Python 笔记

1. 基本数据结构

1) 列表(List)，用“[ ]” 表示，特点：可重复，类型可不同

```
lista=['a','b',1,2]

# 添加元素：
lista.append(3)         #['a','b',1,2,3]
# 插入元素：
lista.insert(2,'c')     #['a','b',1,2,3,'c']
# 连接链表：
lista.extend([7,8])     #['a','b',1,2,3,'c',7,8]
# 获取长度：
len(lista)              #8
# 获取索引：
lista.index(2)          #3
lista.index('e')        #引发异常
# 删除元素：
lista.remove(3)         #['a','b',1,2,'c',7,8]
# 删除最后元素
lista.pop()             #['a','b',1,2,'c',7]
# 遍历元素：
for item in lista:
    print(item)
```

list的sort函数和内置sorted函数

```
list.sort(key=None,reverse=False)  在原来的序列上操作，不会返回新序列
cmp:可选参数，如果指定了该参数，会使用该参数的方法排序
key:进行比较的元素，只有一个参数，具体函数的参数取决于可迭代对象
```

```
reverse：排序规则，reverse=True 降序，reverse=False 升序(默认)

sorted(iterable,key=None,reverse=None) 会返回一个新的序列

a=[2,1,4,9,6]
a.sort()
print(a)    # [2, 1, 4, 9, 6] [1, 2, 4, 6, 9]

c=[2,1,4,9,6]
d=sorted(c)
print(c,d)   # [2, 1, 4, 9, 6] [1, 2, 4, 6, 9]

L=[('a',3),('d',2),('c',1)]
a=sorted(L,key=lambda x:x[0])
print(a)    # [('a', 3), ('c', 1), ('d', 2)]
```

2) 元组(Tuple)，用“( )”表示，特点：只读，不能修改

```
tuple1=(1,2,3,4,'a')

# 访问元素：
# tuple1[2]           #3
# tuple1[-1]          #'a'
# tuple1[1:3]         # (2,3)
# 搜索元素：
print(1 in tuple1)  #True
# 遍历元素：
for item in tuple1:
    print(item)
```

3) 字典(Dictory)，用“{ }”表示，特点：键和值之间一对一关系，以无序的方式存储

```
dict1={'name':'yeoman','age':24,'sex':'Male'}

# 覆盖元素：
dict1['name']='yuanm'
# 删除元素：
del dict1['sex']
# 清空元素：
dict1.clear()
# 返回字典中(key,value)元组列表
print(dict1.items())
# 返回字典中的键的列表
print(dict1.keys())
# 范湖字典中值的列表
print(dict1.values())
# 若key存在，则删除并返回dict[key]。若不存在返回default值，若没给出default，则会引发异常
dict1.pop(key[,default])
```

4) 集合(Set)，用“{ }”，特点：无序不重复元素(支持联合，交，差，对称差操作，不支持索引和分片)

```
lst = [1, 1, 0]

# 创建集合
lst_set = set(lst)  # {0,1}
# 遍历元素
for item in lst_set:
    print(item)
# 添加元素
lst_set.add(10)  # {0,1,10}
# 删除元素
lst_set.discard(6)  # 不存在不引发异常
lst_set.remove(6)  # 不存在引发异常
lst_set.pop()  # 随机删除一个元素
```

2. enumerate函数

将一个可遍历对象组合为一个索引序列，同时列出索引和索引下标。
enumerate(sequence,[start=0])

```
lst=[10,50,85,-12]

for i,value in enumerate(lst,1):
    print(i,value)
>>
1 10
2 50
3 85
4 -12

list(enumerate(lst,1))     # [(1, 10), (2, 50), (3, 85), (4, -12)]
```

3. isinstance(object, classinfo) 和 type(object)函数

判断一个对象是否是已知类型

```
例子：
a=2
isinstance(a,int)       # return True
type(a)==int            # return True
```

区别 $\begin{cases} isinstance：不会认为子类是父类的一种类型 \\ type：会认为子类是父类的一种类型 \end{cases}$

4. divmod()函数

把除数和余数运算结果结合起来，返回一个包含商和余数的元组(a//b, a%b)

```
res = divmod(7, 2)
print(res)
>>(3, 1)

res = divmod(8, 2)
print(res)
>>(4, 0)
```

5. any()和all()函数

any：用于判断给定的可迭代参数是否全部为False，如果有一个为True，则返回True。
all：----------True，----------False，------False 。
元素 0,'',False 全部算为False

```
# -----any------
print(any(['a', 'b', 'c', 'd']))
>>True

print(any(['a', 'b', '', 'd']))
>>True

print(any([0, '', False]))
>>False

print(any(['a', 'b', 'c', 'd']))
>>True

print(any([]))
>>False

print(any(()))
>>False

# -----all------
print(all(['a', 'b', 'c', 'd']))
>>True

print(all(['a', 'b', '', 'd']))
>>False

print(all([0, '', False]))
>>False

print(any(['a', 'b', 'c', 'd']))
>>True

print(all([]))
>>True

print(all(()))
>>True
```

6. Python魔术方法

Python可以让一些函数不需要被显示的调用的时候被执行，这种方法以__开头和结尾

```
__init__  Python的构造函数
__del__   Python的析构函数
__getitem__(self,key) 返回键对应的值
__setitem__(self,key,value)  设定给定键的值
__delitem__(self,key) 删除给定键对应的元素
__len__() 返回元素数量
__call__ 如果在类中实现了__call__方法，那么实例对象将成为一个可调用对象，即将累的实例表现的像函数一样，判断
一个对象是否是可调用对象们可以用callable
```

```
__call__函数例子：

class Entity:
    def __init__(self, size, x, y):
        self.x = x
        self.y = y
        self.size = size

    def __call__(self, x, y):
        self.x, self.y = x, y


e = Entity(1, 2, 3)      # 创建实例
e(4, 5)                  # 实例可以像函数那样执行，并传入x,y值修改对象的x,y
```

7. Python中的参数传递

1) def F(x,y):　　#常见的
2) def F(arg1,arg2=value2): #提供默认值
3) def F(*arg1):　#不确定参数个数，在函数内部都被存在**以形参名为标识符的元组**中， F(1,2,3) -> arg1=(1,2,3)
4) def F(**arg1): #加两个星号，参数在内部被存放在**以形参名为标识符的字典**中，F(a=1,b=2) -> arg1={'a':1,'b':2}

8. 获取当前目录

```
import os
print(os.getcwd())
```

9. 可变和不可变对象 && 深拷贝和浅拷贝

```
可变对象和不可变对象区别在于：对象本身是否可变
Python内置的类型中
    可变的对象有：list，dict，set
    不可变的对象有：int, float, string, tuple

不可变对象的  优点是：减少重复值对内存的占用
              缺点是：要修改值，如果内存中没有会重新开辟内存，并把新地址和变量名绑定
Python函数虽然是引用传递，但 对于可变对象：其内容被修改
                             对于不可变对象：其内容不能被修改
```

深拷贝和浅拷贝
直接赋值： 对象的引用
浅拷贝： 拷贝父对象，不拷贝内部的子对象 b=a.copy() a变b也变
深拷贝： copy模块的deepcopy方法，完全拷贝父对象和子对象 b=copy.deepcopy(a) a与b无关，深拷贝

copy() 进行对象浅拷贝，它复制了对象，但对于对象的元素依然使用原始引用
copy.deepcopy() 深拷贝，拷贝对象和元素。可变对象：创建新的。不可变对象：指向相同的位置

10. 迭代器、生成器、装饰器

迭代器：一个带状态的对象，调用next()方法返回容器中的下一个值，任何是实现了__iter__和__next__方法的对象
都是迭代器
法一：使用__next__()

```
x = [1, 2, 3, 4, 5]
y = iter(x)
print(y.__next__())
```

法二：使用for循环

```
x = [1, 2, 3, 4, 5]
for i in x:
    print(i)
```

生成器：一个包含yield的函数，当生成函数被调用时会返回一个迭代器，每次请求一个值，就会执行生成器中的代码，
直到遇到一个yield表达式或return语句。yield表示要生成一个值，return表示要停止生成器
法一：通过yield

```
def generator(low, high):
    while low < high:
        yield low
        low += 1
for i in generator(1, 10):
    print(i, end=' ')
```

法二：通过列表生成器

```
a=[i*2 for i in range(1,10)]
```

装饰器：本质是一个函数，用@语法糖让已有的函数不做任何改动的情况下，增加新的功能

```
import logging
def use_log(func):
    def wrapper(*args, **kargs):
        logging.warning('%s is running' % func.__name__)
        return func(*args, **kargs)

    return wrapper
@use_log
def bar():
    print("I am bar")
@use_log
def haha():
    print("I am haha")

bar()
haha()
```

11. with用法

```
with适用于对资源进行访问的场合，确保使用中发生异常会执行必要的清理工作，释放资源
with工作原理是：  with后面语句执行后，返回对象-enter-()方法被调用，将返回值赋值给as后面的变量
                  代码块执行完毕后，调用返回对象的-exit-()方法
```

12. __future__

```
首行添加
    from __future__ import *  # 将新版本特性引入到当前版本中
    from __future__ import print_function/division/absolute_import
```

13. 操作txt文档

1) 读取txt文档

```
# 方法一：readline()
f = open('./result.txt')
line = f.readline()
while line:
    print(line)
    line = f.readline()
f.close()

# 方法二：
f = open('./result.txt')
for line2 in open('./result.txt'):
    print(line2)

# 方法三：
f = open('./result.txt', 'r')
lines = f.readlines()
for line3 in lines:
    print(line3)
```

2) 写入txt文档

```
# python保存numpy数据
import numpy as np
data = np.array([1, 2, 3])
np.savetxt("result123.txt", data)

# 保存list数据
file = open('data.txt', 'w')
list_data = [1, 2, 3]
file.write(str(list_data))
file.close()
```

14. XML文件

xml中一般有一些分类：根节点，子节点，属性，内容，标签
特点：节点可以嵌套节点，子节点可以继续嵌套

```
利用ElementTree.parse解析xml文件
import ElementTree as ET

# 解析xml文件
tree=ET.parse("xx.xml")     # 解析xml文件
root=tree.getroot()         # 获取根节点
print(root.tag)             # 顶层标签
for child in root:      # 遍历第二层
    print(child.tag,child.attrib,child.text)
    for subchild in child:  # 遍历第三层
        print(subchild.tag,subchild.attrib,subchild.text)

for node in root.iter('year'):  #遍历所有year节点
    print(node.tag,node.text)

# 写入xml文件
tree=ET.ElementTree(root)
tree.write('xx.xml',encoding='utf-
8',xml_declaration=True,short_empty_element=False)
```

15. json文件操作

```
将字典保存到本地
import json
with open('data.json','w') as f:
    json.dump(data,f)

从本地json文件读取
with open('data.json','r') as f:
    data=json.load(f)
```