Python第一阶段

1. Python的安装和简介

1.1 Python基础常识

1.1.1 简介

Python是一门面向对象的解释性计算机设计语言

1.1.2 Python语言特色

1. python是一门解释性语言
    - 解释性语言：在系统中运行时需要使用解释器（如Java、php等）
    - 编译性语言：在系统中运行时不需要使用解释器（如C，C++）
2. 弱类型语言
    - 弱类型语言：变量在使用之前不需要提前声明变量的类型就可以直接使用
    - 强类型语言：变量在使用之前需要提前声明变量的类型
3. 面向对象的语言
    - Python支持面向对象的编程，也在一定程度上支持面向过程和面向函数
4. 胶水语言
    - Python的底层和扩张库都是由C语言写成，可以很好支持C和C++

1.2.3 Python语言的优点

1. 简单
2. 易学
3. 速度快（这里指python的开发速度相对较快，python的运行速度相对一般）
4. 免费，开源
5. 高层语言（这里的高低指的是距离硬件的远近，python距离硬件相对比较远）
6. 可移植性（python在linux，windows，mac os都可以使用）
7. 可扩张性
8. 可嵌入性

1.2 Python安装及版本检测

1.2.1 Python windows上安装需要注意事项

勾选add python to paith 若没有勾选就需要手动去配置环境变量

1.2.2 Python版本检测

1. 方法一：在cmd中直接输入python，第一行会显示python的当前版本
2. 方法二：在cmd中输入python -V
3. 方法三：打开idle

2. Python基本语法

2.1 注释、语句分类、关键字

2.1.1 注释

- 定义：即注解，解释。分为单行注释和多行注释
- 作用：

```
1.给代码做出标注，进行解释和说明。方便别人阅读和理解代码
2.在debug的时候，可以通过注解来一行一行查找问题
```

2.1.1.1 单行注释

以#号开始，#后面的内容就是注解的内容。计算机不会去阅读#后面的内容

2.1.1.2 多行注释

以''' 或者""" 将注释内容包裹起来

2.1.1.3 注释的选择原则

单行注释 # 里面可以使用多行注释 ''' 或者 """

多行注释''' 或者 """ 里面可以使用单行注释#

多行注释中可以使用另一种多行注释。如：''' 中可以使用""" 在"""中可以使用'''

2.1.2 Python 语句分类

Python语句分为 单行语句 和 代码组（代码块）

单行语句：一行python代码

代码组：特定的语言结构，标志是：和缩进（如if ，while等）

2.1.3 关键字

- 定义：关键字指系统已经使用的具有特殊功能的保留标识符
- 查看系统保留关键字的方法：

```
import keyword
print(keyword.kwlist)
```

3. Python变量及数据类型

3.1 变量

- 变量的定义：变量就是可以变化的量（在Python中更像是把变量的值贴到值上面，之后使用这个值就直接用贴在它上面的名字即可）
- 变量赋值：（三种方式）

```
方法一：（基本格式）
    变量 = 值
方法二：（给多个变量赋相同的值）
    变量1 = 变量2 = 变量3 ... = 值
方法三：（给多个变量赋不同的值）
    变量1，变量2，变量3... = 值1，值2，值3...
```

- 获取变量的类型：（两种方法）

```
1. type（）
    print（type（变量））
2. isinstance（）-------> isinstance（查看的变量，类型）  返回的值是bool True or False
    print（isinstance（4，int））
```

- 获取变量在内存中的id：

```
id（）
    print（id（变量））
```

- 更改变量的值：（对变量重新赋值即可）

```
val = 1
val = 2
print(val)
```

- 变量的命名规则：

```
1. 使用英文，禁止使用中文
2. 可以使用数字，但是不能用数字开头
3. 特殊符号只能使用下划线_
4. 区分大小写
5. 命名必须要有意义
6. 避免和系统保留的关键字冲突
```

3.2 数据类型

1. Number　整型　　(包含 ：int、float、bool、complex)
2. String　字符串
3. List　　列表
4. Tuple　 元组
5. Set　　集合
6. Dict　　字典

ps：

```
* Number中包含： int、float、bool、complex
* 容器数据类型： String、List、Tuple、Set、Dict
* 有序数据类型： String、List、Tuple
* 无序数据类型： Set、Dict
```

3.2.1 Number类型

3.2.1.1 Number类型：int

- int整型的声明方式：（十进制、二进制、八进制、十六进制）

```
1. 十进制（0-9）
    变量 = 十进制数字
2. 二进制（0-1）
    变量 = 0b二进制数字
3. 八进制（0-7）
    变量 = 0o八进制数字
4. 十六进制（0-9a-f）
    变量 = 0x十六进制数字
```

3.2.1.2 Number类型：float

- float浮点型声明方式：（小数和科学记数法）

```
1. 小数：
    变量 = 带小数点的数字
    num = 3.14
2. 科学计数法    e相当于10
    变量 = 314e-2
```

3.2.1.3 Number类型：bool

- bool类型只有两个值：True 和 False （True和False的首字母都要大写！）
- True：表示肯定的，确定的答案
- False：表示否定的答案

3.2.1.4 Number类型：complex

- 复数定义：由实数部分和虚数部分组成，实数部分是实际存在的数字，虚数部分是现实世界中不存在的数字。在计算机中虚数的单位为j。
- 声明一个复数的方法：

```
方法一：
    变量 = a + bj
方法二：
    变量 = complex（实数 + 虚数）            实数和虚数都只要填入纯数字即可，虚数部分不需要填入单位：j
    如：
    complex1 = complex（3，4）    ==     complex1 = 3 + 4j
```

3.2.2 String 字符串类型

- String在所有语言中都是使用最多的数据类型
- String类型声明方式：（三种）

```
方法一：单引号 ''
    变量 = '字符串内容'
方法二：双引号 " "
    变量 = "字符串内容"
方法三：三引号 ''' ''' 或者 """  """
    变量 = '''字符串内容'''  或者   变量 = """字符串内容"""
```

- String声明方式选择原则：

```
1. 单引号中可以使用双引号
2. 双引号中可以使用单引号
3. 三引号中可以使用单双引号
```

- 转义字符：\

```
1. 转义单引号：\'   （使单引号失去原有的特殊意义）
    str1 = '古人说：\'nnb\''
2. 转义双引号：\"   （使双引号失去原有的特殊意义）
    str1 = "古人说，\"nnb\""
3. 换行操作符：\n
    str1 = "人之初，\n性本善。"
4. 缩进操作符：\t
    str1 = "\t人之初，性本善"
5. 对\自身进行转义：\\
    str1 = '在字符串中输出\\'
6. 续行符：\
    str1 = '在单行语句\'
           '过长时使用，自动续行'
7. 原字符串：在字符串前面加上 r 或者 R ，转义字符串中所有的字符
    str1 = r'你好，\n再见\t！'
```

3.2.3 List 列表类型

- 列表类型的标志：[ ] 中括号
- 列表的声明方式：

```
变量 = [值1，值2，值3......]
```

- 创建一个空列表：

```
方法一：    变量 = []
方法二：    变量 = list()
```

- 列表索引值

```
正向索引： 0   1   2   3   4
list1 = [100,200,300,400,500]
反向索引：-5  -4  -3  -2  -1
```

- 通过索引访问和修改列表中特定的值

```
列表[index]
列表[index] = 新的值      可以通过下标来修改列表中的值
```

3.2.4 Tuple 元组类型

- 元组类型的标志：，逗号
- 元组类型的声明方式：

```
方法一：
    变量 = (值1，值2，值3....)
方法二：
    变量 = 值1，值2，值3....
```

- 元组索引值

```
正向索引    0   1   2   3   4
tuple1 = (100,200,300,400,500)
反向索引   -5  -4  -3  -2  -1
```

- 通过索引查看元组中的某个值

```
元组[index]
元组中的值不能随意修改
```

3.2.5 Set 集合类型

- 集合的标志：无
- 集合是无序容器，无法通过索引来访问其中的某个值
- 集合中所有的数据都是唯一的，出现重复的会只保留唯一的其他删除
- 集合的声明方式：

```
    变量 = {100,200,300,400,500}
```

- 创建空集合：

```
    变量 = set( )
```

3.2.6 Dict 字典类型

- 字典的标志：{}
- 字典是无序数据，无法通过索引来访问其中的某个值
- 字典的声明方式：

```
    变量 = {键：值，键：值，键：值...}
```

- 访问和修改字典中的值 ：

```
字典是无序容器，不能使用索引来访问字典中的值。但是可以通过字典特有的键来访问字典中的值：
    dict[键]
修改字典中的值：
    dict[键] = 新的值
```

- 创建一个空字典：

```
变量 = dict()
    dict1 = dict()
变量 = {}
    dict1 = {}
```

3.2.7 查看数据类型的方法

3.2.7.1 type ( )

正常工作的时候不能用，因为效率太低了。工作原理是把目标数据和所有数据类型一一匹配询问，找到同目标数据类型相同的类型。

```
result = type( 变量名 )
print( result )
```

3.2.7.2 isinstance ( )

工作效率比较高，把目标变量和指定的类型做比对，如果目标变量和指定类型为相同类型，则返回True；否则返回False

```
* 方法一，使用isinstance查看单一类型
    isinstance(变量，类型)
    如：
    var = 123
    result = isinstance(var,int)
    print(result)

* 方法二，使用isinstance查看目标变量是否属于两个类型中
    isinstance（变量，(类型一，类型二)）
    如：
    var = 123
    result = isinstance（var，（int，str））
    print（result）
```

3.3 数据类型转换

3.3.1 自动数据类型转换

1. 自动数据类型转换是系统自发的，不需要人工干预的
2. 自动数据类型转换会从精确度较小的数据类型向精确度较高的数据类型进行转换
3. 自动数据类型转换多发生在运算或者判断的时候

```
如：
    num1 = 25
    num2 = 25.0
    print(num1 + num2)
又如：
    if 1:
        print('1会自动转化成True')
```

3.3.2 强制数据类型转换

3.3.2.1 int（ ）

将其他数据类型转换成整型

```
整型：      无需转换
浮点数：    去掉小数部分，只保留整数部分
布尔值：    True 转换成 1，False 转换成 0
复数：      复数无法转换成整型
字符串：    只有纯整数字符串才能转换成int。
列表，元组，集合，字典都不能转换成整型
```

3.3.2.2 float（ ）

将其他数据类型转换成浮点数

```
整型：            在整数后面加上.0
浮点数：     无需转换
布尔值： True 转换成 1.0     False 转换成 0.0
复数：            复数无法转换成浮点数
字符串： 只有纯整数字符串和纯浮点数字符串才能转换成浮点数
列表，元组，集合，字典都不能转换成浮点数
```

3.3.2.3 bool（ ）

其他类型中，相当于bool中的False的情况

```
整型：             0
浮点型：          0.0
bool：      False
复数：            0j  或者  0+0j
字符串： ''(空字符串)
列表：            [] 或者 list()
元组：            () 或者 tuple()
集合：            set()
字典：            {} 或者 dict()
```

3.3.2.4 complex()

其他类型转换成复数complex（ ）

```
整型：            整型+0j
浮点数： 浮点数+0j
布尔值： True：1+0j    False：0j 或者 0+0j
复数：            无需转换
字符串： 字符串中只有纯整数，纯浮点数或者纯复数才能转换成复数
列表、元组、集合、字典都不能转换成复数
```

3.3.2.5 str（ ）

其他类型转换成字符串

```
所有类型都可以转换成字符串，系统会自动给其他类型加上''让它变成字符串
如果要打印出带引号的字符串，可以使用命令repr()
        如：
        list1 = [1,2,3]
        list1 = str(list1)
        print(repr（list1）)
```

3.3.2.6 list（ ）

其他类型转换成列表

整型、浮点型、复数、布尔值都不能转换成列表
字符串：字符串转换成列表，把字符串中的每个值转换成列表中的每一个值，顺序保持不变
列表： 无需转换
元组： 把元组中的每个值转换成列表中的值，顺序保持变
集合： 把集合转换成列表中的每一个值，顺序随机
字典： 把字典中的键转换成列表中的值，顺序随机 **只把字典中的键进行转换**

3.3.2.7 tuple（ ）

其他类型转换成元组

整型、浮点数、复数、布尔值都不能转换成元组
字符串：字符串中的每个字符转换成元组中的一个值，顺序不变
列表： 列表中的每一个值转换成元组中的值，顺序不变
元组： 无需转换
集合： 集合中的每个值转换成元组中的每个值，顺序随机
字典： 字典中的每个键转换成元组中的一个值，顺序随机 **只把字典中的键进行转换**

3.3.2.8 set（ ）

其他类型转换成集合

整型、浮点型、复数、布尔值都不能转换成集合
字符串： 字符串中的每个字符转换成集合中的一个值，顺序随机
列表： 列表中的每个值转换成集合中的一个值，顺序随机
元组： 元组中的每个值转换成集合中的一个值，顺序随机
集合： 无需转换
字典： 字典中的每个键转换成集合中的一个值，顺序随机

3.3.2.9 dict（ ）

其他类型转换成字典

```
* 整型、浮点型、复数、布尔值都不能转换成字典
* 字符串： 不能转换成字典
* 列表： 只有特定的列表格式可以转换成字典
    1. 二级列表（且第二级容器的长度一样）
    list1 = [[键，值],[键，值],[键，值],[键，值],[键，值]]     or      list1 = [(键，值),(键，值),
(键，值),(键，值)]

    2. 一级列表：只含有两个字符的一级列表
    list1 = ['键值','键值','键值','键值']

* 元组： 只有特定的元组格式才可以转换成字典
    1. 二级元组（且第二级容器的长度一样）

    tuple1 = ((键，值),(键，值),(键，值),(键，值))
```

```
    2. 一级元组：只含有两个字符的一级元组
    tuple1 = （'键值','键值','键值','键值'）

* 集合： 只有特定的集合格式才可以转换成字典
    1. 二级集合（且第二级容器的长度一样）
    set1 = {(键，值),(键，值),(jia),()}
```

4. Python运算符

1. 算术运算符
2. 比较运算符（关系运算符）
3. 逻辑运算符
4. 位运算符
5. 赋值运算符
6. 成员运算符
7. 身份运算符

4.1 算术运算符

```
**              幂运算（最高优先级）
*               乘法运算
/               除法运算
//              取商运算（地板除）
%               取余运算
+               加法运算
-               减法运算
```

4.2 比较运算符（关系运算符）

比较运算符运算结果为布尔值： True 、False

```
<=              小于等于
>=              大于等于
<               小于
>               大于
==              等于
!=              不等于
```

4.3 赋值运算符

变量 = 变量 操作符 值 ----------> 变量 操作符= 值

```
**=             幂运算赋值
*=              乘法赋值
/=              除法赋值
//=             取商赋值
%=              取余赋值
+=              加法赋值
-=              减法赋值
=               普通赋值
```

4.4 逻辑运算

逻辑运算是布尔值之间的运算

1. not 逻辑非运算 （ 逻辑运算中优先级最高 ）

```
真变假，假变真
not True == False
not False == True
```

2. and 逻辑与运算

```
有假则假
只有当 and 左右的两个条件都满足的时候 and 的结果才是为真
True and False == False
True and True == True
False and False == False
```

3. or 逻辑或运算

```
有真则真
当 or 左右两个条件有一个满足，则 or 的结果就是为真
True or False == True
False or False == False
True or True == True
```

4. xor 逻辑异或运算

```
不同为真，相同为假
在python中不支持异或操作。
True xor False == True
True xor True == False
False xor False == False
```

4.5 位运算符

位运算就是在二进制基础上进行的逻辑运算

```
&               按位与运算
|               按位或运算
~               按位反转
^               按位异或运算
<<              左移                    << 1 左移1位，相当于*2
>>              右移                    >> 1 右移1位，相当于//2
```

4.6 成员运算符

检测变量是否在容器类数据中

```
in ：
        变量 in 容器类数据（string,list,tuple,set,dict），检测字典的时候，只检测字典的键
        检测变量是不是在容器中

not in ：
        变量 not in 容器类数据
        检测变量是不是不在容器中
```

4.7 身份运算符

判断两个变量在内存中的地址是否相同。

```
is：
        检测两个变量是否是同一个值

is not：
        检测两个变量是否不是同一个值
```

```
当数值相同的时候，哪些数据类型的id会相同
1. 整数：          -5以上的整数
2. 浮点数： 0以上的浮点数
3. 布尔值： 永远相同
4. 复数：          实数部分为0，虚数部分在0j以上
5. 字符串： 永远相同
```

5.流程控制

1. 流程:

   > 做事情的顺序就是流程,计算机中的流程就是代码执行的顺序.默认是从上到下

2. 流程分类:

   1. 顺序结构
   2. 分支结构(单向分支,双向分支,多向分支,巢状结构)
   3. 循环结构(while循环 死循环 for...in循环)

5.1 顺序结构

python默认的代码执行顺序,默认从上到下执行代码

5.2 分支结构

5.2.1 单向分支

基本格式:

```
if 条件判断:
        条件为真时,执行语句
        ...
```

如:

```
num = 10
if num > 9:
        print(num)
```

5.2.2 双向分支

基本格式:

```
if 条件判断:
        条件为真的时候执行代码
        ...
else:
        条件为假的时候执行代码
        ...
```

如:

```
num = 10
if num > 10:
        print(num)
else:
        print(num,'比10小')
```

5.2.3 多向分支

基本格式:

```
if 条件1判断:
        条件1为真的时候执行代码
        ...
elif 条件2判断:
        条件2为真的时候执行代码
        ...
elif 条件3判断:
        条件3为真的时候执行代码
        ...
else:
        条件1,条件2,条件3都不满足执行代码
        ...
```

如:

```
day = 1
if day == 1:
        print('黄焖鸡')
elif day == 2:
        print('面条')
elif day == 3:
        print('快餐')
elif day == 4:
        print('汉堡')
else:
        print('不吃了')
```

5.2.4 巢状分支

基本结构:

```
if 条件1判断:
        条件1满足执行代码
        if 条件2判断:
                条件2满足执行代码
                if 条件3判断:
                        条件1,条件2,条件3都满足执行代码
                else:
                        条件3不满足执行代码
        else:
                条件2不满足执行代码
else:
        条件1不满足执行代码
```

如:

```
schooldoor = True
buildingdoor = True
```

```
classdoor = True


if schooldoor == True:
        print('校门开了,走到教学楼')
        if buildingdoor == True:
                print('教学楼门开了,走到教室')
                if classdoor == True:
                        print('教室门开了,走进教室开始学习')
                else:
                        print('教室门没开,班长帮忙开个门')
        else:
                print('教学楼门没开,班主任帮忙开个门')
else:
        print('校门没开,大爷帮忙开个门')
```

5.3 循环结构

5.3.1 while 循环

基本格式:

```
while 条件判断:
        条件为真时,执行代码
        ....
```

如: 计算0-100(包含100)所有数之和

```
num = 0
total = 0
while num <= 100:
        total += num
        num += 1
print(total)
```

又如: 输出十行十列的星星

```
i = 0
while i < 10:
        j = 0
        while j < 10:
                print('✨',end= '')
                j += 1

        print('\n',end='')
        i += 1
```

又如: 使用单循环输出10行10列的星星

```
i = 0
while i < 100:
        print('✨',end = '')
        if i % 10 == 9:
                print()
        i += 1
```

又如: 输出十行十列星星,隔行变色

```
i = 0
while i < 10:
        j = 0
        while j < 10:
                if i % 2 == 0:
                        print('✨',end = '')
                else:
                        print('❤',end = '')
                j += 1

        print()
        i += 1
```

又如: 使用单循环实现隔行变色

```
i = 0
while i < 100:
        if (i // 10) % 2 == 0:
                print('✨',end = '')
        else:
                print('❤',end = '')
        if i % 10 == 9:
                print()
        i += 1
```

5.3.2 死循环

基本格式:

```
while True:
        循环内容
        ...
```

如: 判断用户输入的密码是否正确

```
error = False
passwd = 123
```

```
while True:
        userinput = input('请输入密码:')

        # 检查密码格式是否正确
        for each in userinput:
                if each not in '0987654321':
                        error = True
                        break

        if error == True:
                print('密码格式错误,请重新输入纯数字密码!')
                error = False
                continue
        else:
                if userinput == str(passwd):
                        print('密码正确,登录成功!')
                        break
                else:
                        print('密码错误,请重新输入!')
```

5.3.3 for ... in 循环

基本格式:

```
for 变量 in 容器:
        循环体
        ...
```

遍历除字典外的不等长的二级容器(列表,元组,集合)

```
for 变量1 in 容器:
        for 变量2 in 变量1:
                循环体
                ...
```

遍历字典中的键和值:

```
for key in dict1:
        print(key,dict1[key])

或者

for key,value in dict1.items():
        print(key,value)
```

遍历等长的二级容器;

```
list1 = [
  ['a','b','c'],
  ['d','e','f'],
  ['g','h','i']
]
for x,y,z in list1:
        print(x,y,z)
```

5.3.4 break 和 continue

break: 终止循环

如: 输出0-100(包含100)的数字,遇到44则停止循环

```
i = 0:
while i <= 100:
        if i == 44:
                break
        print(i)
        i += 1
```

continue: 跳过本次循环,开始下一次循环. 开始下一次循环的同时要进行判断

如: 输出0 -100(包含100)的数字,遇到任何带4的则跳过

```
i = 0
while i <= 100:
        if i % 10 == 4 or i // 10 == 4:
                i += 1
                continue
        else:
                print(i)
                i += 1
```

6.函数

- 定义: 把具有特定功能的代码打包就是函数
- 作用
    1. 提高代码的复用率
    2. 提高开发效率
    3. 便于程序的维护

6.1 函数声明方式

def 函数名( ) :

pass(函数内容)

函数名( ) # 调用函数

6.2 函数名命名规则

1. 使用英文,禁止使用中文
2. 可以使用数字,但是不能用数字开头
3. 所有的符号只能用_下划线
4. 命名要有意义
5. 严格区别大小写
6. 避免和系统保留的关键字重名
7. 避免和系统保留的函数重名

6.3 函数的参数

6.3.1 形参

函数在定义过程中,( )中的参数就是形参.形参没有实际意义,只是为了占位接收实参的值

6.3.1.1 普通形参

- 没有实际意义,只是为了占位,等待接收实参对他赋值.
- 如果定义了形参,形参也没有设置默认值,那么在调用函数的时候如果没有实参就会报错

如:

```
def hanshu(a,b):
        print(a)
        print(b)
hanshu(1,2)
```

6.3.1.2 带有默认值的形参

- 在函数定义的时候如果给了形参默认值,那么就是带有默认值的形参
- 那么如果调用的时候如果没有传入实参,形参会按默认值进行运算
- 若调用的时候传入了实参,那么实参的值会覆盖形参的默认值,按实参的值进行运算

如:

```
def hanshu(a = 1, b = 2):
        print(a)
        print(b)


hanshu()
hanshu(3,5)
```

6.3.1.3 普通收集形参 *

- 在不确定有多少实参传入的时候,可以使用普通收集形参: * 变量名
- 收集到的实参会组成一个元组
- 形参优先级: 普通形参 > 普通收集形参 > 关键字收集形参

如:

```
def hanshu(*tuple1):
        total = 0
        for each in tuple1:
                total += each
        print(total)

hanshu(1,2,3,4,5,6,7)
```

6.3.1.3 关键字收集形参 **

- 关键字收集形参只能收集关键字实参的值
- 收集到的关键字实参会组成一个字典,关键字作为字典的键,实参的值作为字典的值
- 形参优先级: 普通形参 > 普通收集形参 > 关键字收集形参

如: 传入人名和年纪,计算总和

```
def age(**dict1):
        total = 0
        for key in dict1:
                total += dict1[key]
        print(total)

age('徐凤年'= 22 ,'徐骁'= 55 , '徐龙象'= 18)
```

age(党元=22,党宝=25)

6.3.2 实参

6.3.2.1 普通实参

- 在调用函数的时候,( )中的值就是实参.若没有通过关键字的方式进行传参的就是普通实参
- 实参的优先级: 普通实参 > 关键字实参

如:

```
def hanshu(a):
        print(a)

hanshu('哗啦啦')
```

6.3.2.2 关键字实参

- 在调用函数的时候,通过关键字=值的方式来进行传参,这就是关键字实参
- 实参的优先级: 普通实参 > 关键字实参

如:

```
def hanshu(a,b):
        print(a)
        print(b)

hanshu(a='关键',b='字')
```

6.4 函数的返回值

- 函数按照有无返回值可以分为两类:
    1. 有返回值的函数 — 含有return , 函数执行之后,会返回一个结果,可以用变量接受
    2. 执行过程的函数 — 没有返回值的函数, 没有return , 函数执行只有不会返回一个结果,用变量接受为 none
- return的作用

1. 为函数的运行返回一个结果
2. 终止函数执行,一旦函数运行到了return,则在return之后的代码将不会运行

6.5 函数文档

6.5.1 查看函数文档的方法

- 方法一: help( ) —help(函数名)

  如:

  ```
  help(print)

  help(id)

  help(type)
  ```

- 方法二: 函数名. _ _ doc _ _

  如:

```
print.__doc__

id.__doc__

type.__doc__
```

6.5.2 定义函数文档的方法

```
def 函数名():
        '''
        功能:
        参数:
        返回值:
        '''

        函数内容...
```

6.6 变量的作用域

6.6.1 全局变量

1. 在任何区域或者页面都有效的变量
2. 在全局环境中直接声明的变量就是全局变量
3. 在局部环境中修改全局变量的值需要使用global关键字

```
var = '我是全局变量'
def hanshu():
        jubuvar = '我是局部变量'
```

6.6.2 局部变量

1. 在特定的局部区域有效的变量就是局部变量
2. 在函数内部(局部环境中)声明的变量就是局部变量
3. 在内部函数中修改内部函数外部的非全局变量,需要使用nonlocal关键字

6.6.3 局部变量和全局变量的关系

1. 在全局环境中不能直接使用局部变量和内部函数,需要闭包之后才能使用
2. 在局部环境中可以访问全局变量,但是不能修改全局变量的值,需要使用global关键字

6.6.4 global 关键字

1. 作用一:使全局变量能进入局部环境

```
var = '我是全局变量'

def hanshu():
        global var
        var = '全局变量进入局部环境,修改全局变量'

hanshu()
print(var)
```

2. 作用二:将局部变量升级为全局变量 **可以在全局对变量进行操作**

```
def hanshu():
        global var
        var = '我升级成全局变量'

hanshu()
print(var)
```

6.6.5 内部函数

1. 定义: 在一个函数的内部在定义一个函数,再定义的函数就是内部函数. 内部函数本质上相当于一个局部变量

```
def outer():
        def inner():
                print('我是内部函数')
        inner()

outer()
```

2. 内部函数的作用域:
    1. 内部函数不能直接在全局环境中使用,需要使用闭包将它带到全局环境中
    2. 在函数内部可以调用内部函数
    3. 在函数内部需要先定义内部函数,之后才能调用

6.6.6 闭包

1. 定义: 让局部变量和内部函数在函数的外部可以使用即闭包
2. 作用: 让局部变量和内部函数突破局部作用域

闭包实现方法:

```
        1. 方法一: 使用global关键字实现闭包
```

```
lists = []

def hanshu():
        global lists
```

```
        a = '局部变量a'
        b = '局部变量b'

        def inner1():
                print('内部函数inner1')
        def inner2():
                print('内部函数inner2')

        lists = [a,b,inner1,inner2]

hanshu()

num_a = lists[0]
num_b = lists[1]
neibu1 = lists[2]
neibu2 = lists[3]

neibu1()
neibu2()
```

2. 方法二: 使用return实现闭包

```
def hanshu():

        a = '局部变量a'
        b = '局部变量b'

        def inner1():
                print('我是内部函数inner1')
        def inner2():
                print('我是内部函数inner2')

        return [a,b,inner1,inner2]

lists = hanshu()

num_a = lists[0]
num_b = lists[1]
neibu1 = lists[2]
neibu2 = lists[3]

neibu1()
neibu2()
```

3. 方法三: 使用return内部函数实现闭包

```
def hanshu():

        a = '局部变量a'
```

```
        b = '局部变量b'

        def inner1():
                print('我是内部函数inner1')
        def inner2():
                print('我是内部函数inner2')

        def bibao():
                return [a,b,inner1,inner2]

        return bibao


result = hanshu()
lists = result()

num_a = lists[0]
num_b = lists[1]
neibu1 = lists[2]
neibu2 = lists[3]

neibu1()
neibu2()
```

6.6.7 nonlocal 关键字

用于在内部函数中修改内部函数外部的非全局变量的值

如:

```
def user():

        jubuvar = '我是局部变量'

        def change(newvar):
                nonlocal jubuvar
                jubuvar = newvar

        def look():
                print(jubuvar)

        def bibao():
                return [jubuvar,change,look]

        return bibao

result = user()
lists = result()

jubu = lists[0]
modif = lists[1]
```

```
kan = lists[2]

modif('修改局部变量')

kan()
```

6.7 lambda 表达式

1. 基本格式:

   ```
   函数名 = lambda 形参1,形参2,... : (自带return) 函数内容
   ```

   如:

   ```
   total = lambda no1,no2,no3 : no1 + no2 + no3

   result = total(1,2,3)
   print(result)
   ```

2. 带有分支结构的lambda表达式:

   ```
   函数名 = lambda 形参1,形参2,... : 条件为真时执行结果 if 条件判断 else 条件为假时执行结果
   ```

   如:

   ```
   large = lambda no1,no2 : no1 if no1 > no2 else no2
   ```

   又如:

   ```
   large = lambda no1,no2,no3: no1 if no1 > no2 and no1 > no3 else no2 if no2 > no3 else no3
   ```

6.8 递归函数

在函数中调用函数自身的函数就叫做递归函数

如:

```
def digui(num):

        print(num)

        if num > 0:
                digui(num - 1)
        else:
                print('-------------------')

        print(num)
```

---

7. 字符串

7.2 字符串基本操作

1. 字符串连接操作: +

   ```
   string1 = '大风车'
   string2 = '吱呀吱呀转'
   result = string1 + string2
   print(result)
   ```

2. 字符串复制操作: *

   ```
   string1 = '女神'
   result = string1 * 3
   print(result)
   ```

3. 字符串索引操作: [ ]

   ```
   string1 = '我是字符串'
   print(string1[4])
   print(string1[-1])
   ```

4. 字符串分片操作: [::]

   ```
   分片 [起始位置:结束位置:间隔]      --- 不包含结束位置

   string1 = '我是字符串'
   string2 = string1[1:3]
   string3 = string1[::2]
   ```

7.2 字符串函数

7.2.1 大小写相关

capitalize( )

使字符串第一个第一个字母大写

```
string1 = 'i love you'
result = string1.capitalize()
print(result)
```

title( )

使字符串字母满足标题化格式(所有单词首字母大写)

```
string1 = 'i love you'
result = string1.title()
print(result)
```

upper( )

使字符串中的所有字母都大写

```
string1 = 'who is your dad ?'
result = string1.upper()
print(result)
```

lower( )

使字符串中所有字母都小写

```
string1 = 'who is your Dad ? '
result = string1.lower()
print(result)
```

swapcase( )

将字符串中的大小写互换

```
string1 = 'I am your Dad !'
result = string1.swapcase()
print(result)
```

7.2.2 获取长度及出现次数

len( )

len( )为内置函数

获取字符串或其他容器的长度

```
string1 = '我是字符串'
result = len(string1)
print(result)
```

count( )

count(指定字符,起始,终止)

获取指定字符在指定范围内出现的次数,若没有出现过,返回 0

```
string1 = 'the only way to longer the day is to steal time from night!'
result = string1.count('o',2,6)
print(result)
```

7.2.3 获取索引值

find( )

获取指定字符在字符串中指定范围内第一次出现的索引值. 若没有出现则返回 -1

find(指定字符,起始,终止)

```
string1 = '8000 words in the world, only the love can kill man'
result = string1.find('a')
print(result)
```

index( )

获取指定字符在字符串中指定范围内第一次出现的索引值,若没有出现则直接报错

```
string1 = '8000 words in the world, only the love can kill man'
result = string1.index('a')
print(result)
```

7.2.4 检测类字符串函数

startswith( )

检测字符串是否以指定字符在指定范围内开头

startswith(指定字符,起始,终止)

```
string1 = 'you are the sun in the world'
result = string1.startswith('a',4,8)
print(result)
```

endswith( )

检测字符串是否以指定字符在指定范围内结尾

endswith(指定字符,开始,结束)

```
string1 = 'you are the sun in the world'
result = string1.endswith('w',-5)
print(result)
```

istitle( )

检测字符串是否满足标题格式

```
string1 = 'we all forgot the story of the passtime'
result = string1.istitle()
print(result)
```

isupper( )

检测字符串是否满足全部大写

```
string1 = 'WE ARE THE DOTA-ALLSTARS'
result = string1.isupper()
print(result)
```

islower( )

检测字符串是否满足全部字母小写

```
string1 = 'we are dota-allstars'
result = string1.islower()
print(result)
```

isdigit( )

和 isnumeric( ) isdecimal( ) 相同

判断字符串是否全部都是数字

```
num = '123456'
result = num.isdigit()
print(result)
```

isalnum( )

判断字符串是否由字母或者其他文字或者数字组成 (不包含符号)

```
string1 = 'sad is the reason of thinking too much'
result = string1.isalnum()
print(result)
```

isalpha( )

判断字符串是否由字母或者其他文字组成(不包含符号和数字)

```
string1 = 'these are only words'
result = string1.isalpha()
print(result)
```

isspace( )

判断字符串是否由空白字符组成

```
string1 = '\n\t\r\\ '
result = string1.isspace()
print(result)
```

7.2.5 切割和拼接

split( )

将字符串按指定字符进行切割,切割之后放入列表内

split(指定字符)

```
string1 = '我-有一剑-可开天门!'
list1 = string1.split('-')
print(list1)
```

join( )

用指定字符将列表中的元素拼接成字符串

'指定字符'.join(列表)

```
list1 = ['我','有一剑','可开天门']
string2 = '!'.join(list1)
print(string2)
```

splitlines( )

使用字符串中的换行符号\n来切割字符串

```
string1 = '天不生李淳罡,\n剑道万古如长夜'
list1 = string1.splitlines()
print(list1)
```

7.2.6 填充类

zfill( )

指定字符长度,用0从左到右进行填充

```
string1 = '123'
result = string1.zfill(10)
print(result)

0000000123
```

center( )

指定字符长度,原字符串居中,使用指定字符填充到指定长度

center(长度,指定字符)

```
string1 = '呵呵姑娘'
result = string1.center(10,'_')
print(result)

---呵呵姑娘---
```

ljust( )

指定字符长度,原字符串居左,使用指定字符填充至指定长度

ljust(长度,指定字符)

```
string1 = '呵呵姑娘'
result = string1.ljust(10,'-')
print(result)

呵呵姑娘------
```

rjust( )

指定字符长度,原字符串居右,使用指定字符填充至指定长度

rjust(长度,指定字符)

```
string1 = '呵呵姑娘'
result = string1.rjust(10,'-')
print(result)

------呵呵姑娘
```

7.2.7 清除字符串类

strip( )

清除字符串左右两边的指定字符

```
string1 = '___呵呵___'
result = string1.strip('_')
print(result)

呵呵
```

lstrip( )

清除字符串左边指定的字符

```
string1 = '___呵呵___'
result = string1.lstrip('_')
print(result)

呵呵___
```

rstrip( )

清除字符串右边指定的字符

```
string1 = '___呵呵___'
result = string1.rstrip('_')
print(reuslt)

___呵呵
```

7.2.8 字符的替换操作

maketrans( )

字典名 = ''.maketrans(所有要替换的字符,新的字符)

制作存有替换映射关系的字典

```
string1 = '十年修得宋玉树,百年修得徐凤年,千年修得吕洞玄'
dict1 = ''.maketrans('十百千','百千万')
```

translate( )

字符串.translate(字典名)

将字符串按照存有映射关系的字典进行修改

```
newstring = string1.translate(dict1)
print(newstring)

百年修得宋玉树,千年修得徐凤年,万年修得吕洞玄
```

7.2.9 format( ) 格式化

format( ) 基本格式:

```
string1 = '{}在{}看见了{}'
result = string1.format('我','莆田','刘姥姥')
print(result)

我在莆田看见了刘姥姥
```

format( ) 四种传参方式:

1. 按位置进行传参

```
string1 = '{}在{}看见了{}'
result = string1.format('我','莆田','刘姥姥')
print(result)

我在莆田看见了刘姥姥
```

2. 按位置标识传参

```
string1 = '{2}在{1}看见了{0}'
result = string1.format('刘姥姥','莆田','我')
print(result)

我在莆田看见了刘姥姥
```

3. 按关键字传参

```
string1 = '{who}在{where}看见了{what}'
result = string1.format(what='刘姥姥',where='莆田',who='我')
print(result)

我在莆田看见了刘姥姥
```

4. 使用容器传参

```
string1 = '{0[0]}在{0[1]}看见了{0[2]}'
result = string1.format(['我','莆田','刘姥姥'])
print(result)

我在莆田看见了刘姥姥
```

format( )实现居中 居左 居右

1. 居中填充 center( )　{:填充物^长度}

```
string1 = '我喜欢吃{:*^11}你喜欢么'
result = string1.format('小龙虾')
print(result)

我喜欢吃****小龙虾****你喜欢么
```

2. 居左填充 ljust( )　{:填充物<长度}

```
string1 = '我喜欢吃{:*<11}你喜欢么'
result = string1.format('小龙虾')
print(result)

我喜欢吃小龙虾********你喜欢么
```

3. 居右填充 rjust( ) {:填充物>长度}

```
string1 = '我喜欢吃{:*>11}你喜欢么'
result = string1.format('小龙虾')
print(result)

我喜欢吃*******小龙虾你喜欢么
```

format( ) 实现进制转换

1. 二进制 {:b}

```
num = '{}转换成二进制为:{:b}'
result = num.format(10,10)
print(result)
```

2. 八进制 {:o}

```
num = '{}转换成八进制为:{:o}'
result = num.format(10,10)
print(result)
```

3. 十进制: {:d}

```
num = '{}转换成十进制为:{:d}'
result = num.format(10,10)
print(result)
```

4. 十六进制: {:x}

```
num = '{}转换成十六进制为"{:x}'
result = num.format(10,10)
print(result)
```

8. 内建函数

8.1 类型转换相关

int( ):

 将其他类型转换成整型

float( ):

 将其他类型转换成浮点型

bool( ):

 将其他类型转换成布尔值

complex( ):

 将其他类型转换成复数

str( ):

 将其他类型转换成字符串

list( ):

 将其他类型转换成列表

tuple( ):

 将其他类型转换成元组

set( ):

 将其他类型转换成集合

dict( ):

 将其他类型转换成字典

8.2 变量相关

type( ):

 查看变量的类型

id( ):

 查看变量的内存id

print( ):

 打印

locals( ):

 查看当前环境中的所有变量

8.3 数学相关

- abs( )

  获取一个数值的绝对值

```
num = -99
result = abs(num)
print(result)
```

- max( )

  获取序列或者多个参数的最大值

```
list1 = [1,2,3,4,5,6,7,8,9]
result = max(list1)
print(result)

result = max(1,2,3,4,5,6,7,88,0)
print(result)
```

- min( )

  获取序列或者多个参数的最小值

```
list1 = [1,2,5,6,4,1,345,68,2]
result = min(list1)
print(result)

result = min(1,214,5,6,7235,67)
print(result)
```

- round( )

  四舍五入,可以加参数来指定保留几位小数

```
num = 3.1415
result = round(num,3)
print(num)
```

- pow( )

  计算指定数字的n次方

```
result = pow(2,3)   # 2 ** 3
print(result)
```

- bin( )

  将数字转换成二进制

```
result = bin(10)
print(result)
```

- oct( )

  将数字转换成八进制

```
result = oct(10)
print(result)
```

- hex( )

  将数字转换成十六进制

```
result = hex(10)
print(result)
```

8.4 Ascii码相关

- chr( )

  将指定的Ascii码转换成对应字符

```
result = chr(97)
print(result)
```

- ord( )

  将指定的字符转换成ascii码

```
result = ord('a')
print(result)
```

- eval( )

  将字符串转换成代码

```
string1 = 'num + 1'
num = 1
num = eval(string1)
print(num)
```

9. 数学模块

在使用数学模块之前需要先导入模块

import math

- ceil( )

  向上取整

```
num = 0.1
result = math.ceil(num)
print(result)
```

- floor( )

  向下取整

```
num = 1.9
result = math.floor(num)
print(result)
```

- pow( )

  和内置函数功能一样,计算指定值的n次方,返回值是浮点数

```
result = math.pow(2,3)
print(result)
```

- sqrt( )

  开平方根

```
result = math.sqrt(4)
print(result)
```

- fabs( )

  功能和内建函数abs( )功能一样,取绝对值,返回值是浮点数

```
num = -9
result = fabs(num)
print(result)
```

- modf( )

  将指定数字拆分成两部分,一部分为小数部分,一部分为整数部分

```
num = 12.55
result = math.modf(num)
print(result)
```

- copysign( )

  让前面数字的符号和后面数字的符号始终保持一致

```
result = math.copy(-9,-2)
print(result)
```

- fsum( )

  功能和sum( )一样,计算序列的和. 返回值是浮点数

```
list1 = [1,2,3,4,5,6]
result = math.fabs(list1)
print(result)
```

- Pi 和 e

```
result = math.pi
print(result)

result = math.e
print(result)
```

10. 随机数模块

使用随机数模块前需要先载入该模块

import random

- random( )

  随机获取 0-1 不包含1的随机浮点数

```
num = random.random()
print(num)
```

- shuffle( )

  将一个序列随机打乱,直接改变原序列的顺序

```
list1 = [1,2,4,5,6,7,89]
random.shuffle(list1)
print(list1)
```

- choice( )

  随机从一个序列中取出一个元素

```
list1 = [1,2,4,5,7,8,5,3]
result = random.choice(list1)
print(result)
```

- uniform( )

  随机获取指定范围内的一个浮点数

```
result = random.uniform(1,100)
print(result)
```

- randrange( )

  随机获得指定范围内的整数

```
result = random.randrange(1,100)
print(result)
```

11. 列表

11.1 列表基本操作

11.1.1 创建列表

11.1.1.1 创建空列表

```
list1 = list()

list1 = []
```

11.1.1.2 创建有数据的列表

```
list1 = [1,23,4,5]
list2 = [1]
```

11.1.2 访问列表中的值

列表名[索引值]

```
list1 = ['徐凤年','姜泥','裴南苇','青鸟']
result = list1[1]
print(result)
```

11.1.3 修改列表中的值

列表名[索引值] = 新的值

```
list1[1] = '白狐'
print(list1[1])
```

11.1.4 删除列表中的值

del 列表名[索引值]

```
del list1[1]
pirnt(list1)
```

del 可以删除任何变量

```
del list1
```

11.2 列表的序列操作

11.2.1 序列相加 +

```
list1 = [1,2,3]
list2 = [4,5,6]
list3 = list1 + list2
print(list3)
```

11.2.2 序列相乘 *

```
list1 = ['徐凤年']
list2 = list1 * 3
print(list2)
```

11.2.3 分片 [: :]

[起始:终止:间隔] 都不包含终止位置

```
list1 = [1,2,3,4,5,6,8,9]
result = list1[1:2]
print(result)

result = list1[:4]
print(result)

result = list1[::2]
print(result)
```

11.2.4 成员检测

in not in

```
list1 = ['徐凤年','姜泥','王初东','裴南苇']
result = '徐凤年' in list1
print(result)

result = '白狐脸' not in list1
print(result)
```

11.2.5 len( ) \ max( ) \ min( )

```
list1 = [1,23,4,5,6,7,8]
result = len(list1)
print(result)

result = max(list1)
print(result)

result = min(list1)
print(result)
```

11.3 遍历列表

遍历一级列表

```
list1 = ['徐凤年','姜泥','白狐脸','裴南苇','鱼幼薇','青鸟']
for each in list1
        print(each)


num = 0
while num < len(list1):
        print(list1[num])
```

遍历等长二级列表

```
list1 = [
  ['北凉','徐凤年','徐骁'],
  ['西蜀','曹常卿','姜泥'],
  ['离阳','人猫','元本溪']
]
for x,y,z in list1:
        print(x)
        print(y)
        print(z)
```

遍历不等长的二级列表

```
list1 = [
  ['北凉','徐凤年'],
  ['曹常卿','姜泥'],
  ['离阳','人猫','元本溪','徐骁']
]

for i in list1:
        for j in i:
                print(j)
```

11.4 列表推导式

11.4.1 普通格式

```
list1 = [1,2,3,4,5,6,7,8,9]
list2 = [i * 2 for i in list1]
print(list2)
```

11.4.2 带有条件的列表推导式

```
list1 = [1,2,3,4,5,6,7,8]
list2 = [i for i in list1 if i % 2 == 0]
print(list2)
```

11.4.3 多循环列表推导式

```
list1 = [1,2,3,4,5,6,7,8]
list2 = [9,10,11,12,13,14]
list3 = [i + j for i in list1 for j in list2]
print(list3)
```

11.4.4 带条件的多循环列表推导式

```
list1 = [1,2,3,4,5,6,7,8]
list2 = [9,10,11,12,13,14]
list3 = [i + j for i in list1 for j in list2 if list1.index(i) == list2.index(j)]
print(list3)
```

11.5 列表函数

- append( )

  向列表的末尾添加元素

```
list1 = [1,2,3,4]
list1.append(5)
print(list1)
```

- insert( )

  向列表指定索引位置前添加指定元素

  insert(索引,值)

```
list1 = [1,2,3,4]
list1.insert(1,'人')
print(list1)
```

- extend( )

  使用一个列表去扩充另一个列表

```
list1 = [1,2,3,4,5,6]
list2 = [7,8,9,10,11]
list1.extend(list2)
print(list1)
```

- pop( )

  将列表指定位置的值取出来. 默认取出最后一位

```
list1 = [1,2,3]
result = list1.pop(0)
print(result)
print(list1)

result = list1.pop()
print(result)
print(list1)
```

- remove( )

  将列表中的指定元素删除

  remove(值)

```
list1 = ['人猫','徐凤年','姜泥']
list1.remove('人猫')
print(list1)
```

- clear( )

  将列表清空

```
list1 = [1,2,3]
list1.clear()
print(list1)
```

- copy( )

  复制列表,复制得到的列表和原列表的内存id不同

```
list1 = [1,2,3,4]
list2 = list1.copy()
print(list2)
print(id(list2))
```

- count( )

  计算字符串中指定字符出现的次数 不同于字符串的函数count(指定字符,起始,终止)

```
list1 = [1,1,1,23,44,24,15,66,3]
result = list1.count(1)
print(result)
```

- index( )

  获取列表中指定元素出现的索引值

```
list1 = ['徐骁','徐凤年','姜泥','裴南苇']
result = list1.index('徐凤年')
print(result)
```

- reverse( )

  将列表排列顺序颠倒,直接改变原列表

```
list1 = [1,2,3,4]
list1.reverse()
print(list1)
```

- sort( )

  将列表按从小到大的顺序排列. 可用参数key reverse

```
list1 = [1,2,3,4,5]
list1.sort()
print(list1)
```

  将列表按从大到小的顺序排列

```
list1.sort(reverse=True)
print(list1)
```

  将列表按自定义的规则排列

```
def func(num):  #一定要有形参
        return num // 10                # 一定要有return

list1.sort(key=func,reverse=True)
print(list1)
```

12.元组

12.1 创建元组

12.1.1 创建空元组

方法一: 使用tuple( )

```
tuple1 = tuple()
print(tuple1,type(tuple1))
```

方法二:使用( )

```
tuple1 = ()
print(tuple1,type(tuple1))
```

12.1.2 创建一个元素的元组

```
tuple1 = (1,)
print(tuple1,type(tuple1))

tuple1 = 1,
print(tuple1,type(tuple1))
```

12.1.3 创建多个元素的元组

```
tuple1 = (1,23,2,4,5)
print(tuple1,type(tuple1))
```

12.2 元组的基本操作(增删改查)

元组不能直接增加,删除,修改元素,

元组基本操作只支持查看元组内的元素: 元组名[索引]

```
tuple1 = (1,23,4,5,5,67)
print(tuple1[1])
```

12.3 元组的序列操作

12.3.1 序列相加

```
tuple1 = (1,2,3,4)
tuple2 = (5,6,7,8)
result = tuple1 + tuple2
print(result)
```

12.3.2 序列相乘

```
tuple1 = (1,2,3)
result = tuple1 * 2
print(result)
```

12.3.3 分片 [: :]

```
tuple1 = ('徐凤年','徐骁','姜泥','青鸟','王仙之','曹长青','白狐脸儿')
print(tuple1[:2])
print(tuple1[3:])
print(tuple1[1:4])
print(tuple1[::2])      # [起始:终止(不包含):间隔]
```

12.3.4 成员检测

```
tuple1 = ('徐凤年','徐骁','姜泥','青鸟','王仙之','曹长青','白狐脸儿')
result = '徐凤年'in tuple1
print(result)

result = '啦啦啦' not in tuple1
print(result)
```

12.4 元组的序列函数

12.4.1 len() , max(), min(), tuple()

```
tuple1 = ('徐凤年','徐骁','姜泥','青鸟','王仙之','曹长青','白狐脸儿')

length = len(tuple1)
print(length)

tuple2 = (1,23,22,4,5,53,6)
print(max(tuple2))
print(min(tuple2))

list1 = [1,2,3]
tuple3 = tuple(list1)
print(tuple3)
```

12.5 元组的遍历

12.5.1 遍历普通元组

```
tuple1 = ('徐凤年','徐骁','姜泥','青鸟','王仙之','曹长青','白狐脸儿')
for i in tuple1:
        print(i)
```

12.5.2 遍历等长的二级元组

```
tuple1 = ((1,2),(3,4),(5,6))
for x,y in tuple1:
        print(x,y)
```

12.5.3 遍历不等长的二级元组

```
tuple1 = ((1,2),(3,4,5),(6,7,8,9))
for i in tuple1:
        for j in i:
                print(j)
```

12.5.4 访问多级元组中的值

```
tuple1 = ((1,2),(3,4,5),(6,7,8,9))
print(tuple1[0][1])
```

12.6 元组推导式

元组推导式的结果是一个生成器,生成器如果不遍历一遍的话,是不会使用的.

12.6.1 普通元组推导式

```
tuple1 = (1,2,3,4,5,6)

tuple2 = (i * 10 for i in tuple1)

for each in tuple2:  # 如果不遍历的话,tuple2这个生成器将一直存在但是不做任何操作
        print(each)
```

12.6.2 带有条件的元组推导式

```
tuple1 = (1,2,3,4,5,6)

tuple2 = (i * 10 for i in tuple1 if i % 2 == 0)

for each in tuple2:
        print(each)
```

12.6.3 多循环元组推导式

```
tuple1 = (1,2,3,4,5)
tuple2 = (10,20,30,40)

tuple3 = (i + j for i in tuple1 for j in tuple2)

for each in tuple3:
        print(each)
```

12.6.4 带有条件判断的多循环元组推导式

```
tuple1 = (1,2,3,4)
tuple2 = (5,6,7,8)

tuple3 = (i * j for i in tuple1 for j in tuple2 if tuple1.index(i) == tuple2.index(j))

for each in tuple3:
        print(each)
```

12.7 元组专用函数

1. index( ) 查看指定元素在元组中的索引值

```
tuple1 = ('华为','小米','大米','三星','iphone')

print(tuple1.index('华为'))

print(tuple1.index('三星'))
```

2. count( ) 计算指定元素在元组中出现的次数

```
tuple1 = ('华为','华为','三星','三星','iphone')

print(tuple1.count('华为'))

print(tuple1.count('三星'))
```

13. 字典

13.1 创建字典

13.1.1 创建空字典

方法一: 使用{ }来创建空字典

```
dict1 = {}
print(dict1,type(dict1))
```

方法二: 使用 dict( ) 来创建空字典

```
dict1 = dict()
print(dict1,type(ditc1))
```

13.1.2 创建多个元素的字典

方法一: 使用 { } 来创建多个元素的字典

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
print(dict1,type(ditc1))
```

方法二: 使用dict( ) 来创建多个元素的字典

```
#1.dict(字典)
dict1 = {{'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}}
print(dict1,type(dict1))
```

```
#2.dict(等长二级容器)
tuple1 = (('小丽','马丽丽'),('静静','刘文静'),('瑶瑶','王瑶'),('紫薇','孙紫薇'))
dict1 = dict(tuple1)
print(ditc1,type(dict1))

#3.dict(关键字传参)
dict1 = dict(小丽='马丽丽',静静='刘文静',瑶瑶='王瑶',紫薇='孙紫薇')
print(dict1,type(dict1))

#4.dict(zip(键的容器,值的容器))
keys = ('小丽','静静','瑶瑶','紫薇')
values = ('马丽丽','刘文静','王瑶','孙紫薇')
dict1 = dict(zip(keys,values))
```

13.2 字典的基本操作(增删改查)

13.2.1 字典中直接增加新元素

字典[键] = 值

如果键不存在在字典中,则直接将这个键值对添加到字典中作为新元素

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}

dict1['班长'] = '戴帅林'

print(dict1)
```

13.2.2 字典中直接删除元素

del 字典[键]

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
del dict1['小丽']
print(dict1)
```

13.2.3 字典中直接修改元素

```
dict1 =  {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict1['小丽'] = '马冬梅'

print(dict1)
```

13.2.4 字典中查看元素的值

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}

print(dict1['小丽'])
```

13.3 字典的序列操作

序列相加,序列相乘,分片这些序列操作字典都不支持

字典只能进行成员检测(只检测字典中的键)

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = '小丽' in dict1
print(result)

reslut = '王贷' not in dict1
print(result)
```

13.4 字典的序列函数

len( ) , max( ) , min( ), dict( )

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}

length = len(dict1)
print(length)

dict2 = {1:'数字',2:'数字',3:'数字',4:'数字',5:'数字',6:'数字',}
maxnum = max(dict2)
print(maxnum)

minnum = min(dict2)
print(minnum)
```

13.5 遍历字典

13.5.1 遍历一级字典

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}

#方法一:
for i in dict1:
        print(i , dict1[i])

#方法二:
for i,j in dict1.items():
        print(i,j)
```

13.6 字典推导式

13.6.1 基本字典推导式

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}

dict2 = {key:'@'+value for key,value in dict1.items()}

print(dict2)
```

13.6.2 带有条件的字典推导式

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}

dict2 = {key:value for key,value in dict1.items() if len(key) == len(value)}

print(dict2)
```

13.6.3 带有循环的字典推导式

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict2 = {'班长':'戴帅林'}

dict3 = {key+i : value+j for key,value in dict1.items() for i,j in dict2.items()}

print(dict3)
```

13.6.4 带有判断的多循环字典推导式

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict2 = {'班长':'戴帅林'}

dict3 = {key+i : value+j for key,value in dict1.items() for i,j in dict2.items() if len(value) ==
len(j)}

print(dict3)
```

13.7 字典的专用函数

1. ### clear( ) 清空字典

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict1.clear()
print(dict1)
```

2. ### copy( ) 复制字典

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict2 = dict1.copy()
print(dict2)
```

3. ### fromkeys( ) 将容器中的值作为键,设定的第二个参数作为所有键的值,创建字典

```
list1 = [1,2,3,4,5]
dict1 = {}.fromkeys(list1,'数字')
print(dict1)

{1:'数字',2:'数字',3:'数字',4:'数字'}
```

4. ### get( ) 通过键获取字典中的值

```
# 若键存在,则返回值
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = dict1.get('小丽')
print(result)

# 若键不存在,则返回None.
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = dict1.get('班长')
print(result)

# 若键不存在,但是设定了返回值.则返回返回值
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = dict1.get('班长','不在')
print(result)
```

5. ### items( ) 将字典转换成等长的二级元组

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = dict1.items()
print(result)
```

6. ## keys( ) 将字典中的键取出来,放进容器中

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
list1 = dict1.keys()
print(list1)
```

7. ## values( ) 将字典中的值取出来,放进容器中

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
list1 = dict1.values()
print(list2)
```

8. ## pop( ) 删除字典中指定键值对 pop(指定的键,默认值)

```
# 移除存在的键和值
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = dict1.pop('小丽')
print(dict1,result)

# 移除不存在的键,未设置默认值,则报错!!! pop(不存在的键,还tm没设置默认值) 等着报错吧!!!
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = dict1.pop('班长')
print(dict1,result)

# 移除不存在的键,设置了默认值 pop(不存在的键,设置了默认值),则返回默认值
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = dict1.pop('班长','不在')
print(dict1,result)
```

9. ## popitem( ) 随机移除出字典中的一个键值对

```
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
result = dict1.popitem()
print(result)
```

10. ## setdefault( ) 向字典中添加元素

```
# 若要添加的键不存在,则新增进字典中
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict1.setdefault('班长','戴帅林')
print(dict1)

# 若要添加的键已经在字典中,则不做任何操作
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict1.setdefault('小丽','马丽丽')
print(dict1)
```

11. ## update( ) 更新字典,直接改变原字典

```
# update(关键字传参)
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict1.update(小丽='马丽丽',静静='刘文静',瑶瑶='大熊太美',紫薇='戏精')
print(dict1)

# update(新的字典)
dict1 = {'小丽':'马丽丽','静静':'刘文静','瑶瑶':'王瑶','紫薇':'孙紫薇'}
dict1.update({'小丽':'马里奥','静静':'静毛线','瑶瑶':'大熊太美','紫薇':'戏精'})
print(dict1)
```

14.集合

集合的特征:

- 集合是无序数据
- 集合中所有元素都是唯一的,集合自带去重功能
- 集合中可以包含Number(int,float,bool,complex) , string ， tuple 和 冰冻集合

14.1 创建集合

14.1.1 创建空集合

```
使用set() 来创建空集合

set1 = set()
print(set1,type(set1))
```

14.1.2 创建具有多个元素的集合

```
set1 = {1,2,3,4,5,6}
print(set1,type(set1))
```

14.2 集合的基本操作(增删改查都不行)

14.3 集合的序列操作

集合只支持成员检测这一个序列操作

14.2.1 成员检测 in　not in

```
set1 = {1,2,3,44,51,6}
result = 1 in set1
print(result)

result = 222 not in set1
print(result)
```

14.5 集合的序列函数

len() , max() , min() , set()

```
set1 = {1,2,3,4,5,6}

length = len(set1)
print(length)

maxnum = max(set1)
print(maxnum)

minnum = min(set1)
print(minnum)

list1 = [1,2,3,4,5,6,7]
set1 = set(list1)
print(set1,type(set1))
```

14.6 遍历集合

14.6.1 遍历一级集合

```
set1 = {1,23,45,3,7,54}
for each in set1:
        print(each)
```

14.6.2 遍历二级集合(等长)

```
set1 = {('徐骁','徐凤年'),('西楚姜皇帝','姜泥'),('谢灵云','白狐脸'),('无名氏','呵呵')}
for x,y in set1:
        print(x,y)
```

14.6.3 遍历二级集合(不等长)

```
set1 = {('不才','任然'),('张楚','窦唯','何勇'),('陈粒')}

for i in set1:
        for j in i:
                print(j)
```

14.7 集合推导式

14.7.1 基本集合推导式

```
set1 = {1,2,3,4,5,6,7,8}
result = {i*2 for i in set1}
print(result)
```

14.7.2 带条件判断的集合推导式

```
set1 = {1,2,3,4,5,6,7,8}
result = {i * 10 for i in set1 if i % 2 == 0}
print(result)
```

14.7.3 多循环集合推导式

```
set1 = {'窦唯','Gai','万晓利'}
set2 = {'女儿情','天干物燥','噢,乖'}

result = {i + j for i in set1 for j in set2}
print(result)
```

14.7.4 带条件判断的多循环集合推导式

```
set1 = {'窦唯','Gai','万晓利'}
set2 = {'女儿情','天干物燥','噢,乖'}

result = {i + j for i in set1 for j in set2 if len(i) == len(j)}
```

14.7 集合的函数操作

add( ) 向集合中添加元素

```
set1 = {'窦唯','Gai','万晓利'}

set1.add('朴树')
print(set1)
```

pop( ) 从集合中随机取出一个元素

```
set1 = {'窦唯','Gai','万晓利'}
result = set1.pop()
print(result)
```

remove( ) 从集合中删除指定元素

```
set1 = {'窦唯','Gai','万晓利'}

# remove(不存在的元素)   ---  报错
set1.remove('窦唯老婆')

# remove(存在的元素)     --- 正常删除
set1.remove('窦唯')
```

discard( ) 从集合中删除指定元素

```
set1 = {'窦唯','Gai','万晓利'}

# discard(不存在的元素)  ---  不做任何操作
set1.discard('窦唯老婆')

# discard(存在的元素)    ---  正常删除
set1.discard('窦唯')
```

clear( ) 清空集合

```
set1 = {'窦唯','Gai','万晓利'}

set1.clear()

print(set1)
```

copy( ) 复制集合

```
set1 = {'窦唯','Gai','万晓利'}
set2 = set1.copy()
print(set2,type(set2))
```

14.8 集合之间运算函数

difference( ) 计算一个集合相对另一个集合的差集

```
set1 = {'徐凤年','徐骁','李义山'}
set2 = {'徐凤年','姜泥','青鸟'}

result = set1.difference(set2)
print(result)
```

difference_update( ) 计算一个集合相对另一个集合的差集,并直接改变原集合

```
set1 = {'徐凤年','徐骁','李义山'}
set2 = {'徐凤年','姜泥','青鸟'}

set1.difference_update(set2)
print(set1)
```

intersection( ) 计算一个集合和另一个集合的差集

```
set1 = {'徐凤年','徐骁','李义山'}
set2 = {'徐凤年','姜泥','青鸟'}

result = set1.intersection(set2)
print(result)
```

intersection_update( ) 计算一个集合和另一个集合的交集,并直接改变原集合

```
set1 = {'徐凤年','徐骁','李义山'}
set2 = {'徐凤年','姜泥','青鸟'}

set1.intersection_update(set2)
print(set1)
```

union( ) 计算两个集合的并集

```
set1 = {'徐凤年','徐骁','李义山'}
set2 = {'徐凤年','姜泥','青鸟'}

result = set1.union(set2)
print(result)
```

update( ) 计算两个集合的并集,并更新原集合

```
set1 = {'徐凤年','徐骁','李义山'}
set2 = {'徐凤年','姜泥','青鸟'}

set1.update(set2)
print(set1)
```

symmetric_difference( ) 计算两个集合的对称差集

```
set1 = {'徐凤年','徐骁','李义山'}
set2 = {'徐凤年','姜泥','青鸟'}

result = set1.symmetric_difference(set2)
print(result)
```

symmetric_difference_update( ) 计算两个集合的对称差集并更新原集合

```
set1 = {'徐凤年','徐骁','李义山'}
set2 = {'徐凤年','姜泥','青鸟'}

set1.symmetric_difference_update(set2)
print(set1)
```

14.9 集合检测类函数

issuperset( ) 检测一个集合是否是另一个集合的超集

```
set1 = {'徐凤年','徐骁','李义山','姜泥','青鸟'}
set2 = {'徐凤年','姜泥','青鸟'}

result = set1.issuperset(set2)
print(result)
```

issubset( ) 检测一个集合是否是另一个集合的子集

```
set1 = {'徐凤年','徐骁','李义山','姜泥','青鸟'}
set2 = {'徐凤年','姜泥','青鸟'}

result = set2.issubset(set1)
print(result)
```

isdisjiont( ) 检测两个集合是否没有交集

```
set1 = {'徐凤年','徐骁','李义山','姜泥','青鸟'}
set2 = {'徐凤年','姜泥','青鸟'}

result = set1.isdisjoint(set2)
print(result)
```

14.10 冰冻集合

1. 创建空冰冻集合

```
set1 = frozenset()
print(set1,type(set1))
```

2. 创建多个元素的冰冻集合

```
set1 = frozenset({1,2,3,4,5,6,8,7,9})
print(set1,type(set1))
```

3. 遍历冰冻集合

```
set1 = frozenset({1,2,3,4,5,6,8,7,9})
for i in set1:
        print(i)
```

4. 冰冻集合推导式

```
result = {i+2 for i in set1}
print(result)
```

5. 冰冻集合函数

```
冰冻集合没有专用的函数
但是
冰冻集合可以使用所有不改变原集合的集合函数
```

15. 文件操作

15.1 文件操作的步骤

1. 打开文件
2. 对文件进行操作
3. 关闭文件

15.1.1 文件写入操作

```
# 1. 使用open()打开指定文件
fp = open('/Users/apple/desktop/01.txt','w')

# 2. 对文件进行写入操作
fp.write('写入操作,输入........')

# 3. 关闭文件
fp.close()
```

15.1.2 文件读取操作

```
# 1. 使用open()打开指定文件
fp = open('/Users/apple/desktop/01.txt','r')

# 2. 对文件进行读取操作
result = fp.read()
print(result)

# 3. 关闭文件
fp.close()
```

15.2 文件操作函数

1. open() 打开或新建文件

```
变量 = open('文件绝对路径','打开模式')

打开模式:

基础模式:( w,a,x,r )
w:      write   写入模式 若文件不存在,则新建. 若已经存在,打开文件并清空原有内容
a:  append      追加模式 若文件不存在,则新建. 若已经存在,则打开文件,则原有内容基础上添加
x:      xor             异或模式 若文件不存在,则新建. 若已经存在,则报错
r:      read    读取模式 可执行读取文件的相关操作

增强模式( +,b ) 只能和基础模式配合使用
+:      plus    读写模式 在w,a,x 或者 r 的基础上加+,则实现读写都可的功能
b:      byte    位模式            在位模式下进行读或者写或者读写的操作

组合方式:
w:              若文件不存在,则新建文件等待写入操作. 若文件已存在,则打开文件并清空原内容,之后可以进行写入
操作.
a:      若文件不存在,则新建文件等待写入操作. 若文件已存在,则打开文件并清空原内容,之后可以进行写入操作.
x:              若文件不存在,则新建文件等待写入操作. 若文件已存在,则直接报错!
r:              若文件不存在,则报错! 若文件已存在,则打开文件,并可进行读取相关的操作

w+:             若文件不存在,则新建文件等待写入操作. 若文件已经存在,则打开文件并清空内容,之后可执行写入或
读取
r+:             若文件不存在,则报错! 若文件已经存在,则打开文件,之后可执行写入操作和读取操作
x+:             若文件不存在,则新建文件等待写入操作, 若文件已经存在,则报错!
a+:             若文件不存在,则新建文件等待写入操作, 若文件已经存在,则打开原文件,在原文件的基础上可追加或
读取

wb:             若文件不存在,则新建文件等待写入操作.若文件已经存在,则打开文件并清空内容.所有写入需要用
encode()编码
rb:             若文件不存在,则报错! 若文件已经存在,则打开文件可进行读取操作.所有读取内容要用decode()解
码
ab:             若文件不存在,则新建文件等待写入操作,若文件已经存在,则打开文件并追加内容.所有写入需要用
encode()编码
xb:             若文件不存在,则新建文件等待写入操作,若文件已经存在,则报错!
```

```
wb+:    若文件不存在,则新建文件且可读可写.若文件已存在,清空内容,可读可写.读要用decode()解码.写要用encode()编码
rb+:    若文件不存在,则报错! 若文件已存在,则打开文件,可读可写.读要用decode()解码.写要用encode()编码
ab+:    若文件不存在,则新建文件且可读可写.若文件已存在,打开原内容,追加内容.读要用decode()解码.写要用encode()编码
xb+:    若文件不存在,则新建文件且可读可写,若文件已存在,直接报错! 读要用decode()解码.写要用encode()编码
```

如: 使用r+模式打开文件

```
# 打开文件
fp = open('/Users/apple/desktop/01.txt','r+')   # 若此时01.txt不存在,则直接报错

# 先读取原有内容
content = fp.read()
print(content)

# 写入新内容
fp.write('写入内容')

# 移动指针
fp.seek(0) #单位是字节 0表示移动到开头位置

# 再读取现在的内容
content = fp.read()
print(content)

# 关闭文件
fp.close()
```

如: a+b模式打开或新建文件

```
# 打开文件
fp = open('/Users/apple/desktop/01.txt','a+b')

# 追加内容
fp.write('在原有基础上追加内容'.encode())

# 移动指针到开头
fp.seek(0)

# 读取全部内容
content = fp.read().decode()
print(content)

# 关闭文件
fp.close()
```

2. read() 读取文件内容

1. 文件.read( ) 默认读取全部文件内容

```
fp = open('/Users/apple/desktop/01.xtx' ,'r')

content = fp.read()
print(content)

fp.close()
```

2. 文件.read(指定字符个数) 读取指定的字符个数

```
fp = open('/Users/apple/desktop/01.txt','r')

content = fp.read(20)
print(content)

fp.close()
```

3. readline() 读取一行文件内容

文件.readline( ) 默认读取一行内容

文件.readline(指定字符个数) 默认读取指定字符长度内容,超过一行按一行算,不够一行就不够呗

```
fp = open('/Users/apple/desktop/01.txt','r')

# 读取一行文件的内容
content = fp.readline()
print(content)

# 移动指针
fp.seek(0)

# 读取指定字符长度,若不满足一行则只读5个字符,若超过一行,则只读一行字符
content = fp.readline(5)
print(content)

# 关闭文件
fp.close()
```

4. readlines() 一次读取多行内容

```
# 打开文件
fp = open.(''/Users/apple/desktop/01.txt','r')
```

```
# 默认读取全部行
content = fp.readlines()
print(content)

# 移动指针
fp.seek(0)

# 读取指定字符长度的行数
content = fp.readlines(20)  # 字符长度若不足一行,则按一行算,若超过一行,则显示下一整行
print(content)

# 关闭文件
fp.close()
```

5. truncate() 文件截取

保留指定的字节长度,其他都删除 单位:字节!!!!

```
fp = open('/Users/apple/desktop/01.txt'.'r+')

fp.truncate(12)  # 保留指定的字节长度,其他都删除

fp.close()
```

6. tell() 获取当前打开的文件的指针位置

```
fp = open('/Users/apple/desktop/01.txt','r')

num = fp.tell()
print(num)

fp.close()
```

7. seek() 将指针移动指定的字节长度

```
fp = open('/Users/apple/desktop/01.txt','r')

fp.seek(21)

content = fp.read()   # read() 的内容是在指针之后的内容才能读取
print(content)

fp.close()
```

8. 自定义函数获取指定行数

```
# 自定义函数实现读取文件中的指定行数

def myReadLines(path,hang=-1,sep='\n'):
# path:要打开的文件路径  hang:显示文件内容的行数  sep:当前操作系统中默认的换行符号

    # 打开指定文件
    fp = open(path,'r')

    # 获取文件内所有内容
    content = fp.read()

    # 使用换行符号切割字符串,放入列表lists中
    #若最后一行切完是一个空字符串,则需要分片去掉最后一个 lists = content.split(sep)[0:-1]
    lists = content.split(sep)

    # 判断用户输入的行数,并确认最终输出的总行数
    maxhang = len(lists)

    if hang < 0 or hang >= maxhang:
        hang = maxhang
    else:
        hang = int(hang)

    result = [i + sep for i in lists[:hang]]  # 用分片来控制最终的列表的长度

    # 如果要用列表的形式直接呈现,直接输入result就可以了
    #print(result)

    # 遍历最终的列表result 输出内容
    for i in result:
        print(i,end='')

myReadLines('/Users/apple/desktop/01.txt',5)
```

16. os (操作系统)模块

使用os模块之前要先导入

import os

16.1 os模块函数

16.1.1 文件夹类os操作

1. os.listdir()

功能: 获取指定路径文件夹下所有的文件及文件夹的列表

格式: os.listdir(目标文件夹路径)

返回值: 所有文件及文件夹的列表

```
lists = os.listdir(/Users/apple/desktop)
print(lsits)
```

2. os.mkdir()

功能: 在指定路径创建空文件夹

格式:os.mkdir(目录路径,0o权限数组) 未设定则按默认权限创建

返回值:创建的文件夹绝对路径

```
os.mkdir('/Users/apple/dekstop/a',0o755)

os.mkdir('/Users/apple/desktop/b)
```

3. os.rmdir()

功能: 删除指定路径的空文件夹

格式: os.rmdir(目录路径)

返回值:None

```
os.rmdir('/Users/apple/desktop/a')
```

4. os.makedirs()

功能: 递归创建空文件夹

格式: os.makedirs(目录路径)

返回值: 创建的目录的字符串路径

```
os.makedirs('/Users/apple/desktop/aaa/bb/c')
```

5. os.removedirs()

功能: 递归删除空文件夹

格式: os.removedirs(目录路径)

返回值: None

```
os.removedirs('/Users/apple/desktop/aaa/bb/c')
```

6. os.getcwd()

功能: 获取当前默认工作的文件夹路径

格式: os.getcwd()

返回值:当前工作的目录路径字符串

```
pathnow = os.getcwd()
print(pathnow)
```

7.os.chdir()

功能: 改变当前工作的文件夹路径

格式: os.chdir(目的地路径)

返回值:None

```
# 改变文件夹,在桌面创建新文件

os.chdir('/Users/apple/desktop')
fp = open('01.txt','w')
print(os.getcwd())
fp.close
```

16.1.2 通用os操作

1. os.rename( )

功能: 重命名文件夹

格式: os.rename(原文件夹或文件路径,新名文件夹或文件路径)

返回值: None

```
os.rename('/Users/apple/desktop/01.txt','/Users/apple/desktop/02.txt')
```

2. os.stat( )

功能: 获取指定文件夹或文件的相关信息(属性)

格式: os.stat(指定文件夹或文件的目录)

返回值: 包含属性信息的元组

```
result = os.stat('/Users/apple/desktop/01.txt)
print(result)
```

3. os.getenv()

功能: 获取当前系统的环境变量信息

格式: os.getenv(获取的环境变量名称) 'PATH' 要大写

返回值: 字符串

```
result = os.getenv('PATH')
print(result.split(':'))
```

4. os.putenv()

功能: 设置环境变量信息

格式: os.putenv(环境变量参数,新增值)

返回值: None

```
os.putenv('PATH','/')
os.system('mydir')
```

5. os.system()

功能: 在python中使用系统命令

格式: os.system(系统命令)

返回值: 系统命令返回的结果

```
os.system('ls')
```

16.1.3 os模块中子模块path

1. os.path.abspath()

功能: 判断指定的路径是不是绝对路径

格式: os.path.abspath(指定路径)

返回值: True: 是绝对路径 False: 不是绝对路径

```
result = os.path.abspath('/Users/apple/desktop)
print(result)
```

2. os.path.dirname()

功能: 获取路径中的路径部分

格式: os.path.dirname(指定路径)

返回值: 返回路径部分

```
dname = os.path.dirname('/Users/apple/desktop')
print(dname)
```

3. os.path.basename()

功能: 获取路径中文件的主体部分(文件名.扩展名)

格式: os.path.basename()

返回值: 返回指定路径的主体部分

```
bname = os.path.basename('/Users/apple/desktop)
print(bname)
```

4. os.path.join()

功能: 将两个路径连接起来,合成一个路径

格式: os.path.join(路径1,路径2)

返回值: 合成之后的路径

```
fullpath = os.path.join('/Users/apple','desktop')
print(fullpath)
```

5. os.path.split()

功能: 将指定路径的文件拆分成路径和主体部分,放入元组中

格式: os.path.split(指定绝对路径)

返回值: 元组 (路径部分,主体部分)

```
path1 = '/Users/apple/desktop/01.txt'
tuple1 = os.path.split(path1)
print(tuple1)
```

6. os.path.splitext()

功能: 将指定路径的文件拆封成路径主体 和 扩展名,放入元组

格式: os.path.splitext(指定路径)

返回值: 元组 (名称,扩展名)

```
path1 = '/Users/apple/desktop/01.txt'
tuple1 = os.path.splitext(path1)
print(tuple1)
```

7. os.path.getsize()

功能: 获取指定路径的文件大小.无法获取文件夹大小!

格式: os.path.getsize(指定文件路径)

返回值: 文件大小

```
path1 = '/Users/apple/desktop/01.txt'
size = os.path.getsize(path1)
print(size)
```

8. os.path.isdir()

功能: 判断指定路径是不是文件夹

格式: os.path.isdir(指定路径)

返回值: True 是文件夹 False 不是文件夹

```
result = os.path.isdir('/Users/appple/desktop')
print(result)
```

9. os.path.isfile()

功能: 判断指定路径是不是文件

格式: os.path.isfile(指定路径)

返回值: True 是文件 False 不是文件

```
result = os.path.isfile('/Users/apple/desktop/01.txt')
print(result)
```

10. os.path.islink()

功能: 判断指定路径是不是快捷方式

格式: os.path.islink(指定路径)

返回值: True 是链接 False 不是链接

```
result = os.path.islink('/Users/apple/desktop/01.txt')
print(result)
```

11. os.path.getctime()

功能: 获取指定路径的文件或者文件夹的创建时间

格式: os.path.getctime(指定路径)

返回值: 时间戳

```
creattime = os.path.getctime('/Users/apple/desktop/test/')
print(creattime)
```

12. os.path.getatime()

功能: 获取指定路径的文件或者文件夹的访问时间

格式: os.path.getatime(指定路径)

返回值: 时间戳

```
activetime = os.path.getatime('/Users/apple/desktop')
print(activetime)
```

13. os.path.getmtime()

功能: 获取指定路径文件或者文件夹的修改时间

格式: os.path.getmtime(指定路径)

返回值: 时间戳

```
modifytime = os.path.getmtime('/Users/apple/desktop/01.txt)
print(modifytime)
```

14. os.path.exists()

功能: 判断指定路径的文件或者文件夹是否存在

格式: os.path.exists(指定路径)

返回值: True 存在 False 不存在

```
result = os.path.exists(指定路径)
print(result)
```

15. os.path.isabs()

功能: 判断指定路径是不是一个绝对路径

格式: os.path.isabs(指定路径)

返回值: True 是绝对路径 False 不是绝对路径

```
result = os.path.isabs('/Users/apple/desktop/01.txt')
print(result)
```

16. os.path.samefile()

功能: 判断两个路径指向的是不是同一个文件

格式: os.path.samefile(指定路径1,指定路径2)

返回值: True 是相同文件 False 不是同一个文件

```
path1 = '../../desktop/01.txt
path2 = '/Users/apple/desktop/01.txt'
result = os.path,sanmefile(path1,path2)
print(result)
```

16.1.4 os模块中的值

1. os.curdir

print(os.dir)

当前文件夹符号

用 . 来表示

2. os.pardir

print(os.pardir)

当前文件夹的上一层目录 即父级文件夹

用 .. 表示

3. os.name

print(os.name)

当前系统的内核名称

win — > nt linux/unix -> posix

4. os.linsep

print(os.sep)

当前系统的默认换行符

win -> \r\n linux/unix -> \n

5.os.sep

print(os.sep)

当前系统的路径分隔符

win -> / or \ linux/unix -> /

6. os.extsep

print(os.extsep)

当前系统的文件名和后缀之间的分隔符

win/linux/unix -> .

16.1.5 自定义函数获取文件夹大小

```
# 自定义函数获取文件夹大小
import os

def get_dir_size(path):
    # 获取指定文件夹的文件信息
    lists = os.listdir(path)

    # 初始化大小计数
    size = 0
    # 通过拼接获取完整路径
```

```
    for i in lists:
        fullpath = os.path.join(path,i)

        # 判断路径是不是文件,是文件则获取大小并累加到size.
        if os.path.isfile(fullpath) or os.path.islink(fullpath):
            size += os.path.getsize(fullpath)

        # 判断路径是不是文件夹,如果是文件夹则递归计算大小,累加到size中
        elif os.path.isdir(fullpath):
            size += get_dir_size(fullpath)
    return size

# 调用函数
result = get_dir_size('/Users/apple/desktop/python教材')
print(result)
```

17. shutil 高级系统模块

使用shutil高级系统模块需要先导入该模块

import shutil

17.1 复制功能函数

17.1.1 文件复制类函数

1. shutil.copy()

功能: 将指定路径的文件复制到另一个路径

格式: shutil.copy('原文件路径','目标路径')

返回值: 目标路径

```
import shutil
shutil.copy('/Users/apple/desktop/01.py','/Users/apple/desktop/test/a.py')
```

2. shutil.copy2()

功能: 复制指定路径的文件和文件信息到指定另一个路径

格式: shutil.copy2('原文件路径','目标路径')

返回值: 目标路径

```
import shutil
shutil.copy2('/Users/apple/desktop/01.py','/Users/apple/desktop/test/a.py')
```

3. shutil copyfile()

功能: 复制指定文件的内容到另一个文件中(默认清空另一个文件)

格式: shutil.copyfile('原文件路径','目标路径')

返回值: 目标文件路径

```
import shutil
shutil.copyfile('/Users/apple/desktop/01.txt','/Users/apple/desktop/test/a.txt')
```

4. shutil copyfileobj()

功能: 复制指定文件的内容到另一个文件中(可选择打开模式)

格式: shutil.copyfileobj(open('原文件路径','打开模式'),open('目标地址','打开模式'))

```
import shutil
shutil.copyfileobj(open('/Users/apple/desktop/01.txt','r'),open('/Users/apple/desktop/test/a.txt'
,'a'))
```

17.1.2 文件夹复制类函数

1. copytree()

功能: 复制一个文件夹到指定位置

格式: shutil.copy('原文件夹路径','指定路径')

```
import shutil
shutil.copytree('/Users/apple/desktop/test/','/Users/appple/desktop/a/')
```

2. copymod()

功能: 复制一个文件夹的权限给另一个文件夹(两个必须都存在)

格式: shutil.copymod('原文件夹路径','指定路径')

```
import shutil
shutil.copymod('/Users/apple/desktop/test/','/Users/apple/deskt/a/)
```

3. copystat()

功能: 复制一个文件夹的相关信息给另一个文件夹(两个都必须存在)

格式: shutil.copystat('原文件夹路径','目标文件夹路径')

```
import shutil
shutil.copystat('/Users/apple/desktop/a','/Users/apple/desktop/test/')
```

17.1.3 文件夹(非空)递归删除函数

1. rmtree()

功能: 递归删除非空文件夹 (os.removedirs()只能递归删除空文件夹)

格式: shutil.rmtree('删除文件夹的路径')

```
import shutil
shutil.rmtree('/Users/apple/desktop/test')
```

17.1.4 文件和文件夹通用函数

1. move()

功能: 剪切,将指定文件剪切到另一个位置

格式: shutil.move('原文件路径','指定路径')

```
import shutil

shutil.move('/Users/apple/desktop/a/01.py','/Users/apple/desktop')

shutil.move('/Users/apple/PycharmProjects','/Users/apple/desktop/')
```

17.1.5 系统相关函数

1. which()

功能: 查找系统命令所在的文件路径

格式: shutil.which('系统命令')

返回值: 命令所在的系统变量PATH

```
import shutil
result = shutil.which('ls')
print(result)
```

2. disk_usage()

功能: 获取指定系统磁盘的使用情况

格式: shutil.disk_usage('系统磁盘路径')

```
import shutil
result = shutil.disk_usage('/')
print(result)
```

17.1.6 归档和解档函数

1. shutil.make_archive()

功能: 创建一个归档文件,指定归档文件的格式.再将其他文件或文件夹放入归档文件中

格式: shutil.make_archive('归档文件路径','归档文件格式','放入的文件或文件夹路径')

```
import shutil
shutil.make_archive('/Users/apple/desktop/guidan','zip','/Users/apple/desktop/test')
```

2. shutil.unpack_archive()

功能: 将归档文件夹中的全部文件解包到指定路径

格式: shutil.unpack_archive('归档文件路径','输出路径')

```
import shutil
shutil.unpack_archive('/Users/apple/desktop/guidan.zip','/Users/apple/desktop/nimabi')
```

3. shutil.get_archive_formats()

功能: 获取当前系统允许的压缩文件格式

```
import shutil
result = shutil.get_archive_formats()
print(result)
```

4. shutil.get_unpack_foramats()

功能: 获取当前系统中允许的解包格式

```
import shutil
result = shutil.get_unpack_formats()
print(result)
```

18. zipfile模块-zip压缩

进行压缩操作之前要先导入压缩模块

import zipfile

18.1 zipfile模块常用函数

1. zipfile.ZipFile()

功能: 创建一个压缩文件

格式: zipfile.ZipFile(1.创建压缩文件位置, 2.打开模式, 3.是否压缩, 4.压缩文件是否大于2G)

```
参数:
1. 创建压缩文件绝对路径
2. 打开模式
        w : 新建一个压缩文件夹,或者覆盖一个已有的zip文档
        a : 将数据追加到一个现存的zip文档中
        r : 打开一个已有的zip文件
3. 压缩方式:
        zipfile.ZIP_STORED              不存储不进行压缩(默认)
        zipfile.ZIP_DEFLATED    对文件进行压缩
4. 压缩文件是否大于2G
        若创建的压缩文件要大于2G,则将zip64 设为 True
        若创建的压缩文件不需要2G,则默认False
```

2. zipfile.write()

功能: 将指定文件添加到zip文件中

格式: zipfile.write(要添加的文件,添加后新名字,压缩方式)

```
参数:
1. 要添加的文件:
        要写入压缩文件中的添加文件的绝对路径
2. 添加后的新名字:
        在压缩文件中的名字,如果不需要更改则不需要传参即可
3. 压缩方式:
        压缩方式,若指定则可以单独设定,不指定则按创建zip文件时设定的进行
```

3. extractall()

功能: 从zip压缩文件中解压缩所有的文件

格式: zipfile.extractall(指定输出路径)

4. extarct()

功能: 从zip压缩文件中取出指定的文件

格式: zipfile.extract(指定文件,指定输出路径)

18.2 压缩文件操作范例

```
import zipfile

# 打开或者创建一个压缩文件
zp = zipfile.ZipFile('/Users/apple/desktop/01.zip','w',zipfile.ZIP_DEFLATED)

# 向创建好的压缩文件中添加要压缩的文件
zp.write('/Users/apple/desktop/01.txt)
zp.write('/Users/apple/desktop/test.py','hellotest.py')

# 关闭压缩文件
zp.close()
```

18.3 解压文件操作范例

```
import zipfile

# 打开压缩文件
zp = zipfile.ZipFile('/Users/apple/desktop/01.zip','r')
# 将需要的指定文件或者全部文件解压缩出来
zp.extract('01.txt','/Users/apple/desktop/aa')
zp.extractall('/Users/apple/desktop/bbb')

# 关闭压缩文件
zp.close()
```

18.4 zipfile模块其他函数

1. zipfile.namelist()

功能: 获取zip文件中的所有文件列表

格式: zipfile.namelist()

```
zp = zipfile.ZipFile('/Users/apple/desktop/01.zip','r')

print(zp.namelist())

zp.close()
```

2. zipfile.infolist()

功能: 获取zip文件中的所有信息列表

格式: zipfile.infolist()

```
zp = zipfile.ZipFile('/Users/apple/desktop/01.zip','r')

print(zp.infolist())

zp.close()
```

3. zipfile.getinfo()

功能: 获取zip文件中指定文件的信息

格式: zipfile.getinfo(指定文件)

```
zp = zipfile.ZipFile('/Users/apple/desktop/01.zip','r')

print(zp.getinfo('test.txt'))

zp.close()
```

19. tar 模块

使用tar模块之前需要先导入模块

import tar

19.1 tar 模块常用函数

1. tar.open()

功能: 创建或者打开压缩文件

格式: tar.open('创建或者打开的压缩文件名','打开模式')

注意: 打开模式中 使用w则默认不压缩 要压缩的话使用w:gz等压缩格式

2. tar.add()

功能: 向压缩文件中添加内容

格式: tar.add('添加到压缩文件中的文件或文件夹路径','可为空新名字')

3. tar.extract()

功能: 将压缩文件中的指定文件解压到指定路径

格式: tar.extract('指定路径','解压目标路径')

4. tar.extractall()

功能: 将压缩文件中的所有文件解压到指定路径

格式: tar.extarctall('解压目标路径')

5. tar压缩范例

```
import tar
tarfp = tar.open('/Users/apple/desktop/01.tar','w:gz')

tarfp.add('/Users/apple/desktop/01.py')
tarfp.add('/Users/apple/desktop/test/')


tarfp.close()
```

6. tar解压范例

```
import tar
tarfp = tar.open('/Users/apple/desktop/01.tar','r')

tarfp.extract('01.py','/Users/apple/desktop/')
tarfp.extarctall('/Users/apple/desktop/a/')

tarfp.close()
```

20. calendar 日历模块

使用日历模块之前需要先导入日历模块

import calendar

20.1 日历模块函数

1. calendar.calendar()

功能: 获取指定年份的日历字符串

格式: calendar.calendar(年份)

```
import calendar
result = calendar.calendar(2017)
print(result)
```

2. calendar.month()

功能: 获取指定年月的日历字符串

格式: calendar.month(年份,月份)

```
import calendar
result = calendar.month(2017,10)
print(result)
```

3. calendar.monthcalendar()

功能: 指定年份和月份获取一个时间矩阵列表

格式: calendar.monthcalendar(年份,月份)

```
import calendar
result = calendar.monthcalendar(2017,10)
print(result)
```

4. calendar.monthrange()

功能: 通过指定的年月,获取该月份第一天是周几,一共多少天

格式: calendar.monthrange(年份,月份)

```
import calendar
result = calendar.monthrange(2017,10)
print(result)
```

5. calendar.isleap()

功能: 判断指定年份是不是闰年

格式: calendar.isleap(年份)

```
import calendar
result = calendar.isleap(2017)
print(result)
```

6. calendar.leapdays()

功能: 判断两个指定年份之间有多少个闰年

格式: calendar.leapdays(开始年份,结束年份)

```
import calendar
result = calendar.leapdays(2000,2011)
print(result)
```

7. calendar.weekday()

功能: 通过指定年月日,计算这一天是周几

格式: calendar.weekday(年份,月份,日期)

注意: 0–6 表示 周一 — 周天

```
import calendar
result = calendar.weekday(2017,10,13)
print(result)
```

8. calendar.timegm()

功能: 将时间元组转换成时间戳

格式: calendar.timegm(时间元组)

```
import calendar
ttp = (2018,1,1,0,0,0,0,0,0)
result = calendar.timegm(ttp)
print(result)
```

21. time 日历模块

21.1 时间术语解释

21.1.1 UTC时间

```
UTC时间又称为世界协调时间,特指格林尼治天文台所在的位置的时间,也叫格林尼治时间.
中国的时区是东八区,比世界协调时间快了8个小时
```

21.1.2 夏令时

> 夏令时就是通过在夏季将时间人为调快1个小时.

21.1.3 时间元组

```
ttp = (年,月,日,时,分,秒,周几,第几天,是否夏令时)
年 : 4位数字
月 : 1-12
日 : 1-31
时 : 0-23
分 : 0-59
秒 : 0-59
周几 : 0-6 对应 周一 - 周天
是否不是夏令时: 0是,其他不是
```

21.2 时间模块的值

1. timezone

功能: 获取UTC和当前时区时间戳的差值 (UTC时间戳 - 当前时区时间戳)

```
import time
print(time.timezone)
```

2. altzone

功能: 在夏令时的情况下,获取UTC时间和当前时区的差值

```
import time
print(time.altzone)
```

3. daylight

功能: 检测是否是夏令时,0 就是 夏令时 非零不是夏令时

```
import time
print(time.daylight)
```

21.3 时间模块的函数

1. time.asctime()

功能: 把时间元组转换成可读字符串

格式: time.asctime(时间元组)

```
import time
result = time.asctime((1992,2,1,21,33,44,0,0,0))
print(result)
```

2. time.localtime()

功能: 获取当前的时间元组

格式一: time.localtime()

返回值: 当前的时间元组

格式二: time.localtime(时间戳)

返回值: 指定时间戳转换成本地时间元组

```
import time
result = time.localtime()
print(result)

result = time.localtime(1231424)
print(result)
```

3. time.gmtime()

功能: 获取当前UTC时间元组

格式一: time.gmtime()

返回值: 当前UTC时间元组

格式二: time.gmtime(12414413)

返回值: 将指定时间戳转换成UTC时间元组

```
import time
result = time.gmtime()
print(result)

result = time.gmtime(135454426)
print(result)
```

4. time.ctime()

功能: 获取本地时间的字符串格式

time.ctime() == time.asctime(time.locatime())

```
import time
result = time.ctime()
print(result)
```

5. time.mktime()

功能: 将时间元组转换成时间戳

格式: time.mktime(时间元组)

```
import time
result = time.mktime((1992,2,1,23,45,22,0,0,0))
print(result)
```

6. time.clock()

功能: 获取当前cpu时间,多用于计算程序运行时间

格式: time,clock()

```
import time

starttime = time.clock()

lists = [i *  2 for i in range(1,10000)]

endtime = time.clock()

ptime = endtime - starttime
print(ptime)
```

7. time.perf_counter()

功能: 获取当前cpu时间,可计算sleep()的时间.推荐使用

格式: time.perf_counter

```
import time

starttime = time.perfcounter()

time.sleep(2)

endtime = time.perf_counter()

ptime = endtime - starttime
print(ptime)
```

8. time.sleep()

功能: 使程序睡眠,在此处等待指定的秒数

格式: time.sleep(秒数)

```
import time

time.sleep(15)

print('有完没完了测试需要15s么!?')
```

9. time.strftime()

功能: 指定时间字符的格式,将指定的时间元组转换成指定格式字符

格式: time.strftime('字符格式',时间元组)

字符格式中:

%Y 代表年

%m 代表月

%d 代表日

%H 代表时

%M 代表分

%S 代表秒

```
import time
ttp = (1992,2,1,0,2,3,0,0,0)

result = time.strftime('%Y年%m月%d日,%H时%M分%S秒',ttp)
print(result)
```

10. time.strptime()

功能: 将格式化之后的时间字符按之前的格式还原到时间元组

格式: time.strptime('格式化之后的字符串','格式化的格式')

```
import time

result = time.strptime('1992年2月1日,0时2分3秒','%Y年%m月%d日,%H时%M分%S秒')

print(result)
```

11. time.time()

功能: 获取本地的时间戳

格式: time.time()

```
import time
print(time.time())
```

Python第二阶段

1、多进程

1.1进程创建方法一————fork

fork系统自带方法，仅用于Linux系统

- 程序执行到os.fork时,操作系统会创建一个新的进程(子进程),然后复制父进程的所有信息到子进程中
- 然后父进程和子进程都会从fork()函数中得到一个返回值,在子进程中这个值一定是0,而父进程中是子进程的id号

```
import os

p = os.fork()
print(p)
```

- os.getpid():当前进程的ID
- os.getppid():当前进程的父进程ID--parent:双亲,忽略性别(father,只是父亲的意思),这里用parent更严谨

```
os.getpid()  #获取子进程的ID
os.getppid()  #获取父进程的ID
```

1.2多次fork问题

子进程在复制父进程资源的时候，也将父进程的执行进程也复制过去了，所以子进程中的代码并非全部从头到尾执行。

1.3多进程修改全局变量

多进程中,每个进程中所有的数据(包括变量)都各拥有一份,互不影响

1.4进程创建方法二———-multiprocessing

multiprocessing模块提供了一个Process类来代表一个进程对象

1.5自定义进程类——继承 Process

1.从multiprocessing模块中导入Process类 2.进程操作 1.创建 创建子进程时，只需要传入一个执行函数和函数的参数，创建一个Process实例

创建格式：p=Process(target=函数名) 2.启动 启动格式：进程对象名.start()

```
Process([group[,target[,name[,args[,kwargs]]]]])

   target:表示这个进程实例所调用的对象
   arge:表示调用对象的位置参数元祖
   kwargs:表示调用对象的关键字参数字典
   name:为当前进程实例

常用方法:
   start():启动进程实例(创建子进程);
   join([timeout]):是否等待进程执行结束,或等待多少秒,阻塞
```

```
#例子：
from multiprocessing import Process
import os

# 子进程要执行的代码
def download():
    print("开启进程id:%d"%os.getpid())
if __name__=="__main__":
    print("主进程id:%d"%os.getpid())
    p=Process(target=download) #函数名不加括号

    p.start()
```

- 实例：从键盘输入一个整数,分别开启两个进程来计算这个数的累加和和阶乘
  第一个进程用系统提供给我们的类
  第二个进程需要自己定义

```
from multiprocessing import Process
```

```
def sum(msg):
    j=0
    for i in range(1,msg+1):
        j+=i
    print('%d的累加和是%d'%(msg,j))

if __name__=="__main__":
    num=input('请输入一个整数：')
    num=int(num)

    p1=Process(target=sum,args=(num,))
    p1.start()
    p1.join()
    print('都计算完毕')
**************************

# 创建进程实例2
# 阶乘
from multiprocessing import Process

class JieChen(Process):
    def __init__(self  ): # 参数a用来传入一个数，控制所求阶乘的范围
        Process.__init__(self)

    def run(self):
        pass

    def jiechen(self,a):
        j=1
        for i in range(1,a+1):
            j*=i
        print('%d的阶乘是%d'%(a,j))



# 主进程的入口
if __name__=="__main__":
    # 实例化自定义的类
    p=JieChen()
    p.start()
    p.jiechen(20)
    p.join()
    print('进程执行完毕')
```

1.6进程池pool

- 使用multiprocessing模块提供的Pool方法.

```
#开启进程池
```

```
def shangchuang(msg):
    print('开始上传第%d音乐'% msg)
    time.sleep(1)
    print('上传第%d首音乐成功'% msg)

if __name__=='__main__':
    # 创建进程池
    p = Pool(5)

    for i in range(0,10):
        p.apply_async(func=shangchuang,args=(i,))
    p.close()
    p.join()
    print('上传所有音乐成功')
```

1.7进程间通信——Queue

- 使用multiprocessing模块的Queue实现多进程之间的数据传递

初始化Queue()对象时（例如：q=Queue()），若括号中没有指定最大可接收的消息数量，或数量为负值，那么就代表可接受的消息数量没有上限（直到内存的尽头）；
Queue.qsize()：返回当前队列包含的消息数量；
Queue.empty()：如果队列为空，返回True，反之False ；
Queue.full()：如果队列满了，返回True,反之False；
Queue.get([block[, timeout]])：获取队列中的一条消息，然后将其从列队中移除，block默认值为True；
1）如果block使用默认值，且没有设置timeout（单位秒），消息列队如果为空，此时程序将被阻塞（停在读取状态），直到从消息列队读到消息为止，如果设置了timeout，则会等待timeout秒，若还没读取到任何消息，则抛出"Queue.Empty"异常；
2）如果block值为False，消息列队如果为空，则会立刻抛出"Queue.Empty"异常；
Queue.get_nowait()：相当Queue.get(False)；
Queue.put(item,[block[, timeout]])：将item消息写入队列，block默认值为True；
1）如果block使用默认值，且没有设置timeout（单位秒），消息列队如果已经没有空间可写入，此时程序将被阻塞（停在写入状态），直到从消息列队腾出空间为止，如果设置了timeout，则会等待timeout秒，若还没空间，则抛出"Queue.Full"异常；
2）如果block值为False，消息列队如果没有空间可写入，则会立刻抛出"Queue.Full"异常；
Queue.put_nowait(item)：相当Queue.put(item, False)；

```
from multiprocessing import  Pool,Process,Queue,Manager

#开启2个进程，向进程池里读写信息，不是使用进程池pool***************
def write(q):
    for i in ['小米','苹果','华为','VIVO']:
        print('Starting! Write %s' % i)
        q.put('%s公司好屌的样子！' % i)
        print('写入%s成功' % i)

def read(q):
    while True:

        if not q.empty():
```

```
            for msg in range(q.qsize()):
                message = q.get()
                print('读出消息%s'%message)
        else:
            break


if __name__=='__main__':
    q = Queue()
    # 写入
    q_write = Process(target=write, args=(q,))
    q_write.start()
    q_write.join()

    # 读取
    q_read = Process(target=read, args=(q,))
    q_read.start()
    q_read.join()
```

- 使用进程池Pool创建进程
  如果要使用Pool创建进程，就需要使用multiprocessing.Manager()中的Queue()，而不是multiprocessing.Queue()。

```
from multiprocessing import  Pool,Process,Queue,Manager
# ***************开启2个进程，使用进程池pool*********************
def write(q):
    for i in ['小米', '苹果', '华为', 'VIVO']:
        print('Starting! Write %s' % i)
        q.put('%s公司好屌的样子！' % i)
        print('写入%s成功' % i)

def read(q):
    while True:
        if not q.empty():
            for msg in range(q.qsize()):
                message = q.get()
                print('读出消息%s' % message)
        else:
            break


if __name__ == '__main__':
    q = Manager().Queue()
    p = Pool()

    p.apply_async(func=write,args=(q,))
    p.apply_async(func=read,args=(q,))

    p.close()
    p.join()
```

2、多线程

自定义线程类——继承Thread