

列表操作与陷阱排查

A 段六题练列表的增删改查、切片与拷贝，亲手复现并修正讲义里的三个常见错误；B 段用一份成绩单从解析做到排名、再做到脏数据，最后看清列表查找的代价；C 段学 git 的本地用法。

截止：小作业 2 · 10 月 27 日 周二 23:59

本次目标

- 熟练使用列表的基本操作与切片
- 亲手复现并修正三个常见错误：`[[0]*n]*m`、浅拷贝、边遍历边删除
- 养成用 `id()` 验证猜想的习惯
- 用嵌套列表完整处理一份真实形态的数据，并发现它的局限
- 学会在本地用 git 记录自己的修改

AI 政策提醒

本周处于 **Level 0**：不得使用 AI 生成或补全代码，请关闭编辑器的 AI 补全。遇到不会的先自己调试，再问同学和助教。

类型提示继续写

上周起要求每个函数都写类型提示。列表的写法是 `list[int]`、`list[str]`，嵌套列表是 `list[list[int]]`。B 段的记录形状复杂，写清楚类型对自己最有帮助。

5-1 成绩统计（热身）

给定一个成绩列表，输出人数、平均分、最高分、最低分和及格率。平均分保留两位小数。

```
输入: [87, 92, 45, 78, 55, 95]
输出:
人数 6
平均 75.33
最高 95
最低 45
及格率 66.67%
```

5-2 手写 max

不使用 `max()`，实现 `my_max(nums: list[int]) -> int | None` 返回列表中的最大值。空列表返回 `None`。

想清楚再写

初始值取多少？取 0 行不行？如果列表里全是负数呢？

5-3 有序去重

实现 `dedup(nums: list[int]) -> list[int]`，去掉重复元素并保持原有顺序。

```
输入: [3, 1, 3, 5, 1, 7]
输出: [3, 1, 5, 7]
```

不能用 `set()`：它不保证顺序（L6 讲）。

5-4 矩阵转置

实现 `transpose(matrix: list[list[int]]) -> list[list[int]]` 返回矩阵的转置。

输入: `[[1, 2, 3], [4, 5, 6]]`
输出: `[[1, 4], [2, 5], [3, 6]]`

5-5 找出并修正 Bug

下面的代码想把列表里所有偶数翻倍，期望得到 `[1, 4, 3, 8, 5, 12]`，但结果不对。先运行一次，看看得到的是什么；然后找出原因，写出修正版本，并在提交说明里用一两句话解释原因。

```
a = [1, 2, 3, 4, 5, 6]
for i in range(len(a)):
    if a[i] % 2 == 0:
        a.insert(i, a[i] * 2)
```

5-6 拷贝实验

写一段代码，用 `id()` 验证以下三件事，并把输出贴在提交说明里：

1. `b = a` 之后，`a` 和 `b` 是同一个对象
2. `b = a[:]` 之后，`a` 和 `b` 是不同对象
3. 但对嵌套列表，`b = a[:]` 之后 `a[0]` 和 `b[0]` 仍然是同一个对象

B 段·深入：一份成绩单，从头处理到尾

A 段的六道题互不相干。这一段只处理一份成绩单，但要处理四遍：先读进来，再排出名次，再面对脏数据，最后看清这套做法在多大规模上会失效。

数据是三十行文本，每行是「学号 姓名 小测1 小测2 小测3」：

```
RAW = [  
    "2026001 zhangsan 91 57 59",  
    "2026002 lisi 70 81 55",  
    "2026003 wangwu 43 65 66",  
    "2026004 zhaoliu - 78 90",  
    "2026005 qianqi 90 90 61",  
    "2026006 sunba 73 91 73",  
    "2026007 zhoujiu 60 53 75",  
    "2026008 wushi 105 60 70",  
    "2026008 wushi 60 70 80",  
    "2026009 zhengyi 60 61 69",  
    "2026010 chener 57 48 70",  
    "2026011 chusan 59 99 46",  
    "2026002 lisi 10 10 10",  
    "2026012 weisi 43 59 89",  
    "2026013 jiangwu 95 96 48",  
    "2026014 shenliu 53 73 62",  
    "2026015 hanqi 64 73 96",  
    "2026016 yangba 59 71 85",  
    "2026017 zhujia 64 41",  
    "2026018 qinshi 59 81 93",  
    "2026011 chusan 88 88 88",  
    "2026019 youyi 97 73 62",  
    " 2026020 xuer 70 80 90 ",  
    "2026021 hesan 82 90 80",  
    "2026022 lvsi 61 55 60",  
    "2026023 shiwu 58 x 88",  
    "2026024 zhangliu 63 93 95",  
    "2026019 youyi 40 40 40",  
    "2026025 kongqi 41 100 67",  
    "2026026 caoba 99 50 48",  
]
```

三十行里混着几种脏数据，B-3 才处理；B-1 和 B-2 先假定每行都是干净的。

不要用文件读写，那是第 12 讲的内容。把上面这段复制到代码里。

姓名为什么用拼音

因为这一行是按空格切开的。名字里一旦出现空格，字段数就变了，后面所有下标全错。真实的数据格式里，「分隔符会不会出现在数据本身里」是第一个要问的问题。CSV 要给含逗号的字段加引号，也是这个原因。

B-1 读进来

实现 `parse_roster(lines: list[str]) -> list[list]`，把每行文本解析成一条记录：

```
["2026001", "zhangsan", [87, 92, 45]]
```

学号保留为字符串（学号不参与算术运算），三次小测的分数放进一个列表。

```
输入: ["2026001 zhangsan 87 92 45", "2026002 lisi 78 65 90"]
输出: [["2026001", "zhangsan", [87, 92, 45]],
        ["2026002", "lisi", [78, 65, 90]]]
```

用 `split()`，不要数空格

`"2026001 zhangsan 87 92 45".split()` 直接给出五个字段，连续多个空格也能正确处理。L3 的练习里用过它，L6 会系统地讲。三个分数要用 `int()` 转换：`split()` 返回的都是字符串。

B-2 排出名次

实现 `rank(records: list[list]) -> list[list]`，按总分从高到低排序，总分相同的按学号从小到大。返回：

```
[[1, "2026003", "wangwu", 274],  
 [2, "2026002", "lisi", 233],  
 [3, "2026001", "zhangsan", 224]]
```

名次从 1 开始连续编号，不处理并列。

`sort()` 不返回列表

```
ranked = records.sort()    # 错: sort() 返回 None
```

要么先 `records.sort()` 再使用 `records`，要么用 `sorted(records)` 得到一个新列表。讲义里讲过这一条，但它仍然是这道题最常见的错误。

列表之间怎么比大小

Python 比较两个列表时，从第一个元素开始逐个比较，分出大小就停止。所以只要把排序依据按优先级放到列表前面，`sort()` 不带任何参数就能排对。想让总分从高到低、学号从低到高，试试排 `[-总分, 学号]`。

B-3 脏数据

真实的数据不会这么干净。完整的 `RAW` 里混着这些情况：

情况	RAW 里的例子	怎么处理
缺考	2026004 zhaoliu - 78 90	- 按 0 分计入，这条记录保留
列数不对	2026017 zhujiu 64 41 (少一次小测)	整条丢掉
分数非法	2026008 wushi 105 60 70、2026023 shiwu 58 x 88	不是整数或不在 0-100，整条丢掉
学号重复	2026002、2026011、2026019 各出现两次	保留先出现的那条
多余空格	2026020 xuer 70 80 90	正常保留

实现 `clean(lines: list[str]) -> list[list]`，返回清洗后的记录，形状和 `parse_roster` 一样。按上表从上到下的顺序检查：先看列数，再看分数是否合法，最后在保留下来的记录里去重。

同时在提交说明里写清楚：丢掉了几条、分别因为什么。

怎么判断一段文字是不是数字

`"105".isdigit()` 是 `True`，`"x".isdigit()` 是 `False`。有了它，就能在调用 `int()` 之前先检查一下。直接 `int("x")` 会报错，程序当场停止。

这是字符串的方法，L6 才会系统地讲，这里先用这一个。注意 `"-5".isdigit()` 也是 `False`（负号不是数字），这一题里正好合用。

顺序不是随便定的

这道题规定「先验合法性，再去重」。反过来会出问题：

```
2026008 wushi 105 60 70    ← 分数越界
2026008 wushi  60 70 80    ← 同一个学号，这条是好的
```

这两行就在 **RAW** 里，而且是相邻的两行。

如果先去重，第一条先占用了这个学号，第二条被当成重复丢掉：好数据被删了，坏数据留了下来。坏数据随后又因分数越界被丢掉，于是这个学生的记录就完全没有了。

在提交说明里再举一个自己想到的、两种顺序结果不同的例子。数据清洗中的分歧，大多出在这类地方。

B-4 这套做法在什么规模上失效

现在做几种查询，都用列表来做：

1. 写 `find(records, sid)`，按学号找出一条记录。
2. 在 `find` 里加一个计数器，统计为了找到目标，一共比较了多少次。分别查排在第 1 条、第 15 条、第 30 条的学号，各记下比较次数。
3. 写 `has_duplicate(records)`，判断有没有重复学号——只用列表，不用 `set`。同样统计比较了多少次。

然后回答：

1. `find` 的比较次数和记录条数是什么关系？
2. `has_duplicate` 呢？它比 `find` 严重在哪里？
3. 这门课有 75 人，这套做法完全够用。如果是全校三万人的成绩库，每天要查几万次，会发生什么？
4. 你希望有一种什么样的容器，能让「按学号查一条记录」这件事不用挨个比？

第 4 问就是下一讲

L6 讲字典。字典按键取值的速度几乎不受数据量影响，一万条和一百万条差不多快。代价是键必须不可变。记住这一节里遇到的困难，下周讲到字典时，就知道它是用来解决什么问题的。

C 段 · 基本功：git 入门

到这一周，代码已经有一定的规模了：B 段那几个函数改错一次，就可能改不回原来的样子。git 用来解决这个问题：它记住每一个版本，随时可以退回去。

这周只在自己电脑上用。下周学怎么推到 GitHub。

第一次使用要先配置身份

```
git config --global user.name "你的名字"
git config --global user.email "你的邮箱"
```

每次提交都会记下是谁提交的，所以 git 要先知道你是谁。这一步只做一次。

六个命令

命令	作用
<code>git init</code>	在当前目录建一个仓库。只在项目开始时做一次
<code>git status</code>	当前有哪些改动、哪些已经暂存。 不确定的时候就执行一次
<code>git add 文件</code>	把改动放进 暂存区 ，表示「这些我要提交」
<code>git commit -m "说明"</code>	把暂存区里的改动记成一个版本
<code>git log --oneline</code>	看提交历史，一行一条
<code>git diff</code>	看 还没暂存 的改动； <code>git diff --staged</code> 看已暂存的

`commit` 忘了 `-m` 会掉进编辑器

不带 `-m`，git 会打开一个编辑器让你写提交信息，很多人第一次遇到会以为终端卡住了。如果进去了：多半是 vim，按 `Esc` 再输入 `:q!` 回车退出；是 nano 则按 `Ctrl+X`。退出之后带上 `-m` 重新执行。

`git add .` 没有看上去那么无害

`.` 是「当前目录下所有东西」，会把 `__pycache__/`、`.DS_Store`、几百兆的数据文件一并加进来，而且一旦提交过就很难彻底清除。先 `git status` 看清楚，再决定加什么。

想长期排除某些文件，在项目根目录建一个 `.gitignore`，一行一条：

```
__pycache__/  
.DS_Store  
*.pyc
```

commit 不是备份

提交只是把版本存进这台电脑上的仓库，硬盘坏了一样会丢。下周把仓库推到 GitHub，才算有了备份。

C-1 走一遍

在本周实验的目录里操作，把每一步的命令和输出记进提交说明：

1. `git init`，然后执行 `git status`，它输出了什么？
2. 建一个 `.gitignore`，至少写上 `__pycache__/`
3. `git add` 你的实验代码，再 `git status`，看输出有什么变化
4. `git commit -m "lab05 A 段"` 提交第一版
5. 任意修改一个函数，再执行 `git diff`。注意行首的 `-` 和 `+` 分别是什么意思
6. `git add` 之后再执行 `git diff`，为什么这次什么都没有输出？换成 `git diff --staged` 试试
7. 提交第二版，`git log --oneline` 看历史

C-2 想一想

1. 既然最终都要 commit，为什么 git 还要多一个「暂存区」？什么场景下你会只想提交一部分改动？
2. `git log` 里的提交信息该怎么写？比较下面两条，三个月后回头看，哪一条更有用：`"改了一下"` 和 `"B-2 排序改用 [-总分, 学号]，修掉并列时顺序不定的问题"`

提交

A 段六题在评测平台上提交，每题独立判分。B 段的 `parse_roster`、`rank`、`clean` 也在平台上判；B-4 和 C 段的回答写在提交说明里。

判分是怎么做的

平台会用一组测试用例调用提交的函数，比对返回值。所以：函数要 `return` 结果，不要 `print`（除非题目明确要求输出）。这一点在 L3 讲过，是最常见的丢分原因。

提交前自查

- ☐ 每个函数都有类型提示
- ☐ 5-5 的错误原因解释了
- ☐ 5-6 的三段 `id()` 输出贴上了
- ☐ B-3 说明了丢掉几条、为什么，并自己举了一个换顺序会出错的例子
- ☐ B-4 的四个问题都回答了
- ☐ C 段的命令输出和两个问题都写了