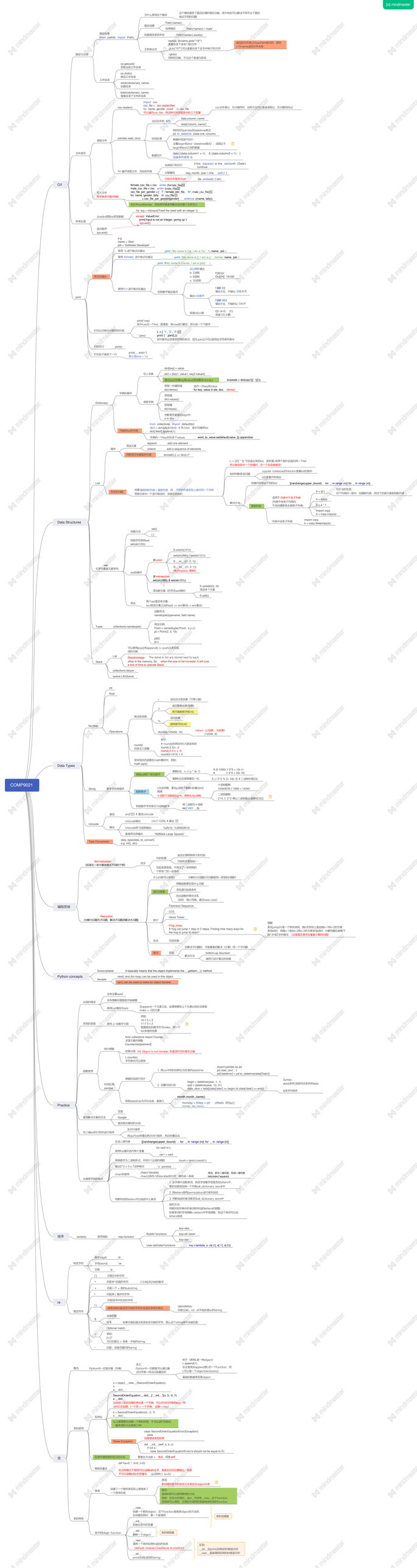
COMP9021
Os
路径与目录
路径处理
os.path、pathlib、import、Path
为什么使用这个模块
这个模块提供了路径处理的跨平台功能，其中有些可以解决不同平台下路径格式不同的问题
路径创建
Path("name")
拼接路径
Path("name") / 'trial'
检查路径是否存在
.exists()
Path("name").exists()
文件路径
os.getcwd()
获取当前工作目录
os.chdir()
修改工作目录
os.listdir(dictionary_name)
创建目录
os.mkdir(dictionary_name)
查看目录下文件的名称
文件读写
csv reader()
import _csv
csv_file = open('readcsv')
for name, gender, count in csv_file
可以遍历csv 文件，用这种方式能遍历csv中的三个变量
csv文件读以，为分隔符的，这种方法可以实现读取以，为分隔符的txt
读取文件
访问文件内的数据
data[column_name]
pandas.read_csv()
时间处理
将时间以Pandas的Dataframe格式，pd. to_datetime（data.one_column）
根据时间进行切片
设置loc=df[and（Dataframe格式）， 选取位字
begin=df[and之间的数据
数据切片
data=[data.column1 > 5] & [data.column2 < 1]
连接条件使用 &
写入文件
用字典来打印文件
for name, gender, count in dict_file
f.write（name, gender, count）
f.close()
for name, gender, count in sorted(dictionary_name.items())
f.write（name, gender, count）
f.close()
分隔数据
分隔符
split（','）
日期文件是否为空
file. endswith （'.txt'）
异常处理
出现异常时的处理
try: arg = int(input("Feed the seed with an integer:"))
except: ValueError:
print("Input is not an integer, giving up.")
sys.exit()
退出程序
sys.exit()
print
格式化输出
name = 'Alex'
job = 'Software Developer'
使用 % 进行格式化输出
print ("My name is %s, I am a %s", %(name , job))
使用 format() 进行格式化输出
print ("My name is {}, I am a {}" .format （name , job ））
print ("My name is {name}, I am a {job}" ）
使用 f 进行格式化输出
print f"My name is {name}, I am a {job}" ）
控制数字输出格式
以n进制输出
b: 二进制
o: 八进制
x: 十进制
f:[0]f[d]
.[n]f[d]: 10100'
输出小数点
f:{0:d}
输出 d 位，不够以 0 填补齐
f:{0: d}
输出 d 位，不够时以 0 补齐
保留n位小数
f:3.1415: .2f
保留 2 位小数
打印以空格为分隔符的列表
print("row)
其中row为一个list，原理是，将row进行解压，并分成一个个数字
print（" ".join(L)）
L = ['1', '2', '3']
print（' '.join(L)）
将字数字从原来使用逗号的方式，现为join之后可以使用空字符串对其中
打印空行
print()
打印后不换行了一行
print(, , end=")
默认是end = '\n'
Data Structures
Dictionary
字典的操作
写入字典
dict[key] = value
dict = {key1: value1 , key2:value2}
通过dictkey和value来创建dictionary
brackets = dict(zip(k[], v[]))
读取字典
获取一对键值对
dict.items()
改代一对key和value
for key, value in ele_dict. items()
获取值
dict.values()
获取键
dict.keys()
判断是否某键在dicyy中
e in dict
可使用了进行格式化输出
from collections import defaultdict
dict = defaultdict(int) # 传入int，表示可能的list
dict[key].append(1)
字典的一个key对应多个value s
word_to_value.setdefault(value, []).append(w)
List
操作
添加元素
append
add one element
extend
add a sequence of elements
不影响原来数据的长度
len(list1) == len(L)?
特殊切片
创建 创建指定的列表 / 数据的拷贝，时，对原的列表实际上指向同一个内存
等号对另一个新的数据时，会修改原数
L = [0] * 5 "空间固定的数列从，其实是，将两个指针指向同一个list
所以修改其中一个的值，另一个也会被改变
如何判断是否问题
Jupyter notebook中的ipyhon查看id的指向
id(变量名)内存地址
创建内容相同不同的list
[randrange(upper_bound), for _ in range [n] for _ in range [m]]
解决方法：
复制列表
适用于 列表中不含子列表
列表中有多个列表
不会创建新表去复制子列表
b = a[:]
切片法的实现：
切了列表的一部分，创建新列表，将切下的部分复制到新列表
b = list(a)
b = a * 1
import copy
b = copy.copy(a)
列表中含有子列表
import copy
b = copy.deepcopy(a)
Tuple
collections.namedtuple()
创建形式
namedtuple(typename, field name)
用法示例
Point = namedtuple("Point", 'x y z')
pt = Point(2, 3, 10)
print
pt.x
Stack
List
可以调用pop()和append() 分 push()实现栈的功能
Disadvantage： The items in list are stored next to each other in the memory. So， when the size of list increase ,it will cost a lot of time to operate Stack
collections.deque
queue LifoQueue
Data Types
Number
int
float
Operations
除法和除数
/
四舍五入
%
四舍五入取整
round()
四舍五入取整
round(12345, -3)
return （12345， %3)取整
（12345, 6)
round(1.231,1) -> 0
round(3.1415) -> 3
使多数的函数的math模块中，例如：
math.sqrt()
String
数字字符操作
字符以外？如以数字
取整
字符以外？如以数字
字符数字字符与ord数字
Unicode
查找
ord('汉字') 查找UnicOde
输出
Unicodeb输出
chr(110035) # 输出 '汉'
Unicode转为字符数组
'\u20ac'.hr
'\U0002E5A7'.hr
直接用名称输出
'\N{Black Large Square}'
Type Conversion
data_type(data_to_convert)
e.g. int(), str()
编程思维
Normalization
(将某一些内容格式不同的个例)
优点
代码优雅
省去处理特殊的个别代码
代码的可读性
写起来更容易，不用为了一种特殊的个别写了另一种条件
代码的可维护性
Recursion
(分解大问题为子问题，解决子问题来解决大问题)
特点
分解问题为子问题
每个问题都是同样的问题
子问题解决方法
递归搜索
例子
Fibonacci Sequence
汉诺塔
Hanoi Tower
Frog Jump
A frog can jump 1 step or 2 steps, Finding how many ways for the frog to jump N steps?
限制：
某青蛙jump的第一个时间段，跳n步实际上是由跳n-1和n-2的次数来组成的，而跳n-1和跳n-2和n-3的次数来组成的，分解到最后就剩下1和2步或2步的情况（这是递归思考的重要计算的问题）
优点
代码简洁
缺点
在解决子问题时，可能重复的解决（分割）同一个子问题
bottom-up recursion
解决方法
缓存已经计算过的结果
Python concepts
Subscriptable
It basically means that the object implements the __getitem__() method
Iterable
next() and (for loop) can be used in this object
(item) can be used to make an object iterable
Practice
出现的顺序
没有按顺序输出
按照入栈的顺序输出
使用LIFO的Stack
在queue中一次只出来
字典的思维
使用 // 动数字分组
例如
16 // 5 = 3
11 // 5 = 2
按照除以的数字作为index，用一个list来储存结果
计数器
from collections import Counter
求某元素的频数
Counter(list)[element]
函数使用
统计频数
如果出现 'int' Object is not iterable, 检查变量名可能遮盖了函数
L.count(e)
统计某个元素出现次数
时间处理
pandas
时间处理
1. 将csv中的字段转化为时间对象datetime
import pandas as pd
pd.read_csv('r')
pd[datetime'] = pd.to_datetime(data['Date'])
2. 设置时间切片
begin = datetime(year, 1, 1)
end = datetime(year, 12, 31)
date_slice = data[data['date'] >= begin & data['date'] <= end]
Syntax:
data[条件]选择符合条件的data
起来到条件
转换Datetime为月份名称，星期几
month, month_name()
monday = friday + pd
offsets .BDay()
monday , day_name()
数学解决方案的方法
找度
Google
查找相关模块的文档
对二维list的列表的排序
先对行排序
用zip*list对整数进行再次的行排序，然后转置回去
生成二维列表
[randrange(upper_bound), for _ in range [n] for _ in range [m]]
使用for循环迭代两个变量
for w2 in L:
var1 = var2
将数字当二进制形式，并统计出现的频数
count = bin(n).count(1)
输出2^x + 3 x 7这种格式
x , pandas
char的使用
chan("literable)
char与ord为对Ibyrate进行分类，两方式一一对应
用法：到分二维列表，形成一维列表
list(chan*square)
判断列的字letter可以组成什么单词
1. 从字典中选取单词，然后字母数字表和letter中，看的这能否加到一个列表list, dictionary_word中
2. 用letter的两个sorted进行比较的结合
3. 判断组成的单词是否在list, dictionary_word中
指的方法
判断对字典中的单词和对应的letter比较的结：
如果单词的字母组数=letter的字母组数，那这个单词可以由letter组成
排序
sorted()
排序规则
key=function
Builtin functions
key=abs
key=str.lower
key=len
User-defined functions
key = lambda x: (x[0], x[1], x[2])
re
特定字符
数字字母的
\d
字母(word)
\w
空格
\s
[ ]
匹配[]中的字符
^
匹配除*后面的字符
[^2-4]匹配2-4的数字
+
匹配一个 + 的字(substring)
*
匹配任意 * 数量的字符
{ }
匹配加 * 的数量的字符
匹配加加号的数量字符
( )
捕获组和非捕获组中间的字符 比如说有特定格式
(abcdef)xy
匹配abc, cd, ef并将结果以为string
$
结束匹配
\d*$
如果它结尾是0或者数字结尾的字符，那么这个string就符合规范
Optional match
?
例如
[a-z]?
可以匹配a + 或者 + 并继续string
匹配：连续匹配匹配不到string
re
概念
Python中一切皆对象（字典）
定义：
Python中一切都是可以通过键值对的字典一样访问其属性的
例子：[表格].属性=lifeobject
L.append(1)
在这里其实append是L的一个function，而L只是一个object的dictionary
基础的数据类型是object
类的使用
x = object.__new__(SecondOrderEquation)
x
x.__init__
SecondOrderEquation.__init__(x, 3, -2, 7)
x.__dict__
这样做了其实就是的查找是一个字典，可以内动态的声明属性一样
动态的创建（一个变量 = 一个字典，创建 x.key）
x = SecondOrderEquation(3, -2, 7)
x .__dict__
实例化
一个类中的实例化，用class的方式
class SecondOrderEquation(object):
pass
raise Exception
自定义的错误类型的类
raise SecondOrderEquationError('a should not be equal to 0')
实例中调用的自定义的方法
常常在方法前 + 类名, 或者self.
def find(self, *, b=0, c=0):
if not a :
raise SecondOrderEquationError('a should not be equal to 0')
继续定义
在这种情况下调用可以编辑的名字，直接在对应的变量后，但是
不可以调用a.x的变量名 （此调用(1, b=2) ）
类的特性
继承
创建了一个新的类实际上是继承了一个原本的类
新的类的特点
新的类的特点可以使用原类的方法
例如：自定义的类的__init__中没有__new__这个function，但是可以调用，这是因为调用的是继承来的原类的function
类的创建器
类中的Magic function
__new__
创建一个新的对象时，这个function会调用object的方法的，在创建实例时，第一个被调用
__init__
初始化新对象的变量
__del__
删除一个object
类的销毁器
__repr__
调用一个类的实例的时候的信息
[default: module.Classname at position]
定义：
__str__和repr的区别的详细显示的
__repr__是编程用的简单的字符显示的
__str__
print实用的显示string