一、Python

1. 写一个动物类（Animal），初始化函数输入名称（name），分类（category），类型均为字符串，再写一个狗类（Dog）继承动物类，初始化函数输入名称（name），分类（category）和颜色（color）。

```
class Animal():
    def __init__(self, name, category):
        self.name = name
        self.category = category

class Dog(Animal):
    def __init__(self, name, category, color):
        super().__init__(name, category)
        self.color = color
```

2. DataFrame

```
#%%
import string
import random
import numpy as np
import pandas as pd
#%% md
# 随机生成一些名字和分数
#%%
name = set()
while len(name) < 100:
    name.add(''.join(random.choice(string.ascii_lowercase) for _ in range(
name = list(name)

df_score = pd.DataFrame({'name': name, 'score': np.random.randint(80, 100,
df_score.head()
#%% md
# 给随机名字分配班级
#%%
classes = ['A', 'B', 'C']
df_class = pd.DataFrame({'name': name, 'class': [random.choice(classes) fo
df_class = df_class.sample(frac=1).reset_index(drop=True)
df_class.head()

##%% md
# 题目 1： 按照名字合并分数和班级
##%%
df_all = pd.merge(df_score, df_class, on='name')
print(df_all)
```

```
##%% md

# 题目 2： 取出 A 班的成绩表，按照分数降序排序
##%%
df_A_score = df_all.loc[df_all['class']=='A']
df_A_score.sort_values(by='score', inplace=True, ascending=False)
print(df_A_score)
##%% md

# 题目 3： 计算 A、B、C 班的平均分
##%%
res = [] # 一次为A、B、C三班的平均值
for c in classes:
    res.append(df_all.loc[df_all['class']==c]['score'].mean())
print(res)
print('done')
##%%
```

3. 编写正则表达式，搜索一句话里的所有微信号。

比如：

对方在“团购1群”中加我好友，他的微信号是wxid_12345，昵称”xyz”，我的微信号

期望输出：['wxid_12345', 'lz1234']

加分输出：[{'start': 22, 'end': 32, 'value': 'wxid_12345', 'type': '微信号'}, {'start': 48, 'end': 54, 'value': 'lz1234', 'type': '微信号'}]

```
import re
s = '对方在“团购1群”中加我好友，他的微信号是wxid_12345，昵称”xyz”，我的微
wxh = re.compile(r'微信号')
wxid = re.compile(r'[0-9a-zA-Z_]+')
re_wxh = [i.span() for i in wxh.finditer(s)]
re_wxid = [i.span() for i in wxid.finditer(s)]

def bs(t):
    l, r = 0, len(re_wxid)
    while l < r:
        m = (l+r) >> 1
        if re_wxid[m][0] >= t:
            r = m
        else:
            l = m + 1
    return l
res = [] # 结果
for i,j in re_wxh:
```

```
        _id = bs(j)
        _id = re_wxid[_id]
        res.append({
            'start': _id[0],
            'end': _id[1],
            'value': s[_id[0]: _id[1]],
            'type': '微信号'
        })
    print(res)
```

二、算法类

4. 介绍一下BERT？BERT和BiLSTM 有什么区别？BERT 和 Attention是什么关系？解释Q，K，V分别的作用？

    1. Bert是google在近两年提出的语言模型，采用transformer的网络堆叠结构，主要利用attention机制提取文本中长依赖关系、参数量大

    2. Bert和BiLSTM不同主要有以下几点：

        - 基础网络结构不同，bert使用transformer，BiLSTM使用两个不同方向的LSTM组合。
        - Bert在空间允许的情况下可以计算任意两个词之间的以来关系，这种计算难度不随词之间的距离增加；BiLSTM难以捕捉长距离的依赖关系
        - Bert需要的显存空间比一般的BiLSTM要多，参数更多
        - Bert比一般BiLSTM要深

    3. Bert使用transformer作为基础架构，而transformer主要包括self-attention层和全连接层，其中self-attention层起主要作为，用来捕捉文本之间的依赖关系

    4. Bert中attention的计算方式

$$Q, K, V = L^Q(\hat{Vec}), L^K(\hat{Vec}), L^V(\hat{Vec})$$
$$Vec = \text{softmax}(QK)V$$

    其中Q，K，V都是通过对词向量Vec进行全连接层变化得到的，Q代表query，K代表key，V代表value，通过Q和K得到当前词和其余词之间的权值，再通过V进行加权平均得到新的词向量表达

5. 介绍卷积运算的过程，比如输入一个 3通道，尺寸为 (5, 5) 的图像，如果要输出 6 通道，尺寸为 (3, 3)，卷积层的参数应该如何设置？假设输出矩阵 A 的尺寸为 (6, 3, 3)，它的第一个数字A[0, 0, 0] 是如何计算出来的，具体到加减乘除。

    1. 在不适用padding的情况下，可以使用3*36*3的卷积核，步长设为1
    2. 假设输入矩阵为B，$K_{i,0}^{j,k}$代表第$i, 0$个卷积核下标为i, j的元素，$B^{i,j}$表示输入矩阵下标为i, j的元素

$$A_{0,0,0} = \sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3} K_{i,0}^{j,k} * B^{j,k}$$

三、编程类

6. 排好序的数组，从中找到相加等于target数字的两个数的所有可能

```
# 比如，
# 输入数组：[2, 3, 4, 7, 10, 12]
# Target数：14
# 输出： [(2, 12), (4, 10)]
# def get_all_combs(array, target)
```

```
def get_all_combs(array, target):
    al = len(array)
    def bs(t, l=0, r=al):
        while l < r:
            m = (l+r) >> 1
            if array[m] >= t:
                r = m
            else:
                l = m + 1
        return l
    res = []
    for i, t in enumerate(array):
        j = bs(target-t, i+1)
        if j!=al and array[j]==target-t:
            res.append([array[i], array[j]])
    return res
```

7. 输入一个变形数组，及某个数字，输出这个数字在变形数组中的位置

```
# def find_num(array, num)  要求算法复杂度为 log(n)
# 变形的方法是将一个排好序的数组某个位置之前的数放到数组末尾，比如
# 原始数组为：
# 2  3  6  9  10  12  24  34
# 得到以下变形数组（将2 3 6放在了数组末尾）：
# 9  10  12  24  34  2  3  6
# 比如，输入24和变形数组，输出 3
#    说明：
# 1.不需要写变形的code，输入已经是一个变形的数组了
# 2.不知道具体将多少个数放在了数组的末尾
```

```
def find_num(array, num):
    nl = len(array)
    if array[-1] < array[0]:
```

```
        l, r = 0, nl-1
        while r-l>1:
            m = (l+r) >> 1
            if array[m] > array[l]:
                l = m
            else:
                r = m
        array = array[r:] + array[:r]
        idx = r
    else:
        idx = 0

    l, r = 0, nl
    while l < r:
        m = (l+r) >> 1
        if array[m] >= num:
            r = m
        else:
            l = m + 1
    if l!=nl and array[l]==num:
        if l >= nl-idx:
            l -= nl-idx
        else:
            l += idx
        return l
    return -1
```

四、解决问题

8. 判断输入文本中的要素的角色性质，请阐述完整的解决方案，不需要实现。

```
# 可以从几个方面：数据设计，模型设计，训练方法等等方面；
# 比如：
# 2018年7月1日16时许，我的手机13911111111接到电话19622222222，对方叫强尼，
#       要求输出：
#     {
#         '嫌疑人要素': [
#             '强尼',
#             '6222333333333333',
#             '19622222222'
#         ],
#         '非嫌疑人要素': [
#             '王小强',
#             '13911111111',
#             '5555555@qq.com'
#         ]
#     }
```

1. 数据设计：
    - 对嫌疑人要素和非嫌疑人要素进行提取，把相应的要素编码为词向量。
    - 在设计三种词向量类型编码，一种嫌疑人要素类型，一种非嫌疑人类型，一种普通类型
2. 模型设计：
    - 使用Bert或者相关的LSTM模型对文本进行建模
    - 输入包括词向量、嫌疑人/非嫌疑人要素对应向量、词向量类型向量
    - 输出为角色性质，bert可以使用分类表示位，LSTM可以使用加权求和或者最后一位来进行分类
3. 训练方法：
    - 使用有监督的训练方式，可以使用交叉熵作为loss
    - 如果数据不均衡，可以通过过采样，或者差异权重loss等方式平衡样本对loss的贡献