

Complexity, Exceptions and Testing

Every slide from the lecture, with what was said over it. The slides are rendered natively — the text is selectable, searchable, and prints cleanly.



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Introduction to Computation (CS1602) · Lecture 11

Complexity, Exceptions and Testing

Instructor: Tao Huang

Part 5: Robustness and the Real World · Fall 2026

1

Slide 1 · Complexity, Exceptions and Testing

L10结束于三个问题：实现要做多少工作、失败归谁处理、凭什么相信它是对的。本讲分别回答。

分两个45分钟；后面32页完整机制与旧例供课后查阅，不计入课堂必讲。



| Three Ways a Program Fails 程序的三种失败

Way 方式	Symptom 症状	This lecture 本讲的应对
Too slow	small data is fine, real data never finishes	growth and cost
It crashes	unexpected input ends the program	exception handling
Quietly wrong	no error, wrong answer	assertions and tests

| 三种失败对应三种证据，后面每一段回收一条。

2

Slide 2 · Three Ways a Program Fails 程序的三种失败

三者看起来无关，其实是同一个问题的三个面：凭什么说这段程序真的可用。

真实世界不承诺输入合法、数据干净、规模不大；这一讲给三种证据。



| Two Reports on the Same File 同一份文件的两种处理

Run 运行	Observed 观察
Small sample	both finish instantly
Full file	one finishes, one does not

| 先回答：凭什么说其中一个更好？What makes one of them better?

3

Slide 3 · Two Reports on the Same File 同一份文件的两种处理

先不给代码，只给两种实现在一小段和完整数据上的表现。让学生说出判断依据。

计时是最直接的做法，但下一页用一道题把这个依据变成可检验的问题。

I Quiz 1 · Counting Iterations



扫码作答 · 1 题

1 数一数内层循环一共执行几次

Submit this round once. 按已教规则推理。

<https://taohuang.info/cs1602/zh/quiz/11/>

开场只考已经学过的循环与函数，不涉及复杂度记号。

先让学生动手数一遍，第9页回顾原题分布，不重复提交。

I Lecture Structure



- 1 Cost and growth 成本与增长** measure work, not seconds
- 2 Failure and exceptions 失败与异常** expected problems and clear boundaries
- 3 Assertions and testing 断言与测试** checks written down as code
- 4 Five prime methods 素数五法** this week's solo assignment

5

Slide 5 · Lecture Structure

顺序由问题推进：先量化成本，再处理可预期的失败，然后检查不该发生的错误，最后用在本周作业上。

没有单独的AI政策页；第11周实验按Level 1要求声明。

Part V · Robustness and the Real World

本讲与第12周同属第五部分。本讲负责成本、异常与测试；第12周处理随机数、编码与文件。

先建立「工作量」这个可比较的量，再谈失败和证据。

I · Cost and Growth 成本与增长

先承认计时的作用，再说明它为什么不能单独作为比较依据。

替代方案是数步数：只依赖输入规模，不依赖机器。



Why a Stopwatch Is Not Enough 计时为什么不作为唯一依据

What changes 影响因素

Why it matters 为什么影响结论

Machine speed	the same code takes different seconds on different laptops
Background load	the number moves when the machine is busy
Interpreter version	a newer CPython optimizes different things
Input value	same size, different data, different work

计时只回答「这一次多久」，不回答「这个算法多贵」。

8

Slide 8 · Why a Stopwatch Is Not Enough 计时为什么不作为唯一依据

同一段代码在不同条件下差别明显，计时给出的结论不稳固。

所以需要有一个与机器无关的量：数基本操作的次数。下一页给出这个量。



Count Steps, Not Seconds 数步数

- `**`Basic operation
- `total = 0` once, for runs `n` times, the body once per pass, return once
- So this function takes about **`n + 2 steps`**, independent of the machine

9

Slide 9 · Count Steps, Not Seconds 数步数

回顾开场题：内层一共 6 次，就是把每一步加起来的結果。

步数只依赖输入规模。把 `n+2` 换成 `n2`，两者的差别在下一页用表讲清楚。



Big-O Keeps the Leading Term 只保留最高阶项

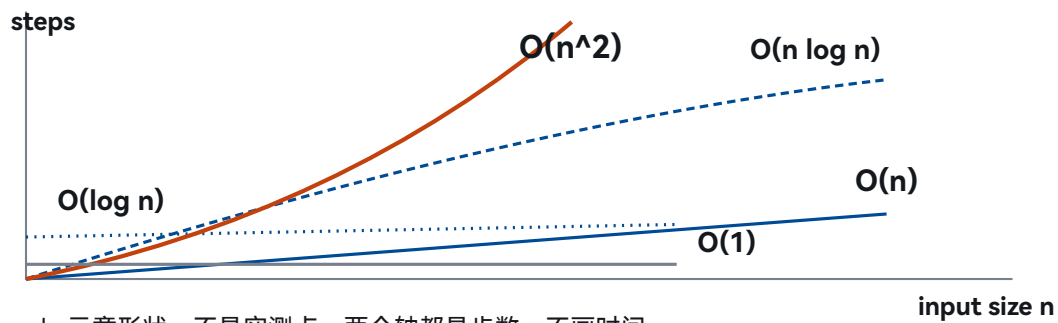
Expression 表达式	Kept 保留	Dropped 丢掉
$n^2 + 3n + 5$	n^2	$3n$ and 5 are lower order
$5n$	n	the constant factor 5
$100n$	n	a bigger constant is still a constant
3	1 (written $O(1)$)	no n at all

O(...) 描述形状；具体多少步要看实现。

只关心 n 很大时的趋势，所以去掉低阶项和常数系数。

这是描述增长形状，不是精确步数： $O(n)$ 不意味着正好 n 步。

I The Ladder of Growth 增长阶



Shapes only 示意形状，不是实测点；两个轴都是步数，不画时间。

$O(1)$ $a[i]$ · $O(\log n)$ binary search · $O(n)$ walking a list · $O(n \log n)$ sorted(a) · $O(n^2)$ a loop inside a loop · $O(2^n)$ naive recursive Fibonacci. 曲线只表示形状，不是实测点。

11

Slide 11 · The Ladder of Growth 增长阶

这张图是本讲的坐标系：阶数越高，曲线越早离开底部。右边一系列例子全是学生已经写过的代码，不是新知识。

下一页给一个本机的量级锚点，让他们能估算一个规模跑不跑得完。

I A Practical Anchor 一个可用的量级估计



Measurement 测量	n	Best of 3 取最小
Python loop	3,000,000 steps	0.043 s
Built-in sum	2,000,000 items	0.012 s
Steps per second	—	about 7×10^7

Measured on the build machine 本机构建时实测，同一进程内重复取最小值。

12

Slide 12 · A Practical Anchor 一个可用的量级估计

这个数字是 Python 级循环的口径：一秒约七千万次循环体。

由此可以估算： $n=10^6$ 的 $O(n)$ 约 0.01 秒；同样规模的 $O(n^2)$ 是 10^{12} 步，要跑几个小时。内置函数在 C 中执行，会快一到两个数量级，所以这个锚点是下界。



Prediction · Which Cost Grows Faster?

Which one costs more when n becomes large? 先预测，再往下看。

```
1 def value_at(numbers: list[int], index: int) -> int:
2     total = 0
3     for i in range(len(numbers)):
4         total += 100 * i
5     return total + numbers[index]
```

13

Slide 13 · Prediction · Which Cost Grows Faster?

先投票：一个常数很大的线性循环，和一个什么都没有的常数操作，谁更贵。

本页只看单次调用的表面，下一页让 n 变大再比较。



Constants Do Not Catch Up 常数追不上阶数

n	$100n$ ($O(n)$)	n^2 ($O(n^2)$)	Smaller 更少的一方
10	1,000	100	$O(n^2)$
100	10,000	10,000	tie
1,000	100,000	1,000,000	$O(n)$
10,000	1,000,000	100,000,000	$O(n)$

常数大只输在小规模；阶数高迟早输。

14

Slide 14 · Constants Do Not Catch Up 常数追不上阶数

n



Costs of Python Operations Python 操作的复杂度

Operation 操作	Cost 复杂度	Note 说明
<code>a[i], len(a)</code>	$O(1)$	position is computed, length is stored
<code>a.append(x), a.pop()</code>	$O(1)$	touches the end only
<code>a.insert(0, x), a.pop(0)</code>	$O(n)$	everything after shifts
<code>x in a (list)</code>	$O(n)$	scans from the front
<code>x in s, d[k] (set/dict)</code>	$O(1)$	looked up by hash
<code>sorted(a)</code>	$O(n \log n)$	comparison sort
<code>a[i:j], s1 + s2</code>	$O(j-i), O(n+m)$	builds a new object

列表查找线性、集合字典常数，是选容器的第一条依据。

15

Slide 15 · Costs of Python Operations Python 操作的复杂度

这些是常见结论，不是全部实现细节。重点记两条：列表头部插入删除是线性，字典与集合的键查找是常数。

下一页用实测验证第二条，这正是 L06 容器选型那句结论的依据。



List or Set, Measured 实测列表与集合

Container 容器	n = 30,000	n = 60,000
list: <code>x in a</code>	0.0127 s	0.0500 s
set: <code>x in s</code>	0.043 ms	0.043 ms

同一百次查询，只换容器。列表慢约 297 倍，n 翻倍后时间翻约四倍。

16

Slide 16 · List or Set, Measured 实测列表与集合

n

Prediction · Building a String



Is the loop slower, and by how much? 先给出倍数，再往下看。

```
1 words = ["hello"] * 80_000
2
3 result = ""
4 for word in words:
5     result += word
6
7 # or collect and join
8 same = "".join(words)
```

17

Slide 17 · Prediction · Building a String

先投票，并说出理由。学生最常见的理由是「字符串不可变，所以每次都复制」。

下一页用三种写法实测校正这个直觉的适用范围。这是本讲最重要的一处修订。



| In-Place Concatenation 就地拼接的实测结果

Kind 写法	n = 20,000	n = 80,000
plain s += w	0.467 ms	0.00166 s
keeping every prefix	0.0235 s	1.1580 s
"".join(words)	0.049 ms	0.195 ms

| CPython 对只有一个引用的字符串原地扩展；中间结果被别处引用时就要复制。

18

Slide 18 · In-Place Concatenation 就地拼接的实测结果

实测：普通循环与 join 在同一量级，每次都把中间结果留活的写法慢了三个数量级。

原因是 CPython 对只有单一引用的字符串可以原地扩展。结论必须写成「取决于有没有别的引用」，不能写成「循环拼接一律平方级」。写代码仍然推荐 join，因为它不依赖这条实现细节。



| Why Binary Search Is $O(\log n)$ 二分查找为什么是对数

Measurement 测量	n	Best of 3 取最小
1,000 elements	10 comparisons	$\log_2(1000) = 9.97$
1,000,000 elements	20 comparisons	halved 20 times
1,000,000,000 elements	30 comparisons	halved 30 times

| Measured on the build machine 本机构建时实测，同一进程内重复取最小值。

19

Slide 19 · Why Binary Search Is $O(\log n)$ 二分查找为什么是对数

L07 写过二分查找，当时把复杂度留到这里。每比较一次候选范围减半，问题是从 n 降到 1 要除几次 2，答案是对数。

十亿个元素最多三十次比较，这就是对数复杂度，也是排好序的数据值钱的原因。



I Space Complexity 空间复杂度

- **$O(1)$ space**: a running total — a few variables whatever n is
- **$O(n)$ space**: a list of length n , built before returning
- **The same notation** describes memory
- **Memoisation trades space for time**

20

Slide 20 · Space Complexity 空间复杂度

同样的记号用来描述内存占用，这一点学生第一次见。

生成器如何把 $O(n)$ 空间降回 $O(1)$ 在 L13 讨论，这里先建立概念。



I Improve the Order First 先降阶再优化常数

- **1. Change the algorithm
- **2. Change the data structure
- **3. Avoid recomputation
- **4. Use built-ins

21

Slide 21 · Improve the Order First 先降阶再优化常数

顺序不能颠倒：内置函数快是常数变小，救不了错的算法。本机实测 `sum(range(2*106))` 比手写循环快约 2.4 倍，两者都是 $O(n)$ 。

这就是「先降阶、再优化常数」的具体含义，也是 Knuth 那句话的落点。

I Quiz 2 · Logarithmic Growth



扫码作答 · 1 题

1 折半次数与对数的关系

Submit this round once. 按已教规则推理。

<https://taohuang.info/cs1602/zh/quiz/11/>

22

Slide 22 · Quiz 2 · Logarithmic Growth

检查是否理解折半次数就是对数，不涉及新记号。

第19页已经给出方法，本题验收；下页讲评原题分布。

II · Failure and Exceptions 失败与异常

从成本转到失败。先区分可预期的外部问题和不该发生的内部错误，这一区分在第三部分决定用异常还是断言。

顺序不能颠倒：先看得见的崩溃，再谈不声不响的错。

| Syntactically Valid, Still Fails 语法正确仍会失败



code

```
1 x = 10
2 y = 0
3 print(x / y)
```

Actual exception · final line 实际异常

```
ZeroDivisionError:
division by zero
```

24

Slide 24 · Syntactically Valid, Still Fails 语法正确仍会失败

这段代码语法完全正确，Python 顺利编译了，运行时才出错。这种错误叫异常。

回溯的最后一行给出类型与消息；不处理的话程序在这里终止，后面的语句一行都不执行。

| Two More Runtime Failures 另外两种运行时失败



Code 代码

Actual exception 实际异常

```
print(value + spam * 3)
```

NameError: name 'spam' is not defined

```
print(text + 2) # text is "2"
```

TypeError: can only concatenate str (not "int") to str

NameError: a name was never defined. **TypeError**: two types cannot be combined. 两种都只有执行到那一行才会发生。

25

Slide 25 · Two More Runtime Failures 另外两种运行时失败

上一页是除零，这一页再补两种最常见的运行时错误。

共同点是名字和类型都检查不出来，必须真的执行到那里才知道。下一页开始处理它们。



| try and except

code

```
1 def safe_divide(  
2     a: float,  
3     b: float,  
4 ) -> float | None:  
5     try:  
6         return a / b  
7     except ZeroDivisionError:  
8         print("cannot divide by zero")  
9         return None  
10  
11 print(safe_divide(10, 2))  
12 print(safe_divide(10, 0))  
13 print("program continues")
```

output

```
5.0  
cannot divide by zero  
None  
program continues
```

26

Slide 26 · try and except

try 体正常执行时跳过 except；一旦抛出异常，立刻跳到匹配的分支。

这正好解决了 L03 留下的问题：那时候说输入非数字程序会崩，L11 讲怎么办。



| Catch Specific Types 捕获具体类型

code

```
1 def parse_age(text: str) -> int | None:  
2     try:  
3         return int(text)  
4     except ValueError:  
5         print(f"{text!r} is not an integer")  
6         return None  
7  
8 for s in ["18", "abc", "3.14", ""]:  
9     print(s, "->", parse_age(s))
```

output

```
18 -> 18  
'abc' is not an integer  
abc -> None  
'3.14' is not an integer  
3.14 -> None  
' ' is not an integer  
-> None
```

27

Slide 27 · Catch Specific Types 捕获具体类型

捕获具体的类型，程序可以给用户一句有用的提示并继续。

下一页把三种不同的失败放进同一段 try，说明分支顺序。

Prediction · Handler Order



Which handler runs? 先预测输出顺序。

```
1 def classify(value: str) -> str:
2     try:
3         number = int(value)
4     except Exception:
5         return "general"
6     except ValueError:
7         return "specific"
8     else:
9         return f"ok {number}"
10    finally:
11        print("cleanup")
12
13 print(classify("abc"))
```

28

Slide 28 · Prediction · Handler Order

先投票：宽类型写在前面、具体类型写在后面，捕获到的会是哪一个。

常见直觉是「更具体的会先匹配」。下一页用实测结果回答，并说明裸 except 为什么更糟。

| First Match Wins 先匹配先执行



code

```
1 def classify(value: str) -> str:
2     try:
3         number = int(value)
4     except Exception:
5         return "general"
6     except ValueError:
7         return "specific"
8     else:
9         return f"ok {number}"
10    finally:
11        print("cleanup")
12
13 print(classify("abc"))
```

output

```
cleanup
general
```

29

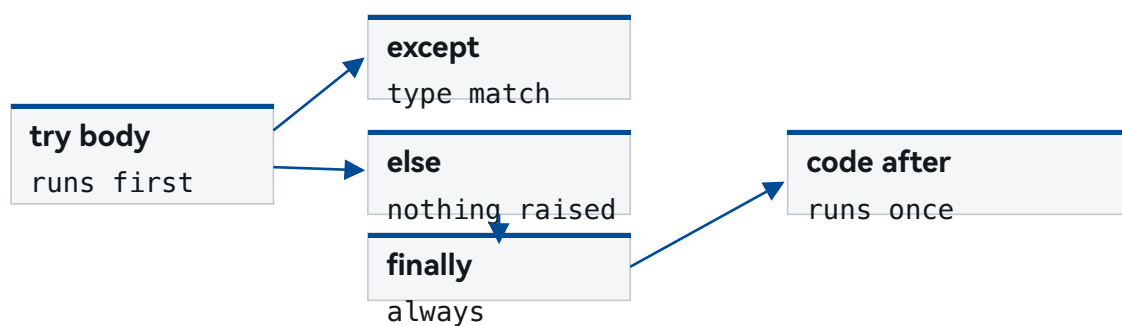
Slide 29 · First Match Wins 先匹配先执行

实际先打印 cleanup 再返回 general。异常按书写顺序从上到下匹配，宽类型在前会遮住后面的具体类型。

裸 except 等价于捕获一切，包括拼写错误和 KeyboardInterrupt；要兜底就写 except Exception as error 并把消息打印出来。



I else and finally



finally runs on every way out 每条离开路径都执行，return 也一样。

实测 10 / 2 走 else 与 finally, 10 / 0 走 except 与 finally: 10 / 2: / ok, result 5.0 / cleanup / 10 / 0: / failed / cleanup

else 只在 try 体没有抛异常时执行，放成功之后才该做的事；finally 无论如何都执行，放清理工作。

实测四行输出说明两条路径的差别：成功走 else 和 finally，失败走 except 和 finally。下一页检查 finally 里的 return。

Prediction · finally and return



What does the function return?

```
1 def f() -> str:
2     try:
3         return "from try"
4     finally:
5         return "from finally"
6
7 print(f())
```

31

Slide 31 · Prediction · finally and return

先投票：try 里已经准备好返回 from try，finally 里又写了一个 return，函数最终返回什么。

常见答案是 from try。下一页给出实测结果，并说明 finally 里的 return 还会抑制正在传播的异常。

| finally Cannot Be Skipped 清理代码不写 return



code

```
1 def f() -> int:
2     try:
3         return 1 / 0
4     finally:
5         return 42
6
7 print(f())
```

output

42

32

Slide 32 · finally Cannot Be Skipped 清理代码不写 return

实际返回 42，而不是抛出 `ZeroDivisionError`：finally 里的 return 既覆盖返回值，也丢弃待传播的异常。

finally 在任何离开 try 的路径上都执行，这正是清理代码不要顺手写 return 的理由。with 的异常抑制是同一类边界。

I raise 主动抛出



code

```
1 def set_age(age: int) -> int:
2     if age < 0:
3         raise ValueError(f"age cannot be negative: {age}")
4     if age > 150:
5         raise ValueError(f"unreasonable age: {age}")
6     return age
7
8 print(set_age(18))
9 print(set_age(-5))
```

Actual exception · final line 实际异常

```
ValueError:
age cannot be negative: -5
```

33

Slide 33 · raise 主动抛出

需要调用方无法忽略时用 raise。返回 None 容易被忽略，错误会在离原因很远的地方才爆发。抛异常立刻停下，回溯直接指向出问题的位置。本周实验里的 ValueError 就是这条规则的应用。



I Custom Exceptions 自定义异常

code

```
1 class InsufficientFunds(Exception):
2     def __init__(self, balance, amount):
3         super().__init__("too low")
4         self.balance = balance
5         self.amount = amount
6
7     def withdraw(balance, amount):
8         if amount > balance:
9             raise InsufficientFunds(balance, amount)
10        return balance - amount
11
12    try:
13        withdraw(100, 500)
14    except InsufficientFunds as e:
15        print("caught:", e)
16        print("short by:", e.amount - e.balance)
```

output

```
caught: too low
short by: 400
```

34

Slide 34 · Custom Exceptions 自定义异常

自定义异常继承 Exception，L10 的继承在这里派上用场。

好处是调用方能精确捕获这一种失败，而不是笼统地抓 ValueError。异常类保持简单，通常只带需要的信息。

HALFWAY

Break

两个45分钟的分界。上半节讲了成本、异常与清理。

下半节讲假设检查、测试的写法，以及素数五法的规模与成本。



| with and Cleanup 上下文管理与清理

code

```
1 with open("demo.txt", "w") as handle:
2     handle.write("hello")
3 print("closed:", handle.closed)
4
5 try:
6     with open("demo.txt") as reader:
7         raise ValueError("body failed")
8 except ValueError:
9     print("still closed:", reader.closed)
```

output

```
closed: True
still closed: True
```

36

Slide 36 · with and Cleanup 上下文管理与清理

with 保证退出时执行清理，等价于写好的 try/finally。

只有 __enter__ 成功返回，__exit__ 才会被调用；块里抛异常，文件同样关闭。文件操作的完整内容在 L12。



| Check First or Catch 先检查还是先捕获

Style 风格	Shape 写法	When it reads better 什么时候更清楚
LBYL	if key in d: use it	simple preconditions
EAFP	try it, catch KeyError	the normal path is the main line
Both are correct	choose by intent	neither is a rule

| 看意图，不背规则。

37

Slide 37 · Check First or Catch 先检查还是先捕获

Python

I Quiz 3 · Handler Order



扫码作答 · 1 题

1 分支顺序与 finally 的执行位置

Submit this round once. 按已教规则推理。

<https://taohuang.info/cs1602/zh/quiz/11/>

38

Slide 38 · Quiz 3 · Handler Order

同时检查分支顺序与 finally 的位置。所有语法在第34页前都已教完。

下页讲评原题分布，不再发问题。

III · Assertions and Testing 断言与测试

进入第三种失败。断言把「我以为一定成立」写成可执行检查，测试把观察值与期望值比较。

两者都指向证据，但一个检查程序自身的前提，一个检查对外行为。



| assert Writes Down an Assumption 断言写下假设

code

```
1 def average(nums: list[float]) -> float:
2     assert len(nums) > 0, "empty list"
3     return sum(nums) / len(nums)
4
5 print(average([1, 2, 3]))
```

output

2.0

40

Slide 40 · assert Writes Down an Assumption 断言写下假设

assert 条件, 消息 的意思是：我认为这里条件一定成立，不成立就说明程序有 bug，立刻停下。

用法是描述「这里必须成立」，不是给用户看的提示。下一页检查它在优化模式下的行为。

Prediction · Assertions Under -O



What changes when assertions are disabled?

```
1 def average(nums: list[float]) -> float:
2     assert len(nums) > 0, "empty list"
3     return sum(nums) / len(nums)
4
5 print(average([1, 2, 3]))
```

41

Slide 41 · Prediction · Assertions Under -O

同一个函数，一次在普通模式下运行，一次加上 -O 参数。先预测两次的结果有什么不同。

常见直觉是「关掉断言只是少了一层检查」。下一页给出两次运行的实际结果，其中一次会以除零错误结束。



I Assertions Can Be Removed 断言会被移除

Mode 模式	__debug__	Result 结果
python3 example.py	__debug__ = True	AssertionError: empty list
python3 -O example.py	__debug__ = False	ZeroDivisionError: division by zero

断言是开发期的检查，不能校验用户输入；那要用 if 加 raise，不会被优化掉。

42

Slide 42 · Assertions Can Be Removed 断言会被移除

正常模式下断言生效，对空列表抛出 AssertionError；-O 模式下断言被跳过，函数继续执行到除零，抛出的变成 ZeroDivisionError。

所以断言不能校验用户输入。另外 assert (条件, 消息) 写成元组永远为真，检查会完全失效。

Quiz 4 · Assertion Side Effects



扫码作答 · 1 题

1 断言表达式里的副作用

Submit this round once. 按已教规则推理。

<https://taohuang.info/cs1602/zh/quiz/11/>

43

Slide 43 · Quiz 4 · Assertion Side Effects

这道题检查断言表达式里的副作用：条件看起来成立，实际却已经改了状态。

全部语法已教，下页讲评。

Assertions as Documentation 断言作为文档



Place 位置	Written as 写成	Catches 能查出什么
Precondition	<code>assert sorted(nums) == nums</code>	an argument that is out of contract
Postcondition	<code>assert 0 <= i < len(nums) or i == -1</code>	a result the code promised but did not deliver

断言描述契约；契约变了，断言跟着变。

44

Slide 44 · Assertions as Documentation 断言作为文档

断言把「输入必须有序」这类前置条件写成可执行检查，比注释可靠，因为注释会过期。

后置条件同样可以写：返回的下标处确实是目标，或者返回 -1 且目标确实不在其中。下一页把同一件事交给 pytest。

I pytest 把测试正规化



- **Files and functions** both start with `test_`
- **Expectations are ordinary assert**
- **`pytest.raises`** when a failure is the expected outcome
- **`python3 -m pip install pytest`**

45

Slide 45 · pytest 把测试正规化

L08 用检查循环比对期望值，pytest 把这套动作标准化。规则只有三条，先记住文件名与函数名都要以 `test_` 开头。

下一页给一次真实运行的输出，包括一个故意写错的用例。

I Running pytest 运行与失败诊断



Actual pytest -q output · real run 真实运行输出（终端行按面板宽度折行）

```
...F
[100%]
===== FAILURES =====
=====
E   ValueError: cannot divide by zero
mymath.py:6: ValueError: cannot divide by zero
===== short test summary info =====
FAILED test_mymath.py::test_divide_by_zero - ValueError: cannot divide by
zero
1 failed, 3 passed in 0.00s
```

46

Slide 46 · Running pytest 运行与失败诊断

这段输出来自真实执行：三个用例通过，一个失败。

输出给出三件事：哪个用例失败、它期望什么（ZeroDivisionError）、实际发生什么（ValueError）。这正是它比手写循环有用的地方。完整诊断格式与 -v、-k 参数在参考区。



I What to Test 该测什么

Category 类别	Example 例子
Normal	a typical input
Empty	an empty list, an empty string
Single	exactly one element
Boundary	zero, negatives, largest and smallest
Failure	does invalid input raise as agreed

每个分支一个用例；空与单个最容易漏。

47

Slide 47 · What to Test 该测什么

实用起点是每个 if 分支至少走一遍。

L08 的边界清单在这里继续使用：AI 最常漏的仍然是「空」和「一个」。下一页把这张表用在一个别人写的模块上。



| Test Code You Did Not Write 给别人的代码写测试

Excerpt · complete definition in notes

```
def longest_word(text):  
    words = text.split()  
    return max(words, key=len)
```

- `word_count("")` -> 0
normal 正常返回
- `longest_word("")` -> **ValueError** max() got an empty sequence
- `average_word_length("")` -> **ZeroDivisionError**
division by zero

Complete runnable example 完整程序

48

Slide 48 · Test Code You Did Not Write 给别人的代码写测试

本周实验给的模块，三个函数各有一句文档，都没有说空字符串怎么办。

实测三个里有两个在空串上失败。文档没有约定的输入，正是容易出问题的地方。完整模块与文档在讲义。

I Prediction · How Many Fail on Empty Input?



How many of the three fail on an empty string?

```
1 def word_count(text: str) -> int:
2     """Count the words."""
3     return len(text.split())
4
5 def longest_word(text: str) -> str:
6     """Return the longest word. On a tie, the first one."""
7     words = text.split()
8     return max(words, key=len)
9
10 def average_word_length(text: str) -> float:
11     """Return the mean word length."""
12     words = text.split()
13     return sum(len(w) for w in words) / len(words)
```

49

Slide 49 · Prediction · How Many Fail on Empty Input?

先让学生给出数字和理由，再往下看。

这是「文档与实现不一致」的例子，也是测试存在的意义：先把期望写下来，再看实现是否满足。

I Expected and Actual 期望与实际



Actual output 实跑输出

```
word_count: 0
longest_word: ValueError
average: ZeroDivisionError
```

- **Two of three fail** on the empty string 三个里两个失败
- **The docstrings never mention it**
- **A test writes the expectation down first**

50

Slide 50 · Expected and Actual 期望与实际

实测 `word_count("")` 返回 0 正常；`longest_word("")` 抛 `ValueError`，`average_word_length("")` 抛 `ZeroDivisionError`。

原因分别是 `max()` 收到空序列和除数为零。测试的作用是先把期望写下来，再看实现是否满足。

I How the Platform Marks Work 平台如何判分



```
1 def average(nums: list[float]) -> float:
2     print(sum(nums) / len(nums))
3
4 got = average([1, 2, 3])
5 print(got == 2.0)
```

51

Slide 51 · How the Platform Marks Work 平台如何判分

平台做的就是返回值和期望值比较，和上面 clamp 的检查循环是同一件事。

用 print 的函数返回 None，比较永远为假——这就是从 L03 起那句「函数要 return 不要 print」背后的机制，不是一条孤立的规则。

I Quiz Review · Assertions and Testing



Question	Actual result	Reason
Q4 assert not events.append("hit")	1	append returns None, so the assert passes; the element is already in the list
Q5 lru_cache over a list argument	TypeError: unhashable type: 'list'	a cache must build a key from its arguments

52

Slide 52 · Quiz Review · Assertions and Testing

Q4

| The Cost of the Five Methods 五种方法的成本



Method 方法	Time 时间复杂度	Space 空间	Idea 思路
brute	$O(n^2)$	$O(1)$	every p tries every d below it
sqrt	$O(n \sqrt{n})$	$O(1)$	a factor below $\sqrt{p}$ is enough
$6k+1$	$O(n \sqrt{n})$	$O(1)$	skip multiples of 2 and 3
sieve	$O(n \log \log n)$	$O(n)$	mark the whole range at once
Miller-Rabin	$O(n \log^3 n)$	$O(1)$	a few modular powers per number

53

Slide 53 · The Cost of the Five Methods 五种方法的成本

这是本周个人作业 2 的理论依据。前三种同阶、常数不同；筛法降了阶，代价是一份长度为 n 的标记数组。

复杂度只说明增长形状，实际耗时还要看常数，下一页用实测说明。



I Measured Prime Counts 素数方法的实测

Method 方法	n = 2000	n = 100000
brute	0.0042 s	too slow
sqrt	0.473 ms	0.0864 s
6k±1	0.195 ms	0.0288 s
sieve	0.036 ms	0.0021 s
Miller-Rabin	0.455 ms	0.0267 s

五种方法在小规模都得 303，大规模都得 9592；先验证再计时。

54

Slide 54 · Measured Prime Counts 素数方法的实测

平方根剪枝与 $6k \pm 1$ 在同一阶，常数差约三倍；筛法一直最快，代价是那份 $O(n)$ 空间。

报告要求先验证正确性再计时，慢方法允许选可完成的规模，但不能换成另一种算法冒充。



I Quiz 5 · Caching and Keys



扫码作答 · 2 题

- 1 缓存要求可哈希的参数 2 缓存键必须唯一确定子问题

Submit this round once. 按已教规则推理。

<https://taohuang.info/cs1602/zh/quiz/11/>

55

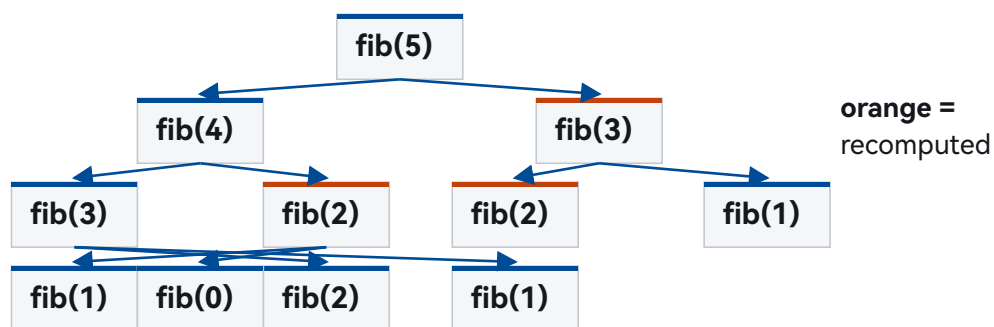
Slide 55 · Quiz 5 · Caching and Keys

两题分别检查「缓存要求可哈希的参数」与「键必须唯一确定子问题」。

全部语法已教，第57页后讲评。



Memoization: Remember Computed Results 记忆化



One call tree 一次调用树; fib(3) and fib(2) are computed again and again.

实测 $\text{fib}(20) = 6765$ calls = 21,891; $\text{fib}(25) = 75025$ calls = 242,785; $\text{fib}(28) = 317811$ calls = 1,028,457。n 加 3，调用次数大约翻四倍，结果只增长到三倍左右。

L07 发现朴素斐波那契重复计算量巨大，并把修复留到这里。图中同一个 fib(3)、fib(2) 被反复算，这就是调用次数远超结果的原因。

根源是同一个子问题被算了无数遍，解决办法是把算过的记下来。



The Cache Key Identifies a Subproblem 缓存键与子问题

Key on amount alone. What comes back?

```
1 def ways(amount: int, coins: list[int], index: int) -> int:
2     if amount in memo:
3         return memo[amount]
4     if amount == 0:
5         return 1
6     if amount < 0 or index >= len(coins):
7         return 0
8     memo[amount] = (ways(amount - coins[index], coins, index)
9                     + ways(amount, coins, index + 1))
10    return memo[amount]
11
12 memo: dict[int, int] = {}
13 print("key = amount: ", ways(10, [1, 2, 5, 10], 0))
```

57

Slide 57 · The Cache Key Identifies a Subproblem 缓存键与子问题

正确答案是 11，只用 amount 作键得到 129。

原因是 ways 有两个会变的参数：只看 amount，会把「用前两种面额凑出 10 分」和「用全部四种凑出 10 分」当成同一个状态，于是缓存了一个属于别的子问题的值。下一页对照两种键。



Two Conditions for a Correct Cache 缓存正确的两个条件

Condition 条件	Why 理由
The key identifies the subproblem uniquely	every varying argument belongs in it: (amount, index), not amount
Same input, same output	a function of the clock, of randomness or of outside state cannot be cached

11 = 11, ways(100) = 2156, calls = 801, cache entries = 400; fib(28) = 317811, calls = 55, cache entries = 27

58

Slide 58 · Two Conditions for a Correct Cache 缓存正确的两个条件

换成元组键之后结果正确，缓存里的条目数正好是所有 (amount, index) 的合法组合数。

记忆化把指数级降到线性，代价是存下这些结果，这是又一处空间换时间。



I functools.lru_cache

code

```
1 from functools import lru_cache
2
3 @lru_cache(maxsize=None)
4 def fib(n: int) -> int:
5     if n <= 1:
6         return n
7     return fib(n - 1) + fib(n - 2)
8
9 print(fib(10))
10 i = fib.cache_info()
11 print(i.hits, i.misses, i.currsz)
12 print(fib(10))
13 print("hits:", fib.cache_info().hits)
14 print(fib(100))
```

output

```
55
8 11 11
55
hits: 9
354224848179261915075
```

59

Slide 59 · functools.lru_cache

标准库提供现成实现，装饰器本身在 L14 展开。本页只讲三件事：它能用、参数必须可哈希、maxsize 会限制条目数。

cache_info 给出命中与未命中次数；第二次算 fib(10) 时命中数增加，说明没有重新递归。Q5 的 TypeError 正是这条要求的直接后果。



I Solo Assignment 2 · Counting Primes

- Five genuine methods
- Range $0 \leq p < n$, $n \leq 100000$; $n \leq 2$
- Verify first, then time
- Miller-Rabin bases 2 and 3 are certain for $n < 1,373,653$
- Three solo assignments: 15% total; this one: 5%

本周任务是素数计数；大整数的分组表示留到第12周。

60

Slide 60 · Solo Assignment 2 · Counting Primes

按现行配置第10周启动、第11周提交，三份个人作业各占 5%。

MR 的确定性只在本题域内成立，不能外推到任意大整数。完整参考解不公开，助教核对各方法是否为真实实现。



I Choosing a Method 五种方法怎么选

Situation 情形	Method 方法	Why 理由
many numbers, modest range	sieve	one pass marks the whole range
one very large number	Miller-Rabin	$O(1)$ space, a few modular powers
this week, $n \leq 100000$	any trial division	all finish; only the constant differs

先看规模和空间，再选方法。

61

Slide 61 · Choosing a Method 五种方法怎么选

差别在适用规模与空间：筛法要一份长度为 n 的标记数组，适合一次问出整个区间；Miller-Rabin 每次只判断一个数，空间是常数。

本周任务限定 $n \leq 100000$ ，三种试除法都能完成，区别在耗时，不在阶数。



| Lab 11 · Fixed Interfaces 固定接口

Entry 入口

Contract 契约

w11-complexity-measure	comparable timings; no printing
w11-string-build	three builders agree; build_keep is the comparison
w11-memoize	the key holds every varying argument
w11-safe-convert	specific types only; no bare except
w11-config-review	a missing file differs from malformed content
w11-custom-exception	set_celsius raises TemperatureError
w11-tests	four cases per function, one pytest.raises

62

Slide 62 · Lab 11 · Fixed Interfaces 固定接口

七项沿用现稿任务，接口在实验页写明，提交页就地展开题面。

第11周按 Level 1 要求提交使用声明；本地保存与验收不等于线上提交。



| Common Misjudgements 常见误判

Written 常见写法

What actually happens 实际发生

except: catches everything safely	it swallows typos and KeyboardInterrupt
assert validates user input	assertions are removed under -O
return in finally just returns early	it replaces the value and discards the exception
assert (cond, msg) adds a message	a non-empty tuple is always true
more tests means better tests	covering the branches matters, not the count
read the implementation, then write tests	write the expectation from the docstring first

63

Slide 63 · Common Misjudgements 常见误判

六条都对应本讲实测过的代码，不是风格偏好。

最后一条是第48页的直接结论，也是 L08「先写测试」在本周的具体要求。

I Three Kinds of Evidence 小结



- **Growth
- **Boundaries
- **Checks

64

Slide 64 · Three Kinds of Evidence 小结

回到第2页的三种失败：太慢用增长阶回答，会崩用失败边界回答，悄悄算错用可执行检查回答。

第12周处理随机数、编码与文件，那里的输入更不干净，本讲的三种证据会继续使用。主线到此结束，后面32页供课后查阅。

Reference 参考

参考保留完整的异常机制、旧稿的复杂度与哈希知识、素数五法的完整实现与提交规格。

片段标明依赖，长程序链接完整讲义，不设独立运行按钮。

| Worst, Best and Average Case 最坏最好与平均



Case 情况	Definition 定义	Role here 本课的处理
Worst	the most expensive input at that size	the default
Best	the cheapest input	rarely worth quoting
Average	the expectation over inputs	needs a distribution assumption

66

Slide 66 · Worst, Best and Average Case 最坏最好与平均

同一个算法在同样规模下，不同输入的耗时可能不同。

本课不特别说明时都按最坏情况分析；平均情况的结论离开了分布假设就不成立。

| Asymptotic Meaning 渐近含义



- $O(f(n))$ is an upper bound
- $n + 2$ is $O(n)$, and also $O(n^2)$ — we write the tightest bound
- **Dropping constants is safe** only when the input is large and the orders differ
- **Same order means the constant is all that is left**

67

Slide 67 · Asymptotic Meaning 渐近含义

$n + 2$ 确实被 n^2 从上方界住，所以写成 $O(n^2)$ 不算错，只是不紧。

平方根剪枝与 $6k \pm 1$ 就是「同阶、只差常数」的例子。

| Polynomial Evaluation 多项式求值



Actual output 实跑输出

5.4642884629601965e+70

- **Naive 朴素写法** uses x^{**} power every iteration
- **Horner** replaces powers with repeated multiply-and-add
- **Same mathematics**

68

Slide 68 · Polynomial Evaluation 多项式求值

400 项系数时两种写法实测相差约 37 倍，阶数相同、常数不同。

这不是新算法，是同一个式子的两种写法，值得知道有这个形式。

| List Memory Model 列表内存模型



- Elements are stored contiguously
- Index corresponds to element one to one
- **Together:** $a[i]$ costs $O(1)$; $a.insert(0, x)$ and $a.pop(0)$ cost $O(n)$
- **Measured** inserting at the front: $n = 5,000$ in 0.0015 s, 10,000 in 0.0046 s, 20,000 in 0.0199 s

69

Slide 69 · List Memory Model 列表内存模型

这两条假设推出上面全部结论，是第15页那张表的依据。

n 翻倍时间大约翻四倍，因为每一次插入本身是线性操作，而它被调用了 n 次。旧稿手写的 insert/find/remove 实现保留在讲义。

| Hash Table 哈希表



Idea 想法

Consequence 推论

A position function computes where a value is stored

lookup does not scan

More slots than elements

space for time

Two keys can land on one slot

collisions exist and have standard remedies

70

Slide 70 · Hash Table 哈希表

字典和集合的 $O(1)$ 来自位置函数，代价是预留空间。

本课不要求你设计哈希函数，只要求知道契约；碰撞的存在不否定平均情况下的常数查找。



I hash() and Contracts 哈希契约

Expression 表达式	Result 实测结果
<code>hash(1) == hash(1.0)</code>	True
<code>hash(True) == hash(1)</code>	True
<code>hash((1, 2))</code>	works
<code>hash([1]) / hash({"a": 1})</code>	TypeError: unhashable type
a class that defines <code>__eq__</code> only	<code>__hash__</code> becomes None

相等则同哈希；可变对象不做键。

71

Slide 71 · hash() and Contracts 哈希契约

规则是「相等的对象必须有相等的哈希值」，1、1.0、True 三者相等，所以哈希值相同。

可变容器不可哈希：内容会变，如果能当键，改内容就会让键找不到。定义 `__eq__` 而不定义 `__hash__` 的类，实例自动变成不可哈希。



| Custom __hash__ 自定义哈希

- Equal objects must hash equally
- The hash need not be unique
- Mutable objects make unsafe keys
- Easier route

72

Slide 72 · Custom __hash__ 自定义哈希

要让自己写的对象当字典键，就得同时给出一致的 __eq__ 和 __hash__。

实践中多数情况用一个元组当键就够了，不必自己实现哈希。



| Binary Search 二分查找

Actual output 实跑输出

2 -1

- The +1 and -1 keep it moving
边界加减一，否则死循环
- Sorted input is required
- -1 means absent

73

Slide 73 · Binary Search 二分查找

保留原来的 while 实现与边界说明，和 L07 的递归版本对照。

排序前提在第三部分的断言里写成了可执行检查。

| Recursion Cost 递归成本



n	Calls 调用次数	Answer 结果
20	21,891	6,765
25	242,785	75,025
28	1,028,457	317,811

74

Slide 74 · Recursion Cost 递归成本

调用次数大约是结果的三倍多，说明绝大多数调用都在重复算已经算过的东西。

递归深度也有上限：本机 `sys.getrecursionlimit()` 为 1000，超过就抛 `RecursionError`。所以朴素 `fib(2000)` 不是慢，是直接崩；记忆化版本每次只往下走一层，没有这个问题。



| Memoized Fibonacci 记忆化斐波那契

- Dictionary cache
- List cache
- Both are $O(n)$ time and $O(n)$ space
- `@lru_cache(maxsize=...)` caps the entries

75

Slide 75 · Memoized Fibonacci 记忆化斐波那契

`fib(28)` 记忆化后调用 55 次、缓存 27 个条目，因为只有 28 个不同的子问题。

字典与列表两种写法在讲义里都给了，区别只在查找常数和内存的实际使用量。



| Space and Recursion 空间与递归

- Each recursive call saves its own state
- Naive `fib` occupies $O(n)$ of stack
- An iterative version finishes in $O(1)$
- Generators take this further

76

Slide 76 · Space and Recursion 空间与递归

同样算 `fib`，迭代版本只用两个变量，递归版本要保留整条调用链。

L13 讲生成器时，会把「一次只产生一个值、不建整个列表」这条思路用到极致。

I raise Forms 抛出形式



Actual output · real run 实跑输出

```
value: alpha
logged: 'missing'
the caller decides
```

- **Class plus argument**
- **Bare raise**
- **from keeps the cause**
- **Returning None instead**

裸 raise 记下信息后继续往上抛；异常链用 from 保留因果，讲义有完整例子。

77

Slide 77 · raise Forms 抛出形式

函数记下信息后用裸 raise 继续往上抛，回溯保留，最终由调用方决定怎么处理。

返回值与抛异常的选择在这里一目了然：返回 None 的话，打印出来的是一句话；抛异常的话，调用方必须表态。

I Multiple Handlers 多分支



Form 写法	Meaning 含义
except (KeyError, IndexError):	tuple form
except:	catches everything, including KeyboardInterrupt
except Exception as error:	catch-all with a message
sibling clauses	do not catch each other

78

Slide 78 · Multiple Handlers 多分支

同一段 try 的多个 except 不捕获彼此抛出的异常，它会往上走。

需要兜底时先记录再 raise，这样既留下信息又没有把错误吞掉。



I Custom Exception Hierarchy 自定义异常层级

Level 层级	Purpose 作用
class ConfigError(Exception)	a base for one family of problems
class ConfigMissing(ConfigError)	callers can catch the whole family or one subclass
fields	carry the values the handler needs, such as the rejected value

79

Slide 79 · Custom Exception Hierarchy 自定义异常层级

一个模块里定义多个相关异常时，常见做法是先定基类再派生。

L09 的 Temperature 改成抛异常后，set_celsius 从返回 False 变成抛 TemperatureError，调用方就不能再忽略返回值了。



I Resource and OS Errors 资源与系统错误

Exception 异常	Typical source 常见来源
OSError, FileNotFoundError, PermissionError	files, directories, permissions
ValueError	a failed conversion, such as int("abc")
TypeError	mismatched types
KeyError / IndexError	missing key / index out of range

80

Slide 80 · Resource and OS Errors 资源与系统错误

with



I assert Forms 断言写法

Form 写法	Result 结果
<code>assert condition</code>	raises AssertionError with no message
<code>assert condition, "message"</code>	the message is shown
<code>assert (condition, "message")</code>	always true
side effects	still happen

消息写在逗号后面，不要包进括号。

81

Slide 81 · assert Forms 断言写法

第三行是最容易犯的错：多写一对括号，检查就完全失效。

第四行见 Q4: `assert not events.append("hit")` 通过，但元素已经进了列表。



I __debug__ and Optimized Mode 优化模式

Setting 设置	Effect 作用
<code>__debug__</code>	a constant; False under optimisation
<code>python3 -O</code>	assert statements are removed
<code>python3 -OO</code>	also drops docstrings
<code>PYTHONOPTIMIZE</code>	the same effect through the environment
cached bytecode	the file name carries the optimisation level

82

Slide 82 · __debug__ and Optimized Mode 优化模式

实测：正常模式下 `__debug__` 为 True，-O 下为 False，断言被跳过。

所以断言不能用来校验用户输入，那条路径会被优化掉。



I Assertion Pitfalls 断言的误用

Misuse 误用	What goes wrong 后果
validating user input	removed under -O, so the check disappears
handling an expected failure	the bug is treated as a user error
side effects in the expression	the effect happens even when the assert passes
catching AssertionError	the bug is hidden

83

Slide 83 · Assertion Pitfalls 断言的误用

断言失败意味着程序有 bug，正确的做法是让它崩掉，这样才看得到回溯。

开发期与生产期的差别就在这里：断言是开发期的检查。



I pytest Conventions and Options 约定与选项

Item 项目	Meaning 含义
-v	show every test name and result
-q	progress dots and a summary only
-k expression	run only tests whose names match
pytest.raises(Error)	expect the given exception
pytest.raises(Error, match="...")	also check the message
pytest.approx(x) == y	float comparison without precision surprises

84

Slide 84 · pytest Conventions and Options 约定与选项

浮点比较值得单独说：0.1 + 0.2 == 0.3 实测为 False，二进制小数存不下这三个数。

要比较浮点结果就用 pytest.approx 或给一个容差。

| Notes and a Real Suite 讲义示例与真实测试



- The notes run as single files
- pytest needs discovery
- So the notes show a checking loop
- The real pytest output is quoted verbatim

85

Slide 85 · Notes and a Real Suite 讲义示例与真实测试

两者检验的是同一件事：期望值与实际值不一致时，给出足够的信息定位问题。

讲义保留可单文件运行的说明，同时给出真实 pytest 输出，不手写终端文本。

| Platform Grading 平台判分



Situation 情形	Result 结果
the function returns a value	compared against the expected value
the function prints instead	returns None; None == expected is never true
the expected outcome is an exception	the test expects the type, not a return value

86

Slide 86 · Platform Grading 平台判分

平台调用你的函数，把返回值和期望值比较。

这就是从 L03 起那句「函数要 return 不要 print」背后的机制。

I Edge Case Checklist 边界清单



Category 类别	Examples 例子
normal	a typical input
empty	an empty list, an empty string
single	exactly one element
boundary	0, negatives, largest and smallest, equal bounds
repetition	duplicate elements, consecutive spaces
failure	does invalid input raise as agreed

87

Slide 87 · Edge Case Checklist 边界清单

每个分支至少一个用例。

让 AI 帮忙补用例时，仍然要自己先把这张表列一遍：L08 说了，「空」和「一个」正是最容易漏掉的两类。

| textstats Under Test 测试第三方模块



Actual output 实跑输出

```
2  
ValueError
```

- Repeated spaces do not make an empty word 连续空格不算单词
- The docstring never pins down the empty string
- The discrepancy is the finding

88

Slide 88 · textstats Under Test 测试第三方模块

文档只说了「返回最长的单词」，没有约定空字符串。

测试的价值就是把没约定的输入摆出来，让差异变成可见的事实。

| Prime Counting · Brute and Sqrt 暴力与平方根剪枝



- Brute
- `for...else` runs the else when the loop was never broken
- Sqrt
- Measured $n = 2000$: brute 0.0042 s, sqrt 0.00047 s

89

Slide 89 · Prime Counting · Brute and Sqrt 暴力与平方根剪枝

`for...else` 在这里正好合用：循环没有被 `break` 打断，就是一个因子都没找到， p 是素数。

两种写法在 $n \leq 2$ 时都返回 0。



| Prime Counting · $6k \pm 1$ 剪枝

- Rule out multiples of 2 and 3 first
- What remains is $6k \pm 1$
- Candidates become 5, 7, 11, 13, ...
- Order unchanged at $O(n \sqrt{n})$, about three times faster

90

Slide 90 · Prime Counting · $6k \pm 1$ 剪枝

阶数没变，只是试除次数少了约三分之二。

这是「同阶不同常数」最直观的例子，和第67页的渐近含义呼应。



| Prime Counting · Sieve 筛法

- Mark the whole range at once
- Start marking at $p * p$
- $O(n \log \log n)$ time, $O(n)$ space
- **Measured** $n = 100000$ in 0.00212 s, the fastest of the five

91

Slide 91 · Prime Counting · Sieve 筛法

筛法一直最快，代价是那份长度为 n 的标记数组。

小范围时它的初始化成本相对更高，这一点在实测表里也能看出来。

| Prime Counting · Miller–Rabin



- Write $n-1$ as $d \times 2^r$
- Test $\text{pow}(a, d, n)$ for a few bases
- $\text{pow}(\text{base}, d, n)$ is the three-argument built-in
- Bases 2 and 3 are certain for $n < 1,373,653$
- $O(\log^3 n)$ per number, $O(1)$ space

92

Slide 92 · Prime Counting · Miller–Rabin

适用范围必须说清楚：用底数 2 和 3， $n < 1,373,653$ 时结论是确定的，本题域内 $n \leq 100000$ 成立。

超出这个界之后，这几个底数只是概率性的证据，不能称对任意大整数确定。

I Timing Fairly 公平计时



Requirement 要求

Reason 理由

verify correctness first

all five must agree at the same small size

fix the size and the machine

otherwise the numbers are not comparable

warm up, then take the minimum

avoid scheduler noise

a slow method may use a smaller size but it may not swap in another algorithm

93

Slide 93 · Timing Fairly 公平计时

报告里的耗时要要有可比性，所以这几条是硬要求。

本节的秒数来自构建环境一次测量，只用来比较量级与倍数，不承诺在你的机器上得到同样的数字。



| Solo Assignment 2 · Specification 提交规格

Item 项目	Agreement 约定
five functions	count_primes_brute / _sqrt / _6k / _sieve / _mr
range	$0 \leq p < n$; $0 \leq n \leq 100000$
small values	$n \leq 2$ returns 0
invalid input	a negative n raises ValueError
report	method notes, timing table, call counts, usage declaration
weight	three solo assignments: 15% total; this one 5%

94

Slide 94 · Solo Assignment 2 · Specification 提交规格

按现行配置覆盖第10、11周，截止读统一校历。

完整参考解不公开，助教核对各方法是否为真实实现；报告里要写清楚各自能完成的规模。



| Lab 11 · Required Boundaries 必须满足的边界

Entry 入口	Required boundary 边界
w11-complexity-measure	the helper returns the smallest of repeats
w11-string-build	the kept-prefix version is the comparison
w11-memoize	count_ways and count_ways_memo agree
w11-safe-convert	unexpected failures must escape
w11-config-review	the file is closed on every path
w11-custom-exception	a rejected update keeps the old state
w11-tests	the suite must fail a wrong module

95

Slide 95 · Lab 11 · Required Boundaries 必须满足的边界

原始分不改学期权重或日历；测试题的评分不只看测试是否通过，也看是否覆盖了空、单个、边界和异常。

| Summary and Connection 小结与衔接



Thread 主线	Answer 回答
Too slow	count steps and keep the leading term
It crashes	catch what you can handle, raise what you cannot
Quietly wrong	assertions inside, tests outside
This week	five prime methods, with correctness before timing
Next	randomness, Unicode and files — messier input

96

Slide 96 · Summary and Connection 小结与衔接

本讲的三种证据会继续使用：增长阶比较算法，失败边界划分责任，可执行检查写下期望。

第12周处理随机数、编码与文件，那里的输入更不干净，正好检验本讲的失败边界与测试写法。