**5.20+5.21 学习日记**

* ~~1.学习 python~~

学习内容：

Python 变量、数据类型、条件语句、循环（推荐《Python Crash Course》第 1-3 章）。

练习：编写脚本实现批量重命名文件（例如将 test1.txt 改为 test\_1.txt）。

安装 Python 环境（推荐 PyCharm Community 版），熟悉 Jupyter Notebook 交互式编程。

成果：完成 3 道 LeetCode Easy 题（如两数之和、反转字符串）。

学习过程：

1.安装 anaconda（是一个开源的 python 发行版，预装了几百个常用的科学计算库，使得环境配置变的简单）

* anaconda 的下载
* anaconda 的安装（一直出错）

Preparing transaction: ...working... done

Executing transaction: ...working... done

ERROR conda.core.link:\_execute(950): An error occurred while installing package 'defaults::plotly-5.24.1-py312hfc267ef\_0'.解决办法：将安装包安装到 D 盘就没有问题了

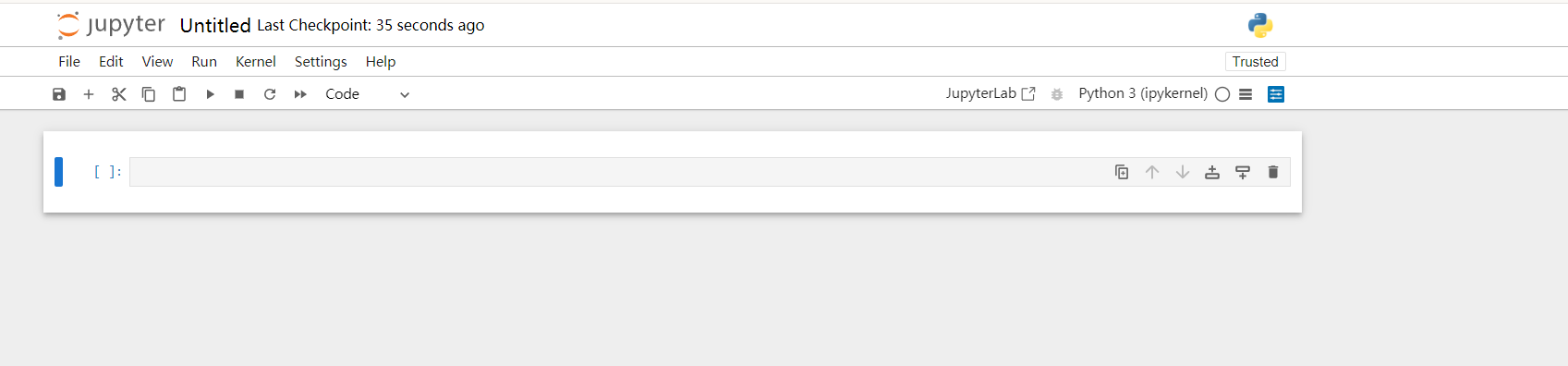
2.JupyterNode 启动后就可以进行代码运行，这个可以交互式进行代码输出，如下图

有 code，markdown，raw 三种形式

code:写代码执行

markdown:可以进行 css 安排文本格式

raw:不会执行其中的内容



3.学习（因为之前学过 python，但是我对文件的操作，异常，类以及爬虫都不是很熟悉，囧！！)

* python 数据类型，
* 列表，组织列表进行排序 sort（reverse=true[倒序]），sorted（）->一次性的，cars.reverse（）-->将元素倒着打印，length（）-->计算列表长度
* 遍历整个列表，for 循环：，用缩进来表示结构体
* 文件和异常类：从文件中读取数据，（访问文件后关闭）

flie\_path=' '

with open（file\_path） as file\_object://同目录下， 斜杠方向为/

contents = flie\_object.read（） //读取文件的全部内容

print(contents.rstrip())

* 进行逐行读取，使用 for 循环：  
   for line in flie\_object:
* 将读取的内容放到列表中，然后输出列表中的内容：

flie\_path=' '

with open（file\_path） as file\_object://同目录下， 斜杠方向为/

lines=file\_object.readlines（） //读取文件的全部内容，放到 line 这个列表中

for line in lines:

print(line)

* 使用文件中的内容

flie\_path=' '

with open（file\_path） as file\_object://同目录下， 斜杠方向为/

lines=file\_object.readlines（） //读取文件的全部内容，放到 line 这个列表中

pi\_string=' '

for line in lines:

pi\_string+=line.strip()

print（pi\_strin[:52]）//用切片进行部分输出

print(len(pi\_string))

* 在文档中进行匹配

flie\_path=' '

with open（file\_path） as file\_object://同目录下， 斜杠方向为/

lines=file\_object.readlines（） //读取文件的全部内容，放到 line 这个列表中

pi\_string=' '

for line in lines:

pi\_string+=line.strip()

if birthday in pi\_string:

print("")

* python 创建文件，并向其中写入语句

file\_name=" "//不存在则自动创建

With open（file\_name，' w'）as file\_object://定义写入模式，会清空文件中的内容

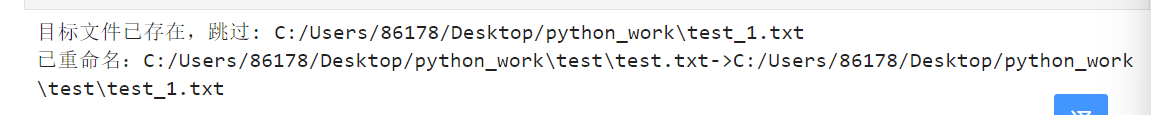
file\_object.write(" ")

4.练习：编写脚本实现批量重命名文件（例如将 test1.txt 改为 test\_1.txt）。

* 使用os.walk()的方式查找指定文件

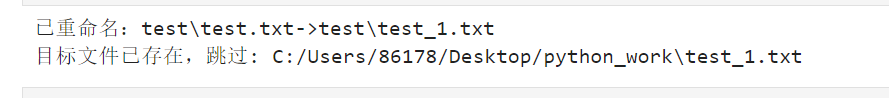
总结：需要使用os.walk(),os.path.join(),os.rename()/os.replace(),注意编程细节

|  |
| --- |
| Python 批重命名文件os.py #读取该目录下的所有文件，找到文件名是test1，并且是txt格式 #需要引入os模块，实现与操作系统交互,os模块可以执行文件操作，目录操作，环境变量管理，进程管理等任务 #os.walk()方法，创建一个生成器，用以生成所要查找的目录及其子目录下的所有文件 #os.walk(top[,topdown=True[,onerror=none[,followlinks=False]]]) #其中：top根目录下的每个文件，包含他自己，返回（dirpath,dirname,filenames）三元组：文件夹路径，文件夹名字，文件夹 #topdown:目录按照目录树自上而下输出 #将文件名改为test\_1.txt,使用os中的replace函数 import os root\_path="C:\Users\86178\Desktop\python\_work" name=test for root,dirs,files in os.walk(root\_path): #for file in files:  #if name in file and file.endswith(".txt"):  if name in files and files.endswith(".txt"):  os.replace('test.txt','test\_1.txt')  #这个是相对路径容易找不到目标文件，要使用绝对路径，才不会报错  print("已修改") #这个代码有缺陷 #首先是os.walk函数返回的的是一个生成器，files是一个列表，要在列表中遍历,看是否有同名文件 #os.replace()对于已经存在的目标文件会直接覆盖目标文件的内容，只留下源文件的内容 #os.rename()对已经存在的目标文件操作会报错，不可以跨磁盘操作 import os name="test" root\_path="C:/Users/86178/Desktop/python\_work" for root,dirs,files in os.walk(root\_path):  for file in files:  if name in file and file.endswith(".txt"):  old\_file\_path = os.path.join(root, file) *# 完整路径* new\_file\_name = file.replace(name, f"{name}\_1") *# 修改文件名* new\_file\_path = os.path.join(root, new\_file\_name)  print("已修改") #这个代码逻辑不完整，没有考虑文件名中包含test，这样就不是目标文件了，以及test\_1文件是否存在，是否选择覆盖 #优化后的代码如下： import os name="test.txt" root\_path="C:/Users/86178/Desktop/python\_work" for root,dirs,files in os.walk(root\_path):  for file in files:  if file==name:  old\_path=os.path.join(root,file)  new\_name=file.replace(name,"test\_1.txt")#参数为路径，应该使用字符串的替换操作  new\_path=os.path.join(root,new\_name)  if not os.path.exists(new\_path):  os.rename(old\_path,new\_path)  print(f"已重命名：{old\_path}->{new\_path}")  else:  print(f"目标文件已存在，跳过: {new\_path}") |



* 使用os.listdir(path)的方式进行查找：这里使用了递归的算法，使用的主要函数有os.listdir(),os.path.isdir()

|  |
| --- |
| Python import os root\_path="C:/Users/86178/Desktop/python\_work" name="test.txt" def use\_listdir(path,file\_name):  for dirs in os.listdir(path):  if os.path.isdir(os.path.join(root\_path,dirs)):  use\_listdir(dirs,file\_name)  else:  if dirs==name:  old\_path=os.path.join(root,file)​  new\_name=dirs.replace(name,"test\_1.txt")#参数为路径，应该使用字符串的替换操作​  new\_path=os.path.join(root,new\_name)​  if not os.path.exists(new\_path):​  os.rename(old\_path,new\_path)​  print(f"已重命名：{old\_path}->{new\_path}")​  else:​  print(f"目标文件已存在，跳过: {new\_path}") use\_listdir(root\_path,name) |



* 使用glob模块（是一个用于文件路径模式匹配的标准库模块）

\*:匹配任意数量的字符

？：匹配单个字符

[]:匹配指定范围的字符

\*\*：递归匹配所有子目录

常用函数：

glob.glob(pattern,\*,recursive=False)

* 返回匹配pattern的所有文件路径列表
* 如果recursive=True,则递归搜查子目录

注：glob模块不能返回指定文件

|  |
| --- |
| Python import glob import os  root\_path="C:/Users/86178/Desktop/python\_work"​ name="test.txt" full\_pattern=os.path.join(root\_path,name) print(full\_pattern) |

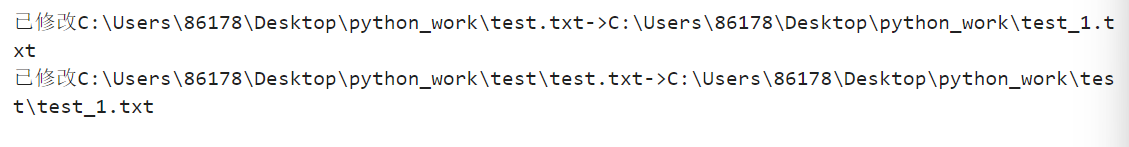


* pathlib模块(正解，快速，简洁，方便)

rglob()递归搜索所有子目录,返回所有匹配的Path对象

file.parent:读取父目录

|  |
| --- |
| Python from pathlib import Path root\_path="C:/Users/86178/Desktop/python\_work" dir\_path=Path(root\_path) for i,file in enumerate(dir\_path.rglob("test.txt"),1):#为了遍历生成对象  new\_name=file.parent/"test\_1.txt"  file.rename(new\_name)   print(f"已修改{file}->{new\_name}") |



5.异常

|  |
| --- |
| Python try:  print(5/0) '''出现异常会执行print语句''' except ZeroDivisionError:  print(" ") |

6.完成 3 道 LeetCode Easy 题（如两数之和、反转字符串）

有一些机器人分布在一条无限长的数轴上，他们初始坐标用一个下标从 0 开始的整数数组 nums 表示。当你给机器人下达命令时，它们以每秒钟一单位的速度开始移动。

给你一个字符串 s ，每个字符按顺序分别表示每个机器人移动的方向。'L' 表示机器人往左或者数轴的负方向移动，'R' 表示机器人往右或者数轴的正方向移动。

当两个机器人相撞时，它们开始沿着原本相反的方向移动。

请你返回指令重复执行 d 秒后，所有机器人之间两两距离之和。由于答案可能很大，请你将答案对 109 + 7 取余后返回。

|  |
| --- |
| Python import random import math mode=2\*10^9 length=random.randint(2,10^5)#机器人数量 site=[random.randint(-mode,mode)for i in range(length)]#机器人的初始位置 s=''.join(random.choices(['L','R'],k=length))#机器人运动的初始方向 print(s,length,site) d=int(input("请输入执行时间：")) def robot\_motion(spot,s,i):  if s[i]=='L':  spot[i]-=1  elif s[i]=='R':  spot[i]+=1 def aftercrash(s,i,j):  #处理两个机器人的碰撞  #因为字符串是不可变的，所以要将其转化为列表再进行合并，碰撞后应该交换两个机器人的运动方向  s=list(s)  if s[i]=='R':  s[i]='L'  elif s[i]=='L':  s[i]='R'  if s[j]=='R':  s[j]='L'  elif s[j]=='L':  s[j]='R'  s=''.join(s)#创建一个新的字符串变量，并让“局部变量S”指向它不会改变全局变量s的值要返回修改后的值  return s times=1 while(d>0): #移动-->检测碰撞->处理碰撞-->重复 #比较两个数的值性能太差了  '''for i in range(0,length):  for j in range(0,length):  if site[i]-site[j]==0 and i!=j:  s=aftercrash(site,s,i)  s=aftercrash(site,s,j)'''  for m in range(length): #移动所有机器人  robot\_motion(site,s,m)  # 第二步：检测并处理所有碰撞  # 使用双层循环检测所有机器人对  for i in range(length):  for j in range(i+1, length):#碰撞是相互的只需要检查你n\*(n+1)/2次  if site[i] == site[j] and i != j:  # 处理碰撞，反转两个机器人的方向  s = aftercrash(s, i, j)  print(f"{times}秒后，机器人的位置是 {site}")  times+=1  d-=1 sum=0 for i in range(length):  for j in range(i+1, length):  num=abs(site[i]-site[j])  sum+=num print(sum%(10\*\*9+7)) |

|  |
| --- |
| Python class Solution:  def sumDistance(self, nums: List[int], s: str, d: int) -> int:  mod = 10\*\*9 + 7  n = len(nums)  pos = [nums[i] - d if s[i] == 'L' else nums[i] + d for i in range(n)]  pos.sort()  return sum([(pos[i] - pos[i - 1]) \* i \* (n - i) for i in range(1, n)]) % mod |

7.类的创建和使用：

* 创建dog类

|  |
| --- |
| Python class Dog():#首字母大写 #对象属性 #行为->方法  def \_\_init\_\_(self,name,age):  #初始化属性,self必不可少，在形参前面  self.name=name  self.age=age  def sit(self):  #模拟小狗收到命令时蹲下  print(f"{self.name}is now sitting") my\_dog=Dog("willie",6) print(f"My dog'name is {my\_dog.name}") |

* 创建多个实例

使用类和实例：

|  |
| --- |
| Python class Car():  def \_\_init\_\_(self,make,model,year):  self.make=make  self.model=model  self.year=year  def get\_descriptive\_name(self):  long\_name=f"{self.year} {self.make} {self.model}"  return long\_name.title() my\_car=Car('china','aodi',2019) print(my\_car.get\_descriptive\_name()) |

给属性指定默认值：

|  |
| --- |
| Python class Car():  def \_\_init\_\_(self,make,model,year):  self.make=make  self.model=model  self.year=year  self.odo=0  def get\_descriptive\_name(self):  long\_name=f"{self.year} {self.make} {self.model}"  return long\_name.title()  def read\_odo(self):  print(f"This car has {self.odo} miles on it") my\_car=Car('china','aodi',2019) print(my\_car.get\_descriptive\_name()) my\_car.odo=23 my\_car.read\_odo() |

通过方法修改属性的值：

|  |
| --- |
| Python class Car():  def \_\_init\_\_(self,make,model,year):  self.make=make  self.model=model  self.year=year  self.odo=0  def get\_descriptive\_name(self):  long\_name=f"{self.year} {self.make} {self.model}"  return long\_name.title()  def read\_odo(self):  print(f"This car has {self.odo} miles on it")  def update\_odo(self,mileage):  self.odo=mileage my\_car=Car('china','aodi',2019) print(my\_car.get\_descriptive\_name()) my\_car.update\_odo(1024) my\_car.read\_odo() |

继承

* 子类的方法

|  |
| --- |
| Python class ElectricCar(Car):  #电动汽车父类在子类前面  def \_\_init\_\_(self,make,model,year):  super().\_\_init\_\_(make,model,year)   my\_tesla=ElectricCar("tesla",'models',2019) print(my\_tesla.get\_descriptive\_name()) |

* 定义子类的方法

|  |
| --- |
| Python class ElectricCar(Car):  #电动汽车父类在子类前面  def \_\_init\_\_(self,make,model,year):  super().\_\_init\_\_(make,model,year)  self.battery\_size=75    def describe\_battery(self):  print(f"This car has a {self.battery\_size}-")#访问类中的属性要用self   my\_tesla=ElectricCar("tesla",'models',2019) print(my\_tesla.get\_descriptive\_name()) my\_tesla.describe\_battery() |