

词频统计、容器选型与 Git 远程同步

A 段用六道函数题和一道选型题组合字典、集合、字符串；B 段比较列表与集合的查找代价；C 段接着上周的 Git，练习私有仓库与远程同步。

截止：小作业 2 · 10 月 27 日 周二 23:59

本次目标

- 用字典保存对应关系、分组和统计次数
- 让列表保留顺序，让集合加快成员判断
- 把字符串拆成字段，清洗后转换类型
- 通过计数和测量说明容器的适用场景
- 区分 Git 的本地提交与远程同步

上一周把一批数据放进列表，这周要进一步判断：需要顺序、对应关系，还是只需要成员关系？先用 A 段练习具体操作，再在 B 段比较代价，最后把代码同步到自己的私有远程仓库。

AI 政策

本周仍是 **Level 0**，不得让 AI 生成或补全代码。遇到问题先运行小例子、检查中间结果，再问同学和助教。

类型提示与提交

六道编程题的函数名和参数按题面写，每个函数都写类型提示。代码在「小作业 2」对应题目中提交，用 `return` 返回结果；题面没有要求在函数里打印。6-7、B 段和 C 段分别有答题卡，不要把这些回答混进代码文件。

容器的类型提示

写法	含义
<code>list[int]</code>	整数列表
<code>set[str]</code>	字符串集合
<code>tuple[str, int]</code>	依次为字符串、整数的二元素元组
<code>dict[str, list[str]]</code>	键是字符串，值是字符串列表
<code>dict[str, dict[str, int]]</code>	姓名对应一份“科目 → 整数分数”的字典

外层和内层的类型都要写清楚。类型提示不做运行时检查，代码仍要按输入约定处理数据。

A 段 · 基本操作与组合

6-1 反转字典

实现 `invert(d: dict[str, int]) -> dict[int, str]`，返回键和值对调后的**新字典**，不修改输入字典。

```
invert({"a": 1, "b": 2}) → {1: "a", 2: "b"}
invert({})                → {}
```

按输入字典的遍历顺序写入结果。同一个结果键被写入多次时，后面的值覆盖前面的值。

在本题提交说明里回答：**如果原字典有两个键对应同一个值，反转后会怎样？** 给出一个具体输入和输出，并解释哪些信息无法保留。

6-2 保序去重

实现 `unique_ordered(items: list[int]) -> list[int]`，去掉重复整数，保留首次出现的顺序，返回列表。

```
unique_ordered([3, 1, 3, 5, 1, 7]) -> [3, 1, 5, 7]
unique_ordered([9, 1, 9, 0])       -> [9, 1, 0]
unique_ordered([])                  -> []
```

必须用集合判断是否出现过，不能在已保留元素的列表里用 `in` 逐项查找。先转成集合再排序也不行，那会失去首次出现的顺序。

两个容器各做一部分

集合负责“是否出现过”，列表负责“按首次出现的顺序保存”。不要要求一个集合同时完成这两个任务。

在本题提交说明里解释：为什么集合适合反复判断是否出现过？不要把“平均查找快”说成“每次一定只做一步”。B 段会用实验进一步比较。

6-3 共同好友

实现 `common(a: list[str], b: list[str]) -> list[str]`，返回两个名单中都出现的名字，每个名字只出现一次，按 Python 字符串的字典序排列，返回列表。

```
common(["张三", "李四", "王五"], ["李四", "王五", "赵六"])
    -> ["李四", "王五"]
common(["c", "a", "a"], ["a", "c", "c"])
    -> ["a", "c"]
common([], ["a"]) -> []
```

使用 `sorted()` 的默认顺序即可。中文名字也按字符串规则比较，不要求按拼音或笔画排序。大小写不同的名字不合并。

先比较成员，再确定顺序

两个名单的共同成员对应集合的交集，但题目要求返回有序列表，不是集合。

6-4 按首字符分组

实现 `group_by_first(words: list[str]) -> dict[str, list[str]]`，以单词的第一个字符作键，同组单词按输入中的先后顺序保存，**重复单词也保留**。

输入列表可以为空；其中每个单词都保证是**非空字符串**。不需要统一大小写。

```
group_by_first(["banana", "apple", "avocado", "apple"])
→ {"b": ["banana"], "a": ["apple", "avocado", "apple"]}
group_by_first(["a", "b", "a"])
→ {"a": ["a", "a"], "b": ["b"]}
group_by_first([]) → {}
```

第一次出现时要先建立组

直接写 `groups[w[0]].append(w)`，键第一次出现时会报 `KeyError`。可以先判断键在不在，或用 `setdefault()` 取得已有列表、必要时插入新列表。每一组要有自己的列表，不能用 `dict.fromkeys(keys, [])` 让所有组共用一个列表。

6-5 解析成绩单

实现 `parse(lines: list[str]) -> dict[str, dict[str, int]]`，把每行 `姓名, 科目, 分数` 解析成嵌套字典 `{姓名: {科目: 分数}}`。

本题保证每行有三个逗号分隔的字段，姓名和科目非空且不含逗号，分数能转换为 0–100 的整数。**三个字段的两端都可能**有空白，需要分别去掉；不要求处理缺字段或非法分数。

同一学生的不同科目都要保留；同一学生、同一科目出现多次时，**保留最后一条的分数**。空输入返回空字典。

```
parse(["张三,数学,87", "张三,物理,92", "李四,数学,65"])
→ {"张三": {"数学": 87, "物理": 92}, "李四": {"数学": 65}}
parse([" 张三 , 数学 , 60 ", "张三,数学,90"])
→ {"张三": {"数学": 90}}
parse([]) → {}
```

按步骤检查中间值

`split(",")` 先拆三个字段, `strip()` 去两端空白, `int()` 转分数。第一次遇到某个姓名, 要先建立这个人的内层字典; 后来增加科目时, 不能替换整份内层字典。

6-6 词频统计

实现 `top_words(text: str, n: int) -> list[tuple[str, int]]`, 返回出现最多的前 n 个单词及次数, 每项是 (单词, 次数) 元组。

本题采用以下简单规则:

1. 用不带参数的 `split()` 按连续空白分词。
2. 忽略大小写。
3. 去掉词段两端的标点, 至少处理 `.,!?:;`; 中间的标点不作分隔。
4. 清洗后是空字符串的词段跳过, 不统计空词条。
5. 次数从多到少排列; 次数相同, 按单词的字典序从小到大排列。

n 保证是非负整数。 n 为 0 时返回空列表; n 大于不同单词的数量时返回全部, 不补空项。空文本或只有上述标点和空白的文本, 返回空列表。

```
top_words("The cat. The DOG! the cat", 2)
→ [("the", 3), ("cat", 2)]
top_words("dog cat dog cat", 2)
→ [("cat", 2), ("dog", 2)]
top_words("... cat! CAT ;", 10)
→ [("cat", 2)]
top_words("cat", 0) → []
top_words("", 3) → []
```

先验证每一阶段

分词、清洗、计数、排序分别看结果。只在最后打印结果，很难判断错误发生在哪一步。具名 key 函数可以返回 `(-次数, 单词)`，先比负次数，相同再比单词。`Counter.most_common()` 的并列顺序按首次出现，不能直接代替本题要求的排序。

6-7 容器选型（不写代码）

在「L06·容器选型」答题卡里，为每个场景写出选择和理由。可以组合几种容器，但要说明各自的作用：

1. 记录 116 名学生的学号到姓名的对应。
2. 保存一次考试所有人的分数，之后计算平均分和最高分。
3. 反复判断一个单词是否在一万个常用词组成的词表中。
4. 表示平面上的一个点，并用它作字典键。
5. 记录一个班选修的所有课程，同一门课不重复。
6. 按提交时间顺序保存一个学生的多次提交记录。

理由要对应具体操作

「字典快」不够具体。要说清楚按什么查询、是否需要顺序、是否允许重复、是否需要增删。元组作键还要检查组成元素是否可哈希，不能只写“元组不可变”。

B 段·深入：列表与集合的查找代价

A 段里列表和集合经常配合使用。这里比较它们处理同一组数据、同一组查询时的代价，不只看一次运行谁更快。

B-1 先数比较次数

下面的函数在列表里逐项查找，返回“找到了没有”和元素比较次数：

```
def contains_counted(items: list[int], target: int) -> tuple[bool, int]:
    checks = 0
    for x in items:
        checks += 1
        if x == target:
            return True, checks
    return False, checks
```

分别取 n 为 1000、2000、4000，生成 `items = list(range(n))`。查询 `0`、`n // 2`、`n - 1`、`n`，记录每次的结果和比较次数。

在 B 段答题卡中贴表，并说明：目标的位置或是否存在，怎样影响比较次数？为什么不能只查第一个元素就评价列表查找？

B-2 用集合做同样的查询

先执行一次 `members = set(items)`，再用 `target in members` 查询同样的四个目标。确认真假结果和 B-1 一致。

回答：

1. 集合为什么通常不需要从头逐项比较？
2. 为什么仍不能说集合的每次查询一定只有一次比较？
3. 从列表构造集合本身要处理多少个输入元素？只查询一次时，是否一定值得先转换？

本题不要求测出集合内部的真实比较次数。B-1 数的是自己写的列表循环，不能把集合内部实现假定为“每次一步”再编一列计数。

B-3 测量时分开准备与查询

使用 L5 讲过的 `time.perf_counter()` 测量耗时。例如：

```
import time

start = time.perf_counter()
members = set(items)
build_seconds = time.perf_counter() - start
print("构造集合耗时: ", build_seconds)
```

把“构造集合”和“执行全部查询”分别计时。B-1 的函数只用于数比较次数；查询计时分别使用 `target in items` 和 `target in members`，查询序列、循环结构和重复轮数相同，计时区间里不要打印。

每组至少测三次，记录结果。若一次查询太短、波动太大，可以把同一组查询重复执行多轮，但两种容器的轮数要相同。

在答题卡中说明：计数给出了什么规律，计时给出了什么证据？不能把某次测出的倍数当成所有机器、所有输入的固定结论。

B-4 根据任务选方案

写下下面两种场景的选择，并结合 B-1 到 B-3 说明理由：

- 数据只有几个元素，整个程序只查询一次。
- 数据有很多元素，程序要反复查询是否出现过，还要保留原始顺序。

第二种场景可以同时保留列表和集合，但会增加存储。说明获得了什么便利、付出了什么代价。L11 会用复杂度记号把这些关系说得更精确。

C 段 · 基本功：Git 的远程同步

Lab 5 的 `commit` 把版本保存在本机。远程仓库保存另一份版本历史，便于备份和在别的目录继续工作。`commit` 不会自动上传，`push` 才会把提交发送到远程。

六个操作

操作	作用
<code>git branch -M main</code>	把当前分支改名为 main，第一次建立远程对应时使用
<code>git remote add origin 地址</code>	把一个远程地址记为 origin，不上传文件
<code>git remote -v</code>	查看配置的远程地址
<code>git push -u origin main</code>	上传 main 的提交，并记住对应的远程分支
<code>git clone 地址 新目录</code>	下载仓库历史，在一个新目录里开始工作
<code>git pull --ff-only</code>	取得远程新提交；无法直接快进时停止，不自动合并

私有仓库与认证

在 GitHub 建一个**私有空仓库**，不要预先添加 README、许可证或 `.gitignore`，以免它先产生另一段历史。本机仓库沿用 Lab 5 已有的提交。

- 作业代码不要放进公开仓库，不要上传其他同学的代码。
- 认证按 GitHub 提示进行。HTTPS 认证不能把账户登录密码当作 Git 密码；可使用凭据管理器或个人访问令牌。
- 令牌、密码、私钥不能写进代码、仓库地址或提交说明，也不要截图给别人。
- 如果 GitHub 无法访问或认证失败，请找助教，不要反复尝试强制推送。可以使用下面的本地替代方式练习同一套操作。

C-1 推送与取得新版本

在 C 段答题卡中记录每一步的命令和关键输出，认证过程不截图：

1. 在本机实验目录运行 `git status`，确认哪些文件已经提交。只提交本人实验代码，确认 `.gitignore` 排除了缓存和不应上传的文件。

2. 将当前分支命名为 main，添加自己的私有远程仓库为 origin，再用 `git remote -v` 确认地址。如果已有 origin，先查看，不要重复添加或随意覆盖。
3. 执行 `git push -u origin main`，在远程页面确认自己的提交出现。
4. 到原实验目录之外，用 `git clone` 创建一个新副本，比较两份目录中的 `git log --oneline`。
5. 回原目录，在 README 中增加一行实验说明，提交并推送。
6. 在新副本里先确认没有未提交的改动，再执行 `git pull --ff-only`。检查新提交和 README 是否出现。

不用强制覆盖解决报错

这一练习不需要 `--force`、`reset --hard` 或删除 Git 历史。`--ff-only` 停下来，说明需要先弄清两边的历史，找助教一起检查。

C-2 网络不可用时的本地替代

Git 的远程地址也可以是本机另一个目录。新建一个独立的空目录，在其中运行：

```
git init --bare --initial-branch=main
```

`--bare` 建立的是没有工作文件、只存历史的仓库，在本练习中充当远程端；`--initial-branch=main` 让它默认使用 main，和本机分支对应。把这个目录的绝对路径用作 origin，完成 C-1 的 push、clone、再次提交与 pull。新副本要放在实验目录之外，不要把仓库套在仓库里面。

本地替代可以检验操作流程，但它仍在同一台电脑上，不能防止硬盘损坏造成的丢失。在答题卡里说明采用了哪种方式。

C-3 说明每一步的区别

在答题卡中回答：

1. 修改文件、`git add`、`git commit`、`git push` 分别改变了哪里？为什么只 `commit` 还没有远程备份？
2. `clone` 后为什么已经能看到此前的提交？`pull` 的作用是什么？
3. 为什么本次默认用私有仓库，为什么认证信息不能放进提交说明？

提交

入口是 [小作业 2](#)。A 段六道编程题分别提交代码，按原有测试自动判分，每题 4 分，合计 24 分不变。

- 6-1、6-2 题目要求的说明，写在各自提交代码时弹出的提交说明框。
- 6-7 在「L06·容器选型」答题卡中回答。
- B 段在「L06·列表与集合的查找代价」答题卡中提交表格、观察和解释。
- C 段在「L06·Git 远程同步」答题卡中提交命令与说明；不要贴认证信息。

三张答题卡中的问答由助教评阅，不自动判分。保存并提交后，可以重新打开检查是否保留了回答。

提交前自查

- ☐ 六个函数都有类型提示，返回类型符合题面
- ☐ 6-1 没改输入，重复值覆盖的说明有具体例子
- ☐ 6-2 保留首次出现顺序，用集合检查是否出现过
- ☐ 6-3 返回排序后的列表，重复名字只出现一次
- ☐ 6-4 组内顺序和重复单词都保留，首次出现不会报错
- ☐ 6-5 去掉各字段的两端空白，分数是整数，重录科目保留后值
- ☐ 6-6 测过空文本、纯标点、并列次数和 n 超过词数
- ☐ 6-7 的六条选择都有对应需求的理由

- ☐ B 段的计数与计时分开记录，没有虚构集合内部计数
- ☐ C 段完成同步或本地替代，没有提交任何秘密信息