第一次作业内容

10 / 0

程序填空题

1.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
list = [2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
ans = ___(list) # 将list转换成字符串
print(ans,end='')

你的答案：

```
list = [2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
ans =str(list) # 将list转换成字符串
print(ans,end='')
```

正确

2.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
list = [2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
ans = ___(list) # 对list进行升排序
print(ans,end='')

你的答案：

```
list = [2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
ans = sorted(list) # 对list进行升排序
print(ans,end='')
```

正确

3.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
list = [2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
ans = ___(list) #计算list所有元素之和
print(ans,end='')

你的答案：

```
list = [2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
ans = sum(list) #计算list所有元素之和
print(ans,end='')
```

正确

4.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
list = [2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
ans = ___(list) #计算list最大的元素
print(ans,end='')

你的答案：

```
list = [2, 4, 0, 6, 10, 7, 8, 3, 9, 1, 5]
ans = max(list) #计算list最大的元素
print(ans,end='')
```

正确

5.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
ans = ___(3,5) # 生成实部为3，虚部为5的复数ans
print(ans,end='')

你的答案：

```
ans =complex(3,5) # 生成实部为3，虚部为5的复数ans
print(ans,end='')
```

正确

6.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
num_hex = '0x1E'
ans = ___(num_hex, 16) # 将number转化成十进制
print(ans,end='')

你的答案:

```
num_hex = '0x1E'
ans = int(num_hex, 16) # 将number转化成十进制
print(ans,end='')
```

正确

7.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
number = 30
num_hex = ___(number) # 将number替换成十六进制
print(num_hex,end='')

你的答案:

```
number = 30
num_hex = hex(number) # 将number替换成十六进制
print(num_hex,end='')
```

正确

8.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
number = 30
num_hex = ___(number) # 将number替换成二进制
print(num_hex,end='')

你的答案：

```
number = 30
num_hex = bin(number) # 将number替换成二进制
print(num_hex,end='')
```

正确

9.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
string = 'Hello World'
ans = string.___('utf-8') # 使用utf-8编码格式进行编码
print(ans,end='')

你的答案：

```
string = 'Hello World'
ans = string.encode('utf-8') # 使用utf-8编码格式进行编码
print(ans,end='')
```

正确

10.# 将下面的代码复制到答题框中，并将其中的'___'替换成相应的python内置函数，以实现注释中的所描述的效果
# 打印0到12之间的整数
for i in ___(13):
___(i,end=' ')

你的答案：

```
# 打印0到12之间的整数
for i in range(13):
        print(i,end=' ')
```

正确

返回

第二次作业内容

0 / 2

编程题

1.编写函数，计算出N-M之间的素数，并返回素数的个数
函数输入：N,M
函数输出：素数个数
计算N-M之间的素数，并返回素数个数
（先将下面三条代码复制到答题框）
def fact(N,M):# 加提示
在下方填充代码，注释掉pass
pass

你的答案：

```
#pass
    s=0
    for i in range(N,M+1):
        flag=True
        for j in range(2,i-1):
            if i%j==0:
                flag=False
                break
        if i==1:
            flag=false
        if flag:
            s=s+1
    return s
```

2.企业发放的奖金根据利润提成。利润低于或等于10万元时，奖金可提10%；
利润高于10万元，低于或等于20万元时，低于10万元的部分按10%提成，高于10万元的部分，可提成7.5%；
20万到40万之间时，高于20万元的部分，可提成5%；
40万到60万之间时高于40万元的部分，可提成3%；
60万到100万之间时，高于60万元的部分，可提成1.5%；
高于100万元时，超过100万元的部分按1%提成，编写函数，实现：
输入当月利润，输出应发放奖金总数。

函数输入：公司利润，函数输出：应发的奖金
(先将下面三条语句复制到答题框)
def fact(x):
# 在下方填充代码，注释掉pass
pass\

你的答案：

```
def fact(x):
# 在下方填充代码，注释掉pass
# pass\
    if x<=10:
        return (x*0.1)
    elif x<=20:
        return (10*0.1+(x-10)*0.075)
    elif x<=40:
        return (10*0.1+10*0.075+(x-20)*0.05)
    elif x<=60:
        return (10*0.1+10*0.075+20*0.05+(x-40)*0.03)
    elif x<=100:
        return (10*0.1+10*0.075+20*0.05+20*0.03+(x-60)*0.015)
    else:
```

返回

第三次作业内容

0 / 1

编程题

1. # 现有一个地图，由横线和竖线组成（参考围棋棋盘），且每个点（点为横线与竖线的交点）种有胡萝卜，且胡萝卜有着不同的重量。
# 在地图上，有一只兔子，每次行走只能沿线向右或者向上移动到临近点，并吃掉该点的胡萝卜，并计算累积的胡萝卜的重量。
# 试补充fact函数，计算出兔子从地图的起点(地图的左下角)到终点(地图右上角）的所需要吃的胡萝卜的最小重量的值为多少，并用return返回结果。
def fact(inputmap):
# 在下方填充代码，注释掉pass.
# 示例：参数inuputmap=[[1, 3, 4],
# [2, 1, 2],
# [4, 3, 1]]
# 其中数组的下标为地图的坐标，数组的值为地图上该坐标点的胡萝卜重量
# 现在要求的是从兔子从map[0][0]走到map[2][2]所需要吃的胡萝卜的最小重量的值。
# 求得路径应为map[0][0]-> map[1][0]-> map[1][1]-> map[1][2]-> map[2][2]
# 最小重量的值为1+2+1+2+1=7
pass
# 测试示例
map = [[1, 3, 4], [2, 1, 2], [4, 3, 1]]
print(fact(map)) # 输出为7

你的答案：

```
    # 计算第一行和第一列的累积胡萝卜重量
    for i in range(cols):
        carrot_sum[0][i] = sum(inputmap[0][:i+1])
    for i in range(1, rows):
        carrot_sum[i][0] = sum(inputmap[j][0] for j in range(i+1))

    # 计算其它位置的累积胡萝卜重量
    for i in range(1, rows):
        for j in range(1, cols):
            carrot_sum[i][j] = inputmap[i][j] + min(carrot_sum[i-1][j],
carrot_sum[i][j-1])

    # 返回从起点到终点的最小累积胡萝卜重量
    return carrot_sum[-1][-1]
```

返回

第四次作业内容

0 / 3

编程题

1. # 请将"_"处替换成能实现注释中功能的代码。

```
# 示例：
# #定义列表lst，值为 1，3，4
# lst = _
# 根据注释中说明可知，此处"_"应替换成[1, 3,4],即有
# lst = [1, 3,4]

# 请注意，下面的代码中answer，是用来保存同学们补充代码之后的代码执行结果的，用来批改作业的。
# 先将下面的代码复制到答题界面
def fact():
answer = []
# 字典dictionary
# 根据{'Beijing':'010','Guangzhou':'020'} 创建字典对象dict
mydict =_
answer.append(mydict.copy())

# 添加{'Shanghai':'021'}元素
mydict_
answer.append(mydict.copy())

# 判断dict是否存在'Shenzhen'这个key，并将结果存到tag中
tag=_

return answer,tag
```

你的答案：

```
def fact():
        answer = []
        # 字典dictionary
        # 根据{'Beijing':'010','Guangzhou':'020'} 创建字典对象dict
        mydict ={'Beijing':'010','Guangzhou':'020'}
        answer.append(mydict.copy())

        # 添加{'Shanghai':'021'}元素
        mydict['Shanghai']='021'
        answer.append(mydict.copy())

        # 判断dict是否存在'Shenzhen'这个key，并将结果存到tag中
        tag='Shenzhen'in mydict.keys()
```

2.# 请将符号_处替换成能实现注释中功能的代码。

示例：
#定义列表lst，值为 1，3，4
lst = _
根据注释中说明可知，此处_应替换成[1, 3,4],即有
lst = [1, 3,4]

# 请注意，下面的代码中answer，是用来保存同学们补充代码之后的代码执行结果的，用来批改作业的。
# 先将下面的代码复制到答题界面(注意，示例代码的缩进可能需要自己添加)
def fact():
answer = []
# 定义列表myList，值为'Google','Microsoft','Apple'
myList = _
answer.append(myList.copy())

在后面添加元素'Facebook'
myList._
answer.append(myList.copy())

# 插入元素'Oracle'在'Google'之后（即要插入的位置的索引为1）
myList._(1, 'Oracle')
answer.append(myList.copy())

# 对myList进行排序
myList._()
answer.append(myList.copy())

# 获取myList的长度
length = _(myList)

# 切片操作
# 获取myList列表中下标为2，3的元素，存到mylist1中
mylist1 = myList_
answer.append(mylist1.copy())

# 获取myList列表中下标为1以后的元素，存到mylist2中
mylist2=myList_

answer.append(mylist2.copy())

# 获取myList列表中下标为偶数的元素，存到mylist3中
mylist3=myList_
answer.append(mylist3.copy())

# 获取myList列表中下标为奇数的元素，存到mylist4中
mylist4=myList_
answer.append(mylist4.copy())

删除尾部元素
myList._
answer.append(myList.copy())

删除索引为1的元素
myList._
answer.append(myList.copy())

return answer,length

你的答案：

```
def fact():
        answer = []
# 定义列表myList，值为'Google','Microsoft','Apple'
        myList = ['Google','Microsoft','Apple']
        answer.append(myList.copy())

        # 在后面添加元素'Facebook'
        myList.append('Facebook')
        answer.append(myList.copy())

# 插入元素'Oracle'在'Google'之后（即要插入的位置的索引为1）
        myList.insert(1, 'Oracle')
        answer.append(myList.copy())
```

3.# 请将符号_处替换成能实现注释中功能的代码。

示例：
#定义列表lst，值为 1，3，4
lst = _
根据注释中说明可知，此处_应替换成[1, 3,4],即有
lst = [1, 3,4]

# 请注意，下面的代码中answer，是用来保存同学们补充代码之后的代码执行结果的，用来批改作业的。
# 先将下面的代码复制到答题界面(注意，示例代码的缩进可能需要自己添加)
def fact():
answer = []
# 集合set
# 创建集合s1，内有元素1，2，3，4
s1 = _
answer.append(s1.copy())

```
# 创建集合s2，内有元素3，4，5，6，7
s2 =_
answer.append(s2.copy())

# 集合s1添加元素5
s1._(5)
answer.append(s1.copy())

# 集合s1删除元素4
s1._(4)
answer.append(s1.copy())

# 求出集合s1和集合s2的交集s4
s4 =_
answer.append(s4.copy())

# 求出集合s1和集合s2的并集s5
s5 =_
answer.append(s5.copy())

return answer
```

你的答案:

```
def fact():
        answer = []
# 集合set
# 创建集合s1，内有元素1，2，3，4
        s1 = {1,2,3,4}
        answer.append(s1.copy())

# 创建集合s2，内有元素3，4，5，6，7
        s2 ={3,4,5,6,7}
        answer.append(s2.copy())

# 集合s1添加元素5
        s1.add(5)
        answer.append(s1.copy())
```

返回

第五次作业内容

0 / 1

编程题

1.# 本次作业是为了帮助同学们巩固类的相关知识
# 请根据注释,注释掉下面代码中的pass，填充相应的代码，完成有关栈操作的相关函数
# 先将下面代码复制到答题框
# 最小栈类
class fact:
total = 0 # 记录构造实例的个数
# 初始化
def __init__(self):
fact.total += 1
self.stack = [] # 存储栈中的元素
self.min = None # 栈中最小的值

# 压栈
def push(self, num):
pass

# 弹栈
def pop(self):
pass

# 取栈顶的值
def top(self):
pass

# 返回栈中最小的值
def getMin(self):
pass

定义类方法getClassTotal(minStack),返回total的值
@classmethod
def getClassTotal(fact):
pass
定义静态方法getStaticTotal(), 返回total的值
@staticmethod
def getStaticTotal():
pass

你的答案：

```
class fact:
    total = 0  # 记录构造实例的个数

    def __init__(self):
        fact.total += 1
        self.stack = []
        self.min = None

    def push(self, num):
        self.stack.append(num)
        if self.min is None or num < self.min:
            self.min = num

    def pop(self):
```

返回

第六次作业内容

0 / 1

编程题

1.# 请注释掉下面代码中的pass，填充相应的代码

函数输入：文件路径（见调用示例）
函数输出：该路径下所有文件中单词出现的频率（以字典数据类型进行返回，去除掉长度不大于2的单词）
注意：1、在统计单词出现的频率时，所有的单词均应先变成小写。
2、在统计单词时，从非字母字符（包括空格）的下一个字符作为单词的字母，直到
非字母字符（包括空格）截止。
3、数据为整个email文件夹
例如： Increase volume ofEjacu1ate 分离出来的单词为increase ,volume , ofejacu ,ate

def fact(path):
# 在下方填充代码，注释掉pass
pass

# 主函数
if __name__ == “__main__ “:
# 示例:
print(fact('./email/'))

你的答案：

```
import os
import re

def fact(path):
    word_frequency = {}  # 用于存储单词频率的字典

    # 遍历指定路径下的所有文件
    for root, dirs, files in os.walk(path):
        for file in files:
            file_path = os.path.join(root, file)
            # 使用指定编码打开文件
            with open(file_path, 'r', encoding='ISO-8859-15') as f:
                content = f.read()
                # 将内容转换为小写字母
```

返回

第七次作业内容

0 / 2

编程题

1. # 请将"_"处替换成能实现注释中功能的代码。

```
# 示例：
# #定义列表lst，值为 1，3，4
# lst = _
# 根据注释中说明可知，此处"_"应替换成[1, 3,4],即有
# lst = [1, 3,4]

import numpy as np

def fact():
# 定义answer，用于检查结果是否正确,用于批改作业
answer=[]


# 创建一维的narray对象arr1，内有元素1，2，3，4，5，6，7，8，9要求使用arange()函数
arr1 = _
# 将结果添加到answer，用于检查
answer.append(arr1.copy())

# 将arr1转换成3*3的矩阵arr2
arr2 = _
# 将结果添加到answer，用于检查
answer.append(arr2.copy())

# 使用linsapce()函数，生成首位是0，末位是10，含5个数的等差数列arr3,元素类型为float
arr3 = _
# 将结果添加到answer，用于检查
answer.append(arr3.copy())

# 创建3*4的全1矩阵arrOnes，元素类型为int
arrOnes = _
# 将结果添加到answer，用于检查
answer.append(arrOnes.copy())

# 创建3*4的全0矩阵arrZeros，元素类型为int
arrZeros = _
```

```
# 将结果添加到answer，用于检查
answer.append(arrZeros.copy())

# 创建3阶单位矩阵arrUnit，元素类型为int
arrUnit = _
# 将结果添加到answer，用于检查
answer.append(arrUnit.copy())

# 创建一个3*3的矩阵matrix1，内有元素[[1,3,3],[6,5,6],[9,9,9]],元素类型为int
matrix1 = _
# 将结果添加到answer，用于检查
answer.append(matrix1.copy())

# 获取矩阵matrix1的逆为matrix2
matrix2 = _
# 将结果添加到answer，用于检查
answer.append(matrix2.copy())

# 打印矩阵matrix1中的最大值
maxOfMatrix1= _

# 打印矩阵matrix1每一列的最大值
ColumnMax= _
# 将结果添加到answer，用于检查
answer.append(ColumnMax.copy())

# 打印矩阵matrix1每一行的平均值
LineMean = _
# 将结果添加到answer，用于检查
answer.append(LineMean.copy())

# 打印矩阵matrix1每一列的方差
variance = _
# 将结果添加到answer，用于检查
answer.append(variance.copy())

# 截取矩阵matrix1的第1，2行，存到matrix3
matrix3 = _
# 将结果添加到answer，用于检查
answer.append(matrix3.copy())

# 截取矩阵matrix1的第1，2行，第2，3列,存到matrix4
matrix4 = _
# 将结果添加到answer，用于检查
answer.append(matrix4.copy())

# 截取矩阵matrix1中大于3的元素
maxList = _
# 将结果添加到answer，用于检查
```

answer.append(maxList.copy())

return answer,maxOfMatrix1

你的答案：

```
import numpy as np

def fact():
    # 定义answer，用于检查结果是否正确,用于批改作业
    answer=[]

    # 创建一维的narray对象arr1，内有元素1，2，3，4，5，6，7，8，9要求使用arange()
函数
    arr1 = np.arange(1, 10)
    # 将结果添加到answer，用于检查
    answer.append(arr1.copy())

    # 将arr1转换成3*3的矩阵arr2
    arr2 = arr1.reshape(3, 3)
```

2. # 请将"_"处替换成能实现注释中功能的代码。

示例：
#定义列表lst，值为 1，3，4
lst = _
根据注释中说明可知，此处"_"应替换成[1, 3,4],即有
lst = [1, 3,4]

import numpy as np

def fact():
# 定义answer，用于检查结果是否正确,用于批改作业
answer=[]

# 创建一维的narray对象arr1，内有元素1，2，3，4，5，6，7，8，9要求使用arange()函数
arr1 = _
# 将结果添加到answer，用于检查
answer.append(arr1.copy())

# 将arr1转换成3*3的矩阵arr2
arr2 = _
# 将结果添加到answer，用于检查
answer.append(arr2.copy())

# 使用linsapce()函数，生成首位是0，末位是10，含5个数的等差数列arr3,元素类型为float
arr3 = _
# 将结果添加到answer，用于检查
answer.append(arr3.copy())

# 创建3*4的全1矩阵arrOnes，元素类型为int
arrOnes = _

```
# 将结果添加到answer，用于检查
answer.append(arrOnes.copy())

# 创建3*4的全0矩阵arrZeros，元素类型为int
arrZeros = _
# 将结果添加到answer，用于检查
answer.append(arrZeros.copy())

# 创建3阶单位矩阵arrUnit，元素类型为int
arrUnit = _
# 将结果添加到answer，用于检查
answer.append(arrUnit.copy())

# 创建一个3*3的矩阵matrix1，内有元素[[1,3,3],[6,5,6],[9,9,9]],元素类型为int
matrix1 = _
# 将结果添加到answer，用于检查
answer.append(matrix1.copy())

# 获取矩阵matrix1的逆为matrix2
matrix2 = _
# 将结果添加到answer，用于检查
answer.append(matrix2.copy())

# 打印矩阵matrix1中的最大值
maxOfMatrix1= _

# 打印矩阵matrix1每一列的最大值
ColumnMax= _
# 将结果添加到answer，用于检查
answer.append(ColumnMax.copy())

# 打印矩阵matrix1每一行的平均值
LineMean = _
# 将结果添加到answer，用于检查
answer.append(LineMean.copy())

# 打印矩阵matrix1每一列的方差
variance = _
# 将结果添加到answer，用于检查
answer.append(variance.copy())

# 截取矩阵matrix1的第1，2行，存到matrix3
matrix3 = _
# 将结果添加到answer，用于检查
answer.append(matrix3.copy())

# 截取矩阵matrix1的第1，2行，第2，3列,存到matrix4
matrix4 = _
# 将结果添加到answer，用于检查
```

answer.append(matrix4.copy())

# 截取矩阵matrix1中大于3的元素
maxList = _
# 将结果添加到answer，用于检查
answer.append(maxList.copy())

return answer,maxOfMatrix1

你的答案:

```
import numpy as np

def fact():
    # 定义answer，用于检查结果是否正确,用于批改作业
    answer=[]

    # 创建一维的narray对象arr1，内有元素1，2，3，4，5，6，7，8，9要求使用arange()
函数
    arr1 = np.arange(1, 10)
    # 将结果添加到answer，用于检查
    answer.append(arr1.copy())

    # 将arr1转换成3*3的矩阵arr2
    arr2 = arr1.reshape(3, 3)
```

返回

第八次作业内容

0 / 3

编程题

1.# 请将"_"处替换成能实现注释中功能的代码。

```
# 示例：
# #设置列表lst中位置3的值为1
# lst_
# 根据注释中说明可知，此处"_"应替换成[3]=1,即有
# lst[3]=1
```

请先将下面的代码复制到答题框，注意添加缩进

```
# 导入相关的包
import pandas as pd
import numpy as np
from numpy import nan as NA

def fact():
answer = []

df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'], 'data1': range(7)})
df2 = pd.DataFrame({'key': ['a', 'b', 'd'], 'data2': range(3)})

# 指定key这一列取交集
dataframe1 = _

answer.append(dataframe1)

# 取df1和df2的交集
dataframe2 = _

answer.append(dataframe2)

# 取左连接，df1左连接df2
dataframe3 = _

answer.append(dataframe3)

df3 = pd.DataFrame({"lkey": ["b", "b", "a", "c", "a", "a", "b"], "data1": range(7)})
df4 = pd.DataFrame({"rkey": ["a", "b", "d"], "data2": range(3)})
```

```
# 取df3，df4的交集
dataframe4 = _

answer.append(dataframe4)

s1 = pd.Series([0, 1], index=["a", "b"])
s2 = pd.Series([2, 3, 4], index=["c", "d", "e"])
s3 = pd.Series([5, 6], index=["f", "g"])

# 将多个Series拼接成一个DataFrame,即一个Series就是DataFrame的一列数据
dataframe5 = _

answer.append(dataframe5)


df5 = pd.DataFrame({"a": [1, NA, 5, NA], "b": [NA, 2, NA, 6], "c": range(2, 18, 4)})
df6 = pd.DataFrame({"a": [5, 4, NA, 3, 7], "b": [NA, 3, 4, 6, 8]})

# 用df6的数据为df5中的数据打补丁
dataframe6 = _

answer.append(dataframe6)

data = pd.DataFrame(np.arange(6).reshape(2, 3), index=pd.Index(["上海", "北京"],
name="省份"),
columns=pd.Index([2011, 2012, 2013], name="年份"))
# 将data的列所引转换到行索引
result1 = _

answer.append(result1)

# 将result1的行索引转化为列索引
result2 = _

answer.append(result2)

# 将result1的行索引转化为列索引，指定要转化为层次化索引的名称为"省份"
result3 = _

answer.append(result3)

data1 = pd.DataFrame({"k1": ["one"] * 3 + ["two"] * 4, "k2": [1, 1, 2, 3, 3, 4, 4]})
# 使用DataFrame的内置函数去除重复数据，默认保留第一次出现的值
result4 = _

answer.append(result4)

return answer
```

你的答案：

```
        data = pd.DataFrame(np.arange(6).reshape(2, 3), index=pd.Index(["上海",
"北京"], name="省份"),
                            columns=pd.Index([2011, 2012, 2013], name="年份"))
# 将data的列所引转换到行索引
        result1 = data.stack()

        answer.append(result1)

# 将result1的行索引转化为列索引
        result2 = result1.unstack()

        answer.append(result2)

# 将result1的行索引转化为列索引，指定要转化为层次化索引的名称为"省份"
```

2.# 请将"_"处替换成能实现注释中功能的代码。

示例:
#设置列表lst中位置3的值为1
lst_
根据注释中说明可知，此处"_"应替换成[3]=1,即有
lst[3]=1

请先将下面的代码复制到答题框（注意添加缩进）：

```
# 从pandas库导入Series,DataFrame
from pandas import Series, DataFrame

def fact():

answer = []
# 创建字典data
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
'year': [2000, 2001, 2002, 2001, 2002],
'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}

# 创建列表col，index1
col = ['year', 'state', 'pop', 'debt']
index1 = ['one', 'two', 'three', 'four', 'five']

# 利用data创建DataFrame对象frame1,并指定该对象列为col，索引为index1
frame1 = _
answer.append(frame1.copy())

# 排序
# 根据索引，对frame1进行降序排序，并指定轴为1
frame2 = frame1._
answer.append(frame2.copy())

# 根据值，对frame1的year列进行排序(升序）并打印
frame3 = frame1._
answer.append(frame3.copy())

# 处理缺失数据
# 对于frame1，只要有某行有NaN就全部删除
```

```
frame4 = frame1._
answer.append(frame4.copy())

# 对于frame1，某行全部为NaN才删除
frame5 = frame1._
answer.append(frame5.copy())

# 填充缺失数据
# 对于frame1，将元素为NaN替换成0
frame6 = frame1._
answer.append(frame6.copy())

return answer
```

你的答案：

```
from pandas import Series, DataFrame

def fact():

        answer = []
# 创建字典data
        data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}

# 创建列表col，index1
        col = ['year', 'state', 'pop', 'debt']
        index1 = ['one', 'two', 'three', 'four', 'five']
```

3.# 请将"_"处替换成能实现注释中功能的代码。

```
# 示例：
# #设置列表lst中位置3的值为1
# lst_
# 根据注释中说明可知，此处"_"应替换成[3]=1,即有
# lst[3]=1

请先将下面的代码复制到答题框（注意添加缩进）：
# 从pandas库导入Series,DataFrame
from pandas import Series, DataFrame

def fact():
answer =[]
# 创建列表lst
lst = [4, 7, -5, 3]

# 使用列表list生成Series对象obj
obj =_
answer.append(obj.copy())

# 创建数组index
index1 = ['d', 'b', 'a', 'c']
```

```
# 创建数据为lst，索引为index1的Series对象obj2
obj2=_
answer.append(obj2.copy())

# 将obj2中索引值为d对应的值赋值为6
obj2_
answer.append(obj2.copy())

# 将obj2中索引值为d对应的值存储到ans1中
ans1_
answer.append(ans1)

# 从obj2找出大于0的元素并存储到ans2中
ans2_
answer.append(ans2)

# 创建字典sdata
sdata = {'Ohio': 45000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

# 利用sdata生成Series对象obj3
obj3= _
answer.append(obj3.copy())

# 创建列表states
states = ['California', 'Ohio', 'Oregon', 'Texas']

# 创建数据为sdata，索引为states的Series对象obj4
obj4= _
answer.append(obj4.copy())

# 将obj3和obj4进行相加，相同索引部分相加，存储到obj5
obj5 = _
answer.append(obj5.copy())

# 指定obj4的名字为population
obj4_
answer.append(obj4.copy())

# 指定obj4的索引的名字为state
obj4_
answer.append(obj4.copy())
return answer
```

你的答案：

```
# 使用列表lst生成Series对象obj
        obj = Series(lst)
        answer.append(obj.copy())

# 创建数组index
        index1 = ['d', 'b', 'a', 'c']

# 创建数据为lst，索引为index1的Series对象obj2
        obj2 = Series(lst, index=index1)
        answer.append(obj2.copy())

# 将obj2中索引值为d对应的值赋值为6
        obj2['d'] = 6
        answer.append(obj2.copy())
```

返回

作业三

```
def fact(inputmap):
    # 获取地图的行数和列数
    rows = len(inputmap)
    cols = len(inputmap[0])

    # 创建一个二维数组用于记录累积胡萝卜重量
    carrot_sum = [[0] * cols for _ in range(rows)]

    # 计算第一行和第一列的累积胡萝卜重量
    for i in range(cols):
        carrot_sum[0][i] = sum(inputmap[0][:i+1])
    for i in range(1, rows):
        carrot_sum[i][0] = sum(inputmap[j][0] for j in range(i+1))

    # 计算其它位置的累积胡萝卜重量
    for i in range(1, rows):
        for j in range(1, cols):
            carrot_sum[i][j] = inputmap[i][j] + min(carrot_sum[i-1][j], carrot_sum[i][j-1])

    # 返回从起点到终点的最小累积胡萝卜重量
    return carrot_sum[-1][-1]
```

作业四

```
def fact():
        answer = []
        # 字典dictionary
        # 根据{'Beijing':'010','Guangzhou':'020'} 创建字典对象dict
        mydict ={'Beijing':'010','Guangzhou':'020'}
        answer.append(mydict.copy())

        # 添加{'Shanghai':'021'}元素
        mydict['Shanghai']='021'
        answer.append(mydict.copy())

        # 判断dict是否存在'Shenzhen'这个key，并将结果存到tag中
        tag='Shenzhen'in mydict.keys()

        return answer,tag

def fact():
        answer = []
# 定义列表myList，值为'Google','Microsoft','Apple'
        myList = ['Google','Microsoft','Apple']
        answer.append(myList.copy())

        # 在后面添加元素'Facebook'
        myList.append('Facebook')
        answer.append(myList.copy())

# 插入元素'Oracle'在'Google'之后（即要插入的位置的索引为1）
        myList.insert(1, 'Oracle')
        answer.append(myList.copy())

# 对myList进行排序
        myList.sort()
        answer.append(myList.copy())
```

```
# 获取myList的长度
    length = len(myList)

# 切片操作
# 获取myList列表中下标为2，3的元素，存到mylist1中
    mylist1 = myList[2:4]
    answer.append(mylist1.copy())

# 获取myList列表中下标为1以后的元素，存到mylist2中
    mylist2=myList[1:]
    answer.append(mylist2.copy())

# 获取myList列表中下标为偶数的元素，存到mylist3中
    mylist3=myList[::2]
    answer.append(mylist3.copy())

# 获取myList列表中下标为奇数的元素，存到mylist4中
    mylist4=myList[1::2]
    answer.append(mylist4.copy())

# 删除尾部元素
    myList.pop()
    answer.append(myList.copy())

# 删除索引为1的元素
    myList.pop(1)
    answer.append(myList.copy())

    return answer,length

def fact():
    answer = []
# 集合set
# 创建集合s1，内有元素1，2，3，4
    s1 = {1,2,3,4}
    answer.append(s1.copy())

# 创建集合s2，内有元素3，4，5，6，7
    s2 ={3,4,5,6,7}
    answer.append(s2.copy())

# 集合s1添加元素5
    s1.add(5)
    answer.append(s1.copy())

# 集合s1删除元素4
    s1.remove(4)
    answer.append(s1.copy())

# 求出集合s1和集合s2的交集s4
    s4 =s1.intersection(s2)
    answer.append(s4.copy())

# 求出集合s1和集合s2的并集s5
    s5 =s1.union(s2)
    answer.append(s5.copy())

    return answer
```

作业五：

```
class fact:
    total = 0  # 记录构造实例的个数

    def __init__(self):
        fact.total += 1
        self.stack = []
        self.min = None

    def push(self, num):
        self.stack.append(num)
        if self.min is None or num < self.min:
            self.min = num

    def pop(self):
        if len(self.stack) == 0:
            return None
        else:
            num = self.stack.pop()
            if num == self.min:
                self.min = min(self.stack) if len(self.stack) > 0 else None
            return num

    def top(self):
        return self.stack[-1] if len(self.stack) > 0 else None

    def getMin(self):
        return self.min

    @classmethod
    def getClassTotal(cls):
        return cls.total

    @staticmethod
    def getStaticTotal():
        return fact.total
```

作业六：

```
import os
import re

def fact(path):
    word_frequency = {}  # 用于存储单词频率的字典

    # 遍历指定路径下的所有文件
    for root, dirs, files in os.walk(path):
        for file in files:
            file_path = os.path.join(root, file)
            # 使用指定编码打开文件
            with open(file_path, 'r', encoding='ISO-8859-15') as f:
                content = f.read()
                # 将内容转换为小写字母
                content = content.lower()
                # 使用正则表达式匹配出所有单词
                words = re.findall(r'\b[a-z]{3,}\b', content)
                # 统计单词频率
                for word in words:
                    if len(word) > 2:  # 排除长度不大于2的单词
                        if word not in word_frequency:
                            word_frequency[word] = 1
                        else:
```

```
                    word_frequency[word] += 1

    return word_frequency

# 主函数
if __name__ == "__main__":
    # 示例
    print(fact('C:\\Users\\Leoti\\Documents\\Tencent Files\\3107362992\\FileRecv\\email 2\\email'))
```

作业七:

```
import numpy as np

def fact():
    # 定义answer，用于检查结果是否正确,用于批改作业
    answer=[]

    # 创建一维的narray对象arr1，内有元素1，2，3，4，5，6，7，8，9要求使用arange()函数
    arr1 = np.arange(1, 10)
    # 将结果添加到answer，用于检查
    answer.append(arr1.copy())

    # 将arr1转换成3*3的矩阵arr2
    arr2 = arr1.reshape(3, 3)
    # 将结果添加到answer，用于检查
    answer.append(arr2.copy())

    # 使用linspace()函数，生成首位是0，末位是10，含5个数的等差数列arr3,元素类型为float
    arr3 = np.linspace(0, 10, 5)
    # 将结果添加到answer，用于检查
    answer.append(arr3.copy())

    # 创建3*4的全1矩阵arrOnes，元素类型为int
    arrOnes = np.ones((3, 4), dtype=int)
    # 将结果添加到answer，用于检查
    answer.append(arrOnes.copy())

    # 创建3*4的全0矩阵arrZeros，元素类型为int
    arrZeros = np.zeros((3, 4), dtype=int)
    # 将结果添加到answer，用于检查
    answer.append(arrZeros.copy())

    # 创建3阶单位矩阵arrUnit，元素类型为int
    arrUnit = np.eye(3, dtype=int)
    # 将结果添加到answer，用于检查
    answer.append(arrUnit.copy())

    # 创建一个3*3的矩阵matrix1，内有元素[[1,3,3],[6,5,6],[9,9,9]],元素类型为int
    matrix1 = np.array([[1, 3, 3], [6, 5, 6], [9, 9, 9]], dtype=int)
    # 将结果添加到answer，用于检查
    answer.append(matrix1.copy())

    # 获取矩阵matrix1的逆为matrix2
    matrix2 = np.linalg.inv(matrix1)
    # 将结果添加到answer，用于检查
    answer.append(matrix2.copy())

    # 打印矩阵matrix1中的最大值
    maxOfMatrix1 = np.max(matrix1)
```

```
    # 打印矩阵matrix1每一列的最大值
    ColumnMax = np.max(matrix1, axis=0)
    # 将结果添加到answer，用于检查
    answer.append(ColumnMax.copy())

    # 打印矩阵matrix1每一行的平均值
    LineMean = np.mean(matrix1, axis=1)
    # 将结果添加到answer，用于检查
    answer.append(LineMean.copy())

    # 打印矩阵matrix1每一列的方差
    variance = np.var(matrix1, axis=0)
    # 将结果添加到answer，用于检查
    answer.append(variance.copy())

    #截取矩阵matrix1的第1，2行，存到matrix3
    matrix3 = matrix1[:2, :]
    # 将结果添加到answer，用于检查
    answer.append(matrix3.copy())

    # 截取矩阵matrix1的第1，2行，第2，3列,存到matrix4
    matrix4 = matrix1[:2, 1:3]
    # 将结果添加到answer，用于检查
    answer.append(matrix4.copy())

    # 截取矩阵matrix1中大于3的元素
    maxList = matrix1[matrix1 > 3]
    # 将结果添加到answer，用于检查
    answer.append(maxList.copy())

    return answer, maxOfMatrix1



import numpy as np

def fact():
    # 定义answer，用于检查结果是否正确,用于批改作业
    answer=[]

    # 创建一维的narray对象arr1，内有元素1，2，3，4，5，6，7，8，9要求使用arange()函数
    arr1 = np.arange(1, 10)
    # 将结果添加到answer，用于检查
    answer.append(arr1.copy())

    # 将arr1转换成3*3的矩阵arr2
    arr2 = arr1.reshape(3, 3)
    # 将结果添加到answer，用于检查
    answer.append(arr2.copy())

    # 使用linspace()函数，生成首位是0，末位是10，含5个数的等差数列arr3,元素类型为float
    arr3 = np.linspace(0, 10, 5)
    # 将结果添加到answer，用于检查
    answer.append(arr3.copy())

    # 创建3*4的全1矩阵arrOnes，元素类型为int
    arrOnes = np.ones((3, 4), dtype=int)
    # 将结果添加到answer，用于检查
    answer.append(arrOnes.copy())

    #       3*4       0       arrZeros                    int
```

```
    创建     的全 矩阵          ，元素类型为
    arrZeros = np.zeros((3, 4), dtype=int)
    # 将结果添加到answer，用于检查
    answer.append(arrZeros.copy())

    # 创建3阶单位矩阵arrUnit，元素类型为int
    arrUnit = np.eye(3, dtype=int)
    # 将结果添加到answer，用于检查
    answer.append(arrUnit.copy())

    # 创建一个3*3的矩阵matrix1，内有元素[[1,3,3],[6,5,6],[9,9,9]],元素类型为int
    matrix1 = np.array([[1, 3, 3], [6, 5, 6], [9, 9, 9]], dtype=int)
    # 将结果添加到answer，用于检查
    answer.append(matrix1.copy())

    # 获取矩阵matrix1的逆为matrix2
    matrix2 = np.linalg.inv(matrix1)
    # 将结果添加到answer，用于检查
    answer.append(matrix2.copy())

    # 打印矩阵matrix1中的最大值
    maxOfMatrix1 = np.max(matrix1)

    # 打印矩阵matrix1每一列的最大值
    ColumnMax = np.max(matrix1, axis=0)
    # 将结果添加到answer，用于检查
    answer.append(ColumnMax.copy())

    # 打印矩阵matrix1每一行的平均值
    LineMean = np.mean(matrix1, axis=1)
    # 将结果添加到answer，用于检查
    answer.append(LineMean.copy())

    # 打印矩阵matrix1每一列的方差
    variance = np.var(matrix1, axis=0)
    # 将结果添加到answer，用于检查
    answer.append(variance.copy())

    #截取矩阵matrix1的第1，2行，存到matrix3
    matrix3 = matrix1[:2, :]
    # 将结果添加到answer，用于检查
    answer.append(matrix3.copy())

    # 截取矩阵matrix1的第1，2行，第2，3列,存到matrix4
    matrix4 = matrix1[:2, 1:3]
    # 将结果添加到answer，用于检查
    answer.append(matrix4.copy())

    # 截取矩阵matrix1中大于3的元素
    maxList = matrix1[matrix1 > 3]
    # 将结果添加到answer，用于检查
    answer.append(maxList.copy())

    return answer, maxOfMatrix1
```

作业八：

```
import pandas as pd
```

```
import numpy as np
from numpy import nan as NA

def fact():
        answer = []
        df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'], 'data1': range(7)})
        df2 = pd.DataFrame({'key': ['a', 'b', 'd'], 'data2': range(3)})

# 指定key这一列取交集
        dataframe1 = pd.merge(df1, df2, on='key', how='inner')

        answer.append(dataframe1)

# 取df1和df2的交集
        dataframe2 = pd.merge(df1, df2, how='inner')

        answer.append(dataframe2)

# 取左连接，df1左连接df2
        dataframe3 = pd.merge(df1, df2, on='key', how='left')

        answer.append(dataframe3)

        df3 = pd.DataFrame({"lkey": ["b", "b", "a", "c", "a", "a", "b"], "data1": range(7)})
        df4 = pd.DataFrame({"rkey": ["a", "b", "d"], "data2": range(3)})

# 取df3，df4的交集
        dataframe4 = pd.merge(df3, df4, left_on='lkey', right_on='rkey')

        answer.append(dataframe4)

        s1 = pd.Series([0, 1], index=["a", "b"])
        s2 = pd.Series([2, 3, 4], index=["c", "d", "e"])
        s3 = pd.Series([5, 6], index=["f", "g"])

# 将多个Series拼接成一个DataFrame,即一个Series就是DataFrame的一列数据
        dataframe5 = pd.concat([s1, s2, s3], axis=1)

        answer.append(dataframe5)


        df5 = pd.DataFrame({"a": [1, NA, 5, NA], "b": [NA, 2, NA, 6], "c": range(2, 18, 4)})
        df6 = pd.DataFrame({"a": [5, 4, NA, 3, 7], "b": [NA, 3, 4, 6, 8]})

# 用df6的数据为df5中的数据打补丁
        dataframe6 = df5.combine_first(df6)

        answer.append(dataframe6)

        data = pd.DataFrame(np.arange(6).reshape(2, 3), index=pd.Index(["上海", "北京"], name="省份"),
                            columns=pd.Index([2011, 2012, 2013], name="年份"))
# 将data的列所引转换到行索引
        result1 = data.stack()

        answer.append(result1)

# 将result1的行索引转化为列索引
        result2 = result1.unstack()

        answer.append(result2)
```

```
# 将result1的行索引转化为列索引，指定要转化为层次化索引的名称为"省份"
    result3 = result1.unstack(level="省份")

    answer.append(result3)

    data1 = pd.DataFrame({"k1": ["one"] * 3 + ["two"] * 4, "k2": [1, 1, 2, 3, 3, 4, 4]})
# 使用DataFrame的内置函数去除重复数据，默认保留第一次出现的值
    result4 = data1.drop_duplicates()

    answer.append(result4)

    return answer




from pandas import Series, DataFrame

def fact():

    answer = []
# 创建字典data
    data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
    'year': [2000, 2001, 2002, 2001, 2002],
    'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}

# 创建列表col，index1
    col = ['year', 'state', 'pop', 'debt']
    index1 = ['one', 'two', 'three', 'four', 'five']

# 利用data创建DataFrame对象frame1,并指定该对象列为col，索引为index1
    frame1 = DataFrame(data, columns=col, index=index1)
    answer.append(frame1.copy())

# 排序
# 根据索引，对frame1进行降序排序，并指定轴为1
    frame2 = frame1.sort_index(axis=1, ascending=False)
    answer.append(frame2.copy())

# 根据值，对frame1的year列进行排序(升序)，并打印
    frame3 = frame1.sort_values(by='year')
    answer.append(frame3.copy())

# 处理缺失数据
# 对于frame1，只要有某行有NaN就全部删除
    frame4 = frame1.dropna()
    answer.append(frame4.copy())

# 对于frame1，某行全部为NaN才删除
    frame5 = frame1.dropna(how='all')
    answer.append(frame5.copy())

# 填充缺失数据
# 对于frame1，将元素为NaN替换成0
    frame6 = frame1.fillna(0)
    answer.append(frame6.copy())

    return answer
```

```
from pandas import Series, DataFrame

def fact():
    answer =[]
# 创建列表lst
    lst = [4, 7, -5, 3]
# 使用列表list生成Series对象obj
    obj = Series(lst)
    answer.append(obj.copy())

# 创建数组index
    index1 = ['d', 'b', 'a', 'c']

# 创建数据为lst，索引为index1的Series对象obj2
    obj2 = Series(lst, index=index1)
    answer.append(obj2.copy())

# 将obj2中索引值为d对应的值赋值为6
    obj2['d'] = 6
    answer.append(obj2.copy())

# 将obj2中索引值为d对应的值存储到ans1中
    ans1 = obj2['d']
    answer.append(ans1)

# 从obj2找出大于0的元素并存储到ans2中
    ans2 = obj2[obj2 > 0]
    answer.append(ans2)

# 创建字典sdata
    sdata = {'Ohio': 45000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

# 利用sdata生成Series对象obj3
    obj3 = Series(sdata)
    answer.append(obj3.copy())

# 创建列表states
    states = ['California', 'Ohio', 'Oregon', 'Texas']

# 创建数据为sdata，索引为states的Series对象obj4
    obj4 = Series(sdata, index=states)
    answer.append(obj4.copy())

# 将obj3和obj4进行相加，相同索引部分相加，存储到obj5
    obj5 = obj3 + obj4
    answer.append(obj5.copy())

# 指定obj4的名字为population
    obj4.name = 'population'
    answer.append(obj4.copy())

# 指定obj4的索引的名字为state
    obj4.index.name = 'state'
    answer.append(obj4.copy())
    return answer
```