第 3 章　获取高考原始成绩数据

——Python 的字符串、列表、for 循环、pandas 和文件

本 章 使 命

获取高考原始数据就是本章的使命。 同时,通过本章的学习,能够了解使用计算机生成模拟数据的思想和方法,并初步掌握 Python 中的字符串、列表、for 循环、pandas 和 CSV 文件的读写方法。

3.1 Question——提出问题

要解决高考平行志愿录取问题的第一项工作就是对高考原始成绩进行赋分,而历年的高考原始数据存储在各省教育厅,只有考生本人及有关投档高校才可以查阅,其他任何单位或个人都无权查询成绩信息。我们似乎可以向省教育厅申请使用高考原始数据。但由于高考数据涉及大量考生真实敏感信息,没有特殊需求,省教育厅一般不会向外提供。

我们如何用计算机模拟生成高考原始成绩数据呢?

3.2 What——探索问题本质

3.2.1 AI 助学

我的问题:高考包括哪些成绩数据?

文心一言回答:高考成绩主要包括以下几方面的成绩数据。

统一高考科目成绩:包括语文、数学、外语三科的成绩。其中,语文和数学以原始分计入总成绩,满分均为 150 分。外语满分也为 150 分,其中听力部分占 30 分(部分省份听力成绩不计入总分),笔试部分占 120 分。外语成绩通常是按照考生外语笔试成绩的 1.25 倍计入总分,并按照四舍五入的原则取整。

选择科目成绩:是指考生根据本省的高考政策,从物理、历史、化学、生物学、思想政治、地理等科目中自主选择的科目成绩。这些科目的成绩计入高考总分。具体计入方式根据各

省的高考政策而定，有的以原始分计入，有的以转换后的等级分计入，但每科满分通常为 100 分。

此外，高考还可能包括综合素质评价的成绩，这部分成绩主要考察学生的品德、思想政治素质、实践能力、创新精神等方面。不过，综合素质评价的具体实施和成绩使用方式因地区而异，需要参照当地的高考政策。

以上是关于高考成绩数据的基本构成，具体政策可能会因地区和年份的不同而有所差异。如果您需要更详细的信息，建议参考当地教育部门或相关机构发布的高考政策文件。

借助 AI 的帮助，我们对考生的高考成绩数据有了一个宏观的了解，即包括语文、数学和外语三门科目成绩以及自主选择的三门科目（即选科）成绩。这与我们对某直辖市的高考成绩的了解是一致的。

3.2.2　探索问题本质

分析某直辖市的高考原始成绩数据，至少包括学生的基本数据和 6 门成绩数据，即准考证号、考生姓名、语文成绩、数学成绩、英语成绩以及 3 门选考科目成绩。在不影响算法设计的情况下，我们将问题简化，默认每位学生高考选考科目相同，都选择了地理、历史和政治参与考试。因此，高考成绩相关数据包括考生的准考证号、考生姓名及 6 门科目成绩。

由于高考成绩真实与否并不影响高考平行志愿录取算法的设计与实现，在无法获得真实高考原始数据的情况下，我们可以用计算机模拟生成考生准考证号、姓名和 6 门科目成绩数据，这样就可以完成高考原始成绩的获取任务。

用计算机模拟的成绩数据要与真实数据具有相同的特性。高考成绩一般具有如下特征：

数学、语文、英语成绩范围为 0～150 分；其余三门选考科目成绩范围为 0～100 分。

总成绩和各科成绩均符合正态分布。图 3-1 是某直辖市某年高考成绩分布图，由图可知，真实的高考原始数据符合正态分布，即大部分考生的高考成绩在成绩平均值左右波动，成绩特别高和特别低的考生数量很少。

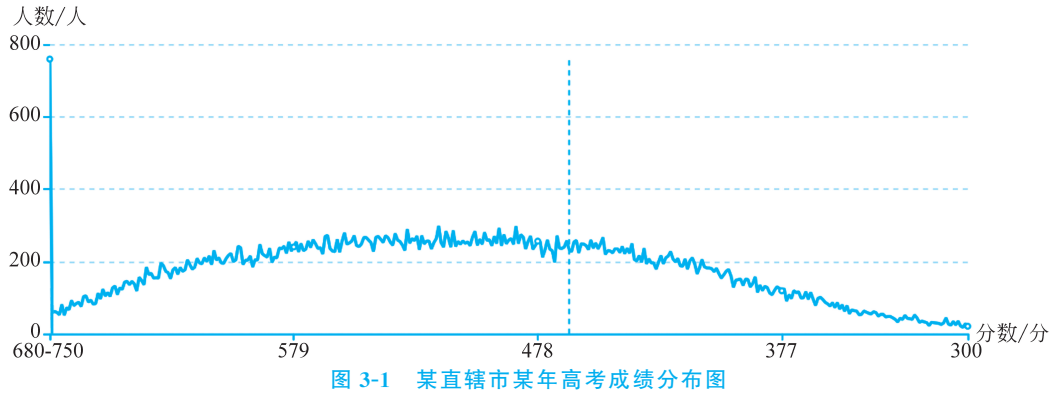


图 3-1　某直辖市某年高考成绩分布图

至此，我们要模拟的高考原始成绩数据将与表 3-1 所示的数据类似。其中，我们规定了准考证号的构成规则，即 KS＋5 位的考生序号，默认考生不超过 99 999 人。

表 3-1 考生信息结构

准考证号	姓名	语文	数学	英语	历史	地理	政治
KS00001	张珊	109	120	114	78	82	90
KS00002	李思	123	116	137	84	92	94
KS00003	王武	133	129	132	83	85	78
…	…	…	…	…	…	…	

因此，本章的任务就是用计算机模拟生成与表 3-1 类似的、符合高考成绩特征的高考原始成绩数据，即需要设计生成数据的算法并用 Python 语言实现，还要将生成的数据进行存储。

利用“分治算法”思想，可以将模拟生成高考原始成绩数据这个任务划分成以下要解决的两个子任务：

（1）用 Python 模拟高考原始成绩数据，包括准考证号、考生姓名和 6 门成绩。

（2）将模拟生成的高考原始成绩数据存储到文件中。

3.3 How——拓展求解问题必备的知识和能力

使用 Python 语言模拟生成高考原始成绩，需要解决如下问题：

（1）如何表示考生数据。

（2）如何模拟生成考生数据。

（3）如何保存模拟生成的考生数据。

3.3.1 如何用 Python 表示考生数据

考生的成绩数据包括准考证号、考生姓名和 6 门成绩。其中，准考证号、考生姓名是字符串，6 门成绩是整数。

3.3.1.1 用 String 类型表示字符串

Python 中的 String 是一种用于表示字符串的数据类型，准考证号和考生姓名等一串字符类型的信息就可以用 String 类型来表示。

1. 定义字符串

字符串是由零个或多个字符组成的序列。Python 中的字符串可以写在一对单引号、双引号或三引号中。

【例 3-1】 使用 String 类型保存字符串型数据。

```
1   s1="Hello World!"
2   s2='你好,世界!'
```

```
3   s3="It's a cat"                                    #如果把's 换成"会怎样呢?
4   s4='It"s a dog'                                    #如果把"s 换成'会怎样呢?
5   s5='同学你好,\n 欢迎学习 Python!'
6   s6= '''
7   同学你好,
8   欢迎学习 Python!
9   '''
10  print(s1)                                          #输出:Hello World!
11  print(s2)                                          #输出:你好,世界!
12  print(s3)                                          #输出:It's a cat
13  print(s4)                                          #输出:It"s a dog
14  print(s5)
15  print(s6)
```

在上面的代码中：

第 1 行定义了一个字符串变量 s1，其值为由两个双引号“" "”括起来的字符串“hello world!”。

第 2 行定义了一个字符串变量 s2，其值为由两个单引号“' '”括起来的字符串‘你好，世界!’。

第 3 行和第 4 行定义了两个字符串变量 s3 和 s4，s3 和 s4 中保存的字符串有一个共同特点，就是字符串本身已经包含一个单独的单引号或双引号，所以包含这个字符串的引号必须与字符串内的引号相区分，不能同时使用一种引号格式。即字符串内的单个引号如果是单引号，字符串外的那对引号就只能是双引号；如果字符串内的单个引号是双引号，字符串外的那对引号就只能是单引号。这样，程序才能正确识别字符串，否则程序就会报错。

第 5 行代码定义了一个字符串变量 s5，其值为一个字符串，但这个字符串内部包含一个特殊字符‘\n’，这个字符在 Python 中是转义字符，含义是换行，即输出字符串“同学你好，”之后会输出一个换行符，然后在新的一行继续输出字符串“欢迎学习 Python!”。第 3 行和第 4 行代码中的问题也可以使用转义字符“\'”代表单引号或使用“\"”代表双引号解决。即可将代码修改成 s3＝"It\"s a cat"，s4＝'It\'s a dog'，程序同样可以正确运行。什么是转义字符？Python 中还有哪些转义字符？有兴趣的读者可以自行查阅。

第 6～9 行代码定义了一个字符串变量 s6，其值用一对三引号括起来，一对三引号括起来的字符串输出时会按照多行字符串的原样输出。

程序的运行结果如图 3-2 所示。

```
Hello World!
你好，世界!
It's a cat
It"s a dog
同学你好,
欢迎学习Python!

同学你好,
欢迎学习Python!
```

图 3-2　例 3-1 运行结果

现在，我们已经可以用 String 类型表示考生数据中的准考证号和考生姓名了。

2. 访问字符串

Python 中通过“字符串变量名[下标]”的形式访问字符串中的某个字符。下标是指某个字符在字符串中的位置，既可以从前往后数每个字符的位置，也可以从后往前数每个字符的位置。注意，从前往后数时，下标从 **0** 开始数，依次增加 **1**；从后往前数时，下标从 **－1** 开始，依次减少 **1**。

例如，字符串“我喜欢学习 Python!”每个字符的下标如表 3-2 所示。

表 3-2 字符串中的下标

字 符 串	我	喜	欢	学	习	P	y	t	h	o	n	!
从前往后数，下标取值	0	1	2	3	4	5	6	7	8	9	10	11
从后往前数，下标取值	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

字符串的截取可以通过在字符串后添加“[下标开始位置：下标结束位置]”的形式截取指定范围内的字符串，得到子串。

【例 3-2】 访问字符串中的部分字符。

```
1  s='我喜欢学习 Python!'
2  print(s[3:5])                    #截取从下标 3 开始到下标 5 之前，即 4 结束
3  print(s[-7:-1])                  #截取从下标-7 开始，到-1 之前结束，即-2 结束
4  print(s[5:-1])                   #截取从下标 5 开始，到-1 之前结束，即-2 结束
5  print(s[:11])                    #截取从下标 0 开始，到 11 之前结束，即 10 结束
6  print(s[-7:])                    #截取从下标-7 开始，到最后一个字符结束
7  print(s[:])                      #截取全部字符
8  print(s[3])                      #截取一个字符，下标为 3
```

在上面的代码中：

第 1 行代码定义了一个字符串变量 s1，其值为“我喜欢学习 Python!”。

第 2 行代码打印截取的字符串子串，截取位置从下标 3 开始，至下标 5 之前结束，所以截取的是下标为 3 和 4 的字符串子串，即“学习”。

第 3 行代码打印截取的字符串子串，截取位置从下标−7 开始，至下标−1 之前结束，所以截取的是下标为−7、−6、……、−2 的字符串子串，即“Python”。

第 4 行代码打印截取的字符串子串，截取位置从下标 5 开始，至下标−1 之前结束，所以截取的是下标为 5、6、……、10(−2)的字符串子串，即“Python”。

第 5 行代码打印截取的字符串子串，下标开始位置省略，代表截取从下标 0 开始，至下标 11 之前结束，所以截取的是下标为 0、1、……、10 的字符串子串，即“我喜欢学习 Python”。

第 6 行代码打印截取的字符串子串，下标结束位置省略，代表截取到最后一个字符，并且包含最后一个字符，所以截取的是下标为−7、−6、……、−1 的字符串子串，即“Python!”。

第 7 行代码打印截取的字符串子串，截取开始位置和结束位置都省略，代表截取全部字符，即“我喜欢学习 Python!”。

第 8 行代码打印截取的字符串中的某单个字符，下标位置为 3，所以结果为“学”。

程序运行结果如图 3-3 所示。

```
学习
Python
Python
我喜欢学习Python
Python!
我喜欢学习Python!
学
```

图 3-3 例 3-2 运行结果

3. 操作字符串

1) 常用操作

Python 字符串的基本操作包括字符串拼接、字符串重复和判断是否是子串等。我们可以使用加号(+)对字符串进行拼接、使用乘号(*)对字符串进行重复、使用保留字(in 或 not in)判断一个字符串是否是另一个字符串的子串。

【例 3-3】　字符串的＋、＊、in 或 not in 操作方法示例。

```
1   str1 = 'Hello'
2   str2 = ' World'
3   str3 = str1 + str2                          #拼接字符串
4   print(str3)                                 #输出:Hello World
5   str4 = 'Hello'
6   str5 = str4 * 3                             #重复字符串
7   print(str5)                                 #输出:HelloHelloHello
8   s1 = '我喜欢学习 Python!'
9   s2 = 'Python'
10  s3 = 'C++'
11  print(s2 in s1)
12  print(s3 in s1)
13  print(s3 not in s1)
```

在上面的代码中：

第 1 行定义了一个字符串变量 str1，其值为“Helllo”。

第 2 行定义了一个字符串变量 str2，其值为“ World”。

第 3 行使用＋将字符串 str1 和 str2 拼接成一个字符串，并赋值给 str3。

第 4 行打印输出拼接后的字符串 str3。

第 5 行定义了一个字符串变量 str4，其值同样为“Helllo”。

第 6 行使用＊将 str4 字符串重复三次生成一个新的字符串，并赋值给 str5。

第 7 行打印输出重复后的字符串 str5。

第 8 行定义了一个字符串变量 s1，其值为“我喜欢学习 Python!”。

第 9 行定义了一个字符串变量 s2，其值为“Python”。

第 10 行定义了一个字符串变量 s3，其值为“C++ ”。

第 11 行使用 in 保留字判断 s2 是否是 s1 的子串，并将判断结果输出。

第 12 行使用 in 保留字判断 s3 是否是 s1 的子串，并将判断结果输出。

第 13 行使用 not in 保留字判断 s3 是否不是 s1 的子串，并将判断结果输出。

程序运行结果如图 3-4 所示。

```
Hello World
HelloHelloHello
True
False
True
```

图 3-4　例 3-3 运行结果

第 11 行、12 行、13 行的输出结果分别为 True、False、True，分别表示 s2 是 s1 的子串，s3 不是 s1 的子串，s3 不是 s1 的子串。

Python 标准库中提供了许多操作字符串的方法，比如将其他数据类型转换为字符串的 **str** 函数（方法）；填充字符串至指定长度的 **zfill** 函数；求字符串长度的 **len** 函数；将英文单词首字母转为大写的 **title** 函数；将英文字母转为大写的 **upper** 函数；将英文字母转为小写的 **lower** 函数；去除字符串两端空白的 **strip** 函数；字符串替换 **replace** 函数；字符串分割的 **split** 函数；将字符串格式化的 **format** 函数等。有需求或感兴趣的读者可以查阅官方文档获取更多用法，下面仅介绍 str 方法和 zfill 方法的具体使用方法。

2）str 方法与 type 函数

str 方法用于将各种类型的数据转换为字符串类型。例如，可以将数字转换成字符串，也可以将列表转换为字符串。Python 中提供的 **type** 函数用于查看数据的类型。

【例 3-4】 str 方法和 type 方法的使用。

```
x=123
print(x)
print(type(x))                          #输出变量 x 的数据类型
s1 = str(123)                           #将数字 123 转变为字符串"123"
print(s1)                               #输出转化后的变量 s1
print(type(s1))                         #输出变量 s1 的数据类型
ls=[1,2,3]
s2=str(ls)                              #将列表[1,2,3]转变为字符串"[1,2,3]"
print(s2)                               #输出转化后的变量 s2
print(type(s2))                         #输出变量 s2 的数据类型
```

程序运行结果如图 3-5 所示。

3）zfill 方法

zfill 方法的功能是在字符串左侧添加若干“0”字符，以达到指定长度。

具体语法：

```
str.zfill(width)
```

其中，str 为要填充的字符串，width 为指定的字符串长度。

【例 3-5】 参考表 3-1 准考证号的格式，前两位为“KS”，后 5 位为考生的序号。使用 zfill 方法生成准考证号。

```
#生成指定长度的字符串
no1 = '1'
#把字符 1 左侧拼接 4 个 0 返回;'KS'与返回字符串拼接成一个新字符串
s_no1 = 'KS'+no1.zfill(5)
print(s_no1)
```

程序运行结果如图 3-6 所示。

```
123
<class 'int'>
123
<class 'str'>
[1, 2, 3]
<class 'str'>
```

图 3-5 例 3-4 运行结果

```
KS00001
```

图 3-6 例 3-5 运行结果

关于 String 类型，可扫描二维码学习更多细节。

3.3.1.2 用列表表示考生的数据

在例 2-13 中我们已经知道可以使用列表这种数据类型来表示多项信息。那么，Python 中列表具体是什么样的数据类型？准考证号和考生姓名也可以放在列表里吗？如何定义和

访问列表？列表有哪些常用的方法呢？接下来介绍 Python 中的列表。

1. 定义列表

列表(list)是一种有序的组合数据类型，可以存放**零到多个**元素，每一元素可以是数值型、字符串型、列表以及后面要介绍的字典等任意一种数据类型。

定义列表时，所有元素都写在一对方括号“[]”中，每两个元素之间用逗号分隔。不包含任何元素的列表称为**空列表**。

【例 3-6】 定义存储一名考生数据的列表，并输出列表中的内容。

```
#定义存储一名考生数据的列表
student=['KS00001','张珊',109,120,114,78,82,90]
print(student)                                    #输出该列表
```

运行程序例 3-6，结果如图 3-7 所示。

```
['KS00001', '张珊', 109, 120, 114, 78, 82, 90]
```

图 3-7　例 3-6 运行情况

现在，我们已经可以表示一名考生的数据了。

2. 二维列表

由于列表包含的元素还可以是列表，我们可以使用列表的这一特性存储表 3-1 中的所有考生的数据。此时，列表中的每个元素都是保存某位考生数据的列表，整个列表就存储了所有考生的数据。这种列表嵌套列表的结构，称为**二维(二级)列表**。

【例 3-7】 定义二维列表存储表 3-1 中所有考生的数据，要求每个元素是每名考生的数据。

```
#定义二维列表存储表 3-1 中的考生数据，列表元素是每名考生的数据
stu1=[
        ['KS00001','张珊',109,120,114,78,82,90],        #保存第一位考生数据
        ['KS00002','李思',123,116,137,84,92,94],        #保存第二位考生数据
        ['KS00003','王武',133,129,132,83,85,78]         #保存第三位考生数据
    ]
print(stu1)
```

上面代码中的二维列表 stu1 采用的是按行存储方式，它的每个元素是一个一维列表，存储的是一名考生的准考证号、姓名和 6 门课程的成绩。

程序例 3-7 的运行结果如图 3-8 所示。

```
[['KS00001', '张珊', 109, 120, 114, 78, 82, 90], ['KS00002', '李思', 123, 116, 137, 84, 92, 94], ['KS00003', '王武', 133, 129, 132, 83, 85, 78]]
```

图 3-8　例 3-7 运行结果

现在，我们已经可以表示多名考生的数据了。

【例 3-8】 定义二维列表存储表 3-1 中的考生数据，要求每个元素是数据的一个属性。

```
#定义二维列表存储表 3-1 中的考生数据，列表元素是属性信息
stu2=[
```

```
    ['KS00001','KS00002','KS00003'],          #按顺序保存所有准考证号
    ['张珊','李思','王武'],                    #按顺序保存所有考生姓名
    [109,123,133],                            #按顺序保存语文成绩
    [120,116,129],                            #按顺序保存数学成绩
    [114,137,132],                            #按顺序保存英语成绩
    [78,84,83],                               #按顺序保存历史成绩
    [82,92,85],                               #按顺序保存地理成绩
    [90,94,78]                                #按顺序保存政治成绩
]
print(stu2)
```

上面代码中的二维列表 stu2 采用的是按列存储方式，它的每个元素仍然是列表，但存储的是考生数据中的属性，包括准考证号、姓名、语文成绩、数学成绩、英语成绩、历史成绩、地理成绩、政治成绩。

程序例 3-8 的运行结果如图 3-9 所示。

```
[['KS00001', 'KS00002', 'KS00003'], ['张珊', '李思', '王武'], [109, 123, 133], [120, 116, 129], [114, 137, 132], [78, 84, 83], [82, 92, 85], [90, 94, 78]]
```

图 3-9　例 3-8 运行结果

3. 访问列表

Python 中列表的访问（又称检索），通过“列表变量名[下标]”的形式访问列表中的某个元素。**下标**是指某个元素在列表中的位置，既可以从前往后数每个元素的位置，也可以从后往前数每个元素的位置。注意，从前往后数时，下标从 **0** 开始数，依次增加 **1**；从后往前数时，下标从 **−1** 开始，依次减少 **1**。

例如，对于 student=['KS00001','张珊',109,120,114,78,82,90]这个列表，各个元素的下标如表 3-3 所示。

表 3-3　列表 student 各元素的下标

列 表 元 素	KS00001	张珊	109	120	114	78	82	90
从前往后数，下标取值	0	1	2	3	4	5	6	7
从后往前数，下标取值	−8	−7	−6	−5	−4	−3	−2	−1

另外，还可以截取列表中的部分元素形成一个**新列表**。具体用法如下：

```
ls[下标开始位置:下标结束位置]
```

其中，ls 是定义好的列表变量名，ls[下标开始位置：下标结束位置]表示将 ls 中从下标开始位置开始，到下标结束位置之前的那一个元素（**不包括下标结束位置的元素**），取出来形成一个新的列表。下标开始位置和下标结束位置均可以省略，下标开始位置省略表示从下标为 **0** 的元素开始访问；下标结束位置省略表示访问到最后一个元素（包括最后一个元素）；两者均省略代表截取整个列表。

【例 3-9】 对列表中部分元素的访问。

```
student=['KS00001','张珊',109,120,114,78,82,90]
print(student[1:4])    #截取下标为 1,2,3 的元素形成新列表'张珊',109,120]
print(student[-3:-1]) #截取下标为-3,-2 的元素形成新列表[78,82]
```

```
print(student[3:6])    #截取下标为 3,4,5 的元素形成新列表[120,114,78]
print(student[:4])     #截取下标为 0,1,2,3 的元素形成新列表['KS00001','张珊',
                       #109,120]
print(student[-3:])    #截取下标为-3,-2,-1 的元素形成新列表[78,82,90]
print(student[:])      #取所有元素['KS00001','张珊',109,120,114,78,82,90]
print(student[3])      #取单个元素,下标为 3,返回值为单个元素 120
```

程序例 3-9 运行结果如图 3-10 所示。

```
['张珊', 109, 120]
[78, 82]
[120, 114, 78]
['KS00001', '张珊', 109, 120]
[78, 82, 90]
['KS00001', '张珊', 109, 120, 114, 78, 82, 90]
120
```

图 3-10　例 3-9 运行结果

二级列表的访问可以通过多个方括号“[]”叠加的方式依次访问每级列表中的元素。

【例 3-10】　访问二级列表中的元素。

```
#按行存储考生数据
stu1=[
    ['KS00001','张珊',109,120,114,78,82,90],        #保存第一位考生数据
    ['KS00002','李思',123,116,137,84,92,94],        #保存第二位考生数据
    ['KS00003','王武',133,129,132,83,85,78]         #保存第三位考生数据
    ]
print(stu1[0])                                       #访问第一位考生数据
print(stu1[2])                                       #访问第三位考生数据
#stu1[0]返回的是存储第一名考生数据的列表,可以继续通过[]访问其中的元素
print(stu1[0][2])                                    #访问第一名考生的语文成绩
```

程序例 3-10 运行结果如图 3-11 所示。

```
['KS00001', '张珊', 109, 120, 114, 78, 82, 90]
['KS00003', '王武', 133, 129, 132, 83, 85, 78]
109
```

图 3-11　例 3-10 运行结果

4. 列表的常用操作

1）拼接列表

通过拼接运算符即加号“+”可以将多个列表连接在一起，生成一个新列表。

【例 3-11】　列表的拼接。

```
stu1=[
      ['KS00001','张珊',109,120,114,78,82,90],      #保存第一位考生数据
      ['KS00002','李思',123,116,137,84,92,94],      #保存第二位考生数据
      ['KS00003','王武',133,129,132,83,85,78]       #保存第三位考生数据
]
stu2=[
      ['KS00004','李明',142,125,145,89,95,93],
```

```
        ['KS00005','徐晓丽',98,102,119,75,69,81]
]
stu3=['KS00006','程鑫鑫',113,129,138,76,82,88]
m_stu1 = stu1+stu2
print(m_stu1)
m_stu2 = stu1+stu3
print(m_stu2)
```

上述程序中，第 11 行代码将列表 stu1 和 stu2 拼接成一个新列表 m_stu1，第 13 行代码将列表 stu1 和 stu3 拼接成一个新列表 m_stu2。程序例 3-11 运行结果如图 3-12 所示。

```
[['KS00001', '张珊', 109, 120, 114, 78, 82, 90], ['KS00002', '李思', 123, 116, 137, 84, 92, 94], ['KS00003', '王武', 133, 129, 132, 83, 85, 78], ['KS00004',
'李明', 142, 125, 145, 89, 95, 93], ['KS00005', '徐晓丽', 98, 102, 119, 75, 69, 81]]
[['KS00001', '张珊', 109, 120, 114, 78, 82, 90], ['KS00002', '李思', 123, 116, 137, 84, 92, 94], ['KS00003', '王武', 133, 129, 132, 83, 85, 78], 'KS00006',
'程鑫鑫', 113, 129, 138, 76, 82, 88]
```

图 3-12　例 3-11 运行结果

2）计算列表长度

使用 **len()**方法可以获取一个列表中包含的元素数量，即列表的长度。

【例 3-12】 计算列表长度。

```
stu1=[
    ['KS00001','张珊',109,120,114,78,82,90],  #保存第一位考生数据
    ['KS00002','李思',123,116,137,84,92,94],  #保存第二位考生数据
    ['KS00003','王武',133,129,132,83,85,78]   #保存第三位考生数据
    ]
print('stu1 中的考生人数为:',len(stu1))
```

程序例 3-12 运行结果如图 3-13 所示。

3）获取列表中的最大元素、最小元素

使用 **max()**方法可以获取一个列表中的最大元素的值，使用 **min()**方法可以获取一个列表中最小元素的值。

【例 3-13】 按列存储多名考生的成绩信息，获取语文的最高分和外语最低分。

```
stu2=[
    ['KS00001','KS00002','KS00003'],          #按顺序保存所有准考证号
    ['张珊','李思','王武'],                    #按顺序保存所有考生姓名
    [109,123,133],                            #按顺序保存语文成绩
    [120,116,129],                            #按顺序保存数学成绩
    [114,137,132],                            #按顺序保存英语成绩
    [78,84,83],                               #按顺序保存历史成绩
    [82,92,85],                               #按顺序保存地理成绩
    [90,94,78]                                #按顺序保存政治成绩
]
print('三名考生中语文最高成绩为',max(stu2[2]))#语文成绩的最大值
print('三名考生中英语最低成绩为',min(stu2[4]))#英语成绩的最小值
```

程序例 3-13 运行结果如图 3-14 所示。

```
stu1中的考生人数为:  3
```

图 3-13　例 3-12 运行结果

```
三名同学中语文最高成绩为 133
三名同学中英语最低成绩为 114
```

图 3-14　例 3-13 运行结果

4）向列表中增加元素

（1）使用 **append()** 方法可以向列表末尾追加元素。这种方法每次可以向列表的末尾追加一个元素。

（2）使用 **insert()** 方法可以将一个元素插入列表的指定位置。insert()方法有两个参数，第一个参数用来指定插入的下标位置（下标从 0 开始计数），第二个参数是要插入的元素值。

【例 3-14】 向列表中增加元素。

```
stu1=[
      ['KS00001','张珊',109,120,114,78,82,90],    #保存第一位考生数据
      ['KS00002','李思',123,116,137,84,92,94],    #保存第二位考生数据
      ['KS00003','王武',133,129,132,83,85,78]     #保存第三位考生数据
]
ls1 = ['KS00005','李明',142,125,145,89,95,93]
ls2 = ['KS00004','徐晓丽',98,102,119,75,69,81]
stu1.append(ls1)
print(stu1)
stu1.insert(3,ls2)
print(stu1)
```

上述程序中的第 8 行代码是将列表 ls1 作为一个元素插入二维列表 stu1 的最后；第 10 行代码是在将 ls2 插入 stu1 第 4 个元素的位置。程序例 3-14 运行结果如图 3-15 所示。

```
[['KS00001', '张珊', 109, 120, 114, 78, 82, 90], ['KS00002', '李思', 123, 116, 137, 84, 92, 94], ['KS00003', '王武', 133, 129, 132, 83, 85, 78], ['KS00005', '李明', 142, 125, 145, 89, 95, 93]]
[['KS00001', '张珊', 109, 120, 114, 78, 82, 90], ['KS00002', '李思', 123, 116, 137, 84, 92, 94], ['KS00003', '王武', 133, 129, 132, 83, 85, 78], ['KS00004', '徐晓丽', 98, 102, 119, 75, 69, 81], ['KS00005', '李明', 142, 125, 145, 89, 95, 93]]
```

图 3-15　例 3-14 运行结果

请思考并试一试：如果 insert 方法中给出的第一个参数值超出列表的下标范围，会怎么样呢？

5）查询列表中的元素

使用运算符 **in** 或 **not in** 查询某元素是否存在列表中，返回结果是逻辑真（True）或逻辑假（**False**），分别表示元素在列表中或不在列表中。

【例 3-15】 判断列表中是否存在某个数据元素。

```
stu1=[
      ['KS00001','张珊',109,120,114,78,82,90],
      ['KS00002','李思',123,116,137,84,92,94],
      ['KS00003','王武',133,129,132,83,85,78],
      ['KS00004','李明',142,125,145,89,95,93],
      ['KS00005','徐晓丽',98,102,119,75,69,81]
]
print('张珊' in stu1[0])                    #查找元素张珊是否存在于列表中
```

上述程序中的第 8 行代码中，由于张珊在第一个列表元素 stu1[0] 中，因此“'张珊' in stu1[0]”返回的结果为 True。例 3-15 运行结果如图 3-16 所示。

```
True
```

图 3-16　例 3-15 运行结果

请思考并试一试：如果要判断学生“贾兰”是否在列表 stu1 中，应该如何处理？会输出

什么结果？

列表的常用操作还有很多。例如，使用 sort()方法对列表中的元素排序；使用 index()方法查找某个元素在列表中首次出现的位置；使用 count()方法统计列表中某个元素出现的次数；使用 del()方法删除列表中某个元素；使用 clear()方法清空列表；等等。此处不再一一展开，有需求和感兴趣的读者可以自行查阅相关文献。

3.3.2 如何用 Python 模拟生成考生数据

前面已经明确了要模拟的考生数据，包括准考证号、姓名和 6 科成绩等信息，也知道了一名考生的数据可以放在一个一维列表中，所有考生的数据可以放在一个二维列表中。那么，下面要解决的问题就是如何模拟生成这些考生的数据。

3.3.2.1 AI 助学

我的问题：模拟生成数据的方法有哪些？

文心一言回答：模拟生成数据的方法有很多种，以下是常见的方法。

（1）随机数生成法：通过随机数生成器生成一系列随机数，可以模拟各种分布的数据。

（2）参数建模法：根据已知的数据分布规律和参数，建立数学模型，通过模型生成符合要求的数据。

（3）机器学习方法：利用机器学习算法对大量数据进行训练和学习，然后生成新的数据。

（4）自然语言处理法：通过自然语言处理技术，从已有的文本数据中提取有用的信息，生成新的数据。

（5）图像处理法：通过图像处理技术，从已有的图像数据中提取有用的信息，生成新的数据。

（6）传感器数据模拟：利用传感器模拟数据，可以模拟各种传感器数据，如温度、湿度、压力等。

（7）蒙特卡洛模拟：通过蒙特卡洛方法，模拟各种随机事件和概率分布，生成符合要求的数据。

这些方法各有优缺点，应根据具体需求选择合适的方法。同时，也需要注意数据的准确性和可信度，以避免误导和产生不必要的损失。

在 AI 的帮助下，我们了解了用计算机模拟生成数据有很多方法。对于初学者，感觉随机数生成法比较简单，可以先选择学习和使用这个方法。

3.3.2.2 随机数生成数据的方法

Python 标准库中的 random 模块用来生成随机数，可以生成随机浮点数、整数、字符串，还可以随机选择列表中的一个元素等。Python 标准库中的函数不能直接使用，必须用 **import** 关键字导入后才能使用。

Random 模块包含了大量的方法，用于不同场景的随机数生成，有需求或感兴趣的读者可以查阅官方文档获取更多用法。下面仅介绍常用的几个方法。

1. random()方法

random.random()可以随机生成一个实数，取值范围为[0,1)。

2. uniform()方法

random.uniform(x,y)可以随机生成一个实数，取值范围为[x,y]。

3. randint(a,b)方法

random.randint(a,b)可以随机生成指定范围内的一个整数，取值范围为[a,b]。

【例 3-16】 random 常用方法示例。

```
1   import random                                  #加载 random
2   print(random.random())                         #随机产生一个[0,1)的实数
3   print(random.uniform(2,4))                     #随机产生一个[2,4]的实数
4   print(random.randint(0,100))                   #随机产生一个[0,100]的整数
```

例 3-16 运行结果如图 3-17 所示。

现在，我们已经可以随机生成一门课程的高考成绩了。

4. choice(seq)方法

random.choice(seq)可以随机选择列表或字符串中的一个元素。

【例 3-17】 从一组表示姓的字符串中随机选择一个字符串，再从一组表示名的字符串中随机选择一个字符串，组成一个人的姓名。

```
1   import random                                  #加载 random
2   #表示姓的列表
3   last_names = ['赵', '钱', '孙', '李', '周', '吴', '郑', '王', '冯', '陈', '褚',
    '卫', '东方', '沈', '韩', '杨', '欧阳', '秦', '尤', '许', '姚', '邵', '堪', '黄埔',
    '祁', '毛', '禹', '狄', '米', '贝', '明', '臧', '计', '伏', '成', '戴', '谈', '宋',
    '茅', '庞','熊', '纪', '舒', '屈', '项', '祝', '董', '梁']
4   #表示名字的列表
5   first_names = ['明', '红', '强', '芳', '建', '恺', '凯', '超', '梓涵', '玉', '宝强',
    '钗', '丽', '云', '来', '圆', '媛媛', '一', '天', '东胜', '栋', '致', '茜', '倩',
    '志摩', '国庆', '年', '泽', '熙', '那', '和', '远', '允', '妙', '家', '伟', '贤',
    '然', '笑', '冉', '乾', '坤', '琼', '潇', '雯', '瑶', '翊', '墨', '华']
6   #随机生成一个中文姓名
7   last_name = random.choice(last_names)          #在姓氏列表中随机选择一个姓氏
8   first_name = random.choice(first_names)        #在姓名列表中随机选择一个姓名
9   print(last_name+first_name)                    #将姓氏与姓名拼接成一个名字
```

运行一次程序会生成一个姓名，再运行一次程序将会随机产生另一个姓名。例 3-17 程序的一次运行结果如图 3-18 所示。

```
0.27156203755930386
3.0589665536700124
86
```

图 3-17　例 3-16 运行结果

戴强

图 3-18　例 3-17 运行结果

现在，我们已经可以随机生成一名考生的姓名了。

3.3.2.3　模拟生成多个数据

我们已经能够生成一个成绩，那么如何生成一名考生的 6 门科目的成绩呢？又如何生

成多名考生的数据呢？

事实上，这样的处理都是不断重复某一操作的过程。对于一名考生的 6 门科目的成绩，先随机生成一门科目的成绩，再随机生成第 2 门科目的成绩，重复此项操作，直到完成所有 6 门科目的成绩的生成。对于生成所有考生的数据也类似，先随机生成一个考生的数据，再生成下一个考生的数据，直到完成所有考生的数据生成。

这种重复操作的处理过程可以使用 Python 中的循环语句实现。下面介绍 Python 循环语句的使用方法。

1. range 函数

range 函数用来生成一个数据序列，其语法格式为：

```
range([beg],end,[step])
```

其中，**beg** 为起始数值，**end** 表示终止数值（注意生成数据中不包括 end），**step** 为步长（允许为负值）。如果 step 省略，则默认以 1 为步长；如果 beg 省略，则默认从 0 开始。

【例 3-18】 range 函数的功能示例。

```
print(range(0,6))                    #生成数据序列为:0,1,2,3,4,5
print(range(1,101,2))                #生成的数据序列为 100 以内的奇数,1,3,5,7,…
```

2. for 语句

for 语句是专门用于处理循环的语句，其语法格式为：

```
for 循环变量名 in 可迭代对象:
    语句序列
```

其中，可迭代对象可以理解成一个数据序列，如字符串、列表或 range 方法返回的数字序列。

for 循环执行过程： 循环变量依次取可迭代对象中的每个值，执行“语句序列”进行相应的数据处理，当没有迭代对象时循环停止。

【例 3-19】 用 for 循环生成某名考生 6 门课程的高考成绩（此处假设各门课程成绩的最高分均为 100 分）。

```
import random                                  #加载 random
Score=[]                                       #定义一个空列表,用于存放 6 科成绩
for i in range(0,6):                           #循环变量 i 从 0 开始到 5 结束
      Score.append(random.randint(0,100))      #随机生成 1 科成绩放入列表中
print(Score)
```

上述程序中的第 3 行代码和第 4 行代码是 for 循环语句，通过第 3 行代码中的循环变量 i 从 0 到 5 的 6 次取值，重复了 6 次操作，模拟生成了 6 科课程。第 4 行代码是要重复进行的操作，random.randint(0,100)是随机生成一个分数为 0～100 的成绩，然后使用列表 Score 的 append()方法，将生成的成绩追加到列表中。

例 3-19 的运行结果如图 3-19 所示。

```
[19, 42, 32, 92, 32, 66]
```

图 3-19 例 3-19 运行结果

【例 3-20】 假设准考证号是以“KS”开头、长度固定为 7 位的字符串，取值由“KS00001”依次增加，“KS00002”、“KS00003”、“KS00004”……。使用 for 循环生成 10 个准考证号。

```
1   s_no=[]                                    #定义一个空列表
2   for i in range(1,11):                      #循环变量 i 从 1 开始到 10 结束
3       num ="KS" + str(i).zfill(5)            #生成第 i 个准考证号
4       s_no.append(num)                       #将生成的准考证号添加到列表 s_no 中
5   print(s_no)
```

在程序的第 3 行代码中，首先使用 str(i)将循环变量的值转换为 String 类型；然后调用字符串的 zfill()方法，生成形如“00001”的字符串（当 i＝1 时）；最后再使用“＋”将“KS”和“00001”进行拼接，生成准考证号。例 3-20 的运行结果如图 3-20 所示。

```
['KS00001', 'KS00002', 'KS00003', 'KS00004', 'KS00005', 'KS00006', 'KS00007', 'KS00008', 'KS00009', 'KS00010']
```

图 3-20　例 3-20 运行结果

关于循环语句和 for 循环，可扫描二维码学习更多细节。

3.3.2.4　模拟生成相同的数据

细心的读者应该已经发现了一个问题，每次运行程序随机生成的数据都不一样。如果想每次运行程序都能随机生成一样的数据，我们可以通过 random 的 seed()方法，设置一个随机种子，使得每次运行程序生成的数据都一样。

【例 3-21】　随机种子的使用。

```
1   import random
2   #不加随机种子
3   print("不加随机种子生成的数据:")
4   for i in range(3):
5       score1=[]
6       #随机生成 6 个 100 以内的整数,并加入 score1 列表中
7       for j in range(6):
8           score1.append(random.randint(0,100))
9       print(score1)
10  #加随机种子
11  print("加随机种子生成的数据:")
12  for i in range(3):
13      score2=[]
14      #随机生成 6 个 100 以内的整数,并加入 score2 列表中
15      random.seed(1)
16      for j in range(6):
17          score2.append(random.randint(0,100))
18      print(score2)
```

上面代码中的 seed()是随机数生成器 random 的一种方法，用于设置生成随机数的种子。种子是一个整数。通过设置相同的种子，就可以得到相同的随机数序列。

第 3～9 行代码，没有添加随机种子，循环三次，每次随机生成的数据均不相同。

第 11～18 行代码也是循环三次，但每次生成的数据均相同，这是由于在第 15 行代码中，为每次循环都设置了同一个种子值 1，所以三次循环生成的随机数是相同的。当然也可以设置不同的种子，生成不同的随机数序列。

例 3-21 的运行结果如图 3-21 所示。

现在，我们已经可以生成多科成绩数据以及多名考生的数据了。

```
不加随机种子生成的数据:
[63, 97, 57, 60, 83, 48]
[100, 26, 12, 62, 3, 49]
[55, 77, 97, 98, 0, 89]
加随机种子生成的数据:
[17, 72, 97, 8, 32, 15]
[17, 72, 97, 8, 32, 15]
[17, 72, 97, 8, 32, 15]
```

图 3-21　例 3-21 运行结果

3.3.3　如何用 Python 保存考生数据

使用前面的方法，已经能够使用 Python 模拟生成考生的数据了。但是，我们发现每次运行程序都可以生成考生数据，但程序运行结束或关闭计算机后，必须重新运行程序才能获得考生数据。这是由于运行程序生成的数据被存储在计算机的内存中，而内存中的数据不会被永久保存，程序结束或计算机关机都会丢失。

是不是有什么方法不需要每次都去运行程序获得考生数据，而是将一次生成的考生数据永久保存起来，用于后面实现高考平行志愿录取任务呢？

事实上，我们可以将生成的考生数据存储在文件里，文件被保存在外存（硬盘、U 盘等）中，这样就可以实现这些数据的长期保存及重复利用。这就要学习将内存中的数据保存到文件中的方法。下面介绍将数据存储到文件中涉及的几个概念，以及使用 Python 对 CVS 文件进行操作的相关方法。

3.3.3.1　关于数据的几个概念

1. 数据元素

数据元素也称为结点或记录。一个数据元素可由若干数据项（属性）组成。例如，表 3-1 所示的考生数据，每名考生的数据就是一个数据元素，包括准考证号、姓名和语文、数学、英语、历史、地理、政治 6 科成绩数据。

2. 一维数据

一维数据是指数据元素由一个因素即可确定。例如，一名考生的数据就是一个一维数据，如表 3-4 所示。表中的阴影部分是一维数据，可以看出，一个数据项只由该数据项所在位置这一个因素就可以确定。例如，第 1 位的数据是准考证号，第 2 位的数据是姓名，第 8 位的数据是政治成绩。我们知道，一维数据可以使用 Python 中的列表来表示。

表 3-4　一维数据示例

含义	准考证号	姓名	语文	数学	英语	历史	地理	政治
位置	1	2	3	4	5	6	7	8
数据	KS00001	张珊	109	120	114	78	82	90

3. 二维数据

二维数据由多个一维数据组成，二维数据中的某项数据需要由两个因素共同决定。如

表 3-5 所示，多名考生的数据就是二维数据。表 3-5 中的阴影部分是二维数据，可以看出，一项数据需要由该数据项所在的行号和所在的列号两个因素才能确定。例如，116 这个成绩，需要由该数据所在的行号 2 和所在的列号 4 这两个因素才能确定。我们也已经知道，二维数据可以使用 Python 中的二维列表来表示。

表 3-5　二维数据示例

含义	准考证号	姓名	语文	数学	英语	历史	地理	政治
列号 / 行号	1	2	3	4	5	6	7	8
1	KS00001	张珊	109	120	114	78	82	90
2	KS00002	李思	123	116	137	84	92	94
3	KS00003	王武	133	129	132	83	85	78
…	…	…	…	…	…	…	…	…

4. 计算机文件

计算机文件（以下简称文件），是一种存储在计算机硬盘、软盘、光盘等存储介质上的数据的集合，内容可以是文本、图像、音频、视频、程序等。一个文件的名字通常由一个文件名和一个文件扩展名构成。文件扩展名来标识它所属的类型，便于识别和管理。

5. CSV 文件

CSV（Comma-Separated Values）是一种国际通用的数据存储格式文件，文件扩展名为 .csv，可以使用记事本、Excel 等软件打开 CSV 文件。CSV 文件中每行对应一个一维数据，数据各数据项之间默认用英文逗号分隔。如果有缺失数据，也要保留逗号，使得各项数据都有它的位置。CSV 文件中的多行一维数据就够构成了一个二维数据。CSV 文件的第一行可以是列标题，也可以直接存储数据（即没有列标题）。

图 3-22 示意了用 Excel 打开的带列标题的 CSV 文件和不带列标题的 CSV 文件。

10001	140	145	138	95	96	89
10002	130	139	146	91	95	80
10003	150	148	148	52	77	62
10004	142	143	147	41	87	79
10005	135	139	138	53	92	97
10006	122	149	150	95	52	76
10007	132	147	141	73	69	87
10008	143	136	136	70	81	85
10009	148	149	129	100	44	62
10010	150	143	135	42	90	68
10011	134	140	147	94	49	70
10012	120	127	143	98	85	90

(a) 不带列标题CSV文件

准考证号	语文	数学	英语	历史	政治	地理
10001	140	145	138	95	96	89
10002	130	139	146	91	95	80
10003	150	148	148	52	77	62
10004	142	143	147	41	87	79
10005	135	139	138	53	92	97
10006	122	149	150	95	52	76
10007	132	147	141	73	69	87
10008	143	136	136	70	81	85
10009	148	149	129	100	44	62
10010	150	143	135	42	90	68
10011	134	140	147	94	49	70

(b) 带列标题CSV文件

图 3-22　带列标题 CSV 文件和不带列标题 CSV 文件

6. 类与对象

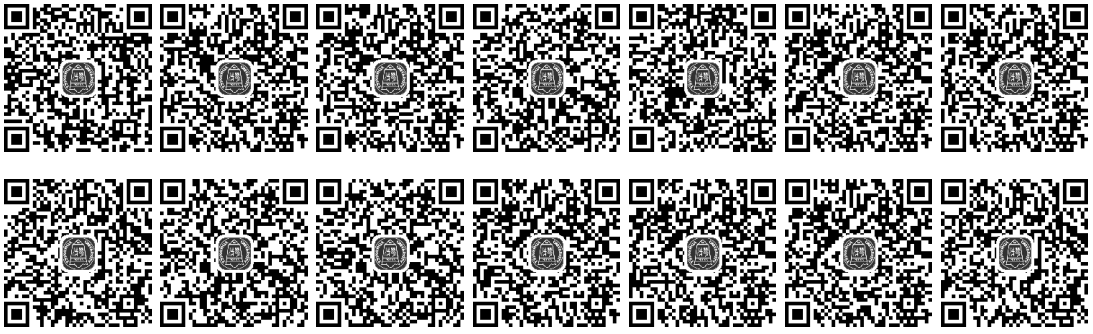
类是对同一类对象的一个抽象，而**对象**则是类中一个具体的实体。例如，圆是一个类，而半径 5cm，圆心（0，0）的圆就是一个具体的对象。OOP（Object Oriented Programming）就是通过抽象对象特征来编写程序的一种方法，称为面向对象程序设计方法。Python 中的库都采用了 OOP。

类通常规定了一个对象可以包括哪些数据，以及对这些数据进行哪些处理。类中的数

据用来表示对象的静态特征——属性，类中进行的处理用来表示对象的动态特征——行为。类中要进行的处理在很多场合被称为**方法**。一个类可以有（创建）多个对象。例如，计算圆的面积和周长的问题，可以抽象出一个关于圆的类，类中的数据部分包括圆心的位置和半径；类中的处理部分包括输入圆心、输入半径、计算面积、计算周长和输出结果等。

事实上，我们已经接触过很多次类、对象和方法。例如，例 3-21 中的 score1 是一个列表类的对象，append()是列表类的一个方法，score1.append(random.randint(0,100))是对象 score1 执行具体方法 append()，将随机生成的数据追加到对象 score1 自己的数据中。

关于类与对象，可扫描二维码学习更多细节。



3.3.3.2 使用 Pandas 进行 CSV 文件的读写操作

Pandas 是 Python 中专门用于数据处理和数据分析的第三方库，需要下载和安装。它提供了强大的数据处理和分析功能，包括数据读取、清洗、转换、合并、分析、统计和可视化等。我们使用 Pandas 读写考生数据的文件。

Pandas 提供了 **Series** 和 **DataFrame** 两种数据结构，分别用于处理一维数据和二维数据。这两种数据结构能够满足处理金融、统计、社会科学、工程等领域里绝大部分问题的需求。

1. Series 简介

Series 用于处理一维数据。第 4 章将进一步介绍。

2. DataFrame 简介

DataFrame 是具有行标签和列标签的二维表格型数据结构，与 Excel 表类似。我们要处理的考生数据是二维数据，就可以使用 Pandas 中的 DataFrame。

DataFrame 是一个抽象的**类**，要使用 DataFrame 中的具体方法，如读写文件，必须有一个具体的 DataFrame 类**对象**。创建一个具体的 DataFrame 对象，需要使用 pandas.DataFrame()方法，其语法如下：

```
pandas.DataFrame(data, index, columns, …)
```

其中，pandas.DataFrame()方法主要参数的含义如下：

- Data 是所创建的 DataFrame 对象中的数据来源，其类型可以是列表、字典、Series 或 DataFrame 对象等。
- index 是 DataFrame 对象数据的行标签，如果未指定则默认为 RangeIndex，即(0,1,2,…,n−1)，n 为行数。
- columns 是列标签，如果未指定也默认为 RangeIndex，即(0,1,2,…,m−1)，m 为列数。

有需要或感兴趣的读者可以查阅 Pandas 官方文档，查看 pandas.DataFrame 更多的参数。

【例 3-22】　创建并输出 DataFrame 对象。

```
'''
1.使用 pandas 前需先使用 import 将 pandas 加载到程序中
2.可以用 as 关键字给 pandas 起个别名，程序中用到 pandas 的地方都可以换成别名
'''
import pandas as pd                                    #pd 是 pandas 的别名
stu1=[
    ['KS00001','张珊',109,120,114,78,82,90],    #保存第一位考生数据
    ['KS00002','李思',123,116,137,84,92,94],    #保存第二位考生数据
    ['KS00003','王武',133,129,132,83,85,78]     #保存第三位考生数据
    ]
#创建 DataFrame 对象 df,df 对象的二维数据来自 stu1, 并设置列标题
df = pd.DataFrame(stu1,columns=['准考证号','姓名','语文', '数学', '英语', '历史', '地理', '政治'])
#输出 DataFrame 对象 df
print(df)
```

例 3-22 运行结果如图 3-23 所示。

	准考证号	姓名	语文	数学	英语	历史	地理	政治
0	KS00001	张珊	109	120	114	78	82	90
1	KS00002	李思	123	116	137	84	92	94
2	KS00003	王武	133	129	132	83	85	78

图 3-23　例 3-22 运行结果

在图 3-23 中，最左侧的列的“0，1，2”是默认的行标题；最上面的“准考证号 姓名 语文 数学 英语 历史 地理 政治”则是在第 12 行代码中创建 df 对象时设置的列标题。

3. CSV 文件的读写操作方法

Pandas 中 DataFrame 的 to_csv()方法是将 DataFrame 对象的数据写入 CSV 文件；Pandas 的 read_csv()方法用于读取 CSV 文件中的数据，并返回一个 DataFrame 的对象。

1）写文件

DataFrame 的 to_csv()方法的功能是写文件。

该方法有很多参数可以设置，除了必须给出写入文件的路径和名称外，其他参数都可以缺省，即使用默认值。一些常用的参数含义如下：

- header（bool or list of str，default＝True）：是否写入列名作为文件的第一行。如果给定为字符串列表，则假定它是列名的别名。
- index（bool，default＝True）：是否将行索引写入文件。
- mode（str）：对于路径或类文件对象，写入的模式('w' 或 'a')。默认为 'w'，表示重写，'a'表示追加方式写入。
- encoding（str，optional）：使用的字符集编码类型。

其他参数不再一一展开，有兴趣和有需求的读者可以自行查阅。

【例 3-23】　使用 Pandas 将表 3-1 中的三位考生数据写入 D 盘的 myproject 目录下名为“写入的考生数据.csv”文件中。

使用 Pandas 的 to_csv()方法非常简单，只需在例 3-22 的代码最后添加如下的第 10 行代码即可。