# 第一章 python基础应用

#本章主要介绍一些在python的基础应用，主要从以下几个方面来介绍：

一、 python基础路径

- 路径操作
- 模块引用

二、 python常用语法结构

- 运算逻辑符
- 控制流语句
- 常用关键字
- 其他python输入及格式化输出
- lambda函数

三、 python常用类型/序列函数

- 列表
- 字典
- tuple
- 元组
- 常用序列函数

## 1.python基础路径

### 1.1 路径操作

#### 1.1.1 绝对路径和相对路径

文件路径就是文件在电脑中的位置，表示文件路径的方式有两种，相对路径和绝对路径。其中，相对路径就是相对于当前文件的路径，它相对于程序的当前工作目录；本地路径是指文件在硬盘上真正存在的路径，总是从根文件夹开始。

在表示路径中我们常用点（.）和点点（..）文件夹，它们不是真正的文件夹，而是可以在路径中使用的特殊名称。单个的句点（“点”）用作文件夹目名称时，是“这个目录”的缩写。两个句点（“点点”）意思是父文件夹。

[2]
```
##处理绝对路径和相对路径
import os
print(os.path.abspath('.'))#返回一个相对路径的绝对路径
print(os.path.abspath('.\data'))#返回一个相对路径的绝对路径
#检查一个路径是否绝对路径，如果是则返回true
print(os.path.isabs('.'))
print(os.path.isabs(os.path.abspath('.')))
```

```
F:\jupyter\浩彬分享的代码
F:\jupyter\浩彬分享的代码\data
False
True
```

[3]
```
print(os.path.abspath('.'))
```

```
F:\jupyter\浩彬分享的代码
```

[5]
```
```

### 1.1.2 路径的生成与拼接

[7]
```
#使用os.path.join生成路径
from pathlib import  Path
home = "D:\\jupyter"
data_folder=os.path.join(home,"python","Code")
data_filename = os.path.join(data_folder,"Chapter
2","ionosphere.data")
print(data_filename)
```

```
D:\jupyter\python\Code\Chapter 2\ionosphere.data
```

[6]
```
print('1:',Path()) #取当前路径
print('2:',Path('a','b','c/d')) #字符合成
print('3:',Path('/etc'))  #直接写入绝对路径
```

```
1: .
2: a\b\c\d
3: \etc
```

#举例路径的生成与分解

[ ]
```
### 创建新文件夹。如果中间路径的文件夹不存在，那么就会把中间文件夹创建上
os.makedirs('C:\\onedrive\\jupyter\\testmkdir\\1')
#也可以使用mkdir创建，但是中间目录必须存在，否则报错
os.mkdir('C:\\onedrive\\jupyter\\testmkdir2\\2')
```

[8]
```
#举例路径的生成与分解
p = Path('a','b','f/z')
print(p)
p2 = p / 'new_str'
print(p2)
p3 = '/new_str' / p
print(p3)
p4 = p3 / p
print(p4)
```

```
a\b\f\z
a\b\f\z\new_str
\new_str\a\b\f\z
\new_str\a\b\f\z\a\b\f\z
```

[9]
```
p3.joinpath('etc','ab',Path('http')) # joinpath 拼接，等效于
p3.joinpath('etc','ab','http')
```

```
WindowsPath('/new_str/a/b/f/z/etc/ab/http')
```

[10]
```
print(p4.parts)#路径分割: parts方法
```

```
('\\', 'new_str', 'a', 'b', 'f', 'z', 'a', 'b', 'f', 'z')
```

# 获取路径

[14]
```
##路径的获取
p4 = Path('/new_str/a/b/f/z/a/b/f/z')
print(p4.parent)#获取目录
print(p4.parents)#获取父目录

for x in p4.parents:
    print(x)
```

```
\new_str\a\b\f\z\a\b\f
<WindowsPath.parents>
\new_str\a\b\f\z\a\b\f
\new_str\a\b\f\z\a\b
\new_str\a\b\f\z\a
\new_str\a\b\f\z
\new_str\a\b\f
\new_str\a\b
\new_str\a
\new_str
\
```

[13]
```
str(p)
```

```
'\\etc'
```

[11]
```
p = Path('/etc')
print(str(p),'\n',bytes(p))
```

```
\etc
 b'\\etc'
```

### 1.1.3 其它一些路径扩展操作

[15]
```
p = Path('/a/b/c/d/g.tar.gz.a')

print(p.name) #取basename
print(p.suffix) #取目录最后一部分的扩展名，最后一个小数点部分。包括小数点。
print(p.suffixes) #取从左到右第一个小数点右侧的部分，小数点分割。每部分都带小数点。
print(p.stem) #取basename 排除最后一个小数点部分
print(p.with_name('233.tgz')) #补仓后缀
print(p.with_suffix('.txt')) #补后缀
```

```
g.tar.gz.a
.a
['.tar', '.gz', '.a']
g.tar.gz
\a\b\c\d\233.tgz
\a\b\c\d\g.tar.gz.txt
```

[16]
```
p = Path('../tmp/')
print(p.cwd())  #获取当前所在目录
print(p.home())  #获取当前用户的家目录
print(p.is_dir()) #判断是否为目录
print(p.is_file())#判断是否为文件
print(p.is_symlink()) #判断是否为软连接
print(p.is_socket()) #判断是否为socket 文件
print(p.is_block_device()) #判断是否为块设备
print(p.is_char_device()) #判断是否为字符设备
print(p.is_absolute()) #判断是否为绝对路径
print(p.resolve()) #返回绝对路径，推荐使用
print(p.absolute()) #返回绝对路径
print(p.exists()) #判断文件或者目录是否存在
```

```
F:\jupyter\浩彬分享的代码
C:\Users\x1c
False
False
False
False
False
False
False
F:\jupyter\tmp
F:\jupyter\浩彬分享的代码\..\tmp
False
```

[ ]
```
p5 = Path('/b/b/c')
p5.as_uri() #返回文件的uri

#说明：不能使用相对路径
```

[17]
```
#设置路径变量
p1 =Path('/jupyter/tmp/a/b')
p2 =Path('/jupyter/tmp/a/b/3.txt')

# 创建文件
p1.mkdir(parents= True,exist_ok= True)
p2.touch()

# 检测文件是否创建成功
print(p2.exists())

#在a/b下创建新文件
p1 /= 'readme.txt'
p1.touch()
print(p1)
print(p1.exists())
```

```
True
\jupyter\tmp\a\b\readme.txt
True
```

## 1.2 模块引用

在Python中，模块就是一个有.py扩展名、包含Python代码的文件。一般分为import第三方库和自己创建的方法模块：

### 1.2.1 引用第三方库

[ ]
```
# 直接导入Python的内置基础数学库，当要使用其中的某个变量可以使用math.cos
import math
print(math.cos(math.pi))

# 可以起个别名，方便使用或避免名字冲突
import math as m
print(m.cos(m.pi))

# 从math中导入cos函数和pi变量,可以直接使用cos
from math import cos, pi
print(cos(pi))
```

[ ]

### 1.2.2 引用创建的py文件

本文件为**python**基础，若调用同级目录，可以直接**import**

目录结构如下

- script
  - --python基础
  - --myfun.py

[19]
```
#可以直接使用
import myfun
#myfun.add(3,4)
```

本文件为**python**基础，若调用子目录下的模块，需要增加**from import**

目录结构如下

- script
  - --python基础
  - --myfun.py
  - --sub_fun
    - --sub_myfun.py

```
[ ]  from sub_fun import sub_myfun
     sub_myfun.sub_add(4,5)
```

本文件为**python**基础，若调用其他子目录下的模块，需要把所调用**py**文件对应上一级目录的路径添加到配置环境中，然后**from import**

目录结构如下

- upp_fun
  - --upp_myfun
- script
  - --python基础
  - --myfun.py
  - --sub_fun
    - --sub_myfun.py

```
[ ]  #指定路径
     import sys
     sys.path.append('C:\\onedrive\\jupyter')
     #应该也可以sys.path.append('..')
     from upp_fun import upp_myfun
     upp_myfun.upp_add(4,5)
```

## **init.py** 文件说明

在python模块的每一个包中，都有一个**init**.py文件（这个文件定义了包的属性和方法）然后是一些模块文件和子目录，假如子目录中也有**init**.py 那么它就是这个包的子包了。当你将一个包作为模块导入的时候，实际上导入了它的**init**.py 文件。

一个包是一个带有特殊文件 **init**.py 的目录。**init**.py 文件定义了包的属性和方法。其实它可以什么也不定义；可以只是一个空文件，但是必须存在。如果 **init**.py 不存在，这个目录就仅仅是一个目录，而不是一个包，它就不能被导入或者包含其它的模块和嵌套包。

**init**.py 中还有一个重要的变量，叫做**all**。 当全部导入模块时（from 模块 import *），import 就会把注册在包**init**.py 文件中 **all** 列表中的子模块和子包导入到当前作用域中来

**init**.py的作用：

package的标识，不能删除 定义package中的**all**，用来模糊导入 编写Python代码(不建议在**init**中写python模块，可以在包中在创建另外的模块来写，尽量保证**init**.py简单）

# 2. python常用语法结构

## 2.1 python运算符

https://www.runoob.com/python/python-operators.html

- 算术运算符
- 比较（关系）运算符
- 赋值运算符
- 逻辑运算符
- 位运算符

```
[ ]  a+b          a加b
     a-b          a减b
     a*b          a乘以b
     a/b          a除以b
     a//b         a除以b，结果只取整数部分
     a**b         a的b次幂
     a&b          a或b都为True，则为True；对于整数，取逐位AND
     a|b          a或b有一个为True，则为True；对于整数，取逐位OR
     a^b          对于布尔，a或b有一个为True，则为True，二者都为True，为False；
对于整数，取逐位EXCLUSIVE-OR
     a==b         a等于b，则为True
     a!=b         a不等于b，则为True
     a<b，a<=b     a小于（或小于等于）b，则为True
     a〉b，a>=b    a大于（或大于等于）b，则为True
     a is b       a和b引用同一个Python对象，则为True
     a is not b    a和b引用不同的Python对象，则为True
```

```
[ ]  #is比较与==运算符不同,s 检查的是两个对象是否相同，而不是相等
     a = b = [1,2,3]
     c = [1,2,3]
     a == b, a == c,a is b,a is c
     #变量 x 和 y 指向同一个列表，而c 指向另一个列表，虽然两个的值都是一样，但是并
     不是同一个对象，所以这里 a is c 返回 False
```

## 2.2 控制流语句

### 2.2.1 for循环

for循环是在一个集合（列表或元组）中进行迭代，或者就是一个迭代器。

[21]
```
#range可以接受三个参数，分别是起始值，上限（不包括上限），步长
for i in range(10,19,2):
    print(i)

#for循环在一行，[for]
x= [1 for i in range(5)]
x
```

```
10
12
14
16
18

[1, 1, 1, 1, 1]
```

### 2.2.2 if、elif和else

if是最广为人知的控制流语句。它检查一个条件，如果为True，就执行后面的语句。可以只使用if而不使用else 可以使用if-elif-else的结构，中间可以包括多个elif。

[1]
```
##猜数字游戏
import random
print('请猜一个1-20的数字，你有6次机会')
answer = random.randint(1,20)
for i in range(1,3):
    print('请输入：')
    num = int(input())
    if num > answer:
        print("太大了，再猜一个")
    elif num < answer:
        print('太小了，再猜一个')
    else:
        break
if num == answer:
    print('恭喜你猜中了！正确答案是%d' %answer )
else:
    print('已经超过限制次数了，正确答案是%d' %answer)
```

```
请猜一个1-20的数字，你有6次机会
请输入：
2
太小了，再猜一个
请输入：
34
太大了，再猜一个
已经超过限制次数了，正确答案是5
```

### 2.2.3While循环

while循环指定了条件和代码，当条件为False或用break退出循环，代码才会退出

```
[ ]  ##使用标志
     #导致程序结束的事件有很多时，如果在一条while 语句中检查所有这些条件，将既复杂又困难。
     #在要求很多条件都满足才继续运行的程序中，可定义一个变量
     #用于判断整个程序是否处于活动状态。这个变量被称为标志
     prompt = "\nTell me something, and I will repeat it back to you:"
     prompt += "\nEnter 'quit' to end the program. "
     active = True
     while active:
         message = input(prompt)
         if message == 'quit':
             active = False
         else:
             print(message)
```

### 2.2.3 pass、continue、break

```
[ ]  #pass是Python中的非操作语句。代码块不需要任何动作时可以使用（作为未执行代码的占位符）；因为Python需要使用空白字符划定代码块，所以需要pass
     x=0
     if x < 0:
         print('negative!')
     elif x == 0:
         pass
     else:
         print('positive!')

     #continue,跳过该次循环;if+continue,符合条件的跳出循环
     #输出双数
```

```
i = 0
while i < 10:
    i=i+1
    if i%2 >0:
        continue
    print(i)

#break，跳出整个循环
i=1
while 1:
    print(i)
    i = i+1
    if i >5:
        break
```

### 2.2.4 try..except

try:

代码1

except:

代码2

finally：

代码3

代码1发生异常就执行代码2，无论正常与否都要执行代码3.

```
[ ]
#try和except
#raise：主动触发异常
try:
    raise EOFError
    print('EOFError 错误!')
except EOFError:
    print('EOFError异常。')

#try和else
list = [1, 2, 3, 4, 5, 6, 7, 8]
try:
    list.append(100)
    print(list[8])
except IndexError:
    print('数组越界')
else:
    print(list)

#try和finally
list = [1, 2, 3, 4, 5, 6, 7, 8]
```

```
try:
    list.append(100)
    print(list[10])
except IndexError:
    print('数组越界')
finally:
    print(list)

#raise 抛出一个异常，一旦执行raise语句，后面的代码就不执行了,可以结合try 使用
try:
    raise EOFError
    print('Hello World !')  #不执行
except EOFError:
    print('EOFError 错误')    #  打印 EOFError错误
```

[3]
```
try:
    raise EOFError
    print('EOFError 错误!')  #不执行
except EOFError:
    print('hello')
```

```
hello
```

[ ]
```
##with...as with语句时用于对try except finally 的简化
f=open('file_name','r')
try:
    r=f.read()
except:
    pass
finally:
    f.close()

等价于

with open('file_name','r') as f:
    r=f.read()
```

## 2.3 常用关键字

[4]
```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more
```

```
Here is a list of the Python keywords.  Enter any keyword to get more
help.

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass
```

### 2.3.1内置常量False、None、True

```
[ ]   #False == 0 ,True == 1,type(False),type(None)
```

### 2.3.2 逻辑与 或 非 and or not¶

优先级：not and or

```
[5]   #  x and y      如果 x 为 False 、空、0，返 回 x，否则返回 y
      #  x or y       如果 x 为 False、 空、0，返回  y，否则返回x
      #  not  x      如果 x 为 False、 空、0，返回 True，否则返回False

      a = '1'
      b = 1
      a and b,a or y,not a
```

```
(1, '1', False)
```

### 2.3.2 用来定义的关键字def、class、lambda

- def : 定义一个函数或方法；
- class: 定义一个类对象；
- lambda：定义一个匿名函数；

```
[ ]
class Student():
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print('name:', self.name)
        print('age:', self.age)

stu = Student('didi', 20)
stu.info()
```

```
[6]
def short_function(x):
    return x * 2

#等价于
equiv_anon = lambda x: x * 2

def apply_to_list(some_list, f):
    return [f(x) for x in some_list]
#虽然你可以直接编写[x *2for x in ints]，但是这里我们可以非常轻松地传入一个自定义运算给apply_to_list函数。
ints = [4, 0, 1, 5, 6]
apply_to_list(ints, lambda x: x * 2)
```

```
[8, 0, 2, 10, 12]
```

### 2.3.3 控制流关键字（参考以上语句部分）

if...elif...else...:条件判断；

for...in...:对可迭代对象循环遍历

for...in...else...:遍历正常完成则执行else后的代码；

continue:终止本次循环，继续下一次循环；

break：跳出循环；

while：循环结构；

return：函数返回值；

### 2.3.4 变量命令空间关键字

- global:将模块空间变量引入到局部空间修改；

- nonlocal:将本局部空间的外层空间变量引入到本层局部空间修改，用来嵌套函数内；

```
[ ]  #定义全局变量：变量名全部大写
     NAME = 'didi'

     #得到NAME
     def get_NAME():
         return NAME

     #重新设置NAME的值
     def set_NAME(name_value):
         global NAME
         NAME = name_value

     print(get_NAME())
     set_NAME('dididididi')
     print(get_NAME())
```

```
[ ]  #关键字nonlocal的作用与关键字global类似，使用nonlocal关键字可以在一个嵌套的函数中修改嵌套作用域中的变量
     def foo1():
         num = 2
         def foo2():
             nonlocal num
             num = 5
         foo2()
         print('num = ', num)

     foo1() # num = 5
```

### 2.3.5 del/yield

```
[7]  list = [1, 2, 3, 4, 5, 6, 7]
     del list[1]
     print(list)
```

```
[1, 3, 4, 5, 6, 7]
```

迭代器：所有你可以用在for...in...语句中的都是可迭代的:比如lists,strings,files...因为这些可迭代的对象你可以随意的读取所以非常方便易用,但是你

必须把它们的值放到内存里,当它们有很多值时就会消耗太多的内存.
生成器：生成器也是迭代器的一种,但是你只能迭代它们一次.原因很简单,因为它们不是全部存在内存里,它们只在要调用的时候在内存里生成。
yield 是一个类似 return 的关键字，只是这个函数返回的是个生成器
当你调用这个函数的时候，函数内部的代码并不立马执行 ，这个函数只是返回一个生成器对象
当你使用for进行迭代的时候，函数中的代码才会执行

[9]
```
# #yield在函数中的功能类似于return，不同的是yield每次返回结果之后函数并没有退出，而是每次遇到yield关键字后返回相应结果，并保留函数当前的运行状态，等待下一次的调用。
# 如果一个函数需要多次循环执行一个动作，并且每次执行的结果都是需要的，这种场景很适合使用yield实现。
# 包含yield的函数成为一个生成器，生成器同时也是一个迭代器，支持通过next方法获取下一个值。

def fun():
    for i in range(20):
        x=yield i
        print('good',x)
a=fun()
a.__next__()
x=a.send(5)
#print(x)
```

```
good 5
```

## 2.4 python输入及其格式化输出

### input输入

在python2.7中式需要raw_input()

[62]
```
#input可以接受一个参数，作为提示信息，当然参数可以是一个变量，方便输入更长的内容
prompt = "If you tell us who you are, we can personalize the messages you see."
prompt += "\nWhat is your first name? "
name = input(prompt)
print("\nHello, " + name + "!")
```

```
If you tell us who you are, we can personalize the messages you see.
What is your first name? lyt

Hello, lyt!
```

[63]
```
#需要注意的是，接受的输入即使是数字也会被转化为字符串，所以要想获得数字输入，要加int()
height = input("How tall are you, in inches? ")
height = int(height)
if height >= 36:
    print("\nYou're tall enough to ride!")
else:
    print("\nYou'll be able to ride when you're a little older.")
```

```
How tall are you, in inches? 100

You're tall enough to ride!
```

## 格式化输出

字符串格式化时百分号后面有不同的格式符号，代表要转换的不同类型，具体的表示符号如下面所示。

| 格式符号 | 表示类型 |
| --- | --- |
| %s | 字符串 |
| %r | 不管什么都打印出来(很多时候和%s类似，但是%r还会输出进一步的对象) |
| %d/%i | 十进制整数 |
| %u | 十进制整数 |
| %o | 八进制整数 |
| %x/%X | 十六进制整数 |
| %e/%E | 科学计数 |
| %f/%F | 浮点数 |
| %% | 输出% |

[13]
```
name = 'haobin'
zone = 'com'
print('www.%s.%s'%(name,zone))
#小数
mean = 0.17234
print('%.2f%%' % (mean * 100))
```

```
www.haobin.com
17.23%
```

[14]
```
##一样
print("I am %s years old." % 22)
print("I am %r years old." % 22)

## 不一样,%r把单引号也输出了
text = "I am %d years old." % 22
```

```
print("I said:%s." % text)
print("I said:%r." % text)

#不一样，%r把对象也输出了
import datetime
d =datetime.date.today()
print("%s" % d)
print("%r" % d)
```

```
I am 22 years old.
I am 22 years old.
I said:I am 22 years old..
I said:'I am 22 years old.'.
2019-09-02
datetime.date(2019, 9, 2)
```

[15]
```
#print的其他使用
print('Hello')
print('Hello',end=' ')#修改不用换行符结尾
print('Cats','Dogs','mice')
print('Cats','Dogs','mice',sep=',')
```

```
Hello
Hello Cats Dogs mice
Cats,Dogs,mice
```

[57]
```
#格式符号为数字时前面可以加为数和补缺位如：%[0][总位数][.][小数位数]来设定要转换的样式
fValue = 8.123
print('%06.2f'%fValue) # 保留宽度为6的2位小数浮点型
```

```
008.12
```

**format的使用**

位置匹配

① 不带参数，即{}

② 带数字参数，可调换顺序，即{1}、{2}

③ 带关键字，即{a}、{to}

[60]
```
# 不带参数
print('{} {}'.format('hello','world'))    # 结果：hello world
# 带数字参数
```

```
print('{0} {1}'.format('hello','world'))    # 结果：hello world
# 参数顺序倒乱
print('{0} {1} {0}'.format('hello','world'))    # 结果：hello
world hello
# 带关键字参数
print('{a} {tom} {a}'.format(tom='hello',a='world'))    # 结果：
world hello world
```

```
hello world
hello world
hello world hello
world hello world
```

[61]
```
# 通过索引
coord = (3, 5)
print('X: {0[0]};  Y: {0[1]}'.format(coord))    # 结果：'X: 3;  Y:
5'
# 通过key键参数
a = {'a': 'test_a', 'b': 'test_b'}
print('X: {0[a]};  Y: {0[b]}'.format(a))    # 结果：'X: test_a;
Y: test_b'
```

```
X: 3;  Y: 5
X: test_a;  Y: test_b
```

我们还可以通过在格式化指示符后面添加一个冒号来进行精确格式化。

格式转换
'b' - 二进制。将数字以2为基数进行输出。
'c' - 字符。在打印之前将整数转换成对应的Unicode字符串。
'd' - 十进制整数。将数字以10为基数进行输出。
'o' - 八进制。将数字以8为基数进行输出。
'x' - 十六进制。将数字以16为基数进行输出，9以上的位数用小写字母。
'e' - 幂符号。用科学计数法打印数字。用'e'表示幂。
'g' - 一般格式。将数值以fixed-point格式输出。当数值特别大的时候，用幂形式打印。
'n' - 数字。当值为整数时和'd'相同，值为浮点数时和'g'相同。不同的是它会根据区域设置插入数字分隔符。
'%' - 百分数。将数值乘以100然后以fixed-point('f')格式打印，值后面会有一个百分号。

[59]
```
print('{:b}'.format(3))
print('{:c}'.format(20))
print('{:d}'.format(20))
print('{:o}'.format(20))
print('{:x}'.format(20))
print('{:e}'.format(20))
print('{:g}'.format(20.1))
print('{:f}'.format(20))
```

```
print('{:n}'.format(20))
print('{:%}'.format(20))
```

```
11
□
20
24
14
2.000000e+01
20.1
20.000000
20
2000.000000%
```

### 匿名（lambda）函数

Python支持一种被称为匿名的、或lambda函数。它仅由单条语句组成，该语句的结果就是返回值。它是通过lambda关键字定义的，这个关键字没有别的含义，仅仅是说“我们正在声明的是一个匿名函数”。lambda函数之所以会被称为匿名函数，与def声明的函数不同，原因之一就是这种函数对象本身是没有提供名称name属性。

```
[ ]  def short_function(x):
         return x * 2

     #等价

     equiv_anon = lambda x: x * 2
```

```
[ ]  def apply_to_list(some_list, f):
         return [f(x) for x in some_list]
     #虽然你可以直接编写[x *2for x in ints]，但是这里我们可以非常轻松地传入一个
     自定义运算给apply_to_list函数。
     ints = [4, 0, 1, 5, 6]
     apply_to_list(ints, lambda x: x * 2)
```

```
[ ]  def add_numbers(x, y):
         return x + y
     add_five = lambda y: add_numbers(5, y)
     #add_numbers的第二个参数称为“柯里化的”（curried）。这里没什么特别花哨的东
     西，因为我们其实就只是定义了一个可以调用现有函数的新函数而已。
```

# 四、数据结构和序列

## 4.1 元组

元组是一个固定长度，不可改变的Python序列对象。创建元组的最简单方式，是用逗号分隔一列值

[77]
```
tu=(1)
tu2=(1,)
tu3=(1,2,3,4)
tu4=1,2,3,4
type(tu4)
```

```
tuple
```

[78]
```
#用tuple可以将任意序列或迭代器转换成元组
tup = tuple('string')
tup
```

```
('s', 't', 'r', 'i', 'n', 'g')
```

[79]
```
#元组是不可变的对象，但如果元组中的某个对象是可变的，比如列表，可以进行修改
tup = tuple(['foo', [1, 2], True])
tup[1].append(3)
tup
```

```
('foo', [1, 2, 3], True)
```

[82]
```
#可以用加法和乘法串联
tup1 = (4, None, 'foo') + (6, 0) + ('bar',)
tup2 = ('foo', 'bar') * 5
tup1,tup2
```

```
((4, None, 'foo', 6, 0, 'bar'),
 ('foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar'))
```

[85]
```
#如果你想将元组赋值给类似元组的变量，Python会试图拆分等号右边的值
```

```
tup = (4, 5, 6)
a, b, c = tup
a,b,c
```

```
(4, 5, 6)
```

[86]
```
#简单的元组替换
a , b = 1, 2
b , a = a,b
a,b
```

```
(2, 1)
```

[89]
```
#Python最近新增了更多高级的元组拆分功能，允许从元组的开头“摘取”几个元素。
#它使用了特殊的语法*rest，这也用在函数签名中以抓取任意长度列表的位置参数
values = 1, 2, 3, 4, 5
a,b,*rest = values
a,b,rest
#rest的部分是想要舍弃的部分，rest的名字不重要。作为惯用写法，许多Python程序
员会将不需要的变量使用下划线，即*_
```

```
(1, 2, [3, 4, 5])
```

[90]
```
#count（也适用于列表），它可以统计某个值得出现频率
a = (1, 2, 2, 2, 3, 4, 2)
a.count(2)
```

```
4
```

## 4.2 列表

与元组对比，列表的长度可变、内容可以被修改。列表属于可变的数据类型，因此可以添执行加、删除，或者是搜索列表中的项目。列表可以嵌套

[108]
```
li = [1,2,3,4]
li2 = ['a','b','c','d']
li3 = [1,2,'a','b']
li,li2,li3
```

```
([1, 2, 3, 4], ['a', 'b', 'c', 'd'], [1, 2, 'a', 'b'])
```

[115]
```
a_list = [2, 3, 7, None]
a_list.append('dwarf')#append在列表末尾添加元素
a_list.insert(1, 'red')#insert可以在特定的位置插入元素
a_list
```

```
[2, 'red', 3, 7, None, 'dwarf']
```

[111]
```
a_list.pop(2)#pop移除并返回指定位置的元素
```

```
3
```

[118]
```
a_list.append('red')#再末尾加上'red'。这样就有两个'red'值
a_list.remove('red')#remove会先寻找第一个值并除去
a_list
```

```
[2, 3, 7, None, 'dwarf', 'red']
```

[119]
```
#extend方法可以追加多个元素
x = [4, None, 'foo']
x.extend([7, 8, (2, 3)])
x
```

```
[4, None, 'foo', 7, 8, (2, 3)]
```

[121]
```
#切片可以选取大多数序列类型的一部分，切片的基本形式是在方括号中使用。
#切片的起始元素是包括的，不包含结束元素。
seq = [7, 2, 3, 7, 5, 6, 0, 1]
seq[1:5],seq[:5]#start或stop都可以被省略，省略之后，分别默认序列的开头和
结尾
```

```
([2, 3, 7, 5], [7, 2, 3, 7, 5])
```

[77]
```
#遍历
```

```
lists = ["m1", 1900, "m2", 2000]
#1.使用iter()
for val in iter(lists):
    print(val)
#2.enumerate遍历方法
for i, val in enumerate(lists):
    print(i, val)
#3.索引遍历
for index in range(len(lists)):
    print(lists[index])
```

```
m1
1900
m2
2000
0 m1
1 1900
2 m2
3 2000
```

## 4.3 字典

字典可能是Python最为重要的数据结构。它更为常见的名字是哈希映射或关联数组。它是键值对的大小可变集合，键和值都是Python对象。字典使用花括号{}包括，键和值之间用冒号：分割，每对键-值”用逗号，分割 键必须唯一，即一个键不能对应多个值，不过多个键可以指向一个值。

[5]
```
#创建字典的方法之一是使用尖括号，用冒号分隔键和值：
empty_dict = {}
d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]}
d1
```

```
{'a': 'some value', 'b': [1, 2, 3, 4]}
```

[6]
```
d1[7] = 'an integer'#插入字典中的元素，7为key，'an integer'为value
d1
```

```
{7: 'an integer', 'a': 'some value', 'b': [1, 2, 3, 4]}
```

[7]
```
#del关键字或pop方法（返回值的同时删除键）删除值
d1[5] = 'some value'
print(d1)
```

```
d1['dummy'] = 'another value'
print(d1)
del d1[5]
print(d1)
```

```
{'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer', 5: 'some value'}
{'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer', 5: 'some value',
'dummy': 'another value'}
{'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer', 'dummy':
'another value'}
```

[8]
```
ret = d1.pop('dummy')
ret,d1
```

```
('another value', {7: 'an integer', 'a': 'some value', 'b': [1, 2, 3,
4]})
```

[9]
```
#keys和values是字典的键和值的迭代器方法。虽然键值对没有顺序，这两个方法可以用
相同的顺序输出键和值
list(d1.keys()),list(d1.values())
```

```
(['a', 'b', 7], ['some value', [1, 2, 3, 4], 'an integer'])
```

[12]
```
#keys()返回键,values()返回值和items()方法返回键值对
#返回结果并不是真正的列表，因此不能被append和修改，但可以使用for循环或者强制
转换
#字典转为列表
spam =  {'color':'red','age':42}
spam.values()
for i in spam.items():
    print(i)
#把返回结果换为列表
list(spam.keys())
```

```
('color', 'red')
('age', 42)
```

```
['color', 'age']
```

[76]
```
#字典的遍历
#1.遍历key值 在使用上，for key in a和 for key in a.keys():完全等价。
a={'a': '1', 'b': '2', 'c': '3'}
print("遍历key值:")
for key in a:
```

```
    print(key+':'+a[key])
#2.遍历value值
print("遍历value值:")
for value in a.values():
    print(value)
#3.遍历字典项
print("遍历字典项:")
for kv in a.items():
    print(kv)
#4.遍历字典健值
print("遍历字典健值:")
for key,value in a.items():
    print(key+':'+value)
```

```
遍历key值:
a:1
b:2
c:3
遍历value值:
1
2
3
遍历字典项:
('a', '1')
('b', '2')
('c', '3')
遍历字典健值:
a:1
b:2
c:3
```

[10]

```
#update方法可以将一个字典与另一个融合
d1.update({'b' : 'foo', 'c' : 12})
d1
```

```
{7: 'an integer', 'a': 'some value', 'b': 'foo', 'c': 12}
```

[13]

```
#字典总是明确地记录键和值之间的关联关系，但获取字典的元素时，获取顺序是不可预测的。
#要以特定的顺序返回元素，一种办法是在for 循环中对返回的键进行排序。
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
}
for name in sorted(favorite_languages.keys()):
    print(name.title() + ", thank you for taking the poll.")
```

```
Edward, thank you for taking the poll.
Jen, thank you for taking the poll.
Phil, thank you for taking the poll.
Sarah, thank you for taking the poll.
```

[15]
```
#setdefault方法,比defaultdict方便,为字典中某个键设置一个默认值，当该键没有任何值时使用它。
#空字典可以直接增加变量，但是如果在Python中如果访问字典中不存在的键，会引发KeyError异常。
#由于键值是空的，所以不能直接访问
words = ['apple', 'bat', 'bar', 'atom', 'book']
by_letter = {}
for word in words:
    letter = word[0]
    by_letter.setdefault(letter, []).append(word)
by_letter
```

```
{'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

[ ]
```
#collections模块有一个很有用的类，defaultdict，它可以进一步简化上面。传递类型或函数以生成每个位置的默认值
##defaultdict就是一个本身带有默认值的dict
#defaultdict类的初始化函数受一个类型作为参数，当接所访问的键不存在的时候，可以实例化一个值作为默认值
from collections import defaultdict
by_letter = defaultdict(list)
for word in words:
    by_letter[word[0]].append(word)
```

[66]
```
# 字典转成dataframe
#1.直接使用pd.DataFrame()。
dict = {'a':'apple','b':'banana'}
import pandas as pd
df = pd.DataFrame([dict])
df
```

| | a | b |
|---|---|---|
| **0** | apple | banana |

[48]
```
#2.使用DataFrame的from_dict()方法。
df = pd.DataFrame.from_dict(dict,orient='index')
df = df.reset_index().rename(columns = {'index':'id'})
df
```

| | id | 0 |
|---|---|---|
| **0** | a | apple |
| **1** | b | banana |

[41]
```
#3.将字典转变为Series，再转为DataFrame。
df = pd.DataFrame(pd.Series(dict), columns=['fruits'])
df = df.reset_index().rename(columns={'index':'id'})
df
```

| | id | fruits |
|---|---|---|
| **0** | a | apple |
| **1** | b | banana |

[42]
```
#4.reset_index().rename()方法，将作为index的keys变为DataFrame中的一列
df = pd.DataFrame([dict]).T
df = df.reset_index().rename(columns={'index':'id'})
df
```

| | id | 0 |
|---|---|---|
| **0** | a | apple |
| **1** | b | banana |

[73]
```
#5.多个值，通过创建子字典
test_0 = {"id":[1,1,2,3,3,4,5,5],"price":
[5,6,8,3,4,6,9,5],"amount":[1,1,2,1,1,1,2,1],"status":
['sale','sale','no','no','sale','no','sale','no']}
index = {'id','price','amount'}   #创建存储想要的key的集合
test_1 = {key:value for key,value in test_0.items()if key in
index}   #建立子字典
pd.DataFrame(test_1)
```

| | amount | id | price |
|---|---|---|---|
| **0** | 1 | 1 | 5 |
| **1** | 1 | 1 | 6 |
| **2** | 2 | 2 | 8 |

| | amount | id | price |
|---|---|---|---|
| **3** | 1 | 3 | 3 |
| **4** | 1 | 3 | 4 |
| **5** | 1 | 4 | 6 |
| **6** | 2 | 5 | 9 |
| **7** | 1 | 5 | 5 |

[64]
```
#根据key 找value
d = {'a' : [1, 2, 3], 'b' : [4, 5],'c':[1,2,3,4] }
key = ['a','c']
value=[]
for k in key:
    value.append(d[k])
value
```

```
[[1, 2, 3], [1, 2, 3, 4]]
```

[54]
```
#根据value找key
d = { 'a' : [1, 2, 3], 'b' : [4, 5],'c':[1,2,3] }
value = [1,3]
t=[k for k,v in d.items() if v == [1,2,3]]
print(t)
```

```
['a', 'c']
```

## 4.4 集合

集合是无序的不可重复的元素的集合。你可以把它当做字典，但是只有键没有值。

[16]
```
#两种方式创建集合：通过set函数或使用尖括号set语句
set([2, 2, 2, 1, 3, 3]),{2, 2, 2, 1, 3, 3}
```

```
({1, 2, 3}, {1, 2, 3})
```

集合支持合并、交集、差分和对称差等数学集合运算

| 函数 | 替代语法 | 说明 |
| --- | --- | --- |
| a.union(b) | a\|b | 集合a和b中的所有不重复元素 |
| a.update(b) | al=b | 设定集合a中的元素为a与b的合并 |
| a.intersection(b) | a&b | a和b中交叉的元素 |
| a.intersection_update(b) | a&=b | 设定集合a中的元素为a与b的交叉 |
| a.difference(b) | a-b | 存在于a但不存在于b的元素 |
| a.difference_update(b) | a-=b | 设定集合a中的元素为a与b的差 |
| a.symmetric_difference(b) | a^b | 只在a或只在b的元素 |
| a.symmetric_difference_update(b) | a^=b | 集合a中的元素为只在a或只在b的元素 |

[18]
```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7, 8}
a.union(b),a.intersection(b)
```

```
({1, 2, 3, 4, 5, 6, 7, 8}, {3, 4, 5})
```

[19]
```
c = a.copy()
c |= b
c
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

[20]
```
d = a.copy()
d &= b
d
```

```
{3, 4, 5}
```

## 4.5常用序列函数

### enumerate函数

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。 enumerate(sequence,

```
[start=0])
sequence   一个序列、迭代器或其他支持迭代对象。
start   下标起始位置。
```

```
[124]  seasons = ['Spring','Summer','Fall','Winter']
       for i,ele in enumerate(seasons):
           print(i,ele)
```

```
0 Spring
1 Summer
2 Fall
3 Winter
```

## sorted函数

sorted函数可以从任意序列的元素返回一个新的排好序的列表

```
[126]  sorted([7, 1, 2, 6, 0, 3, 2]),sorted('horse race')
```

```
([0, 1, 2, 2, 3, 6, 7], [' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r', 'r',
's'])
```

## zip函数

zip可以将多个列表、元组或其它序列成对组合成一个元组列表.zip可以处理任意多的序列，元素的个数取决于最短的序列

```
[1]  seq1 = ['foo', 'bar', 'baz']

     seq2 = ['one', 'two', 'three']

     zipped = zip(seq1, seq2)

     list(zipped)
```

```
[('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
```

```
[2]  seq3 = [False, True]
```

```
list(zip(seq1, seq2, seq3))
```

```
[('foo', 'one', False), ('bar', 'two', True)]
```

[3]
```
#zip的常见用法之一是同时迭代多个序列，可能结合enumerate使用
for i, (a, b) in enumerate(zip(seq1, seq2)):
    print('{0}: {1}, {2}'.format(i, a, b))
```

```
0: foo, one
1: bar, two
2: baz, three
```

## reversed函数

[4]
```
list(reversed(range(10)))#reversed可以从后向前迭代一个序列
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

[23]
```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more
help.

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass
```

```
x and y      如果 x 为 False 、空、0，返 回 x，否则返回 y
x or y       如果 x 为 False、 空、0，返回  y，否则返回x
not  x     如果 x 为 False、 空、0，返回 True，否则返回False
```

[28]
```
a = '1'
b = 1
a and b,a or y,not a
```

```
(1, '1', False)
```

[ ]