第 6 讲 · 讲义

元组、字典、字符串与集合

课件的每一页，配上讲这一页时说的话。幻灯片是网页原生渲染的，文字可以选中、可以搜索，也可以直接打印。



第 1 页 · Tuples, Dicts, Strings and Sets

> 上一讲用列表存放和处理一批数据，重点是引用、修改和拷贝。今天继续处理一批数据，但任务变了：固定一条记录、按姓名查成绩、解析文本、比较两个名单。
>
> 不同任务适合不同容器。每介绍一种容器，都先看列表在哪里不方便，最后再把五种容器放在一起比较。

Where Lecture 5 Ended

- A list stores references; `b = a` does not copy it
- Lists are mutable; integers and strings are immutable
- `copy()` copies one level; nested objects may still be shared
- Comprehensions build a new list; do not change length while iterating

> 第 5 讲的引用模型今天继续使用。元组、字典、集合里的元素也都是对象。判断一次操作的影响，要看它是在修改共享对象，还是让一个名字改指别的对象。
>
> 今天有一个需要特别核对的边界：外层容器不能修改，是否意味着里面所有对象都不能修改?

Campus Movie Night 校园电影夜

One event, several kinds of data.



A ticket identifies **one seat**: row and number.

A snack order contains a list that **can grow**.

> 先看电影夜的实际任务。票上有排号和座号，两项共同确定一个座位；零食清单却允许继续加东西。这里先不用代码。
>
> 一条记录的字段可以固定，而记录引用的清单仍可能变化。下一页用一个简短操作，让学生判断这种组合会发生什么。

随堂小测 · 订单修改



扫码作答

1 执行报错后，零食清单有没有改变？

https://taohuang.info/cs1602/zh/quiz/6/

> 手机中有两行代码：order 保存一个座位元组和一个零食列表，然后执行 order[1] += ["popcorn"]。问题同时判断是否报错、列表是否改变。先收预测，不提示“元组元素引用不能替换”。
>
> 这题会在元组一节末尾显式回顾。课件导出的二维码页保留问题主题，完整题面随小测显示。

Containers and Their Tasks 容器与任务

Learning Goals 学习目标



list	tuple	dict	set	str
Keep order 顺序与增删	Fix a record 固定记录与引用	Find by key 键值设计与查找	Test membership 成员判断与名单	Process text 拆分与格式化

5

第 5 页 · Containers and Their Tasks 容器与任务

> 这页列的是学习目标。列表继续负责顺序和增删；元组用于固定记录；字典建立键值对应；集合处理成员关系；字符串负责文本。
>
> 今天重点是根据任务设计表示，并解释操作为何有这样的结果。方法细节保留在参考页，课堂先沿电影夜任务理解结构。

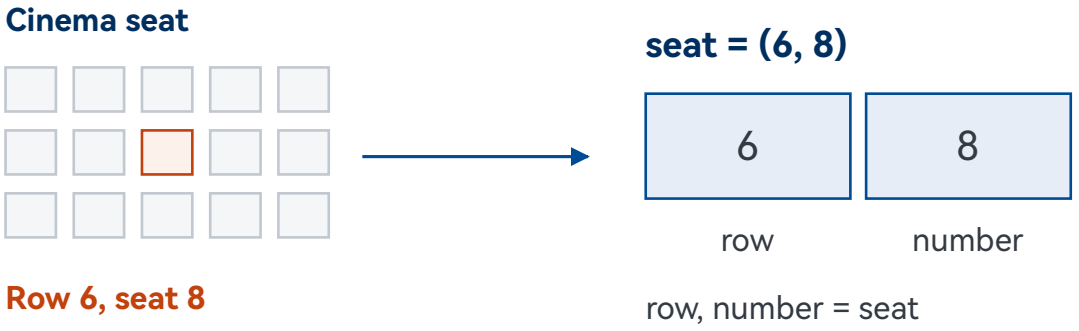
PART 1 OF 6

Tuples

> 列表可以增删。一个坐标却通常只有 x 和 y 两项，不应该有人再 append 第三项。元组用来固定这一组位置。它的访问方式和列表很像，但元素引用不能替换。

A Fixed Sequence（元组）

A seat record keeps **row and number together**.



list 也能保存坐标；tuple 用来限制这条记录的原地修改。

回到电影票。第 0 项是排号，第 1 项是座号，两项作为一个整体传递。用 seat = (6, 8) 表示，再用 row, number = seat 取出。

列表也能保存这两个数字，但列表允许原地替换和增删。使用元组可以阻止这些操作，表达当前记录不应被原地改写的约定。元组并不自动检查排号是否合法、是否永远只有两项；这是程序的数据约定。下一页直接尝试替换其中一个位置。

Item Assignment Is Not Supported

A tuple rejects **replacement of an item** at runtime.

code

```
seat = (6, 8)
seat[0] = 9
```

Actual exception 实际异常

```
TypeError: 'tuple' object does not support item assignment
```

> Construct a new tuple when different contents are needed.

> 给元组元素赋值会报 TypeError。这是执行到这一行时发生的运行时错误，不是语法错误。
>
> 元组没有 append、remove、sort 这类原地修改方法。若要得到不同的内容，需要构造另一个元组。

The Comma Makes the Tuple

The **comma** creates a tuple; parentheses can group an expression.

code

```
1  a = (1)
2  b = (1,)
3  c = 1, 2, 3
4  print(type(a).__name__)
5  print(type(b).__name__)
6  print(c)
7  print(tuple())
```

Actual output 实跑输出

```
int
tuple
(1, 2, 3)
()
```

单元素元组需要逗号：`(1,)`。`(1)` 仍是整数。

括号本来就用于分组，所以 (1) 仍是整数。要构成只有一项的元组，需要逗号。多个值用逗号隔开时，括号通常可以省略；空元组写 () 或 tuple()。

这是返回单个元素的元组时最容易漏掉的标点。

Packing and Unpacking（打包与解包）

Packing groups values; unpacking assigns them to **matching names**.

code

```
point = 3, 4
x, y = point
print(x, y)
(a, b), c = ((1, 2), 3)
print(a, b, c)
first, *rest = (1, 2, 3, 4)
print(first, rest)
```

Actual output 实跑输出

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

Nested structures must match. A starred target collects a list of remaining items.

逗号把多个值打包成元组；左边写几个名字，可以把元素逐个解包。嵌套结构也可以对应解包。

最后一行赋值里的星号收集剩余元素，得到列表 [2, 3, 4]。它和函数参数的 *args 都能收集多个值，但 *args 收到元组，赋值解包的星号收到列表。

Unpacking Needs Matching Lengths

The number of values must match the receiving names.

code

```
1  a, b = (1, 2, 3)
```

Actual exception 实际异常

```
ValueError: too many values to unpack (expected 2)
```

不匹配会报 ValueError。带星号的名字收集剩余值，得到 list。

11

第 11 页 · Unpacking Needs Matching Lengths

左边两个名字、右边三个值，对应不上，报 ValueError。数量不确定时才用带星号的解包；数量已知时明确写出来，更容易发现数据不符合要求。

Returning a Record

A function can return **one tuple** containing several related results.

code

```
1  def bounds(nums: list[int]) -> tuple[int, int]:
2      return min(nums), max(nums)
3
4  result = bounds([3, 9, 5])
5  print(result)
6  low, high = result
7  print(low, high)
```

Actual output 实跑输出

```
(3, 9)
3 9
```

Keep the returned tuple as one record or unpack it into two names.

12

第 12 页 · Returning a Record

第 3 讲的“返回多个值”，实际上返回一个元组。调用者可以把它当作一个整体，也可以解包。

这里约定输入非空；空列表的处理不是这一页的主题。两个返回值的先后位置固定，适合用元组表示。

Fixed Slots, Mutable Contents

The tuple keeps the same references; its **snack list can still change**.

code

```
order = ((6, 8), ["cola"])
order[1].append("popcorn")
print(order)
```

Actual output 实跑输出

```
((6, 8), ['cola', 'popcorn'])
```

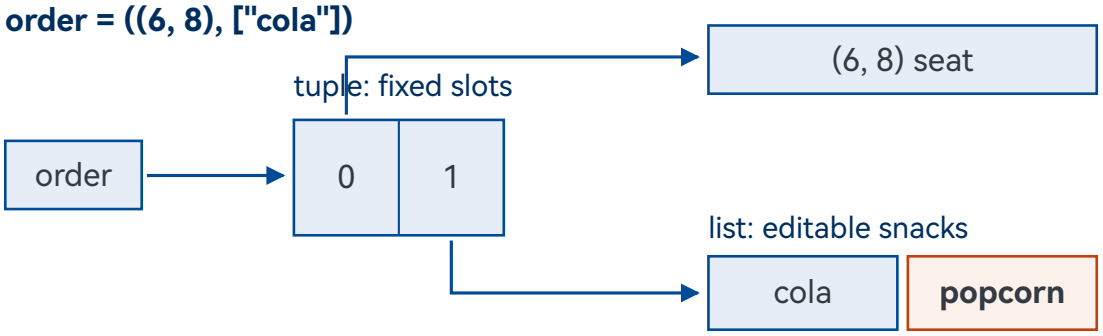
append 修改零食列表，不会替换 order 中指向列表的引用。

用订单再次区分两层对象。append 修改零食列表，元组第 1 项仍然指向这个列表。座位由第 0 项引用的元组表示。

这里直接调用列表方法，没有给元组位置赋值。随后先看普通列表上的 +=，再回顾开场题。

Two Levels of References

Follow the reference to identify **which object changes**.



order[1].append("popcorn")：列表改变，元组仍指向同一个列表。

> 左边 order 指向元组。第 0 项引用座位元组，第 1 项引用零食列表。append 改变列表内容，但没有替换这两个引用。
>
> 不可变限制的是外层元组的这些位置，不能据此推断所有嵌套对象都不可变。

List + and +=

+: new list

```
a = ["cola"]
b = a
a = a + ["popcorn"]
print(a)
print(b)
print(a is b)
```

output

```
['cola', 'popcorn']
['cola']
False
```

+=: same list

```
a = ["cola"]
b = a
a += ["popcorn"]
print(a)
print(b)
print(a is b)
```

output

```
['cola', 'popcorn']
['cola', 'popcorn']
True
```

列表的 + 生成新列表；+= 扩展原列表，别名也会看到变化。

> 先用普通列表讲清 +=。左边加号产生新列表，a 改指新对象，b 保留原列表；右边 += 在原列表上扩展，所以两者仍是同一个对象。
>
> 这一步是理解开场题的前提。不同类型的复合赋值可能有不同实现，不能把列表的 += 当成简单的文本替换。接下来回到 order[1] 这个赋值目标。

Opening Quiz Review 开场题回顾

Q1 · TypeError; the snack list has changed.

Opening Q1 开场原题

```
order = ((6, 8), ["cola"])
order[1] += ["popcorn"]
```

Actual exception 实际异常

```
TypeError: 'tuple' object does not support item assignment
```

After exception 异常后状态

```
((6, 8), ['cola', 'popcorn'])
```

1 · Read 取得列表

Get the list referenced by `order[1]`.

2 · Extend 原地扩展

List += adds `"popcorn"` to that list.

3 · Assign 尝试写回

Tuple slot assignment raises `TypeError`.

报错不会自动撤销之前已经完成的修改。元组的两个引用仍然没有被替换。

16

第 16 页 · Opening Quiz Review 开场题回顾

现在正面回顾开场第一题，正确选项是 C：报 TypeError，但零食列表已经增加了 popcorn。左边异常和最终对象状态都来自 Python 3.13 实跑；状态是在捕获异常后另外检查的，不表示普通脚本报错后会自动继续执行。

对这个列表上的 +=，先取得列表，再原地扩展，最后还要把结果写回赋值目标 order[1]。最后一步违反元组的限制，但前一步已经完成。外层两个引用始终未被替换。这里不教授 try/except，构建时只用它保存异常后的观测状态。

A Separate Snack Snapshot



Choose a **separate immutable record** when later edits must not affect it.

code

```
1  snacks = ["cola"]
2  order = ((6, 8), tuple(snacks))
3  snacks.append("popcorn")
4  print(order)
5  print(snacks)
```

Actual output 实跑输出

```
((6, 8), ('cola',))
['cola', 'popcorn']
```

本例元素是字符串。tuple() 只收集这一层的引用，不会递归冻结任意嵌套对象。

17

第 17 页 · A Separate Snack Snapshot

如果需求是保存当时的零食记录，就在边界上把零食转为一个元组。这些元素都是字符串，后续往原列表里追加不会改变记录。

不能把 tuple() 当成任意嵌套数据的深拷贝；这里成立是因为这一层收集的是不可变字符串。需要修改座位时可以构造新订单，再明确让名字指向新订单。

PART 2 OF 6

Dictionaries

一条记录可以用元组表示。但一个班有很多记录，知道“第 37 条是谁”并不方便。要查询的是某个姓名或学号对应的成绩，这需要按键访问的字典。

Finding an Order 按票号查订单

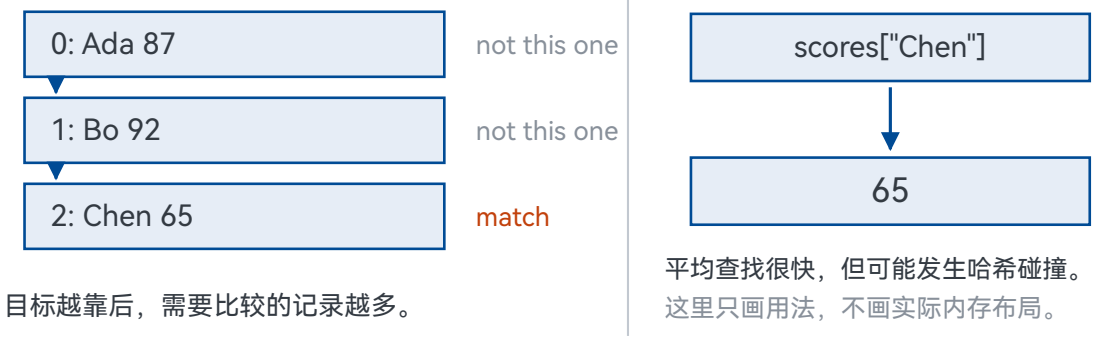
The question is **which order matches this ticket?**



A list search checks records **one by one**. A dictionary maps **ticket ID → order**.

> 电影夜有很多订单。工作人员拿到票号时，要找这张票对应哪份零食，而不是第几份订单。左图是逐条翻记录，右图是按标识定位订单的任务。
>
> 右图只帮助理解按编号查找的需求，不代表真实计算机哈希表就是实体柜子。随后用精确的教学模型解释内部过程。

By Position or by Key



左边是把记录放进列表，再逐个比较姓名。目标越靠后，需要检查的记录越多。右边写 scores["Chen"]，直接表达“查这个姓名的成绩”。

字典使用哈希表，通常能很快查到；哈希碰撞仍然可能发生。这里画的是访问方式示意，不是实际 CPython 的槽位或地址。

Scanning Order Records

A list stores order; finding a key may require **scanning every record**.

code

```
orders = [(17, "cola"), (10, "tea"),
          (25, "popcorn")]
checks = 0
for ticket, snack in orders:
    checks += 1
    if ticket == 25:
        print(snack, checks)
        break
```

Actual output 实跑输出

```
popcorn 3
```


> 用已有知识先做一次列表查询。目标 25 在第三条，因此比较三次。若目标不在列表里，就需要检查所有记录。
>
> 把示例扩到更多订单时，扫描的比较次数会随目标位置和记录条数增长。这里先数操作，不依赖机器计时。

Dictionary Lookup 字典查找机制

key → hash → candidate slot → **key check** → value

- **Hash 哈希值**: compute an integer from the key.
- **Index 索引**: use it to choose a candidate slot.
- **Compare 比较**: verify the stored key before returning its value.
- **Probe 探查**: try another slot if different keys collide.

> 相同的键必须能再次沿查找规则找到；碰撞时不能覆盖另一个键。

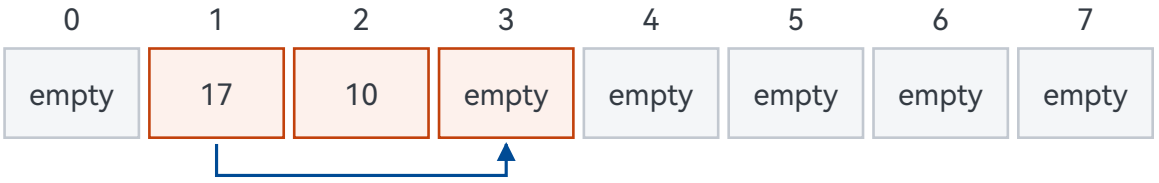
> 哈希表不是直接把姓名或票号当作连续数组下标。先从键算出哈希值，再据此确定候选位置，最后核对键。
>
> 两个不同键可能得到同一个哈希值，也可能在缩到有限槽位时落在同一个位置，因此必须处理碰撞。查找仍要比较键，不能只比较哈希值。接下来用可手算的模型展示。

A Collision 碰撞与插入

Different keys can choose **the same initial slot**.

key 25 → home = 25 % 8 = 1

Teaching model 教学模型：modulo + linear probing



17 ≠ 25 → 10 ≠ 25 → empty: insert 25 here

> 模型仅用来解释机制；CPython 的哈希与探查规则更复杂。

第 23 页 · A Collision 碰撞与插入

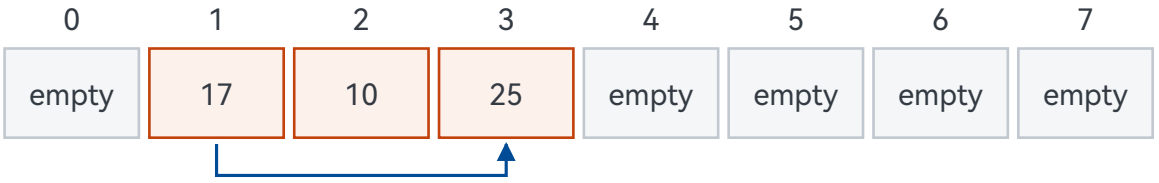
> 教学模型使用 key % 8 决定起始槽位，冲突后向右逐格探查并循环。表中已有 17 和 10。插入 25 时从槽位 1 开始，先遇到 17，再遇到 10，槽位 3 为空，因此放到那里。
>
> 这是故意简化的线性探查模型。真实 CPython 会计算对象哈希，并使用更复杂的探查序列，不能把本页算法当成 Python 内部的逐行实现。

Lookup after a Collision 碰撞后的查询

A lookup follows the **same probe rule** used for insertion.

key 25 → home = 25 % 8 = 1

Teaching model 教学模型：modulo + linear probing



17 ≠ 25 → 10 ≠ 25 → 25: found

模型仅用来解释机制；CPython 的哈希与探查规则更复杂。

第 24 页 · Lookup after a Collision 碰撞后的查询

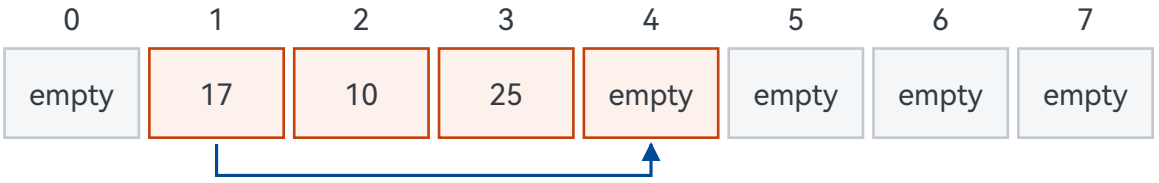
> 查找 25 也从槽位 1 开始，不能看见不是 25 就说没有。按相同探查规则检查后，在槽位 3 找到。返回这条记录关联的值。图中只保留键，让探查过程看得清楚。

An Unsuccessful Lookup 查找失败

In this model, a **never-used empty slot** ends an unsuccessful search.

key 33 → home = 33 % 8 = 1

Teaching model 教学模型：modulo + linear probing



17 ≠ 33 → 10 ≠ 33 → 25 ≠ 33 → empty: absent

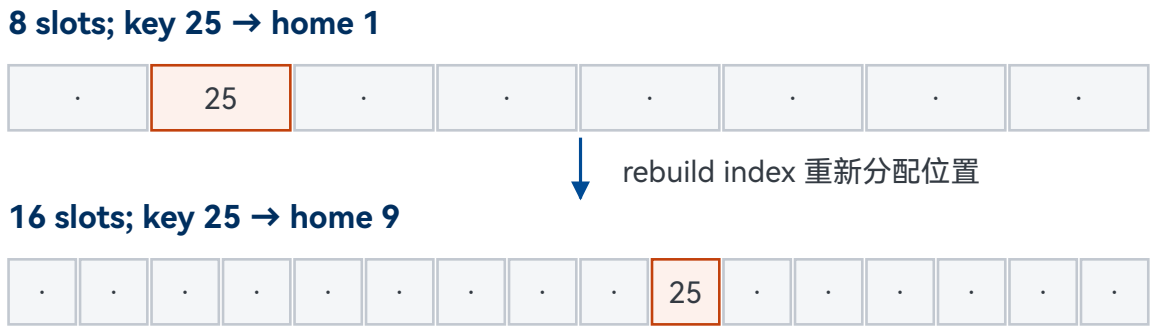
模型仅用来解释机制；CPython 的哈希与探查规则更复杂。

查找 33，同样从槽位 1 出发，依次核对 17、10、25，最后遇到从未使用过的空槽位 4，因此确定没有 33。

教学模型暂不处理删除。真实哈希表删除后可能保留占位标记，不能把“已删除”与“从未使用过”混为一谈；这一边界在参考页保留。

Growth and Reindexing 扩容与重建索引

More empty slots reduce crowding; growing the table requires **rebuilding its index**.



平均查找通常高效；碰撞和扩容都需要额外工作。

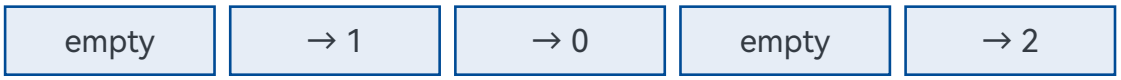
槽位太拥挤时，碰撞和探查都会增多。扩容提供更多空槽位，但候选位置与表大小有关，原来的记录需要重新建立索引。图中 25 在 8 槽时起点是 1，在 16 槽时起点是 9。

本页只展示一个键的起点变化，不把扩容画成“直接在旧数组后面加几个空格”。扩容本身需要处理已有记录，因此某一次插入可能比平时贵。日常平均高效不等于每次操作都只有一步。

Lookup and Insertion Order 查找与插入次序

Fast lookup and ordered iteration can use **different parts of the representation**.

Lookup index 查找索引



Entries in insertion order 插入次序



查找依据键；遍历依据插入次序。两种职责可以分开实现。

概念图：不表示所有 Python 实现或所有字典的精确内存布局。

27

第 27 页 · Lookup and Insertion Order 查找与插入次序

学生可能会问：既然按哈希位置存，为什么字典还能保持插入顺序？这两项职责可以分开。上面的索引用于找到记录，下面的条目按插入顺序遍历。图中的槽位和编号只是概念示意，不是实际 CPython 内存转储。

CPython 常用的紧凑字典就采用索引指向条目的组织方式，具体还有不同存储形式。本课程只需要知道按键查找和保持插入顺序并不矛盾。后面的代码会验证更新已有键不移动位置。

Keys and Values（键与值）

A dictionary maps a **key** to a value; `in` checks keys.

code

```
1  scores = {"Ada": 87, "Bo": 92}
2  print(scores["Ada"])
3  print(len(scores))
4  print("Ada" in scores)
5  print(87 in scores)
```

Actual output 实跑输出

```
87
2
True
False
```

Searching values requires a separate traversal or a reverse mapping.

> 花括号里，每一项写成键、冒号、值。按键取值用方括号。一个键只对应一个值，多个键可以对应相同的值。
>
> 对字典使用 in，判断的是键，不是值。所以 87 in scores 是 False，尽管 87 是其中一个成绩。

Nested Dictionaries（嵌套字典）

Each lookup selects one level: **student, then subject**.

code

```
results = {"Ada": {"Math": 87, "Physics": 92}}
print(results["Ada"])
print(results["Ada"]["Math"])
results["Ada"]["Math"] = 90
print(results["Ada"]["Math"])
```

Actual output 实跑输出

```
{'Math': 87, 'Physics': 92}
87
90
```

Two brackets mean two lookups. Updating one subject preserves the other subjects.

> 字典的值也可以是字典。第一层键是学生姓名，得到这个学生的成绩字典；第二层键是科目，得到该科成绩。
>
> 两个方括号就是两次已经讲过的按键查找。Lab 6 的解析成绩单用这个结构，不需要先学新的访问语法。

KeyError

A missing key raises `KeyError`; a missing record may indicate a data error.

code

```
1  scores = {"Ada": 87}
2  print(scores["Chen"])
```

Actual exception 实际异常

```
KeyError: 'Chen'
```

> Choose whether absence means invalid data or an expected case.

第 30 页 · KeyError

不存在的键会报 KeyError。报错给出的 Chen 是程序尝试查询的键。

有时缺失意味着数据错误，应该暴露出来；有时缺失很正常，比如计数器第一次见到一个词。下一页讲后者怎么处理。

get() Does Not Insert

`get()` returns a fallback for a missing key **without inserting it**.

code

```
1  scores = {"Ada": 87}
2  print(scores.get("Chen"))
3  print(scores.get("Chen", 0))
4  print(scores)
```

Actual output 实跑输出

```
None
0
{'Ada': 87}
```

查询成绩时，“缺记录”与“考了零分”需要区分；计数从 0 开始则很合理。

> get 找得到就返回值，找不到就返回默认值；没写默认值时返回 None。它不会修改字典。
>
> 默认值要按问题决定。查询成绩时用 0 可能把“缺记录”误认为“考了零分”；统计次数时，第一次没出现过按 0 计才合适。

setdefault() Can Insert



`setdefault()` inserts only when the key is missing, then returns the value.

code

```
1  scores = {"Ada": 87}
2  print(scores.setdefault("Chen", 0))
3  print(scores)
4  print(scores.setdefault("Ada", 100))
5  print(scores["Ada"])
```

Actual output 实跑输出

```
0
{'Ada': 87, 'Chen': 0}
87
87
```

If the key exists, its old value is returned and left unchanged.

> setdefault 在键不存在时把默认值写进去，再返回这个值；键存在时返回原来的值，不覆盖。
>
> 最后两行说明，Ada 已有 87，传入默认值 100 并不会把它改成 100。这是它和普通赋值的区别。

Adding, Updating and Removing

Assignment inserts or updates; pop( ) removes **and returns** a value.

code

```
1  scores = {"Ada": 87, "Bo": 92}
2  scores["Chen"] = 65
3  scores["Ada"] = 90
4  del scores["Bo"]
5  print(scores)
6  print(scores.pop("Chen"))
7  print(scores)
```

Actual output 实跑输出

```
{'Ada': 90, 'Chen': 65}
65
{'Ada': 90}
```

A new dictionary key inserts an entry; an out-of-range list index raises IndexError.

> 同一个赋值语句，键不存在时新增，存在时更新。del 按键删除；pop 按键删除并返回值。
>
> 字典和列表不一样：列表不能通过给越界下标赋值来增加元素，字典则可以通过新键赋值增加一项。

Insertion Order（插入顺序）

Iteration follows **insertion order**, not sorted key order.

code

```
d = {"Bo": 92, "Ada": 87}
d["Bo"] = 95
d["Chen"] = 65
print(list(d))
print(d.popitem())
print(list(d))
```

Actual output 实跑输出

```
['Bo', 'Ada', 'Chen']
('Chen', 65)
['Bo', 'Ada']
```

更新已有键不移动位置；删除后重新插入，会排到末尾。

> 现代 Python 字典保持插入顺序，不是排序后的顺序。更新已有键 Bo 的值不改变位置，新增 Chen 排在最后。
>
> popitem 返回并删除最后插入的一对键值。旧课件说“字典无序”，今天不能沿用。

Keys, Values and Items

`items()` supplies `(key, value)` pairs that can be unpacked in a loop.

code

```
1  scores = {"Ada": 87, "Bo": 92}
2  print(list(scores.keys()))
3  print(list(scores.values()))
4  print(list(scores.items()))
5  for name, mark in scores.items():
6      print(name, mark)
```

Actual output 实跑输出

```
['Ada', 'Bo']
[87, 92]
[('Ada', 87), ('Bo', 92)]
Ada 87
Bo 92
```

> These methods return views. `list()` is used here only to display their contents.

> keys 取键，values 取值，items 取键值对。items 每一项是一个二元素元组，循环里直接解包成 name 和 mark。
>
> 这些方法返回视图，不直接返回列表。这里套 list 是为了展示；实际遍历不需要先变成列表。

Changing Size During Iteration

Adding or deleting keys changes the structure being traversed.

code

```
scores = {"Ada": 87, "Bo": 92}
for name in scores:
    if scores[name] < 90:
        del scores[name]
```

Actual exception 实际异常

```
RuntimeError: dictionary changed size during iteration
```

遍历时不要增删键。更新已有键的值与改变键集合是两种操作。

遍历字典时不应增删键：可能报 RuntimeError，也可能遗漏原有项或遍历新增项。本例删除后改变了大小，因此报错。错误发生在运行时，语法本身合法。

只更新已有键的值，不改变键集合；这里的删除会改变大小，所以不能和遍历同一个字典混在一起。

Keep the Iteration Separate

Collect first, delete second: traverse a separate list of keys.

code

```
scores = {"Ada": 87, "Bo": 92, "Chen": 65}
remove = [k for k, v in scores.items() if v < 90]
for name in remove:
    del scores[name]
print(scores)
```

Actual output 实跑输出

```
{'Bo': 92}
```

The second loop visits the separate removal list while changing the dictionary.

> 先遍历 scores，收集要删除的键；再遍历独立的 remove 列表，修改 scores。两次循环操作的对象不同。
>
> 也可以用字典推导生成新的字典：花括号里写键和值。字典推导的完整示例保留在参考页。

Hashable Keys（可哈希的键）

Lookup needs a **stable hash**; equal keys must have equal hashes.

code

```
1  d = {"Ada": 87, (3, 4): "point", 42: "answer"}
2  print(d[(3, 4)])
3  print(hash((3, 4)) == hash((3, 4)))
```

Actual output 实跑输出

```
point
True
```

数、字符串与各项均可哈希的元组可作键；列表、字典、集合不可。

38

第 38 页 · Hashable Keys（可哈希的键）

> 字典键需要可哈希：哈希值在对象生存期间稳定，相等的键必须有相等的哈希值。这里不背具体哈希数值，只观察同一个键产生相同的值。
>
> 数、字符串、元素都可哈希的元组可以作键。列表不行；元组里有列表也不行。下一页演示这个边界。

An Unhashable Tuple

A tuple is hashable **only if every element is hashable**.

code

```
key = ([1, 2], 3)
d = {key: "value"}
```

Actual exception 实际异常

```
TypeError: unhashable type: 'list'
```

外层不可变仍不够：含有 list 的 tuple 不能作字典键。

元组本身不可变，仍可能不可哈希：它含有一个列表。不要用“元组不能修改”推出“所有元组都能作键”。

本课程使用内置类型时，按实际类型和组成元素检查。自定义类型的哈希规则到后面再讨论。

Designing Keys and Values 键值设计

Requirement 需求	Representation 表示	Consequence 结果
Find one order	ticket ID → order	Unique ID avoids repeated names 同名不冲突
Keep all orders	student ID → list of orders	Appending preserves history 保存多次记录
Find by seat	(row, number) → order	Tuple combines a composite key 组合键
Keep latest only	student ID → latest order	Assignment deliberately replaces history 更新覆盖

> 先决定记录的身份和保留规则，再写增删改查。

40

第 40 页 · Designing Keys and Values 键值设计

字典最重要的设计选择是键代表什么、值需要保留什么。姓名可能重复，因此标识订单应选唯一票号。一个学生多次订购时，如果字典只保存单个订单，赋值会覆盖旧订单；需求若是保留全部，就应让值为列表。

排号和座号共同唯一确定座位，可以组合成全由整数构成的元组键。容器不会替程序决定保留历史还是覆盖，必须先明确需求。

PART 3 OF 6

Processing Text

字典已经能保存记录。但原始数据往往是一行文本，要先拆成姓名、科目、分数，再放进容器。第 3 讲讲过字符串的基本操作，这里补处理文本的方法。所有方法都不修改原字符串，但返回类型各不相同。

split(): Text into Fields

`split()` turns one text record into **a list of string fields**.

code

```
1  line = "Ada,Math,87"
2  parts = line.split(",")
3  print(parts)
4  print(parts[0], int(parts[2]))
```

Actual output 实跑输出

```
['Ada', 'Math', '87']
Ada 87
```

Every field is still a string. Convert the score explicitly with `int()`.

42

第 42 页 · split(): Text into Fields

split 把字符串按分隔符拆成列表。这里逗号划分字段，得到三个字符串；成绩要继续用 int 转换。

分隔符也可能出现在字段内部，所以复杂文件要用专门解析器。这一页只处理已经约定好的简单格式。

Whitespace or One Space

`split()` collapses whitespace; `split(" ")` preserves empty fields.

code

```
1  s = "  Ada   Bo\tChen  "
2  print(s.split())
3  print(s.split(" "))
```

Actual output 实跑输出

```
['Ada', 'Bo', 'Chen']
['', '', 'Ada', '', '', 'Bo\tChen', '', '']
```

空字段也可能是有效数据，按输入格式决定切分规则。

> 不带参数的 split 按连续空白切分，并忽略两端空白。空格、Tab、换行都属于空白。
>
> split(" ") 只按单个空格切开，会保留空字段，也不会把 Tab 当分隔。两种写法不是一个规则，不能随意替换。

join(): Fields into Text



The separator joins **strings** into one result, with nothing extra at the ends.

code

```
1  parts = ["Ada", "87", "Math"]
2  print(" / ".join(parts))
3  print("".join(parts))
4  print("\n".join(parts))
```

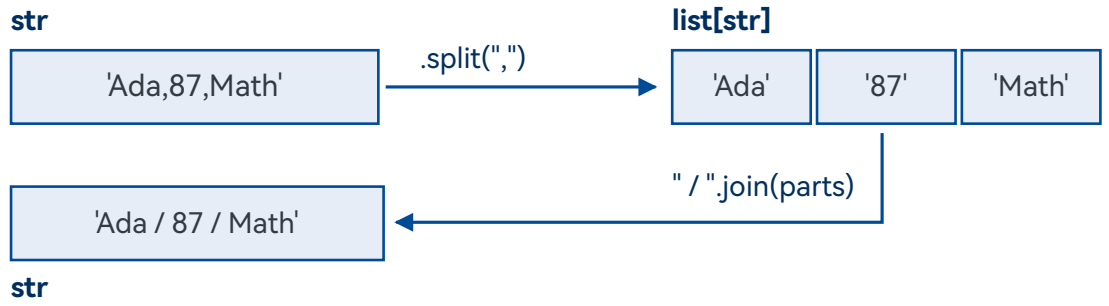
Actual output 实跑输出

```
Ada / 87 / Math
Ada87Math
Ada
87
Math
```

> Each item must be a string. Convert numbers before joining.

> join 写在分隔字符串后面，参数是一组字符串。它在相邻项之间插入分隔符，最后不额外加一个分隔符。
>
> 数要先用 str 转换，不能直接 join 一个整数列表。split 和 join 常一起出现，但不保证互相完全还原，例如默认 split 会丢掉多余空白。

Text, Fields and Text Again



split 返回列表；join 返回字符串。原字符串不变。

第 45 页 · Text, Fields and Text Again

> 从左上角的字符串出发，split 生成右边的字符串列表；再用新的分隔符 join，得到左下角的新字符串。每个框上的类型都写出来了。
>
> 原来的字符串没有变化。图里字段和结果由 Python 实际计算，和上一页使用的是同一类操作。

strip(): Only the Ends

`strip()` removes specified characters **at the ends**, leaving the middle intact.

code

```
1  s = "  Ada   Bo  \n"
2  print(repr(s.strip()))
3  print(repr(s.lstrip()))
4  print(repr(s.rstrip()))
5  print("cat!".strip(".!?"))
```

Actual output 实跑输出

```
'Ada   Bo'
'Ada   Bo  \n'
'  Ada   Bo'
cat
```

chars 参数表示一组字符，不表示必须整体匹配的前缀或后缀。

> strip 去掉两端字符，默认去空白；lstrip 只去左端，rstrip 只去右端。中间三个空格不会消失。
>
> 参数是要去掉的字符集合，不是一个完整前缀或后缀。词频题要去掉词两端的标点，就可以用这个参数。

PART 4 OF 6

Dictionaries in Use

> 前面分别讲了字典和文本处理，现在把它们组合起来。统计文章里的每个词、按首字母分组、按月份查天数，都需要一个键对应一个值。

Word Counts（词频统计）

Choose **word → count**; each occurrence updates one dictionary entry.

code

```
text = "the cat the dog the cat"
counts: dict[str, int] = {}
for word in text.split():
    counts[word] = counts.get(word, 0) + 1
print(counts)
```

Actual output 实跑输出

```
{'the': 3, 'cat': 2, 'dog': 1}
```

The first occurrence starts from 0; subsequent occurrences retrieve and increment the old count.

往年用完整演讲做词频统计，这个应用很好，课堂先用短句看清每一次更新。键是词，值是次数。

第一次遇到一个词，get 默认返回 0，加一后写入 1；下次取出旧次数再加一。没有必要为每一个可能的词预先建变量。

Sorting with a Function

key receives a function; `sorted()` compares **the values it returns**.

code

```
1  counts = {"the": 3, "cat": 2, "dog": 1}
2  def count_of(pair: tuple[str, int]) -> int:
3      return pair[1]
4
5  top = sorted(counts.items(), key=count_of, reverse=True)
6  print(top[:2])
```

Actual output 实跑输出

```
[('the', 3), ('cat', 2)]
```

> key=count_of passes the function object without calling it at this point.

49

第 49 页 · Sorting with a Function

items 提供词和次数的元组。count_of 接受一个元组、返回次数，sorted 按这个结果排序。reverse=True 是从大到小。

key=count_of 传的是函数对象，没有括号；不是此处先调用它。第 3 讲函数也是对象的结论，在这里派上用场。

Tuple Comparison 元组比较

Tuple comparison checks **the first differing pair of elements**.

code

```
1  print((2, "cat") < (3, "ant"))
2  print((2, "cat") < (2, "dog"))
```

Actual output 实跑输出

```
True
True
```

> 排序键先放主要依据，再放并列时使用的次要依据。

50

第 50 页 · Tuple Comparison 元组比较

先比较第 0 项，不同就能决定结果；相同才比较下一项。下面的双重排序依据使用这个规则。

Two Sorting Priorities

A tuple key compares **count first, word second**.

code

```python
pairs = [("dog", 2), ("cat", 2), ("the", 3)]
def order(pair: tuple[str, int]) -> tuple[int, str]:
    return -pair[1], pair[0]

print(sorted(pairs, key=order))
```

Actual output 实跑输出

```
[('the', 3), ('cat', 2), ('dog', 2)]
```

先比较负次数，再比较单词；次数相同的词按字典序排列。

51

第 51 页 · Two Sorting Priorities

> Lab 6 要求次数从多到少，相同时按字母序。排序键返回一个元组，先比负次数，再比单词。次数越大，负数越小，所以默认从小到大恰好把高次数放在前面。
>
> 这两项顺序不能颠倒。本页只用已经讲过的元组比较和具名函数，不提前使用 lambda。

A Lookup Table

A table represents a fixed input–output mapping without a chain of branches.

code

```python
days = {1: 31, 2: 28, 3: 31, 4: 30}
def month_days(month: int) -> int:
    return days[month]

print(month_days(2), month_days(4))
```

Actual output 实跑输出

```
28 30
```

Specify the supported input range and decide how missing inputs should be handled.

52

第 52 页 · A Lookup Table

> 往年把查表和一长串 if-elif 对比，这个例子值得保留。当分支只是在输入和固定输出之间建立对应，用字典就很清楚。
>
> 这里只写前四个月，二月按非闰年处理。实际月份表要写完整；输入范围也要按接口约定检查，不能把这个短例子当成通用日期函数。

PART 5 OF 6

Sets

> 如果只关心一个词是否出现过，而不关心次数，就不需要为每个词保存一个值。集合只记录成员关系。它适合去重、检查出现过没有，以及比较两组成员。

Repeated Scans 重复扫码

Admission asks **“has this ticket appeared before?”**



Keep ticket IDs in a **set** when only membership matters.

Keep a count or event log when **repetitions matter**.

> 回到电影夜入口。一张票第一次扫码可以进场，再扫同一张票需要识别出来。如果只需要判断有没有出现过，集合保存票号就足够，不需要给每张票附一个无意义的 True。
>
> 如果需求是统计扫码次数或调查重复扫码的时间，就不能只保留 set；需要字典计数或列表日志。去重会主动丢失重复次数，这是设计取舍。

Membership at the Gate

A set remembers **which ticket IDs have appeared**, not how many times.

code

```python
seen = set()
for ticket in [17, 25, 17, 10]:
    if ticket in seen:
        print(ticket, "repeat")
    else:
        seen.add(ticket)
        print(ticket, "admit")
print(sorted(seen))
```

Actual output 实跑输出

```
17 admit
25 admit
17 repeat
10 admit
[10, 17, 25]
```


> 每次先查成员，再决定是否加入。17 第二次出现时集合已有这个键，因此能识别为重复。最后集合只保留三个不同票号，不保留四次扫码事件的次序和次数。
>
> 本例是单进程顺序处理，不在这里展开多设备并发闸机的一致性问题。

Set Lookup and Its Trade-offs 集合机制与取舍



- Hash the member, inspect a candidate slot, **compare**, then probe on collisions.
- Adding an equal member leaves **one member**, not two copies.
- Typical average membership lookup is fast; **extra storage** supports the index.
- No positional access, no guaranteed order, and **no occurrence counts**.

> set 的优势来自面向成员查找的组织方式。不要把它当作顺序表。

集合也使用哈希表，但它处理的是成员关系；字典处理键到值的映射。它们共享哈希与碰撞处理的思路，不表示 set 在实现上就是一个把值删掉的 dict。

加入相等的成员不会再增加一项。查找通常平均高效，需要额外空间维持稀疏索引；遇到大量碰撞时也会增加探查。集合的几个限制恰好对应它没有保存的信息：没有位置、没有保证的插入顺序、没有重复次数。

Distinct Members（集合）

A set stores **distinct members**, with no positional order guaranteed.

code

```
1  nums = [3, 1, 3, 2, 1]
2  s = set(nums)
3  print(sorted(s))
4  print(len(s), 2 in s)
5  print(set("banana") == {"b", "a", "n"})
```

Actual output 实跑输出

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

`sorted()` makes this output stable. It does not give the set an internal positional order.

集合不保存重复元素，也不保证顺序。为了让课件输出稳定，这里用 sorted 展示，而不是把集合恰好打印的顺序当规律。

set 可以从列表、元组、字符串、字典等可迭代对象构造；给字典时取到的是键。

No Positional Index

A set answers **membership**, not “which item is at position zero?”.

code

```
1  s = {1, 2, 3}
2  print(s[0])
```

Actual exception 实际异常

```
TypeError: 'set' object is not subscriptable
```

需要展示顺序时，明确排序规则，或另外保存 list。

集合没有按位置访问的接口，s[0] 报 TypeError。集合里恰好有整数 0 时，也不能把下标 0 当成查询这个值。

判断成员用 in，若确实要按顺序处理，先根据明确规则排序或使用列表。

Hashable Members

Set members follow the same **hashability requirement** as dictionary keys.

code

```
1  s = {(1, 2), "Ada", 3}
2  print((1, 2) in s)
3  s.add([4, 5])
```

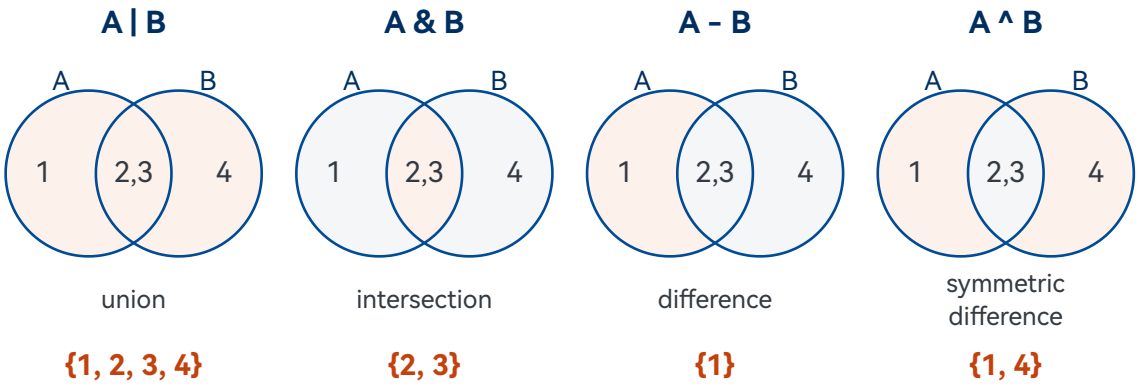
Actual exception 实际异常

```
TypeError: unhashable type: 'list'
```

> set 本身不可哈希；需要固定成员集合时可以使用 frozenset。

> 数、字符串、各项都可哈希的元组可以放进集合，列表不行。这和字典的键要求一致。
>
> set 自己不可哈希，因此也不能直接作字典键。若需要不可变集合，Python 提供 frozenset，列在参考页。

Four Set Operations（集合运算）



第 60 页 · Four Set Operations（集合运算）

> 这张图借鉴往年第 43 页的差集和对称差图，并补齐并集、交集，用相同布局对照。A 是 {1, 2, 3}，B 是 {2, 3, 4}，橙色区域是结果。
>
> 并集取两边所有成员；交集只取重叠处；A-B 只取 A 独有成员；对称差取只在一边的成员。数字和结果由实际集合运算算出。

Two Screenings 两场电影的观众

Choose the relation before choosing the operator.

Question 问题	Operation 运算	Meaning 含义
Attended both	A & B	Intersection 交集
Attended either	A | B	Union 并集
Only the first	A - B	Difference 差集
Exactly one screening	A ^ B	Symmetric difference 对称差

> 把刚才的集合区域图对应到两个电影场次的观众编号。用集合先把自然语言需求准确写成关系，再写符号。
>
> “只来过一场”是对称差，和“只来过第一场”不同。用几个真实编号手算后，再检查代码结果。接下来的课程名单例子使用同样关系，说明这个表示能迁移到别的任务。

Operations on Course Lists

Set operations describe **relationships between two groups**.

code

```
a = {"Math", "Physics", "Python"}
b = {"Physics", "Art", "Python"}
print(sorted(a & b))
print(sorted(a - b))
print(sorted(a | b))
print(sorted(a ^ b))
```

Actual output 实跑输出

```
['Physics', 'Python']
['Math']
['Art', 'Math', 'Physics', 'Python']
['Art', 'Math']
```

交集是共同课程；差集只取左边独有；对称差取两侧独有。

> 把集合运算用在两个班的课程名单上。交集是共同课程，差集是只有 A 班上的课程，并集是至少一个班上的课程，对称差是仅一个班上的课程。
>
> 符号和第 4 讲的位运算相同，但操作对象不同，含义是集合关系。

Deduplication with Order

A **list keeps order** while a **set checks whether a value has appeared**.

code

```
1  nums = [3, 1, 3, 5, 1, 7]
2  seen: set[int] = set()
3  out: list[int] = []
4  for x in nums:
5      if x not in seen:
6          seen.add(x)
7          out.append(x)
8  print(out)
9  print(list(dict.fromkeys(nums)))
```

Actual output 实跑输出

```
[3, 1, 5, 7]
[3, 1, 5, 7]
```

顺序由 out 保存，seen 只负责成员查询。两个容器各自承担一种任务。

63

第 63 页 · Deduplication with Order

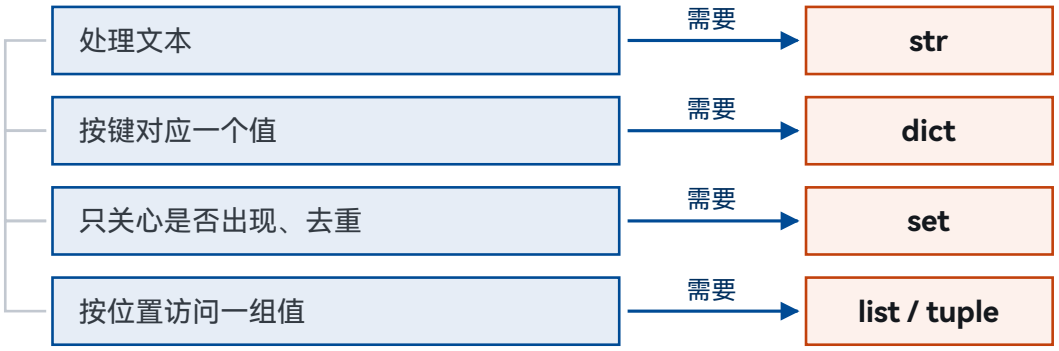
> Lab 6 的保序去重，把两个任务分开：列表保存首次出现的顺序，集合快速判断是否见过。不能把 set 转回列表就认为顺序还在。
>
> dict.fromkeys 也能保留首次插入顺序。方法表中的这个工具和手写循环得到同样结果，但实验题明确要求用集合加速成员判断。

PART 6 OF 6

Choosing and Sharing Containers

> 现在五种容器都见过了。最后把它们放在一起，并检查几条共同规则：真假值、遍历、参数传递，以及修改和重新绑定。

A Container Decision Guide



list 可增删；tuple 的槽位固定。先按任务选择，再考虑快慢。

第 65 页 · A Container Decision Guide

> 先判断任务：文本用 str；按键对应一个值用 dict；只关心成员用 set；按位置访问序列再选择 list 或 tuple。
>
> 这是一张初步判断图，不是永远只用一个容器。保序去重就是列表和集合配合。元组固定槽位，但内部仍可能有可变对象，选完后还要检查组成。

Container Properties 容器特点

Type	What it preserves 保存什么	What it restricts 限制
list	Positions, repeats; editable 顺序与重复	Membership search scans items 成员查找需扫描
tuple	Positions, repeats; fixed references 固定引用	No item replacement; nested objects may mutate
dict	Key → value; insertion order 对应与次序	One value per equal key; hashable keys
set	Distinct membership 不同成员	No positions, counts, or guaranteed order
str	An immutable text sequence 不可变文本	Text operations; methods do not mutate it

比较容器保存的信息和主动限制的操作。list 和 tuple 都有位置、允许重复；前者可变，后者固定引用。dict 保存键值关系和插入顺序；set 只保存不同成员。str 是处理文本的专门序列。

容器的限制也是接口的一部分。选择 set 就不应再要求它提供原始次数；选择单值字典就需要确认覆盖是否符合需求。

Operation Costs 操作代价

n = number of stored items 元素数

Operation 操作	Sequence 序列	Hash container 哈希容器
Find an item / key	list / tuple: scan up to n items	dict / set: average O(1), worst O(n)
Read by position	list / tuple: O(1)	dict uses a key; set has no index
Keep event order	list appends; keeps repetitions	dict keeps key insertion order; set does not
Build a set of n values	Reading input already takes O(n)	Index building costs time and extra space

> 假设哈希与相等比较代价有界。集合运算通常处理多项成员，不能一概说是 O(1)。

这里 n 表示元素数。O(1) 先按“操作次数通常不随元素数线性增长”理解，O(n) 按“可能逐个处理所有元素”理解，严格分析在 L11。

表中的哈希代价假设键的哈希与相等比较代价有界，并且关注的是通常平均情况。列表按位置读取同样很快，因此不能说字典所有操作都更快。把列表转成集合本身要读取数据并建立索引，一次查询不一定值得转换；反复查询才可能摊平这些准备工作。

A Combined Representation 容器组合设计

One task can need **several containers with different responsibilities**.

- **tuple**: one seat `(row, number)`.
- **list**: scan events in arrival order, including repeats.
- **dict**: ticket ID → order; choose latest or full history.
- **set**: ticket IDs already admitted.

> 每增加一份索引，都要考虑额外空间以及数据更新是否同步。

回到同一个电影夜，把几种容器放进同一份设计中。元组表示一条座位记录；列表保留全部扫码事件的先后和重复；字典按票号找到订单；集合加速查询已经入场的票号。

它们可以共享同样的编号，但各自回答不同的问题。要讨论内存和一致性：多个容器保存相关状态时，需要在同一处理流程中按约定更新，不能只更新其中一个。

Iteration Is Not a Snapshot



Rebinding a loop variable does not replace a container item.

code

```
1  a = [1, 2, 3]
2  for x in a:
3      x = 99
4  print(a)
5
6  scores = {"Ada": 87, "Bo": 92}
7  for name in scores:
8      scores[name] += 1
9  print(scores)
```

Actual output 实跑输出

```
[1, 2, 3]
{'Ada': 88, 'Bo': 93}
```

更新字典中已有键的值不会改变键集合；增删键需要单独处理。

给循环变量 x 重新赋值，不会替换列表的元素；修改 x 指向的可变对象则可能影响原容器。for 并不提前复制整个容器。

字典循环中更新已有值不会改变键集合，这里合法；增加或删除键则不能这样做。要改变结构，先复制、先收集，或新建容器。

Three Minutes with Your Neighbour

按学号查询成绩，同时保留提交顺序。应该用一个容器，还是组合几种？

Explain the representation and which operations it supports.

讨论三分钟。问题没有唯一写法，重要的是说明需求和操作。

参考思路：每条记录可以是元组；列表保存所有提交的先后；字典按学号定位最新记录或一组记录。若一个学生能提交多次，要先决定保存全部还是只保存最后一次，不能让同名键覆盖悄悄替你决定。

随堂小测 · 第 2 轮

扫码作答

1 get 与 setdefault
2 split 的两种规则
3 差集与对称差
4 哈希碰撞后的查找
5 重复事件与容器组合

https://taohuang.info/cs1602/zh/quiz/6/

第 71 页 · 随堂小测 · 第 2 轮

> 这一轮的五题都使用已经讲过的语法，不再重复开场题。
>
> 第二题核对 split 是否产生空字段；第三题区分差集和对称差；第四题沿探查规则查找碰撞键；第五题根据是否保留重复和顺序选择容器组合。
>
> 讲评时让学生先说规则，再用输出验证，而不是记选项顺序。

What to Take Away

- Tuple: fixed references; nested objects may still be mutable
- Dict: key → value; insertion order; hashable keys
- String methods: no in-place mutation; different return types
- Set: membership without duplicates or guaranteed order
- Choose by the task; document sharing and changes to inputs

> 回到开场：order[1] 上的 += 先修改列表，再因元组位置赋值失败而报错。再回到今天的主线：不同任务选择不同容器。
>
> 不能只背“元组不可变、字典查找快、集合去重”。还要能解释边界：元组能不能作键要看元素，字典的有序是插入顺序，集合不保证顺序，字符串方法返回类型各不相同。

Lab 6 — 词频统计与容器选型

- **6-1** invert: swap keys and values; explain repeated values
- **6-2** unique_ordered: list for order, set for membership
- **6-3 / 6-4** common names and groups by first letter
- **6-5** parse text into nested dictionaries
- **6-6** top words: clean, count, sort with two priorities
- **6-7** choose containers and explain the requirements

B 段比较查找代价；C 段练习 Git 远程同步。6-7、B、C 分别有答题卡。

Lab 6 的七题组合今天的工具。6-1 要解释原字典值重复时会发生什么；6-2 明确要求用集合检查是否出现过；6-5 处理字段两端的空白。

6-6 是完整词频统计：统一大小写、去两端标点、跳过空词、计数、按次数和字典序排序。6-7 不写代码，在独立答题卡里写选择理由。每个函数继续写类型提示。

B 段比较列表和集合的查找代价，先计数再计时；C 段承接上周的 Git，练习私有远程仓库与同步。6-7、B、C 各有答题卡，不要把回答混进源码或误写到别题的提交说明里。

Next Lecture

Recursion（递归）

Solve a smaller instance of the same problem, then combine the answer.

> 下一讲把复杂问题分成更小的同类问题，用递归来组织计算。今天的嵌套容器为递归提供了很自然的例子：列表里面还有列表，要处理每一层，就会遇到同一种问题。
>
> 参考页从下一页开始，保留几个常用补充；完整方法与其他细节在课程讲义中查阅。

PRACTICAL PATTERNS

Reference Pages

> 参考区只保留字典推导、分组、共享初值、配对遍历、常用文本方法和格式说明。详细方法清单、旧格式接口与不常用内容在课程讲义中查阅。

Dictionary Comprehension（字典推导）

A dictionary comprehension builds a **new mapping** from selected pairs.

code

```
scores = {"Ada": 87, "Bo": 92, "Chen": 65}
passed = {k: v for k, v in scores.items() if v >= 90}
print(passed)
print(scores)
```

Actual output 实跑输出

```
{'Bo': 92}
{'Ada': 87, 'Bo': 92, 'Chen': 65}
```

The expressions before and after `:` create a key and a value. The original stays unchanged.

> 和列表推导一样，for 提供元素，if 决定是否保留，前面的表达式生成结果。字典推导要生成一对键和值。
>
> 这是一个新字典，原来的 scores 不受影响；是否替换原名字，可以再根据需求决定。

Grouping Orders 按名字分组



Use **key → list** when one key needs to retain several values.

code

```
orders = [("Ada", "cola"), ("Bo", "tea"),
          ("Ada", "popcorn")]
groups = {}
for name, item in orders:
    groups.setdefault(name, []).append(item)
print(groups)
```

Actual output 实跑输出

```
{'Ada': ['cola', 'popcorn'], 'Bo': ['tea']}
```

首次出现时建立列表；之后追加到已有列表，保留组内顺序与重复。

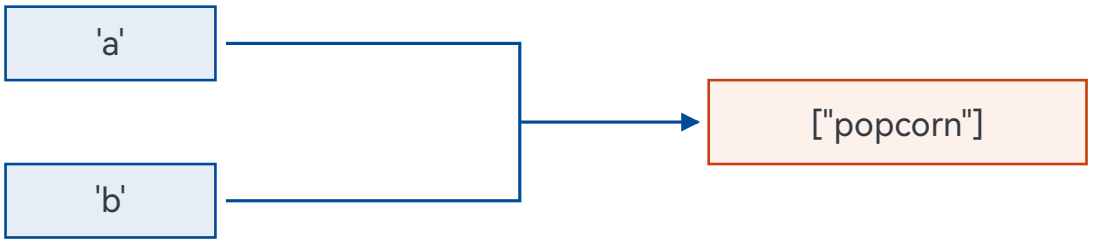
每个名字对应一个列表，保存这个人的所有零食。第一次遇到名字时，setdefault 插入空列表并返回；之后返回已有列表，再追加这一项。

每次调用中的 [] 都是新创建的默认对象，但只有缺键时才会插入。已有名字会继续使用它原来的列表。这个写法适合按类别分组，同时保留组内顺序和重复。

Shared Defaults 共享的初值

`fromkeys(keys, [])` assigns **one shared list** to every key.

dict.fromkeys(["a", "b"], [])



两个键引用同一个列表；通过 a 追加，b 也看得到。

> fromkeys 只接收到一个默认值对象，不会为每个键重新运行一次列表构造。图中的 a 和 b 指向同一列表，因此通过 a 追加 popcorn，b 也能看到。
>
> 需要独立列表时用字典推导，让 [] 在每轮重新求值。下一页实跑对照。

Independent Default Lists

Create `[]` **inside each iteration** when groups need separate lists.

code

```
shared = dict.fromkeys(["a", "b"], [])
shared["a"].append("popcorn")
print(shared)
separate = {key: [] for key in ["a", "b"]}
separate["a"].append("popcorn")
print(separate)
```

Actual output 实跑输出

```
{'a': ['popcorn'], 'b': ['popcorn']}
{'a': ['popcorn'], 'b': []}
```


> 第一份字典共享列表，第二份在推导的每轮创建新列表。因此输出能直接看出 b 是否跟着变化。
>
> 这里不要求背某个“危险方法”，而是再次使用同一个对象引用模型判断构造时创建了几个列表。

zip() and enumerate() 配对与编号

`zip()` pairs values; `enumerate()` adds an index to each pair.

code

```
names = ["Ada", "Bo", "Chen"]
marks = [87, 92]
pairs = list(zip(names, marks))
print(pairs)
for i, pair in enumerate(pairs, 1):
    print(i, pair)
```

Actual output 实跑输出

```
[('Ada', 87), ('Bo', 92)]
1 ('Ada', 87)
2 ('Bo', 92)
```

zip 默认到较短输入结束；必须一一对应时，应先检查长度。

zip 按位置配对，短输入结束时停止。这里 Chen 没有对应分数，不会出现在结果中。list 将配对展开以展示，也可以把 zip 的键值对交给 dict 构造字典。

enumerate 为每一对添加序号，这里从 1 开始。它不会改变 zip 的终止规则；输入应一一对应时，需要检查长度。

Useful String Methods 常用文本方法

Expression 写法	Result 结果	Use / boundary 用途与边界
" DOG! ".strip().lower()	'dog!'	Trim whitespace, then change case 去空白与转小写
"DOG!".strip("!?").lower()	'dog'	Remove chosen end punctuation 指定两端标点
"ticket-17".startswith("ticket-")	True	Test a prefix; return bool 判断前缀
"banana".replace("a", "o")	'bonono'	Return replacement text 返回替换结果
"-5".isdigit()	False	Classify characters, not a number 检查数字字符

方法不会修改原字符串；结果的类型决定下一步可以做什么。

split、strip、join 已在正文讲过，这里只补充几个常用组合。默认 strip 去掉两端空白，不会顺便删除感叹号；要删除指定标点，需要明确给出字符集合。

startswith 返回布尔值，replace 返回字符串，原字符串不会被修改。isdigit 判断数字字符，负号和小数点都不属于数字字符，因此不能用它完整判断一个输入能否转换为数。

Format Specification 格式说明

Read each component of **>12,.2f**.

Expression 表达式

```
value = 1234.5
print(repr(f"{value:>12,.2f}"))
```

Result · repr()

```
'    1,234.50'
```

Part 组成	Meaning 含义	Effect 效果
>	Right alignment 右对齐	Padding goes before the value
12	Minimum width 最小宽度	A longer result is not truncated
,	Grouping 千分位	1234 becomes 1,234
.2f	Fixed point with two decimals	Two digits after decimal

把 >12,.2f 分成四部分逐项读。大于号要求右对齐，12 是最小宽度，逗号加千分位，.2f 使用两位小数的定点表示。repr 让前面的空格可见。

宽度不是截断长度，结果太长时会超过这个宽度。格式参数组合还受值类型限制，完整语法查官方文档。