



上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

Introduction to Computation (CS1602) · Lecture 7

Recursion 递归

Instructor: Tao Huang

Part 3: Problem Solving with AI · Fall 2026



COURSE STRUCTURE

Part III · Problem Solving with AI

Problem decomposition 问题分解

Code review and verification 代码审查与验证

| Containers and Nested Data 嵌套数据

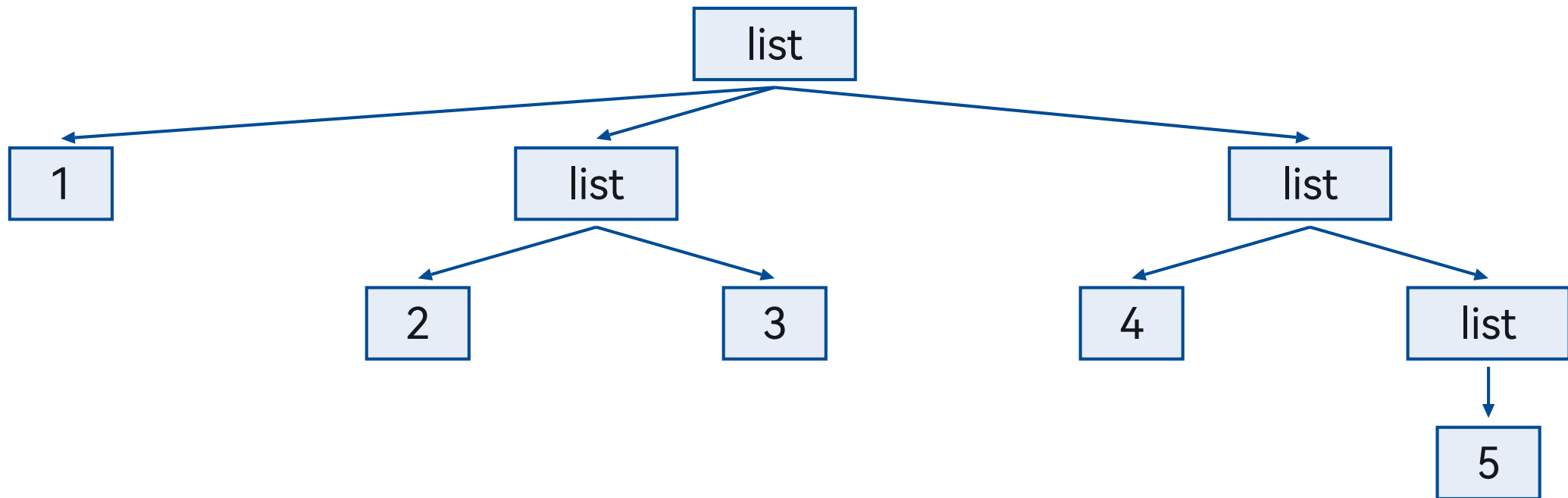
- **List 列表**: an ordered sequence
- **Dictionary 字典**: keys map to values
- **Nesting 嵌套**: a container can contain another container

[1, [2, 3], [4, [5]]]

| **Three outer items, five integers.** How can every integer be processed?

Nested Records 嵌套记录

How can every integer be included when nesting depth varies?



[1, [2, 3], [4, [5]]]

Quiz 1 · Function Calls



扫码作答 · 一题

1 普通函数的调用与返回顺序

Predict the output order, then submit once.

<https://taohuang.info/cs1602/zh/quiz/7/>

| Call and Return Order 调用与返回顺序

code

```
1  def inner() -> int:
2      print("B")
3      return 2
4
5  def outer() -> int:
6      print("A")
7      result = inner()
8      print("C")
9      return result + 1
10
11 print(outer())
```

output

```
A
B
C
3
```



| Lecture Structure

- | | | |
|----------|-------------------------------|---|
| 1 | Calls and termination | 调用与终止 · local state, return order and reachable base cases |
| <hr/> | | |
| 2 | Data and subproblems | 递归结构与重复计算 · nested lists, gcd, Fibonacci and power |
| <hr/> | | |
| 3 | Divide and conquer | 分治 · binary search and merge sort |
| <hr/> | | |
| 4 | Hanoi and backtracking | 汉诺塔、枚举与回溯 · instruction order and legal partial choices |
| <hr/> | | |

I • Recursive Functions

A caller resumes after a callee returns.

Can the callee be the same function?

| Factorial Recurrence 阶乘递推关系

Nonnegative integer n · input domain 非负整数

Concrete example 具体例子

$$3! = 3 \times 2! = 3 \times 2 \times 1$$

$$2! = 2 \times 1!; 1! = 1 \times 0!$$

General rule 递推关系

$$n! = n \times (n - 1)! \text{ for } n \geq 1$$

$$0! = 1 \cdot \text{direct answer 基准}$$

| 先核对输入范围与零的阶乘，再分析子调用返回后如何乘以当前 n 。

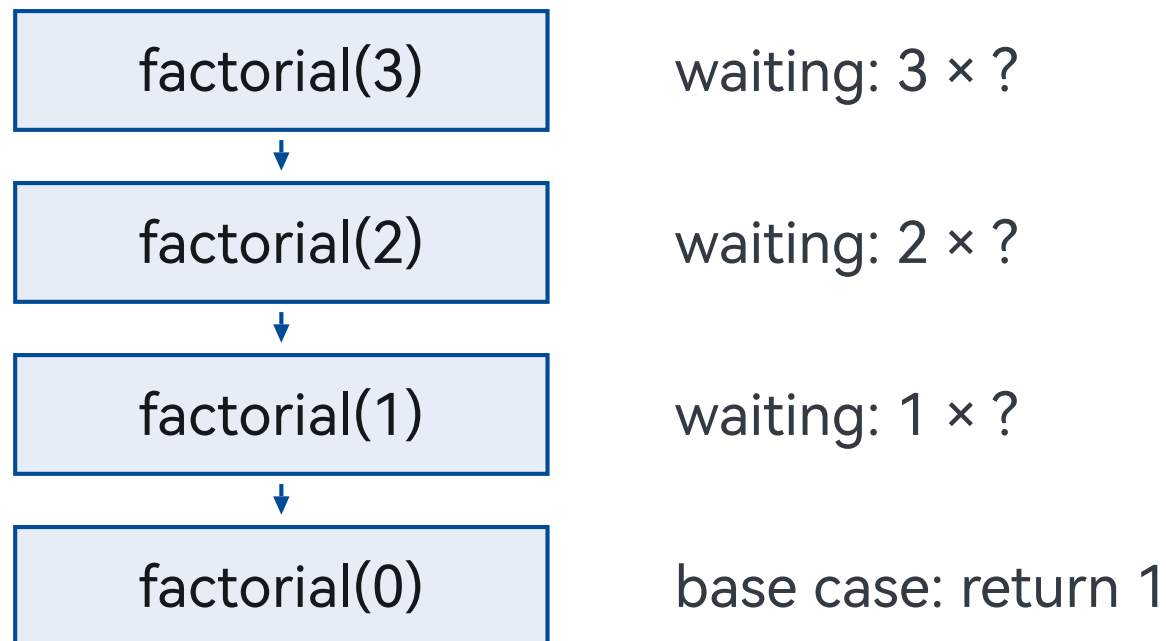
| Recursive Factorial 递归阶乘函数

What does factorial(3) return? 输入: 非负整数

```
1 def factorial(n: int) -> int:
2     if n == 0:
3         return 1
4     return n * factorial(n - 1)
5
6 print(factorial(3))
```

| The Call Chain 调用链

Call order



Each caller resumes after its callee returns.

| Local Variables 局部变量

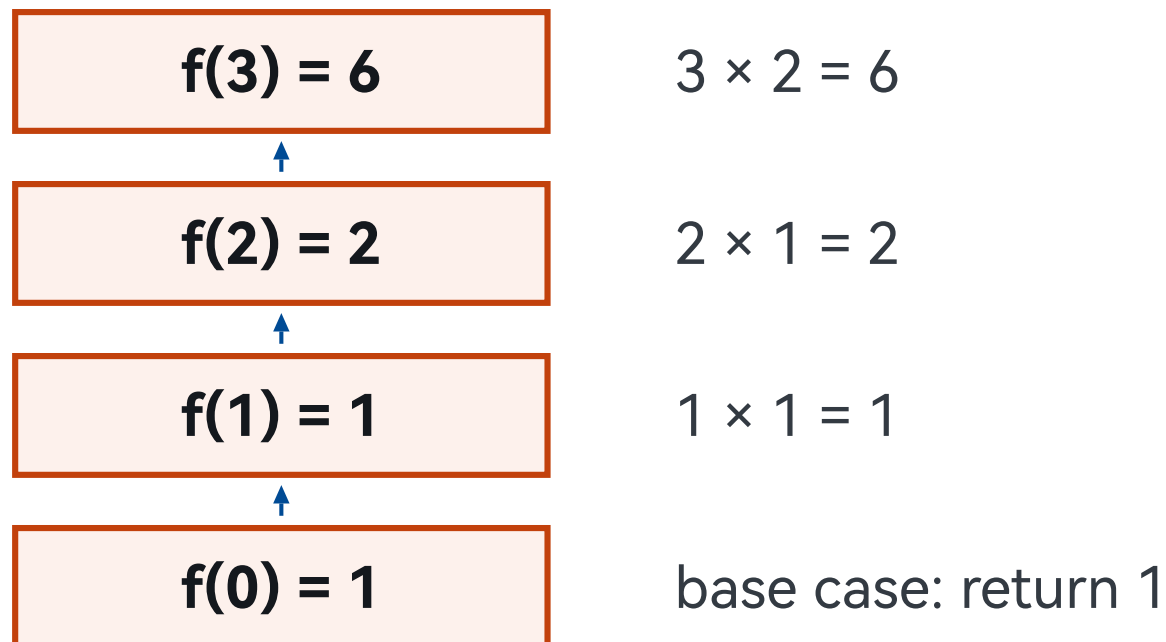
factorial(0)	local n = 0	current
factorial(1)	local n = 1	waiting: 1 × ?
factorial(2)	local n = 2	waiting: 2 × ?
factorial(3)	local n = 3	waiting: 3 × ?

Logical call frames; not a CPython memory layout.

Logical call frames 调用帧示意; not a CPython memory layout.

| The Return Chain 返回顺序

Return order



Each caller resumes after its callee returns.

| Tracing Recursive Calls 递归调用追踪

code

```
1  def factorial(n: int, depth: int = 0) -> int:
2      pad = "  " * depth
3      print(f"{pad}enter {n}")
4      if n == 0:
5          result = 1
6      else:
7          result = n * factorial(n - 1, depth + 1)
8      print(f"{pad}return {result}")
9      return result
10
11 print(factorial(3))
```

output

```
enter 3
  enter 2
    enter 1
      enter 0
        return 1
      return 1
    return 2
  return 6
6
```



| Base Case and Progress 基准与终止度量

Check 检查	Meaning 含义	Factorial example 阶乘
Base case 基准	Complete the task directly; no recursive call	<code>factorial(0) → 1</code>
Progress 终止度量	A lower-bounded measure approaches the base case	3 → 2 → 1 → 0

- **Measure the remaining work:** a parameter, length or unfilled rows.
- **Not every parameter must decrease:** placed queens increase while remaining rows decrease.

| Nonterminating Recursion 非终止递归

No base case 没有基准

```
def bad(n: int) -> int:  
    return bad(n - 1)
```

bad(3)

- **No direct return** for any input

No progress 参数不变

```
def bad(n: int) -> int:  
    if n == 0:  
        return 0  
    return bad(n)
```

bad(3)

- **n stays 3**; the base case is unreachable

| Both independently checked examples raise **RecursionError**.

| An Unreachable Base Case 不可达基准

Will n reach 0? 参数依次为 3, 1, -1, -3, ...

```
1 def down(n: int) -> int:
2     if n == 0:
3         return 0
4     return down(n - 2)
5
6 print(down(3))
```

I Reasoning About Recursion

- | | | |
|-------|----------------|--|
| 1 | Task 任务 | State what the function must return or do . |
| <hr/> | | |
| 2 | Base case 基准 | Check that the direct result is correct. |
| <hr/> | | |
| 3 | Subproblem 子问题 | Assume its specified result is correct. |
| <hr/> | | |
| 4 | Combination 组合 | Derive the current result from that result. |
| <hr/> | | |
| 5 | Progress 终止 | Prove that a base case can be reached . |
| <hr/> | | |

| Recursion Depth 递归深度

Finite but too deep 有限但过深

```
def countdown(n: int) -> int:
    if n == 0:
        return 0
    return countdown(n - 1)

print(countdown(5000))
```

Actual exception 实际异常

```
RecursionError:
maximum recursion depth exceeded
```

- **Finite:** $5000 \rightarrow 4999 \rightarrow \dots \rightarrow 0$
- **Too deep:** the interpreter stops before the base case.
- **Depth limit:** not a measurement of disk space or RAM.

Build: **Python 3.12.9** · `sys.getrecursionlimit()` → **1000**. Consider a loop or explicit state.

| Recursion and Iteration 递归与循环

Recursion 递归

```
def factorial(n: int) -> int:
    if n == 0:
        return 1
    return n * factorial(n - 1)

print(factorial(10))
```

output

3628800

Iteration 循环

```
def factorial_loop(n: int) -> int:
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

print(factorial_loop(10))
```

output

3628800

II • Recursive Data and Subproblems

A call has its own local state.

How does problem structure determine the next call?



| Type Checks 类型检查

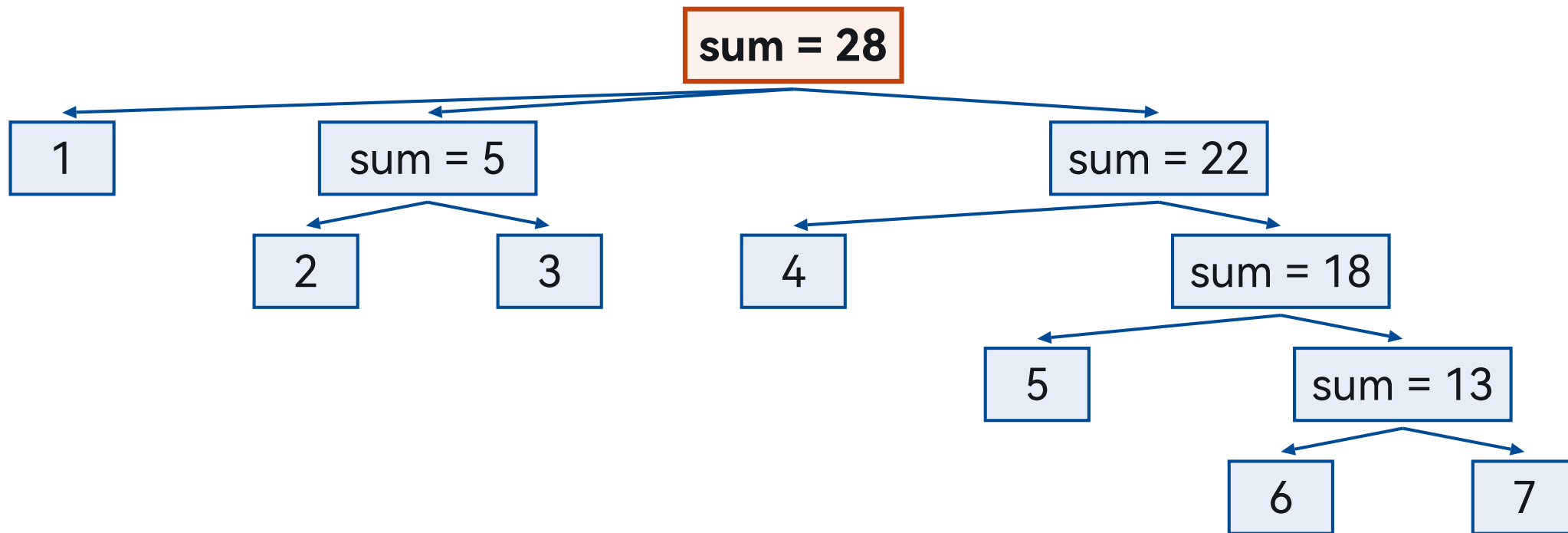
code

```
1 for x in [3, [1, 2], []]:  
2     print(isinstance(x, list))
```

output

```
False  
True  
True
```

| Summing Nested Lists 嵌套列表求和



Integer leaves: $1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$.

| Recursive Sum

code

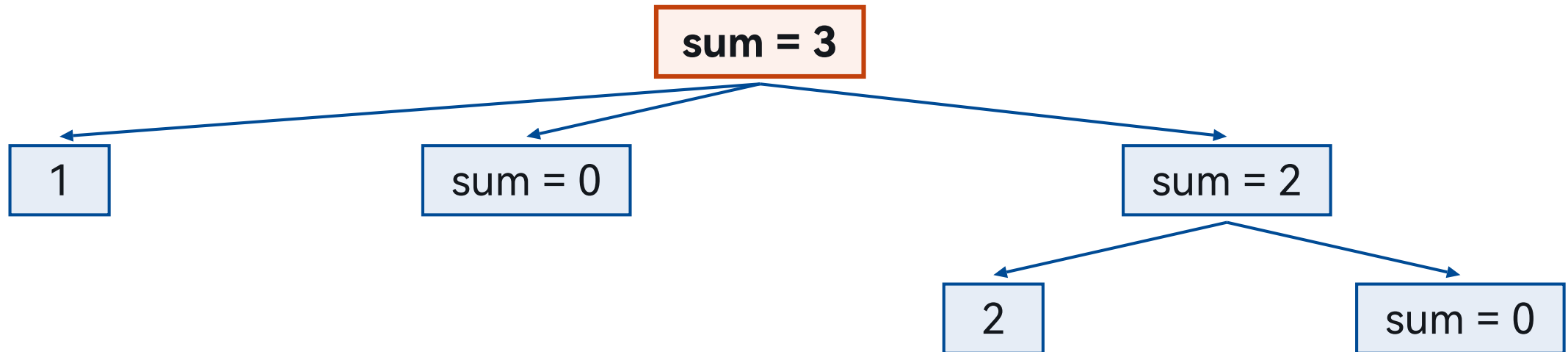
```
1  def deep_sum(items: list) -> int:
2      total = 0
3      for x in items:
4          if isinstance(x, list):
5              total += deep_sum(x)
6          else:
7              total += x
8      return total
9
10 print(deep_sum([1, [2, 3], [4, [5, [6, 7]]]]))
```

output

28

| Empty Lists and Nested Calls

Predict `deep_sum([1, [], [2, []]])` before following the sums.



Empty lists return 0; total = 3.

| Euclid's Algorithm 欧几里得算法

$\gcd(a, b) = \gcd(b, a \% b)$ when $b > 0$

Current pair 当前参数	Next pair 下一参数	Remainder 余数
(48, 18)	(18, 12)	$48 \% 18 = 12$
(18, 12)	(12, 6)	$18 \% 12 = 6$
(12, 6)	(6, 0) → 6	$12 \% 6 = 0$

Nonnegative integers; $\gcd(a, 0) = a$. The second argument strictly decreases, not necessarily the first.

| Recursive gcd 最大公约数

code

```
1 def gcd(a: int, b: int) -> int:
2     if b == 0:
3         return a
4     return gcd(b, a % b)
5
6 print(gcd(48, 18))
7 print(gcd(1071, 462))
8 print(gcd(0, 5))
```

output

```
6
21
5
```

Fibonacci Recurrence 斐波那契递推

code

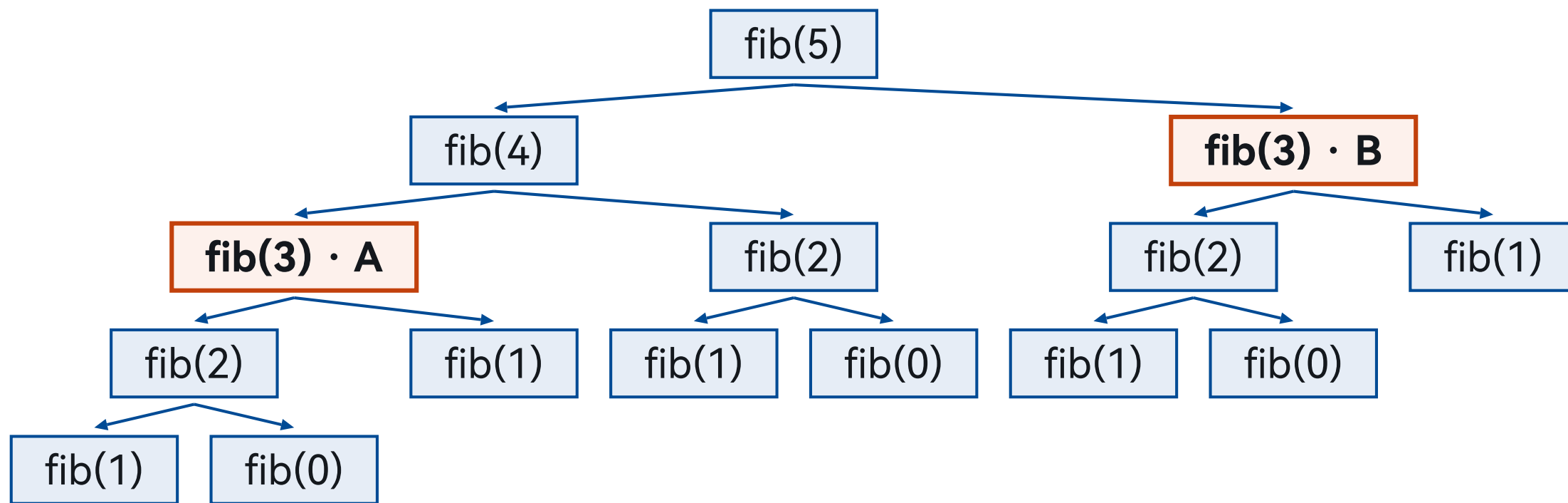
```
1 def fib(n: int) -> int:
2     if n <= 1:
3         return n
4     return fib(n - 1) + fib(n - 2)
5
6 for n in range(10):
7     print(fib(n), end=" ")
```

output

0 1 1 2 3 5 8 13 21 34

Repeated Subproblems 重复子问题

How many separate calls to `fib(3)` appear in `fib(5)`?



15 calls; `fib(3) · A` and `fib(3) · B` are separate calls.

| Counting Function Calls 调用次数统计

code

```
1 def fib_counted(n: int) -> tuple[int, int]:
2     if n <= 1:
3         return n, 1
4     left, c1 = fib_counted(n - 1)
5     right, c2 = fib_counted(n - 2)
6     return left + right, 1 + c1 + c2
7
8 for n in [5, 10, 20, 25]:
9     value, calls = fib_counted(n)
10    print(n, value, calls)
```

output

```
5 5 15
10 55 177
20 6765 21891
25 75025 242785
```

| Fibonacci Call Counts

n	fib(n)	Function calls
5	5	15
10	55	177
20	6,765	21,891
25	75,025	242,785

Counts include the initial call and every base-case call.

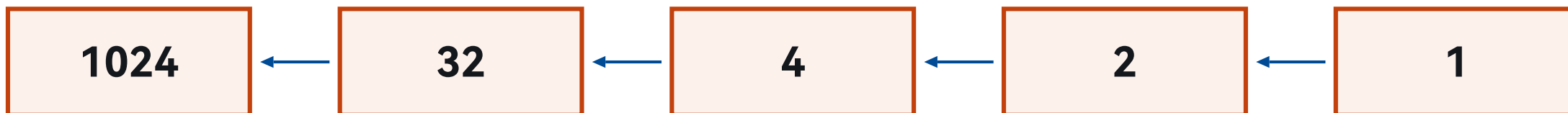
Measured calls 实际调用次数; includes base-case calls.

| Fast Exponentiation 快速幂

Exponent n : each call uses $n // 2$



Returned value for $a = 2$



One recursive result per call; half is used twice.

| Recursive Power

code

```
1 def power(a: int, n: int) -> int:
2     if n == 0:
3         return 1
4     half = power(a, n // 2)
5     if n % 2 == 0:
6         return half * half
7     return half * half * a
8
9 print(power(2, 10))
10 print(power(3, 5))
```

output

```
1024
243
```

| Result Reuse 返回值复用

The even-exponent branch 偶数指数分支

One call 一次调用

```
half = power(a, n // 2)
return half * half
```

Two calls 两次调用

```
return (power(a, n // 2)
        * power(a, n // 2))
```

- **Compute once**
- **Use the returned value twice**
- **Both calls execute**
- **The same subtask is repeated**

| 比较计算量时，要区分复用返回值与再次调用；Python 不会自动缓存相同调用。

III · Divide and Conquer 分治

Smaller tasks reduce the current problem.

How can sorted subranges be used?

| Binary Search Preconditions 二分查找前提

Sorted input; inclusive range [lo, hi]. 查找目标: 15

target = 15

Sorted input; inclusive range [lo, hi]

0	1	2	3	4	5	6	7
1	3	5	7	9	11	13	15

| lo = 0 | | | | | | | hi = 7 |

Indices 0–7; values increase from left to right.

| Search Range 查找区间

target = 15

Blue: current interval; underline: mid

		0	1	2	3	4	5	6	7
[0, 7]	mid = 3	1	3	5	<u>7</u>	9	11	13	15
[4, 7]	mid = 5	1	3	5	7	9	<u>11</u>	13	15
[6, 7]	mid = 6	1	3	5	7	9	11	<u>13</u>	15
[7, 7]	mid = 7	1	3	5	7	9	11	13	<u>15</u>

Found: return index 7.

| Recursive Binary Search

One recursive call; -1 for an empty range. 闭区间

```
1  def search_range(nums: list[int], target: int,  
2      lo: int, hi: int) -> int:  
3      if lo > hi:  
4          return -1  
5      mid = (lo + hi) // 2  
6      if nums[mid] == target:  
7          return mid  
8      if nums[mid] < target:  
9          return search_range(nums, target, mid + 1, hi)  
10     return search_range(nums, target, lo, mid - 1)
```

| Unsuccessful Search 查找未命中

Target 8: why does the final range contain no elements?

target = 8

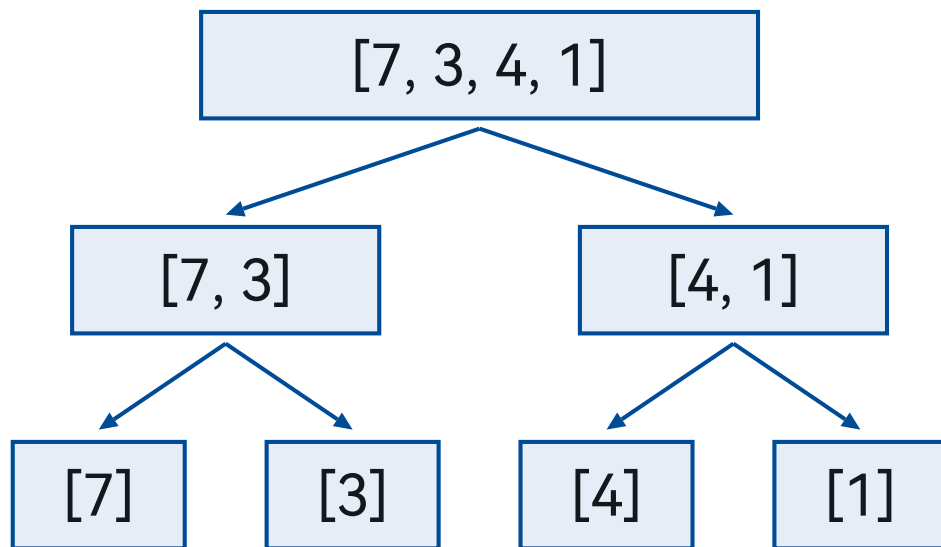
Blue: current interval; underline: mid

		0	1	2	3	4	5	6	7
[0, 7]	mid = 3	1	3	5	<u>7</u>	9	11	13	15
[4, 7]	mid = 5	1	3	5	7	9	<u>11</u>	13	15
[4, 4]	mid = 4	1	3	5	7	<u>9</u>	11	13	15
[4, 3]	empty	1	3	5	7	9	11	13	15

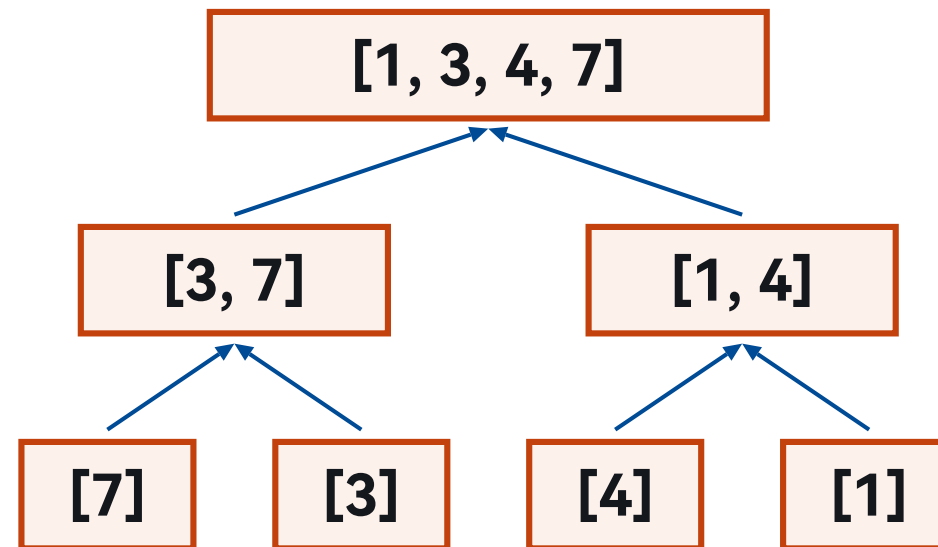
lo > hi: empty interval, return -1.

Merge Sort 归并排序

Split



Return



Base cases: one-element lists are already sorted.

Merging Sorted Lists 有序列表归并

Which unused element should be appended next? 合并有序列表

a = [3, 7] b = [1, 4]

Before	Compare	Append	Merged prefix
i=0, j=0	3 vs 1	1 (b[j])	[1]
i=0, j=1	3 vs 4	3 (a[i])	[1, 3]
i=1, j=1	7 vs 4	4 (b[j])	[1, 3, 4]
i=1, j=2	b exhausted	7 (a[i])	[1, 3, 4, 7]

i and j point to the first unused item in each input.

| The Merge Function

Excerpt · compare candidates

```
def merge(  
    a: list[int], b: list[int]  
) -> list[int]:  
    result = []  
    i = 0  
    j = 0  
    while (i < len(a)  
           and j < len(b)):  
        if a[i] <= b[j]:  
            result.append(a[i])  
            i += 1  
        else:  
            result.append(b[j])  
            j += 1
```

Excerpt · finish the same function

```
    result.extend(a[i:])  
    result.extend(b[j:])  
    return result
```

a and b must be sorted
Inputs are not modified

- **Both inputs sorted** before the loop
- **One side exhausted**: append the remainder
- **New result**: neither input is modified

| Recursive Merge Sort

Excerpt · see complete program

```
def merge_sort(
    nums: list[int]
) -> list[int]:
    if len(nums) <= 1:
        return nums.copy()
    mid = len(nums) // 2
    left = merge_sort(nums[:mid])
    right = merge_sort(nums[mid:])
    return merge(left, right)
```

Actual output 实跑输出

[1, 2, 3, 3, 4, 5, 7]

- **Requires merge(a, b)** from the complete program.
- **Base case returns a copy**; neither input is changed.
- **Two halves** are sorted, then merged.

[Complete runnable example](#) 完整程序

| Recursive Algorithm Structure

Algorithm 算法	Subproblem 子问题	After the call 返回后
Binary search	One chosen subrange	Pass back the matching index
Merge sort	Both halves	Merge the sorted results
Power	One smaller exponent	Reuse the computed value

| 比较子问题的数量、规模与组合步骤，不能只看递归语法。

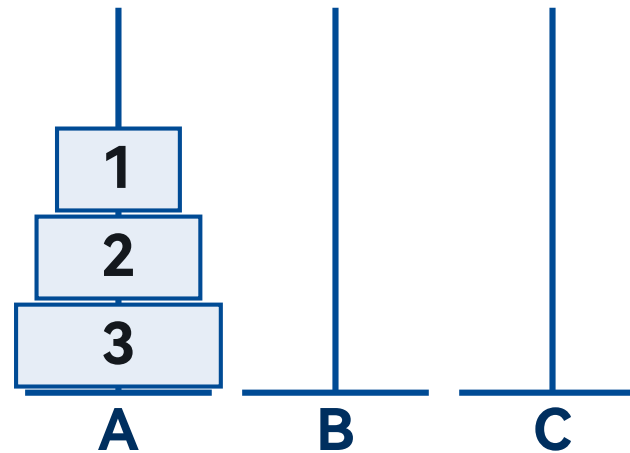


IV · Towers of Hanoi 汉诺塔

Previous examples returned computed values.

How can calls organize a sequence of instructions?

Hanoi Rules



**One disk
per move**

Top disk only

No larger disk
on a smaller disk

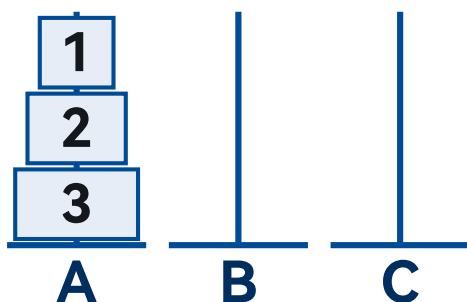
Goal: A → C

Disk number increases with disk size.

Hanoi Decomposition 汉诺塔任务分解

Start

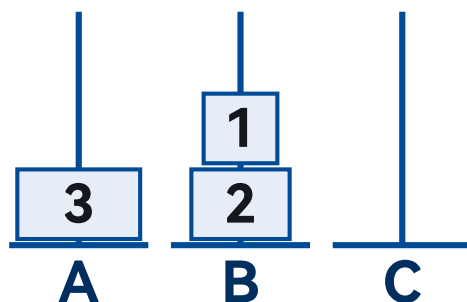
3 disks: $A \rightarrow C$



source: A
target: C
spare: B

Subtask 1

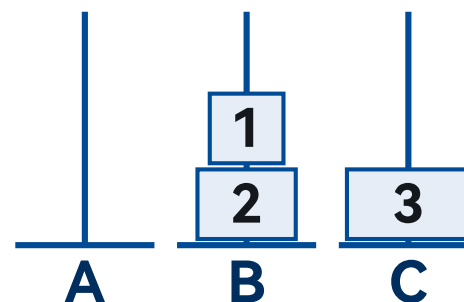
2 disks: $A \rightarrow B$



source: A
target: B
spare: C

Move disk 3

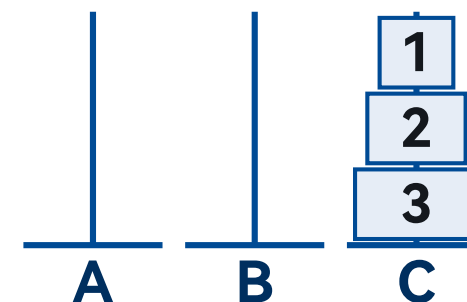
disk 3: $A \rightarrow C$



source: A
target: C
no spare needed

Subtask 2

2 disks: $B \rightarrow C$



source: B
target: C
spare: A

Each subtask transfers two disks; the middle step moves one disk.



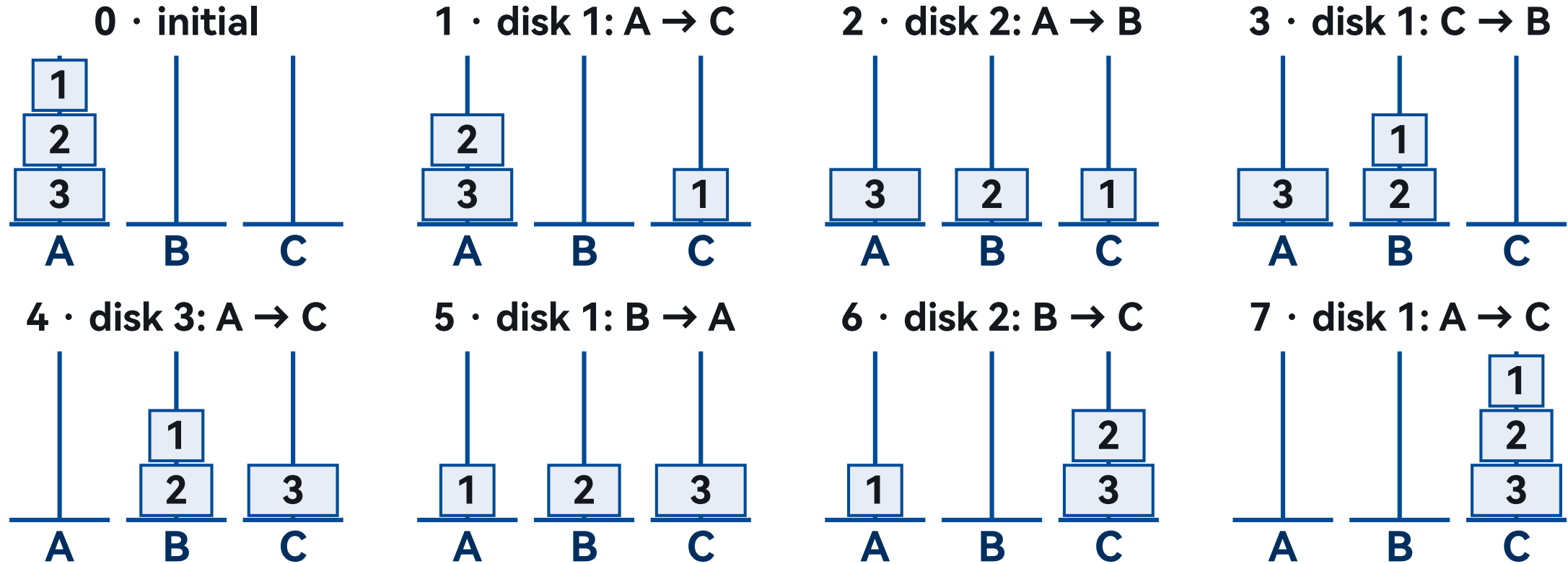
| Source, Target and Spare 三种任务角色

Role 当前角色	Meaning 含义	hanoi(3, A, C, B)
source	Start pillar 起点	A
target	Destination 终点	C
spare	Temporary pillar 备用柱	B
1 First subtask	hanoi($n - 1$, source, spare, target)	
2 Largest disk	Move disk n : source → target	
3 Second subtask	hanoi($n - 1$, spare, target, source)	

| Recursive Hanoi Instructions

```
1  def hanoi(n: int, source: str,  
2      target: str, spare: str) -> None:  
3      if n == 0:  
4          return  
5      hanoi(n - 1, source, spare, target)  
6      print(f"disk {n}: {source} -> {target}")  
7      hanoi(n - 1, spare, target, source)  
8  
9  hanoi(3, "A", "C", "B")
```

Three-Disk Move Sequence



| Hanoi Move Count 移动次数

$$T(n) = 2T(n - 1) + 1 \cdot T(0) = 0$$

- **Two** transfers of $n - 1$ disks + **one** move of the largest disk

Disks 盘数 n	Moves 移动次数 $T(n)$
1	1
2	3
3	7
4	15
5	31

| $T(n) = 2^n - 1$ counts moves, not function calls. Maximum depth grows with n .

I Output and Return Values 输出与返回值

code

```
1 def hanoi(n: int, source: str,  
2         target: str, spare: str) -> None:  
3     if n == 0:  
4         return  
5     hanoi(n - 1, source, spare, target)  
6     print(f"disk {n}: {source} -> {target}")  
7     hanoi(n - 1, spare, target, source)  
8  
9 result = hanoi(1, "A", "C", "B")  
10 print(result)
```

output

```
disk 1: A -> C  
None
```

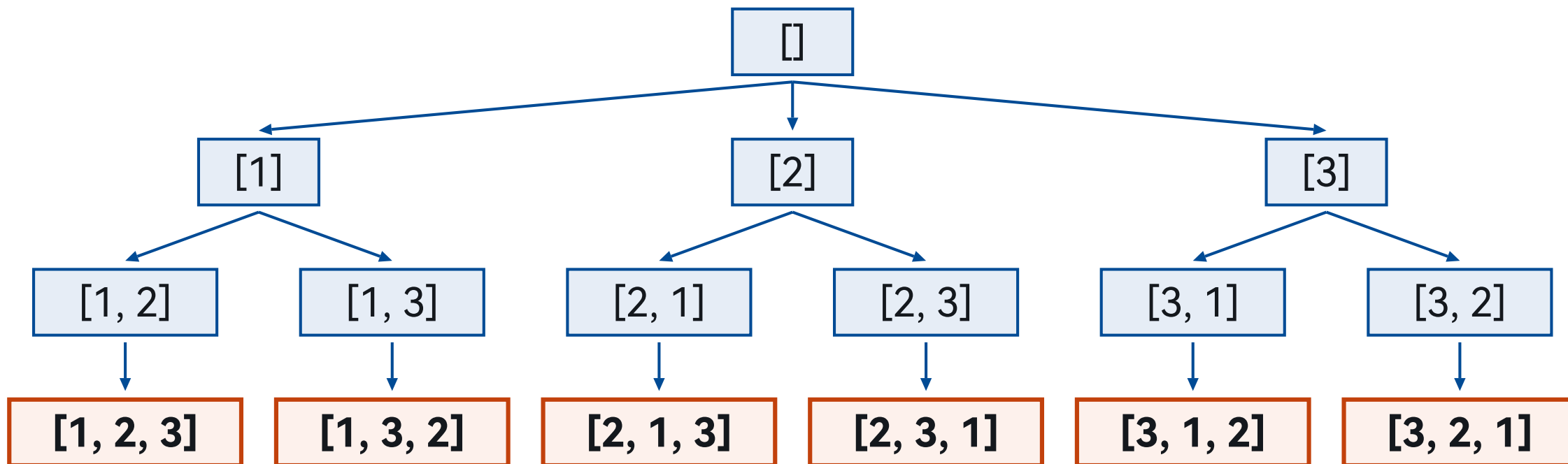


V · Enumeration and Backtracking 枚举与回溯

Calls can organize more than arithmetic.

How can a legal partial choice be extended?

Permutation Choices 排列选择



A node records the chosen prefix; six leaves give six permutations.

| Recursive Permutations

code

```
1 def permutations(  
2     items: list[int]  
3 ) -> list[list[int]]:  
4     if len(items) <= 1:  
5         return [items.copy()]  
6     result = []  
7     for i in range(len(items)):  
8         rest = items[:i] + items[i + 1:]  
9         for p in permutations(rest):  
10             result.append([items[i]] + p)  
11     return result  
12  
13 p = permutations([1, 2, 3])  
14 print(p[:2])  
15 print(len(p))
```

output

```
[[1, 2, 3], [1, 3, 2]]  
6
```

| Partial Solutions 部分解

	0	1	2	3
0		Q		
1				Q
2	Q			
3			Q	

queens = [1, 3, 0, 2]

`queens[row] = column`

row 0 → column 1

row 1 → column 3

row 2 → column 0

row 3 → column 2

One queen per row, column and diagonal.

`queens[row] = column`; zero-based indices 零基下标。

I Queen Placement 皇后位置检查

Same column or diagonal? 同列: $c == col$; 对角线: 距离相等

```
1 def is_safe(queens: list[int], col: int) -> bool:
2     row = len(queens)
3     for r, c in enumerate(queens):
4         if c == col or abs(row - r) == abs(col - c):
5             return False
6     return True
```

| Counting Queens Solutions

Excerpt · see complete program

```
def count_queens(  
    n: int, queens: list[int]  
) -> int:  
    if len(queens) == n:  
        return 1  
    total = 0  
    for col in range(n):  
        if is_safe(queens, col):  
            total += count_queens(  
                n, queens + [col])  
    return total
```

Actual output 实跑输出

2
92

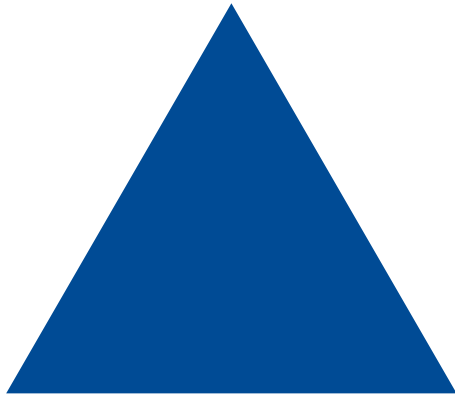
- **Requires is_safe:** conflicts are checked before recursion.
- **queens + [col]** creates a new partial list.
- **Pruning 剪枝:** invalid branches stop early.

[Complete runnable example 完整程序](#)

| Self-Similar Fractals 自相似分形

$n = 0$

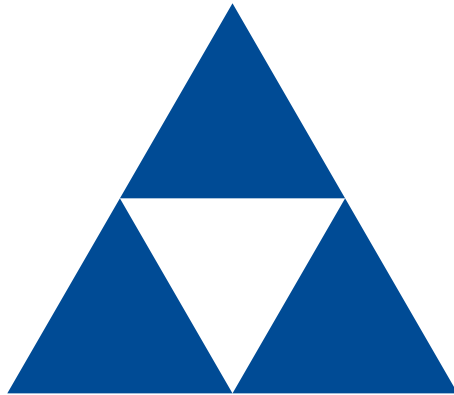
base triangle



1 triangle

$n = 1$

$3 \times$ order 0



3 triangles

$n = 2$

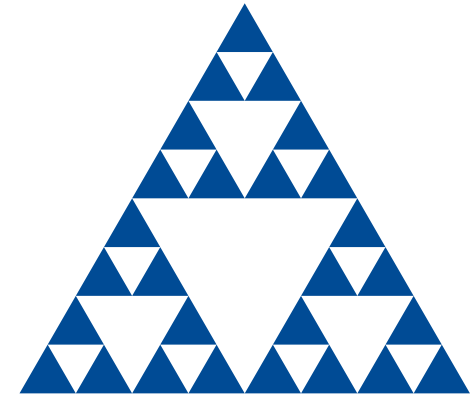
$3 \times$ order 1



9 triangles

$n = 3$

$3 \times$ order 2



27 triangles

Three copies of the previous order, each at half scale.

| Recursion Design and Choice

- 1 **Define 设计** Task → base case → progress → combination

- 2 **Match the structure** **Nested or branching tasks** may suit recursion.

- 3 **Check repeated work** Identify **repeated subproblems** and opportunities for reuse.

- 4 **Check scale and state** **Depth, input size, state and storage** all matter.

| Sequential accumulation often suits a loop; other conversions may require an **explicit stack**.

Quiz 2 · Recursive Reasoning



扫码作答 · 四题

- 1 基准情形能否到达
- 2 每次调用的局部参数
- 3 同一子问题的重复调用
- 4 汉诺塔的三个任务角色

All four topics have been covered. Submit once.

<https://taohuang.info/cs1602/zh/quiz/7/>

Quiz Review: Termination and Local State

Question 题目	Reasoning 推理	Outcome 结果
Q2 · reachability	$3 \rightarrow 1 \rightarrow -1 \rightarrow -3 \rightarrow \dots$ Zero is never reached.	RecursionError
Q3 · local n	$\text{total}(0) \rightarrow 0$ $\text{total}(1) \rightarrow 1 + 0$ $\text{total}(2) \rightarrow 2 + 1$	$0 \rightarrow 1 \rightarrow 3$

A decreasing number may skip the base case. Each call keeps **its own local n**.

I Quiz Review: Reuse and Hanoi Roles

Q4 · result reuse

```
half = power(a, n // 2)
return half * half
```

Reuse a value, not two calls

- **One calculation, two uses** of its value

Q5 · first Hanoi subtask

```
hanoi(3, "A", "C", "B")
```

Before moving disk 3:
hanoi(2, "A", "B", "C")

- **A → B**: move the two smaller disks first.
- **C** is the spare pillar in this subtask.



| Lab 7 · Recursion and Tools

Part 实验段	Learning tasks 任务	Checks 核对
A · recursion	Digits, reversal, palindrome, flatten, digit sum and coin combinations	Zero / empty inputs; recursive and loop digit sums
B · Josephus	Simulate and recurse 约瑟夫环	Small cases and numbering conventions
C · tools	grep / find / wc sort / uniq / pipes	Search and summarize; submit operation records

Each code or explanation goes to its own entry. Large digit inputs also require **depth checks**.

| Solo Assignment 1 · Restricted Hanoi

Movement rule 规则	Permitted moves 移动	End-to-end count 次数
Ordinary Hanoi	Any pair of pillars	$2^n - 1$
Adjacent pegs only	$A \leftrightarrow B \leftrightarrow C$; no direct $A \leftrightarrow C$	$3^n - 1$

- **Move lists**, not just a count
- **Legality and final-state checks**; explain the recurrence

| Course grade: **5%** · due **10 Nov, 23:59 GMT+8**.

| Code Review and Verification

- **Same task, smaller or nested input:** recursive decomposition
- **Separate local state:** each call keeps its own variables
- **Combination:** returned results or ordered actions

L08 · Code Review and Verification

- **Input assumptions → base cases → progress → results and state changes**

| How can these claims be checked against a program?



AFTER CLASS

Reference 参考

Complete traces, implementations and checks

Further recurrences, drawing and iterative versions

| Factorial Trace

Complete output 完整输出

```
enter 3
  enter 2
    enter 1
      enter 0
        return 1
      return 1
    return 2
  return 6
6
```

- **enter**: begin a new call
- **return**: finish that call
- **Indentation**: one level deeper per indent
- **Returned values**: 1, 1, 2, 6
- **Last 6**: the outer print()

| Explicit Stack Traversal 显式栈遍历

code

```
1  def sum_stack(items: list) -> int:
2      pending = items.copy()
3      total = 0
4      while pending:
5          x = pending.pop()
6          if isinstance(x, list):
7              pending.extend(x)
8          else:
9              total += x
10     return total
11
12 print(sum_stack([1, [2, 3], [4, [5]]]))
```

output

15

| Binary Search: Complete Interface

Excerpt · see complete program

```
def binary_search(  
    nums: list[int], target: int  
) -> int:  
    hi = len(nums) - 1  
    return search_range(  
        nums, target, 0, hi)  
  
# search_range: page 38
```

Actual output 实跑输出

```
[] 1 -1  
[3] 3 0  
[3] 2 -1  
[2, 2, 2] 2 1
```

- **Requires search_range**; the complete example defines both functions.
- **Sorted input** is required.
- **Any matching index** is allowed for duplicates.

[Complete runnable example 完整程序](#)

I Merge: Comparison Loop

Code / excerpt 代码或片段

```
def merge(
    a: list[int], b: list[int]
) -> list[int]:
    result = []
    i = 0
    j = 0
    while (i < len(a)
           and j < len(b)):
        if a[i] <= b[j]:
            result.append(a[i])
            i += 1
        else:
            result.append(b[j])
            j += 1
```

- **Sorted prefix:** result stays sorted.
- **Consumed:** a[:i] and b[:j].
- **Next candidates:** a[i] and b[j].
- **Choose the smaller** unused element.

| Merge Sort: Complete Example

Excerpt · see complete program

```
# End of merge:
result.extend(a[i:])
result.extend(b[j:])
return result

def merge_sort(
    nums: list[int]
) -> list[int]:
    if len(nums) <= 1:
        return nums.copy()
    mid = len(nums) // 2
    left = merge_sort(nums[:mid])
    right = merge_sort(nums[mid:])
    return merge(left, right)
```

Actual output 实跑输出

[1, 2, 3, 3, 4, 5, 7]

- **Complete merge** is required; this page shows its ending.
- **Empty** → []; duplicates are retained.
- **Inputs unchanged**; copies, slices and result lists use storage.

[Complete runnable example 完整程序](#)

| Permutation Verification 排列结果核验

Excerpt · see complete program

```
from itertools import (  
    permutations as ref_perm)  
  
items = [1, 1, 2]  
mine = permutations(items)  
ref = [list(p)  
        for p in ref_perm(items)]  
print(sorted(mine) == sorted(ref))  
  
# Requires our permutations.
```

Actual output 实跑输出

```
[] True  
[1] True  
[1, 2, 3] True  
[1, 1, 2] True
```

- **Requires our permutations** from the complete example.
- **Convert tuples to lists** before comparison.
- **Keep multiplicities**: a set alone loses repeated results.

[Complete runnable example](#) 完整程序

| Coin Combinations 硬币组合

Branch 分支	Subtask 子任务	Progress 终止度量
Use current coin	<code>ways(amount - coin, index)</code> Requires $\text{amount} \geq \text{coin}$; reuse is allowed.	Amount decreases
Skip current coin	<code>ways(amount, index + 1)</code> Do not return to earlier coins; order is ignored.	Fewer denominations remain

- **amount = 0** → **one combination**; positive amount with no coins → zero
- **Distinct positive denominations**; nonnegative amount

I Queens: Search and Verification

Method 方法	Candidates 候选		Conflict check 冲突检查		
Partial search 回溯	Extend a legal prefix		Before recursion; new lists need no in-place undo		
Full enumeration 枚举	$8! = 40320$ column permutations		Rows/columns distinct; check every diagonal pair		
n	4	5	6	7	8
Solutions 解数	2	10	4	40	92

| Both complete checks give **92** eight-queens arrangements.

| Text Sierpinski Triangle

Code / excerpt 代码或片段

```
def sierpinski(
    n: int
) -> list[str]:
    if n == 0:
        return ["*"]
    smaller = sierpinski(n - 1)
    pad = " " * (2 ** (n - 1))
    top = []
    bottom = []
    for line in smaller:
        top.append(
            pad + line + pad
        )
        doubled = line + " " + line
        bottom.append(doubled)
    return top + bottom
```

- **n = 0**: one star
- **smaller**: solve the previous order once
- **Top**: one centered subtriangle
- **Bottom**: two subtriangles
- **Result**: a list of text lines

| Sierpinski Output and Counts

n = 3 · actual output

```
      *
     * *
    *  *
   * * * *
  *      *
 * *      * *
*   *   *   *
* *   *   * *
* * * * * * * *
```

n	Rows 行数	Stars 星号
0	1	1
1	2	3
2	4	9
3	8	27
4	16	81

| Rows: 2^n · Stars: 3^n

| Hanoi: Minimum Move Count

1 Before largest disk Transfer all **$n - 1$ smaller disks** to the spare pillar.

2 Largest disk At least **one move** to the target.

3 After largest disk Transfer the **$n - 1$ smaller disks** onto it.

Lower bound 下界

Construction 构造达到

$$T(n) \geq 2T(n - 1) + 1$$

$$T(0) = 0; T(n) = 2^n - 1$$

The recursive algorithm uses exactly these stages. Extra largest-disk transfers cannot reduce the bound.

| Input Size 输入规模与递归上限

10^{1000} has **1001 digits** · a valid finite input

- **One digit per call** may exceed the current depth limit.
- **Accepted input sizes** must be documented.
- **Suitable loop or stack**: consider the scale of the task.

| Current build limit: **1000**. 实验须比较输入规模与实现限制，不以调高递归上限代替分析。

I Operation Costs 运算次数与整数大小

power(2, 10) · exponent chain: $10 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 0$

Exponent 指数	10	5	2	1
Multiplications 乘法	1	2	1	2

- **Six multiplications** across four non-base calls
- **power(2, 100)** · actual result:

Actual integer result 实际整数值

1267650600228229401496703205376

Fewer arithmetic operations do not make each operation fixed-time; **integer size also grows.**

| Binary Representation 二进制表示

code

```
1 def binary_text(n: int) -> str:
2     if n < 2:
3         return str(n)
4     return binary_text(n // 2) + str(n % 2)
5
6 for n in [0, 1, 2, 5, 13]:
7     print(n, binary_text(n))
```

output

```
0 0
1 1
2 10
5 101
13 1101
```

| Counting Set Bits 二进制中 1 的个数

code

```
1 def count_bits(n: int) -> int:
2     if n == 0:
3         return 0
4     return count_bits(n // 2) + n % 2
5
6 for n in [0, 1, 4, 5, 7]:
7     print(n, count_bits(n))
```

output

```
0 0
1 1
4 1
5 2
7 3
```



| Decimal Digits and Reversal

Decomposition 整数分解

Meaning 含义

$n \% 10$

Final digit 末位

$n // 10$

Remaining prefix 前缀

- **Digits(n) = 1** for $0 \leq n < 10$
- **Digits(n) = 1 + Digits(n // 10)** for $n \geq 10$
- **120 → 21** as an integer; string reversal can preserve the leading zero.

| 比较整数与字符串逆序的结果；同时检查反复统计前缀位数带来的额外计算。

| Digit Sum 数位和

Code / excerpt 代码或片段

```
# String traversal:
for ch in str(n):
    total += int(ch)

# Arithmetic loop:
while n > 0:
    total += n % 10
    n //= 10
```

- **Recursive decomposition** for nonnegative integers:
- **sum_digits(0) = 0**
- **sum_digits(n) = n % 10 + sum_digits(n // 10)**
- **Compare the same outputs** across both loops and recursion.
- **Large inputs** require a depth check.

| 循环片段须补齐 total 初始化与完整上下文；实验实现独立完成。

| A Third-Order Recurrence 三阶递推

code

```
1 def third_order(n: int) -> int:
2     if n <= 3:
3         return n
4     return (3 * third_order(n - 1)
5           - 2 * third_order(n - 2)
6           + third_order(n - 3))
7
8 values = [third_order(n)
9           for n in range(1, 9)]
10 print(values)
```

output

```
[1, 2, 3, 6, 14, 33, 77, 179]
```

| A Two-Dimensional Recurrence 二维递推

code

```
1  def grid_value(m: int, n: int) -> int:
2      if m == 0:
3          return 1
4      if n == 0:
5          return m
6      return (grid_value(m - 1, n)
7              + grid_value(m, n - 1)
8              - grid_value(m - 1, n - 1))
9
10 for m in range(5):
11     print([grid_value(m, n) for n in range(5)])
```

output

```
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[2, 2, 2, 2, 2]
[3, 3, 3, 3, 3]
[4, 4, 4, 4, 4]
```

Ackermann Function 阿克曼函数

code

```
1 def ackermann(m: int, n: int) -> int:
2     if m == 0:
3         return n + 1
4     if n == 0:
5         return ackermann(m - 1, 1)
6     return ackermann(m - 1, ackermann(m, n - 1))
7
8 print(ackermann(0, 1))
9 print(ackermann(1, 1))
10 print(ackermann(2, 2))
```

output

```
2
3
7
```

| Turtle Drawing 绘图基本操作

Code / excerpt 代码或片段

```
import turtle

pen = turtle.Turtle()
pen.forward(80)
pen.left(90)
pen.forward(40)
turtle.done()
```

- **forward(d)**: move along the current heading.
- **left(a)**: turn anticlockwise by a degrees.
- **Start**: (0, 0), heading 0°
- **After forward(80)**: (80, 0)
- **After left(90), forward(40)**: (80, 40), heading 90°

| **Local Python only**: requires a graphics window; no browser-run button.

| Recursive Star Drawing 递归五角星

Code / excerpt 代码或片段

```
def star(pen, size: float) -> None:
    if size <= 10:
        return
    for i in range(5):
        pen.forward(size)
        star(pen, size / 3)
        pen.left(216)

star(pen, 360)
turtle.done()
```

- **Size chain:** $360 \rightarrow 120 \rightarrow 40 \rightarrow 13\frac{1}{3} \rightarrow \text{stop}$
- **Five segments** per non-base call
- **780 drawn segments; 781 calls** including base cases
- **Position and heading** return to the starting state.
- **Pen stub verified;** not a GUI screenshot.

| Permutations by Insertion 插入法排列

code

```
1 def insert_perms(n: int) -> list[tuple[int, ...]]:
2     if n == 0:
3         return [()]
4     result = []
5     for p in insert_perms(n - 1):
6         for i in range(n):
7             result.append(p[:i] + (n,) + p[i:])
8     return result
9
10 for p in insert_perms(3):
11     print(p)
```

output

```
(3, 2, 1)
(2, 3, 1)
(2, 1, 3)
(3, 1, 2)
(1, 3, 2)
(1, 2, 3)
```

| Lexicographic Permutations 字典序排列

- | | | |
|---|--------------|-----------------------------------|
| 1 | Input 输入 | Distinct, sorted items |
| 2 | Branch order | Choose the first item in order. |
| 3 | Subproblem | Recurse on the ordered remainder. |

Actual lexicographic order 实际字典序

[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]

| Compare sequences at the **first differing position**. Not every result is internally increasing.

| Partial Permutations and Combinations

Order matters 排列

```
from itertools import (  
    permutations)  
  
items = [1, 2, 3]  
for p in permutations(items, 2):  
    print(p)
```

output

```
(1, 2)  
(1, 3)  
(2, 1)  
(2, 3)  
(3, 1)  
(3, 2)
```

Order ignored 组合

```
from itertools import (  
    combinations)  
  
items = [1, 2, 3]  
for c in combinations(items, 2):  
    print(c)
```

output

```
(1, 2)  
(1, 3)  
(2, 3)
```

| Displaying Nested Results with pprint

Excerpt · display only

```
from pprint import pprint

pprint(permutations([1, 2, 3]),
       width=20)

# Complete definition in the notes.
```

Actual formatted output 实际排版输出

```
[[1, 2, 3],
 [1, 3, 2],
 [2, 1, 3],
 [2, 3, 1],
 [3, 1, 2],
 [3, 2, 1]]
```

- **Display only**: no sorting, deduplication or new permutations
- **width=20** guides formatting; it is not a hard truncation limit.

| Fibonacci with an Array

code

```
1  def fib_array(n: int) -> int:
2      if n == 0:
3          return 0
4      values = [0] * (n + 1)
5      values[1] = 1
6      for i in range(2, n + 1):
7          values[i] = values[i - 1] + values[i - 2]
8      return values[n]
9
10 print(fib_array(0))
11 print(fib_array(20))
```

output

```
0
6765
```

| Fibonacci with Two Variables

code

```
1 def fib_pair(n: int) -> int:
2     previous = 0
3     current = 1
4     for i in range(n):
5         previous, current = current, previous + current
6     return previous
7
8 print(fib_pair(0))
9 print(fib_pair(20))
```

output

```
0
6765
```

| Loop State Updates

Recursive parameters 递归参数

```
def gcd(a: int, b: int) -> int:
    if b == 0:
        return a
    return gcd(b, a % b)

print(gcd(48, 18))
```

output

6

Loop state 循环状态

```
def gcd_loop(
    a: int, b: int
) -> int:
    while b != 0:
        a, b = b, a % b
    return a

print(gcd_loop(48, 18))
```

output

6

Adjacent-Peg Hanoi 相邻杆汉诺塔

$A \leftrightarrow B \leftrightarrow C$ · no direct $A \leftrightarrow C$ move

Step 步骤

Transfer 移动任务

1 · smaller disks

Transfer $n - 1$ disks $A \rightarrow C$.

2 · largest disk

Move disk n : $A \rightarrow B$.

3 · smaller disks

Transfer $n - 1$ disks $C \rightarrow A$.

4 · largest disk

Move disk n : $B \rightarrow C$.

5 · smaller disks

Transfer $n - 1$ disks $A \rightarrow C$.

n	0	1	2	3	4	5
Moves	0	2	8	26	80	242

$T(0) = 0$; $T(n) = 3T(n - 1) + 2 = 3^n - 1$. 核验移动序列，并证明三次小盘组转移都不可省略。