d}'.format(1,2,3,4)
'   1   2   3   4'
>>> '{:.1%}'.format(0.12721)
'12.7%'
>>> '{:+.2f}'.format(1.2)
'+1.20'
```

也可以用% 运算符来实现格式化输出，与 format 相比更加直观，但功能不如后者。

```
>>> '%.2f' % 2.213
'2.21'
>>> '%2d:%02d:%02d' % (15,2,3)
'15:02:03'
>>> '%s : %ds' % ('time', 2)
'time : 2s'
```

3 使用字符串、列表和元组

3.1 序列

关于序列对象，重点需要掌握的就是 range 函数，它有 3 种使用方法：

```
range(stop)  ->  [0,1,...,stop-1]
range(start, stop)  ->  [start,...,stop-1]
range(start, stop, step) -> [start,start+step,...,<小于stop 的最大值 >]
```

另一个就是序列的切片运算，和 range 一样也有 3 种用法：

```
>>> a = [1,2,3,4,5]
>>> a[1:4]
[2, 3, 4]   # a[1], a[2], a[3]
>>> print(a[:3], a[3:]) # a[:x] + a[x:] == a
[1, 2, 3] [4, 5]
>>> a[1:-2]
[2, 3]  # 负数代表从尾端数起，-1 是最后一个
# 步进为负
>>> a[::-1]
[5, 4, 3, 2, 1]
>>> a[::-2]
[5, 3, 1]
```

需要注意的是，对于序列，如果你想复制一个对象用于操作，同时保持原对象不变，是不能用`a=my_list`这样的赋值方式的，因为这样 a 和 my_list 其实指的是同一个对象。要取得一份序列的复制，可以使用`my_list[:]`这样的写法。

```
>>> my_list = [1,2,3]
# id 代表一个对象的唯一识别值
>>> id(my_list)
1591860046512
>>> a = my_list
>>> id(a)
1591860046512  # 与 my_list 的 id 相同
>>> my_copy = my_list[:]
```

```
>>> print(my_copy)
[1, 2, 3]
>>> id(my_copy)
1591868623296  # id 不同，是新的对象
```

3.2 字符串

需要注意的是，字符串属于“**不可变**”的序列。也就是说，不能使用`s[0]='a'`这样的语句来赋值。字符串的 strip、replace 等方法其实是返回了一个新的字符串，而不是在原字符串的基础上修改。一些常用的字符串方法如下：

```
# 输出末尾不带空格的一种方法
>>> ' '.join(map(str,[1,2,3]))
'1 2 3'
>>> 'this is harsh'.index('i')
2
>>> 'this is harsh'.find(' is')
4
>>> 'himself'.endswith('self')
True
```

3.3 列表与元组

列表常用的方法有 append, clear, count, extend, index, insert, pop, remove, reverse, sort。可以在 IDLE 中输入**`print([].sort.__doc__)`**来查看 sort 的用法。需要注意的是，index 函数只会返回第一个找到的位置，如果有多个相同的元素，需要手动用循环处理。上述函数中，除了 count，index 之外都会改变原列表。

```
>>> a = [1,0,3,2]
# 默认按从小到大排序，传入 reverse=True 则相反
>>> a.sort(reverse=True)
>>> a
[3, 2, 1, 0]
# 根据 sin 值的大小来排序
>>> a.sort(key=math.sin)
```

```
>>> a
[0, 3, 1, 2]
```

元组可简单地看作不可变的列表，但注意，“不可变” 仅仅对元组的直接元素生效，即，如果元组包含了一个可变对象，则这个对象本身是可变的：

```
>>> a = ([], 'di')
>>> a[0].append(1)
>>> a
([1], 'di')
```

字符串、列表、元组之间可以用 str,list,tuple 三个函数实现互相转换。由于它们都是可迭代对象，因此都具有如下运算：

```
>>> len("face")       # 取得长度
4
>>> [1,2,3] + [4,5]   # 合并
[1, 2, 3, 4, 5]
>>> (1,)*4            # 重复
(1, 1, 1, 1)
>>> 2 in [1,2,3]      # 存在性
True
>>> for x in (1,2,3) # 迭代
```

4 条件、循环

条件、循环相关的语句有 if,while,for 三个。需要知道的是，else 子句不仅可以用在 if 中，也可以用在 while 和 for 语句中，当且仅当循环结束时执行一次。那么它与直接放在循环体之后的语句有什么不同呢？不同点就在于如果循环**非正常结束**，例如使用了 break 语句，那么 else 子句将会被跳过，所以可以在 else 语句块中写一些正常结束循环时才执行的代码。例如，判断用户输入的数是否为素数：

```
N = int(input())
for i in range(2, N):
    if N%i == 0:
        break
else:
    print('{} is prime'.format(N))
```

5 集合与字典

5.1 集合

集合 (*set*) 是一个**无序的不重复元素序列**。可以使用大括号 { } 或者 set() 函数创建集合，注意：创建一个空集合必须用 set()，因为 { } 是用来创建一个空字典。

```
# 创建集合
>>> basket = {'apple','orange','apple','pear','orange'}
>>> print(basket)
{'orange', 'pear', 'apple'}
# 判断元素是否在集合中
>>> 'crabgrass' in basket
False
# 类似列表推导式，集合也可这样用
>>> a = {x for x in 'abracada' if x not in 'abc'}
>>> a
{'r', 'd'}

# 两个集合间的运算
>>> a = set('arcadia')
>>> b = set('ala')
>>> a
{'c', 'r', 'a', 'd', 'i'}
# 差集，即 a 中包含而 b 不包含的元素
>>> a - b
{'i', 'c', 'd', 'r'}
# 并集
>>> a | b
{'c', 'a', 'r', 'd', 'i', 'l'}
# 交集
>>> a & b
{'a'}
# 对称差集，即 (a-b) | (b-a)
```

```
>>> a ^ b
{'c', 'r', 'l', 'd', 'i'}
```

集合常用的内置方法如下，具体使用方法可以查看相应的 docstring：

表 1: 类型列表

方法	描述
A.add(a)	为集合 A 添加元素
A.discard(a)	从 A 中删除元素 a，即使不存在也不会报错
A.clear()	清空集合 A
A.copy()	返回集合 A 的一份拷贝
A.difference(B,C,...)	返回 A - B - C - ...
A.difference_update	同上，但会改变原集合 A
A.symmetric_difference(B)	返回 A,B 的对称差集
A.symmetric_difference_update	同上，但会改变原集合 A
A.intersection(B,C,...)	返回交集
A.union(B,C,...)	返回并集
A.isdisjoint(B)	如 A,B 不相交，则返回 True
A.issubset(B)	A 是否为 B 的子集
A.issuperset(B)	A 是否为 B 的超集
A.update()	用序列给集合添加元素（见下）

```
>>> a = {1, 2, 3}
>>> a.update({4, 5})
>>> a
{1, 2, 3, 4, 5}
>>> a.update([0,6], (7,8))
>>> a
{0, 1, 2, 3, 4, 5, 6, 7, 8}
>>> a.update('01')
>>> a
{0, 1, 2, 3, 4, 5, 6, 7, 8, '1', '0'}
```

5.2 字典

字典是另一种可变容器模型，且可存储任意类型对象。字典的每一项由键和值组成，**键必须是唯一的**，但值则不必。值可以取任何数据类型，但键必须是**不可变的**，如字符串，数字或元组。基于这个原因，作为键的元组里也不能包含可变对象，如列表、集合等。创建字典有 3 种方法：

```
# 以键：值的方式创建字典
>>> dict1 = {'name': 'Bob', 'age': 27}
>>> len(dict1), 'age' in dict1
(3, True)   # in 用于判断某个键是否存在
# 用字典的 fromkeys 方法创建字典
>>> seq = ('name', 'age', 'gender')
>>> dict2 = dict.fromkeys(seq)
>>> dict2
{'name': None, 'age': None, 'gender': None}
>>> dict3 = dict.fromkeys(seq, 10)
>>> dict3
{'name': 10, 'age': 10, 'gender': 10}

# 通过键值对的方式创建字典
# 键值对由两个元素的列表或元组构成
>>> a = [('name','Bob'),('age',27)]
>>> dict(a)
{'name': 'Bob', 'age': 27}
```

访问字典有两种方式，一种是用键直接访问，比如`dict1['name']`，另一种是通过字典的 get 方法，`dict1.get('name')`，不同点在于，如果相应的键不存在，前者会引发错误，后者会返回`None`。

为字典添加值也有两种方式，一种是直接对键赋值`dict1[key] = value`；另一种使用字典的 update 方法,用其他字典来扩充自身:`dict1.update(dict2)`。

删除某个字典元素可以使用`del(dict1[key])`的形式,也可以使用`dict1.pop(key)`，两者在对应的键不存在时都会引发错误。要清空字典，可以使用`dict1.clear()`.

字典的遍历可使用 dict 类的 keys,values,items 方法，一些具体的例子见下：

```python
d = {'name':'Ace','age':17}
for k in d.keys():          # 遍历键
    print(k, end=' ')       # 输出：name age

for v in d.values():        # 遍历值
    print(v, end=' ')       # 输出：Ace 17

for k,v in d.items():       # 同时遍历键值对
    print('{}:{}'.format(k, v))
# 输出
name:Ace
age:17
```

6 函数

6.1 值传递与引用传递

在 python 中，字符串 str，元组 tuple，数字 int,float 等属于**不可变类型** (*immutable*)，而列表 list，字典 dict 等则是**可变类型** (*mutable*)。当作为函数的形参时，如`func(a)`，如果是不可变类型，传递的只是 a 的值，而不是 a 对象本身。比如在函数内部修改 a 的值，只是修改另一个复制的对象，不会影响 a 本身。而对于可变类型，传递的是对象本身，在函数里修改 a 后，外部的 a 也会随之改变。

```python
# 引用传递
def changeThis(mylist):
    '修改传入的列表'
    mylist.append([1,2])
    print('函数内取值: ', mylist)

mylist = [10,20,30]
changeThis(mylist)
print('函数外取值: ', mylist)
# 输出
# 函数内取值:  [10, 20, 30, [1, 2, 3, 4]]
# 函数外取值:  [10, 20, 30, [1, 2, 3, 4]]
```

6.2 参数绑定

定义函数时，可指定两种参数：必需参数和非必需参数。其中，**必需参数**在调用时可以使用 **位置参数**和**关键字参数**两种形式，它们的区别就是传递时前者要根据与定义相同的顺序排列，而后者可以打乱。

注意：位置参数必须在所有关键字参数前面，不然 Python 没法知道你的参数该怎样正确对应。以下是一些函数定义和调用的例子：

```
def f(x, y, z):
    pass

# 合法的调用
f(1, 2, 3)
f(1, z=3, y=2)
f(y=2, x=1, z=3)

# 错误的调用
f(1, z=3, 2)
f(x=1, 2, 3) # 即使第一个位置正确也不行
```

非必需参数必须放在所有必须参数后面，分为默认参数、不定长参数两种。其中，如果默认参数没有传值，则使用指定的值：

```
def g(x, y=100):
    pass

# 正确的调用
g(1)    # x=1, y=100
g(1, 2) # x=1, y=2
g(y=2, x=1)
g(x=1)
```

不定长参数用于接受任意个参数，可以以元组或字典两种方式导入：

```
def h(x, *y):
    pass
```

```
# 正确的调用
h(1)      # x=1, y=()
h(1,2)    # x=1, y=(2,)
h(1,2,3)  # x=1, y=(2,3)

# 注意：不能直接用关键字形式给 y 传值
t = (2,3)
h(1, y=t) # 错误
# 要实现上述功能，可以先将元组或列表解构
h(1, *t)      # 正确，y=(2,3)
h(1, *[2,3])  # 正确，y=(2,3)
```

以字典导入时，采用关键字语法：

```
def w(x, **y):
    pass

# 正确的调用
w(1)             # x=1, y={}
w(1, a=2, b=3)  # x=1, y={'a':2, 'b':3}

# 同样，要传入一个字典，也可以先将其解构
# 但注意，字典的键只能是字符串
d = {'a':2, 'b':3}
w(1, **d)  # y = {'a':2, 'b':3}

# 语法正确，但结果不一定是你所想的那样
w(1, y=d)  # y = {'y': {'a': 2, 'b': 3}}
```

在综合运用时，**一定要注意各种参数的先后顺序**。下面是一些判断题，请判断哪些函数的定义方式是正确的：

1. `def f1(x, y=2, *z)`
2. `def f2(x, *y, **z)`

3. `def f3(x, **y, *z)`

4. `def f4(x, *y, z=3)`

5. `def f5(x, *y, z=3, **t)`

答案：3 错，其他全对。但请注意，4,5 定义的 z 只能通过关键字语法赋值，它的优先级比不定长参数高，因此不会被后面的字典吸收。

6.3 常用的内置函数

6.3.1 sorted

sorted 函数用于给可迭代对象排序，包括列表、元组和字符串，返回按升序排列的列表。该函数有两个关键字参数，key 和 reverse，key 接收一个函数，这个函数会在每一个元素上调用，根据其结果进行排序。reverse 默认为 False，为 True 时则按降序排序。

```
>>> sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]      # 默认为升序

# key，根据第二项进行排序
def f(x):
    return x[1]
>>> sorted([('a',4),('b',2),('c',1)], key=f)
[('c', 1), ('b', 2), ('a', 4)]
```

6.3.2 map

map 函数接受一个函数、一个或多个可迭代对象。简单来说，map(f,[x1,x2,...]) = [f(x1),f(x2),...]. 但注意，map 返回的并不是一个列表，而是一个内建的 map 对象，虽然也属于可迭代对象，很多行为和列表相同，但不能直接打印，需要先用 list() 转换为列表。下面是几个例子：

```
>>> list(map(int, input().split()))  # 1 2 3
[1, 2, 3]

# 多个列表
```

```
def f(x, y, z):
    return x+y == z
>>> list(map(f, [1,1,1], [2,3,4], [3,4,6]))
[True, True, False]
```

6.3.3 eval

eval 用于计算一个 Python 表达式的值。实际上，你在 IDLE 里输入的每一行，Python 都是通过调用 eval 来计算的。下面是几个例子：

```
# 各位数字的值
>>> eval('+'.join('123'))
6
# 读取字符串里的 Python 结构
>>> eval('[1,2,3,4]')
[1, 2, 3, 4]
```

6.3.4 all & any

all 和 any 都用于可迭代对象，其中 all 当且仅当所有元素都为 True 时返回 True，any 则只要一个为 True 就返回 True。元素除了 0、None、False、空（空字符串、空列表...）之外都算 True。我们可以简单地写出这两个函数：

循环版本：

```
def all(lst):
    for element in lst:
        if not element:
            return False
    return True

def any(lst):
    for element in lst:
        if element:
            return True
    return False
```

递归版本：

```
def all(lst):
    if len(lst) == 0:
        return True
    else:
        return lst[0] and all(lst[1:])

def any(lst):
    if len(lst) == 0:
        return False
    else:
        return lst[0] or any(lst[1:])
```

6.4 模块

Python 自带很多模块，这些模块涵盖了各方面的需求，有时候需要什么功能，就可以直接引用，十分方便。使用 import 语句导入模块：

```
# 如果要使用几个数学函数
import math # 仅引入模块，使用具体函数时要加 math 前缀
from math import sin,cos # 不引入模块，仅引入需要的函数
from math import * # 不引入模块，导入所有函数和全局变量

# 查看模块的可用内容
>>> import math
>>> dir(math)
[ ... , # 省略部分内容
'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
'perm', 'pi', 'pow', 'prod', 'radians', 'remainder',
'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

祝大家都有不错的成绩！