Python笔记

注：此笔记主要针对想在面试中使用Python，并且已经有一定Python基础的同学

1 list

1.1 创建 list

```
a = [0, 1, 2]
a.append(3) # a = [0, 1, 2, 3]
a += [4] # a = [0, 1, 2, 3, 4]
```

1.2 list 排序

```
a = [1, 3, 2]
a.sort() # 这里a在排序之后会变成[1, 2, 3]
# 注意，如果b = a.sort()，则b会是None
b = sorted(a) # 这里a不会变，b会包含排序好的a中的元素

# 下面是自定义排序
a = ['x', 'z', 'y']
dic = {'x': 0, 'y': 1, 'z': 2}
# 我想让a中的元素按照他们在dic中对应的value排序
a.sort(key = lambda x: dic[x])
b = sorted(a, key = lambda x: dic[x])

# Python排序默认从小到大，如果想从大到小，可以加负号，比如
a.sort(key = lambda x: -dic[x])
# 或者
a.sort(key = lambda x: dic[x], reverse = True)
```

1.2 list comprehension

```
# 这个是语法糖，可以把for循环写到list里面，同样地，dict也有类似的comprehension
b = range(3) # b = [0, 1, 2]
a = [i * i for i in b] # a = [0, 1, 4]
```

1.3 在面试中使用list构建矩阵（这里有个坑需要避免）

```
# 构造一个m*n的矩阵
# 下面是错误方法
a = [[0 for i in range(n)] for i in range(m)]
# 如果m = 2, n = 2, 则a = [[0, 0], [0, 0]]
a[0][0] = 1 #此时a = [[1, 0], [1, 0]]
# 所以这种方法构造出的矩阵是不对的，实际是一个数组被重复了m次

# 下面是正确的方法，利用list乘法的特性
a = [[0] * n for i in range(m)]
```

2 tuple

2.1 tuple的特点：不可变长

```
# 可以直接从list来初始化tuple
a = tuple([1, 2]) # a = (1, 2)

# tuple和list相比，最大的特点就是，一旦定义，不可改变长度以及内容，就像一个常数一样使用

# tuple的这个特性可以让我们用来同时比较多个数是否都相等
a = (1, 2)
b = (1, 2)
a == b # True

a = (1, 3)
a == b # False
```

2.2 tuple的写法：可以去掉括号

```
# 以下写法等价
a = (1, 2, 3)
a = 1, 2, 3

# 于是我们可以这样玩
x, y = y, x #交换两个变量的值
```

```
# 在一个函数中返回一个tuple: return res1, res2
# 然后这样调用:
res1, res2 = f()
```

3 dict

3.1 dict comprehension

```
a = {i: i * i for i in range(3)} # a = {0: 0, 1: 1, 2: 4}
```

3.2 dict 遍历

```
a = {0: 0, 1: 1, 2: 4}
# 判断一个元素是否在dict的key_set里面
# 不要这样写:
if 0 in a.keys(): do_something
# 要这样写:
if 0 in a: do_something

# 遍历key的时候这样写:
for i in a:
  print i, a[i]

# 遍历value:
for value in a.values():
  print value

# 同时遍历key和value:
for key, value in a.iteritems():
  print key, value
```

3.3 defaultdict类

```
# Python有个很好用的类，可以在里面填入默认值
# 如果你用dict的话，会这样:
a = {}
print a['key'] # 这里会报错，'key'不在a里面
```

```
# 如果你用defaultdict
from collections import defaultdict
a = defaultdict(int)
print a['key'] # 这里会输出0

a = defaultdict(list)
print a['key'] # 这里会输出[]（一个空的list）
```

4 set

4.1 初始化

```
a = set() # 空的set
a = set([1, 2, 2, 3]) # 用一个list初始化这个set，里面重复的元素会被干掉
a = {1, 2, 3} # 用一个常量set初始化这个set

a.add(4) # 向set里面添加值
```

4.2 set基本操作

```
a = set()
b = set()
c = a | b # 并集
c = a & b # 交集
c = a - b # 差集
c = a ^ b # (a | b) - (a & b)
```

5 string

5.1 strip

```
a = '\tabc\r\n'
b = a.strip('\r\n\t') # b = 'abc'
b = a.strip('\n\r\t') # b = 'abc'
# strip里面的参数比较特殊，你传入一个字符串，它会把字符串里面的每个字符都strip掉，所以传参数的时
# 还有一个函数叫rstrip，只切去字符串结尾的字符，这个在读文件洗数据的时候十分常用（读进来的文件时
```

5.2 split

```
a = ' abc  \t\t ab '
b = a.split() # b = ['abc', 'ab']
# split不传参数的时候功能十分强大，可以把所有连续的空格和tab符号都去掉

a = 'abc  ab' # 注意这里有两个连续的空格
b = a.split() # b = ['abc', 'ab']
b = a.split(' ') # b = ['abc', '', 'ab']
# 当split包含参数的时候，就是普通的标准split了
```

5.3 join

```
# 可以用join构造string
a = ' '.join(['1', '2', '3']) # a = '1 2 3'
# 注意用来join的list里面每个元素都必须是str类型
```

6 list 以及 str 下标

```
# 以下对数组适用，对str同样也适用

a = [0, 1, 2, 3]

# 取最开头元素
b = a[0] # b = 0

# 取最末尾元素
b = a[-1] # b = 3

# 取某一段数组
b = a[1:3] # b = [1, 2]

# 从开头一直取到某一个元素
b = a[0:3] # b = [0, 1, 2]
b = a[:3] # b = [0, 1, 2]

# 从某个元素取到末尾，跟上面稍有不同
```

```
# 以下是错误写法
b = a[1:-1] # b = [1, 2]
# 以下是正确写法
b = a[1:] # b = [1, 2, 3]

# 可以指定步长
# 步长为2
b = a[0:4:2] # b = [0, 2]
b = a[::2] # b = [0, 2]
# 步长为负，则数组会被反过来取
b = a[::-1] # b = [3, 2, 1, 0]

# 以上的方法可以让我们反转一个数组，反转数组还可以直接用reverse
a.reverse() # 这时候 a = [3, 2, 1, 0]
```

7 generator 和 iterator

7.1 generator

```
# 下面是一个generator的例子
def yrange(n):
  i = 0
  while i < n:
    yield i
    i += 1

y = yrange(3)
print y # 输出: <generator object yrange at 0x401f30>
a = y.next() # a = 0
a = y.next() # a = 1
a = y.next() # a = 2
a = y.next() # 这里会报错，因为到头了

# 因此，通俗来讲，函数中yield关键字会使得函数的返回值变成一个generator。yield可以被理解成retur
# 理解了上面一段话之后，再看上面的yrange函数，相信你会有新的理解

# 对generator的遍历可以像对list的遍历一样写
y = yrange(3)
```

```
a = []
for i in y:
  a.append(i)
# a = [0, 1, 2]

# 对generator取下一个值，可以使用内建（built-in）函数next，或者generator的本身自带的函数next
y = yrange(3)
y.next() # 0
next(y) # 1

# generator的好处就是省内存，不用一次把整个list放进内存
# xrange 和 xreadlines 两个函数的返回值虽然不是generator类型，但是他们的作用和generator很相似

# 比如range和xrange的区别
a = range(4) # a = [0, 1, 2, 3]
a = xrange(4) # a是一个object，它的数据类型就是xrange类型
# 所以，当你想进行大量循环的时候，不妨使用 for i in xrange(BIG_NUMBER)
# 因为range(BIG_NUMBER)是一个长度为BIG_NUMBER的数组，很占内存，而xrange(BIG_NUMBER)是一个ge

# 再比如读文件的时候
# 不好的写法
with open('test.txt') as f:
  for line in f.readlines():
    # f.readlines() 是一个list，里面每个元素是文件的每一行
    do_something

# 好的写法
with open('test.txt') as f:
  for line in f.xreadlines():
    # f.xreadlines() 是一个object，它的数据类型是file
    do_something

# 需要注意的是由于xrange不支持next操作，但是xreadlines()支持
a = xrange(5)
a.next() # 报错
next(a) # 报错

with open('test.txt') as f:
  a = f.xreadlines()
```

```
  print a.next() # 正确
  print next(a) # 正确
```

7.1 iterator

```
# 内建函数iter用来返回一个iterator，它的输入是一个iterable的object
a = iter([1, 2, 3])
print a # <listiterator object at 0x1004ca850>
# 同样可以用next来遍历a
a.next() # 1
next(a) # 2
```

8 lambda 表达式，map，reduce 和 filter

8.1 lambda 表达式

```
# lambda表达式在Python里面就是匿名函数，当你需要定义一个函数，又只想调用一次的话，给这个函数命名
lambda x, y: x + y
# 以上的式子表示定义了一个lambda表达式，输入有两个，输出是把两个输入相加
```

8.2 map

```
# map 是用来简化for循环写法的语法糖
a = [0, 1, 2]
b = map(str, a) # b = ['0', '1', '2']
# 因为str(0) == '0'

a = [0, 1, 2]
def f(x):
  return x + 1
b = map(f, a) # b = [1, 2, 3]
b = map(lambda x: x + 1, a) # b = [1, 2, 3]
```

8.3 reduce

```
# reduce 也是用来简化for循环写法的语法糖
a = [1, 2, 3, 4]
```

```
b = reduce(lambda x, y: x + y, a) # b = 10
```

8.4 filter

```
# filter 也是用来简化for循环写法的语法糖
a = [0, 0, 1, 0, 2]
b = filter(lambda x: x > 0, a) # b = [1, 2]
b = filter(lambda x: x, a) # b = [1, 2]
# filter接收的函数，如果返回值为True，则元素会被留下，否则会被剔除。最后留下的所有元素组成一个新l
# 注意，对于数字来说，返回值为0就是False，非零就是True
```

9 其他零碎知识和我个人的习惯

9.1 True & False

```
# 对于数字来说，0是False，非零是True，因此我会这样写
# 假设x是整数
if x: do_something # 优雅
if x != 0: do_something # 不优雅

# 对于list、dict和set来说，空是False，非空是True，因此我会这样写
# 假设x是list、dict或set
if not x: return 1 # 优雅
if len(x) == 0: return 1 # 不优雅
```

9.2 None

```
# 面试中的一个好习惯就是，对于函数的参数，首先判断传进来的东西是不是空指针（None）
def f(x):
  if x is None: # 优雅
    return 'error'
```

9.3 in

```
# 如果a是list、dict或set，你可以判断一个元素是否在a中
# 对于dict，判断的是这个元素是否是dict的keys中的一个
if x in a:
```

```
  do_something
if x not in a:
  do_something
# 但是，尽量不要对list用in，因为那样太慢了（要遍历list来查找）
```

9.4 Python 的函数是 pass by reference values 的

```
# C++是pass by value
# Java是pass by reference
# Python和以上二者都不一样，是pass by reference values的

a = [1, 2, 3]
def f(x):
  x = 1
f(a) # a仍然是[1, 2, 3]

a = [1, 2, 3]
def f(x):
  x.append(4)
f(a) # a会变成[1, 2, 3, 4]
```

9.5 关于代码风格

```
# 代码风格推荐参考 Google Python Style Guide
```