一、字符串定义和处理

1. 字符串的基本概念

- **定义**：字符串是由单个或多个字符组成的有序序列，用于表示文本数据。
- **创建方式**：用单引号 `'`、双引号 `"` 或三引号 `'''` / `"""` 包裹。

> 案例 1：创建字符串

```
# 单引号字符串
str1 = 'Hello World'

# 双引号字符串（与单引号功能相同）
str2 = "Python 编程"

# 三引号字符串（支持多行文本）
str3 = '''第一行文本
第二行文本
第三行文本'''

print(str1)
print(str2)
print(str3)
```

- 单引号与双引号可嵌套使用（如 `'He said "Hi"'`）。
- 三引号适合定义多行字符串（如文档注释、HTML 代码等）。

2. 字符串的索引与切片

> 字符串是有序序列，每个字符有唯一位置编号（索引），可通过索引访问或截取部分字符（切片）。

> 索引
>
> - 正向索引：从左到右，起始值为 `0`。
> - 反向索引：从右到左，起始值为 `-1`。

> 案例 2：访问字符串中的字符

```
s = "Python"
print(s[0])   # 正向索引：第1个字符，输出 'P'
print(s[2])   # 正向索引：第3个字符，输出 't'
print(s[-1])  # 反向索引：最后1个字符，输出 'n'
print(s[-3])  # 反向索引：倒数第3个字符，输出 'h'
```

> 切片语法：`字符串[start:end:step]`
>
> - `start`：起始索引（包含，默认 0）。
> - `end`：结束索引（不包含，默认字符串长度）。

- `step`：步长（间隔，默认 1，可为负数）。

案例 3：字符串切片操作

```
s = "abcdefgh"

# 截取从索引2到5的字符（不包含5）
print(s[2:5])   # 输出 'cde'

# 从开头截取到索引4
print(s[:4])    # 输出 'abcd'

# 从索引5截取到结尾
print(s[5:])    # 输出 'fgh'

# 步长为2（间隔1个字符）
print(s[::2])   # 输出 'aceg'

# 反向切片（步长为-1，倒序）
print(s[::-1])  # 输出 'hgfedcba'
```

3. 字符串的常用操作

（1）字符串拼接与重复

- 拼接：用 `+` 连接多个字符串。
- 重复：用 `*` 重复字符串多次。

案例 4：拼接与重复

```
s1 = "Hello"
s2 = "World"

# 拼接
print(s1 + " " + s2)  # 输出 'Hello World'

# 重复
print("-" * 10)       # 输出 '----------'（10个连字符）
print(s1 * 3)         # 输出 'HelloHelloHello'
```

（2）字符串长度

用 `len()` 函数获取字符串中字符的个数（包含空格和符号）。

案例 5：计算长度

```
s = "Python 编程"
print(len(s))  # （'Python'6个字符 + 空格1个 + '编程'2个 = 9 注意：中文每个字符算1个，输出9）
```

4. 字符串常用方法

字符串方法是内置函数，用于处理字符串（格式：`字符串.方法名()`）。

(1) 大小写转换

- `lower()`：全部转为小写。
- `upper()`：全部转为大写。
- `title()`：每个单词首字母大写。

案例 6：大小写处理

```
s = "Hello Python WORLD"
print(s.lower())   # 输出 'hello python world'
print(s.upper())   # 输出 'HELLO PYTHON WORLD'
print(s.title())   # 输出 'Hello Python World'
```

(2) 查找与替换

- `find(sub)`：查找子串 `sub` 的起始索引，找不到返回 `-1`。
- `replace(old, new)`：将所有 `old` 子串替换为 `new`。

案例 7：查找与替换

```
s = "I like Python, Python is fun"

# 查找
print(s.find("Python"))  # 输出 7（第一个'Python'的起始索引）
print(s.find("Java"))    # 输出 -1（未找到）

# 替换
print(s.replace("Python", "Java"))  # 输出 'I like Java, Java is fun'
```

(3) 去除空白

- `strip()`：去除字符串两端的空白（空格、换行符 `\n`、制表符 `\t` 等）。
- `lstrip()` / `rstrip()`：仅去除左 / 右侧空白。

案例 8：去除空白

```
s = "  \tHello World\n  "
print(s.strip())   # 输出 'Hello World'（去除两端空白）
print(s.lstrip())  # 输出 'Hello World\n  '（仅去除左侧空白）
```

(4) 分割与连接

- `split(sep)`：按分隔符 `sep` 分割字符串，返回列表。
- `join(iterable)`：用字符串连接可迭代对象（如列表）中的元素。

案例 9：分割与连接

```python
# 分割
s = "apple,banana,orange"
fruits = s.split(",")  # 按逗号分割
print(fruits)  # 输出 ['apple', 'banana', 'orange']

# 连接
new_str = "-".join(fruits)  # 用连字符连接列表元素
print(new_str)  # 输出 'apple-banana-orange'
```

5. 字符串不可变性

> 字符串创建后，其内容不可修改（不能直接通过索引修改字符），但可通过重新赋值或切片生成新字符串。

> 案例 10：不可变性演示

```python
s = "Python"
# s[0] = "p"  # 错误！字符串不可修改

# 正确方式：生成新字符串
s_new = "p" + s[1:]
print(s_new)  # 输出 'python'
```

6. 转义字符

> 用于表示特殊字符（如换行、引号），以反斜杠 `\` 开头。

转义字符	含义
`\n`	换行
`\t`	制表符（Tab）
`\'`	单引号
`\"`	双引号
`\\`	反斜杠

> 案例 11：转义字符使用

```python
print("He said \"I'm fine\"")  # 输出 He said "I'm fine"
print("第一行\n第二行")         # 输出：第一行（换行）第二行
print("姓名\t年龄")             # 输出：姓名    年龄（Tab分隔）
```

7. 综合案例

> 案例 12：用户输入处理
>
> 需求：接收用户输入的姓名（可能包含前后空格），将首字母大写，其余小写，并输出欢迎信息。

```python
name = input("请输入姓名：").strip()  # 去除两端空格
name_formatted = name.title()          # 格式化姓名
print(f"欢迎您，{name_formatted}！")

# 示例：输入 '  alice smith  ' → 输出 '欢迎您，Alice Smith！'
```

> 案例 13：邮箱验证（简化版）
>
> 需求：判断输入的邮箱是否包含 `@` 和 `.`，且 `@` 在 `.` 之前。

```python
email = input("请输入邮箱：")
if "@" in email and "." in email:
    at_index = email.find("@")
    dot_index = email.find(".")
    if at_index < dot_index:
        print("邮箱格式有效")
    else:
        print("邮箱格式无效（@应在.之前）")
else:
    print("邮箱格式无效（缺少@或.）")
```

8. 课堂练习

1. 编写程序，输入一个字符串，输出其反转后的结果（如输入 "abc" 输出 "cba"）。
2. 统计字符串中大写字母、小写字母和数字的个数（如 "Abc123" → 大写 1，小写 2，数字 3）。
3. 输入一个手机号（11 位数字），用字符串切片将中间 4 位替换为 `*`（如 "13800138000" → "138 8000"）。

9. 常见错误与注意事项

1. **索引越界**：访问超出字符串长度的索引（如 `s = "abc"`，`s[3]` 会报错）。
2. **混淆 `find()` 和 `index()`**：`find()` 找不到子串返回 `-1`，`index()` 会直接报错。
3. **忘记字符串不可变性**：试图直接修改字符（如 `s[0] = 'A'`）会报错，需生成新字符串。
4. **转义字符使用错误**：在不需要转义的场景多写 `\`（如路径中的 `\` 需用 `\\` 或前缀 `r` 表示原始字符串，如 `r"C:\test"`）。

二、列表和集合

1. 列表（list）

> 列表是有序、可变的元素集合，允许存储重复值，元素类型可混合（整数、字符串、列表等）。
>
> （1）列表的创建
>
> 用方括号 `[]` 或 `list()` 函数创建，元素之间用逗号分隔。

```python
# 基本创建
nums = [1, 2, 3, 4]
fruits = ["苹果", "香蕉", "橙子"]
mixed = [1, "hello", 3.14, True]  # 混合类型

# 空列表
empty_list = []
empty_list2 = list()

print(fruits)  # 输出：['苹果', '香蕉', '橙子']
```

（2）列表的访问与切片

- **访问**：通过索引（正向从 0 开始，反向从 -1 开始）访问元素。
- **切片**：`列表[start:end:step]`，规则同字符串切片（左闭右开）。

```python
fruits = ["苹果", "香蕉", "橙子", "葡萄"]

# 访问单个元素
print(fruits[0])   # 正向索引：第1个元素 → '苹果'
print(fruits[-1])  # 反向索引：最后1个元素 → '葡萄'

# 切片操作
print(fruits[1:3])   # 索引1到2（不包含3）→ ['香蕉', '橙子']
print(fruits[:2])    # 从开头到索引1 → ['苹果', '香蕉']
print(fruits[2:])    # 从索引2到结尾 → ['橙子', '葡萄']
print(fruits[::2])   # 步长为2 → ['苹果', '橙子']
```

（3）列表的修改（可变特性）

列表是可变的，可通过索引直接修改元素，或通过方法添加 / 删除元素。

```python
# 修改元素
fruits = ["苹果", "香蕉", "橙子"]
fruits[1] = "西瓜"  # 将索引1的元素改为'西瓜'
print(fruits)  # 输出：['苹果', '西瓜', '橙子']

# 添加元素
fruits.append("葡萄")  # 末尾添加
print(fruits)  # 输出：['苹果', '西瓜', '橙子', '葡萄']

fruits.insert(1, "草莓")  # 在索引1处插入
print(fruits)  # 输出：['苹果', '草莓', '西瓜', '橙子', '葡萄']

# 删除元素
del fruits[2]  # 删除索引2的元素
print(fruits)  # 输出：['苹果', '草莓', '橙子', '葡萄']

fruits.remove("橙子")  # 删除指定值（只删第一个）
print(fruits)  # 输出：['苹果', '草莓', '葡萄']

last = fruits.pop()  # 删除末尾元素并返回
print(last, fruits)  # 输出：葡萄 ['苹果', '草莓']
```

（4）列表的常用方法

方法	描述
`len(list)`	计算列表长度
`list.count(x)`	统计元素 x 在列表中出现的次数
`list.index(x)`	查找元素 x 的第一个索引（不存在则报错）
`list.sort()`	对列表排序（默认升序，原地修改）
`list.reverse()`	反转列表（原地修改）

```
nums = [3, 1, 4, 1, 5, 9]

print(len(nums))        # 长度 → 6
print(nums.count(1))    # 1出现的次数 → 2
print(nums.index(5))    # 5的索引 → 4

nums.sort()             # 排序
print(nums)             # 输出：[1, 1, 3, 4, 5, 9]

nums.reverse()          # 反转
print(nums)             # 输出：[9, 5, 4, 3, 1, 1]
```

> （5）列表嵌套
>
> 列表中的元素可以是另一个列表，形成多维列表。

```
# 二维列表（矩阵）
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# 访问嵌套列表元素
print(matrix[1][2])  # 第2行第3列 → 6

# 遍历二维列表
for row in matrix:
    for num in row:
        print(num, end=" ")
    print()
# 输出：
# 1 2 3
# 4 5 6
# 7 8 9
```

> 补充：列表推导式

> 基本语法：
>
> - 变量名 = [表达式 for 变量 in 列表]
> - 变量名 = [表达式 for 变量 in 列表 if 条件]

> 案例：创建一个0-9的列表

```
# 使用for循环实现
list1 = []
for i in range(0, 10):
    list1.append(i)
print(list1)

# 使用列表推导式实现
list2 = [ i for i in range(10) ]
print(list2)
```

> 列表推导式 + if条件判断

> 案例2：生成0-9之间的偶数（i%2 == 0）序列

```
list = [i for i in range(10) if i % 2 == 0]
print(list)
```

> **练习**：使用列表推导式生成平方数集合

2. 集合（set）

> 集合是无序、可变的元素集合，**不允许重复值**，适合去重和集合运算。

> （1）集合的创建
>
> 用花括号 `{}` 或 `set()` 函数创建，空集合必须用 `set()` （`{}` 表示空字典）。

```
# 基本创建
nums = {1, 2, 3, 4}
fruits = {"苹果", "香蕉", "橙子"}

# 去重特性（自动保留唯一值）
duplicates = {1, 2, 2, 3, 3, 3}
print(duplicates)  # 输出：{1, 2, 3}

# 从列表创建集合（去重）
list_with_dups = [1, 2, 2, 3]
set_from_list = set(list_with_dups)
print(set_from_list)  # 输出：{1, 2, 3}

# 空集合
empty_set = set()
```

> （2）集合的基本操作
>
> - **添加元素**：`add()`（单个元素）、`update()`（多个元素，接受可迭代对象）。
> - **删除元素**：`remove()`（删除指定元素，不存在则报错）、`discard()`（删除指定元素，不存在不报错）、`pop()`（随机删除一个元素）。

```python
s = {1, 2, 3}

# 添加
s.add(4)
print(s)  # 输出：{1, 2, 3, 4}

s.update([5, 6])  # 添加多个元素
print(s)  # 输出：{1, 2, 3, 4, 5, 6}

# 删除
s.remove(3)
print(s)  # 输出：{1, 2, 4, 5, 6}

s.discard(10)  # 删除不存在的元素，无报错
print(s)  # 输出：{1, 2, 4, 5, 6}

s.pop()  # 随机删除（结果可能因环境而异）
print(s)  # 例如：{2, 4, 5, 6}
```

3. 列表与集合的对比及转换

特性	列表（list）	集合（set）
有序性	有序（可通过索引访问）	无序（不可通过索引访问）
重复性	允许重复元素	不允许重复元素（自动去重）
索引	支持索引和切片	不支持索引和切片
适用场景	需要保留顺序、允许重复值时	需要去重、进行集合运算时

```python
# 列表转集合（去重）
my_list = [1, 2, 2, 3]
my_set = set(my_list)
print(my_set)  # 输出：{1, 2, 3}

# 集合转列表（恢复为序列，但顺序不保证）
new_list = list(my_set)
print(new_list)  # 可能输出：[1, 2, 3]（顺序不确定）
```

4. 综合案例

> 案例 1：学生选课去重
>
> 需求：统计两个班级选了相同课程的学生（交集），以及所有选课学生的总名单（去重）。

```python
# 两个班级的选课学生（可能有重复）
class1 = ["张三", "李四", "王五", "赵六"]
class2 = ["王五", "赵六", "孙七", "周八"]

# 转为集合
set1 = set(class1)
set2 = set(class2)

# 交集（共同选课的学生）
```

```
common = set1 & set2
print("两个班级都选的学生：", common)  # 输出：{'王五', '赵六'}

# 并集（所有选课学生，去重）
all_students = set1 | set2
print("所有选课学生：", all_students)  # 输出：{'张三', '李四', '王五', '赵六', '孙七', '周八'}
```

> 案例 2：列表数据清洗
>
> 需求：将列表中的空字符串和重复元素去除，并按字母顺序排序。

```
dirty_list = ["apple", "", "banana", "apple", "orange", "", "banana"]

# 去除空字符串
clean_list = [x for x in dirty_list if x != ""]
print(clean_list)  # 输出：['apple', 'banana', 'apple', 'orange', 'banana']

# 去重（转为集合再转回列表）
unique_list = list(set(clean_list))
print(unique_list)  # 输出：['apple', 'banana', 'orange']（顺序可能不同）

# 排序
unique_list.sort()
print(unique_list)  # 输出：['apple', 'banana', 'orange']
```

5. 课堂练习

1. 编写程序，输入 5 个整数存入列表，计算列表中所有偶数的和。
2. 用集合找出两个列表中的不同元素（如列表 A=[1,2,3]，列表 B=[2,3,4]，结果为 [1,4]）。
3. 定义一个嵌套列表 `scores = [[90, 85], [88, 92], [76, 80]]`，计算每个子列表的平均分并打印。

6. 常见错误与注意事项

1. **集合无序性导致的问题**：不能依赖集合的“顺序”，如需有序去重，可先转为集合去重，再转回列表排序。
2. **集合元素类型限制**：集合元素必须是不可变类型（如整数、字符串、元组），不能包含列表（列表是可变的）。
3. **列表索引越界**：访问超出列表长度的索引会报错（如 `list = [1,2]`，`list[2]` 报错）。
4. **混淆 `remove()` 和 `del`**：列表的 `remove(x)` 按值删除，`del list[i]` 按索引删除；集合没有 `del`，只能用 `remove()` 或 `discard()`。

三、字典和元组

1. 字典（dict）

> 字典是**无序**（Python 3.7+ 后为有序）的**键值对（key-value）**集合，通过键快速访问值，适合存储具有映射关系的数据（如用户信息、配置参数等）。

> （1）字典的创建

用花括号 `{}` 或 `dict()` 函数创建，键值对用 `key: value` 表示，多个键值对用逗号分隔。

```
# 基本创建
person = {
    "name": "张三",
    "age": 20,
    "gender": "男"
}

# 空字典
empty_dict = {}
empty_dict2 = dict()

# 用 dict() 函数创建（键为字符串时可省略引号）
person2 = dict(name="李四", age=22, gender="女")

print(person)  # 输出：{'name': '张三', 'age': 20, 'gender': '男'}
```

(2) 字典的键（key）特性

- 键必须是**不可变类型**（如整数、字符串、元组），不能是列表（可变类型）。
- 键具有**唯一性**，重复的键会被最后一个值覆盖。

```
# 合法的键（不可变类型）
valid_dict = {
    1: "整数键",
    "name": "字符串键",
    (1, 2): "元组键"
}

# 不合法的键（列表是可变类型，会报错）
# invalid_dict = {[1, 2]: "列表键"}  # 报错：TypeError

# 重复的键（后者覆盖前者）
duplicate_keys = {"a": 1, "a": 2}
print(duplicate_keys)  # 输出：{'a': 2}
```

(3) 字典的访问与修改

- **访问值**：通过 `字典名[key]` 或 `get()` 方法（键不存在时 `get()` 返回 `None` 或指定默认值）。
- **修改值**：通过 `字典名[key] = 新值` 直接修改。
- **添加键值对**：为不存在的键赋值即可。
- **删除键值对**：`del 字典名[key]` 或 `pop(key)`（删除并返回值）。

```
person = {"name": "张三", "age": 20}

# 访问值
print(person["name"])  # 输出：张三
print(person.get("age"))  # 输出：20
print(person.get("height", "未知"))  # 键不存在，返回默认值：未知
```

```python
# 修改值
person["age"] = 21
print(person)  # 输出：{'name': '张三', 'age': 21}

# 添加键值对
person["height"] = 175
print(person)  # 输出：{'name': '张三', 'age': 21, 'height': 175}

# 删除键值对
del person["height"]
print(person)  # 输出：{'name': '张三', 'age': 21}

age = person.pop("age")  # 删除并返回值
print(age, person)  # 输出：21 {'name': '张三'}
```

> （4）字典的常用方法

方法	描述
`keys()`	返回所有键组成的视图
`values()`	返回所有值组成的视图
`items()`	返回所有键值对组成的视图（元组形式）
`update(dict2)`	合并字典（用 dict2 的键值对更新当前字典）
`clear()`	清空字典所有键值对

```python
person = {"name": "张三", "age": 20, "gender": "男"}

# 获取所有键
print(list(person.keys()))  # 输出：['name', 'age', 'gender']

# 获取所有值
print(list(person.values()))  # 输出：['张三', 20, '男']

# 获取所有键值对
print(list(person.items()))  # 输出：[('name', '张三'), ('age', 20), ('gender', '男')]

# 合并字典
person.update({"height": 175, "age": 21})  # 更新age，添加height
print(person)  # 输出：{'name': '张三', 'age': 21, 'gender': '男', 'height': 175}

# 遍历字典
for key, value in person.items():
    print(f"{key}: {value}")
# 输出：
# name: 张三
# age: 21
# gender: 男
# height: 175
```

2. 元组（tuple）

元组是**有序、不可变**的元素集合，允许重复值，元素类型可混合，适合存储不可修改的序列数据（如坐标、日期等）。

（1）元组的创建

用圆括号 `()` 或直接逗号分隔元素，空元组用 `()` 或 `tuple()`，单元素元组需在元素后加逗号（避免与表达式混淆）。

```
# 基本创建
point = (3, 4)  # 坐标（x=3, y=4）
dates = ("2023", "10", "01")  # 日期
mixed = (1, "hello", 3.14)  # 混合类型

# 空元组
empty_tuple = ()
empty_tuple2 = tuple()

# 单元素元组（必须加逗号）
single_tuple = (5,)  # 正确
# single_tuple = (5)  # 错误，会被识别为整数5

print(point)  # 输出：(3, 4)
```

（2）元组的访问与切片

元组的访问和切片规则与列表、字符串一致（支持索引和切片），但**不能修改元素**（不可变性）。

```
fruits = ("苹果", "香蕉", "橙子", "葡萄")

# 访问元素
print(fruits[0])   # 输出：苹果
print(fruits[-1])  # 输出：葡萄

# 切片操作
print(fruits[1:3])  # 输出：('香蕉', '橙子')
print(fruits[::2])  # 输出：('苹果', '橙子')

# 不可修改（以下代码会报错）
# fruits[0] = "西瓜"  # 报错：TypeError: 'tuple' object does not support item assignment
```

（3）元组的常用方法

由于元组不可变，方法较少，主要有：

- `count(x)`：统计元素 x 出现的次数。
- `index(x)`：查找元素 x 的第一个索引（不存在则报错）。
- `len(tuple)`：计算元组长度。

```
nums = (1, 2, 3, 2, 4, 2)

print(len(nums))        # 长度 → 6
print(nums.count(2))    # 2出现的次数 → 3
print(nums.index(3))    # 3的索引 → 2
```

> （4）元组的不可变性与应用场景

> **不可变性**：元组创建后元素不能修改、添加或删除，适合存储不希望被篡改的数据（如配置信息、常量等）。
>
> **优势**：比列表更节省内存，可作为字典的键（因不可变），可在多线程环境中安全使用（无需担心被修改）。

```
# 元组作为字典的键（合法）
coords = {
    (3, 4): "点A",
    (5, 6): "点B"
}
print(coords[(3, 4)])  # 输出：点A

# 列表作为字典的键（不合法，会报错）
# invalid_dict = {[3,4]: "点A"}  # 报错：TypeError
```

3. 字典与元组的对比及转换

特性	字典（dict）	元组（tuple）
结构	键值对集合	有序元素序列
可变性	可变（可修改键值对）	不可变（元素不能修改）
访问方式	通过键访问值	通过索引访问元素
适用场景	存储映射关系（如用户信息）	存储不可修改的有序数据（如坐标）

```
# 字典与元组的转换
person_dict = {"name": "张三", "age": 20}

# 字典转元组（转为键值对元组的列表）
items_tuple = tuple(person_dict.items())
print(items_tuple)  # 输出：(('name', '张三'), ('age', 20))

# 元组转字典（需是键值对形式的元组）
tuple_to_dict = dict(items_tuple)
print(tuple_to_dict)  # 输出：{'name': '张三', 'age': 20}
```

4. 综合案例

> 案例 1：学生信息管理
>
> 需求：用字典存储多个学生的信息（姓名、年龄、成绩），实现添加、查询和统计平均分功能。

```python
# 初始化学生字典（键为学号，值为学生信息字典）
students = {
    "001": {"name": "张三", "age": 18, "score": 90},
    "002": {"name": "李四", "age": 19, "score": 85}
}

# 添加学生
students["003"] = {"name": "王五", "age": 18, "score": 92}

# 查询学生信息
print(students.get("002"))  # 输出：{'name': '李四', 'age': 19, 'score': 85}

# 统计平均分
scores = [s["score"] for s in students.values()]
average = sum(scores) / len(scores)
print(f"平均分：{average:.1f}")  # 输出：平均分：89.0
```

> 案例 2：用元组存储固定信息
>
> 需求：用元组存储一周的工作日名称，遍历并打印，尝试修改会报错（体现不可变性）。

```python
workdays = ("周一", "周二", "周三", "周四", "周五")

# 遍历打印
for day in workdays:
    print(day)

# 尝试修改（会报错，体现不可变性）
try:
    workdays[0] = "星期日"
except TypeError as e:
    print(f"错误：{e}")  # 输出：错误：'tuple' object does not support item assignment
```

5. 课堂练习

1. 编写程序，创建一个包含 3 个城市信息的字典（键为城市名，值为包含人口、面积的元组），并打印每个城市的人口密度（人口 / 面积）。
2. 用元组存储 5 个学生的成绩，计算最高分、最低分和平均分。
3. 遍历字典 `{"a": 1, "b": 2, "c": 3}`，将键和值拼接为字符串（如 "a=1"）并存入列表。

6. 常见错误与注意事项

1. **字典键的类型错误**：使用列表作为字典的键会报错，需用不可变类型（如元组）。
2. **单元素元组缺少逗号**：`(5)` 会被识别为整数，单元素元组必须写为 `(5,)`。
3. **修改元组元素**：元组不可变，试图修改元素会报错，需转为列表修改后再转回元组。
4. **字典键不存在**：用 `字典名[key]` 访问不存在的键会报错，推荐用 `get()` 方法并设置默认值。