**Python学习**

**Python的三大特点**

**1.完全面向对象**：函数、模块、字符串、数字都是对象，在python中一切都是对象；完全支持继承、重载、多重继承；支持重载运算符、也支持泛型设计。

**2.拥有一个强大的标准库**：python标准库提供了系统管理、网络通信、文本处理、数据库接口、图形系统、XML处理等额外功能。

**3.大量的第三方模块：**模块包括人工智能、机器学习、科学计算、Web开发、数据库接口、图形系统等多个领域。

**Python的优缺点**

优点：简单易学、免费开源、面向对象、丰富的标准库和第三方库、可扩展性。

缺点：运行速度、国内市场较小、中文资料匮乏。

**注释**

注释的作用：使用自己熟悉的语言，在程序中对某些代码进行标注说明，增强程序的可读性。

单行注释（行注释）

**多行注释（块注释）：一对连续的三个引号（单、双引号都可以）。**

**算数运算符**

+ 加

- 减

\* 乘

/ 除

// 取整除

% 取余数

\*\* 幂

**程序执行原理**

**计算机三大件**

**程序执行原理**

**程序的作用**

**计算机三大件**

1.CPU：中央处理器，负责处理数据/计算。

2.内存：临时存储数据，速度快，空间小，价格高。

3.硬盘：永久存储数据，速度慢，空间大，价格便宜。

**程序执行原理**



1.程序运行之前，程序是保存在硬盘上的。

2.当要运行一个程序时：操作系统首先让CPU把程序复制到内存中；CPU执行内存中的程序代码。

**程序要执行，首先要加载到内存中。**

**Python执行原理**



1.操作系统首先让CPU把Python解释器的程序复制到内存中。

2.Python解释器根据语法规则，从上到下让CPU翻译Python程序中的代码。

3.CPU负责执行翻译完成的代码。

**程序的作用**

程序就是用来处理数据的，变量用来存储数据。

**PyCharm**

**断点调试、单步执行**：按下shift+f9，进入Debugger，按下f7按钮进行单步执行。

**变量**

变量类型：整型（int）、浮点型（float）、布尔型（bool）、复数型（complex）、字符（str）、列表、元组、字典。

数字型变量之间可以直接进行计算。

字符串变量之间使用“+”进行拼接字符串，使用“\*”重复拼接。

**变量的输入**：需要用户在键盘上输入信息，可以使用input函数。

input函数的基本使用

字符串变量 = input（‘提示信息：’）

**变量的格式化输出**

**格式化字符** **含义**

%s 字符串

%d 有符号十进制整数，%5d五位整数，不足在前面添0

%f 浮点数，%.2f表示小数点后两位小数

%% 输出%

**标识符**

标识符可以由字母、下划线、数字组成。

不能以数字开头。

不能与关键字重名。

**关键字**

关键字就是python内部已经使用的标识符

关键字具有特殊的功能和含义

开发者不能定义和关键字相同名字的标识符

**变量的命名规则**

可以视为是一种惯例，并没有绝对的强制性，目的是为了增加代码的识别和可读性。（python中的标识符是区别大小写的）

**判断语句（if）**

**if-elif-else判断语句基本语法**

if 要判断的条件：

条件成立时，要做的事情

elif 要判断的条件:

条件成立时，要做的事情

else：

条件不成立时，要做的事情

if语句以及缩进部分是一个完整的代码块。

（tab键相当于4个空格）

**逻辑运算**

在程序开发中，通常在判断条件时，会需要同时判断多个条件；

只有多个条件都满足，才能够执行后续代码，此时要用到逻辑运算符；

**逻辑运算符**可以把多个条件按照逻辑进行连接，变成更复杂的条件；

Python中的逻辑运算符包括：与and/或or/非not三种。

**与and**

条件1 and 条件2：

......

两个条件同时满足，返回True，

只要有一个条件不满足，返回False。

**或or**

条件1 or 条件2：

......

两个条件只要有一个满足，就会返回True，

只有当两个条件都不满足时，返回False。

**not 条件**

条件 结果

成立 不成立

不成立 成立

**if的嵌套**

在开发中，使用if进行条件判断，如果希望**在条件成立的执行语句中**再增加**条件判断**，就可以使用**if的嵌套**。

if的嵌套的应用场景：在之前条件满足的前提下，再额外增加的判断。

**if的嵌套的语法格式**

if 条件1：

条件1满足时执行的代码

...

if 条件1基础上的条件2：

条件2满足时执行的代码

...

#条件2不满足时

else:

条件2不满足时执行的代码

...

#条件1不满足时

else:  
 条件1不满足时执行的代码

...

**多行缩进：**

**框选住需要缩进的代码，按下tab键；缩进太多需要返回，按下shift+tab。**

**Random包**

random.randint(a,b) 返回[a,b]之间的整数，包含a,b。

**循环**

**程序的三大流程**

顺序：从上到下，顺序执行代码。

分支：根据条件判断，决定执行代码的分支。

循环：让特定代码重复执行。

**while循环基本使用**  
循环的作用就是让指定的代码重复地执行。

**while循环**最常用的场景就是让**执行的代码按照指定的次数重复**执行。

**死循环**

缺少在循环内部的修改循环的判断条件，导致循环持续执行，程序无法终止。

**赋值运算符**

赋值运算符中间不能添加空格

= 简单的赋值运算符

+= 加法赋值运算符

-= 减法赋值运算符

\*= 乘法赋值运算符

/= 除法赋值运算符

//= 取整除赋值运算符

%= 取模（余数）赋值运算符

\*\*= 幂赋值运算符

**循环计算**

在程序开发中，通过利用循环达到重复计算的需求。

**break和continue**

break和continue是专门在**循环体内部**使用的关键字。

**break：**某一条件满足时，退出循环，不再执行后续重复的代码。

**continue：**某一条件满足时，不执行后续重复的代码。

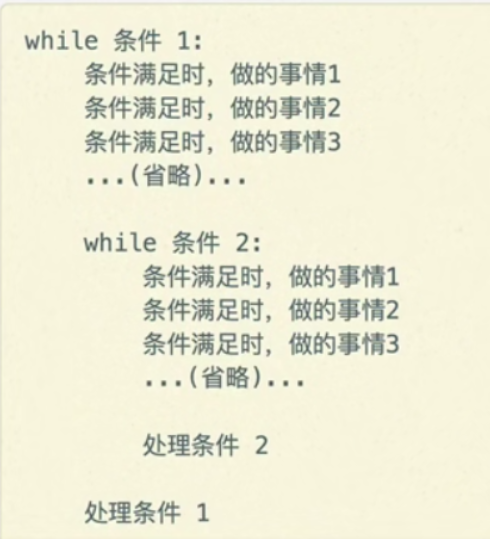
break和continue只针对**当前所在循环**有效。

在使用continue之前，需要确认**循环的计数是否改变**，否则可能**导致死循环**。

**循环嵌套**

**while循环嵌套**

while循环嵌套：while里面还有while。



print函数输出内容之后，会自动在内容末尾增加一个换行

如果不希望在末尾增加换行，可以在print函数输出内容的后面增加**，end=””**

**转义字符：**

\t 横向制表符，输出时在垂直方向对齐。

\n 输出一个换行符。

\\ 反斜杠符号

\’ 单引号

\” 双引号

\r 回车

**函数基础**

**函数的概念**：就是把具有独立功能的代码块组织为一个小模块，在需要的时候调用。

**函数的使用步骤：**  
1.定义函数------**封装**独立的功能

2.调用函数------享受**封装**的成果

**函数的作用：**在开发程序时。，使用函数可以提高编写的效率以及代码的**重用**。

**函数的定义**

格式：  
def 函数名():  
 函数封装的代码

...

1.def是define的缩写。

2.函数名称应该能够表达函数封装代码的功能，方便调用。

3.函数名称的命名应该符合标识符的命名规则。

可以由字母、数字、下划线组成。

不能以数字开头。

不能与关键字重名。

**函数的调用**

通过函数名（）即可完成对函数的调用。

**PyCharm的调试工具**

**F8** step over 可以单步执行代码，会把函数调用看作是一行代码**直接执行**。

**F7** step into可以单步执行代码，如果是函数，会**进入函数内部**。

**函数的文档注释**

给函数**添加注释**，应该在**定义函数的下方**，使用**连续的三对引号**。

在连续的三对引号之间编写对函数的说明文字。

在函数调用位置，按下ctrl+Q可以查看函数说明信息。

**函数的参数**

**函数参数的使用**

在函数名的后面的小括号内部填写参数。

多个参数之间使用“，”分隔。

**参数的作用**

函数，把具有独立功能的代码块组织为一个小模块，在需要的时候调用。

函数的参数，增加函数的通用性，针对相同的数据处理逻辑，能够适应更多的数据。

1.在函数内部，把参数当做变量来使用，进行需要的数据处理。

2.函数调用时，按照定义的顺序，把希望在函数内部处理的数据，通过参数传递。

**形参和实参**

形参：定义函数时，小括号中的参数，用来接收参数，在函数内部作为变量使用。

实参：调用函数时，小括号中的参数，把数据传递到函数内部使用。

**函数的返回值**

一个函数执行结束后，告诉调用者一个结果，以便调用者针对结果做处理。

返回值是函数完成工作后， 最后给调用者的结果。

在函数中使用**return**关键字可以返回结果。

调用函数一方，可以使用变量来接收函数的返回结果。

**注：return表示返回，后续的代码都不会被执行。**

**函数的嵌套调用**

**定义**：一个函数里面又调用了另外一个函数，这就是函数嵌套调用。

**对函数形参进行说明，点击函数名，点击insert documentation ...**

**模块**

**模块是Python程序架构的一个核心概念**

模块就好比是工具包，如果要使用，就得**导入import**这个模块。

每一个以.py结尾的python源代码文件都是一个**模块**。

在模块中定义的**全局变量、函数**都是模块能够提供给外界直接使用的工具。

小结：

可以在**一个Python文件**中**定义变量或者函数**。

然后再**另外一个文件**中使用**import导入**这个模块

导入之后，可以使用**模块名.变量/模块名.函数**的方式，使用变量或函数。

**模块可以让曾经编写的代码方便的被复用。**

**模块名也是一个标识符**

**Pyc文件**

pyc文件是由Python解释器将模块的源码转换为字节码

Python这样保存的字节码是作为一种启动速度的优化。

**高级变量类型**

**列表**

**列表的定义**

List（列表）是Python中使用**最频繁**的数据类型，在其他语言中通常叫**数组**；

专门用来**存储一串信息**；

列表用**[]**定义，**数据**之间用“，”分隔；

列表的**索引**从“0”开始。

**索引**就是数据在**列表中的位置编号**，又可以被称为**下标**。

**对列表进行操作：**

len（列表）：获取列表的长度是n+1。

列表.count（数量）：数据在列表中出现的次数。

列表.sort()：升序排序。

列表.sort(reverse=True)：降序排序。

列表.reverse()：反转、逆序。

列表[索引]：在列表中取值。

列表.index(数据)：获取数据第一次出现时的索引。

列表.insert(索引，数据)：在指定位置插入数据。

列表.append(数据)：在列表末尾增加数据。

列表.extend(列表2)：将列表2的数据追加到列表1。

del 列表[索引]：删除指定索引的数据。

列表.remove[数据]：删除第一个出现的指定数据。

列表.pop：删除末尾数据。

列表.pop(索引)：删除指定索引数据。

列表.clear()：清空整个列表。

**关键字、函数、方法**

关键字是Python内置的、具有特殊意义的标识符。

关键字后面不需要使用括号。

函数封装了独立功能，可以直接调用。

函数需要死记硬背。

**函数名（参数）**

方法和函数类似，同样是封装了独立的功能。

方法需要通过对象来调用，表示针对这个对象要做的操作。

**对象.方法名（参数）**

**列表的循环遍历**

**遍历**就是**从头到尾依次**从**列表**中获取数据

在**循环体内部**针对**每一个元素**，执行**相同的操作**。

在Python中为了提高列表的遍历效率，专门提供了**迭代iteration遍历**。

使用**for**就能过实现迭代遍历。



**元祖**

**元祖的定义**

**Tuple**（元祖）与列表类似，不同之处在于**元祖的元素不能修改**。

元祖表示多个元素组成的序列。

元祖的应用场景：

1.用于**存储一串信息**，数据至今用“，”分隔

2.元祖用（）定义

3.元祖的索引**从0开始**

**索引**就是数据在元祖中的**位置编号**。

**元祖中只包含一个元素时，需要在元素后面增加一个逗号,否则类型不是元祖。**

在元祖中**按照索引取数据**时，同样**使用[]**。

**创建空元祖： empty\_tuple = ()**

len(元祖) 元祖长度

元祖.count(元素) 元素在元祖中出现的次数

元祖.index(元素) 元素在元祖中的索引

元祖[索引] 取出元祖中对应索引的元素

**元祖的循环遍历**

**取值**就是从**元祖**中获取存储在**指定位置的数据**。

**遍历**就是**从头到尾依次**从**元祖**中获取数据。

**在Python中，可以使用for循环遍历所有非数字类型的变量：列表、元祖、字典、字符串。**

**应用场景**

**函数的参数和返回值**，一个函数可以接收任意多个参数，或者依次返回多个参数；

**格式字符串**，格式化字符串后面的（）本质上就是一个元祖；

**让列表不可以被修改**，保护数据安全。

**元祖和列表之间的转换**

元祖-->列表 列表-->元祖

list(元祖) tuple(列表)

列表-->元祖

number\_list = [1,2,3,4,5,6,7]

number\_tuple = tuple(number\_list)

print(type(number\_tuple)) 输出结果:class ‘tuple’

**字典**

**字典的定义**

字典是**除列表之外**Python中**最灵活**的数据类型。

字典同样可以用来**存储多个数据。**

通常用来存储描述一个物体的相关信息。

**列表**是**有序**的对象集合。

**字典**是**无序**的对象集合。

字典用**{}**定义；

字典使用**键值对**存储数据，键值对之间**使用“，”分隔**；

**键（key）是索引**；

**值（value）是数据**；

**键和值**之间使用**：**分隔；

**键**必须是**唯一**的；

**值**可以取**任意数据类型**，**键**只能是**字符串、数字或元祖**。

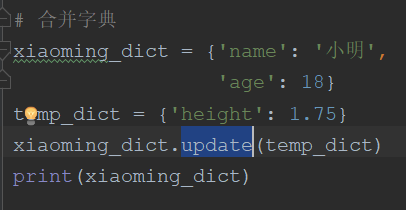
len(字典) 获取字典的键值对数量

字典.keys() 所有key列表

字典.values() 所有value列表

字典.items() 所有（key, value）元祖列表

**合并字典**



**清空字典**

使用clear关键字。

**字典的循环遍历**

**遍历就是依次从字典中获取所有键值对。**

for k in dict:

print(‘%s - %s’ % (k, dict[k]))

**字符串**

**定义**：**字符串**就是**一串字符**，是编程语言中表示文本的数据类型。

在Python中可以使用**一对双引号**“”或者**一对单引号**‘’定义字符串。

可以使用**索引**获取一个字符串中指定位置的字符，**索引计数从0开始**。

可以使用**for循环遍历**字符串中的每一个字符。

**大多数编程语言都是用“”来定义字符串。**

len(字符串) 获取字符串的长度；

字符串.count(字符串) 小字符串在大字符串中出现的次数

字符串[索引] 从字符串中取出单个字符

字符串.index(字符串) 获得小字符串第一次出现的索引

**字符串使用方法**：https://www.cnblogs.com/huangping/p/7724701.html

**使用isspace方法判断空白字符。**

**字符串的查找与替换**

str.find(“str”) 返回查找到字符串的第一个字符的索引

str.replace(“old str”, “new str”) 替换字符串

**find 查找的字符串如果不存在，返回-1。**

**replace 不会修改原有字符串的内容。**

**去除空白字符**

string.strip() 去除string左右两边的空白字符

空白字符包括：\r,\t,\n,空格。

**拆分和拼接字符串**

string.split() 拆分字符串

string.join(seq) 拼接字符串

**字符串的切片**

**切片**方法适用于**字符串、列表、元祖。**

**切片**使用**索引值**来限定范围，从一个大的字符串中切除小的字符串。

**列表和元祖**都是**有序**的集合，都能够**通过索引值**获取到对应的数据。

**字典**是一个**无序**的集合，使用**键值对**保存数据。

**字符串[开始索引:结束索引:步长]**

**逆序时：步长指定为-1.**

注意：1.指定的区间是左闭右开型[开始索引,结束索引]，从起始位开始，到结束位前一位结束（不包含结束位）。

2.从头开始，开始索引数字可以省略，冒号不能省略。

3.到末尾结束，结束索引数字可以省略，冒号不能省略。

4.步长默认为1，如果连续切片，数字和冒号都可以省略。

**公共方法**

**Python内置函数**

len(item) 计算容器中元素个数

del(item) 删除变量

max(item) 返回容器中元素最大值 如果是字典，只对key比较

min(item) 返回容器中元素最小值 如果是字典，只对key比较

字符串比较符合以下规则：“0”<“A”<“a”

**字典之间不能比较大小**

**切片**

**运算符**

**完整的for循环语法**

in 成员运算符

在Python中**完整的for循环语法**如下：

for 变量 in 集合:

循环体代码

else:

没有通过 break 退出循环，循环结束后，会执行的代码

**无限循环**

**循环不停重复**的执行方法，可以使用break退出循环。

while True:

循环体

**TODO注释**

在#后跟上TODO，用于标记需要去做的工作。

**# TODO（作者/邮件） 显示系统菜单**

**变量进阶**

**1.变量的引用**

**变量和数据**都是保存在**内存**中的。

在Python中**函数的参数传递**以及**返回值**都是靠**引用**传递的。

**1.1引用的概念**

在Python中：

**变量和数据**都是分开存储的

**数据**保存在内存中的一个位置（地址）

**变量**中保存着数据在内存中的地址

**变量**中**记录数据的地址**，就叫做引用

使用**id（）函数**可以查看变量中保存数据所在的**内存地址**

**注意**：如果变量已经被定义，当给一个变量赋值的时，本质上是修改数据的引用。

**变量不再对之前的数据引用。**

**变量改为对新赋值的数据引用。**

**可变和不可变类型**

**不可变类型**：内存中的数据不允许被修改

**数字类型、字符串、元祖**

**可变类型：**内存中的数据可以被修改

**列表，字典**

字典中的key只能使用不可变类型的数据

**哈希**

**Python中内置有一个名字叫hash()的函数**

接收一个不可变类型的数据作为参数；

返回结果是一个整数。

**哈希是一个算法，作用是提取数据的特征码**

**局部变量和全局变量**

局部变量就是在函数内部定义的变量，只能在函数内部使用。

全局变量是在函数外部定义的变量，所有函数内部都可以使用这个变量。

**局部变量**

局部变量就是在**函数内部**定义的变量，只能**在函数内部使用**。

函数执行结束后，**函数内部的局部变量，会被系统回收。**

不同的函数，可以定义相同名字的局部变量，各用个的互不影响。

**局部变量的作用：**

**在函数内部使用，临时保存函数内部需要使用的数据**

**局部变量的生命周期**

生命周期：变量从**被创建**到**被系统回收**的过程。

局部变量在**函数执行时**才会被**创建**

**函数执行结束**后局部变量被**系统回收**

**局部变量在生命周期**内，可以用来**存储函数内部临时使用到的数据**

**全局变量**

全局变量是在**函数外部**定义的变量，**所有函数**内部都可以使用这个变量。

**1.函数不可以直接修改全局变量的引用**

在函数内部，可以**通过全局变量的引用来获取对应的数据**。

但是**不允许直接修改全局变量的引用**--使用赋值语句修改全局变量的值。

**2.在函数内部修改全局变量的值**

如果在函数中需要**修改全局变量的值**，需要**使用global**进行声明。

global num

num = 1

3.全局变量定义的位置

为了保证所有的函数都能够正确使用到全局变量，应该讲全**局变量定义在其他函数的上方**。

**函数参数和返回值的作用**

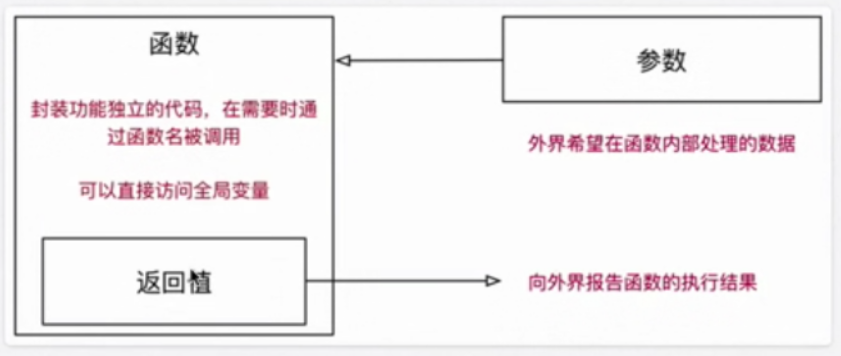
函数根据**有没有参数**以及**有没有返回值**，可以**互相组合**，一共有四种组合形式。

1.无参数，无返回值

2.无参数，有返回值

3.有参数，无返回值

4.有参数，有返回值



**是否需要增加参数或返回值的小技巧**

1.如果函数内部处理的数据不确定，就可以将外界的数据以参数传递到函数内部。

2.如果希望一个函数执行完成后，向外界汇报执行结果，就可以增加函数的返回值。

**函数的返回值进阶**

**返回值**是**函数完成工作**后，最后给**调用者**的**一个结果**。

在函数中使用**return**关键字返回结果。

调用函数的一方，可以**使用变量**来**接收**函数的返回结果。

**利用元祖可以返回多个值**

**函数的参数进阶**

**不可变和可变参数**

无论传递的参数是**可变**还是**不可变**

1.只要**针对参数**使用**赋值语句**，会在**函数内部**修改**局部变量的引用**，**不会影响到外部变量的引用。**

2.如果传递的参数是**可变类型**，在函数内部，使用**方法**修改了数据的内容，**同样会影响到外部的数据**。

3.在Python中，**列表变量**调用**+=**本质上是在执行变量的**extend方法**，不会修改变量的引用。

**缺省参数**

定义函数时，可以给**某个参数**指定一个**默认值**，具有**默认值的参数**就叫做**缺省参数**。

调用函数时，如果没有传入**缺省参数的值**，则在函数内部使用定义函数时指定的**参数默认值**。

函数的缺省参数，**将常见的值设置为参数的缺省值**，从而**简化函数的调用**。

指定函数的缺省参数：

在参数后使用赋值语句，可以指定参数的缺省值。

**缺省参数的注意事项**：

1.缺省参数的定义位置：必须保证带有默认值的缺省参数在参数列表末尾。

2.调用带有多个缺省参数的函数：在调用函数时，如果有多个缺省参数，需要指定参数名，这样解释器才能够知道参数的对应关系。

**多值参数**

定义支持多值参数的函数

有时可能需要**一个函数**能够处理的**参数个数**是**不确定**的，这个时候，就需要用到多值参数。

Python中有**两种**多值参数：

1.参数名前增加**一个\***，可以接收**元祖**。

2.参数名前增加**两个\***，可以接收**字典**。

一般在给多值参数命名时，习惯使用以下两个名字：

1.\*args------存放元祖参数，前面有一个\*。

2.\*\*kwargs---存放字典参数，前面有两个\*。

**元祖和字典的拆包**

在调用带有多值参数的函数时，如果希望：  
1.将一个**元祖变量**，直接传递给args；

2.将一个**字典变量**，直接传递给kwargs。

就可以使用**拆包**，简化参数的传递，**拆包**的方式是：  
1.在**元祖变量前**，增加**一个\***；

2.在**字典变量前**，增加**两个\***。

**递归**

定义：函数调用自身的编程技巧叫做递归。

**递归函数的特点**

**特点**

**一个函数内部调用自己**

函数内部可以调用其他函数，当然也可以调用自己。

**代码特点**

1.函数内部的**代码**是相同的，只是针对的**参数不同**，处理的**结果不同**。

2.当**参数满足一个条件**时，函数不再执行

这个条件非常重要，通常被称为**递归出口**，否则会出现**死循环**

**面向对象（OOP）**

**面向对象编程**

**面向过程---怎么做**

**面向对象---谁来做**

相对于函数，**面向对象**是**更大**的**封装**，根据**职责**在**一个对象中封装多个方法**。

1.在完成某个需求前，首先确定职责---要做的事情（方法）

2.根据职责确定不同的对象，在对象内部封装不同的方法（多个）

3.最后完成的代码，就是顺序的让不同的对象调用不同的方法。

**类和对象**

**类**是对一群具有**相同特征或者行为**的事物的一个**统称**，是**抽象的**，**不能直接使用**。

**特征**被称为**属性**。

**行为**被称为**方法**。

**类**是负责**创建对象**的

**对象**是由类创建出来的一个**具体存在**，**可以直接使用**。

由哪个类创建出来的对象，就拥有在哪个类中定义的：属性、方法。

在程序开发中，应该**先有类，再有对象**。

**类和对象的关系**

**类是模板**，**对象**是根据**类**这个模板创建出来的，应该**先有类，再有对象**。

**类**只有**一个**，而**对象**可以有**很多个**。

**不同的对象**之间**属性**可能会**各不相同**。

**类**中定义了什么**属性和方法**，**对象**中就有什么**属性和方法**，**不能多，也不能少**。

**类的设计**

**类的三要素**：  
1.**类名**，满足大驼峰命名法（每一个单词的首字母大写）

2.**属性**，具有什么**特征**

3.**方法**，具有什么**行为**

**类名的确定**

**名词提炼法**分析整个**业务流程**，出现的**名词**，通常就是找到的类。

属性和方法的确定

对对象的**特征描述**，通常可以定义为**属性**

对象具有的**行为**，通常可以定义为**方法**

**变量、数据、函数都是对象。**

**面向对象基本语法**

**1.dir内置函数**

使用内置函数dir传入**标识符/数据**，可以查看对象内的**所有属性及方法**。



**2.定义简单的类**

面向对象是更大的封装，在一个类中封装多个方法，这样通过这个类创建出来的对象，就可以直接调用这些方法了。

**2.1 定义只包含方法的类**

语法：



**方法**的定义格式和**函数**几乎一样

区别在于**第一个参数**必须是**self**

**类名**的命名规则要符合**大驼峰命名法**

**2.2 创建对象**

当一个类定义完成之后，要使用这个类来创建对象，语法格式：

对象变量 = 类名（）

在计算机中，通常使用**十六进制**表示**内存地址**

**%d**可以**以10进制**输出数字

**%x**可以**以16进制**输出数字

在类封装的方法内部，**self**就表示**当前调用方法的对象自己**。

self：哪个对象调用的方法，self就是哪个对象的引用。

在**调用方法**时，**不需要传入self参数**

在方法内部：

可以通过**self.访问对象的属性**

也可以通过**self.调用其他的对象方法**

对象应该包含有哪些**属性**，应该**封装在类的内部**。

**初始化方法**

当使用类名()**创建对象**时，会**自动**执行以下操作：  
1.在对象的内存中**分配空间**--创建对象

2.为对象的属性**设置初始值**--初始化方法（init）

这个**初始化方法**就是**\_\_init\_\_方法**，\_\_init\_\_是对象的**内置方法**。

**\_\_init\_\_方法**时**专门**用来定义一个类**具有哪些属性**的方法。

**在初始化方法内部定义属性**

在\_\_init\_\_方法内部使用**self.属性名 = 属性的初始值**就可以**定义属性**。

定义属性之后，再**使用类创建的对象**，都会**拥有该属性**。

**改造初始化方法--初始化的同时设置初始值**

如果希望在**创建对象的同时**，**就设置对象的属性**，可以对**\_\_init\_\_方法**进行改造

1.把希望设置的属性值，定义成**\_\_init\_\_方法的参数**

2.在方法内部使用**self.属性=形参**接收外部传递的参数

3.在创建对象时，使用**类名（属性1，属性2.....）**调用

**内置方法和属性**

**\_\_del\_\_方法**

当一个**对象被从内存中销毁前**，会**自动**调用\_\_del\_\_方法。

**生命周期**

1.一个对象从**调用类名（）**创建，生命周期**开始**

2.一个对象的**\_\_del\_\_方法**一旦被调用，生命周期**结束**

3.在对象的生命周期内，可以访问**对象属性**，或者让对象调用方法

**\_\_str\_\_方法**

在Python中，若希望使用print输出**对象变量**时，能够打印**自定义的内容**，就可以**使用\_\_str\_\_方法**。

**注意：\_\_str\_\_方法必须返回一个字符串**

**封装**

1.**封装**是面向对象编程的**一大特点**；

2.面向对象编程的**第一步**---将**属性和方法**封装成一个抽象的**类**中；

3.外界使用**类创建对象**，然后让**对象调用方法**；

4.**对象方法的细节**都被**封装**在**类的内部**。

**在对象的方法内部，可以直接访问对象的属性。**

**同一个类**创建的**多个对象**之间，**属性互不干扰。**

**在开发过程中，被使用的类，应该先进行开发。**

**身份运算符**

**身份运算符**用于**比较**两个对象的**内存地址**是否一致---**是否是对同一个对象的引用。**

在Python中针对None比较时，建议使用**is**判断

**运算符**  **描述**

is is是判断两个标识符是不是引用同一个对象

is not is not是判断两个标识符是不是引用不同的对象

**is和==的区别**

is用来判断**两个变量引用对象是否为同一个**

==用来判断**引用变量的值**是否相等

**私有属性和私有方法**

**应用场景**

在开发中，**对象**的**某些属性或方法**可能只希望**在对象的内部被使用**，而**不希望在外部被访问到**。

**私有属性**就是**对象**不希望公开的**属性**。

**私有方法**就是**对象**不希望公开的**方法**。

**定义方式**

在**定义属性或方法**时，在**属性名或者方法名前**增加**两个下划线**，定义的就是**私有属性或方法**。

**伪私有属性和私有方法**

在开发中，**不要使用这种方式，访问对象的私有属性和私有方法**。

Python中，并**没有真正意义的私有**。

在给**属性/方法**命名时，实际是对**名称**做了一些特殊处理，使得外界无法访问到。

**处理方式**：在**名称**前面加上 **\_类名 => \_类名\_\_名称**

**面向对象三大特征**

1.**封装**根据**职责**将**属性**和**方法封装**到一个抽象的**类**中。

2.**继承实现代码的重用**，相同的代码不需要重复的编写。

3.**多态**不同的对象调用相同的方法，产生不同的执行结果，**增加代码的灵活度**。

**继承**

**单继承**

**继承的概念**：**子类**拥有**父类**的所有**方法**和**属性**

**继承的语法**

class 类名(父类名):  
 pass

子类继承自父类，可以直接享受父类中已经封装好的方法，不需要再次开发。

子类中应该根据职责，封装子类特有的属性和方法。

**父类（基类）/子类（派生类）**

**继承的传递性**

**子类**拥有**父类**以及**父类的父类**中封装的所有的**属性和方法**。

**方法的重写**

**子类**拥有**父类**的所有**方法**和**属性**。

**子类**继承自**父类**，可以直接**享受**父类中已经封装好的方法，不需要再次开发。

当**父类**的方法实现不能满足子类需求时，可以对方法进行**重写**。

**重写父类方法有两种情况：**

**1.覆盖父类的方法：**

如果在开发中，父类的实现方法和子类的实现方法完全不同，就可以使用覆盖的方式，在子类中重新编写父类的方法实现。

具体的实现方法，就相当于在**子类中**定义了一个**和父类同名的方法并且实现**。

编写之后，在运行时，**只会调用子类中重写的方法**，而不会调用父类封装的方法。

**2.对父类方法进行扩展**

如果在开发中，**子类的方法实现**中包含**父类的方法实现**

**父类原本封装的方法实现是子类方法的一部分。**

就可以使用扩展的方式

1.**在子类中重写**父类的方法

2.在需要的位置使用**super().父类方法**，来调用父类方法的执行

3.代码其他的位置针对子类的需求，**编写子类特有的代码实现**

**关于super**

在Python中super是一个**特殊的类**；

super()就是使用super类创造出来的对象；

通常使用的场景就是在重写父类方法时，调用在父类中封装的方法实现。

**父类的私有属性和私有方法**

1.**子类对象不能**在自己的方法内部，**直接**访问到父类的**私有属性**或**私有方法**。

2.**子类对象**可以通过**父类**的**公有方法间接**访问到**私有属性**或**私有方法**。

**私有属性、方法**是对象的隐私，不对外公开，**外界**和**子类**都**不能直接访问**

**私有属性、方法**通常用于做一些内部的事情

**父类的公有方法可以访问父类的私有属性或调用父类的私有方法**

**多继承**

子类可以拥有**多个父类**，并且具有**所有父类**的**属性和方法**。

**语法**

**class 子类名(父类1,父类2......)：**

**pass**

**多继承的使用注意事项**

如果**父类之间**存在**同名的属性或方法**，应该**尽量避免**使用多继承。

**Python中的MRO -- 方法搜索顺序**

**主要用于在多继承时判断方法、属性的调用路径。 例：print(C.\_\_mro\_\_)**

**新式类和旧式类（经典类）**

Object是Python中所有对象提供的基类。

新式类：**以object为基类**的类，**推荐使用**。

旧式类：**不以object为基类**的类，**不推荐使用**。

在定义类时，如果没有父类，建议统一继承自object

**class 类名(object):  
 pass**

**多态**

多态不同的子类对象调用相同的父类方法，产生不同的执行结果。

多态可以增加代码的灵活度；

以继承和重写父类方法为前提；

是调用方法的技巧，不会影响到类的内部设计。

**类的结构**

**实例**

1.使用面向对象开发，第一步是**设计类**。

2.使用**类名()**创建对象，**创建对象**的动作有**两步**：

1.在内存中为对象**分配空间** 2.调用初始化方法为**对象初始化**

3.对象创建后，**内存**中就有了一个对象的**实实在在的存在------ -实例**

通常：

1.创建出来的**对象**叫做**类的实例**；

2.创建出来的**动作**叫做**实例化**；  
3.对象的属性叫做**实例属性**；

4.对象调用的方法叫做**实例方法**

创建出来的对象，**每一个对象**都有自己**独立的内存空间**，保存各不相同的属性。

多个对象的**方法**，**在内存中只有一份**，在调用方法时，需要**把对象的引用传递到方法内部**。

**类是一个特殊的对象**

Python中**一切都是对象**

class AAA: 定义的类属于**类对象**

obj = AAA() **实例对象**

**类对象**在内存中**只有一份**，使用**一个类**可以创建出**很多个不同的对象实例**

类对象可以拥有**类属性**和**类方法**

通过**类名.**的方式可以**访问类的属性**或者**调用类的方法**

**类属性和实例属性**

概念：**类属性**就是给**类对象**中定义的**属性**，通常用来记录与**这个类有关**的特性，**类属性不会用于**记录**具体对象的特征**。

**属性的获取机制**：  
在Python中属性的获取存在一个**向上查找机制**。

**两种方式**：**类名.类属性（推荐使用）**

对象.类属性（不推荐）

**类方法**

**类方法**就是针对**类对象**定义的方法。

在**类方法**内部可以直接访问**类属性**或者调用其他的**类方法**。

**语法： @classmethod**

**def 类方法名(cls):**

**Pass**

类方法需要用**修饰器@classmethod**来标识，告诉解释器是一个**类方法**。

类方法的**第一个参数**应该是**cls**

由**哪一个类调用**的方法，方法内的cls就是**哪一个类的引用**

这个参数和**实例方法**的第一个参数**self**类似

提示使用其他名称也可以，不过**习惯用cls**

通过**类名**.调用**类方法**，**调用方法时**，**不需要传递cls参数**。

**在方法内部：**

可以通过**cls.访问类属性**

也可以通过**cls.调用其他的类方法**

在类方法内部，可以直接使用cls访问**类属性**或**调用类方法**

**静态方法**

在开发时，如果需要在**类**中封装一个方法，这个方法：  
既**不需要**访问**实例属性**或者**调用实例方法**

也**不需要**访问**类属性**或者**调用类方法**

这个时候，可以把这个方法封装成一个**静态方法**

**语法：**

**@staticmethod**

**def 静态方法名():**

**pass**

静态方法需要用**@staticmethod**来标识，告诉解释器这是一个静态方法

通过**类名.**调用静态方法。

**小结**

1.**实例方法**---方法内部需要访问**实例属性**

**实例方法**内部可以使用**类名.**访问类属性

2.**类方法**---方法内部只需要访问**类属性**

3.**静态方法**---方法内部，不需要访问**实例属性**和**类属性**

既需要访问**类属性**，有需要访问**实例属性**，应该封装成**实例方法**

**单例设计模式**

**设计模式**

**设计模式**是**前人工作的总结和提炼**，通常，设计模式都是针对**某一特定问题**的成熟解决方案。

使用**设计模式**是为了可重用代码、让代码更容易被他人理解、保证代码的可靠性

**单例设计模式**

**目的**：让**类**创建的对象，在系统中**只有唯一的一个实例**

每一次执行**类名()**返回的对象，**内存地址都是相同的**

**\_\_new\_\_方法**

使用**类名()**创建对象时，自动调用**\_\_new\_\_方法**为对象**分配空间**

**\_\_new\_\_方法**是**object基类**提供的一个**内置静态方法**，主要作用是：  
1.在内存中为对象**分配空间**

2.**返回对象的引用**

获得对象**引用**后，将作为**第一个参数**，传递给**\_\_init\_\_方法**

**重写\_\_new\_\_方法**

**instance = super().\_\_new\_\_(cls)**

**return instance**

**异常**

**异常的概念**

程序在运行时，如果Python解释器**遇到**一个错误，**会停止程序的执行**，并且**提示一些错误信息**，这就是**异常**。

**程序停止执行并且提示错误信息**这个动作，通常称之为：**抛出异常**。

程序开发时，很难将**所有的特殊情况**都处理的面面俱到，通过**异常捕获**可以针对突发事件做集中的处理，从而保证程序的**稳定性和健壮性**。

**捕获异常**

在开发过程中，如果**对某些代码的执行不能确定是否正确**，可以增加**try(尝试)**来**捕获异常**

捕获异常最简单的**语法格式**

**try:**

**尝试执行的代码**

**except:**

**出现错误的处理**

**try尝试**，下方编写要尝试代码，不确定是否能正常执行的代码

**except如果不是**，下方编写尝试失败的代码

**错误类型捕获**

在程序执行时，可能会遇到**不同类型的异常**，并且需要**针对不同类型的异常，做出不同的响应**，这个时候，就需要捕获错误类型了。

**语法：**

**try:**

**# 尝试执行的代码**

**pass**

**except 错误类型1：  
pass**

**except (错误类型2，错误类型3):  
pass**

**except Exception as result:  
print(“未知错误%s” % result)**

当Python解释器**抛出异常**时，**最后一行错误信息的第一个单词，就是错误类型。**

**捕获未知错误**

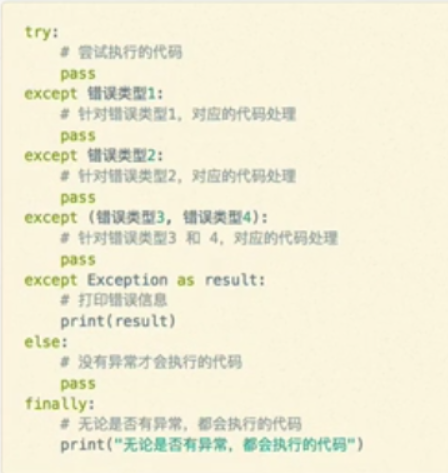
在开发时，**要预判所有可能出现的错误**，还是有一定难度的。

如果希望程序**无论出现任何错误**，都不会因为Python解释器**抛出异常而被终止**，可以增加一个except。

**语法：  
except Exception as result:**

**print(“未知错误%s” % result)**

**完整语法**



**异常的传递**

**异常的传递**---当**函数/方法**执行出现异常，会将异常**传递给函数/方法调用一方**。

如果**传递给主程序**，仍然**没有异常处理**，程序才被终止。

提示：  
在开发中，可以**在主函数中增加异常捕获**

在主函数中调用其他函数，只要**出现异常**，都会**传递到主函数的异常模块中**

这样就不需要在代码中，增加大量的异常捕获，**保证代码的整洁**。

**抛出异常**



**模块**

**模块的两种导入方式**

**1.import 导入**

import 模块名1, 模块名2

或者

import 模块名1

import 模块名2

提示：在导入模块时，每个导入应该独占一行

导入之后，通过**模块名.**来使用**模块提供的工具---全局变量、函数、类**

**使用as指定模块的别名**

如果模块的名字太长，可以使用as指定模块的名称，方便使用。

**import 模块名1 as 模块别名**

**模块别名应该符合大驼峰命名法**

**2.from...import 导入**

若希望**从某一个模块中**，**导入部分工具**，就可以使用**from...import**的方式。

**import 模块名**是**一次性**把模块中的**所有工具全部导入**，并通过**模块名/别名**访问。

**# 从模块中导入某一个工具**

**from 模块名1 import 工具名**

导入之后

不需要通过模块名.

可以直接使用模块提供的工具---全局变量、函数、类。

注意：

如果**两个模块**，存在**同名的函数**，那么**后导入模块的函数**，会**覆盖掉先导入的函数**

开发时，**import**代码统一写在**代码顶部**，容易发现冲突

一旦发生冲突，可以使用**as关键字**对其中一个工具**起一个别名**。

**原则--每个文件都应该是可以被导入的**

**包**

**概念**

包是一个**包含多个模块**的**特殊目录**

目录下有一个**特殊的文件\_\_init\_\_.py**

包名的**命名方式**和变量名一致，**小写字母 ＋ \_**

**好处**

使用**import 包名**可以一次性导入**包**中的**所有模块**

**\_\_init\_\_.py**

要在外界使用**包**中的模块，需要在**\_\_init\_\_.py**中指定**对外界提供的模块列表**

**文件**

**文件的概念及作用**

计算机中的文件，就是存储在某种长期存储设备上的一段数据。

作用：将数据长期存储起来，在需要的时候使用。

**存储方式**

在计算机中，文件是以**二进制**的方式保存在磁盘上的。

**文本文件和二进制文件**

**文本文件：**  
可以使用**文本编辑软件**查看

本质上还是二进制文件

例如：python的源程序

**二进制文件：**  
保存的内容不是给人直接阅读的，而是**提供给其他软件使用的**

例如：图片文件，音频文件，视频文件等

二进制文件不能用**文本编辑软件**查看

**文件操作步骤**

1.打开文件 open

2.读写文件

读：将文件内容读入内存 read

写：将内存内容写入文件 write

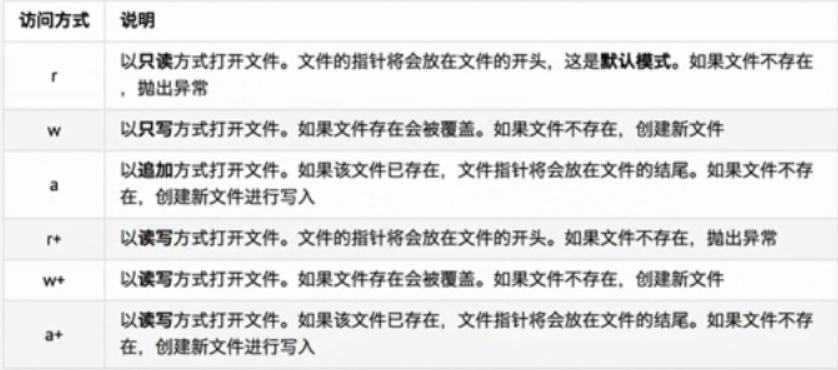
3.关闭文件 close

Open是函数，read/write/close都是方法。

**打开文件的方式**

open函数默认以只读方式打开文件，并且返回文件对象

语法：  
f = open(“文件名”, “访问方式”)



注意：使用读写的方式，会频繁的移动文件指针，会影响到文件的读写效率，开发中更多的时候会以只读、只写的方式来操作文件。

**按行读取文件的内容**

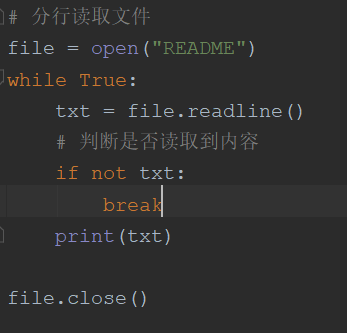
read方法默认会把文件的所有内容一次性读取到内存

如果文件太大，对内存的占用会很严重

**readline方法**

readline方法可以一次读取一行内容

方法执行后，会把文件指针移动到下一行，准备再次读取



**文件/目录的常用管理操作**



**在python2中使用中文，可以在文件的第一行增加**

**# \*-\* coding:utf8 \*-\***

或者 **# coding=utf8**

**在python2中处理中文字符串**

在含有中文字符串的引号前**增加一个小写字母u**。

**eval函数**

**eval()函数**十分强大---**将字符串**当成**有效的表达式**来**求值**并**返回计算结果**

在开发时千万不要使用eval直接转换input的结果