# Python 学习笔记（上）

这份笔记是我在系统地学习python时记录的，它不能算是一份完整的参考，但里面大都是我觉得比较重要的地方。

# 基础知识

## 基本输入输出

将结果输出到指定文件

```
fp=open(r'D:\mytest.txt', 'a+')
print>>fp, "Hello world"
print('Hello, world!', file=fp)     # 3.x
fp.close()
```

## 模块导入与使用

1. `import 模块名`
2. `from 模块名 import 对象名 [as 别名]`
3. 显示预加载模块 `sys.modules.items()`
4. 重新加载模块 `imp.reload()` , `importlib.reload()`
5. 模块添加顺序 当前目录， `sys.path` , 可以用 `append()` 添加自定义文件夹
6. 模块导入顺序:**标准库>第三方扩展>自定义和开发的本地模块**

## `__name__` 属性

1. 脚本作为**模块**被导入，自动被设置为模块名
2. 脚本**独立运行**，被设置为 `__main__`
3. 可以通过 `if __name__ == '__main__: ....` 来控制运行方式

## 编写包

> *包是python 用来组织命名空间和类的重要方式，可以看作是包含大量Python程序模块的文件夹。包的每个目录都必须包含一个* `__init__.py` *文件。主要用于设置* `__all__` *变量以及执行初始化包所需要的代码，其中* `__all__` *中定义的对象可在* `from .. import *` *时被全部正确导入*

## 补充

位运算: `&` `|` `^` （位异或） `<<` `>>`

# Python序列

无序序列：字典、集合
双向索引：列表、元组、字符串

## 列表

尽量从列表尾部进行元素增加与删除操作，否则有的操作会导致大量的元素移动。
列表对象常用方法

| 方法(list.xxx) | 说明 |
| --- | --- |
| `append(x)` | 将元素x添加到列表尾部 |
| `extend(L)` | 将L中所有元素添加到列表尾部 |
| `insert(index,x)` | 将x插入到list的指定位置 |
| `remove(x)` | 从列表中删除首次出现的指定元素 |
| `pop([index])` | 删除列表指定位置的元素，默认最后一个（-1） |
| `clear()` | 删除列表中的所有元素，保留列表 |
| `index(x[,start[, stop]])` | 从start开始到stop返回第一个值为x的元素 |
| `count(x)` | 计算x在列表中的出现次数 |
| `reverse()` | 元素原地翻转 |
| `sort()` | 原地排序 |
| `copy()` | 返回列表对象的**浅复制** |

浅复制，分片复试
eg:

```
b = [1,2,3]
a = b        # 如此修改a或b时两者都会改变
c = b.copy()     # 浅复制，此时b和c的原地修改列表不会改变对方的值

c = b[:]         # 分片的方式复制
```

## 创建与删除

`range([start,] stop[, step])` 产生可迭代对象(惰性求值)
python2.x: `xrange` 和 `range` 与3.x不同

## 元素增加

1. `+` 元素法：本质创建新列表
2. 列表对象的 `append()` 方法：原地修改列表（真正意义上尾部添加）
   **基于值的自动内存管理方式**：对象修改时，不是直接修改变量的值，而是使变量指向新的值（可用id查看）。对于可变序列类型，直接修改变量值与上述相同；使用下标或对象自身方法来增删元素，对象再内存中的起始地址是不变的，仅仅是被改变值的元素地址发生变化。
3. `extend()` 可以将另一个迭代对象的所有元素添加到对象尾部。（不改变内存首地址，原地操作)
4. `insert()` 在任意位置插入元素 优先考虑 `pop()` 和 `append()`
5. 使用**乘法**扩展**列表对象**，新列表是旧列表的重复
   对**包含列表的列表**乘法时，并不是元素的复制而是对已有对象的引用

```
a = [1,2]
a = a*2

a = [1,2]
b = [[a]*2]*3

>>> b[0][0][0] = 1
>>> b
[[[1, 2], [1, 2]], [[1, 2], [1, 2]], [[1, 2], [1, 2]]]

>>> b[0][0][0] = 0
>>> b
[[[0, 2], [0, 2]], [[0, 2], [0, 2]], [[0, 2], [0, 2]]]
```

## 列表元素的删除

1. `del` 删除指定位置上的元素 `del l[3]` or `del(l[3])`
2. `pop()` 删除列表指定位置的元素，默认最后一个（-1）
3. `remove()` 从列表中删除首次出现的指定元素
   问题：**删除列表中指定元素的所有重复**

## 列表元素访问与计数

index
count() 可用于元组、字符串以及range对象

```
range(10).count(3)
(3,3,2,45).count(3)
'ahisndfoqhnfg'.count('ahi')
```

## 成员资格判断

`x in alist` ， `a not in blist` ， `a,b in zip(aList, bList)`
用于可迭代对象（元组、字典、range、字符串、集合）

## 切片操作

直接切片产生的是一个新的列表，但是用切片的方式索引并更新列表，会在旧列表上进行。

```
l[::]
l[::-1]  l的颠倒
l[::2}
l[1::2]
l[3::]
l[3:6:1]
l[3:5}
l[100:]
k = [1,2,3,4,5,6]
k[1:3] = 'abc'
>> k = [1,a,b,c,4,5,6]
```

## 列表排序

`lista.sort(key=lambda x:len(str(x)))` ：用lambda自定义排序
`lista.sort(reverse=True)` ：降序排序

`reversed()` 内置函数支持对列表元素进行逆序排列，不对原表进行修改，返回一个逆序排列后的迭代对象
`newlist = reversed(lista)`

## 序列操作的常用内置函数

| 函数 | 说明 |
| --- | --- |
| cmp(seq1, seq2) | 对两个列表进行比较；1：前者大，-1：后者大，0：元素完全相同 |
| len(seq) | 返回序列的元素个数，同样用于可迭代对象 |

| 函数 | 说明 |
| --- | --- |
| max(),min() | 返回序列中的最大或最小值，要求所有元素可比较大小。字典默认比键，比值使用dict.values() |
| sum(seq) | 求和运算 |
| zip(seq1,seq2,...) | 将多个序列对应位置的元素组合为元组，返回zip对象 |
| enumerat(seq) | 枚举可迭代对象的元素，返回枚举对象。<br>枚举对象的每个元素是包含下标和元素值的元组 |

```
d = {'a': 1, 'b': 2, 'c':3}
for i, v in enumerate(a):   #字典默认枚举key
    print(i, v)
0 'a'
1 'b'
2 'c'

for index, ch in enumerate('FJFSS'):
    print((index, ch), end=',')
```

### 列表推导式!!

## 元组

### 序列解包

字典的解包
`**` 解包键值对， `*` 解包字典键

```
{'x':1 , **{'y':2}}
```

### 生成器推导式

## 字典

### 字典创建与删除

```
1. `a = {}`
2.
3. key = [...]
4. value = [...]
5. d = dict(zip(key, value))
6.
7. d = dict.fromkeys([...])
```

### collections

Counter
defaultdict
OrderedDict: 有序字典

## 集合

集合中不包含重复的元素，由于集合的操作有相应优化，使用集合生成不含重复数的效率较高。
集合推导式

```
1. {x.strip() for x in range('he','she ',''I'}
2.
3. import random
4. x = {random.randint(1,500) for i in range(100)}
```

## 内置方法 `sorted()`

### 与列表对象的 `sort()`

列表对象的 `sort` 在原地对列表进行排序，而内置函数 `sorted` 不是原地排序是返回一个新的序列。并且后者支持元组、字典。

### 字典排序

`key`，`cmp`（python2.x）
使用key参数的值是一个函数对象
该函数只有1个参数，是待比较的元素
只有一个返回值即该元素的关键字，根据其确定元素的大小。

```
1. # lambda
2. sorted(phonebook, key = lambda s:phonebook[s])
3.
4. # itemgetter
5. phonebook = {'L':'123','S':'134','K':'213'}
6. from operator import itemgetter
7. sorted(phonebook.items(), key=itemgetter(0))    # 1则为按值排序
```

**多关键字排序**
先对次关键字排序，再对主关键字排序

# 其他

`enumerate(iterable)`

`sum(iterable, start=0, /)`
Return the sum of a ‘start’ value (default: 0) plus an iterable of numbers
When the iterable is empty, return the start value.
This function is intended specifically for use with numeric values and may reject non-numeric types.

# 数据结构

## 堆

# 选择与循环

# 条件表达式

1. 条件表达式的值只要不是 `False` ,0, `None` 或者空列表，range对象等其它空迭代对象python解释器均认为与 `True` 等价
2. 关系运算符可以连续使用，e.g. `1<2<3`
3. str的join方法 `s.join(iterable)` ，将s作为分隔符插入，返回字符串

# 选择结构

1.单分支结构

```
1. if condition:
2.     action
```

2.双分支选择结构

```
1.
2. if condition:
3.     action1
4. else:
5.     action2
```

三元操作符

```
value1 if condition else value2

b = 5 if a >13 else 9
```

3.多分支选择结构
4.选择结构的嵌套

判断变量是否是type的实例， `isinstantce(x,type)`

## 时间 `time` 和 `datetime`

具体见python标准库笔记

```
import time
date = time.localtime()
y,m,d = date[:3]
# time.struct_time(tm_year=2020, tm_mon=1, tm_mday=16, tm_hour=21, tm_min=25, tm_sec=37, tm_wday=3, tm_yday=16, tm_isdst=0)


import datetime
Today = datetime.date.today()
Today - datetime.date(Today.year,1,1)+datetime.timedelta(days=1)# 今年是第几天
Today.timetuple()   # datetime.date -> time
Today.replace(month = 1)    # 替换月份
now = datetime.now()
datetime.timedelta(weeks = -5)
```

# 循环结构

## for和while

for会自动调用迭代器的next()方法，同时会捕获 `StopIteration` 异常并结束循环

常见用法

```
while 条件表达式:
    循环体
for 变量 in 序列或其他可迭代对象:
    循环体
```

此外两者皆可带 `else` 子句，如果循环因**条件表达式不成立**而结束此时会执行else子句。（break结束的不会执行else）

```
for ..... :
    循环体
else:
    .....

while ..... :
    循环体
else:
    ..... :
```

### 循环结构的优化

1. 尽量减少循环内的不必要计算。
2. 使用多重循环嵌套应尽量减少内层循环中的不必要计算。
3. 尽量引用局部变量。
4. 引用模块的方法可将其转换为局部变量。

```
import math
loc_sin = math.sin      # 法一
a = math.sin(30)        # 原始
b = loc_sin(30)
# 法二
from math import sin as sin
c = sin(30)
```

### `break` 和 `continue`

`break` : 跳出循环
`continue` :终止本次循环并开始下一轮

# 字符串与正则表达式

## 字符串

‘这是一个字符串’
r’ 非转义的原始字符串，例如反斜杠+n不转义为换行’
u’ 表示unicode字符串，代表对字符串进行unicode编码’
b’ 表示python2中的str方式’
e.g. `r'C:\now'` 无论是否原始字符串，都不能以反斜杠作为结尾。

### 字符串格式化

`'% [-] [+] [0] [m] [.n]' % x`
格式字符
字符类：%s, %r, %c, %%,

整数类：%d, %i, %o, %x,
指数、浮点数类：%e, %E, %f/%F, %g, %G

str.format `{0:.2f}`

```
print("the number {0:,} in hex is: {0:#x}".format(5555,55))
print("the number {1:,} in oct is: {1:#o}".format(5555,55))
print("{name},{age}".format(name="Zachary", age="21"))
position = (5,8,11)
print("x{0[0]},y{0[1]},z{0[2]}".format(position))
weather=[('Monday','rain'),('Tuesday','sunny')]
formatter = "Weather of {0[0]} is {0[1]}".format
for item in map(formatter, weather):
    print(item)
for item in weather:
    print(formatter(item))
```

## 字符串常用方法

- 模式匹配
  `find()` , `rfind()` , `index` , `rindex`
  加上 `r` 从后往前找， `find` 不存在返回-1， `index` 不存在抛出异常, `count` 统计子串出现次数

- 分割

| 符号名 | 描述 |
| --- | --- |
| split() | 以指定字符从字符串 左端 开始将其分割成**多个字符** |
| rsplit() | 右端，返回分割结果的列表 |
| partition() | 以指定字符从字符串 左端 开始将其分割 |
| rpartition() | 分成**三部分**：分隔符前，分隔符，分隔符后 |
| | 不指定则为任何空符号（空格，回车） |
| | 还可以指定分割次数split(chr, num) |

```
>>> s.split(',')
['apple', 'grape', 'orange']
>>> s.rsplit(',')
['apple', 'grape', 'orange']
>>> s.partition(',')
('apple', ',', 'grape,orange')
>>> s.rpartition(',')
('apple,grape', ',', 'orange')
```

- 连接（ `join` ）
  使多个字符串连接，并在相邻两个字符串中插入指定字符

```
li = ['apple', 'peach', 'banana']
sep = ','
s = sep.join(li)
```

`+` 也可用于连接但效率较低

```
import timeit
timer1 = timeit.Timer('func1', 'from where import func1')
print(timer1.timeit(Number))
```

- 大小写

| 方法名 | 描述 |
| --- | --- |
| lower() | 转换字符串为小写 |
| upper() | 转换为大写 |
| capitalize() | 首字母大写 |
| title() | 标题式大写 |
| swapcase() | 和标题式相反 |

- `replace`
  replace不是在原来的字符串上进行修改，而是产生一个新的字符串。

```
s = '中国，美国'
s.replace('中国', '中华人民共和国')
```

- `maketrans()` , `translate()`

生成字符映射表， 按映射表关系转换字符串并替换其中的字符。可同时处理多个不同字符，replace无法满足。

```
table = ' '.maketrans('abcdefg123', 'uihjedg&^%')
s = 'htllo worsld fjistw'
s.translate(table)
```

- `strip(), rstrip(), lstrip()`
  删除两端，左端，右端的空白或连续的指定字符
- `eval()`
  尝试把字符串化为python 表达式并求值

- `in` : `'a' in 'apple'`
- `startswith(), endswith()`
  检查字符串是否以指定字符串开始或结束，还可限制检测范围
- 判断是否为

| 方法 | 描述 |
| --- | --- |
| isalnum() | 是否是数字或者字母 |
| isalpha() | 是否是字母 |
| isdigit() | 是否是数字字符 |
| isspace() | 是否是空白字符 |
| isupper() | 是否为大写字母 |
| islower() | 是否为小写字母 |

- `center()` 、 `ljust` 、 `rjust`
  返回指定宽度的新字符串，源字符串居中,左对齐,右对齐，出现在新字符串中

```
>>> 'Hello World!'.center(20, '=')
'====Hello World!===='
```

## 字符串常量

```
import string
string.digits
string.punctuation
string.letters
string.ascii_letters
string.printable    # 可打印字符
string.lowercase
sting.uppercase

# 生成8位随机密码
import random, string
x = string.digits + string.ascii_letters + string.punctuation
p1 = ''.join([random.choice(x) for i in range(8)])
p2 = ''.join(random.sample(x,8))
```

random

| 方法 | 描述 |
| --- | --- |
| choice() | 从序列中任选一个元素 |

| 方法 | 描述 |
| --- | --- |
| getrandbits() | 生成指定二进制位数的随机整数 |
| randrange() | 指定范围内随机数 |
| randint() | 指定范围内整数 |
| sample() | 指定数量不重复元素 |
| betaavariate() | 贝塔分布 |
| gamavariate() | 伽马分布 |
| gauss() | 高斯分布 |

## 可变字符串

> *在Python中字符串属于不可变对象，不支持原地修改，如果需要修改其中的值必须重新创建。然而，如果确实需要一个原地修改的Unicode数据对象，可以使用* `io.StringIO` *对象或* `array` *模块*

```
import io
s = "Hello, world"
sio = io.StringIO(s)
sio.getvalue()
sio.seek(7)          # 更改指针位置 (从x的下一位开始
sio.write("there!")
sio.getvalue()
```

`string`

```
import string
a = string.ascii_letters
b = string.digits
```

## 字符串驻留

*链接：*
*https://www.nowcoder.com/questionTerminal/653cf85ded784dba91eab336f0a0b742?orderByHotValue=1&mutiTagIds=573&page=1&onlyReference=false (https://www.nowcoder.com/questionTerminal/653cf85ded784dba91eab336f0a0b742?orderByHotValue=1&mutiTagIds=573&page=1&onlyReference=false)*
*来源：牛客网- 字符串驻留是一种仅保存一份相同且不可变字符串的方法。*

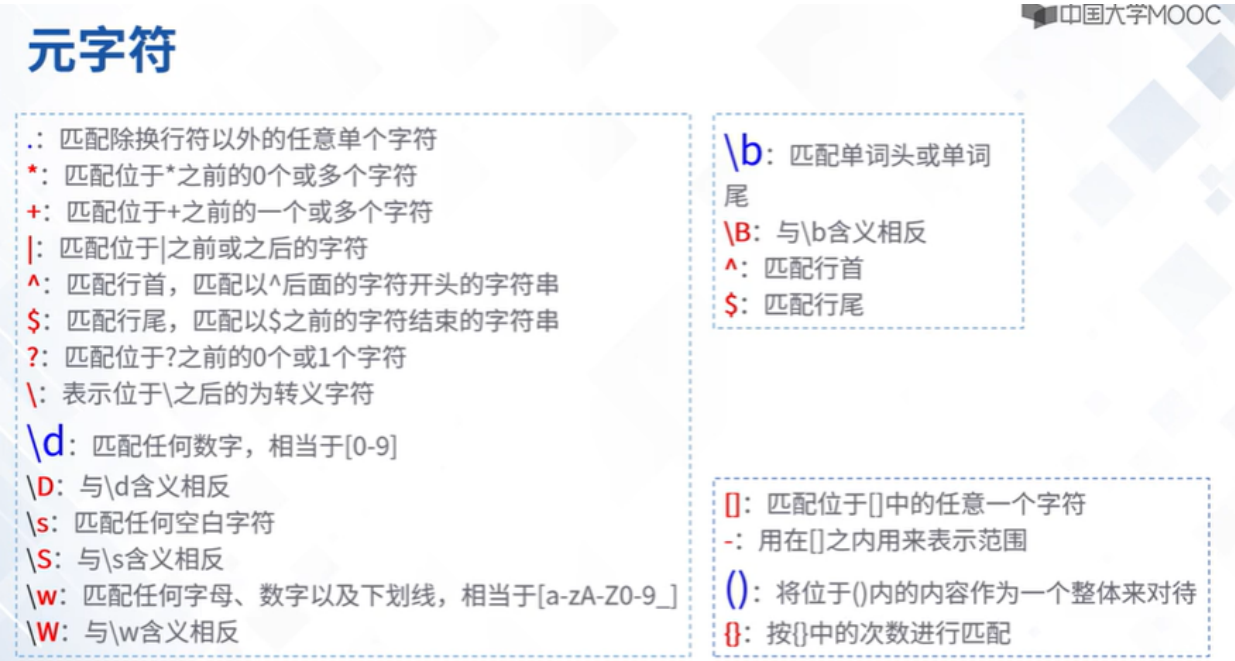
- *原理*
  - *系统维护interned字典，记录已被驻留的字符串对象。*
  - *当字符串对象a需要驻留时，先在interned检测是否存在，若存在则指向存在的字符串对象，a的引用计数减1；*
  - *若不存在，则记录a到interned中。*
- *优点*
  - *在字符串比较时，节省大量内存。非驻留比较效率为o(n)，驻留时比较效率为o(1)。*
- *驻留情况*
  - *字符串只在编译时进行驻留，而非运行时。*
  - *字符串长度为0和1时，默认都采用了驻留机制。*
  - *字符串>1时，且只含大小写字母、数字、 下划线时，才会默认驻留。*
  - *用乘法得到的字符串*
    - *乘数为1时*
      - *仅含大小写字母、数字、 下划线，默认驻留。*
      - *含其他字符串*
        - *长度<=1，默认驻留。*
        - *长度>1，默认不驻留。*
    - *乘数大于1时*
      - *仅含大小写字母、数字、 下划线，长度<=20,默认驻留*
      - *仅含大小写字母、数字、 下划线，长度>20,默认都不驻留*
      - *其他字符串时，和长度无关，不驻留。*
  - *字符串被sys.intern() 指定驻留。*
  - *[-5, 256]之间的整数数字，Python默认驻留。*

# 正则表达式

元字符：适配符、数量定义符、边界定义符、成组定义符

**示例**

`'[\u4e00-\u9fa5]'` ：匹配给定字符串中所有汉字




### `re` 模块的主要方法

| 方法 | 说明 |
| --- | --- |
| compile(pattern[, flags]) | 创建模式对象 |
| search(pattern, string[, flags]) | 在整个字符串中寻找模式，返回match对象或None |
| match(pattern, string[, flags]) | 从字符串的开始处匹配模式，返回match对象或None |
| findall(pattern, string[, flags]) | 列出字符串中模式的所有匹配项 |
| split(pattern, string[, maxsplit=0]) | 根据模式匹配分割字符串 |
| sub(pat, repl[, count=0]) | 将字符串中所有pat匹配项用repl替换 |
| escape(string) | 将字符串中所有特殊正则表达式字符转义 |

### 子模式与match对象

# 函数设计与使用

详见Python语言学习笔记（下）

## 文章信息

标题：Python 学习笔记（上）

作者：快刀切草莓君

分类：编程语言

发布时间：2020年1月27日

最近编辑：2020年3月28日

浏览量：99

## 快刀切草莓君

Weibo (https://weibo.com/3031783235)  Github (https://github.com/Zaaachary)  关于

## 友情链接

Touko (https://wasteland.touko.moe/)  老齐 (https://itdiffer.com/)  Mr_Wang (http://blog.wh241.cn/)

SlyLi (http://blog.slyli.cn/)

互联网ICP备案：闽ICP备18004703号-1 (http://beian.miit.gov.cn/)