一、第一个Python程序

1, 打开Pycharm, 新建项目

2, 创建名为hello.py文件 (Python文件以.py后缀结尾)

b, 在hello.py中输入以下内容

```
print("hello world")
```

c, 在代码区域右键, 选择'运行'文件

二、Python的编码规范

a.在每个语句的后面不用添加分号，每行最好只书写一条语句

b.Python中是通过缩进【4个空格】来区分代码块的

c.尽量避免在文件名中出现中文和空格

三、注释

注释在编写程序的时候，给代码添加的一些解释性的文字

作用：可以提高代码的可读性，以便于以后的参考或者修改

注释是解释性文本，在运行程序的时候，会被自动的跳过，不做处理

分类 单行注释：只能注释掉单行的文字 # xxxxxxx

多行注释【块注释】：三个单引号或者三个双引号

注意：对于多行注释，选用单引号或者双引号都可以

如果单纯使用双引号，双引号的注释不能嵌套

四、输入输出【重点】

1.print输出

作用：将一些特定的数据输出到屏幕上【调试工具】

代码演示：

```
#简单用法:每次只输出一个数据，任何类型都可以
print(10)
print("hello")

#可以通过一条print输出多个数据,使用逗号隔开，最终得到的结果将逗号识别成了空格
print(10,"hello")
```

```
7
8  x = 3
9  y = 4
10 print("x + y = ", x + y)
```

2.input输入

作用：可以将外部的值作为程序中变量的值使用【从控制台获取值】

```
1  #注意：当程序执行到input的时候，程序会停止在这个地方，等待用户的输入，
2  # 当内容输入结束之后，按下enter，此时输入结束
3  #用户输入的值可以通过变量的形式保存起来
4  s = input("请输入内容：")
5  print(s)
6
7  name = input("请输入姓名：")
8  age  = input("请输入年龄：")
9  print(name,age)
10
11
12 #需求:根据控制台输入的年龄计算还能活多少年，假设可以活到100岁
13 death_age = 100
14
15 #注意：通过input获取的数据全部都是字符串
16 age1 = input("请输入现在的年龄：")
17 print(death_age - age1)
18
19
20 #字符串和整型之间的转换
21 """
22 转换
23 int()  将字符串转换为整型
24
25 int:integer,整型，
26 str:string,字符串
27 """
28
29 #修改上面的获取结果的语句
30 print("还可以活",death_age - int(age1),"年")
31
```

```
32
33 #练习：从控制台输入两个数，计算两个数的和，并将结果输出
34 num1 = int(input("请输入第一个数："))
35 num2 = int(input("请输入第二个数："))
36 print(num1 + num2)
37 result = num1 + num2
38 print(result)
```

五、变量

1.变量

1.1. 概念

定义: 程序在运行的过程中，值可以随时发生改变

作用：存储数据，参与运算

1.2. 定义

定义格式：变量名 = 初始值

说明：

变量名： 对于每一个变量，需要给他定义一个名称，定义的规则就是标识符的规则

=： 赋值运算符【主要是为了给变量进行赋值】, 不是数学中的等于,

初始值：第一次给变量赋予的值，在程序运行的过程中，这个值可以随时发生改变

举例：age = 10

age是变量名，10是初始值

```
1 #Python中的变量的定义和数学中变量的定义完全一致
2 x = 3
3 y = 4
4 print(x * y)
5
6 x = "abc"
7 #print(x * y)
8
9 #【扩展】
10 #Python被称为弱类型语言，动态改变变量的数据类型，容错性比较高
11 #Java被称为强类型语言，只要变量被定义，它的数据类型将不能发生任何的改变
```

1.3.删除变量

注意：当定义一个变量，然后删除，相当于这个变量未被定义

代码演示：

```
#定义变量
num = 66
print(num)

#删除变量
#del
#变量被删除之后则相当于这个变量未被定义
del num
print(num)
```

1.4.关键字

关键字: 在Python中，已经被赋予了特殊含义的英文单词

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

1.5.标识符

在Python程序中自定义的一些英文单词

定义合法标识符的规则：

a.只能由数字，字母，下划线组成，不可以是其他的特殊字符【%，&，*，空格等】

b.开头不能是数字

c.不能是关键字

d. 严格区分大小写 例如： age和Age是两个不同的标识符

定义标识符的规范：

a. Python官方要求: 全部用小写字母，不同单词之间使用下划线分隔 stu_score

遵循驼峰命名法【不同的单词之间使用首字母大写的方式进行分隔】

例如：stuScore StuScore

b. 尽量做到见名知意

【注意：规则和规范的区别】

练习：判断下面标识符合法是否

xiaoming 合法

$abc 不合法，有特殊字符

abc_hello 合法

1score 不合法，数字不能开头

score123 合法

扩展:

其他标识符的命名规则:

常量: 常量名所有字母大写，由下划线连接各个单词如MAX_OVERFLOW，TOTAL

函数&方法: 函数名应该为小写，可以用下划线风格单词以增加可读性。如：my_function，my_example_function。

文件名: 全小写,可使用下划线

包: 应该是简短的、小写的名字。如果下划线可以改善可读性可以加入

模块: 与包相同

类: 总是使用首字母大写单词串。如MyClass。内部类可以使用额外的前导下划线

第二天（Day02）

一、Python中的数据类型

```
计算机顾明思议就是可以做数学运算的机器,因此计算机理所当然的可以处理各种数据,
但是计算机能处理的远远不止数值,还有文本,图形,音频,视频网页等各种数据,
不同数据需要定义不同的数据类型
```

Number【数字：整型int，浮点型[小数]float，复数类型complex 】

String【字符串】 str

Boolean【布尔类型】 True真（1）， Flase假（0）

NoneType【空值】

list【列表】 类似c语言的数组

tuple【元组】 不可改变的列表

dict【字典】

set【集合】(了解)

bytes【字节】b'hello'

二、表达式和运算符

1.表达式

操作数和运算符组成, 比如: 1 + 1

作用: 表达式可以求值，也可以给变量赋值

2.运算符【掌握】

2.1. 算术运算符

```
+   -
*【乘法】
/【除法】
%【求余，取模】
**【求幂,次方】
//【取整】
```

```
num1 = 5
num2 = 3
print(num1 + num2)
print(num1 - num2)
print(num1 * num2)
print(num1 / num2)  #浮点型：1.6666666666666667    默认精度16位
print(num1 % num2)  #2
print(num1 ** num2) # 5的3次方
print(num1 // num2) # 获取浮点数的整数部分(向下取整)

#除了+和-之外，其他的算术运算符都是相同的优先级
#出现优先级，解决办法使用括号
print((2 ** 5) * 3)
```

2.2. 赋值运算符

简单赋值运算符：= 给一个变量进行赋值

复合赋值运算符：+= -= %= /= ... 给一个变量进行赋值，同时给变量进行相应的运算

代码演示:

```python
#简单
num1 = 10
#注意：在赋值运算符中，先计算等号右边的表达式，然后将计算的结果赋值给等号左边的变量
num2 = num1 + 10
print(num2)

#复合
num3 = 10
num3 += 100   #等价于num3 = num3 + 100
print(num3)
```

2.3. 关系【条件，比较】运算符

作用：比较大小，得到结果为布尔值

【如果表达式成立，则返回True，如果不成立，则返回False】

```
>     <
>=    <=
 ==【等号】
!= 【不等于】
```

使用场景：if语句，循环

代码演示：

```python
x = 3
y = 5
print(x > y)    #False
print(x < y)

print(x == y)
print(x != y)

print(x >= y)  #False
print(x <= y)  #True
```

2.4. 逻辑运算符

```
and ：与，并且
or：  或，或者
not：  非，取反
```

```
例子：
and:
print(True and True)
print(True and False)
print(False and False)

or:
print(True or True)
print(True or False)
print(False or False)

not:
print(not True)
print(not False)
print(False or False)

age = int(input("请输入年龄："))
height = int(input("请输入身高："))
if age>=18 and age<=30 and height >=170 and height <= 185 :
    print("恭喜，你符合报考飞行员的条件")
else:
    print("抱歉，你不符合报考飞行员的条件")
```

2.5. 成员运算符和身份运算符

```
成员运算符：
    in, not in

 1.in 如果在指定的序列中找到值返回 True，否则返回 False
 2.not in 如果在指定的序列中没有找到值返回 True，否则返回 False。

 例子：
a = 8
list = [1, 2, 3]

if a in list:
    print("a in list true")
else:
```

```
    print("a in list false")

if a not in list:
    print("a not in list true")
else:
    print("a not in list false")
```

身份运算符：

    is, is not

1.is 是判断两个标识符是不是引用同一个对象

2.is not 是判断两个标识符是不是引用自不同对象

示例：

```
a = 20
b = 20

if (a is b):
    print("1 - %d 和  %d 有相同的标识" % (a, b))
else:
    print("1 - %d 和  %d 没有相同的标识" % (a, b))

# 获取对象内存地址
if (id(a) == id(b)):
    print("2 - %d 和  %d 有相同的标识" % (a, b))
else:
    print("2 - %d 和  %d 没有相同的标识" % (a, b))

# 修改变量 b 的值
b = 30
if (a is b):
    print("3 - %d 和  %d 有相同的标识" % (a, b))
else:
    print("3 - %d 和  %d 没有相同的标识" % (a, b))

if (a is not b):
    print("4 - %d 和  %d 没有相同的标识" % (a, b))
else:
    print("4 - %d 和  %d 有相同的标识" % (a, b))
```

```
is 与 == 区别：
is 用于判断两个变量引用对象是否为同一个(同一块内存空间)， == 用于判断引用变量的值是否相等。
例子：
a = [1,2,3]
b = a
s = a is b
print(s)


```

2.6. 位运算符【扩展】

前提：将数字转换为二进制使用

```
&【按位与】
|【按位或】
^【按位异或】
<< 【左移】
>>【右移】
```

代码演示：

```
print(6 & 3)
print(6 | 3)
print(6 ^ 3)
print(6 << 2)
print(6 >> 2)
```

三、分支【重点掌握】

1.代码结构

顺序结构：代码从上往下依次执行

分支结构：根据不同的条件，执行不同的语句

循环结构: 根据指定的条件，重复执行某段代码

2.分支结构-if语句

2.1简单if语句【单分支】

```
语法：
```

```
if 表达式:
        执行语句
```

说明; 要么执行，要么不执行，当表达式成立的之后，则执行语句；如果表达式不成立，则直接跳过整个if语句继续执行后面的代码

注意：表达式为真才执行语句

假：0 0.0 False “” None【空值】

一般情况下，表达式使用都是比较运算符

代码演示：

```
#单分支
num1 = 50
num2 = 60

#需求：当num1 == num2,则给num1重新赋值为100

#在pYthon中，通过缩进来区分代码块
if num1 != num2:
        num1 = 100

print(num1)


#练习：从控制台输入一个数，判断这个数是否是偶数
num = int(input())
if num % 2 == 0:
        print(num,"是一个偶数")

print(num,"不是一个偶数")
```

2.2if-else语句【双分支】

```
语法:
if 表达式:
        执行语句1
else:
        执行语句2
```

说明：如果表达式成立，则执行语句1；如果不成立，则执行语句2

代码演示：

```python
#双分支
#  从控制台输入一个数，判断这个数是否是偶数
num = int(input())

if num % 2 == 0:
        print(num,"是一个偶数")
else:
        print(num,"不是一个偶数")


#练习：从控制台输入一个数字，根据数字打印年龄段
age = int(input())
if age >= 18:
        print("成年人")
else:
        print("未成年人")
```

2.3if-elif-else语句【多分支】

```
语法：
if 表达式1：
        执行语句1
elif 表达式2：
        执行语句2
elif 表达式3：
        执行语句3
。。。。。
else:
        执行语句n
```

说明：实现了多选一的操作，会根据不同的条件从上往下来进行匹配，如果匹配上了，则执行对应的语句，然后直接结束整个if-elif语句，但是，如果所有的条件都不成立的话，则执行else后面的语句

注意：不管if-elif-else有多少个分支，都只会执行其中的一个分支

代码演示：

```python
#多分支
#需求：从控制台输入一个数字，根据数字打印年龄段
age = int(input())
if age < 0:
        print("输入有误")
```

```
elif age <= 3:
        print("婴儿")
elif age <= 6:
        print("儿童")
elif age <= 12:
        print("青少年")
elif age <= 18:
        print("青年")
else:
        print("hello")
```

练习

```
#练习：根据控制台输入的成绩，输出对应的等级
"""
90以上：优秀
80~90：良好
70~80：还行
70以下：加油吧，少年
"""

score = int(input("请输入学生的成绩："))
if score >= 90:
        print("优秀")
elif score >= 80:
        print("良好")
elif score >= 70:
        print("还行")
else:
        print("")
```

2.4嵌套if语句

```
语法：
if 表达式1：
        执行语句1
        if 表达式2：
                执行语句2
```

说明：if语句的嵌套，可以在单分支，双分支，多分支之间进行任意组合

代码演示：

```
score = int(input("请输入学生的成绩："))
if score < 0 or score > 100:
        print("输入有误")
else:
        if score >= 90:
        print("优秀")
        elif score >= 80:
        print("良好")
        elif score >= 70:
        print("还行")
        else:
        print("")

age = int(input("请输入年龄："))
looks = input("请输入您的相貌：")

if age >= 18:
        if looks == "美女":
                print("要微信")
        else:
                print("略过")
```

注意：从语法角度来说，嵌套的层数没有任何的限制，但是，为了代码的可读性和可维护性，嵌套层数不要超过3层

第三天（Day03）

一、循环【掌握】

while循环+for循环(for-in)

1.用法

```
语法：
初始化表达式
while  条件表达式：
        循环体
        循环之后操作表达式

```

```
for 变量名 in 序列：
        循环体
```

2.range

```
range([start,]end[,step])       注：[]表示start和step可写可不写

start: 计数从 start 开始。默认是从 0 开始。
例如range（5）等价于range（0， 5）;
stop: 计数到 stop 结束，但不包括 stop。
例如：range（0，5）是[0, 1, 2, 3, 4]没有5
step：步长，默认为1。
例如：range（0， 5） 等价于 range(0, 5, 1)

功能：生成具有一定规律的序列
```

代码演示：

```
#range()
"""
range([start,]end[,step])
l例如：
range(100)     可以生成一个0~99的整数序列【不包含100】
range（1,100）   可以生成一个1~99的整数序列
range(1,100,2)  可以生成一个1~99之间的奇数序列
"""

#需求1：计算1~100之间所有整数的和
num1 = 1
sum1 = 0
while num1 <= 100:
        sum1 += num1
        num1 += 1
print(sum1)


#借助于range生成一个1~100之间所有整数的序列，然后使用for循环进行遍历这个序列
sum11 = 0
for x in range(1,101):
```

```
        sum11 += x
  print(sum11)

#需求2：计算1~100之间所有偶数的和
第一种：

num2 = 1
sum2 = 0
while num2 <= 100:
        if num2 % 2 == 0:
         sum2 += num2
        num2 += 1
print(sum2)

第二种：

num2 = 0
sum2 = 0
while num2 <= 100:
        sum2 += num2
        num2 += 2
print(sum2)

第三种：

sum22 = 0
for y in range(0,101,2):
        sum22 += y
print(sum22)


```

3.嵌套循环

代码演示：

```
#需求：打印九九乘法表

#while实现
line = 1
```

```
while line <= 9:
        colum = 1
        while colum <= line:
         print("%dx%d=%d"%(colum,line,line*colum),end=" ")
         colum += 1
        print("")
        line += 1


#for实现
#外层循环：控制行
for i in range(1,10):
        #内层循环：控制列
        for j in range(1,i + 1):
        print("%dx%d=%d"%(j,i,i*j),end=" ")
        print("")
```

三. break、continue和pass语句的使用

1.break

作用:跳出循环【直接跳出整个循环，继续执行循环后面的代码】

比喻:

这就好比在操场上跑步，原计划跑 10 圈，可是当跑到第 2 圈的时候，突然想起有急事要办，于是果断停止跑步并离开操场，这就相当于使用了 break 语句提前终止了循环。

代码演示:

```
#break的使用
#1.while
n = 0
while n < 5:
        print("n = %d"%(n))
        #print("n =" ,n)
        #注意：if语句充当的是一个条件判断
        if n == 3:
            break
        n += 1
print("over")


```

```
#2.for
for x in range(1,6):
        print("x = %d"%(x))
        if x == 3:
           break
#结论：不管是while语句还是for语句，break的作用结束整个循环



#3.特殊情况一
#不管while中的条件是否满足，else分支都会被执行
#思考问题：如果在while循环体中出现了break，else分支还会执行吗？-------不会
m = 0
while m < 3:
        print(m)
        if m == 1:
        break
        m += 1
else:
        print("else")



#4.特殊情况二
#当break使用在嵌套循环中的时候，结束的是当前循环【就近原则】
x = 0
y = 0
while x < 20:
        print("hello Python",x)
        x += 1
        while y < 5:
               print("hello Python~~~~",y)
               if y == 2:
                   break
               y += 1
 #break

#注意：break是一个关键字，使用的过程中，单独就可以成为一条语句，后面不能跟任何的变量或者语句
```

2.continue

作用：跳出当前正在执行的循环，继续执行下一次循环

比喻：

仍然以在操作跑步为例，原计划跑 10 圈，但当跑到 2 圈半的时候突然接到一个电话，此时停止了跑步，当挂断电话后，并没有继续跑剩下的半圈，而是直接从第 3 圈开始跑。

代码演示：

```
#continue的使用

#1.for
for i in range(10):
        print(i)
        if i == 3:
           continue
        print("*")

for i in range(10):
        print(i)
        if i == 3:
           break
        print("*")

#总结：continue只是结束当前正在执行的循环，而break表示直接结束整个循环

# 2.while
"""
num = 0
while num < 10:
        print("num = %d"%(num))
        num += 1
        if num == 3:
        continue
"""
num = 0
while num < 10:
        if num == 3:
                num += 1
                continue
```

```
        print("num = %d" % (num))
        num += 1
```

3.pass

Python中的pass是一条空语句

作用：为了保持代码结构的完整性，pass不做任何操作，只是充当了一个占位语句，保证代码可以正常的运行起来

应用场景：if，while，for中使用，可以在代码块的部分不添加任何语句，代码正常运行

代码演示：

```
age = int( input("请输入你的年龄：") )
if age < 12 :
    print("婴幼儿")
elif age >= 12 and age < 18:
    print("青少年")
elif age >= 18 and age < 30:
    print("成年人")
elif age >= 30 and age < 50:
    pass
else:
    print("老年人")

```

4.练习

代码演示：

```
#需求;判断一个数是否是素数【质数】
#方式一
num1 = int(input("请输入一个数："))
#思路：一个数能被其他数整除，将次数记录下来
#条件：在2~num1 - 1的范围内，找到一个数能将num1整除，count1 + 1
count1 = 0
for i in range(2,num1):
    #整除：求余【大数对小数求余】
    if num1 % i == 0:
        count1 += 1

if count1 == 0 and num1 != 1:
    print("是质数")
```

```
else:
    print("不是质数")

#方式二：
#思路：假设num2是质数，寻找不成立的条件【有数能被整除】将假设推翻掉
num2 = int(input("请输入一个数："))
#定义一个布尔类型的变量，用于记录这个数是不是一个质数
is_prime  = True
for j in range(2,num2):
    if num2 % j == 0:
        is_prime = False
        break

if is_prime == True and num2 != 1:
    print("是质数")
else:
    print("不是质数")
```

第四天（Day04）

一、列表list

1.概述

变量：使用变量存储数据，但是，缺点:一个变量每次只能存储一个数据

思考：如果一次性存储多个数据，怎么做?

解决：采用列表

作用：列表相当于是一个容器，可以同时存储多个数据

本质：列表是一种有序的集合

说明：有序指的就是有顺序【数据的存放的顺序和底层存储的顺序是相同的】

代码演示：

```
#需求：求5个人的平均年龄
age1 = 10
age2 = 13
age3 = 16
age4 = 39
age5 = 20

#list
```

```
#在栈空间中有一个变量【列表的名字】
#变量指向了内存堆空间中的一个列表，列表中存储了5个变量
age_list = [10, 13, 16, 39, 20]
```

2.创建列表

语法：变量名 = 列表

列表名称 = [数据1，数据2, ...]

说明：使用[ ]表示创建列表

列表中存储的数据被称为元素

列表中的元素被从头到尾自动进行了编号，编号从0开始，这个编号被称为索引，角标或者下标

索引的取值范围：0~元素的个数 -1【列表的长度 -1】

超过索引的范围：列表越界

代码演示：

```
#语法：列表名【标识符】 = [元素1，元素2.。。。。]
#1.创建列表
#1.1创建一个空列表
list1 = []
print(list1)

#1.2创建一个带有元素的列表
list2 = [52,463,6,473,53,65]
print(list2)

#2.思考问题：列表中能不能存储不同类型的数据？
list3 = ['abc',10,3.14,True]
print(list3)

#注意：将需要存储的数据放到列表中，不需要考虑列表的大小，如果数据量很大的情况，在进行存储数据的时候，列表底层自动扩容
```

3.列表元素的访问

访问方式：通过索引访问列表中的元素【有序，索引：决定了元素在内存中的位置】

3.1获取元素

语法：列表名[索引]

代码演示：

```
#元素的访问
```

```
#创建列表
list1 = [5,51,6,76,98,3]

#需求：获取索引为3的位置上的元素
num = list1[3]
print(num)
print(list1[3])
```

3.2替换元素

语法：列表名[索引] = 值

注意：列表中存储的是其实是变量，所以可以随时修改值

代码演示：

```
#需求：将索引为1位置上的元素替换为100
print(list1[1])
list1[1] = 100
print(list1[1])

#问题：超过索引的取值范围，则会出现索引越界的错误
#解决办法：检查列表索引的取值范围
#print(list1[6])
#IndexError: list index out of range    索引越界
```

3.3 遍历列表

```
#列表的遍历
list2 = [23,54,6,45,56]
#1.直接操作的是元素
for num in list2:
        print(num)

#2.通过索引的方式操作元素
#思路：使用列表生成器生成一个和索引有关的列表 0~len(list2) -1
#len()  是对象（字符、列表、元组等）长度或项目个数。
for index in range(len(list2)):
        #index中保存的是0,1,2....
        n = list2[index]
        print(n)

```

```
#3.同时遍历索引和元素
#enumerate  枚举【类似于一个容器】
#index,n1----->索引，元素值
for index,n1 in enumerate(list2):
        print(index,n1)
```

4.列表的操作

1.1列表元素组合

代码演示:

```
#列表组合【合并】
#使用加号
list1 = [432,435,6]
list2 = ["abc","dhfj"]
list3 = list1 + list2
print(list3)  #[432, 435, 6, 'abc', 'dhfj']
```

1.2列表元素重复

代码演示:

```
#列表元素的重复
#使用乘号
list4 = [1,2,3]
list5 = list4 * 3
print(list5)  #[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

1.3判断元素是否在列表中

代码演示:

```
#判断指定元素是否在指定列表中
#成员运算符   in  not in
list6 = [32,43,546,"hello",False]
print(43 in list6)
print(43 not in list6)
print(100 in list6)
print(100 not in list6)
"""
工作原理：使用指定数据在列表中和每个元素进行比对，只要元素内容相等，则说明存在的
```

```
True
False
False
True
"""
```

1.4列表截取【切片】

代码演示:

```
#列表的截取
list7 = [23,34,6,57,6878,3,5,4,76,7]
print(list7[4])

#使用冒号:
#截取指定的区间：列表名[开始索引：结束索引],特点：包头不包尾      前闭后开区间
print(list7[2:6])

#从开头截取到指定索引，特点：不包含指定的索引
print(list7[0:6])
print(list7[:6])

#从指定索引截取到结尾
#注意：因为包头不包尾，所以如果要取到最后一个元素，可以超过索引的范围，不会报错
print(list7[4:20])
print(list7[4:])
```

5.列表的功能【掌握】

Python内置的功能【函数】

用法

代码演示:

```
#功能的使用：列表名.功能的名字()

#一、添加元素
#1.append()    追加，在列表的末尾添加元素
#特点：是在原列表的基础上操作的
list12 = [1,2,3,4,5]
print(list12)
#追加单个元素
```

```
list12.append(6)
#追加多个元素,不能直接追加，通过列表的形式追加，形成了一个二维列表
list12.append([7,8])
print(list12)

#2.extend()   扩展，在列表的末尾添加元素
#list12.extend(9)   TypeError: 'int' object is not iterable
list12.extend([9,10])
print(list12)

#注意：append可以添加单个元素，但是extend不可以
#append添加多个元素的时候，以整个列表的形式添加进去；但是，extend只添加元素

#3.insert()   插入 ,在指定的索引处插入一个元素,后面的其他元素向后顺延
#insert(索引，插入的数据)
list13 = [1,2,3,4,5]
print(list13)
#需求：在索引为2的位置插入一个数字100
list13.insert(2,100)
print(list13)
#将整个列表作为一个整体，插入到原列表中
list13.insert(2,[7,8])
print(list13)


#二、删除元素
#1.pop()    弹出，移除列表中指定索引处的元素
list14 = [1,2,3,4,5]
print(list14)
#注意1：默认移除的是最后一个元素
#注意2：返回的是被移除的数据
result14 = list14.pop()
print(list14)  #[1, 2, 3, 4]
print(result14)   #5

print(list14.pop(1))
print(list14)

#2.remove()  移除   特点;移除指定元素在列表中匹配到的第一个元素【从左往右】
#remove(元素值)
```

```python
list15 = [1,2,3,4,5,4,6,4]
print(list15)
list15.remove(4)
print(list15)

#3.clear()      清除  清除列表中的所有的元素，原列表变为空列表
list16 = [25,36,673]
print(list16)
list16.clear()
print(list16)


#三、获取
#直接使用功能：功能名称(列表)
#1.len() length,长度，获取列表的长度或者获取列表中元素的个数
list17 = [425.74,8,58679,7,65,65,64,6]
#索引的取值范围：0~len(list17) - 1
length = len(list17)
print(length)

#2.max()  获取列表中的最大值
print(max(list17))

#3.min() 获取列表中的最小值
print(min(list17))

#4.index()     索引,从列表中匹配到的第一个指定元素的索引值
#index(元素值)
list18 = [10,20,30,40,50,30,40,50]
inx1 = list18.index(30)
print(inx1)   #2

inx2 = list18.index(50)
print(inx2)   #4

#5.count()   个数，查找指定元素在列表中出现的次数
print(list18.count(50))   #2

#四、其他用法
#1.reverse()      反转，将列表中的元素倒序输出
```

```python
list19 = [10,20,30,40,50]
#注意;在列表的内部进行反转，并没有生成新的列表
list19.reverse()
print(list19)

#2.sort()    排序,默认为升序排序    注意：在列表的内部操作
list20 = [34,65,768,23]
#列表名.sort()
#升序
#list20.sort()
#降序
list20.sort(reverse=True)
print(list20)

#3.sorted()  排序,默认为升序排序    注意：生成一个新的列表
list21 = [34,65,768,23]
#升序
#list22 = sorted(list21)
#print(list22)
#降序
list23 = sorted(list21,reverse=True)
print(list23)

#按照元素的长度来进行排序
list00 = ["abc","hello","g","fhekfgjahgjkq"]
list24 = sorted(list00,key=len)
print(list24)


#4.拷贝【面试题】
# 赋值
list25 = [23,3,546]
list26 = list25
list26[1] = 100
print(list25)    #[23, 100, 546]
print(list26)    #[23, 100, 546]
print(id(list25))
print(id(list26))

#浅拷贝：内存的拷贝【实体，堆空间】
```

```
list27 = [23,3,546]
list28 = list27.copy()
list28[1] = 200
print(list27)
print(list28)
print(id(list27))
print(id(list28))

#深拷贝
list1 = [23,3,[4,5]]
list2 = copy.deepcopy(list1)


#练习：remove()
list30 = [23,435,5656,6767,435,23,23,54,64,5676,23,23,23]
#需求：移除列表中指定的所有的元素，例如：23
"""
list30.remove(23)
print(list30)
list30.remove(23)
print(list30)
list30.remove(23)
print(list30)
list30.remove(23)
print(list30)
list30.remove(23)
print(list30)
"""
#定义一个变量，用于记录元素的位置【索引】
#问题：remove功能是在列表的内部操作的
num = 0
#length = len(list30)
all  = list30.count(23)
while num < all:
        #依据：remove每次删除的第一次匹配的元素【从左到右】
        list30.remove(23)
        num += 1
print(list30)
```

6.二维列表

一个列表的元素是一个列表

代码演示：

```
#一维列表
list1 = [1,23,5,346]
#二维列表
list2 = [[543,54,6],[234,35,46,4565,767],[65,65,65,565]]

# 图：三维列表
# 视频：四维列表

#处理二维列表：当做一个一维列表使用
subList = list2[1]
print(subList)
print(subList[2])
```

二、布尔值和空值

1.布尔值

一个布尔类型的变量一般有两个值，True,False

作用：用于分支和循环语句中作为条件判断

代码演示：

```
#Boolean
b1 = True
b2 = False

#条件表达式结果布尔值
print(4 > 5)
print(1 and 0)
```

2.空值

Python中的一种特殊的数据类型，使用None表示

区别与0：0是数字类型，None本身就是一种数据类型

代码演示：

```
#空值
```

```
n = None
print(n)   #None
```

三、数字类型Number

1.数字类型分类

1.1整数

可以处理Python中任意大小的整型

代码演示:

```
num1 = 10
num2 = num1
print(num1,num2)

#1.可以连续定义多个同种类型的变量,初始值相同
num3 = num4 = num5 = 100

#2.同时定义多个变量，初始值不同
num6,num7 = 60,70
print(num6,num7)

#3.可以交换两个变量的值【掌握】
#自己实现
nn1 = 22
nn2 = 33

temp = nn1
nn1 = nn2
nn2 = temp
print(nn1,nn2)

n1 = 22
n2 = 33
print(n1,n2)   #22  33

n1,n2 = n2,n1
print(n1,n2)

#4.获取变量在内存中的地址
```

```
print(id(num1),id(num2))
```

1.2浮点数

由整数部分和小数部分组成

注意：浮点数在计算机中运算的时候可能会出现四舍五入

2.系统功能

2.1数学功能

abs(x): absolute 求x的绝对值

max(): 求最大值

min()：求最小值

pow(n,m):求一个数的多少次幂

round(x，n):返回浮点数x的四舍五入值,如果给出n值，则表示舍入到小数点后几位

代码演示：

```
print(abs(-10))

print(max(23,34,6,56,57,6))
print(min(23,34,6,56,57,6))

print(pow(3,5))  // **

print(round(3.456))   #3
print(round(3.656))   #4
print(round(3.656,2))  #3.66
print(round(3.646,1))   #3.6
```

导入math模块，math.功能名()

代码演示：

```
#以下的功能必须导入math模块
import  math

#使用格式：math.功能名称

#19向上取整
print(math.ceil(18.1))
print(math.ceil(18.9))

```

```
#18向下取整
print(math.floor(18.1))
print(math.floor(18.9))

#求平方
print(pow(3,2))
#开平方【掌握】
print(math.sqrt(9))

#获取整数部分和小数部分，得到的结果为元组
print(math.modf(22.3))
```

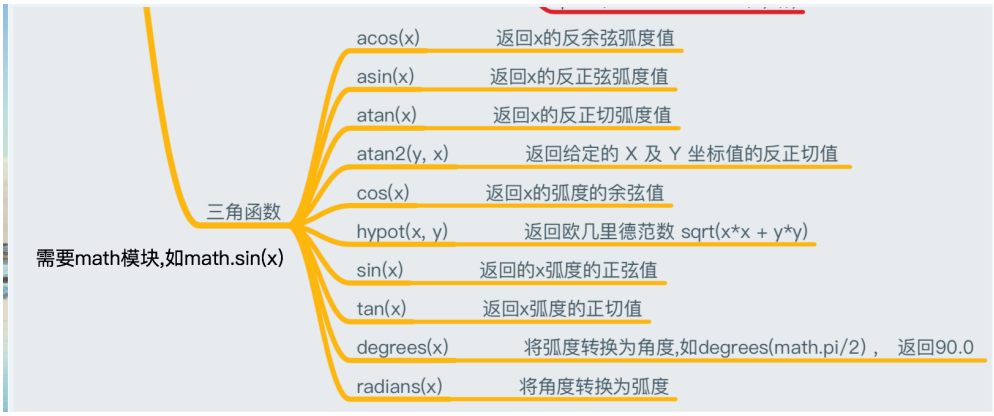
2.2随机数random【掌握】

代码演示:

```
import random

#1.random.choice(列表)  从指定列表中随机选择一个元素出来
#指定列表
num1 = random.choice([1,3,5,7,9])
print(num1)

#列表生成器
num2 = random.choice(range(5))   #等价于[0,1,2,3,4]
print(num2)

#使用字符串，相当于使用了元素为字母的列表
num3 = random.choice("hello")  #等价于["h","e","l","l","o"]
print(num3)

#需求;产生一个4~10之间的随机数
print(random.choice([4,5,6,7,8,9,10]))
print(random.choice(range(4,11)))

#2.random.randrange(start,end,step)
"""
start:指定范围的开始值，包含在范围内，默认为0
end:指定范围的结束值，不包含在范围内
step:步长，指定的递增基数，默认为1
"""
```

```
26
27 #需求1：从1~100之间选取一个奇数随机数
28 print(random.choice(range(1,100,2)))
29 print(random.randrange(1,100,2))
30 #需求2：生成一个0~99之间的随机数
31 print(random.randrange(100))
32
33 #3.random.random()    获取0~1之间的随机数，结果为浮点型
34 n = random.random()
35 #需求：保留小数点后两位
36 print(round(n,2))
37
38 #需求1：获取4~10之间的随机数
39 n1 = random.random() * 6 + 4
40 """
41 [0,1] * 6 --------->[0,6]
42 [0,6] + 4 -------->[4,10]
43 """
44
45 #4.将列表中的元素进行随机排序【了解】
46 list1 = [23,5435,4,6]
47 random.shuffle(list1)
48 print(list1)
49
50 #5.随机生成一个实数，它在[3,9]范围内，结果为浮点型
51 print(random.uniform(3,9))
52
53 #需求：求50~100之间的随机数，包括浮点数
54 n2 = random.uniform(50,100)
```

2.3三角函数功能【了解】



Day05-元组&字典&集合

一、tuple元组

1.概述

和列表相似，本质上是一种有序的集合

元组和列表的不同之处：

a.列表:[ ] 元组：( )

b.列表中的元素可以进行增加和删除操作，但是，元组中的元素不能修改【元素：一旦被初始化，将不能发生改变】

2.创建元组

创建列表:

创建空列表：list1 = [ ]

创建有元素的列表：list1 = [元素1，元素2，。。。。。]

创建元组

创建空元组：tuple1 = ( )

创建有元素的元组：tuple1 = (元素1，元素2，。。。。)

代码演示：

```
#创建空元组：
tuple1 = ()

#创建有元素的元组：
tuple2 = (10,20,30)

#在元组中可以存储不同类型的数据
tuple3 = ("hello",True,100)
```

```
9
10 #注意：创建只有一个元素的元组
11 #按照下面的方式书写，表示定义了一个整型的变量，初始值为1
12 tuple4 = (1)
13 tuple4 = 1
14 #为了消除歧义，修改如下：
15 tuple4 = (1,)
16
17 num1 = 10
18 num2 = (10)
19 print(num1,num2)
```

3.元组元素的访问

代码演示：

```
1 #元组元素的访问
2 #格式：元组名[索引],和列表完全相同
3 tuple1 = (10,20,30,40,50)
4 #1.获取元素值
5 print(tuple1[2])
6 #获取元组中的最后一个元素
7 print(tuple1[4])
8 #print(tuple1[5])  #IndexError: tuple index out of range  索引越界
9
10 #正数表示从前往后获取，负数表示从后往前获取
11 print(tuple1[-1])
12 print(tuple1[-2])
13 print(tuple1[-5])
14 # print(tuple1[-6])   #IndexError: tuple index out of range  索引越界
15
16 #2.修改元素值----->不能修改，本质原因不能修改元素的地址
17 #和列表不同的地方：元组的元素值不能随意的更改
18 #tuple1[1] = 100
19 tuple2 = (1,35,54,[4,5,6])
20 #获取元组中列表中的元素
21 print(tuple2)   #(1, 35, 54, [4, 5, 6])
22 tuple2[3][1] = 50
23 print(tuple2)  #(1, 35, 54, [4, 50, 6])
24
```

```
#3.删除元组
tuple3 = (53,6,7,76)
del tuple3
```

4.元组操作

代码演示:

```
#1.元组组合
#+
tuple1 = (3,43,5,4)
tuple2 = (3,5,45,4)
print(tuple1 + tuple2)

#2.元组重复
#*
print(tuple1 * 3)

#注意：元组组合和元组重复得到的是一个新的元组，原来的元组并没有发生任何改变

#3.判断元素是否在元组中
#成员运算符
#in    not in
print(100 in tuple1)
print(100 not in  tuple1)

#4.元组截取【切片】
tuple3 = (1,23,43,54,54,656,57,6)
print(tuple3[2:4])
print(tuple3[2:])
print(tuple3[:4])
```

5.元组功能

代码演示:

```
#1.获取元组的元素个数
tuple1 = (54,3,5,46,56)
print(len(tuple1))

#2.获取元组中元素的最大值和最小值
```

```
print(max(tuple1))
print(min(tuple1))


#3.元组和列表之间的相互转换:取长补短
#3.1    元组-----》列表
#list()
list1 = list(tuple1)    #int()   float()
print(list1)

#3.2  列表------》元组
#tuple()
list2 = [34,5,46,4]
tuple2 = tuple(list2)
print(tuple2)

#4.遍历元组
#4.1直接遍历元素
for n in tuple1:
        print(element)

#4.2遍历索引
for index in range(len(tuple1)):
        print(tuple1[index])

#4.3同时遍历索引和元素 enumerate枚举
for i,num in enumerate(tuple1):
        print(i,num)
```

6.二维元组

代码演示:

```
#当做一维元组进行处理，实质：一维元组中的元素为一个一维元组
tuple1 = ((2,43,5),(54,65,6),(5,54,54,54))
print(tuple1[1][1])

#遍历二维列表或者二维元组的思路：嵌套循环
#遍历外层元组
for element in tuple1:
```

```
        #遍历内层元组
        for num in elment:
            print(num)
```

二、字典dict

1.概念

列表和元组的使用缺点：

当存储的数据要动态添加、删除的时候，我们一般使用列表，但是列表有时会遇到一些麻烦

```
# 定义一个列表保存，姓名、性别、职业
nameList = ['尼古拉斯.赵四',  '铁憨憨'];

# 当修改职业的时候，需要记忆元素的下标
nameList[2] = '演员'

# 如果列表的顺序发生了变化，添加年龄
nameList = ['尼古拉斯.赵四', 18, '男',  '铁匠']

# 此时就需要记忆新的下标，才能完成名字的修改
nameList[3] = 'xiaoxiaoWang'
```

解决方案：既能存储多个数据，还能在访问元素的很方便的定位到需要的元素，采用字典

语法： {键1: 值1, 键2: 值2, 键3: 值3, ..., 键n: 值n}

说明：键值对: key-value

- 字典和列表类似，都可以用来存储多个数据
- 在列表中查找某个元素时，是根据下标进行的；字典中找某个元素时，是根据'名字'（就是冒号:前面的那个值，例如上面代码中的'name'、'id'、'sex'）
- 字典中的每个元素都由2部分组成，键:值。例如 'name':'班长' ,'name'为键，'班长'为值
- 键可以使用数字、布尔值、元组，字符串等不可变数据类型，但是一般习惯使用字符串，切记不能使用列表等可变数据类型
- 每个字典里的key都是唯一的，如果出现了多个相同的key,后面的value会覆盖之前的value

习惯使用场景：

- 列表更适合保存相似数据，比如多个商品、多个姓名、多个时间
- 字典更适合保存不同数据，比如一个商品的不同信息、一个人的不同信息

2.定义字典

```
#语法：字典名 = {key1:value1,key2:value2.....}
```

```
2
3  #1.创建空字典
4  dict1 = {}
5  print(dict1,type(dict1))
6
7  #2.创建非空字典
8  #方式一
9  dict21 = {"name":"张三","age":18}
10 print(dict21)
11
12 #方式二
13 #dict(key=value),key是一个变量名，value是一个值
14 dict22 = dict(a="avvv",b="2353")
15 print(dict22)
16 dict22 = dict(a=200,b=33)
17 print(dict22)
18
19 #方式三
20 #dict()和zip(序列),zip表示映射
21 #dict(zip([key1,key2,key3....],[value1,value2,value3....]))
22 #注意：key的数量和value的数量可以不一致，以少的作为参考
23 z1 = zip([1,2],["a","b","c"])
24 dict23 = dict(z1)
25 print(dict23)
26
27 dict23 = dict(zip(("name","age"),("aaa",10)))
28 print(dict23)
29
30 dict23 = dict(zip("xyz","abc"))
31 print(dict23)
32
33 #方式四
34 # [(key1,value1), (key2,value2)...] => {key1:value1, key2:value2....}
35 dict24 = dict([("a",10),("b",20),("c",30)])
36 print(dict24)
```

三、set集合【了解】

1.概述

和数学上的集合基本是一样的，
特点:不允许有重复元素，可以进行交集，并集，差集的运算
本质：无序，无重复元素的集合

2.创建

set(列表或者元组或者字典)
代码演示：

```
#注意：set的创建需要借助于list和tuple

#1.通过list创建set
list1 = [432,5,5,46,65]
s1 = set(list1)
print(list1)
print(s1)

#注意1：set中会自动将重复元素过滤掉

#2.通过tuple创建set
tuple1 = (235,45,5,656,5)
s2 = set(tuple1)
print(tuple1)
print(s2)

#3.通过dict创建set
dict1 = {1:"hello",2:"good"}
s3 = set(dict1)
print(dict1)   #{1: 'hello', 2: 'good'}
print(s3)   #{1, 2}

#注意2：set跟dict类似，都使用{}表示，但是与dict之间的区别在于：set中相当于只存储了一组key，没有value
```

3.操作

3.1添加

代码演示：

```
#1.添加
#add()   在set的末尾进行追加
```

```python
s1 = set([1,2,3,4,5])
print(s1)
s1.add(6)
print(s1)

#注意：如果元素已经存在，则添加失败
s1.add(3)
print(s1)
#print(s1.add(3))

#s1.add([7,8,9])   #TypeError: unhashable type: 'list'  list是可变的，set中的元素不能是list类型
s1.add((7,8,9))
#s1.add({1:"a"})  #TypeError: unhashable type: 'dict'  ，dict中的键值对可以改变，set中的元素不能是dict类型
print(s1)

#update()   插入【末尾添加】，打碎插入【直接将元组，列表中的元素添加到set中，将字符串中的字母作为小的字符串添加到set中】
s2 = set([1,2,3,4,5])
print(s2)
s2.update([6,7,8])
s2.update((9,10))
s2.update("good")
#注意：不能添加整型，因为整型不能使用for循环遍历
#s2.update(11)   #TypeError: 'int' object is not iterable
print(s2)
```

3.2删除

代码演示：

```python
#2.删除
#remove()
s3 = set([1,2,3,4,5])
print(s3)
s3.remove(3)
print(s3)
```

3.3遍历

代码演示：

```python
#3.set的遍历
s4 = set([1,2,3,4,5])
for i in s4:
 print(i)

#注意：set是没有索引的，所以不能通过s4[2]获取元素，原因：set是无序的
#print(s4[2])  #TypeError: 'set' object does not support indexing

#注意：获取的是编号和元素值
for i,num in enumerate(s4):
 print(i,num)
```

3.4交集和并集

代码演示：

```python
#4.交集和并集
s4 = set([1,2,3])
s5 = set([4,5,3])

#交集：&【按位与】    and
r1 = s4 & s5
print(r1)
print(type(r1))

#并集:|【按位或】    or
r2 = s4 | s5
print(r2)
```

Day06-字符串

一、string字符串【掌握】

1.概述

由多个字母，数字，特殊字符组成的有限序列

在Python中，使用单引号或者双引号都可以表示字符串

注意:没有单符号的数据类型

'a' "a"

2.创建字符串

代码演示：

```
str1 = "hello"
str2 = "abc1234"
str3 = "***fhhg%%%"
str4 = "中文"
```

3.字符串运算

代码演示：

```
#1.+   字符串连接
s1 = "welcome"
s2 = " to China"
print(s1 + s2)

#注意：在Python中，使用+。只能是字符串和字符串之间。和其他数据类型使用的话不支持
#print("abc" + 10)
#print("123" + 1)
#print(1 + "12" + 12)
#print("hello" + True)

#2. *   字符串重复
s3 = "good"
print(s3 * 3)

#3.获取字符串中的某个字符
"""
类似于列表和元组的使用，通过索引来获取指定位置的字符
注意索引的取值范围【0~长度-1】，同样会出现索引越界
访问方式：字符串名称[索引]
"""
s4 = "abcdef"
print(s4[1])
#print(s4[10])  #IndexError: string index out of range

#获取字符串的长度：len()
#遍历字符串,和list，tuple的用法完全相同
```

```
for element in s4:
        print(element)
for index in range(0,len(s4)):
        print(s4[index])
for index,str in enumerate(s4):
        print(index,str)

#4.截取字符串【切片】
str1 = "hello world"
#指定区间
print(str1[3:7])
#从指定位置到结尾，包含指定位置
print(str1[3:])
#从开头到指定位置，但是不包含指定位置
print(str1[:7])

str2 = "abc123456"
print(str2[2:5]) #c12
print(str2[2:])  #c123456
print(str2[2::2])  #c246
print(str2[::2])   #ac246
print(str2[::-1])  #654321cba    倒序
print(str2[-3:-1])  #45    -1表示最后一个字符

#5.判断一个子字符串是否在原字符串中
#in,  not in
str3 = "today is a good day"
print("good"  in str3)
print("good1"  not in str3)
```

4.格式化输出

通过%来改变后面字母或者数字的含义，%被称为占位符

%d 整数

%f 浮点型，特点：可以指定小数点后的位数

%s 字符串

代码演示：

```
#6.格式化输出
num = 10
```

```
string1 = "hello"
print("string1=",string1,"num=",num)
#注意：变量的书写顺序尽量和前面字符串中出现的顺序保持一致
print("string1=%s,num=%d"%(string1,num))

f = 12.247
print("string1=%s,num=%d,f=%f"%(string1,num,f))
#需求：浮点数保留小数点后两位
print("string1=%s,num=%d,f=%.2f"%(string1,num,f))    #round(12.247,2)
```

5.常用转义字符

通过\来改变后面字母或者特殊字符的含义

\t 相当于tab键

\n 相当于enter键

\b 相当于backspace

代码演示:

```
#7.转义字符
string2 = "hello\tworld"
string21 = "hello    world"
print(string2)
print(string21)

#换行：\n    多行注释
string3 = "hello\nPython"
string31 = """hello
python2354623
"""
print(string3)
print(string31)

#需求："hello"
print("\"hello\"")

#C:\Users\Administrator\Desktop\SZ-Python1805\Day6\视频
print("C:\\Users\\Administrator\\Desktop")
#注意;如果一个字符串中有多个字符需要转义，则可以在字符串的前面添加r,可以避免对字符串中的每个特殊字符进行转义
print(r"C:\Users\Administrator\Desktop")
```

6.常用功能【掌握】

6.1获取长度和次数

代码演示:

```
#1.计算字符串长度  len
#类似于list和tuple的中获取长度的用法
str1 = "hfufhja"
l = len(str1)
print(l)

#2,计算某个字符或者子字符串在原字符串中出现的次数   count
str2 = "this is a good day good day"
#count(str,[start,end])
#在整个字符串中进行查找
print(str2.count("day"))
#在指定区间内进行查找
print(str2.count("day",3,10))
```

6.2大小写转换

代码演示:

```
#注意：使用字符串中的功能，一般情况下，都是生成一个新的字符串，原字符串没有发生任何变化
#3.大小写字母转换
#lower()   将字符串中的大写字母转换为小写
str31 = "Today Is a Good day"
astr31 = str31.lower()
print(astr31)

#upper()   将字符串中小写字母转换为大写
str32 = "Today Is a Good day"
astr32 = str2.upper()
print(astr32)

#swapcase()     将字符串中小写字母转换为大写，大写字母转换为小写
str33 = "Today Is a Good day"
astr33 = str33.swapcase()
print(astr33)

```

```
#capitalize()   将一句英文中首单词的首字母转化为大写，其他小写
str34 = "today Is a Good day"
astr34 = str34.capitalize()
print(astr34)

#title()       将一句英文中每个单词的首字母大写
str35 = "today is a good day"
astr35 = str35.title()
print(astr35)
```

6.3整数和字符串转换

代码演示:

```
#4.字符串和数字之间的转换
#int()     float()      str()
#eval(str)   将str转换为有效的表达式，参与运算，并返回运算结果
num1 = eval("123")
print(num1)
#print("123")
print(type(num1))
print(int("123"))

#eval和int将+和-当做正负号处理
print(eval("+123"))
print(int("+123"))
print(eval("-123"))
print(int("-123"))

#将12+3字符串转换为了有效的表达式，并运算了结果
print(eval("12+3"))    #15
#不成立
#print(int("12+3"))   #ValueError: invalid literal for int() with base 10: '12+3'

print(eval("12-3"))   #9
#print(int("12-3"))    #ValueError: invalid literal for int() with base 10: '12-3'

#print(eval("a123"))  #NameError: name 'a123' is not defined
#print(int("a123"))  #ValueError: invalid literal for int() with base 10: 'a123'

#总结：注意区分eval和int【eval：转换有效的表达式    int:将字符串转换为整型】
```

6.4填充

代码演示:

```
#5.填充【了解】
#center（width[,fillchar]）  返回一个指定宽度的居中字符串，width是填充之后整个字符串的长度，fillchar为需要填充的字符串，默认使用空格填充
str1 = "hello"
print(str1.center(20))
print(str1.center(10,"*"))

#ljust（width[,fillchar]） 返回一个指定宽度的字符串，将原字符串居左对齐，width是填充之后整个字符串的长度
print(str1.ljust(40,"%"))

#rjust width[,fillchar]）  返回一个指定宽度的字符串，将原字符串居右对齐，width是填充之后整个字符串的长度
print(str1.rjust(40,"%"))

#zfill（width）   返回一个指定宽度的字符串,将原字符串居右对齐,剩余的部分使用的数字0填充
print(str1.zfill(40))
```

6.5查找

代码演示:

```
#6.查找【掌握】
str2 = "abcdefhello123hello"
#find（str[,start,end]）  从左到右依次检测，str是否在原字符串中，，也可以指定查找的范围
#特点;得到的子字符串第一次出现的开始字符的下标，如果查找不到则返回-1
print(str2.find("hello"))    #6
print(str2.find("e"))
print(str2.find("yyy"))    #-1
print(str2.find("e",3,10))

#rfind(str[,start,end]）   类似于find，从右向左进行检测
print(str2.rfind("hello"))  #14

#index   和find的使用基本相同，唯一的区别在于如果子字符串查找不到，find返回-1，而index则直接报错
print(str2.index("hello"))
```

```
#print(str2.index("yyy"))   #ValueError: substring not found

#rindex  和rfind的使用基本相同

#max(str)   获取str中最大的字母【在字典中的顺序】
#"abcdefhello123hello"
print(max(str2))

str3 = "46732647"
print(max(str3))

#min（str）  获取str中最小的字母【在字典中的顺序】
```

6.6提取

代码演示:

```
#7.提取字符串
#strip(str)    使用str作为条件提取字符串，除了两头指定的字符串
str1 = "********today is *********a good day*******"
print(str1.strip("*"))   #today is *********a good day

#lstrip(str)    提取字符串，除了左边的指定字符串
str11 = "********today is *********a good day*******"
print(str11.lstrip("*"))

#rstrip()
str12 = "********today is *********a good day*******"
print(str12.rstrip("*"))
```

6.7分割和合并

```
#8.分割和合并【掌握】
#split(str[,num)]
#将str作为分隔符切割原字符串，结果为一个列表,如果制定了num，
#则仅使用num个字符串截取原字符串
str3 = "today is a good day"
print(str3.split(" "))
#['today', 'is', 'a', 'good', 'day']
print(str3.split(" ",2))
```

```
#['today', 'is', 'a good day']



#splitlines(flag)    按照换行符【\n，\r,\r\n】分隔，结果为列表
#flag:False或者不写，则表示忽略换行符；如果True，则表示保留换行符
str4 = """today
is
a
good
day
"""
print(str4.splitlines(True))
#['today', 'is', 'a', 'good', 'day']
#['today\n', 'is\n', 'a\n', 'good\n', 'day\n']

#join(list)
#将原字符串作为连接符号，将列表中的元素分别连接起来，结果为字符串，作用和split是相反的
str5 = "*"
list1 = ["shangsan","lisi","jack"]
print(str5.join(list1))
```

6.8替换

代码演示：

```
#9.替换
#replace(old,new[,max])    用new的字符串将old的字符串替换掉.max表示可以替换的最大次数【从左到右】
str1 = "this is a easy test test test test"
print(str1.replace("test","exam"))
print(str1.replace("test","exam",2))

#使用场景：在一定情境下，可以实现字符串的简单加密，加密规则可以自定义
#maketrans()    创建字符映射的转换表,结果为字典，通过key:value的方式
#translate(table)

t = str.maketrans("aco","123")
print(t)    #{97: 49, 99: 50, 111: 51}

str2 = "today is a good day"
```

```
print(str2.translate(t))  #t3d1y is 1 g33d d1y
```

6.9判断

代码演示：

```
#10.判断
#isalpha()   如果字符串中至少包含一个字符并且所有的字符都是字母，才返回True
print("".isalpha())
print("abc".isalpha())
print("abc123".isalpha())   #False

#isalnum   如果字符串中至少包含一个字符并且所有字符都是字母或者数字的时候才返回True
print("".isalnum())   #False
print("abc".isalnum())
print("abc123".isalnum())
print("123".isalnum())
print("1abc".isalnum())
print("1abc￥".isalnum())  #False

#isupper  如果字符串中至少包含一个字符并且出现的字母必须是大写字母才返回True，数字的出现没有影响
print("".isupper())
print("aBC".isupper())
print("123A".isupper())   #True
print("abc".isupper())

#islower

#istitle   每个单词的首字母必须全部大写才返回True
print("Good Day".istitle())
print("good Day".istitle())

#isdigit() 【掌握】   如果字符串中只包含数字，则返回True
print("abc123".isdigit())
print("2364".isdigit())

#需求：将用户从控制台输入的字符串转化为整型【全数字】
str = input()
if str.isdigit():
 int(str)
```

```
 print("yes")

```

6.10前缀和后缀

代码演示:

```
#11.前缀和后缀【掌握】  子字符串是连续的
#startswith
str1 = "helloPython"
print(str1.startswith("hello"))

#endswith
print(str1.endswith("on"))
```

6.11编解码

代码演示:

```
#12.字符串编码和解码
#注意：主要针对的是中文
#encode()   默认的编码格式为utf-8
str2 = "this is 千锋教育"
print(str2.encode())
print(str2.encode("utf-8"))
print(str2.encode("gbk"))

#decode()   bytes对象
#\xe5\x8d\x83\xe9\x94\x8b\xe6\x95\x99\xe8\x82\xb2
#print(r"\xe5\x8d\x83\xe9\x94\x8b\xe6\x95\x99\xe8\x82\xb2".decode())    错误
```

6.12ASCII码转换

代码演示:

```
#13.ASCII吗的转换
#ord()
print(ord("A"))
print(ord("0"))

#chr()
print(chr(65))
```

```
print(chr(110))
```

Day07

一、函数

1.函数概述

1.1认识函数

需求: 求圆的面积

$s = \pi r^2$

```
# 勾股定理
c = math.sqrt(a**2 + b**2)
```

代码演示:

```
r1 = 6.8
s1 = 3.14 * r1 ** 2

r2 = 10
s1 = 3.14 * r2 ** 2

r3 = 2
s1 = 3.14 * r3 ** 2

r4 = 30
s1 = 3.14 * r4 ** 2

# 函数/公式
f(x, y) = 2x + y + 1
f(1,0) = 2*1 + 0 + 1


# define
def test(r):
        s = 3.14 * r ** 2
 print(s)

```

```
test(6.8)
test(10)
test(30)
```

问题: 代码重复
后期维护成本太高
代码可读性不高
解决问题：函数
在一个完整的项目中，某些功能会被反复使用，那么将这部分功能对应的代码提取出来，当需要使用功能的时候直接使用
本质：对一些特殊功能的封装
优点：
a.简化代码结构，提高应用的效率
b.提高代码复用性
c.提高代码的可读性和可维护性
建议：但凡涉及到功能，都尽量使用函数实现

1.2定义函数

语法：
def 函数名(参数1，参数2，参数3....):
函数体
返回值
说明：
a.函数由两部分组成：声明部分和实现部分
b.def,关键字，是define的缩写，表示定义的意思
c.函数名：类似于变量名，遵循标识符的命名规则，尽量做到顾名思义
d.（）：表示的参数列表的开始和结束
e.参数1，参数2，参数3.... ：参数列表【形式参数，简称为形参】，其实本质上就是一个变量名，参数列表可以为空
f.函数体：封装的功能的代码
g.返回值：一般用于结束函数，可有可无，如果有返回值，则表示将相关的信息携带出去，携带给调用者，如果没有返回值，则相当于返回None

2.使用函数

2.1简单函数

无参无返回值的函数
代码演示：

```python
#1.无参无返回值的函数
#函数的声明部分
def test():
 #函数的实现部分
 #函数体
   print("hello")
```

2.2函数的调用

定义好函数之后，让函数执行

格式：函数名(参数列表)

代码演示：

```python
#print(num)
#test()

#1.无参无返回值的函数
#函数的声明部分
def test():
 #函数的实现部分
 #函数体
 #print("hello")
 for i in range(10):
     print(i)

def test():
 print("~~~~~~")

#注意1：当定义好一个函数之后，这个函数不会自动执行函数体

#2.函数的调用
#格式：函数名(参数列表)
#注意2：当调用函数的时候，参数列表需要和定义函数时候的参数列表保持一致
#注意3：一个函数可以被多次调用
test()
test()
test()
test()

#3.注意4：当在同一个py文件中定义多个同名的函数，最终调用函数，调用的最后出现的函数【覆盖：函数名类似于变量名，相当于变量的重新赋值】
```

```
#4.注意5：自定义函数必须先定义，然后才调用，否则报NameError
```

函数的调用顺序：

```
#函数调用
#1.在一个自定义的函数内部也可以调用函数
#2.函数调用的顺序
def test1():
     print("aaaa")
     test2()
     print("over")

def test2():
     print("bbbb")
     test3()
     test4()

def test3():
    print("cccc")

def test4():
     print("dddd")

test1()

#注意：函数在调用的过程中，相互之间的关系，以及代码执行的先后顺序
```

2.3函数中的参数

参数列表：如果函数所实现的功能涉及到未知项参与运算，此时就可以将未知项设置为参数

格式：参数1,参数2.....

分类：

形式参数：在函数的声明部分，本质就是一个变量，用于接收实际参数的值 【形参】

实际参数：在函数调用部分，实际参与运算的值，用于给形式参数赋值 【实参】

传参：实际参数给形式参数赋值的过程，形式参数 = 实际参数

代码演示：

```
#传参：实际参数给形式参数赋值的过程，形式参数 = 实际参数

#需求：给函数一个姓名和一个年龄，在函数内部将内容打印出来
def myPrint(name,age):
```

```
    print("姓名：%s,年龄：%d"%(name,age))

#调用函数
str = "zhangsan"
num = 19
myPrint(str,num)

"""
传参：
实参给形参赋值
name = "zhangsan"
age = 19
"""

#需求：求两个数的和
def add(num1,num2):
    sum = num1 + num2
    print(sum)

add(10,20)
add(33,2)

#TypeError: add() missing 2 required positional arguments: 'num1' and 'num2'
#实参和形参不匹配
```

2.4值传递和引用传递【面试题】

值传递：传参的过程中传递的是值，一般指的是不可变的数据类型，number,tuple,string

引用传递：传参的过程中传递的是引用，一般指的是可变的数据类型，list，dict, set

代码演示：

```
#值传递
def func1(a):  # a = temp
 a = 10

temp = 20
#传参：temp,但实际上传的是20
func1(temp)
print(temp)   #20

```

```
a = 10
b = a

#引用传递
def func2(list1):
    list1[0] = 100

l = [10,20,30,40]
func2(l)    #list1 = l
print(l[0])


"""
l = [10,20,30,40]
list1 = l
list1[0] = 100

"""
```

总结:

引用传递本质上传递的是内存地址

2.5参数的类型【掌握】

a.必需参数

调用函数的时候必须以正确的顺序传参，传参的时候参数的数量和形参必须保持一致

代码演示:

```
#1.必需参数
def show1(str1,num1):
 print(str)

show1("hello",10)
#show1()
#如果形参没有任何限制，则默认为必需参数，调用函数的时候则必需传参，顺序一致，数量一致
```

b.关键字参数

使用关键字参数允许函数调用的时候实参的顺序和形参的顺序可以不一致，可以使用关键字进行自动的匹配

代码演示:

```
#2.关键字参数
```

```python
def show2(name,age):
    age += 1
    print(name,age)

#正常调用
show2("abc",10)
#show2(10,"abc")

#关键字参数调用函数
#注意1：关键字参数中的关键字其实就是形参的变量名，通过变量名进行传参
show2(age = 20,name = "lisi")
show2(name = "lisi",age = 20)

#注意2：关键字参数只有一个的情况下，只能出现在参数列表的最后
show2("lisi",age = 30)

#错误演示
#show2(40,name = "lisi")   TypeError: show2() got multiple values for argument 'name'
#show2(name = "lisi",40)

#系统的关键字参数
print("",end=" ")
```

c.默认参数

调用函数的时候，如果没有传递参数，则会使用默认参数

代码演示：

```python
#3.默认参数
#注意1：在形参设置默认参数，如果传参，则使用传进来的数据，如果不传参，则使用默认数据
def fun1(name,age=18):
    print(name,age)

fun1("zhangsan",20)
fun1("lisi")
fun1(name = "abc",age = 33)
fun1(name = "hello")

#注意2：在参数列表中，如果所有的形参都是默认参数，正常使用；但是，如果默认参数值只有一个，则只能出现在参数列表的最后面
def fun2(num1 = 10,num2 = 20):
    print(num1.num2)
```

d.不定长参数(可变参数)

可以处理比当初声明时候更多的参数 * **

代码演示:

```
#4.不定长参数【可变参数】
#4.1   *   : 被当做tuple处理，变量名其实就是一个元组名
#注意1: 传参的时候，实参可以根据需求任意传参,数量不确定
#注意2:定义不定长参数时，最好将不定长参数放到参数列表的最后面
#【如果不定长参数出现在参数列表的前面，则在实参列表中使用关键字参数】
def func1(name,*hobby):
    print(name)
    print(hobby)
    print(type(hobby))   #<class 'tuple'>

 #遍历
    for element in hobby:
        print(element)

func1("aaa","anc","aaa","5435","tesrg","gtsrhesh",10,True)

# 4.2  **   :被当做字典处理，变量名就相当于字典名
def func2(**args):
    print(args)
    print(type(args))   #<class 'dict'>

    for k,v in args.items():
        print(k,v)

#注意1: 使用**的时候，实参就必须按照key=value的方式进行传参
func2(x = 10,y = 20)
```

2.6函数的返回值

作用：表示一个函数执行完毕之后得到的结果

使用:return,表示结束函数，将函数得到的结果返回给调用者

代码演示:

```
#1.结束函数，返回数据
#需求：求两个整数的和，并返回
def add(num1,num2):
```

```
    sum1 = num1 + num2
    #print(sum1)

    #将结果返回给调用者
    return sum1

#注意：在同一个代码块中，如果在return后面出现语句，则永远不会被执行
    print("hello")


#注意：如果一个函数由返回值，要么采用变量将返回值接出来，要么将整个函数的调用直接参与运算
r = add(10,20)
print(r)
print(add(10,20))   #30
#print("~~~~",sum1)

total = add(1,2) + 5
print(total)    #8


def func(num1,num2):
    sum2 = num1 + num2


#注意：如果一个函数没有返回值，则整体计算的结果为None
#print(func(10,20))

#如果一个函数没有返回值，则这个函数的调用不能直接参与运算
total1 = func(1,2) + 5  #TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
print(total1)
```

在分支语句中使用return

```
#2.如果一个函数体中有分支，设置了返回值，最好每一个分支都有一个返回值
#需求：输入两个数，比较两个数的大小，返回较大的一个
def compare(num1,num2):
    if num1 > num2:
        return num1
    elif num1 < num2:
        return num2
```

```
    else:
        return True,num1

result = compare(12,12)
print(result)

#注意1：在Python中，不同分支返回的数据类型可以是不相同的
#注意2;在Python中，一个return可以同时返回多个数据，被当做元组处理
```

总结：

自定义一个函数是否需要设置参数：是否有未知项参与运算

是否需要设置返回值：是否需要在函数外面使用函数运算之后的结果

函数使用练习：

```
#需求1：封装函数功能，统计1~某个数范围内能被3整除的数的个数
"""
参数：某个数
返回值：可设置可不设置
"""
def getCount(num):
    count = 0
   for i in range(num + 1):
        if i % 3 == 0:
            count += 1

            #print(count)
   return count

r1 = getCount(1000)
print(r1)
r2 = getCount(100)
print(r2)


#需求2：封装函数功能，判断某年是否是闰年
"""
参数：某年
返回值：可设置可不设置
"""
```

```python
def isLeapYear(year):
    if (year % 4 == 0 and year % 100 != 0) or year % 400 == 0:
        #print("闰年")
        #return "闰年"
        return  True
    else:
        #print("平年")
        #return "平年"
        return False

result = isLeapYear(2020)
print(result)
```

3.匿名函数【掌握】

不再使用def这种的形式定义函数，使用lambda来创建匿名函数

特点:

a.lambda只是一个表达式，比普通函数简单

b.lambda一般情况下只会书写一行，包含参数，实现体，返回值

语法:lambda 参数列表 ： 实现部分

代码演示:

```python
#语法:lambda 参数列表 ： 实现部分

#1.
#需求：求两个数的和
#普通函数
def add(num1,num2):
 sum = num1 + num2

add(num1 = 10,num2 = 20)

#匿名函数本身是没有函数名，将整个lambda表达式赋值给一个变量，然后将这个变量当做函数使用
sum1 = lambda n1,n2:n1 + n2
print(sum1(10,20))

#2.在匿名函数中也可以使用关键字参数
g = lambda  x,y:x ** 2 + y ** 2
print(g(3,4))
print(g(x = 3,y = 4))
```

```
19
20 #3.在匿名函数中也可以使用默认参数
21 h = lambda  x=0,y=0 : x ** 2 + y ** 2
22 print(h())
23 print(h(10))
24 print(h(10,20))
```

Day08

一、函数的特殊用法

1.变量可以指向函数

代码演示:

```
1  #abs------>absolute
2
3  #abs()是一个系统的内置函数【built-in function】
4  print(abs(-10))   #10
5  print(abs)   #<built-in function abs>
6
7  #结论一：abs(-10)是函数的调用，而abs是函数本身
8  x = abs(-20)
9  print(x)   #20
10
11 f = abs
12 print(f)   #<built-in function abs>
13
14 #结论二;函数本身也可以直接赋值给一个变量，也就是说：变量可以指向一个函数    num = 10
15 #如果一个变量指向了一个函数，则可以通过这个变量去调用这个函数
16 print(f(-30))
17
18 #结论三：f = abs，则表示f已经指向了abs所表示的函数，调用abs和调用f实现的效果是一样的
19
20 def test():
21  return  "fjskghs"
22
23 print(test())
24 fun = test
```

```
print(fun())
```

2.函数名是一个变量

代码演示：

```
#函数的特殊用法之函数名是一个变量

#结论一：函数名其实就是指向函数的变量

#abs（）：可以将abs看做一个变量，指向了一个可以计算绝对值的函数

#abs更改指向【变量的重新赋值】

num = 10
num = "hello"

#让abs指向一个整型
print(abs)
abs = 10
print(abs)

#print(abs(-100))
```

3.函数作为参数

代码演示：

```
#函数的特殊用法之函数作为参数

#变量可以指向函数，函数名是一个变量，而函数的形参本身就是一个变量，可以接收实参，那么一个函数就可以接收另一个函数作为参数
#高阶函数【一个函数就可以接收另一个函数作为参数】

#一个简单的高阶函数【需求：求两个数的绝对值的和】
#参数：x和y就是需要参与运算的数据，fun是一个函数
"""
def add(x,y):
 return abs(x) + abs(y)

"""
```

```
def add(x,y,fun):
 return fun(x) + fun(y)     #abs(x) + abs(y)

#将函数名abs作为参数使用
result = add(-5,6,abs)     #x = -5  y = 6   fun = abs  fun()
print(result)


#自定义函数
def show():
 print("abc")

def func(f):
 print("hello")
 f()

func(show)
```

二、偏函数【了解】

默认参数：可以降低函数调用的难度

偏函数：对函数参数做一些控制的函数

注意：偏函数不需要自定义，直接使用【系统函数】

代码演示：

```
import  functools

# 偏函数的使用


#int(x)    可以将字符串或者浮点型转换为整型
#默认：将其中的字符串按照十进制输出
print(int("123"))

#int（）还提供了一个额外的参数：base
#print(int("abc123"))
#base：指明前面数据的进制，int()执行完成之后，最终还是以十进制输出
print(int("123",base = 10))   #123
print(int("123",base = 8))    #83

```

```python

print(int("110",base = 2))
print(int("11010",base = 2))
print(int("110001",base = 2))

#转换大量的二进制，每次传入base = 2麻烦，可以将这个功能提取出来
def customInt(x,base=2):
 return int(x,base)
print(customInt("110"))  #6
print(customInt("11010"))  #26

#上面通过默认参数模仿了偏函数的使用，但是系统提供了功能：functools.partial可以创建一个偏函数【前提；需要导入import  functools】
#参数：需要创建偏函数的原函数名  需要设定的参数
int2 = functools.partial(int,base=2)
print(int2("110"))   #6
print(int2("11010"))  #26

print(int2("110",base=10))   #110

#总结：偏函数
#主要针对的是系统函数，如果系统默认的操作满足不了需求，则可以在这个系统函数的基础上生成一个新的函数【偏函数】，两个函数可以实现不同的需求
```

三、闭包【掌握】

如果在一个函数的内部定义另外一个函数，外部的函数叫做外函数，内部的函数叫做内函数

如果在一个外部函数中定义一个内部函数，并且外部函数的返回值是内部函数，就构成了一个闭包，则这个内部函数就被称为闭包【closure】

代码演示：

```python
"""
如果在一个函数的内部定义另外一个函数，外部的函数叫做外函数，内部的函数叫做内函数

如果在一个外部函数中定义一个内部函数，并且外部函数的返回值是内部函数，就构成了一个闭包，
则这个内部函数就被称为闭包【closure】
"""

#1.最简单的闭包
#外函数
def func(str):
```

```
#内函数【闭包】
 def innerFunc():
     print("hello")

 return innerFunc

#f中存储了外函数func的返回值，而func的返回值是innerFunc,j就相当于f = innerFunc
f = func("abc")   #f = innerFunc
#f（）就相当于innerFunc()
f()

#2.
#a和b被称为外函数中的临时变量【自由变量】
def outer(a):
 b = 10
 def inner():
     #在内函数中可以直接使用外函数中临时变量
     print(a + b)
 return  inner

f1 = outer(5)
f1()

#3.
def outer1(num1):
 def inner1(num2):
     #在内函数中可以直接使用外函数中临时变量
     print(num1,num2)
 return  inner1

f2 = outer1(10)
f2(20)

#应用场景：装饰器
```

四、变量的作用域

1.出现的原因

变量的作用域：变量可以被使用【被访问】的范围

程序中的变量并不是在任意的语句中都可以被访问，访问权限取决于这个变量被定义在哪个位置

2.作用范围划分

局部作用域：L【Local】

函数作用域：E【Enclosing】 将变量定义在闭包外的函数中

全局作用域：G【Global】

内建作用域：B【Built-in】

代码演示：

```
#1.不同作用域变量的定义
num4 = int(2.9)     #B;内建作用域
num3 =  3     #G；全局作用域
def outer():
 num1 = 1     #E:函数作用域

 def inner():
     num2 = 2    #L:局部作用域

     #注意：当所有的变量不同名的时候，在闭包中，可以任意访问四种不同作用域对应的变量
     print(num4,num3,num2,num1)

 return  inner

f = outer()
f()
```

3.变量的查找规则

查找的顺序：L------>E------>G------>B【极端情况：当所有的变量同名的情况下】【面试题】

代码演示：

```
#变量的查找规则

#注意：全局作用域和内置作用域，当重名的时候，谁出现在后面，则先匹配到谁
x = 0
x1 = int(3.3)

def outer1():
 j = 1

```

```
 def inner1():
     i = 2
     #【就近原则】
     print(x)

 return  inner1

f1 = outer1()
f1()
```

```
#函数
def show(a):
 num1 = 10
 print(num1,a)
show(20)
#print(num1,a)
#结论一：在函数中定义的变量【形参，在函数体中定义的变量】，作用域仅限于函数内部
# 【变量的生命周随着函数的出现而出现，函数执行完毕则变量随着被销毁】

#if语句
if True:
 msg = "hello"
 print(msg)
print(msg)


#for循环
for i in range(0,5):
 print(i)
print(i)

#结论二;Python中只有模块【module】、类【class】和函数【def,lambda】才会引入新的作用域
#其他的代码块：if语句，while语句，for语句，try-except语句都不会引入新的作用域
```

4.全局变量和局部变量【掌握】

全局变量：将变量定义在函数的外面

局部变量：将变量定义在函数的内部

注意：局部变量只能在其被声明的当前函数中使用，而全局变量可以在整个程序中使用

代码演示：

```
#全局变量
total = 0

def show():
 print(total)
show()

if True:
 total = 20
 print(total)

total = 10
print(total)

def add(arg1,arg2):
 #arg1,arg2，total1都属于局部变量
 total1 = arg1 + arg2
 print(total1)

add(10,20)
#print(total1)
```

作业讲解：

```
#1.计算1~某个数范围内奇数的和并返回
def fun1(num):
 sum = 0
 for i in range(1,num + 1):
     if i % 2 == 1:
         sum += 1

 return sum

print(fun1(100))

#2.判断某个数是否是质数，返回结果
def fun2(num):
 is_prime = True

```

```
    for x in range(2,num):
        if num % x == 0:
            is_prime = False
            break
    if is_prime and num != 1:
        return "质数"
    else:
        return "不是质数"

fun2(10)
```

5.global和nonlocal关键字的使用【掌握】

使用场景：当内部作用域【局部作用域，函数作用域】想要修改全局变量的作用域的时候

1.global

代码演示：

```
#global

#全局变量
num = 1
def fun1():

    #此时要使用全局变量中的num，需要给编译器做一个声明，声明此处使用num就是使用的全局变量中的num
    global num
    print(num)
    print(id(num))   #1355785312

    #局部变量
    num = 123
    print(num)
    print(id(num))   #1355789216

fun1()


#练习
a = 10
def test():
```

```
 global  a
 a = a + 1
 print(a)

test()
```

2.nonlocal

代码演示:

```
#前提：nonlocal关键字是定义在闭包中
x = 0

def outer():
 x = 1

 def inner():
     #使用nonlocal关键字进行声明，相当于将局部作用域范围扩大了
     nonlocal x
     x = 2
     print("inner:",x)
 #return  inner

 inner()

 print("outer:",x)   #2

outer()
print("global:",x)
"""
原本:
inner: 2
outer: 1
global: 0
"""

"""
修改:
inner: 2
outer: 2
global: 0
"""
```

五、列表生成式和生成器【掌握】

1.列表生成式/列表推导式

list comprehension

系统内置的用于创建list的方式

range(start,end,step)缺点:生成的列表一般情况下都是等差数列

代码演示:

```
#列表生成式

list1 = list(range(1,11))
print(list1)   #[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


#需求：[1, 4, 9, 16, 25]
list2 = []
for x in range(1,11):
 list2.append(x ** 2)
print(list2)  #[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

#1
#使用列表生成式完成上面的需求
#列表生成式的格式：[生成的元素 for-in循环]
list3 = [x ** 2 for x in range(1,11)]
print(list3)   #[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

#2.
#[4,16,36,64,100]
list4 = [x ** 2 for x in range(1,11) if x % 2 == 0]
print(list4)


#3.嵌套for循环，第一个循环就相当于外层循环，第二个循环就相当于内层循环
list5 = [m + n for m in "ABC" for n in "XYZ"]
print(list5)

#["AX","AY","AZ"....]

```

```
"""
for m in "ABC":
 for n in "XYZ":
     print(m + n)

"""

#4
#for k,v in dict.items)():
d = {"x":"1","y":"2","z":"3"}
for k,v in d.items():
 print(k,v)

list6 = [k + "=" + v for k,v in d.items()]
print(list6)   #['x=1', 'y=2', 'z=3']


#练习：使用列表生成式生成一个新的列表，将一个已知列表中的所有的字符变为小写
l1 = ["Hello","GOOD","ABC","kkH"]
#l2 = ["hello","good","abc","kkh"]

newList1 = []
for element in l1:
 str = element.lower()  #lower() 大写转化为小写
 newList1.append(str)
print(newList1)


newList2 = [s.lower() for s in l1]
print(newList2)
```

2.生成器

generator

next()

代码演示:

```
#生成器

#方式一：（），将列表生成式中的[]改成()
```

```python
#列表生成式的类型是list，生成器的类型是generator【当做一种新的数据类型】
r1 = (x ** 2 for x in range(1,6))
print(r1)   #(1,4,9,16,25)
print(type(r1))

"""
for i in r1:
 print(i)
"""

#生成器区别于列表生成式：可以使用next遍历，每调用一次则获取一个元素
#next()
print(next(r1))
print(next(r1))
print(next(r1))
print(next(r1))
print(next(r1))
#注意：当生成器中的元素全部获取完成之后，接着调用next函数的，则会出现StopIteration
#print(next(r1))   #StopIteration异常


#方式二：yield---->让步
#(x for x in range(1,6))----->1,2,3,4,5
def test(n):
 for i in range(1, n + 1):
     #执行到yield的时候，则函数会停止，将yiled后面的变量返回
     yield i ** 2
     #yield后面的代码的执行时机：当调用next函数的时候
     print(i)

t = test(5)
print(t)  #<generator object test at 0x0000019CC432A1A8>
print(next(t))
print(next(t))
print(next(t))
print(next(t))
print(next(t))
```

六、迭代器【掌握】

1.可迭代对象

可迭代对象【实体】：可以直接作用于for循环的实体【Iterable】

可以直接作用于for循环的数据类型：

a.list,tuple,dict,set,string

b.generator【() 和yield】

isinstance:判断一个实体是否是可迭代的对象

代码演示：

```
#一、可迭代对象
from collections import Iterable → from collections.abc import Iterable
或者降python版本到3.9以下


#1.导入
from  collections  import  Iterable

#2.使用isinstance(数据，Iterable)
print(isinstance([],Iterable))
print(isinstance((),Iterable))
print(isinstance({},Iterable))
print(isinstance((x for x in range(10)),Iterable))
print(isinstance("hello",Iterable))

print(isinstance(10,Iterable))   #False
print(isinstance(True,Iterable))  #False

print("****88")
```

2.迭代器

不但可以作用于for循环，还可以被next函数遍历【不断调用并返回一个元素，直到最后一个元素被遍历完成，则出现StopIteration】

目前为止，只有生成器才是迭代器【Iterator】

结论：迭代器肯定是可迭代对象，但是，可迭代对象不一定是迭代器

isinstance:判断一个实体是否是迭代器

代码演示：

```
#二、迭代器
from  collections  import  Iterator
```

```
3
4 print(isinstance([],Iterator))
5 print(isinstance((),Iterator))
6 print(isinstance({},Iterator))
7 print(isinstance("hello",Iterator))
8 print(isinstance((x for x in range(10)),Iterator))   #True
9
10 print("****88")
```

3.可迭代对象和迭代器之间的转换

可以将可迭代对象转换为迭代器：iter()

代码演示：

```
1 #三、虽然list、tuple、dict、set、string都不是迭代器
2 #iter():将list、tuple、dict、set、string的  Iterable转换为Iterator
3 print(isinstance(iter([]),Iterator))
4 print(isinstance(iter(()),Iterator))
5 print(isinstance(iter({}),Iterator))
6 print(isinstance(iter("hello"),Iterator))
```

总结：

a.凡是可以作用于for循环的对象都是Iterable类型

b.凡是可以作用于next函数的对象都是Iterator类型

c.list/tuple/dict/set/string都不是Iterator，可以通过iter()获得一个Iterator对象

Day09

一、装饰器【掌握】

1.案例

代码演示：

```
1 def test():
2     print("拼搏到无能为力，坚持到感动自己")
3
4 f = test  #变量可以指向指向函数，函数名也是一个变量，所以变量可以当做函数调用
5 f()
6
```

```
#思考问题：test增加功能，但是不能修改test函数内部----->装饰器
```

在代码运行期间，可以动态增加函数功能的方式，被称为装饰器【Decorator】
也就是说，在不修改原函数的基础上，给原函数增加功能
好处：在团队开发中，如果两个或者两个以上的程序员会用到相同的功能，但是功能又有细微的差别，采用装饰器：相互不影响，代码简化

2.使用

2.1简单装饰器

代码演示：

```
#1.简单的装饰器
def test():
     print("拼搏到无能为力，坚持到感动自己")

#a.书写闭包
#b.给外部函数设置参数,fun表示的是原函数
def outer(fun):
     def inner():
     # d.给原函数增加功能
         print("hello")

     #c.调用原函数
            fun()
      return inner

#e.使用闭包
f = outer(test)   #f = inner
f()   #inner()

#注意：增加的功能可以写在原函数调用的前面或者后面
#注意：outer函数就被称为装饰器


#练习：给下面的函数添加功能，打印九九乘法表

```

```
def show():
    for i in range(10):
        print(i)
def outer1(fun):
    def inner1():
        fun()
        for i in range(1,10):
            for j in range(1,i+1):
                print("%dx%d=%d"%(j,i,i*j),end="")
                print(" ")
    return inner1
f1 = outer1(show)
f1()
```

2.2有参数的装饰器

```
#2.原函数有参数的装饰器
def getAge(age):
    print(age)

getAge(10)
getAge(-5)

print("************")

#需求：在不修改原函数的基础上，进行数据的过滤：当用户输入age为负数的时候，则置为0
def wrapper(fun):
 #注意：当原函数有参数，装饰器的作用是为了操作原函数中的参数，给inner设置参数
 def inner(num):
     #增加新功能：过滤负数
     if num < 0:
         num = 0

     #调用原函数
     fun(num)  #age = num
 return  inner

f = wrapper(getAge)
f(10)   #num = 10
```

```
24 f(-5)
```

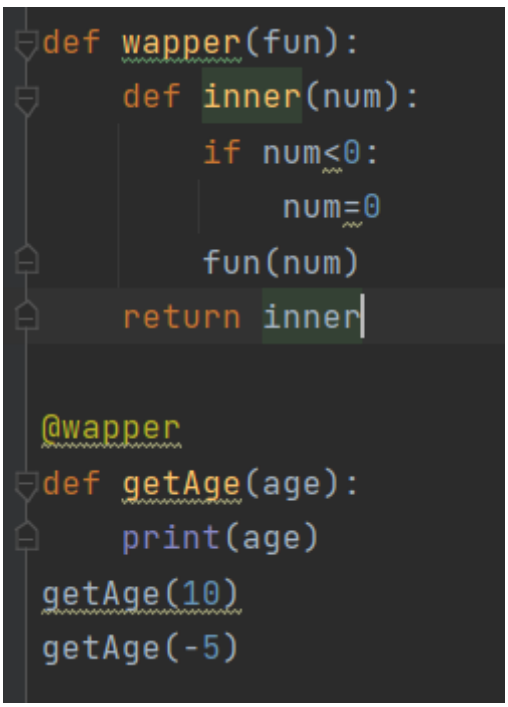
```
def getAge(age):
    print(age)
getAge(10)
getAge(-5)

print("************")
def wapper(fun):
    def inner(num):
        if num<0 :
            num =0
        fun(num)
    return inner

f = wapper(getAge)
f(10)
f(-5)
```

2.3系统的简写

代码演示:

```
1  #3.简化demo2中的操作：@装饰器的名称   应用到原函数中
2
3  #需求：在不修改原函数的基础上，进行数据的过滤：当用户输入age为负数的时候，则置为0
4  def wrapper(fun):
5   #注意：当原函数有参数，装饰器的作用是为了操作原函数中的参数，给inner设置参数
6   def inner(num):
7       #增加新功能：过滤负数
8       if num < 0:
9           num = 0
10
11      #调用原函数
12      fun(num)  #age = num
13  return  inner
14
15 #将wrapper装饰器应用在了getAge函数上，
```

```
@wrapper
def getAge(age):
 print(age)

getAge(10)
getAge(-5)

"""
@wrapper

等价于

f = wrapper(getAge)
f(10)   #num = 10

#注意;当使用@的时候，在同一个文件中，装饰器必须出现的原函数的前面

"""
```



2.4不定长参数的装饰器

代码演示:

```
#4.不定长参数的装饰器

```

```
#应用场景：当同一个装饰器作用于不同函数的时候，这些函数的参数的个数是不相同的
def wrapper(fun):
 def inner(*args):
     print("hello")

     fun(*args)   #a = args[0]   b = args[1]

 return  inner

@wrapper
def fun1(a,b):
 print(a + b)

@wrapper
def fun2(a,b,c,d):
 print(a,b,c,d)

fun1(10,20)   #args = (10,20)
fun2(1,2,3,4)
```

2.5多个装饰器作用于同一个函数

代码演示：

```
#5.将多个装饰器应用到同一个函数上
def wrapper1(fun):
 def inner1():
     print("1~~~~")
     fun()

 return inner1

def wrapper2(fun):
 def inner2():
     print("2~~~~")
     fun()

 return inner2

```

```
@wrapper1
@wrapper2
def show():
 print("hello")

show()

"""
1~~~~
2~~~~
hello
"""

#结论：多个装饰器作用于同一个函数的时候，从第一个装饰器开始，从上往下依次执行，但是，原函数只会被执行一次
```

二、函数递归【掌握】

1.概念

递归函数：一个会调用自身的函数【在一个函数的内部，自己调用自己】

递归调用

递归中包含了一种隐式的循环，他会重复指定某段代码【函数体】，但这种循环不需要条件控制

使用递归解决问题思路：

a.找到一个临界条件【临界值】

b.找到相邻两次循环之间的关系

c.一般情况下，会找到一个规律【公式】

2.使用

代码演示：

```
#案例一
"""
            1 2 3 4 5 6 7  8  9 10  11。。。。
斐波那契数列：1,1,2,3,5,8,13,21,34,55,89.....

解决问题：报一个数，输出数列中对应的数

规律：
a.第一个位置和第二个位置上数是固定的，都是1
```

```
b.第n个位置上的数：第 n - 1 的数  +   第 n - 2 的数

r1 = func1(1)  ------>1
r2 = func1(2)  ------>1
r3 = fun1(3) ------>func1(1) + func1(2)----->1 + 1 = 2
r4 = fun1(4)-------->fun1(3) + fun1(2) ----->func1(1) + func1(2) +  fun1(2) ---->1  + 1 + 1 = 3
r5 = fun1(5) ----->fun1(4) + fun1(3) ----->fun1(3) + fun1(2) + func1(1) + func1(2)--->func1(1) + func1(2) ++ fun1(2) + func1(1) + func1(2)--->5
.....
rn = fun1(n) ----->fun1(n- 1) + fun1(n - 2)
"""

def func1(num):
 #临界值
 if num == 1 or num == 2:
     return 1
 else:
     #print("~~~~",num)
     result = func1(num- 1) + func1(num - 2)    #result = func1(1) + func1(2)  --->1 + 1 =2
     return result

print(func1(10))

#练习;使用递归计算1~某个数之间的和
"""
add(1) = 1   :临界值
add(2) = add(1) + 2
add(3) = add(2) + 3 ---->add(1) + 2 + 3 = 1 + 2 + 3
add(4) = add(3) + 4---->add(2) + 3 + 4 ---->add(1) + 2 + 3 + 4---->1 + 2 + 3 + 4
....
add(n) = add(n - 1) + n
"""
def add(num):

 """
 n = 1
 sum = 0
 while n <= 100:
     sum += n
```

```
    n += 1

return sum

sum1 = 0
for i in range(1,num + 1):
    sum1 += i
return  sum1
"""
#使用递归实现
if num == 1:
    return 1
else:
    return add(num - 1) + num

print(add(100))
```

注意：以后在实际项目中尽量少用递归，如果隐式循环的次数太多，会导致内存泄漏【栈溢出】

优点：简化代码，逻辑清晰

三、栈和队列【了解】

用于存储数据的线性表

栈：在表的一端进行插入和删除

队列：在表的一端进行插入，在表的另一端进行数据的删除

1.栈

Stack

开口向上的容器：先进后出，后进先出

代码演示：

```
#list的底层维护了一个栈的线性表

myStack = []

#插入数据
#数据入栈【压栈】
myStack.append(1)
print(myStack)
myStack.append(2)
print(myStack)
```

```
myStack.append(3)
print(myStack)
myStack.append(4)
print(myStack)   #[1,2,3,4]

#出栈
myStack.pop()
print(myStack)
myStack.pop()
print(myStack)
myStack.pop()
print(myStack)
myStack.pop()
print(myStack)
```

2.队列

queue

水平放置的水管：先进先出，后进后出

代码演示：

```
import  collections   #数据结构的集合

queue  = collections.deque([1,2,3,4])
print(queue)

#入队【存储数据】
queue.append(5)
print(queue)
queue.append(6)
print(queue)

#出队【获取数据】
queue.popleft()
print(queue)
queue.popleft()
print(queue)
queue.popleft()
print(queue)
```

Day10

一、目录遍历

os 用于获取系统的功能，主要用于操作文件或者文件夹

代码演示：

```
import  os

path = r"C:\Users\Administrator\Desktop\SZ-Python"

#获取指定目录下所有的文件以及文件夹，返回值为一个列表
filesList = os.listdir(path)
print(filesList)

#C:\Users\Administrator\Desktop\SZ-Python
#通过初始路径拼接子文件或者子文件夹形成新的路径
filePath = os.path.join(path,"作业")
print(filePath)

#判断指定的路径是否是文件夹【目录】
result = os.path.isdir(filePath)
print(result)
```

```python
import os

path = r"E:\python"
fileList=os.listdir(path)
print(fileList)
```

```
venv\Scripts\python.exe E:/python/day_01/test.py
', 'demo.xls', 'djangoProject', 'flaskProject', 'flaskProject1', 'mydjango']
```

1.使用递归遍历目录

代码演示:

```python
#递归
import os

def getAll(path):
 #1.获取当前目录下所有的文件以及文件夹
 fileList = os.listdir(path)
 print(fileList)

 #2.遍历列表
 for i in fileList:
     #3.拼接路径
     filePath = os.path.join(path,i)

     #4.判断filePath是否是文件夹
```

```
    if os.path.isdir(filePath):
        #文件夹:递归
        getAll(filePath)
    else:
        #文件
        print("文件: ",i)

getAll(r"C:\Users\Administrator\Desktop\SZ-Python")
```

二、包

```
包：初期理解为文件夹

作用：一种管理Python模块命名空间的形式，采用"点语法"    os.path

包和文件夹之间的区别:Python的包中有一个特殊的文件__init__.py文件,前期里面不写任何内容，但是，就是为了告诉编译器，当前这个目录不是普通目录，是一个包

创建方式：选中工程，创建Python package
```

代码演示:

```
"""
1.在Python中，一个py文件其实就是一个模块
2.如果要跨模块调用函数，需要在运行的模块中导入需要使用的模块，调用函数的时候需要指明函数的路径
"""

#第一步：导入模块
#导入格式：包名.模块名
import aaa.textDemo01
import ccc.module

#os.path.isdir()
aaa.textDemo01.test()
ccc.module.test()

#包存在的意义：在团队开发的过程中，为了解决文件命名冲突的问题，只要保证最上层的包命名不相同，就不会与别人的发生冲突
```

三、模块

1.概述

为了解决维护问题，一般情况下，在一个完整的项目中，会将特定的功能分组，分别放到不同的文件中，在使用的过程中，可以单独维护，各个不同的文件之间互不影响，每个.py文件就被称为一个模块，通过结合包的使用来组织文件

封装思路: 函数 => 类 => 模块 => 包 => 项目

优点:

a.提高了代码的可维护性

b.提高了代码的复用性【当一个模块被完成之后，可以在多个文件中使用】

c.引用其他的模块【第三方模块】

d.避免函数名和变量的命名冲突

2 os模块

提供有关于操作系统的函数，处理文件或者文件夹

代码演示:

```
import os
# operation system 操作系统

#1.获取当前操作系统的名称
# nt----->Windows
# posix------>Linux, Mac os
print(os.name)

#2.获取当前系统的环境变量
#以字典的形式返回
print(os.environ)
#通过key获取对应的value
print(os.environ.get("APPDATA"))

#3,获取指定目录下所有的文件或者文件夹的列表
l = os.listdir(r"C:\Users\Administrator\Desktop\SZ-Python")
print(l)

#4.在指定的路径下创建文件夹
#os.mkdir(r"C:\Users\Administrator\Desktop\aaa")

#5.删除文件夹
#os.rmdir(r"C:\Users\Administrator\Desktop\aaa")
```

```python
#删除文件
#os.remove("")

#6.获取文件属性
#print(os.stat(r"C:\Users\Administrator\Desktop\aaa"))

#7.给文件或者文件夹重命名
#注意：当前的文件在关闭状态
#rename(old,new)
#os.rename(r"C:\Users\Administrator\Desktop\aaa",r"C:\Users\Administrator\Desktop\abc")

#os.path模块下
#1.路径的拼接
path = os.path.join(r"C:\Users\Administrator\Desktop\SZ-Python","Day1Code")
print(path)

#2.绝对路径和相对路径【掌握】
"""
绝对路径：带有盘符的路径，缺点：只能在指定的计算机上使用
相对路径：不带盘符的路径，一般情况下是以当前的工程为参照物
 例如：
     aaa/textDemo01.py
     ccc/module.py
"""
#os.rename("bbb/check.py","bbb/show.py")

#3.拆分路径
#注意：返回的结果为元组，默认情况下只会拆分最后的文件或者文件夹
tuple1 = os.path.split(r"C:\Users\Administrator\Desktop\SZ-Python\Day1Code")
print(tuple)

#4.拆分路径，获取指定路径对应的文件的扩展名
print(os.path.splitext(r"C:\Users\Administrator\Desktop\SZ-
Python\Day2Code\assignDemo.py"))

#5.判断指定路径是否是文件夹
print(os.path.isdir("aaa/textDemo01.py"))

#6.判断指定路径是否是文件
print(os.path.isfile("aaa/textDemo01.py"))
```

```
63
64 #7.判断一个指定路径是否存在
65 print(os.path.exists("aaa/textDemo01.py"))
66
67 #8.获取文件的大小【字节】
68 print(os.path.getsize("aaa/textDemo01.py"))
69
70 #9.
71 #获取指定文件夹的父路径
72 print(os.path.dirname(r"C:\Users\Administrator\Desktop\SZ-Python\Day1Code"))
73 #获取当前文件夹的名称
74 print(os.path.basename(r"C:\Users\Administrator\Desktop\SZ-Python\Day1Code"))
```

练习:

```
1  import  os
2  #练习：获取指定目录下所有的py文件或者txt文件
3  """
4  思路:
5  1.判断指定的目录是否存在
6  2.获取指定目录下所有的文件以及文件夹: listdir()
7  3.拼接路径:
8  4.判断拼接之后的路径是否是文件
9  5.判断文件名称的后缀
10 """
11 def getFile(path):
12  #1.判断括号里的文件是否存在的意思，括号内的可以是文件路径。
13  if os.path.exists(path):
14      #2返回指定的文件夹包含的文件或文件夹的名字的列表。
15      fileList = os.listdir(path)
16
17      #3.
18      for fileName in fileList:
19          filePath = os.path.join(path,fileName)
20
21          #4
22          if os.path.isfile(filePath):
23              #5
24              if fileName.endswith("py") or fileName.endswith("txt"):
25                  print(fileName)
26          else:
```

```
            print(fileName,"不是文件")

 else:
     print("指定的路径不存在")

getFile(r"C:\Users\Administrator\Desktop\SZ-Python\Day5Code")
```

3.自定义模块【掌握】

3.1自定义import模块

代码演示:

```
#1.格式：import  包1.包2.模块的名称
#注意1：通过点语法区分包的层级关系
#引入模块

#注意2：如果要同时导入多个模块，有两种方式
#方式一
"""
import os
import datetime
import math
"""
#方式二
import os,math,datetime

#注意3：当导入自定义模块的时候，需要注意包的存在
#注意5：当通过import将模块导入的时候，将模块对应的文件整个加载了一遍
import ccc.module
import moduleTextDemo01

print("***************")

#注意4：当模块有包的层级关系时，需要调用其中函数的时候，需要指明函数的路径
ccc.module.test()     #os.path.isdir()

moduleTextDemo01.fun1()
moduleTextDemo01.fun2()
moduleTextDemo01.fun3()

```

```
print(moduleTextDemo01.num)
```

3.2自定义from-import模块

代码演示：

```
#form 模块名  import 函数名1/类名，函数名2.。。。
#import  moduleTextDemo01
from moduleTextDemo01 import  fun1,fun2,fun3

#注意：采用了form。。。import的方式导入指定的函数之后，可以直接调用函数
fun1()
fun2()
fun3()

#好处：进行局部的导入，避免内存空间的浪费


#注意：当前文件中如果存在和模块中同名的函数的时候，当前文件中的函数仍然会将模块中的函数给覆盖掉
def fun1():
 print("hello")

fun1()
```

3.3自定义from-import*模块

代码演示：

```
#from 。。。。 import  *      *代表全部
"""
下面三种导入方式完全等价：将moduleTextDemo01模块中的所有的内容全部导入
from moduleTextDemo01 import  *
import moduleTextDemo01
from  moduleTextDemo01 import  fun1,fun2,fun3
"""
from moduleTextDemo01 import  *

fun1()
```

总结：在python中，每个py文件其实都是一个模块，如果跨模块调用函数，则采用导入的方式

将不同的功能进行划分，调用函数的时候相对比较方便的

4.name属性和dir函数

4.1name属性

```
#__name__的作用：如果不想让模块中的某些代码执行，可以通过属性仅仅调用程序中的一部分功能
#【写在if判断中的代码只有当前模块被执行的时候才会被执行，检测到是其他的文件在使用当前的模块，则if语句中的代码不会被执行】

def fun1():
 print("aaa")

def fun2():
 print("bbb")

def fun3():
 print("ccc")


#作用：写在下面判断中的代码，只有当前模块运行的时候才会被执行【起到屏蔽的作用】
if __name__ == "__main__":
 fun1()
 fun2()
 fun3()
```

4.2dir函数

代码演示:

```
newdir
import math,moduleTextDemo01

#获取指定模块里面的所有的内容
newdir
print(dir(math))
print(dir(moduleTextDemo01))
```

Day11

一. 系统模块

UTC：国际标准时间, 格林尼治天文时间，UTC+8
时间戳：指定时间距离1970.1.1 00:00:00的秒数
time：时间
datetime:日期
calendar：万年历
os：系统，文件和文件夹

2.1time时间模块

代码演示：

```
import time

#1.获取当前时间对应的时间戳，使用浮点型表示,单位:秒【掌握】
t1 = time.time()
print(t1)

#2.休眠,参数的单位为秒,可以为浮点数【掌握】
print("hello1")
time.sleep(2)
print("hello2")
```

2.2datetime日期模块【掌握】

是对time模块的封装，比time模块更加全面

```
'''
dt_now = datetime.datetime.now()
获取当前的日期对象，包含时间的
dt_ziding = datetime.datetime()
根据指定的日期、时间生成一个日期对象

dt.strftime()  将日期对象转化为指定的格式
dt.date()   获取日期对象中的日期
dt.time()   获取日期对象中的时间
dt.timestamp()  获取日期对象的时间戳
dt.hour\minute\second  获取小时、分钟、秒

datetime.datetime.fromtimestamp()
根据一个时间戳，转化为指定的日期对象
datetime.timedelta()
```

```
生成一个差值对象，可以和日期对象直接进行相加减
参数有，days,hours,minutes,seconds
'''

'''
# %y 两位数的年份表示（00-99）
# %Y 四位数的年份表示（000-9999）
# %m 月份（01-12）
# %d 月内中的一天（0-31）
# %H 24小时制小时数（0-23）
# %I 12小时制小时数（01-12）
# %M 分钟数（00-59）
# %S 秒（00-59）
# %a 本地简化星期名称
# %A 本地完整星期名称
# %b 本地简化的月份名称
# %B 本地完整的月份名称
# %c 本地相应的日期表示和时间表示
# %j 年内的一天（001-366）
# %p 本地A.M.或P.M.的等价符
# %U 一年中的星期数（00-53）星期天为星期的开始
# %w 星期（0-6），星期天为星期的开始
# %W 一年中的星期数（00-53）星期一为星期的开始
# %x 本地相应的日期表示
# %X 本地相应的时间表示
# %% %号本身
'''
```

代码演示：

```
import  datetime

#1.获取当前时间
d1 = datetime.datetime.now()
print(d1)  #2018-05-29 11:20:51.432757

#2.获取指定的时间，通过元组形式
d2 = datetime.datetime(2015,10,1,10,23,23,1234)
print(d2)

#3.将时间格式化
```

```
d3 = d1.strftime("%Y.%m.%d")
print(d3)

#4.将时间字符串转换为datetime实体
d4 = datetime.datetime.strptime(d3,"%Y.%m.%d")
print(d4)

#5.直接进行加减运算
date1 = datetime.datetime(2015,10,1,10,23,23,0)
print(date1)

date2 = datetime.datetime(2015,10,4,10,23,23,0)
date3 = date2 - date1
print(date3)  #3 days, 0:00:00

print(date3.days)
print(date3.seconds)
```

二、面向对象思想

1.面向对象思想设计

基于哲学观点：万物皆对象, 一切皆对象

举例说明：

案例一：我想吃大盘鸡

面向过程 面向对象

1.自己去买菜 1.委托一个人帮忙买菜

2.自己择菜 2.委托一个人帮忙择菜

3.自己做菜 3.委托一个人厨师做菜

4.自己吃 4.自己吃

案例二：小明是一个电脑小白，想要配置一台电脑

面向过程 面向对象

1.小明补充电脑知识 1.委托一个懂电脑的人买零件

2.小明去买零件 2.委托一个人组装

3.小明把零件运回来 3.小明打开玩游戏

4.小明组装

5.小明打开玩游戏

2.面向过程和面向对象的区别

2.1面向过程

在生活案例中：

一种看待问题的思维方式，在解决问题的时候，侧重于问题是怎样一步一步解决的，然后亲力亲为的去解决

在程序中：

代码从上往下依次执行

各个模块之间的关系尽可能的独立的，当import的时候，加载的顺序也是从上往下依次加载

每个模块中的语句结构：顺序，分支，循环

2.2面向对象

在生活案例中：

一种看待问题的思维方式，侧重于找到一个具有特殊功能的个体，然后委托这个个体帮忙完成某件事情，这个个体就被称为对象

好处：可以将复杂的问题简单化，将程序员从执行者变成了指挥者

在程序中：

根据不同的需求执行代码【代码执行顺序不一定】

程序的流程完全由需求决定【对象】

思想：如果对象存在，则直接使用；如果对象不存在，则创建对象

注意：面向对象只是一种思想，并不是一门编程语言

Python是一门面向对象的编程语言，类和对象是 面向对象的核心.

```
示例： 小狗吃食（闻一闻smell、舔一舔lick、咬一咬bite）
            分别采用面向过程和面向对象来分析

面向过程 ： 先闻一闻，然后再舔一舔，最后再咬一咬 (注重过程)
面向对象 ： 小狗是一个对象，它可以闻一闻食物，可以舔一舔食物，可以咬一咬食物.      (不注重过程，注重对象)
```

三、类和对象【掌握】

1.类和对象的概念

类：多个具有特殊功能的个体的集合

对象：在一个类中，一个具有特殊功能的个体，能够帮忙解决某件特定的事情，也被称为实例【instance】

两者之间的关系：类用于描述某一类对象的共同特征，而对象是类的具体的存在

思考问题：先有类还是先有对象?

【在程序中使用的时候，一般是先定义类，然后创建对象】

举例：

类 对象

人 王麻子, 李四, 赵四。。

快递 韵达，中通，圆通。。

SupreHero 蝙蝠侠，蜘蛛侠，美国队长。。

帮忙理解:类其实也是一种数据类型，只不过一般情况下是自定义的，所以可以将类认为是自定义的数据类型，用法和整型，str，list等基本是相同的【定义变量，传参】

2.类的定义

语法：

class 类名( ):

类体

说明：

a.Python中使用class关键字定义类

b.类名只要是一个合法的标识符即可，但是要求：遵循大驼峰命名法则【首单词的首字母大写，不同单词之间首字母大写】

c.通过缩进区分类体

d.类体一般包含两部分内容：对类的特征的描述、对类的行为的描述

代码演示：

```
#类的定义
#类的声明
class MyClass():
 #类的实现
 #类体
 #print("hello")   #一般不会这么书写
 pass

#注意：在同一个py文件中可以同时定义多个类，但是，为了提高代码的可读性，结合模块的使用，最好是一个文件一个类
class MyClass1():
 pass
```

3.类的设计【类体的实现】

三要素：

事物名称【类名】：举例：人

事物的特征【变量】：名词，举例：姓名，年龄。。。。

事物的行为【函数/方法】：动词，举例：吃，跑。。。。

三、类中的方法和变量【掌握】

1.类中的方法和变量的定义

类中的方法和变量是为了描述事物的行为和特征

类中定义的方法被称为成员方法

类中定义的变量被称为成员变量，也被称为属性 [os.name]

成员变量：类具有的特征

成员方法：类具有的行为

类存在的意义：拥有相同特征和行为的对象可以抽取出来一个类，类的存在是为了创建一个具体的对象

代码演示：

```
#定义类
#1.事物的名称：类名
class Person():
 #2.事物的特征：成员变量、属性
 name = ""
 age = 0
 height = 0.0

 #3.事物的行为：成员方法【函数】
 #注意：类中的成员方法区别于普通方法：参数部分一定包含self，而且最好self出现在参数列表的第一个
 #调用函数的时候，self不需要被传参
 #初次之外，成员方法的用法和普通方法的使用完全相同，也可以设置默认参数或者关键字参数，不定长参数

 #注意：self：自己，代表类的实例【对象】
 #此处的self可以是任意的标识符，只不过为了结合其他编程的使用，习惯上使用self
 def eat(self,food):
     print("eating",food)
 def run(self):
     print("running")
```

2.类中方法和属性的使用

2.1创建对象【实例化对象】

已知类，通过类创建对象

对象的创建过程被对象的实例化过程

语法：变量名 = 值

对象名 = 类名()

代码演示：

```
#定义类
#1.事物的名称：类名
class Person():
 #2.事物的特征：成员变量、属性
 name = ""
 age = 0
 height = 0.0

 #3.事物的行为：成员方法【函数】
 #注意：类中的成员方法区别于普通方法：参数部分一定包含self，而且最好self出现在参数列表的第一个
 #调用函数的时候，self不需要被传参
 #初次之外，成员方法的用法和普通方法的使用完全相同，也可以设置默认参数或者关键字参数，不定长参数

 #注意：self：自己，代表类的实例【对象】
 #此处的self可以是任意的标识符，只不过为了结合其他编程的使用，习惯上使用self
 def eat(self,food):
     print("eating",food)
 def run(self):
     print("running")
     print("self的地址：", id(self))


#对象的创建
p1 = Person()
print(p1)

p2 = Person()
print(p2)

#p1和p2被称为对象，变量名，引用，指向了真正的对象
#p1和p2在栈空间中开辟了空间，真正的对象的被存储在堆空间中

#通过对象调用类中的成员方法和访问类中的成员变量
#1.访问属性
#语法：对象.属性名
#赋值：对象.属性 = 值
```

```python
per = Person()
print(per.name)
per.name = "小姐姐"
print(per.name)
per.age = 18
print(per.age)
per.height = 1.70
print(per.height)

#2.调用方法
#语法：对象.函数名(参数列表)
#注意：self不需要被传参，传参的时候注意区分参数的类型【默认参数，不定长参数，关键字参数】
per.run()
print("per的地址：",id(per))
"""
self的地址：  2687721120880
per的地址：  2687721120880
"""
per.eat("apple")

person = Person()
person.name = "张三"
person.age = 20
print(person.name,person.age)
person.run()
person.eat("")

#结论：类中的成员变量和成员方法随着对象的出现而出现
```

总结：

访问变量采用：对象名.属性名

访问方法采用：对象名.方法名(参数列表)

3.内存中的对象

per = Person()

说明：

a.程序中定义的Person类型的变量per实际上是一个变量名，它被存放在栈内存中，他指向实际的Person对象，而真正的Person对象则存放于堆内存中

b.类中的成员变量随着对象的出现而出现，随着对象的消失而消失

c.每个对象的成员变量会在堆空间中开辟一份自己的空间，相互之间互不影响

4.动态绑定属性和限制绑定

```
#1.类的定义
class MyClass():
 #2.成员变量
 """
 num1 = 0
 num2 = 10
 """
 #限制属性
 #注意：被限制的属性的名称通过字符串的方式出现在元组的元素中
 __slots__ = ("num1","num2")

 #3.成员方法
 def fun1(self):
     print("fun1")
 def fun2(self,num):
     print(num)

#4.创建对象
my = MyClass()
#5.访问类中的成员变量
my.num1 = 11
my.num2 = 22
print(my.num1,my.num2)

#6.调用类中的成员方法
my.fun1()
my.fun2(30)

#成员变量随着对象的出现而出现的
#属性的动态绑定【Python是一门动态语言】
my.n = 100
print(my.n)

my1 = MyClass()
#print(my1.n)
```

5.综合案例一

代码演示：

practiceDemo01.py文件【测试模块】

```
"""
需求：使用面向对象的思想描述下面这个情景
开学了，王老师让小明，小花，小丽分别做自我介绍
需要介绍姓名，年龄，爱好，来一段才艺展示
"""
"""
分析：
老师类
 特性：姓名
 行为：让学生做自我介绍

学生类
 特征：姓名，年龄，爱好
 行为：一段才艺展示
"""
#导入
"""
import  practice01.teacher
import  practice01.student
"""
from practice01.teacher import  Teacher
from practice01.student import Student

#1.创建一个老师的对象
wang = Teacher()
wang.name = "王老师"

#2.创建一个学生的对象
xiaohua = Student()
xiaohua.name = "小花"
xiaohua.age = 18
xiaohua.hobby = "唱歌"

#3.让老师执行自己的行为
wang.letStudentIntroduce(wang.name,xiaohua)   #stu = xiaohua
```

```
36
37 xiaoli = Student()
38 xiaoli.name = "小丽"
39 xiaoli.age = 20
40 xiaoli.hobby = "跳舞"
41 wang.letStudentIntroduce(wang.name,xiaoli)
42
43 xiaoming = Student()
44 xiaoming.name = "小明"
45 xiaoming.age = 25
46 xiaoming.hobby = "吹牛逼"
47 wang.letStudentIntroduce(wang.name,xiaoming)
```

teacher.py文件【实体类】

```
1  #老师类
2  class Teacher():
3   #特征：成员变量
4   name = ""
5
6   #行为：成员方法
7   def letStudentIntroduce(self,name,stu):
8       #老师发出指令
9       print(name + "让" + stu.name + "做自我介绍")
10
11      #执行指令
12      stu.introduce(stu.name,stu.age,stu.hobby)
13
14      #不同的学生展示不同的才艺
15      if stu.name == "小花":
16          stu.singSong()
17      elif stu.name == "小丽":
18          stu.dance()
19      else:
20          stu.lie()
```

student.py文件【实体类】

```
1  #学生类
2  class Student():
3   #特征：成员变量
```

```
    name = ""
    age = 0
    hobby = ""

    #行为：成员方法
    def introduce(self,name,age,hobby):
        print("大家好，我是%s，今年%d,爱好%s"%(name,age,hobby))

    #唱歌
    def singSong(self):
        print("娘子~啊哈")

    #跳舞
    def dance(self):
        print("广场舞")

    #吹牛逼
    def lie(self):
        print("我家可穷了，就养了几百头牛")
```

四、构造函数和析构函数

1.构造函数【掌握】

```
采用上面的方式创建对象【直接给成员变量赋值】，很多的类一般倾向于创建成有初始状态的
__init__:构造函数【作用：创建对象，给对象的成员变量赋初始值】
构造函数：构造器
调用的时机：当一个对象被创建的时候，第一个被自动调用的函数
per = Person()

语法：
        def __init__(self,args1,args2....)
                函数体
说明：
        a.之前的写法中并没有显式的定义__init__函数，说明系统默认提供了一个无参的构造函数
        b.args1,args2...一般设置的形参列表和成员变量有关
```

代码演示：

```
#1.构造函数被调用的时机
```

```python
class Check():
 num1 = 0
 str1 = ""

 #构造函数
 def __init__(self):
     print("jfahj")

 def show(self):
     print("show")
#注意：当创建对象的时候，默认调用了系统提供的无参的构造函数
c = Check()
c.show()

#2.给构造函数添加参数
class Check1():
 name = ""
 age = 0
 """
 def __init__(self,n,a):
     print("fajkgak")
 """
 #注意2：当使用构造函数的时候，可以使用无参的，也可以使用有参的，在Python中的解决办法：设置不定长参数
 #注意3：Python中，一个类中只能有一个构造函数
 def __init__(self, *n):
     print("fajkgak")


#注意1：当手动头添加了有参的构造函数之后，系统将不再提供无参的构造函数
c1 = Check1()
c11 = Check1("fsiugh")

#3.有参构造函数的使用
class Check2():
 name = ""
 age = 0

 #构造函数的形参列表：和成员变量有关
 def __init__(self,n,a):
```

```
    print(n,a)
    name = n
    age = a

#注意1：当手动头添加了有参的构造函数之后，系统将不再提供无参的构造函数
c2 = Check2("zhangsan",10)
print(c2.name,c2.age)   #0


#4.self的使用
class Check3():
 name = ""
 age = 0

 #构造函数的形参列表：和成员变量有关
 def __init__(self,n,a):
    print(n,a)
    #self的使用：通过self来区分成员变量和局部变量,所以self.name代表name是一个全局变量【成员变量】
    self.name = n
    self.age = a

c3 = Check3("zhangsan",10)
print(c3.name,c3.age)  #10

#5.使用self之后，可以省略成员变量的定义【掌握】
#self只是一个标识符，可以替换成任意的标识符
class Check4():

 #构造函数的形参列表：和成员变量有关
 def __init__(self,name,age):
    print(name,age)
    #self的使用：通过self来区分成员变量和局部变量,所以self.name代表name是一个全局变量【成员变量】
    self.name = name
    self.age = age

 def show(self):
     print("showing")

```

```
c4 = Check4("lisi",20)
print(c4.name,c4.age)
c4.show()
```

2.析构函数

```
与构造函数正好相反，当对象被销毁的时候自动调用的函数，被称为析构函数
__del__:

删除变量：    del  变量名，此时可以触发析构函数的调用

使用情景：清理工作，比如关闭数据库，关闭文件等
```

代码演示：

```
import  time

class Pig():
 def __init__(self,name,age):
     self.name  = name
     self.age = age
     print("构造函数被执行")


 def show(self):
     print("show")

 #析构函数
 def __del__(self):
     print("~析构函数被调用")


#析构函数被调用的时机：1：当程序运行完成的时候     2：使用del删除变量
p = Pig("abc",10)

del p

#注意：对象释放以后就不能再访问了【相当于根本未创建过这个对象】
#print(p.age)

```

```
time.sleep(5)

#在函数里定义的对象，会在函数结束时自动释放，这样可以用来减少内存空间的浪费
#其实就是作用域的问题
def func():
 per2 = Person("a.a", 1, 1, 1)

func()
```

3.综合案例二

practiceDemo02.py文件【测试模块】

```
"""
需求：富二代思聪开着豪车，很自豪的向他的新女友炫耀

富二代类
特征：姓名
行为：开车，炫耀

汽车类
特征：品牌，颜色
行为：奔驰
"""
#测试模块
from  practice02.car import Car
from practice02.richMan import RichMan

#1.创建一个富二代的对象
wang = RichMan("思聪")

#2.创建一个汽车的对象
c = Car("玛莎拉蒂","闷骚红")
c.run()

#3.让富二代执行自己的行为
wang.driveCar(c)
wang.showCar(c)
```

richMan.py文件【实体类】

```python
class RichMan():
 #构造函数
 def __init__(self,name):
     self.name = name

 #成员函数
 def driveCar(self,car):
     print("富二代%s开着他的豪车%s"%(self.name,car.brand))
 def showCar(self,car):
     print(car.brand,car.color)
```

car.py文件【实体类】

```python
class Car():
 #构造函数
 def __init__(self,brand,color):
     self.brand = brand
     self.color = color

 #成员函数
 def run(self):
     print("%s在马路上奔驰"%(self.brand))
```

Day12

一、封装【private】

1.概念

广义的封装：函数和类的定义本身，就是封装的体现

狭义的封装：一个类的某些属性，在使用的过程 中，不希望被外界直接访问，而是把这个属性给作为私有的【只有当前类持有】，然后暴露给外界一个访问的方法即可【间接访问属性】

封装的本质：就是属性私有化的过程

封装的好处：提高了数据的安全性，提高了数据的复用性

2.属性私有化

如果想让成员变量不被外界直接访问，则可以在属性名称的前面添加两个下划线__,成员变量则被称为私有成员变量

私有属性的特点：只能在类的内部直接被访问，在外界不能直接访问

代码演示：

```
#1.属性不私有化的时候
class Person():
 def __init__(self,name,age):
     self.name = name
     self.age = age

 def myPrint(self):
     print(self.name,self.age)

#通过构造函数给属性赋值
per = Person("张三",10)
per.myPrint()   #张三 10
#通过对象直接访问属性，并且给属性赋值
per.name = "李四"
per.age = 22
per.myPrint()   #李四 22

#2.属性私有化
#写法：在属性的前面添加两个下划线
#用法：只能在类的内部被访问，外界不能直接访问
class Person1():
 def __init__(self,name,age):
     self.name = name
     self.__age = age

 def myPrint(self):
     print(self.name,self.__age)

p1 = Person1("abc",10)
p1.myPrint()   #abc 10
p1.name = "hello"
#其实动态绑定属性，age和__age其实是两个不同的变量
p1.age = 222
```

```
p1.myPrint()
print(p1.age)

#AttributeError: 'Person1' object has no attribute '__age',私有化了，在外界不能直接访问
#print(p1.__age)
```

3.get函数和set函数

get函数和set函数并不是系统的函数，而是自定义的，为了和封装的概念相吻合，起名为getXxx和setXxx

get函数：获取值

set函数：赋值【传值】

代码演示：

```
#3.get函数和set函数
class Person2():
 def __init__(self,name,age):
     self.name = name
     self.__age = age
     #特殊情况一
     self.__weight__ = 20.0
     #特殊情况二
     self._height = 155.0

 def myPrint(self):
     print(self.name,self.__age)

 # 书写私有属性age的get函数和set函数【通过自定义的函数进行私有属性的赋值和获取值，暴露给外界】
 """
 get函数和set函数并不是系统的函数，而是自定义的，为了和封装的概念相吻合，起名为getXxx和setXxx
 get函数：获取值
 set函数：赋值【传值】
 """
 #set函数:给成员变量赋值
 #命名方式：setXxx
 #特点：需要设置参数，参数和私有成员变量有关
 def setAge(self,age):
     #数据的过滤
     if age < 0:
         age = 0
```

```
        self.__age = age
    #get函数：获取成员变量的值
    #命名方式：getXxx
    #特点：需要设置返回值，将成员变量的值返回
    def getAge(self):
        return self.__age

    #注意：有几个私有属性，则书写几对get函数和set函数

p2 = Person2("abc",10)
p2.myPrint()   #abc 10
#print(p2.__age)
#间接的访问了私有的成员变量
print(p2.getAge())
p2.setAge(22)
print(p2.getAge())

p2.setAge(-20)
print(p2.getAge())

#总结：通过将属性私有化之后，然后提供get函数和set函数，外部代码就不能随意更改成员变量的值，这样在一定程度上保证了数据的安全性

#4.工作原理【了解】
#当编译器加载了程序之后，不能直接访问p2.__age,Python解释器把__age解释成_Person2__age
#p2.__age = 100
p2._Person2__age = 100
print(p2.getAge())

#5.特殊情况：尽量不要直接访问
#a.在一个变量的前后各加两个下划线，在Python中被认为特殊成员变量，将不再属于私有变量
#print(p2.__weight__)
#b.特殊变量
#print(p2._height)

#面试题：下面变量的含义
"""
xxx:普通的变量
_xxx:受保护的变量，不建议使用这种形式
__xxx:表示私有的，外界无法直接访问，只能通过暴露给外界的函数访问
```

```
__xxxx__：一般是系统的内置变量，比如：__name__,__solts__,自定义标识符的时候尽量不要使用这种形式
"""
```

4.@property装饰器

装饰器的作用：可以给函数动态添加功能，对于类的成员方法，装饰器一样起作用

Python内置的@property装饰器的作用:将一个函数变成属性使用

@property装饰器：简化get函数和set函数

使用：@property装饰器作用相当于get函数，同时，会生成一个新的装饰器@属性名.settter,相当于set函数的作用

作用：使用在类中的成员函数中，可以简化代码，同时可以保证对参数做校验

代码演示：

```python
class Person1():
 def __init__(self,name,age):
     self.__name = name
     self.__age = age

 def myPrint(self):
     print(self.__name,self.__age)

 """
def setAge(self,age):
     #数据的过滤
     if age < 0:
         age = 0
     self.__age = age

 def getAge(self):
     return self.__age
 """

 #注意：函数的命名方式：变量的名称
 #作用：相当于get函数，设置返回值，将成员变量的值返回
 @property
 def age(self):
     return  self.__age

 #注意：函数的命名方式：需要和@property中函数的命名保持一致
```

```
    #作用：相当于set函数,设置参数，给成员变量赋值
    @age.setter
    def age(self,age):
        if age < 0:
            age = 0
        self.__age = age

    @property
    def name(self):
        return  self.__name

    @name.setter
    def name(self,name):
        self.__name = name


p1 = Person1("abc",10)
p1.myPrint()   #abc 10
#p1.setAge(20)
#print(p1.getAge())

print(p1.age)  #10
p1.age = 18   #相当于调用了set函数，将18传值，实质调用的是@age.setter修饰的函数
print(p1.age) #相当于调用了get函数，将成员变量的值获取出来，实质调用的是@peoperty修饰的函数

p1.name = "zhangsan"
print(p1.name)
```

5.私有方法

如果类中的一个函数名前面添加__,则认为这个成员函数时私有化的

特点：也不能在外界直接调用，只能在类的内类调用

代码演示：

```
class Site():
    def __init__(self,name):
        self.name = name

    def who(self):
        print(self.name)
```

```
        self.__foo()

    #私有成员方法，只能在当前类的内部内调用
    def __foo(self):    #私有函数
        print("foo")

    def foo(self):    #公开函数
        print("foo~~~~")

    #注意：以上两个函数是两个不同的函数，不存在覆盖的问题

s = Site("千锋")
s.who()
#s.__foo()  #AttributeError: 'Site' object has no attribute 'foo'
s.foo()
```

二、继承【extends】

1.概念

如果两个或者两个以上的类具有相同的属性或者成员方法，我们可以抽取一个类出来，在抽取的类中声明公共的部分

被抽取出来的类：父类，基类，超类，根类

两个或者两个以上的类：子类，派生类

他们之间的关系：子类 继承自 父类

注意：

a.object是所有类的父类，如果一个类没有显式指明它的父类，则默认为object

b.简化代码，提高代码的复用性

2.单继承

2.1使用

简单来说，一个子类只能有一个父类，被称为单继承

语法：

父类：

class 父类类名(object):

类体【所有子类公共的部分】

子类：

class 子类类名（父类类名）：

类体【子类特有的属性和成员方法】

说明：一般情况下，如果一个类没有显式的指明父类，则统统书写为object

代码演示：

person.py文件【父类】

```
#1.定义父类
class Person(object):
 #构造函数【成员变量】
 def __init__(self,name,age):
     self.name = name
     self.age = age


 #成员方法
 def show(self):
     print("show")

 def __fun(self):
     print("fun")
```

worker.py文件【子类1】

```
from  extends01.person import Person

#2.定义子类
class Worker(Person):
 #构造函数【成员变量】
 def __init__(self,name,age,job):
     """
     self.name = name
     self.age = age
     """
     self.job = job

     #6.在子类的构造函数中调用父类的构造函数【从父类中继承父类中的成员变量】
     #方式一：super(当前子类，self).__init__(属性列表)
     #super(Worker, self).__init__(name,age)
     #方式二：父类名.__init__(self,属性列表)
     Person.__init__(self,name,age)
     #方式三：super().__init__(属性列表)
     #super().__init__(name,age)
```

```
20
21
22  #成员方法
23  def work(self):
24      print("work")
```

student.py文件【子类2】

```
1  from extends01.person import  Person
2
3  class Student(Person):
4   # 构造函数【成员变量】
5   def __init__(self, name, age, score):
6
7       Person.__init__(self,name,age)
8       self.score = score
9
10  # 成员方法
11  def study(self):
12      print("study")
```

extendsDemo01.py文件【测试模块】

```
1  #测试模块
2  from extends01.person import Person
3  from extends01.worker import Worker
4  from extends01.student import Student
5
6  #3.创建父类的对象
7  p = Person("zhangsan",10)
8  p.show()
9  #p.__fun()
10
11 #4.创建子类的对象
12 w = Worker("aaa",20,"工人")
13 w.work()
14
15 #5.子类对象访问父类中的内容
16 #结论一：子类对象可以调用父类中的公开的成员方法【因为继承，私有方法除外】
17 w.show()
18 #w.__fun()
```

```
#结论二：通过在子类的构造函数中调用父类的构造函数，子类对象可以直接访问父类中的成员变量【私有变量除外】
print(w.name,w.age,w.job)

s = Student("小明",9,90)
s.study()
s.show()
```

2.2特殊用法

代码演示：

```
#6.子类中出现一个和父类同名的成员函数,则优先调用子类中的成员函数
#子类的成员函数覆盖了父类中的同名的成员函数
s = Student("小明",9,90)
s.study()
s.show()

#7.父类对象能不能访问子类中特有的成员函数和成员变量？----->不能
per = Person("gs",10)
#per.work()
```

```
#8.slots属性能否应用在子类中
#结论三：在父类中定义slots属性限制属性的定义，子类中是无法使用，除非在子类中添加自己的限制
#父类
class Student(object):
 __slots__ = ("name","age")

#子类
class SeniorStudent(Student):
 pass


s  = Student()
s.name = "zhangsan"
s.age = 10
#s.score = 90

ss = SeniorStudent()
ss.name = "lisi"
```

```
ss.age = 20
ss.score = 60
```

总结:

继承的特点:

a.子类对象可以直接访问父类中非私有化的属性

b.子类对象可以调用父类中非私有化的成员方法

c.父类对象不能访问或者调用子类 中任意的内容

继承的优缺点:

优点:

a.简化代码，减少代码的冗余

b.提高代码的复用性

c.提高了代码的可维护性

d.继承是多态的前提

缺点:

通常使用耦合性来描述类与类之间的关系，耦合性越低，则说明代码的质量越高

但是，在继承关系中，耦合性相对较高【如果修改父类，则子类也会随着发生改变】

3.多继承

一个子类可以有多个父类

语法:

class 子类类名(父类1，父类2，父类3.。。。):

类体

代码演示:

father.py文件【父类1】

```
class Father(object):
 def __init__(self,money):
     self.money = money

 def play(self):
     print("playing")

 def fun(self):
     print("father中的fun")
```

mother.py文件【父类2】

```
class Mother(object):
 def __init__(self,faceValue):
```

```
        self.faceValue = faceValue

    def eat(self):
        print("eating")

    def fun(self):
        print("mother中的fun")
```

child.py文件【子类】

```
from extends02.father import Father
from extends02.mother import Mother

#定义子类，有多个父类
class Child(Mother,Father):
    def __init__(self,money,faceValue,hobby):
        #调用父类中的构造函数
        Father.__init__(self,money)
        Mother.__init__(self,faceValue)
        self.hobby = hobby

    def study(self):
        print("study")
```

extendsDemo03.py文件【测试模块】

```
from extends02.father import Father
from extends02.mother import Mother
from extends02.child import Child


f = Father(100000)
m = Mother(3.0)

#创建子类对象
c = Child(1000,3.0,"打游戏")
#子类对象调用父类中的成员方法
c.play()
c.eat()

#结论;如果多个父类中有相同的函数，通过子类的对象调用，调用的是哪个父类中的函数取决于在父类列表中出现的先后顺序
```

```
c.fun()
```

4.函数重写【override】

在子类中出现和父类同名的函数，则认为该函数是对父类中函数的重写

4.1系统函数重写

```
__str__
__repr__
```

代码演示：

```
class Animal(object):
 def __init__(self,name,age):
     self.name = name
     self.age = age

 #重写__str__函数,重写之后一般return一个字符串，有关于成员变量
 def __str__(self):
     return "name=%s age=%d"%(self.name,self.age)

 #重写__repr__,作用和str是相同的，优先使用str
 def __repr__(self):
     return "name=%s age=%d"%(self.name,self.age)

a = Animal("大黄",10)
print(a)   #<__main__.Animal object at 0x00000226A87AC240>
print(a.__str__())

#当一个类继承自object的时候，打印对象获取的是对象的地址，等同于通过子类对象调用父类中__str__
#当打印对象的时候，默认调用了__str__函数
#重写__str__的作用：为了调试程序

"""
总结：【面试题】
a.__str__和__repr__都未被重写的时候，使用对象调用的是__str__,此时__str__返回的是对象的地址
b.__str__和__repr__都被重写之后，使用对象调用的是__str__，此时__str__返回的是自定义的字符串
c.重写了__str__，但是没有重写__repr__，则使用对象调用的是__str__，此时__str__返回的是自定义的字符串
d.未重写__str__，但是重写了__repr__，则使用对象调用的是__repr__,此时，__repr__返回的是自定义的字符串
```

```
"""

#使用时机：当一个对象的属性有很多的时候，并且都需要打印，则可以重写__str__，可以简化代码，调试程序
```

4.2自定义函数重写

代码演示：

```
#函数重写的时机：在继承关系中，如果父类中函数的功能满足不了子类的需求，则在子类中需要重写
#父类
class People(object):
 def __init__(self,name):
     self.name = name

 def fun(self):
     print("fun")

#子类
class Student(People):
 def __init__(self,name,score):
     self.score = score
     super(Student,self).__init__(name)

 #重写;将函数的声明和实现重新写一遍
 def fun(self):
     #在子类函数中调用父类中的函数【1.想使用父类中的功能，2.需要添加新的功能】
     #根据具体的需求决定需不需要调用父类中的函数
     super(Student,self).fun()
     print("fajfhak")


s = Student("fhafh",10)
s.fun()
```

三、多态【了解】

一种事物的多种体现形式，函数的重写其实就是多态的一种体现

在Python中，多态指的是父类的引用指向子类的对象

代码演示：

```python
#父类
class Animal(object):
        pass

#子类
class Dog(Animal):
        pass

class Cat(Animal):
        pass

#定义变量
a = []   #a是list类型
b = Animal()  #b是Animal类型
c = Cat()  #c是Cat类型

#isinstance():判断一个对象是否属于某种类型【系统还是自定义的类型】
print(isinstance(a,list))
print(isinstance(b,Animal))
print(isinstance(c,Cat))

print(isinstance(c,Animal))  #True
print(isinstance(b,Dog))   #False

#结论：子类对象可以是父类类型，但是，父类的对象不能是子类类型
```

总结：

简化代码，提高代码的可读性，可维护性

四、类方法和静态方法

类方法：使用@classmethod装饰器修饰的方法，被称为类方法，可以通过类名调用，也可以通过对象调用，但是一般情况下使用类名调用

静态方法：使用@staticmethod装饰器修饰的方法，被称为静态方法，可以通过类名调用，也可以通过对象调用，但是一般情况下使用类名调用

代码演示：

```python
class Test(object):
#1.类属性
age = 100
```

```
4
5  def __init__(self,name):
6    #2.实例属性
7    self.name = name
8
9  #3.成员方法,通过对象调用
10 #必须有一个参数，这个参数一般情况下为self，self代表是当前对象
11 def func(self):
12   print("func")
13
14 #4.类方法
15 """
16 a.必须有一个参数，这个参数一般情况下为cls，cls代表的是当前类
17 b.类方法是属于整个类的，并不是属于某个具体的对象，在类方法中禁止出现self
18 c.在类方法的内部，可以直接通过cls调用当前类中的属性和方法
19 d.在类方法的内部，可以通过cls创建对象
20 """
21 @classmethod
22 def test(cls):
23   print("类方法")
24   print(cls)   #<class 'methodDemo01.Test'>
25   print(cls.age)
26
27   #6
28   #注意：cls完全当做当前类使用
29   c = cls("hello")
30   c.func()
31
32 #7.静态方法
33 @staticmethod
34 def show():
35   print("静态方法")
36
37 t = Test("hjfsh")
38 t.func()
39
40 #5,.调用类方法
41 Test.test()   #类名.类方法的名称()
42 t.test()       #对象.类方法的名称()
43
```

```
#7。调用静态方法
Test.show()
t.show()
```

总结：实例方法【成员方法】、类方法以及静态方法之间的区别

a.语法上

实例方法：第一个参数一般为self，在调用的时候不需要传参，代表的是当前对象【实例】

静态方法：没有特殊要求

类方法：第一个参数必须为cls，代表的是当前类

b.在调用上

实例方法：只能对象

静态方法：对象 或者 类

类方法：对象 或者 类

c.在继承上【相同点】

实例方法、静态方法、类方法：当子类中出现和父类中重名的函数的时候，子类对象调用的是子类中的方法【重写】

代码演示：

```
class SuperClass(object):
@staticmethod
def show():
  print("父类中的静态方法")

@classmethod
def check(cls):
  print("父类中的类方法")

class SubClass(SuperClass):
pass

s = SubClass()
s.show()
s.check()
```

注意：注意区分三种函数的书写形式，在使用，没有绝对的区分

五、类中的常用属性

```
__name__
        通过类名访问，获取类名字符串
```

```
        不能通过对象访问，否则报错

__dict__
        通过类名访问，获取指定类的信息【类方法，静态方法，成员方法】，返回的是一个字典
        通过对象访问，获取的该对象的信息【所有的属性和值】，，返回的是一个字典

__bases__
        通过类名访问，查看指定类的所有的父类【基类】
```

代码演示：

```
class Animal(object):
def __init__(self,arg):
  super(Animal, self).__init__()
  self.arg = arg


class Tiger(Animal):
age = 100
height = 200

def __init__(self,name):
  #super(Tiger, self).__init__(name)
  self.name = name

def haha(self):
  print("haha")

@classmethod
def test(cls):
  print("cls")

@staticmethod
def show():
  print("show")


if __name__ == "__main__":

#1.__name__
print(Tiger.__name__)  #Tiger
```

```
31
32 t = Tiger("")
33 #print(t.__name__)  #AttributeError: 'Tiger' object has no attribute '__name__'
34
35 #2.__dict__
36 print(Tiger.__dict__)  #类属性，所有的方法
37 print(t.__dict__)   #实例属性
38
39 #3.__bases__，获取指定类的所有的父类，返回的是一个元组
40 print(Tiger.__bases__)
```

六、单例设计模式【扩展】

1.概念

什么是设计模式

经过已经总结好的解决问题的方案

23种设计模式，比较常用的是单例设计模式，工厂设计模式，代理模式，装饰模式

什么是单例设计模式

单个实例【对象】

在程序运行的过程中，确保某一个类只能有一个实例【对象】，不管在哪个模块中获取对象，获取到的都是同一个对象

单例设计模式的核心：一个类有且仅有一个实例，并且这个实例需要应用在整个工程中

2.应用场景

实际应用：数据库连接池操作-----》应用程序中多处需要连接到数据库------》只需要创建一个连接池即可，避免资源的浪费

3.实现

3.1模块

Python的模块就是天然的单例设计模式

模块的工作原理：

import xxx,模块被第一次导入的时候，会生成一个.pyc文件，当第二次导入的时候，会直接加载.pyc文件，将不会再去执行模块源代码

3.2使用new

```
1 __new__（）:实例从无到有的过程【对象的创建过程】
```

代码演示：

```python
class Singleton(object):
#类属性
instance = None

#类方法
@classmethod
def __new__(cls, *args, **kwargs):
  #如果instance的值不为None，说明已经被实例化了，则直接返回；如果为NOne，则需要被实例化
  if not cls.instance:
      cls.instance = super().__new__(cls)

  return cls.instance

class MyClass(Singleton):
pass

#当创建对象的时候自动被调用
one = MyClass()
two = MyClass()

print(id(one))
print(id(two))

print(one is two)
```

Day13

一、错误和异常

1.概念

两种容易辨认的错误

语法错误：一些关于语法的错误【缩进】

异常：代码完全正确，但是，程序运行之后，会报出 的错误

exception/error

代码演示：

```
list1 = [23,54,6,6]
print(list1[2])
print(list1[3])
print(list1[4])

print("over")

"""
6
6
Traceback (most recent call last):
File "C:/Users/Administrator/Desktop/SZ-Python/Day15Code/textDemo01.py", line 4, in <module>
 print(list1[4])
IndexError: list index out of range
"""
```

异常特点：当程序在执行的过程中遇到异常，程序将会终止在出现异常的代码处，代码不会继续向下执行

解决问题：越过异常，保证后面的代码继续执行【实质：将异常暂时屏蔽起来，目的是为了让后面的代码的执行不受影响】

2.常见的异常

NameError:变量未被定义

TypeError:类型错误

IndexError:索引异常

keyError:

ValueError:

AttributeError:属性异常

ImportError:导入模块的时候路径异常

SyntaxError:代码不能编译

UnboundLocalError:试图访问一个还未被设置的局部变量

3.异常处理方式【掌握】

捕获异常

抛出异常

3.1捕获异常

try-except-else

语法：

try:

可能存在异常的代码

except 错误表示码 as 变量：

语句1

except 错误表示码 as 变量：

语句2

。。。

else:

语句n

说明：

a.try-except-else的用法类似于if-elif-else

b.else可有可无，根据具体的需求决定

c.try后面的代码块被称为监测区域【检测其中的代码是否存在异常】

d.工作原理：首先执行try中的语句，如果try中的语句没有异常，则直接跳过所有的except语句，执行else；如果try中的语句有异常，则去except分支中进行匹配错误码，如果匹配到了，则执行except后面的语句；如果没有except匹配，则异常仍然没有被拦截【屏蔽】

代码演示：

```
#一、try-except-else的使用

#1.except带有异常类型
try:
 print(10 / 0)
except ZeroDivisionError as e:
 print("被除数不能为0",e)

print("~~~~")
"""
总结：
a.try-except屏蔽了异常，保证后面的代码可以正常执行
b.except ZeroDivisionError as e相当于声明了一个ZeroDivisionError类型的变量【对象】，变量e中携带了错误的信息
"""

#2.try后面的except语句可以有多个
class Person(object):
```

```
 __slots__ = ("name")
try:
 p = Person()
 p.age = 19

 print(10 / 0)
except AttributeError as e:
 print("属性异常",e)
except ZeroDivisionError as e:
 print("被除数不能为0",e)

print("over")

"""
总结:
a.一个try语句后面可以有多个except分支
b.不管try中的代码有多少个异常，except语句都只会被执行其中的一个，哪个异常处于try语句的前面，则先先执行对应的except语句
c.后面的异常不会报错【未被执行到】
"""

#3.except语句的后面可以不跟异常类型
try:
 print(10 / 0)
except:
 print("被除数不能为0")


#4.一个except语句的后面可以跟多种异常的类型
#注意：不同的异常类型使用元组表示
try:
 print(10 / 0)
except (ZeroDivisionError,AttributeError):
 print("出现了异常")


#5.else分支
try:
 print(10 / 4)
except ZeroDivisionError as e:
```

```
 print("出现了异常",e)
else:
 print("hello")

"""
总结:
a.如果try中的代码出现了 异常，则直接去匹配except，else分支不会被执行
b.如果try中的代码没有出现异常，则try中的代码正常执行，except不会被执行，else分支才会被执行
"""

#6.try中不仅可以直接处理异常，还可以处理一个函数中的异常
def show():
 x = 1 / 0

try:
 show()
except:
 print("出现了异常")

#7.直接使用BaseException代替所有的异常
try:
 y = 10 / 0
except BaseException as e:
 print(e)

"""
总结：在Python中，所有的异常其实都是类，他们都有一个共同的父类BaseException，可以使用BaseException将所有异常“一网打尽”
"""
```

try-except-finally

语法:

try:

可能存在异常的代码

except 错误表示码 as 变量:

语句1

except 错误表示码 as 变量:

语句2

。。。

finally:
语句n
说明:不管try中的语句是否存在异常，不管异常是否匹配到了except语句，finally语句都会被执行
作用：表示定义清理行为，表示无论什么情况下都需要进行的操作
代码演示：

```
#二、try-except-finally的使用

#1.
try:
 print(10 / 5)
except ZeroDivisionError as e:
 print(e)

finally:
 print("finally被执行")


#2.特殊情况
#注意：当在try或者except中出现return语句时，finally语句仍然会被执行
def show():
 try:
     print(10 / 0)
     return
 except ZeroDivisionError as e:
     print(e)

 finally:
     print("finally被执行~~~~")

show()
```

3.2抛出异常

raise抛出一个指定的异常对象
语法：raise 异常对象 或者 raise
说明：异常对象通过错误表示码创建，一般来说错误表示码越准确越好
代码演示：

```
#raise的使用主要体现在自定义异常中
```

```
2
3  #1.raise表示直接抛出一个异常对象【异常是肯定存在的】
4  #创建对象的时候，参数表示对异常信息的描述
5  try:
6   raise NameError("hjafhfja")
7  except NameError as e:
8   print(e)
9
10 print("over")
11
12 """
13 总结:
14 通过raise抛出的异常，最终还是需要通过try-except处理
15 """
16
17 #2.如果通过raise抛出的异常在try中不想被处理，则可以通过raise直接向上抛出
18 try:
19  raise NameError("hjafhfja")
20 except NameError as e:
21  print(e)
22  raise
```

4.assert断言

对某个问题做一个预测，如果预测成功，则获取结果；如果预测失败，则打印预测的信息

代码演示:

```
1  def func(num,divNum):
2
3   #语法：assert表达式，当出现异常时的信息描述
4   #assert关键字的作用：预测表达式是否成立，如果成立，则执行后面的代码；如果不成立，则将异常的描述信息打印出来
5   assert (divNum != 0),"被除数不能为0"
6
7   return  num / divNum
8
9  print(func(10,20))
10 print(func(10,0))
```

5.自定义异常

实现思路：
a.定义一个类，继承自Exception类
b.书写构造函数，属性保存异常信息【调用父类的构造函数】
c.重写str函数，打印异常的信息
d.定义一个成员函数，用来处理自己的异常
代码演示：

```
class MyException(Exception):
 def __init__(self,msg):
     super(MyException,self).__init__()
     self.msg = msg

 def __str__(self):
     return self.msg

 def handle(self):
     print("出现了异常")

try:
  raise MyException("自己异常的类型")
except MyException as e:
  print(e)
  e.handle()
```

二、文件读写

1.概念

在Python中，通过打开文件生成一个文件对象【文件描述符】操作磁盘上的文件，操作主要有文件读写

2.普通文件的读写

普通文件包含：txt文件，图片，视频，音频等

2.1读文件

操作步骤：
a.打开文件：open（）
b.读取文件内容：read()
c.关闭文件:close()

说明：最后一定不要忘了文件关闭，避免系统资源的浪费【因为一个文件对象会占用系统资源】
代码演示：

```
#一、打开文件
"""
open(path,flag[,encoding,errors])
path:指定文件的路径【绝对路径和相对路径】
flag:打开文件的方式
 r:只读、
 rb:read binary,以二进制的方式打开，只读【图片，视频，音频等】
 r+:读写

 w:只能写入
 wb:以二进制的方式打开，只能写入【图片，视频，音频等】
 w+:读写

 a:append,如果一个文件不为空，当写入的时候不会覆盖掉原来的内容
encoding：编码格式：gbk,utf-8
errors:错误处理
"""
path = r"C:\Users\Administrator\Desktop\SZ-Python\Day15\笔记\致橡树.txt"
#调用open函数，得到了文件对象
f = open(path,"r",encoding="gbk")

"""
注意：
a.以r的方式打开文件时，encoding是不是必须出现
 如果文件格式为gbk,可以不加encoding="gbk"
 如果文件格式为utf-8,必须添加encoding="utf-8"
b。如果打开的文件是图片，音频或者视频等，打开方式采用rb,但是，此时，不能添加encoding="xxx"
"""

#二、读取文件内容
#1.读取全部内容    ***********
#str = f.read()
#print(str)

#2.读取指定的字符数
#注意：如果每一行的结尾有个"\n",也被识别成字符
"""
```

```
str1 = f.read(2)
print(str1)
str1 = f.read(2)
print(str1)
str1 = f.read(2)
print(str1)


#3.读取整行，不管该行有多少个字符     *********
str2 = f.readline()
print(str2)
str2 = f.readline()
print(str2)
"""

#4.读取一行中的指定的字符
#str3 = f.readline(3)
#print(str3)

#5.读取全部的内容，返回的结果为一个列表，每一行数据为一个元素
#注意：如果指明参数，则表示读取指定个数的字符
str4 = f.readlines()
print(str4)

#三、关闭文件
f.close()
```

其他写法：

```
#1.读取文件的简写形式
#with open()  as 变量

#好处：可以自动关闭文件，避免忘记关闭文件导致的资源浪费
path = "致橡树.txt"
with open(path,"r",encoding="gbk") as f:
 result = f.read()
 print(result)

#2.
try:
 f1 = open(path,"r",encoding="gbk")
```

```
 print(f1.read())
except FileNotFoundError as e:
 print("文件路径错误",e)
except LookupError as e:
 print("未知的编码格式",e)
except UnicodeDecodeError as e:
 print("读取文件解码错误",e)
finally:
 if f1:
     f1.close()
```

读取图片等二进制文件：

```
#1.
f = open("dog.jpg","rb")

result = f.read()
print(result)

f.close()

#2
with open("dog.jpg","rb") as f1:
 f1.read()

#注意：读取的是二进制文件，读取到的内容为
\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x00H\x00H\x00\x00\xff\xdb\x0
```

2.2写文件

操作步骤：

a.打开文件：open()

b.写入数据：write()

c.刷新管道【内部缓冲区】：flush()

d.关闭文件：close().

代码演示：

```
path = "file1.txt"

#1.打开文件
#注意：写入文件的时候，文件可以不存在，当open的时候会自动创建文件
```

```
#读取文件的时候，文件必须先存在，才能open
f = open(path,"w",encoding="utf-8")

#2.写入数据
#注意：将数据写入文件的时候，默认是没有换行的，如果向换行，则可以手动添加\n
f.write("Python高薪就业，走上人生巅峰")

#3.刷新数据缓冲区
#作用：加速数据的流动，保证缓冲区的流畅
f.flush()

#4.关闭文件
f.close()

#简写形式
with open(path,"w",encoding="utf-8") as f1:
 f1.write("hello")
 f1.flush()
```

3.编码和解码

编码：encode，字符串类型转换为字节类型

解码：decode，字节类型转换为字符串类型

注意：编码和解码的格式必须保持一致

代码演示：

```
path = "file2.txt"

#编码:字符串----》字节
with open(path,"wb") as f1:
 str = "today is a good day 今天是个好天气"
 f1.write(str.encode("utf-8"))

#解码：字节----->字符串
with open(path,"rb") as f2:
 data = f2.read()
 print(data)
 print(type(data))

 newData = data.decode("utf-8")
```

```
print(newData)
print(type(newData))
```

4.csv文件的读写

csv:逗号分隔值【Comma Separated Values】

一种文件格式，.csv,本质是一个纯文本文件，可以作为不同程序之间数据交互的格式

打开方式:记事本，excel

4.1读文件

代码演示:

```
#C:\Users\Administrator\Desktop\SZ-Python\Day15\笔记\text.csv
import  csv


#方式一：三部曲
def readCsv1(path):
 #1.打开文件
 csvFile = open(path,"r")

 #2.将文件对象封装成可迭代对象
 reader= csv.reader(csvFile)

 #3.读取文件内容
 #遍历出来的结果为列表
 for item in reader:
     print(item)

 #4.关闭文件
 csvFile.close()

readCsv1(r"C:\Users\Administrator\Desktop\SZ-Python\Day15\笔记\text.csv")

#方式二：简写
def readCsv2(path):
 with open(path,"r") as f:
     reader = csv.reader(f)
     for item in reader:
         print(item)
```

```
29
30 readCsv2(r"C:\Users\Administrator\Desktop\SZ-Python\Day15\笔记\text.csv")
```

4.2写文件

代码演示:

```
1  import  csv
2
3  #1.从列表写入数据
4  def writeCsv1(path):
5   infoList = [['username', 'password', 'age', 'address'],['zhangsan', 'abc123', '17',
   'china'],['lisi', 'aaabbb', '10', 'england']]
6
7   #1.打开文件
8   #注意：如果不设置newline，每一行会自动有一个空行
9   csvFile = open(path,"w",newline="")
10
11  #2.将文件对象封装成一个可迭代对象
12  writer = csv.writer(csvFile)
13
14  #3.写入数据
15  for i in range(len(infoList)):
16      writer.writerow(infoList[i])
17
18  #4.关闭文件
19  csvFile.close()
20
21 writeCsv1("file3.csv")
22
23 #2.从字典写入文件
24 def writeCsv2(path):
25  dic = {"张三":123,"李四":456,"王麻子":789}
26  csvFile = open(path, "w", newline="")
27  writer = csv.writer(csvFile)
28
29  for key in dic:
30      writer.writerow([key,dic[key]])
31
32  csvFile.close()
33
```

```
#3.简写形式
def writeCsv3(path):
 infoList = [['username', 'password', 'age', 'address'], ['zhangsan', 'abc123', '17', 'china'],
             ['lisi', 'aaabbb', '10', 'england']]
 with open(path, "w", newline="") as f:
     writer = csv.writer(f)

     for rowData in infoList:
         writer.writerow(rowData)
```