**学习中遇到的困难以及希望杨老师解答的问题：**

1. print重复了问题**（图1）**这类问题如何排除  
   **没提供代码，无法调试**

怀疑是否是有两条数据，所以循环执行了两次，所以输出两次没有查到呢？具体还是需要有代码才能调试

检查代码的105行，else 是跟谁成“一对儿的”，for..else结构还是if…else结构， 猜测应该是for….else结构

**Python是通过代码缩进来确定程序的逻辑层次的**

1. 赋值=遇到问题,为什么在一行赋值时,没有改变列表的值？**（图2）**
2. 数据结构和int，float等语法在哪些地方会用到**（图3）**

**https://www.jianshu.com/p/dca29e1ce894  
数据逻辑结构是指对象中元素之间的相互关系(Python中数据的逻辑结构有哪些？集合，字典，列表，元组。。。)**

**数据物理结构是指在计算机中的存储形式（链式存储，顺序存储）**

**逻辑结构是面向问题的，物理结构是面向计算机的，其基本的的目标就是将数据及逻辑关系存储到计算机中**

**举例：新闻管理系统，使用哪种逻辑结构表示各条新闻之间的相互关系呢？**

**Int,float是变量值的数据类型 ，Python中变量是没有数据类型的，只有值有数据类型**

**只有明确数据类型，才能知道这个类型的对象具有哪些行为（方法）**

1. 装饰圈不好理解？（是不是装饰器啊？）  
   装饰器就是将函数作为参数，将函数作为返回值

想学装饰器， 需要了解三个知识点 （1，函数如何定义 2，函数如何传参，3 函数的返回值）

**创建一个闭包需要满足以下几点:**

**必须有一个内嵌函数**

**内嵌函数必须引用外部函数中的变量**

**外部函数的返回值必须是内嵌函数**

**闭包与装饰器区别**

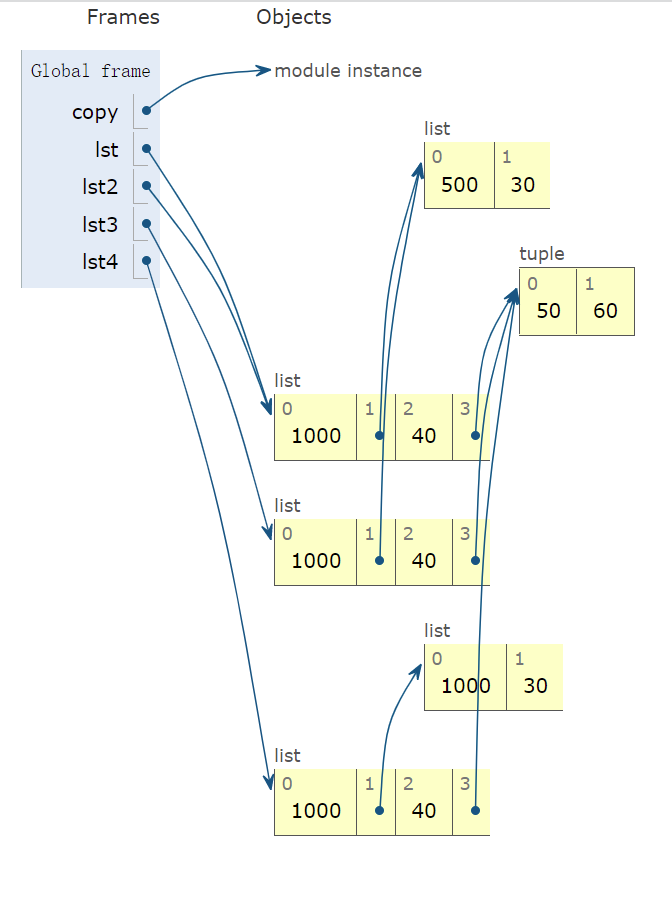
**闭包和装饰器的区别：闭包传递的是变量，而装饰器传递的是函数，除此之外没有任何区别，或者说装饰器是闭包的一种，它只是传递函数的闭包**

1. 上几次课都提到了逻辑内存图，不太了理 解其在代码分析的作用，能否 以赋值、深拷贝和浅拷贝为例，谈谈如何利用逻辑内存图进行深度分析？**（图4-5）  
   不可变对象不涉及到深拷贝的问题**

**深拷贝是从里到外全都复制一次**

**浅拷贝只是复制最外层的对象**

**在线代码调试工具：https://pythontutor.com/render.html#mode=display**

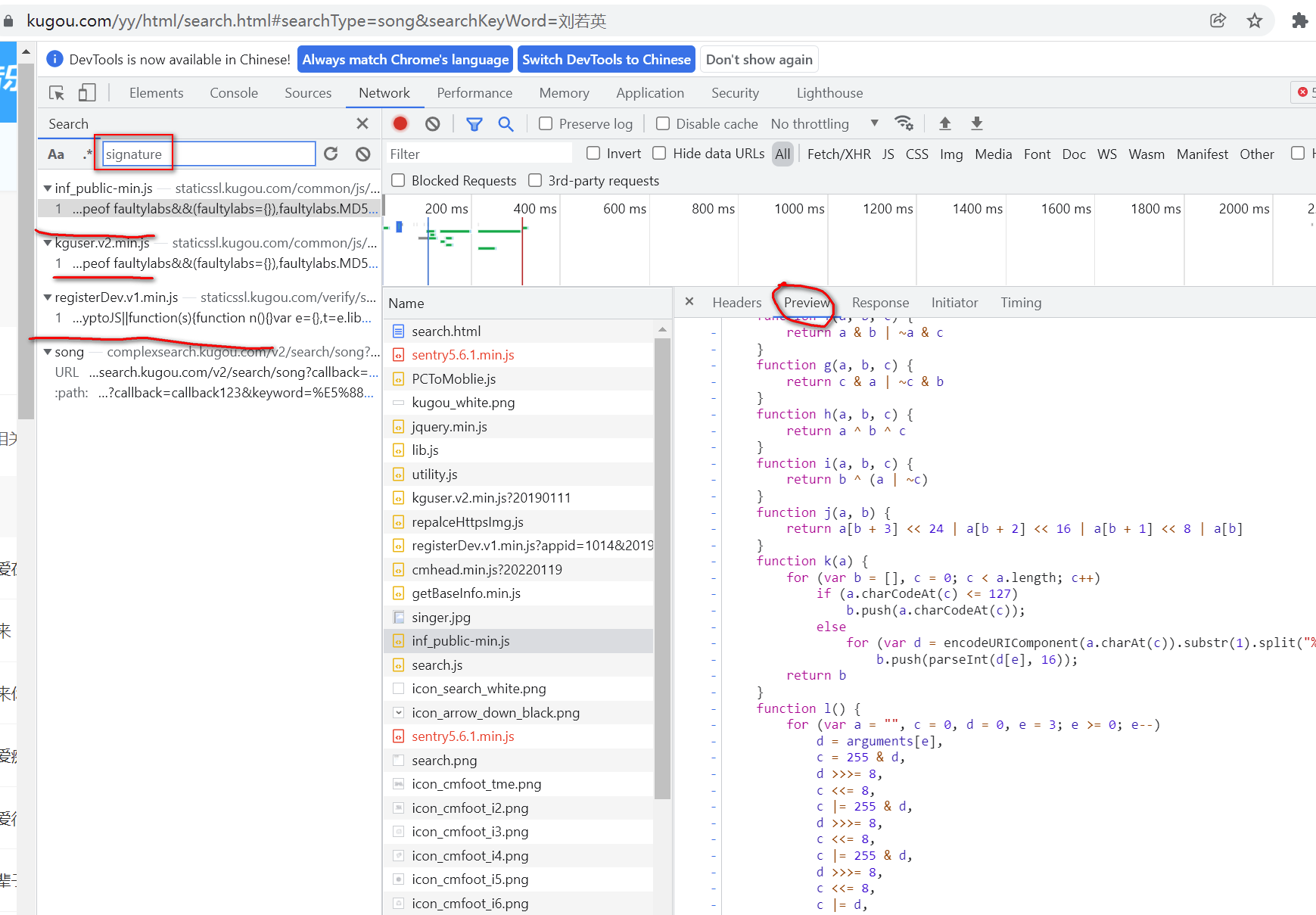


1. 爬芒果TV上的电影，放在Pycharm里面试一下成功的放cmd才可以下载电影是吗？自己写的写到这里不会了看一下。自己写的这个看一下算是python还是java的格式啊？**（图6）（文件1）**芒果TV这个爬虫代码看一下。有问题。  
   没有代码，无法参与调试

通过异常截图，可猜测是代码缩进产生了问题

1. 代码中的加密列表哪里可以获取，希望杨老师举个例子**（图7）（文件2）  
   requests-html是一个最简单的爬虫框架 ，这是一个第三方框架，在使用时，需要安装**

**pip install requests-html**

**需要对JS 有很深的功底**

1. 深拷贝与浅拷贝不理解  
   (参考第5题 )
2. 希望杨老师帮我拓展一下函数和类的内容

**（参考第4题 ，装饰器）**

**1.函数的定义， 2.函数的参数， 3 .函数的返回值**

1. 想让老师讲解一下json模块的相关知识  
   json是Python内置的模块，不需要安装，直接可以使用

序列化与反序列化 可以简单理解为 数据类型的转换 字符串与字典的类型转换

str🡪dict dict--str

Python中一切皆对象，字典叫字典对象，字符串叫字符串对象

|  |  |  |
| --- | --- | --- |
| 序列化 | json.dumps(dict,indent) | 将字典转为字符串 |
| json.dump(dict,file) | 将字典写进文件 |
| 反序列化 | json.loads(dicts) | 将字符串转为字典 |
| json.load(file) | 读取文件中的数据并转为字典 |

1. 海龟画图 递归算法不太清楚  
   海龟画图了解即可 （开发根本不用，装。。。用的）

递归算法 : （不会使用递归的，去复习函数）

本质就是函数的调用（自己调用自己）

1. 爬虫爬取数据时，正则表达式怎么准确定位**（文件3）**

**. : 表示匹配除换行符\n之外的任何单字符**

**\* ：表示零次或多次**

**? :没有？表示贪婪模式，有？表示非贪婪模式**

**贪婪模式表示是尽可能多的匹配，非贪婪模式， 表示是尽可以少的匹配**

**（pattern） 匹配 pattern 并获取这一匹配  
跟在\*或者+后边用时，表示懒惰模式。也称非贪婪模式。就是匹配尽可能少的字符。就意味着匹配任意数量的重复，但是在能使整个匹配成功的前提下使用最少的重复。**

1. 使用浏览器去请求这个服务器里面没有的文件时，正常应该给客户端返回一个404.html页面,但是我这个报了解码的错误。客户端请求信息例如：https://172.16.132.40:8080/xiaowang.jpg**（图8-9）（文件4）**
2. 老师，可以讲一下timeit模块吗class Timer: """Class for timing execution speed of small code snippets. The constructor takes a statement to be timed, an additional statement used for setup, and a timer function. Both statements default to 'pass'; the timer function is platform-dependent (see module doc string). If 'globals' is specified, the code will be executed within that namespace (as opposed to inside timeit's namespace). To measure the execution time of the first statement, use the timeit() method. The repeat() method is a convenience to call timeit() multiple times and return a list of results. The statements may contain newlines, as long as they don't contain multi-line string literals. """ def \_\_init\_\_(self, stmt="pass", setup="pass", timer=default\_timer, globals=None): """Constructor. See class doc string.""" self.timer = timer local\_ns = {} global\_ns = \_globals() if globals is None else globals init = '' if isinstance(setup, str): # Check that the code can be compiled outside a function compile(setup, dummy\_src\_name, "exec") stmtprefix = setup + '\n' setup = reindent(setup, 4) elif callable(setup): local\_ns['\_setup'] = setup init += ', \_setup=\_setup' stmtprefix = '' setup = '\_setup()' else: raise ValueError("setup is neither a string nor callable") if isinstance(stmt, str): # Check that the code can be compiled outside a function compile(stmtprefix + stmt, dummy\_src\_name, "exec") stmt = reindent(stmt, 8) elif callable(stmt): local\_ns['\_stmt'] = stmt init += ', \_stmt=\_stmt' stmt = '\_stmt()' else: raise ValueError("stmt is neither a string nor callable") src = template.format(stmt=stmt, setup=setup, init=init) self.src = src # Save for traceback display code = compile(src, dummy\_src\_name, "exec") exec(code, global\_ns, local\_ns) self.inner = local\_ns["inner"] def print\_exc(self, file=None): """Helper to print a traceback from the timed code. Typical use: t = Timer(...) # outside the try/except try: t.timeit(...) # or t.repeat(...) except: t.print\_exc() The advantage over the standard traceback is that source lines in the compiled template will be displayed. The optional file argument directs where the traceback is sent; it defaults to sys.stderr. """ import linecache, traceback if self.src is not None: linecache.cache[dummy\_src\_name] = (len(self.src), None, self.src.split("\n"), dummy\_src\_name) # else the source is already stored somewhere else traceback.print\_exc(file=file) def timeit(self, number=default\_number): """Time 'number' executions of the main statement. To be precise, this executes the setup statement once, and then returns the time it takes to execute the main statement a number of times, as a float measured in seconds. The argument is the number of times through the loop, defaulting to one million. The main statement, the setup statement and the timer function to be used are passed to the constructor. """ it = itertools.repeat(None, number) gcold = gc.isenabled() gc.disable() try: timing = self.inner(it, self.timer) finally: if gcold: gc.enable() return timing def repeat(self, repeat=default\_repeat, number=default\_number): """Call timeit() a few times. This is a convenience function that calls the timeit() repeatedly, returning a list of results. The first argument specifies how many times to call timeit(), defaulting to 5; the second argument specifies the timer argument, defaulting to one million. Note: it's tempting to calculate mean and standard deviation from the result vector and report these. However, this is not very useful. In a typical case, the lowest value gives a lower bound for how fast your machine can run the given code snippet; higher values in the result vector are typically not caused by variability in Python's speed, but by other processes interfering with your timing accuracy. So the min() of the result is probably the only number you should be interested in. After that, you should look at the entire vector and apply common sense rather than statistics. """ r = [] for i in range(repeat): t = self.timeit(number) r.append(t) return r def autorange(self, callback=None): """Return the number of loops and time taken so that total time >= 0.2. Calls the timeit method with increasing numbers from the sequence 1, 2, 5, 10, 20, 50, ... until the time taken is at least 0.2 second. Returns (number, time\_taken). If \*callback\* is given and is not None, it will be called after each trial with two arguments: ``callback(number, time\_taken)``. """ i = 1 while True: for j in 1, 2, 5: number = i \* j time\_taken = self.timeit(number) if callback: callback(number, time\_taken) if time\_taken >= 0.2: return (number, time\_taken) i \*= 10

代码提供，直接提示Python文件

1. 才回来学习数据结构的时候发现快速排序的某一部没看懂**（图10-15）  
   提供代码调试**
2. 在网上看到一个多线程，服务器程序，看不懂流程**（图16）**@\*\*\*\*\*\*也能调用方法吗，这个参数是什么定义的  
   装饰器
3. 我想问在输入n的情况下如果flag1=False后面不加break，为什么还可以内循环 不是外循环终止了么 加上break才可以终止程序

words=["python","game","food","easy","number","integer"]

flag1=True

while flag1:

word=random.choice(words)

# s列表存贮打乱后单词的index

s=[]

while len(s)<len(word):

value=random.randint(0,len(word)-1)

# 保证s里的索引不会重复

if value not in s:

s.append(value)

# 乱序单词

wrong\_word=""

# 拼接

for i in s:

wrong\_word+=word[i:i+1]

# print(wrong\_word)

while True:

print("单词已生成：",wrong\_word)

guess=input("请输入你心里想的那个单词：")

if guess==word:

fun=input("恭喜，猜对了！！ 要继续么（y/n）")

if fun=="n" or fun=="N":

flag1=False

break

else:

break

else:

print("不对，请重猜.......")

1. 看到有些参考书提供的练习所用的解释器（如：IDLE，ANACONDA等）跟这边用（PYCHARM）有些不一样，请老师叙述一下这几个解释器有什么不同？

**IDLE与PyCharm是代码编辑器**   
IDLE ： 是Python解释器自带的编辑器

PyCharm： 是第三方编辑器

**Anaconda :是Python解释器**

ANACONDA ：是升级版的Python解释器（包含Conda，Python以及一大堆安装好的工具包，比如说numpy,pandas等）

1. 怎么操作读取excel数据转dict返回
2. flag = True和flag=False都是什么意思，什么情况用  
     
   变量赋值 ，布尔类型的变量只能取值为True或False

如果一个变量的值，只有两种情况，就可以使用布尔类型

1. flask框架搭建Web服务器不好理解,需要重新把网络编程那个在巩固一下么

复习下吧 ，（网络编程挺重要的，后面学习其它框架的时候也会用到网络编程的知识）