Python笔记

1. The first program : Hello World

print(“Hello World”)

all the symbol should be the ones of English Mode !!!

2. Python解释器 基本原理是：计算机只认识0和1

但是为什么Python能输出文字呢？

计算机本身不认识Python，但是有解释器程序

3. 使用专业的集成开发工具 第三方IDE

绝大多数是在PyCharm 内完成的

4. 字面量

什么是字面量：在代码中，被写下来的的固定的值

有六种类型的值：数字 字符串 列表 元组 集合 字典

字符串 又称之为文本 是由任意数量的字符组成

注意 字符串是需要用双引号包围起来

整数和浮点数（小数）的写法与现实中一致，只不过字符串 需要双引号

5. 注释

注释是不能被执行的，注释分为两类

第一类 单行注释 用# 开头 但是 建议井号后面要一个空格

这个空格是一个规范！！

第二类 以一对三个双引号引起来

6. 变量

学习目标 理解变量的作用及特征 掌握变量的定义方式

变量可以记录数据

定义名称 变量名称 = 变量的值

附：输出多分数据 格式直接是各个之间加逗号 如

money = 50

print(“钱包还有:”, money)

money = money - 10

print(“钱包还有:”, money)

这就说明变量具有可以改变的特征

附：加减乘除 +-\*/

7. 数据类型

掌握使用type()语句查看数据类型

理解变量无类型而数据有类型的概念

数据是有类型的，目前主要接触 string 字符串

int 整数 float 浮点型 注意这两者都是有符号的

如 print(type(“Hello World))

则会输出 <class ‘str’>

如 a = type(666)

print(a)

则会输出 <class ‘int’>

上述我们查看的都是字面量的数据 但是我们能查看变量储存的数据类型吗

答案是完全可以

但是我们说这样查看的到底是数据的类型还是变量的类型？

事实上，变量是没有类型的，但是变量储存的数据是有类型的

type()是有返回值的！！给出结果

7. 数据类型转化

掌握如何在字符串 整数 浮点数 之间 进行相互转化 并且了解注意事项

数据类型之间在特定的场合之下是可以相互转化的

这种功能是我们以后会经常使用的

如：从文件中读取的数字会被通默认为字符串，我们需要转化为数字类型

或者 学习 input()语句，默认结果为字符串但是我们需要数字也需要转化

常见转化语句

int(x) 将x转化为一个整数 float str 也是类似的

这三个语句都是有返回值的/结果

a = str(11)

print(type(a), a) 则输出 <class ‘str’> 11

注意，这里的语句既可以转化类型，又不会破坏原来的数据

注意事项：想要把字符串转化为数字，必须字符串内的内容全部是数字

万物皆可以转为字符串，因为万物带上双引号它就是字符串

同时注意浮点数转为整数

int\_num = int(11.234)

print(type(int\_num), int\_num) 则会输出 <class ‘int’> 11

这里小数点全部消失了

浮点数表达的含义肯定比整数多；转化的话会丢失精度

8. 可以 标识符

理解什么是标识符 掌握标识符的命名规则 掌握变量的命名规范

我们给很多东西起名字 变量 方法 类……

这些名字我们统一称为标识符

标识符是在编程的时候所使用的一系列名字，用于给变量、类、方法等命名

规则主要有三个：内容限定 大小写敏感 不可使用关键字

命名中只允许出现 英文 中文 数字 下划线 其余内容都不允许

但是注意：1. 不推荐使用中文 2. 数字不可以开头

3. 大小写是能完全区分的

关键字：有一系列词，他们有着特定的用途 我们不可以使用他们做标识符

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

但是也还是要注意：如果我们大小写故意分开来的话，我们不算是占用关键字的

接下来说规范：为了高级

不同的命名对象，有着不同的规则；但是我们目前只知道变量名，所以学习它相关的规范

规则：见名知意 下划线命名法 英文字母全部要小写

同时要简洁明了；多个单词组合形成变量名的时候，要用下滑线分隔

9. 运算符

常见：数学（算数）运算符 赋值运算符

+ - \* / //（取整除，完全只是取整数） % 取余数 \*\*指数

= 赋值运算符

复合赋值运算符 += 加法赋值

%=取模赋值 \*\*=幂赋值 //=取证赋值运算符

10. 字符串的扩展

（1）字符串的三种定义方式

字符串有三种定义法 单引号/双引号/三引号 定义法

三引号定义法 和多行注释一样 ，同时支持换行操作

使用变量接受它，它就是字符串

但是不用变量接受它，就可以作为多行注释使用

问题：如果定义的字符串本身就包含单引号、双引号呢

单引号定义法：可以包含双引号

双引号定义法：可以包含单引号

可以使用转移字符\来将引号解除效果，变成普通字符串

如：“\”我是熊章智”\”

（2）字符串的拼接

如果多个字符串进行拼接，可以使用加号

如print(“我是”+“熊章智 ”)

当然同时是字面量或变量和变量之间可以使用拼接

如 name = 熊章智

print(“我的名字是:” + name + “, 我即将是一名大学学生”)

但是要注意：字符串无法和非字符串进行拼接

（3）字符串格式化

掌握通过占位的方式拼接字符串

如果变量过多，很麻烦；而且字符串无法和非字符串拼接

解决方法：试用以下语法

占位性拼接：%s

其中：%的意义：我要占位；s的意义：将变量变成**字符串**放入占位的地方

注：无论这里的s是什么内容，这里会将数字转化为字符串 ！！

又如：

class\_num = 57

avg\_salary = 16781

message = “Python大数据学科，北京%s期，毕业平均工资：%s” % (class\_num, avg\_salary)

括号内的内容会按照顺序填入‘

这样一来就完成了字符串和数字（整数、浮点型）的拼接

这里的数字都被转化为了字符串，那么有没有什么本办法让数字以原本的面貌进入呢？

%s 将内容转化为字符串，放入占位的位置

%d 将内容转化为整数

%f 将内容转化为浮点型

（4）格式化的精准控制

若： num = 19.99

message = “今天的股价是：%f ” % num

print(message)

则会输出 今天的股价是：19.990000  
我们是用辅助符号m与.n进行数据的宽度和精度控制(注意偶点！) 格式为（m.n）

m, 控制宽度，要求是数字（很少使用）；若设置宽度小于数字本身，不生效

.n , 控制小数点精度，要求是数字，会对小数四舍五入

如%5.2f：表示将宽度控制为5，将小数点精度设置为2

如 num = 11.345

message = “%5.2f” % num

输出它结果是： [空格][空格]11.35

%.2f 表示不限制宽度，只是限制小数精度为2

(5) 字符串的格式化2

快速格式化： 语法为 f”内容{变量}“ 如

name = “熊章智”

birth\_year = 2005

print(f”我是{name},我出生于{birth\_year}”)

这种方法不会理会类型，而且不会做精度控制，在对精度没有要求的时候可以快速使用

（6）对表达式进行格式化

了解什么是表达式：一条具有明确执行结果的代码语句

如 print(“%d” % (1\*1))

print(f”1\*1结果是{1\*1}”)

print（“字符串在python中的类型是：%s” % type(‘字符串’)）

如第三个中 type(‘字符串’)已经是有结果的了

总结：表达式的格式化：

f”{表达式}”

“%s%d%f” % (表达式、表达式、表达式)

注意这里是顿号

股价计算小练习：

假设股票增长系数为1.2 ， 当前股价为19.99，求7天之后，股价是多少

name = "传智播客"  
stock\_price = 19.99  
stock\_code = "003032"  
growth = 1.2  
days = 7  
price = stock\_price \* growth\*\*days  
print(f"""  
公司，{name},股票代码：{stock\_code},当前股价：{stock\_price}  
经过七天的增长后，股价将会达到%.2f  
""" % price)

成功！

11. 数据输入

语句 input() 如

print(“告诉我你是谁？”)

name = input()

print(“哦原来你是%s” % name)

当然，第一个print事实上可以优化

name = input(‘请告诉我你是谁’)

这两个版本功能是等价的

注意！！无论键盘输入的是什么信息，最后获得的数据永远是字符串

若要转化，则要int float str 语法

12. 布尔类型和比较运算符

布尔类型 真与假 比较运算符用于计算 真和假

进行判断只有是和否 则在程序中通过布尔类型来描述

如：

result = 10 > 5

print（“10真的比5大吗？{result}，类型是{type(result)}”）

而类型便是bool <class ‘bool’>

是则是True 否则是 False

比较运算符 == != > < >= <=

一旦开始使用比较运算符的时候，结果便是布尔类型的True 和 False

除了可以定义布尔类型外，还可以通过比较运算符极端得到布尔类型

13. if 语句基本格式

语法 if 要判断的条件:

条件成立时要做的事情

注意！首先是冒号！然后第二行中前面有四个空格！这是十分关键的！程序会根据首行缩进来看执行语句的归属

如： age = int(input(请输入你的年龄))

print(f“我今年已经{age}岁了”)

if a = 18:

print(“我我已经成年了啊”)

print(“时间过得真快”)

注意，input输入的永远会变成字符串

最后这句话无论如何会输出，但是if中的那句话只有True 的时候才会输出

这就是首行缩进的重要性

14. if else 语句

如果不满足呢？可以用if else 实现

if ……:

执行语句

else:

执行语句 十分简单

注意else 后面的仍然要首行缩进四个空格

15. if elif else 语句

if 条件1：

执行1

elif 条件2：

执行2

elif 条件3：

执行3：

。。。。。。

else:

执行N

仍然十分简单；首行缩进不能忘，冒号不能忘

同时注意判断是互斥而且有序的，上一个满足了后面就不会再判断了

num = 5  
if int(input("我心里下想了一个数字，你来猜一下吧:")) == num :  
 print("你猜对啦！")  
elif int(input("不对，再猜一次！只有两次机会了:")) ==num :  
 print("恭喜你还是猜对啦！")  
elif int(input("不对！你只有最后一次猜的机会了:")) == num :  
 print("恭喜你最终还是猜对了！")  
else :  
 print("真可惜！你没有猜出来！")  
print("Nice Gaming!")

16. 嵌套使用

注意首行缩进的递进即可

嵌套判断语句适合多条件、多层次的逻辑判断

嵌套判断语句可以根据需求，自由组合if elif else 来构建多次层次判断

嵌套判断语句，一定要注意空格的缩进！程序通过空格决定层次次序！

实战练习

定义一个数字 范围为1-10

有三次机会猜数字，每次猜不中会提示大了还是小了

import random

num = random.randint(1,10) 可以随机生成1-10的数字

import random  
num = random.randint(1,10)  
print("""我们来玩一个小游戏！  
随机生成了一个1-10的数字，你有三次猜的机会  
每次猜完后，我会告诉你大或者小  
祝你好运！""")  
num\_1 = int(input("请输入你猜测的数字："))  
if num\_1 == num:  
 print ("你猜对了！")  
else:  
 if num\_1 > num:  
 print ("你猜的数字太大了！")  
 else:  
 print ("你猜的数字太小了！")  
 num\_2 = int(input("再猜一次："))  
 if num\_2 == num:  
 print("你猜对了！")  
 else:  
 if num\_2 > num:  
 print("你猜的数字太大了")  
 else:  
 print("你猜的数字太小了")  
 num\_3 = int(input("你只有最后一次猜的机会了！请输入你猜的数字："))  
 if num\_3 == num:  
 print("恭喜你最终还是猜对了！")  
 else:  
 print(f"可惜你失败了，正确数字是:%d" % num)  
print("Nice Gaming!")

17. while循环语句

循环和判断都是核心

（1）基础语法：

while 条件:

条件满足时，做的事情1

条件满足时，做的事情2

条件满足时，做的事情3

只要条件满足会无限执行

这个条件必须是布尔类型或比较运算：结果必须是布尔 有True False

如 i = 0

while i < 100

print(“我是sb”)

i += 1

总结：while条件需要得到布尔类型，True表示继续循环，False 表示结束循环

需要设置循环终止的条件，如i +=1 和i < 100 配合

而且空格缩进和if一样，都需要设置

例子：求1-100的和

num = 1  
sum = 0  
while num <= 100:  
 sum = sum + num  
 num += 1  
print(f"1-100的和为：%d" % sum)

又例：猜数字案例

设置一个1-100随机整数变量，通过while判断输入的数字是否等于随机数

无限次机会，知道猜中为止；每一次猜不中，会提示大了或者小了

并且提示出猜出数字之后，你猜了几次

import random  
num = random.randint(1,100)  
print("""我们来玩猜数字游戏  
假定有一个1-100的数字，每次猜错会提示大还是小  
好消息！你有无限次猜的机会！""")  
guess = int(input("请输入你猜的数字:"))  
i = 0  
while guess != num:  
 if guess > num:  
 print("你猜大了！")  
 i += 1  
 else:  
 print("你猜小了！")  
 i += 1  
 guess = int(input(f"请再猜一次！你已猜:{i}次；请再输入你猜的数字:"))  
i += 1   
print(f"恭喜你猜对了！你一共猜了{i}次")

（2）while 嵌套应用

while 条件1：

做的事情1

做的事情2……

while 条件2：

做的事情1

做的事情2……

需要注意的地方：避免无限循环；注意条件控制

要通过空格缩进确定层次关系

循环案例：九九乘法表

前提：print(“Hello”, end=’’)

print(“World”, end=’’)

则会输出 HelloWorld 达到了不换行的目的

\t 可以使多行字符串对齐

练习打印九九乘法表

i = 1  
while i < 10:  
 j = 1  
 while j <= i :  
 print (f"{j}\*{i}\t=\t{i\*j}\t",end='')  
 j += 1  
 i += 1  
 print() *# 这里面内容是空的，就是换行的意思*

（3）for循环

for是一种“轮询”机制，是对一批内容进行处理

而while循环的条件是自定义的，自行控制循环条件

for 临时变量 in 待处理数据集

循环满足条件时执行的代码

例子：

name = itheima

for x in name ：

print (x)

则会输出：

i

t

h

。。。。。。 字符串中的字母是一个一个取出的

所以for循环称为遍历循环

理论上 for无法产生无限循环（数据集不可能无穷大）

注意循环内的语句是要有空格缩进的

如例子： 数出来 itheima is a brand of itcast z中有多少个a

name = "itheima is a brand of cast"  
i = 0  
for x in name:  
 if str(x)== "a":  
 i += 1  
print(f"这句话里面一共有{i}个a")

其实str不用打，x本身是字符串

（4）range语句

在for中，待处理数据集事实上为序列类型

我们目前只学习了字符串的类型

而range可以获得一个简单的数字序列

如range(5) 可以获得的数据为 {0,1,2,3,4}

从零开始，但是不到num

range(2,5) 获得 {2,3,4} 不含num2

还有 range(num1,num2,step)

range(5,10,2)

获得 {5,7,9}(注意不会拥有10！不含num2!!)  
如i = 0  
for x in range(1,101):  
 if x%2 == 0:  
 i += 1  
print(i)

（5）变量作用域

思考 for i in range(5)

print (i)

print(i)

那么最后这个print可以访问到变量i吗？

规范上是不允许的但是运行起来是可以访问到的

规范：作用范围（作用于）只能限定在for循环内部

如果for循环外部访问临时变量：

实际上可以访问到，但是在编程规范上是不允许的

如果需要访问临时变量，可以预先在外定义它

（6）for 嵌套应用

例子：再次打印九九乘法表

for i in range(1,10):  
 j = 1  
 while j <= i:  
 print(f"{j}\*{i}\t=\t{i\*j}\t",end='')  
 j += 1  
 print()

（7）break 和 continue

两者分别可以直接结束和临时跳过

两者都只能对所在的循环造成影响

综合案例：

某公司账户余额有一万元，给二十人发工资

员工编号1-20，从1开始，依次领取工资，每人1000

领工资时，财务判断员工绩效分1-10（随机生成）若低于五则不发工资换下一位

如果工资发完了，不发了

输出每位员工的状况：因为分不够；发到了，账户剩多少？；没钱了，你没发到

total = 10000  
import random  
for x in range(1,21):  
 num = random.randint(1,10)  
 if num < 5 :  
 print(f"员工{x}绩效仅为{num},\t不合格，\t不能领取工资")  
 else :  
 if total <= 0 :  
 print("剩下员工没钱拿了！以后再说吧！")  
 break  
 else:  
 total = total - 1000  
 print(f"员工{x}绩效为{num},\t合格！\t领取1000元，账户仅剩{total}元了")

或者

money = 10000  
for i in range(1,21):  
 import random  
 score = random.randint(1,10)  
  
 if score > 5:  
 print(f"员工{i}绩效为{score}，不满足，不发工资，下一位")  
 continue  
 if money >= 1000:  
 money -= 1000  
 print(f"员工{i}，满足条件发放工资1000，公司账户余额{money}")  
 else:  
 print(f"余额不足，当前余额为{money}，不足以发工资了，不发了，下个月再来")  
 break

18. 函数

（1）函数介绍

函数式组织好的、可以重复使用的、实现特定功能的代码段

如name = “itheima”

length = len(name)

print(length)

len()是内置的函数，是提前写好的

（2）函数的定义

定义 def 函数名(传入参数)：（无论这里数不输入传入参数，括号是一定要打的）

函数体

return 返回值 (这行在不输入传入参数的时候是可以省略的)

注意这里是有空格缩进的

def say\_hi():

print(“你好！”) # 到这里仅仅是完成了函数的定义

say\_hi() # 注意这里是要有括号的

则会输出 你好！

总结：注意事项：

1. 参数若果不需要，可以省略

2. 返回值如果不需要，可以省略

3. 函数必须先定义再使用

附： \n可以用于换行

（3）函数的参数

传入参数的功能是：在函数进行计算的时候，可以接受外部（调用时）提供的数据

def add(x,y)

result = x + y

print(f”{x}+{y}={result}”)

定义完了之后，每次在使用的时候就可以对xy进行指定

在函数定义中，提供的x和y，称之为：形式参数

表明函数声明将要使用两个参数

参数之间用逗号隔开来

而在使用函数的时候输入的称之为实际参数，是函数真正执行时所使用的两个参数

传入参数的数量是不受限制的

（4）函数的返回值

返回值就是程序中函数完成事情之后最终给调用者的结果

def add(x,y)

result = a+b

return result

r = add(1,2)

print(r)

则会输出3

其中这个r作为变量接受了函数的返回值

注！这个函数体在定义的时候，走到return之后，就不会定义任何你要他执行的东西了！

有一个问题：如果函数没有使用return语句返回数据，那么函数会议后返回值吗

事实上 有

有一个特殊的字面量 None 他的类型是 <class ‘NoneType’>

None 表示空的、无实际意义的意思

def sayhi():

print(“你好！”)

result = sayhi()

print(type(result))

则会输出 <class ‘NoneType’>

def check\_age (age):

if age > 18:

return “SUCCESS”

else:

return None

result = check\_age(16)

if result == None: # 完全等价于 if not result

print(“未成年！”)

附：为什么这里是等价的？None被认为是False not result 则为 not False 则为 True

意思便是如果真（None）则进入该if语句

（5）函数说明文档

通过注释对函数进行解释说明

注意三个引号及解释最好遵守空格缩进（四个）

函数说明的规范：

:param x : 参数x的说明

:return: 返回值的说明

而且这些内容写在def后面，函数体之前！！！！

（6）函数的嵌套使用

如果函数A中调用了另外一个函数B，那么先把函数B中的人物都执行完毕之后才会回到上次函数A执行的位置

事实上还是十分简单而容易理解的

（7）变量的作用域

变量作用域指的是变量作用的范围，即变量在哪里可以使用，在哪里不可以用

局部变量：定义在函数体内部的变量，只能在函数体内部生效

def test():

num = 100

print (num)

test() # 这个可以有输出

print(num)

# 就会发现最后一行这个东西后报错，指示这个num没有定义

全局变量：在函数体外、在函数体内都能够生效的变量

如果有一个数据，在函数A和函数B中都要使用，应该怎么办？

可以讲这个数据储存在一个全局变量之中

num = 100

def test1():

print(num)

def test2():

print(num)

test1()

test2()

则会正常输出两个100

特殊：global 关键词

使用这个关键词可以在函数内部声明全部变量为全局变量

def test():

global num

num = 100

print (num)

print(num)

发现这个时候num就可以显示出来是100了

综合案例：做一个ATM

模拟功能：第一个 主菜单功能 效果：

-----------主菜单-----------

周杰伦，您好，欢迎来到ATM，请选择操作：

查询余额 【输入1】

存款 【输入2】

取款 【输入3】

退出 【输入4】

请输入您的选择

第二个： 查询余额效果

----------查询余额---------

周杰伦，您好，您的余额为：5000000元

存、取款效果

-------------存款-------------

周杰伦，您好，您存款50000元成功

您的余额有：5050000元

money = 5000000  
def menu():  
 print("""----------主菜单----------  
周杰伦，您好，欢迎来到ATM，请选择操作  
查询余额 \t【输入1】  
存款 \t【输入2】  
取款 \t【输入3】  
退出 \t【输入4】""")  
menu()  
choice = int (input("请输入您的选择："))  
def checking(money):  
 print(f"""----------查询余额----------  
 周杰伦，您好，您的余额为：{money}元""")  
def deposit(x,money):  
 money += x  
 print(f"""----------存款----------  
 周杰伦您好，您存款{x}元成功！  
 您的账户余额为：{money}""")  
def withdraw(y,money):  
 money -= y  
 print(f"""----------取款----------  
 周杰伦您好，您取款{quchu}元成功！请保管好您的取款！  
 您的账号余额为：{money}元""")  
while choice != 4:  
 if choice == 1:  
 checking(money)  
 menu()  
 choice = int(input("请输入您的选择："))  
 if choice == 2:  
 cunru = int(input("请输入您存款的数额："))  
 deposit(cunru,money)  
 menu()  
 money += cunru  
 choice = int(input("请输入您的选择："))  
 if choice == 3:  
 quchu = int(input("请输入您取款的数额：！"))  
 withdraw(quchu,money)  
 menu()  
 money -= quchu  
 choice = int(input("请输入您的选择："))  
 while choice == 4:  
 break  
print("欢迎您下次光临！")

19. python 数据容器

（1）入门

了解什么是数据容器：一个可以容纳多分数据的容器

一种可以容纳多分数据的数据类型，容纳的每一分数据称为一个元素

每一个元素是可以为任意类型的数据 如 字符串 数字 布尔 等

数据容器根据不同的特点，分为五类：

列表list 元组 tuple 字符串 str 集合 set 字典 dict

而这些特点有：是否支持重复元素 是否可以修改 是否有序

（2）列表

列表的定义：掌握基本格式

【元素1，元素2……】 字面量

变量名称 = 【元素1，元素2……】 定义变量

变量名称 = 【】

变量名称 = list() 这两种方式可以获得空列表

当然：列表也是可以嵌套的

元素是数据容器中的每一分数据，没有任何限制。甚至可以是列表

（3）列表的下标索引

如何从列表中去除特定位置的数据呢？

我们可以使用下标索引

name\_list = [‘Tom’,’Alice’,’Rose’]

print(name\_list[1]) 结果会输出Alice

从第一个开始，元素的下表为 0 1 2 3

用以上的语法就可以抽出特定位置的元素了

当然从前向后0 1 2

也可以从后往前 ……-3 -2 -1 注意！这里只到-1！

最后，我们还可以取出嵌套列表中的元素

list = [[1,2,3],[4,5,6]]

print(list[0][2])

则会输出3

（4） 列表的常用操作

列表除了定义 使用下标索引获取值

也有很多强大的功能

插入元素 删除元素 请空列表 修改元素 统计元素个数

在python中将函数定义为class中的一员，则它的名字称为方法

class Student:

def add(self,x,y)

return x+y

这个函数式定义在class里面的，那么这个就称为方法

方法和函数功能是一样的，有传入参数，有返回值，只是使用格式不一样

函数的使用 num = add(1,2)

student = Student()

num = student.add(1,2)

list 提供了一系列方法

方法1：查询功能

功能：查找指定元素在列表的下标，如果找不到，报错 ValueError

语法：list.index(元素)

方法2：修改功能

list[下表] = 值

方法3：插入元素

list.insert(下表，元素)

在指定的下表位置插入指定的元素

但是注意：在x插，指的是在x的前面

方法4：追加元素

list.append(元素)

将指定元素追加到列表的尾部

这种方法只能追加一个元素，那么如何追加多个元素呢？

list.extend(其他数据容器)

将其他数据容器的内容取出，依次追加到列表尾部

方法5：元素的删除

del list[下标] 这个就是仅仅能删除

list.pop[下标] 这个不仅能够删除，而且还可以将删除的内容作为一个返回值

可以给自变量附上这个删除的元素的字面量

方法6：删除某元素在列表中的第一个匹配项  
list.remove(元素) 注意是从前往后搜索到的第一个元素

方法 7：清空这个列表

list.clear() 这个括号虽然要写，但是里面也没有值

方法 8：统计某种元素在列表中的数量

list.count(元素)

方法 9：统计列表中总共有多少个元素

len(列表)

以后学完基础之后，会学框架技术

附：一个列表中元素数量最大为 2\*\*63-1 个

列表允许重复元素存在，数据是有序储存的，而且可以修改

age\_list = [21,25,21,23,22,20]  
*# 记录一批学生的年龄*age\_list.append(31)  
*# 在末尾加31*age\_list.extend([29,33,30])  
*# 在末尾加一个新列表*del age\_list[0]  
*# 删除第一个元素*del age\_list[-1]  
*# 删除最后一个元素*print(age\_list.index(31))  
*# 输出元素31的下标*

（5）list的遍历

掌握while和for循环列表

数据容器中有多个元素，则有需求从容期中一次取出元素进行操作

index = 0

while index < len(list)：

元素 = 列表【index】

对元素进行处理

index += 1

for 临时变量 in 数据容器

对临时变量进行处理

在循环控制上：

while是可以自己定义循环条件的，并自行控制

但是for循环不可以自定义循环，只能一个一个取出

在无限循环上

while可以通过条件控制做到无限循环

而for是理论上不可能无限循环的，因为数据容器的元素数量是不可能无穷大的

使用场景上：

while循环适合用于任何想要循环的场景

for循环适用于遍历数据容器中的场景或者简单的固定次数循环场景

list = [1,2,3,4,5,6,7,8,9,10]  
new\_list = []  
index = 0  
while index <len(list):  
 if index%2 != 0:  
 new\_list.append(list[index])  
 index += 1  
 else:  
 index += 1  
 continue  
  
list.extend(new\_list)  
print(list)

20. 元组

为什么需要元组？

列表是可以修改的；如果想要传递信息，不能被篡改，则列表则不合适

元组一旦定义完成，就不可以修改

我们需要封装数据，但是有不希望被封装的被篡改，则使用元祖

语法 使用小括号

定义空元祖：

变量名 = ()

变量名 =tuple()

注意！！如果元祖中只有一个元素，这个元素之后必须打上括号！否则而不是元组类型

元组数据是可以为不同类型的数据，而且可以嵌套

下标索引：语法完全一样

只有三种操作：

index()： 查找某个数据

count():统计某个元素在当前元祖中出现的次数

len():统计元祖中元素的个数

不可以尝试修改元组！否则会报错！

又但是！元祖中如果含有一个list，这个list是可以被修改的！！

21. 字符串（数据容器视角）

字符串同样也是容器中的一员，一个字符串可以存放任意容量的字符

和元组一样，字符串也是不可以修改的

字符串的替换

字符串.replace(字符串1，字符串2)

功能：将字符串内的全部内容还掉，但是注意原来这个字符串并没有消失

而是这是有返回值的！

name = “itheima,itcast”

new\_name = name.replace(“it”,”程序”)

那么输出的话，这个name仍然没有变化，但是new\_name输出为

程序heima,程序cast

字符串的分割

字符串.split(分割字符串)

功能：按照指定内的分割字符串，将字符串划分为过个字符串，并存入列表对象中

注意：字符串本身并没有变化，而是得到了一个列表对象

my\_str = “Hello itheima itcast world”

list = my\_str.split(“ ”)

那么这个list为

【Hello,itheima,itcast,world】

注意，这里是按照空格为划分依据

字符串的规整操作

字符串.strip()

功能：去掉前后空格

字符串.strip(字符串)

去掉前后指定的字符串

字符串特点：

只可以储存字符串 长度任意 支持下表索引 允许重复字符串的存在 不可以修改 支持循环

字符串可以是视为字符的容器，支持下表索引

22. 数据容器（序列）的切片

什么是序列?序列是指：内容连续、有序，可使用下标索引的一类数据容器

列表、元组、字符串，均可以视为序列

常用操作：切片

语法：序列【起始下标：结束下表：步长】

起始下标表示从何处开始，从指定位置开始，可以留空，留空意味着从头开始

结束下标表示何处结束，可以留空，留空则意味着截取到结尾

注意：这个结束下标本身是不要的

步长：取元素的间隔

1：一个个取元素

2：每次跳过一个元素取

步长为负数：反向取元素 注意：如果反向取元素，起始下标结束下标都是要反向来的！

注意：切片操作是不会修改序列的！

序列的切片实践

假设有一个字符串：“万过薪月，员序程马黑来，nothyP学”

请使用任何学过的方法，得到黑马程序员这些内容

第一种方法：

倒叙字符串，切片取出，然后倒序

str = “万过薪月，员序程马黑来，nothyP学”

result = str[::-1][9:14]

print(result)

第二种方法：split分割，replace替换“来”为空，倒序字符串

str = "万过薪月,员序程马黑来,nothyP学"  
result = str.split(",")[1].replace("来"," ")[::-1]  
print(result)

23. 数据容器 集合

目前来说：列表可以修改，支持重复元素且有序

元组、字符串不可以修改，支持重复元素且有序

但是不难看出，这样子的话是有局限性的

局限就在于：它们都支持重复元素

如果场景要求对内容进行去重处理，列表、元组、字符串就不是很实用了

而集合最大的特点就是：不支持重复元素且无序

定义方式基本一致{} 注意使用的是大括号

定义空集合

变量名称 = set() 注意：集合不支持下标索引

操作： 添加新元素

集合.add(元素) 将指定的元素添加到集合中

移除元素：

集合.remove(元素)

随即取出一个元素

集合.pop() 取出之后的元素是可以用于变量的，并且该内容也被集合中去除掉

清空集合 集合.clear()

取两个集合的差集

集合1.difference（集合2） 取出集合1有但是集合2没有的

这个语法的结果是一个新结果

消除两个集合的差集

集合1.difference\_update（集合2）

功能：对比集合1和集合2，在集合1内，删除和集合2相同的元素

结果：集合1被修改，但是集合2是不变的

两个集合的合并

集合1.union（集合2）

功能：将集合1和几何2组合成新集合

结果：得到新集合，而集合1和集合2是不变的

统计集合元素数量

len(集合)

集合的遍历：

集合是没有下标索引的，因此不能使用while循环

可以使用for循环

总结：集合有以下特点

可以容纳多个数据，可以容纳不同类型的数据，数据是无序存在的而且不支持下标索引，不允许重复元素的出现，可以修改，只能支持for循环

list = [1,2,1,2,1,2,1,2,3]  
set = set()  
for x in list:  
 set.add(x)  
for x in set:  
 print(f"{x}")

注意定义空集合的方式（第二行）

24. 字典

（1）基础

字典通过key去找到对应的value

字典也是使用大括号，但是储存的元素为一个一个键值对

{key:value,key,value。。。。。}

dict = dict() 空字典的定义

定义重复key的字典：虽然可以写，但是会存在覆盖现象

字典和集合一样是不支持下标索引的

但是可以用key找到value

定义嵌套字典：Key和value都可以是任意类型的数据，但是只有一个硬性要求

key不可以是字典

如：student\_score =

{

“学生甲”：{

“语文”：123，“数学”：123，“英语”：123

}，“学生乙“：

{

“语文”：133，“数学”：143，“英语”：111

}，”学生丙”：

{

“语文”：125，“数学”：134，“英语”：134

}

}

这样长得像c++的定义方式是可以的！但是字典的格式一定要注意去遵守！

从嵌套字典中获得数据：

score = student\_score [“学生甲”][“英语”]

（2）字典的常用操作

新增元素：

字典【key】 = value

字典被修改，新增了元素；注意这里的key应该是没有出现过的

更新元素

字典【key】 = value 语法是一模一样的，但是这里的key应该是字典里面有的

更重要的是：这两个语法中都是没有.的！

删除元素

字典.pop（Key） 结果：获得key对应的value，同时字典被修改

清空元素

字典.clear()

获得全部的key:

字典.keys() 结果：得到字典中全部的key

遍历字典 第一种：通过获取到全部的key赖完成遍历

for key in keys:

第二种：直接对字典进行for循环，每一次循环都是直接得到key

for key in dict：

统计字典内元素的数量

len(dict)

25. 数据容器的通用功能

len(容器) 统计容器中元素的数量

max(容器) 统计容器中的最大元素

min(容器) 统计容器中的最大元素

list(容器) 将给定容器转化为列表

str(容器) 将给定容器转化为字符串

tuple(容器) 将给定容器转化为元组

set(容器) 将给定容器转化为集合

容器通用排序功能：

sorted(容器，[reserve]) 正向排序

sorted(容器·，【reversed = True】) 反向排序

26. 字符串大小的比较

在程序中。字符串中的所有字符都是有对应的ASCII码表值

字符串是按位比较的，也就是一位位进行比较，只要有一位大，那么整体就大

27. 函数进阶

（1）函数多返回值

def return\_num():

return 1

return 2

result = return\_num()

print(result)

我们定义了一个返回单个数字的函数，没有要输入的参数

在函数中有两个return ， 事实上执行一次return就可以退出当前函数了

所以事实上第二个return根本不会执行。result输出的结果为1

def return\_num():

return 1,2

x,y = return\_num()

print(x) 按照返回值的顺序，写对应顺序的多个变量去接收就可以了

print(y) 变量之间用逗号隔开来，支持不同类型的数据return

（2）函数的多重传参方式

函数在使用方式的不同将参数分为了四种

位置参数、关键字参数、缺省参数、不定长参数

位置参数：调用函数时根据函数定义的位置参数来传递参数

注意：传递的参数和定义的参数的熟悉怒及个数必须一致

关键字参数：函数调用时通过键=值形式传递参数

作用：可以让函数更加清晰，同时也清楚了参数的顺序要求

def user\_info(name,age,gender)

print(f”您的名字是:{name}.年龄是:{age},性别是{gender}”)

user\_info(gender=”男”，name=”小梦”，age = 20)

可以看出来，关键词传参是不需要顺序的

而且可以和位置参数混用，位置参数必须在前面，且匹配参数顺序

函数调用时，如果有位置参数，位置参数必须在关键字参数的前面，但是关键字参数之间是不存在先后顺序的

缺省参数：也叫做默认参数，用于定义函数，为参数提供默认值，调用函数时可不传该默认函数的值（注意：所有位置参数必须出现在默认参数之前，包括函数定义和调用）

def user\_info(name,age,gender=“男”)

print(f”您的名字是:{name}.年龄是:{age},性别是{gender}”)

user\_info(“Tom”，20)

user\_info(“Rose”,20,”女”)

在函数调用的时候，如果为缺省参数传值则会修改掉默认参数值，否则使用这个参数值

不定长参数：也叫作可变参数，用于不确定调用的时候会传递多少个参数（不传也可以）的场景 作用:当调用函数时不确定参数的个数时，可以使用不定长参数

类型：位置传递、关键词传递

def user\_info(\*args):

print(args)

user\_info(“tom”,18)

注意！传进的所有参数都会被args变量收集，它会根据传入参数的位置合并为一个元组，args为元组类型，这就是位置传递

def user\_info(\*\*kwargs):

print(kwargs)

user\_info(name = “tom”, age = 18 , gender = “男”)

参数为“键=值”形式的情况下，所有都会被kwargs接受，同时组成字典

（3）匿名函数

在先前的函数学习中，我们一直使用的函数都是接受数据作为参数输入的

数字、字符串、字典、元组、列表等

但是我们学习的函数本身也可以作为参数传入另一个参数之中

def test\_func(cumpute)

result = compute(1,2)

print(result)

def compute(x,y)

return x+y

test\_func(compute)

函数compute作为参数传入了test\_func函数中进行使用

test\_func需要一个函数作为参数传入，这个函数需要接受两个数字进行计算，计算逻辑由被传入的函数决定

compute函数接受两个数字对其进行计算，最终在函数test\_func内部由传入的compute函数，完成了对数字的计算操作

我传入的根本上是代码的执行逻辑

综上所述：函数本身是可以作为参数的，将函数传入的作用在于传入计算逻辑，而并不是传入数据本身。所以传入函数和传入数据其实还是有区别的！

特殊的：lambda匿名函数

def关键词可以定义带有名称的函数，但是lambda关键词可以定义匿名函数（没有名称）

有名称的函数是可以基于名称重复使用的，但是无名称的匿名函数只能临时使用一次

lambda 传入参数 : 函数体（一行代码）

传入参数表示匿名函数的形式参数 如x,y表示接受两个形式参数

函数体就是函数的执行逻辑，注意：只能写一行，无法写多行代码

如：def test\_func(cumpute)

result = compute(1,2)

print(result)

test\_func(lambda x,y： x + y)

输出的结果为3 这样代码就显得更优雅

使用def 和使用lambda定义函数的功能是完全一致的

只不过是lambda关键字定义的函数是匿名的，无法二次使用

28. 文件

（1）文件的编码

计算机只能识别0和1，那么我们丰富的文本文件是如何被计算机识别的并储存在硬盘中？

答案便是：使用编码技术（密码本）将内容翻译成0和1存入

编码技术便是：翻译的规则，记录了如何将内容翻译为二进制，以及如何将二进制翻译回可识别内容 计算机中有很多编码规则： UTF-8 GBK Big5 等

不同的编码，将内容翻译成二进制也是不同的

UBT-8目前是全国通用的编码格式，除非有特殊要求，否则一律以这个格式进行文件编码

（2）文件的读取

什么是文件？内存中储存的数据在计算机关闭后就会消失。如果要永久保存数据，就要使用硬盘、光盘、U盘等设备。为了便于数据的管理和检索，引入了“文件”的概念

任何都可以被保存为文件，并赋予一个文件名，操作系统以文件为管理单位磁盘中的数据

一般来说，文件可分为文本文件、视频文件、音频文件、图像文件、可执行文件等识别类型

日常中文件操作主要包括打开、关闭、读、写等

在python中，使用open函数可以打开一个已经存在的函数，或者创造一个新文件，语法：

open(name,mode,encoding)

name：是要打开的目标文件名的字符串（可以包含文件所在的具体路径）

mode：设置打开文件的模式（访问模式）：只读、写入、追加

encoding：编码格式 UTF-8

示例代码：

f = open（“python.txt”,“r”,encoding = “UTF-8”）

注意：此时的f是open函数的文件对象，对象是python中一种特殊的数据类型，拥有属性和方法，可以使用对象，属性或对象、方法对其进行访问

r 以只读方式打开文件

w 打开一个文件只用于写入。如果该文件已经存在则会打开文件从头开始编辑并且删除原来的所有内容

a 打开一个文件用于追加。如果该文件已经存在，新的内容将会写到已有内容之后

如果改文件不存在，创造新文件进行写入

读取操作： 文件对象.read(num)

num表示要从文件中读取的数据的长度（单位是字节），如果没有写num，那么就表示读取文件中的所有内容

而 readlines()方法：

readlines可以按照行的形式把整个文件中的内容进行一次性读取，并且返回的是一个列表，其中每一行的数据为一个元素

注意！！下一个read会在上一个read的地方继续开始读取！

f = open("D:/test.txt","r",encoding = "UTF-8")  
lines = f.readlines()  
print(f"lines对象的内容是：{lines}")

lines对象的内容是：['这是一个测试文档\n', '我的名字是熊章智\n', '今年已经十八岁了\n', '我现在在学的编程语言是Python'] （这是输出的内容）

而当我们使用的是readline时，一次只会读取一行

f = open("D:/test.txt","r",encoding = "UTF-8")  
i = 1  
for line in f:  
 print(f"第{i}行的是：{line}")  
 i += 1

文件的关闭：

f.close() 如果不关闭的话，python 会一直占用这个文件

with open 用法：

with open(“python.txt”,”r”)as f：

f.readline s（）

…………

这种方法可以通过with open的语句块中对文件进行操作

可以在操作完成后自动关闭close文件，避免遗忘掉close

（3）文件的写入

快速入门：先打开再写入最后刷新

f = open(“ ”,”w”,encoding = “UTF-8”)

f.write(“Hello world”)

f.flush

f.close (close()方法，带有flush()方法的功能)

直接调用write，内容并未真正写入文件，而是会积攒在程序的内容中，称之为缓冲区

当调用flush的时候。内容会真正写入文件

这样做是避免频繁的操作硬盘，导致效率下降

如果这个文件本身不存在，则这个操作会创造一个文件；如果这个文件本身存在，则w模式会清空原来的所有内容

（4）文件的追加

f = open(“python.txt”,”a”,encoding = ”UTF-8”)

f.write(“Hello World”)

f.flush()

注意： a模式，文件不存在会创建文档；文件存在会在最后追加写入文件

可以使用\n来换行

29. 异常、模块与包

（1）了解异常

当检测到一个错误的时候，python解释器就无法继续执行了，反而出现了一些错误的提示

异常就是程序运行中出现了错误

（2）异常的捕获方法

世界上没有完美的程序，我们要在力所能及的范围内对可能出现的bug进行提前准备和处理，这种行为便是异常处理（捕获异常）

捕获异常的作用就在于：提前假设某处可能出现异常，做还提前准备，当真的出现异常的时候，可以有后续手段

try:

可能发生错误的代码

except:

如果出现异常执行的代码

如：尝试用r模式打开文件，但是如果文件不存在的话，则以w模式打开

捕获指定异常：

try:

print(name)

except NameError as e:

print(“name变量名称为定义错误”)

这句话便代表：如果name是未定义的话，则会单方面爆出这个错误

注意：如果尝试执行的代码的异常类型和要不活的异常类型不一致，则无法捕获

一般try下方只能放一行执行的代码

当捕获多个异常的时候，可以把要捕获的异常类型的名字放到except后，并使用元组进行书写

try:

1/0

except (Nameerror,ZeroDivisionError) as e:

print(“出现……错误”)

捕获全部异常：

try:

……

except Exception as e:

print(“产生错误了”)

但是如果真的没有出现问题的话呢？

try:

……

except Exception as e:

print(“产生错误了，异常的信息是：{e}”)

else:

print(“没有问题”)

当然这个else不是硬性要求

而finally呢？它代表无论是否异常都要执行的代码

try:

……

except Exception as e:

print(“产生错误了”)

else:

print(“没有问题”)

finally:

f.close()

同样，finally也是可写可不写的·

（3）异常的传递

异常是有传递性的。当所有函数都没有捕获异常的时候，程序就会报错

基于函数的调用方式，异常会一层一层向上传递

如果想要捕获异常，只要在函数的最顶级调用中进行try except

（4）模块

模块（Module）是一个Python文件，以.py结尾

模块能定义函数，类和变量，模块里面也可能跟包含可执行的代码

作用：python中有很多各种不同的模块，每一个模块都可以帮助我们快速实现一些功能，比如实现和时间相关的功能就可以使用time模块

我们可以认为模块就是一个工具包，每一个工具包中都有各种不同的工具供我们使用，进而实现各种不同的功能

语法：

【from 模块名】 import [模块|类|变量|函数|\*] 【as 别名】

常见组合：

import 模块名

from 模块名 import 类、变量、方法等

from 模块名 import \*

import 模块名 as 别名

from 模块名 import 功能名 as 别名

注：中括号内的内容是可以不必写的。但是import是一定要写的

第一种：

import time 代表你可以使用time模块中的所有内容了

time.sleep(5) 后面这个sleep就是功能名，这个代码会使程序运行到这里的时候睡眠五秒

第二种：

from 模块名 import 功能名

from time import sleep 表示单独从time模块中调用sleep函数

sleep(5) 程序运行到这里睡眠5秒

第三种：

from time import \* \*代表全部的意思

都是调用time模块中的全部内容，第三种和第一种有什么区别呢？

只是后面如果想要使用sleep函数的话，直接写sleep()就可以了

第四种:

import time as tt 现在time模块改名为tt

tt.sleep(5)

第五种：

from time import sleep as shuimian 现在sleep函数名字变成了shuimian

shuimian(5)

模块的导入一般写在代码的开头位置

（5）自定义模块

python中有很多模块了，但是有时候想要些个性化的模块，那就需要自定义模块了

现在另一个程序中写module.py程序

然后你的主编码中就可以使用module模块了

注意事项：当导入多个模块的时候，且模块内有同名功能，则当调用这个同名功能的时候，调用的是后面导入的模块的功能，也就是说会存在覆盖现象

那么假设我在写一个模组的时候：  
def test(a,b):

print(a+b)

test(1,2)

最后一行的存在是因为我可能想试一下这个模组是否正确；但是这样写的话一旦import，则3会被作为结果直接引入！那么应该怎么办呢？

def test(a,b):

print(a+b)

if \_\_name\_\_ == ‘\_\_main\_\_’:

test(1,2)

这样一来，你可以单独运行module来看模组是否正确

而且不会妨碍你在编码中的正常使用

from module\_test import test  
test(5,6)

又：如果模块文件中有’\_\_all\_\_’变量，当使用from XXX import \*的时候

只能导入这个列表中的元素

module.py

\_\_all\_\_ = [‘test\_A’]

def test\_A():

print(“testA”)

def test\_B():

print(“testB”)

program.py

from module import \*

这里代表：只能使用test\_A了！但是不是\*说是全部吗？

因为\_\_all\_\_变量就是代表了全部\*！

使用这个变量就可以完美控制哪些功能在\* 引入方式下能够被导入

（6）python 包

从物理上看，包就像是一个文件夹，在该文件夹下包含了一个\_\_init\_\_.py文件

该文件夹可用于包含多个模块文件

从逻辑上看，包的本质依然是模块

包的作用：当我们使用的模块文件越来越多时，包可以帮助我们管理这些模块，包的作用就是包含多个模块，但是包的本质依然是模块

快速入门 步骤如下：

1. 新建包’my\_package’

2. 新建包内模块

3. 导入：方式一： import 包名.模块名

方式二： from 包名 import 模块名

方式三：首先必须在’\_\_init.py\_\_’文件中添加’\_\_all\_\_ = []’，控制允许导入的模块列表

再使用 from 包名 import \*

这个all变量就是控制import\*能够导入的内容

（7）安装第三方包

在python程序生态中，有许多第三方包，可以极大帮助我们提高开发效率

科学计算：numpy

数据分析：pandas

大数据计算：pyspark apache-flink

图形可视化：matplotlib pyecharts

人工智能：tensorflow

这些是第三方包，python没有内置，所以需要安装才能导入使用

如何安装呢？ 使用pip程序

pip install -i <https://pypi.tuna.tsinghua.edu.cn/simple> 包名称

或者使用pycharm也可以

案例：

def print\_file\_info(file\_name):  
 f = None  
 try:  
 f = open(file\_name,"r",encoding = "UTF-8")  
 content = f.read()  
 print("文件的全部内容如下：")  
 print(content)  
 except Exception as e:  
 print(f"程序出现问题,原因是：{e}")  
 finally:  
 if f:  
 *# 如果f是none的话，close是不对的；所以要做判断  
 # f是none的话，在if判断中时不会进来的；不是none的话那就是True* f.close()  
def append\_to\_file(file\_name,data):  
 f = open(file\_name,"a",encoding = "UTF-8")  
 f.write(data)  
 f.write("\n")  
 f.close()  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print\_file\_info("")  
 append\_to\_file("", )

def str\_reverse(s):  
 return s[::-1]  
def substr(s,x,y):  
 return s[x:y]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(str\_reverse("黑马程序员") )  
 print(substr("黑马程序员",1,3))

30. 综合案例

折线图 动态柱状图 数学可视化

（1）json数据格式

json 是一种轻量级的数据交互格式，可以按照json指定的格式去组织和封装数据

json本质上是一种带有特定格式的字符串

主要功能：json就是一种在各个编程语言中流通的数据格式，负责不同变成语言中的数据传递与交互，类似于：国际通用语言为英语，而中国56个民族不同地区的通用语言为普通话如何导入？

import json

data = 。。。。。。。

data = json.dumps (data) 这里是把python数据转化为了json数据

data = json.loads(data) 这里是把json数据转化为了python数据

如果要在转化的过程中正确展示中文的话 ensure\_ascii=False

（2）pyecharts简单介绍

可以帮助我们做出数据可视化

快速入门：基础折线图

*# 导包*from pyecharts.charts import Line  
*# 创建一个折线图对象*line = Line()  
*# 给折线图加x轴的数据*line.add\_xaxis(["中国","美国","英国"])  
*# 给折线图加y轴数据*line.add\_yaxis("GDP",[30,20,10])  
*# 通过render，将代码生成图像*line.render()

pyecharts有哪些配置选项

1. 全局配置选项 2. 系列配置选项

全局可以通过 set\_global\_opts方法来进行配置

line.set\_global\_opts(  
 title\_opts = TitleOpts("GDP展示",pos\_left="center",pos\_bottom="1%"),  
 legend\_opts=LegendOpts(is\_show=True),  
 toolbox\_opts=ToolboxOpts(is\_show=True),  
 visualmap\_opts=VisualMapOpts(is\_show=True),  
)

全局配置能做什么？

配置图标的标题 配置图例 配置鼠标移动效果 配置工具栏

31. 面向对象

（1）初始对象

在程序中是可以做到生活中那样设计表格、生产表格、填写表格的组织形式的

在程序中设计表格：我们称之为class（设计类）

class Student:

name = None

在程序中打印生产表格，我们称之为：创造对象

#基于类创造对象

stu\_1 = Student()

在程序中填写表格，我们称之为：对象属性赋值

stu\_q.name = “周杰伦”

class Student:  
 name = None  
 gender = None   
 nationality = None  
 native\_place = None  
 age = None  
*# 创建对象，类比于生活中的打印一张登记表*stu\_1 = Student()  
*# 对象属性进行赋值，类比于生活中填写表格*stu\_1.name = "周杰伦"  
.......   
print(stu\_1.name)

总结：在程序中

设计表格就是设计类（class） 打印表格就是创建对象 填写表格就是对象属性赋值

（2）类的成员方法

class 类名称：

类的属性 就是定义在类中的变量（成员变量）

类的行为 就是定义在类中的函数（成员方法）

创建类的对象： 对象 = 类名称（）

那么什么是类的行为（方法）呢？

class Student:  
 name = None  
 age = None  
 def say\_hi(self):  
 print(f"Hi大家好，我是{self.name}，我今年{self.age}岁了")  
stu = Student()  
stu.name = "周杰伦"  
stu.age = 33  
stu.say\_hi()

值得注意的是，如果没有创建对象，直接Student.say\_hi()的话时会报错的

就像是这个表格原件就拿去填写，这是不合理的

类中定义的行为（函数）我们称之为：成员方法

类中定义的属性（变量）我们称之为：成员变量

在类中不仅可以定义属性来记录数据，也可以定义函数用来记录行为

从现在开始，定义在类内部的函数我们称之为方法

换而言之，函数这个名称就是用于写在外面的函数

在上面这个例子中self关键字十分重要

在类中定义成员方法和定义函数基本一致，但是有着很小的差别

def 方法名称（self,形参1，形参2……）：

方法体

self是成员方法定义中必须填写的

它表示类对象自身的意思；当我们使用类对象调用方法的时候，self也会自动输入

在方法的内部，想要访问类成员的成员变量，必须使用self

（3）类和对象

现实世界的事物和类：在现实中现实事物拥有两个重要的东西：属性和行为

类也有属性和行为，实用程序中的类可以完美描述现实世界的事物

但是为什么还需要创建对象才能去使用呢？其实这只是一种设计思想，类只是程序中的设计图纸，需要基于图纸生产实体（对象）才能正常工作

这种思路就是面向对象编程

（4）构造方法

可以使用\_\_init\_\_()方法，称之为构造方法

可以实现：在创建类对象（构造类）的时候会自动执行

在创建类对象的时候，将传入参数自动传递给\_\_init\_\_方法使用

class Student:  
 name = None 注意：这三行其实可以不写  
 age = None 在下面self的部分就会对变量进行声明  
 tel = None  
 def \_\_init\_\_(self,name,age,tel):  
 self.name = name  
 self.age = age  
 self.tel = tel  
 print("我创建了一个类对象")  
stu = Student("林俊杰",31,"18500006666")

第一个特性：自动执行。构建类对象的时候它就会自动执行

第二个：传入的参数都会改到对应的参数，而且里面的程序逻辑会自己跑起来

案例：输入五个人的学生信息，包含姓名年龄地址三类信息

每完成一次输入后，使用print语句完成信息的输出

在输入五次后自动停止并输出信息

i = 1  
class Student:  
 def \_\_init\_\_(self,name,age,address):  
 self.name = name  
 self.age = age  
 self.address = address  
 print(f"第{i}位的信息已经录入，信息为：【学生姓名：{self.name}；年龄：{self.age}；地址为：{self.address}】")  
stu\_dict = {}for x in range(1,6  
 print(f"第{i}位同学您好！请输入您的姓名、年龄和地址！")  
 stu = Student(input("请输入你的姓名："),input("请输入你的年龄："),input("请输入你的地址："))  
 stu\_dict[f"学生{i}"]={}  
 stu\_dict[f"学生{i}"]["姓名"] = stu.name  
 stu\_dict[f"学生{i}"]["年龄"] = stu.age  
 stu\_dict[f"学生{i}"]["地址"] = stu.address  
 print(f"第{i}位同学的信息录入已经完成！请下一位同学输入信息！")  
 i += 1  
print ("所有同学的信息已经录入！")  
print(stu\_dict)

（5）其它内置方法

\_\_init\_\_()放置方法只是内置方法之一

还要学习四个存放方法

1. \_\_str\_\_ 可以将内容转化为字符串

class Student:

def \_\_init\_\_(slef,name,age):

self.name = name

self.age = age

def \_\_str\_\_(self):

return f”name:{self.name};age:{self.age}”

stu = Student(“周杰伦”,31)

可见在str的作用下，周杰伦字符串得以正确表达

2. \_\_lt­­\_\_ 在类中实现这个方法可以完成小于符号和大于符号比较

class Student:

def \_\_init\_\_(slef,name,age):

self.name = name

self.age = age

def \_\_lt\_\_(self,other)

return self.age < other,age

stu1 = Student(“林俊杰”,31)

stu2 = Student(“周杰伦”,36)

print(stu1 > stu2)

# 会输出False

本来stu1和stu2是不可以比较的，但是在\_\_it\_\_的作用下，age这个可以比较的量单独被拿出来进行比较了

3. \_\_le\_\_ 和\_\_lt\_\_用法一模一样

只不过是支持<= 和 >=罢了

4. \_\_eq\_\_ 无需多言

（6）封装

理解封装的概念 掌握私有成员的使用

面向对象的三大特征：封装 继承 多态

面向对象编程，是许多编程语言都支持的一种编程思想

简单理解是：基于模型（类）去创建实体（对象），使用对象完成功能开发

封装定义：将现实事物的属性和行为封装到类里面，用来描述成员变量和成员方法，从而完成程序对现实世界事物的描述

现实世界中的事物有属性和行为，但是不代表这些属性和行为都是开放给用户的

私有成员：既然现实事物有不公开的属性和行为，那么作为现实事物在程序中映射的类，也应该支持；类中提供了私有成员的形式来支持：私有成员变量、私有成员方法

定义方法十分简单 只要变量名称以\_\_开头就变成私有成员变量

只要方法名称以\_\_开头就会变成私有成员方法

用户在使用私有方法的时候会报错；私有变量赋值虽然不会报错但是无效

注意！！私有的内部是可以使用的！只不过是外部不能使用！！

即：类对象无法访问私有成员，但是类中其他的成员可以访问私有成员

案例：设计一个带有私有成员的手机

私有成员变量： \_\_is\_5g\_enable 类型为bool，True表示开启5g，False表示关闭5g

私有成员方法： \_\_check\_5g()，会判断私有成员\_\_is\_5g\_enable的值

如果是True输出：5g开启

如果是False输出：5g关闭，使用4g网络

公开成员方法： call\_by\_5g()调用它会执行

调用私有成员方法： \_\_check\_5g() 判断5g状态

打印输出： 正在通话中

运行结果： 5g关闭，使用4g网络

正在通话中

class phone:  
 \_\_is\_5g\_enable = False  
 def \_\_check\_5g(self):  
 if self.\_\_is\_5g\_enable == 0:  
 print("5g关闭，使用4g网络")  
 else:  
 print("5g开启")  
 def call\_by\_5g(self):  
 self.\_\_check\_5g()  
 print("正在通话中")  
Phone = phone()  
Phone.call\_by\_5g()

注意：def每一次使用变量或函数，前面必须加self

在这个案例中·，判断和内部的5g边变量是客户不需要知道的，因此选择了私有成员

（7）继承

单继承：class Phoine: class Phone2023(Phone):

被继承的是父类， 继承者称为子类

多继承 class 类名（父类1，父类2，……）：

重要:pass关键词 孤零零一个 就可以完善语法等等让子类得以运行

保证语法的完整性，表示无内容、空的意思

多继承中，如果父类有同名的方法或属性，先继承的优先级高于后继承的

（8）类型注解

在代码中交涉数据交互的地方，提供数据类型的注解（显示的说明）

主要功能：帮助对代码进行类型判断。协助做代码提示

帮助开发者自身对变量进行类型注释

为变量设置类型注解： 变量：类型

var : int = 10

类对象类型注解

class student:

pass

stu : Student = student()

对基础容器类型进行注解：  
my\_list : list = [1,2,3]

对容器类型详细注解：

my\_dict : dict[str,int] = {“itheima”:666}

同样，可以在注释中进行类型注释

def func()

return 10

var = func() # type: int

一般在无法直接看出变量类型的时候才加类型注解

注意：类型注解仅仅是提示性的，而不是决定性的

即使定义错了，代码是不会报错的

同时，我们还要掌握为函数（方法）形参和返回值进行类型注解

def func(data):

data.cpp

func()

本意是想传入一个list 但是程序不知道data是列表

def 函数方法名（形参名：类型，形参名：类型……）：

def 函数方法名（形参名：类型，形参名：类型……） -> 返回值类型：

特殊： Union类型

Union【类型1，类型2……】

my\_dict: dict[str,union[str,int]] = {“name” = ”周杰伦”，”age” = 32}

注意！！使用union的时候必须先导包！

from typing import Union 注意这里的Union的U是大写的

（9）多态

多态：多重状态，即完成某种行为时，适用不同对象会得到不同的状态

多常态作用在继承关系上

比如： 函数方法形参声明接受父类对象

世纪传入父类的子类对象进行工作

即： 以父类做定义说明

以子类做实际工作

用以获得同一行为，不同状态

class Animal():  
 def speak(self):

pass

class Dog(Animal):  
 def speak(self):  
 print(“wang”)

class Cat(Animal):

def speak(self):

print(“miao”)

dog = Dog()

cat = Cat()

def makenoise(animal:Animal)

animal.speak()

makenoise(dog)

makenoise(cat)

父类的speak是空实现，设计的含义：

让父类确定那些方法，但是不实现；具体实现要子类自行确定

这种写法就叫做抽象类（接口）

含有抽象方法的类称为抽象类

方法体是空实现的称之为抽象方法

抽象类的作用：  
对用于顶层设计，以便于子类做具体实现

也是对子类的一种软性约束，要求子类必须复写父类的一些方法

并配合多态使用，获得不同的工作状态

32. SQL入门与实战

（1）什么是SQL？为什么学习它？

SQL语言是人人必备的开发技能，后续学习是要使用SQL的

（2）数据库

数据有两个重要的东西：数据的储存、数据的计算

python能算作数据的计算，但是如何出储存呢？我们需要使用数据库

数据库是如何组织数据并存储的呢？

库 – 表 – 数据 这三个层级组织数据；我们需要借助数据库管理系统

我们是用MySQL数据库

数据库可能涉及到：新增 删除 修改 查询 管理等

SQL语言就是对数据库、数据进行操作、管理、查询的工具

（3）SQL入门使用 掌握使用图形工具操作

在mysql命令行环境下，可以通过以下口令进行操作

show databases 查看有那些数据库

use 数据库名称 使用某个数据库

show tables 查看数据内有哪些表

exit 推出环境 这些是基础命令

使用命令指示符操作MYSQL不是很方便，一般开发者会使用第三方图形化工具使用；可用于它的图形化工具非常多，我们使用Dbeaver

（4）SQL基础与DDL

SQL是结构化查询语言，用于访问和处理数据库的标准的计算机语言

已经成为了统一的数据操作标准语言，成为了操作数据库的专用语言

SQL基于功能分为四类：

DDL 数据定义 库的创建 表的创建删除

DML 新增数据 删除数据 修改数据

DCL 新增用户 删除用户 密码修改 权限管理

DQL 基于需求查询和计算数据

SQL基础语法特征 第一点 大小写不敏感！！

第二点 不管写多少行，每一个操作都是以分号进行结束

第三点 支持注释 单行注释： -- 注释内容（注意—后面一定要有一个空格）

单行注释 # 注释内容 （# 后面可以不加空格，推荐加上）

多行注释 /\* 注释内容 \*/

DDL库管理 学习一下简单的数据管理相关DDL

查看数据库 SHOW DATABASES;

使用数据库 USE 数据库名称；

创建数据库 CREATE DATABASE 数据库名称 CHARSET UTF8；

其中CHARSET UTF8 可以不写，但还是建议写一下！

删除数据库 DROP DATABASE 数据库名称；

查看当前使用的数据库 SELECT DATABASE()；

注意只有查看数据库才会有DATABASES后面的复数 s！

上述关于库的操作，DDL中还有表的相关操作

SHOW TABLES； 查看有哪些表 但是要先选择数据库！！

DROP TABLE； 表名称

DROP TABLE IF EXISTS； 表名称

CREATE TABLE 表名称（

列名称 列类型，

列名称 列类型，

……

）;

列类型有： int 整数 float 浮点数 varchar 文本，长度为文字，做最大长度限制

data 日期类型 timestamp 时间戳类型

其中varchar其实类似于Python中的str

name varchar (20) 即名字为文本，但是名字长度不能超过20

（5）DML语言

DML是对数据库中的表的数据记录进行更新

INSERT INTO 表【（列1，列2……）】VALUES(值1，值2，……)

【，（值1，值2,……）,(……)】

其中，中括号代表里面的内容可以不写，不是你写的时候真的加中括号！

表后面的列表可以不写，不写的时候按照默认的创建顺序进行数据的插入

数据删除 DELETE

DELETE FROM 表名称 【where 条件判断】；

条件判断 = > < <= >= !=

数据更新 UPDATE

UPDATE 表名 SET 列 = 值 【WHERE 条件判断】；

（6）DQL语法

1. 基础查询：

在SQL中，使用select关键字开头的语句进行数据的查询

select 字段列表|\* from 表；

如果是\*代表是查询全部数据

2. 分组聚合 关键字 GROUP BY

SELECT 字段|聚合函数 FROM 表 【where 条件】GROUP BY 列；

聚合函数有：

SUM(列) 求和

AVG(列) 求平均值

MIN(列) 求最小值

MAX(列) 求最大值

COUNT(列|\*) 求数量

如 select gender,avg(age) from student Group by gender;

这里面有着SQL语言限制：只有group by 里面写了什么，前面才能出现什么

当然聚合函数是另外的，不受限制，本身只是一种运算

聚合函数里面的字段是没有限制的！！只要是能扔进函数中进行预算即可

当然，一个SQL语句中是可以写多个聚合函数的

总结：GROUP BY 中出现了哪个列，哪个列才能在SELECT中的非聚合中出现！！

3. 排序分页

可以对查询的结果，使用ORDER BY 的关键词，指定某个列进行排序，语法如下：  
SELECT 列|聚合函数|\* FROM 表

WHERE。。。。。。

GROUP BY ……

ORDER BY ……[ASC]DESC] ASC 是从小到大 DESC 是从大到小

如：

selecet \* from student where age > 20 order by age asc;

同样，可以使用关键词LIMIT，对查询结果进行数量限制或分页显示，语法：  
SELECT 列|聚合函数|\* FROM 表

where ……

GROUP BY ……

GROUP BY ……[ASC|DESC]

LIMIT n[,m]；

如果只有n,这里的n代表显示的数据的数量

但是如果有m，则代表跳过前n条，取后面的m条

select age,count(\*) from student where age > 20 group by age order by age asc limit 3;

代表从表student中按照大于20岁的年龄进行升序排序并且只取前3项数据

select from 是一定要写的，其他都可以按照需求进行省略！！

顺序 from where group by 和聚合函数 select order by limit

(7) Python操作MySQL数据库软件

1. 基础使用

除了图形化工具之外，我们可以使用编程语言来执行SQL从而操作数据库

在Python中，使用第三方数据库：pymysql来完成对MySQL数据库的操作

from pymysql import Connection  
*# 构建到MySQL数据库的连接*conn = Connection(  
 host="localhost", *# 主机名* port=3306, *# 端口默认为3306* user="root", *# 账户* password="www050308"  
)  
print(conn.get\_server\_info())  
*# 执行官非查询性质的SQL  
# 获取到游标对象*cursor = conn.cursor()  
*# 选择数据库*conn.select\_db("test")  
*# 执行sql*cursor.execute("create table test2\_pycharm(id int);") *# 分号可省可不省  
# 执行查询性质的SQL语句  
# 代表从test数据库中的student表中选取全部的数据*cursor.execute("SELECT \* FROM student")  
*# 获取查询结果  
# 用元表来承装数据*results:tuple = cursor.fetchall() *# ：tuple是类型注解 results 是变量，用于接受返回值*for r in results: *# 遍历循环进行输出* print(r)  
conn.close() *# 关闭数据库连接*

2. 数据插入

from pymysql import Connection

conn = Connection(

……)

cursor = conn.cursor() 获取到游标对象

conn.select\_db(“world”)

cursor.execute(“**insert into** student values(……)”)

代表在world数据库里面的student表里面插入数据

conn.commit() 注意！这一步是十分重要的！要手动确认！

conn.close

如果不想手动确认，只需要在Connection(

autocommit = True

) 就可以一劳永逸