

递归、约瑟夫环与查找工具

A 段练习递归函数，B 段比较约瑟夫环的模拟与递推，C 段学习查找和管道。本周同时布置个人作业 1：普通汉诺塔与相邻杆汉诺塔。

截止：小作业 3 · 11 月 17 日 周二 23:59

本次目标

- 为递归函数确定输入范围、基准情形和终止条件。
- 比较数位和的递归与循环实现，观察递归深度限制。
- 用模拟结果核验约瑟夫环的递推结果。
- 使用查找命令和管道处理文本。

A、B、C 三段同时开放，建议分别用约 40、35、15 分钟。A 段占本周小作业分的 60%，B 段占 40%；C 段只登记完成情况，内容纳入期末考核。平台显示题库原始分，助教按课程规定汇总；增加题库原始分不增加学期考核比例。

AI 政策

本周实验按 **Level 0** 执行，不得使用 AI 生成或补全代码。个人作业 1 虽然跨第 7–8 周，整份作业仍按 Level 0 执行。第 8 讲后的政策调整不适用于这份个人作业。

本周实验归入[小作业 3](#)，个人作业使用[个人作业 1](#)入口。两份作业的截止时间分别见提交页，均按北京时间（GMT+8）显示；不要把实验周次当作截止日期。

A 段 · 递归练习

六道代码题各 4 分，A 段自动评测原始分合计 24 分。原有五题仍各 4 分，7-5 新增的 4 分不代表学期成绩增加 4%。7-5 的观察另交文字答题卡，由助教评阅。

递归实现要求还需代码审查。静态检查只能识别规定的语法限制，不能证明函数确实采用递归，也不能证明终止性或正确性。下列自动测试使用的递归深度均低于通常的默认上限；大整数实验单独在本机进行，不提高递归上限。

7-1 递归数位数

实现 `count_digits(n: int) -> int`，用递归计算非负整数的十进制位数。自动测试输入至多 16 位。本题提交文件不得使用 `for`、`while`、推导式或生成器表达式。

```
count_digits(0)      → 1
count_digits(7)      → 1
count_digits(12345) → 5
```

零的位数

十进制写法 `0` 有一位。若每次除以 10，直到参数变成 0，基准情形应怎样处理？

7-2 递归反转字符串

实现 `reverse_string(s: str) -> str`。不使用 `reversed()`、负步长切片（包括 `[::-1]`）、`for`、`while`、推导式或生成器表达式；这些限制适用于本题提交文件。普通正向切片可以使用。自动测试字符串长度不超过 60。

```
reverse_string("")      → ""
reverse_string("a")     → "a"
reverse_string("hello") → "olleh"
```

7-3 回文判断

实现 `is_palindrome(s: str) -> bool`，用递归判断字符串正读与反读是否相同。逐字符比较，不忽略大小写、空格或标点。

```
is_palindrome("")      → True
is_palindrome("a")     → True
is_palindrome("abba")  → True
is_palindrome("abc")   → False
```

子问题

首尾字符相同后，还需判断中间的字串。空字符串和单字符字符串分别应返回什么？

7-4 压平嵌套列表

实现 `flatten(items: list) -> list[int]`，按从左到右的顺序返回所有整数构成的新列表。输入只含整数和列表，嵌套结构有限且没有循环引用；不得修改外层列表或任何内层列表。自动测试的嵌套深度不超过 6，不要求处理任意深度而不受递归上限影响。

```
flatten([1, [2, [3, [4]]]]) → [1, 2, 3, 4]
flatten([])                  → []
flatten([[], [1], [[2], [3]]]) → [1, 2, 3]
```

`isinstance(x, list)` 判断 `x` 是否为列表：整数返回 `False`，列表返回 `True`。
本题可以用循环处理当前列表，用递归处理内层列表。

7-5 数位和：递归与循环

在同一个代码文件中实现两个函数，输入均为非负整数：

```
def sum_digits_rec(n: int) -> int:
    """Return the decimal digit sum recursively; the sum for 0 is 0."""
    pass

def sum_digits_loop(n: int) -> int:
    """Return the decimal digit sum using a loop; the sum for 0 is 0."""
    pass
```

```
sum_digits_rec(0)      → 0
sum_digits_loop(0)     → 0
sum_digits_rec(12034) → 10
sum_digits_loop(12034) → 10
```

递归版不得使用循环、推导式或生成器表达式；循环版不调用递归版或递归辅助函数。自动测试只核验至多 16 位的输入。将两个函数一起提交到小作业 3 的「数位和：递归与循环」代码题。

再在本机分别运行 `n = 10**1000`。这个数有 **1001 位**。保持默认递归上限，在「7-5 递归与循环说明」答题卡中回答：

1. 比较两个实现的长度和可读性，结合具体代码说明判断。
2. 写明 Python 版本、`sys.getrecursionlimit()` 的值、两个实现的实际结果或异常类型。结合逐次除以 10 的调用深度解释观察结果。有限递归为什么仍可能触发 `RecursionError`？循环版的结果是什么？

默认上限因环境而异，不要求各人的异常完全相同。`RecursionError` 表示达到解释器的递归限制，不能据此断言物理内存已经耗尽；不通过调高上限完成本题。

7-6 硬币组合计数

实现 `count_ways(amount: int, coins: list[int]) -> int`，用递归计算组合数。金额为非负整数，面额是**互异的正整数**，每种硬币可使用任意多次，不计使用顺

序。面额列表不要求排序，不得修改它。自动测试金额不超过 40，面额种类不超过 4。

```
count_ways(5, [1, 2, 5]) → 4
count_ways(0, [1, 2, 5]) → 1
count_ways(3, [2])      → 0
count_ways(0, [])       → 1
count_ways(5, [])       → 0
```

金额 5 的四种组合是 5、2+2+1、2+1+1+1、1+1+1+1+1。1+2 与 2+1 是同一组合。

使用或跳过当前面额

使用当前硬币后，金额减少，当前面额仍可继续使用；跳过当前面额后，金额不变，只使用剩余面额。两类组合数相加。

金额为 0 时，空选择算一种组合。递归子调用的金额可能为负，此时返回 0；正金额且没有可用面额时也返回 0。

B 段 · 约瑟夫环

n 个人按 1, 2, ..., n 围成一圈，从编号 1 开始报数，每次数到 k 的人出列。下一人重新从 1 报数，直到只剩一人。输入满足 $1 \leq n \leq 120$ 、 $1 \leq k \leq 1000$ 。

B-1 列表模拟

实现 `josephus_sim(n: int, k: int) -> tuple[list[int], int]`，返回（出列顺序，最后留下的编号）。出列顺序不包括最后留下的人。

```
josephus_sim(7, 3) → ([3, 6, 2, 7, 5, 1], 4)
josephus_sim(1, 9) → ([], 1)
josephus_sim(5, 1) → ([1, 2, 3, 4], 5)
```

记录当前列表中的报数起点。删除一人后，下一次报数起点对应哪个下标？写出 `n = 7, k = 3` 的每轮列表和删除位置。

B-2 递推求最后编号

实现 `josephus_rec(n: int, k: int) -> int`，用递归求最后留下的编号。

先用从 0 开始的编号表示结果 `J(n, k)`：

$$\begin{aligned} J(1, k) &= 0 \\ J(n, k) &= (J(n-1, k) + k) \% n \quad (n > 1) \end{aligned}$$

函数对外返回从 1 开始的编号。解释删除首人后，剩余环的重新编号怎样映射回原来的编号；说明递归的基准情形与终止条件。

B-3 结果核验

将两个函数放在同一个文件，提交到「约瑟夫环：模拟与递推」代码题（原始 4 分）。在本机比较 `n = 1..30`、`k = 1..10` 的全部结果：模拟返回的最后编号应等于递推返回值。

在「约瑟夫环说明」答题卡中提交核验代码和汇总结果，另列 `(n, k) = (1, 9)、(5, 1)、(7, 3)、(10, 20)` 的出列顺序与最后编号。不同输入不能只靠一个已知答案核验。

B-4 两种方法的代价

模拟记录了哪些数据？递推只返回最后编号时，不再保存哪些数据，又需要多少层调用？结合实现说明列表删除与递归调用的代价。无需比较大规模输入的秒数，也不要提高递归上限。

C 段 · 查找与管道

使用 macOS 终端、Linux shell 或 Windows 的 WSL。以下命令不是 PowerShell 语法；WSL 无法使用时，请在实验课向助教说明。所有操作只在自己的练习目录进行。

查找命令

- `grep` 查找文件中匹配的行；`-n` 显示行号，`-F` 按字面文本匹配，`-r` 在目录中递归查找。
- `find` 遍历目录，`-type f` 只选普通文件，`-name '*.py'` 按文件名筛选。通配符加引号，避免由 shell 提前展开。
- `wc -l` 统计换行符数量。以下练习文件每行都有换行符，因此结果等于行数。
- `sort` 排序文本行，`uniq` 合并相邻的重复行；统计不同内容通常先排序再去重。
- 管道 `|` 将前一个命令的标准输出传给后一个命令的标准输入。

用编辑器建立 `words.txt`，内容如下，最后一行也保留换行符：

```
pear
apple
pear
banana
apple
```

先分别运行，再运行管道：

```
grep -n -F 'pear' words.txt
wc -l < words.txt
sort words.txt
uniq words.txt
sort words.txt | uniq
sort words.txt | uniq | wc -l
find . -type f -name '*.py'
grep -rn -F 'print(' .
```

`grep` 查文件内容，`find` 查目录中的文件。`uniq words.txt` 不会合并所有重复词，因为相同词没有相邻；`sort words.txt | uniq | wc -l` 的结果为 3。管道统计的是输出行数，文件名含换行符时，`find ... | wc -l` 不适合作为准确的文件数统计。下面练习只使用没有换行符的普通文件名。

C-1 文本统计与代码查找

1. 贴出上面六条针对 `words.txt` 的命令和输出，解释直接使用 `uniq` 与排序后使用 `uniq` 的差别。
2. 在自己的 Lab 1-7 代码目录运行 `find`，用管道统计 `.py` 普通文件数；写明查找目录。
3. 用 `grep -rn -F 'print('` 查找同一目录中的字面文本，记录部分输出；解释为什么匹配行不一定是实际执行的 `print` 调用。

将命令、输出和解释填入小作业 3 的「查找与管道」答题卡。只登记完成情况，不增加自动分；不提交终端截图或另行上传附件。

个人作业 1· 普通与相邻杆汉诺塔

三次个人作业各占总成绩 5%，合计 15%。本作业在第 7-8 周完成，截止时间由[个人作业 1 提交页](#)统一显示；以下分项是本作业内部的 100 分，不是额外的学期加权分。

7-7 普通汉诺塔

三根杆标为 `A`、`B`、`C`。`n` 为整数，要求支持 $0 \leq n \leq 8$ 。盘号为 `1..n`，盘号越大，盘子越大；初态所有盘子在 A，从底到顶为 `n, ..., 2, 1`，目标是全部移到 C。

每次只移动一根杆顶端的一个盘子，不得把大盘放在小盘上。普通版允许任意两根不同杆之间移动。

```
def hanoi_moves(n: int) -> list[tuple[int, str, str]]:
    """Return an optimal ordinary-Hanoi sequence from A to C."""
    pass
```

返回移动列表，每项必须是（盘号，起点杆，终点杆）元组，不只打印指令或返回步数。 $n = 0$ 返回新空列表；每次调用返回独立列表。序列须合法、完成目标且最优，长度为 $2^{*n} - 1$ 。

```
hanoi_moves(1) -> [(1, "A", "C")]
hanoi_moves(2) -> [(1, "A", "B"), (2, "A", "C"), (1, "B", "C")]
```

7-8 相邻杆汉诺塔

杆的位置固定为 **A-B-C**，只允许 **A↔B**、**B↔C**。**A→C** 和 **C→A** 都不能作为单次移动。其余盘号、输入范围、初态和目标与普通版相同。

```
def adjacent_hanoi_moves(n: int) -> list[tuple[int, str, str]]:
    """Return an optimal A-to-C sequence using adjacent pegs only."""
    pass
```

返回完整移动列表， $n = 0$ 返回新空列表。序列须满足相邻规则、大小规则和终态要求，并达到最少步数 $3^{*n} - 1$ 。

```
adjacent_hanoi_moves(1) -> [(1, "A", "B"), (1, "B", "C")]
```

从最小输入推导：最大盘从 A 到 C 要经过 B。两次最大盘移动之间，小盘应在哪根杆？解释三次 $n-1$ 盘端点转移与两次最大盘移动，得到 $T(0)=0$ 、 $T(n)=3T(n-1)+2$ 。还需说明为什么任何合法方案都不能少于这些步骤，并解出 $T(n)=3^{*n}-1$ ；仅给公式不够。

7-9 移动序列核验

```
def is_valid_sequence(
    n: int,
    moves: list[tuple[int, str, str]],
    adjacent: bool = False,
) -> bool:
    """Check move legality and the final state, not optimality."""
    pass
```

模拟三根杆的状态。 `adjacent=False` 按普通规则检查， `adjacent=True` 还检查相邻规则。不得修改 `moves`。返回 `True` 须满足：

- 每项是长度为 3 的元组，盘号是 `1..n` 的整数（不接受布尔值），杆名是 A、B、C，且起点不同于终点。
- 移动的是起点杆当前的顶盘，终点杆为空或顶盘更大。
- 相邻版没有端点杆之间的直接移动。
- 全部操作结束后，A、B 为空，所有盘子按大小顺序在 C。

不满足任一条件返回 `False`。只要求上述范围内的 `n` 和布尔型 `adjacent`；`moves` 为列表，其中可能含有不符合移动格式的项。`n = 0, moves = []` 返回 `True`。

合法与最优不同。`n = 1` 时，普通序列 `[(1, "A", "B"), (1, "B", "A"), (1, "A", "C")]` 合法且完成目标，应返回 `True`，但不是最优解。最优步数由前两道生成题分别检查，核验函数不能仅比较列表长度。

7-10 递推解释与测试说明

在「汉诺塔说明」答题卡回答：

1. 普通版为什么至少需要 `2T(n-1)+1` 步？给出相应构造、基准情形和最少步数。
2. 相邻版为什么需要 `3T(n-1)+2` 步？描述三次小盘转移和两次最大盘移动，说明下界与构造一致，再推导 `3*n-1`。

3. 给出本机对 `n = 0..5` 的两种移动数、合法性和终态核验结果，以及至少四个非法序列测试：非法盘号、非顶盘移动、大盘压小盘、相邻版直接 A→C。再给一个合法但非最优序列，说明核验函数的返回值。
4. 说明函数划分、类型提示及调用深度与输出长度的区别。用小规模实测讨论时间和存储限制，不尝试生成极大的移动列表。

可选练习：研究 L 型骨牌覆盖棋盘的分治方法，向助教展示思路。本次正式题包没有该选做题的自动评测或加分入口，不增加学期权重。

分值与提交方式

内容	本作业内部原始分	评阅方式
普通版完整、合法、最优的移动序列	30	自动测试，递归设计另作代码审查
相邻版完整、合法、最优的移动序列	40	自动测试，递归设计另作代码审查
核验函数：规则、终态、不修改输入	20	自动测试
递推论证、测试证据与代码质量	10	说明答题卡，助教评阅

把三个函数写在同一份 `solution.py` 中，不实现旧版 `render`，不要求可视化。在个人作业 1 的三个代码题中都提交这份文件的完整内容，另提交说明卡。平台按单份源码评测，不上传多个文件或压缩包。骨架和公开测试均使用模块名 `hanoi_submission`；在本机测试时，将 `solution.py` 复制为 `hanoi_submission.py`，把各题公开测试放在各自目录分别运行，避免同名测试文件覆盖。

自动测试满分只代表通过测试，不代替递归实现和最优性的说明。助教可能抽查代码解释。个人作业 1 整体遵守 Level 0。

提交与自查

小作业 3：提交六道 A 段代码题、7-5 说明卡、约瑟夫环代码题与说明卡、C 段答题卡。个人作业 1：三个代码题共用完整源码，另交汉诺塔说明卡。每份作业所有题都提交后，下载各自的评测报告交到 Canvas；源码与文字答案随报告保存，无需新建附件上传方式。

- ☐ 零、空字符串、空列表与空面额列表均已检查。
- ☐ `flatten` 与硬币题不修改输入，函数签名与骨架一致。
- ☐ 7-5 记录了 1001 位输入的实际运行环境和结果。
- ☐ 约瑟夫环的报数起点、编号转换与对照核验均有说明。
- ☐ C 段有实际命令、输出和管道解释。
- ☐ 汉诺塔生成函数返回移动列表，核验函数区分合法与最优。
- ☐ 个人作业的源码与说明一致，未使用 AI 生成或补全代码。