Python 编程期中复习指南 (序列、选择与循环)

本指南根据你提供的三份东南大学《Python编程》课件（序列结构1、序列结构2、选择与循环结构）汇编而成，旨在帮助你高效备考期中考试。

一、核心概念：可变 vs 不可变 (重中之重!)

这是 Python 序列结构中最核心的概念，贯穿始终。

1. 不可变数据类型 (Immutable)

- **包含：** int, float, str (字符串), tuple (元组)
- **特性：** 对象一旦创建，其值（内容）不能被修改。
- **行为：** 任何试图“修改”不可变对象的操作（如 a = a + 1 或 s = s + 'b'），实际上都是创建了一个新对象，并让变量指向这个新地址。
- **优点：** 因为值不变，所以是“可哈希 (hashable)”的，可以作为字典的键 **(key)** 或集合的元素 **(element)**。

2. 可变数据类型 (Mutable)

- **包含：** list (列表), dict (字典), set (集合)
- **特性：** 对象创建后，其值（内容）可以被原地修改。
- **行为：** append(), remove(), sort(), d[key] = val 等操作都是在原始内存地址上修改，不会创建新对象。
- **缺点：** 因为值可变，所以是“不可哈希”的，不能作为字典的键或集合的元素。

二、核心概念：赋值、浅复制与深复制 (高频考点)

理解 Python 的内存管理方式是避免陷阱的关键。

1. 赋值 (=)

- **行为：** 变量赋值（如 list_2 = list_1）不是复制。
- **本质：** 只是创建了一个新的“引用”或“别名”，两个变量指向同一块内存地址。
- **后果：** 修改 list_2 会影响 list_1（因为它们是同一个东西）。
- id(list_1) == id(list_2)

2. 浅复制 (Shallow Copy)

- **实现：** list_1[:] (切片), list_1.copy(), copy.copy(list_1)
- **行为：** 创建一个新的外层容器（新列表，新地址），但容器内的元素是原始元素的引用。
- **后果 (陷阱)：**
  - 如果列表内是不可变类型 (int, str)，修改 list_2 不影响 list_1。

- 如果列表内是可变类型 (如子列表 [1, 2, [3, 4]])，修改 list_2 中的那个子列表（如 list_2[2].append(5)）会影响 list_1。

3. 深复制 (Deep Copy)

- 实现: copy.deepcopy(list_1) (需要 import copy)
- 行为: 创建一个全新的对象，包括所有嵌套的内部元素，全部递归复制一遍。
- 后果: list_2 和 list_1 完全独立，修改 list_2 绝不会影响 list_1。

三、序列结构详解

类型	格式	有序性	可变性	核心特点
列表	list []	有序	可变	功能最全，增删改查方便
元组	tuple ()	有序	不可变	访问快，安全，可作键
字典	dict {k:v}	无序*	可变	键值对，查找极快
集合	set {}	无序	可变	元素不重复，数学运算
字符串	str ""	有序	不可变	字符序列，方法丰富

*注：课件中定义字典为“无序”，这是 *Python 3.7* 以前的标准。在 *3.7+* 版本中，字典已变为有序。请以课件为准。

1. 列表 (List) []

- 增:
  - append(x): 尾部追加单个元素（原地）。
  - extend(L): 尾部追加另一个序列（原地）。
  - insert(i, x): 在索引 i 处插入 x（原地）。
  - +: 连接两个列表，返回一个新列表，效率较低。
- 删:
  - remove(x): 删除首次出现的 x（原地）。
  - pop(i): 删除并返回索引 i 处的元素（默认-1，即最后一个）。
  - del list[i]: 删除索引 i 处的元素。
- 查:
  - list[i]: 按索引访问（支持负索引）。
  - index(x): 查找 x 首次出现的索引，找不到抛出 **ValueError**。
  - count(x): 统计 x 出现的次数。
- 排序与逆序:
  - sort(): 原地排序。
  - sorted(): 内置函数，返回一个新的排好序的列表。
  - reverse(): 原地逆序。
  - reversed(): 内置函数，返回一个迭代器。
- 切片:
  - list[start: stop: step]
  - 切片返回的是一个新列表（浅复制）。
  - list[::-1] 是逆序列表的快捷方式。

2. 元组 (Tuple) ()

- **特性:** 访问速度比列表快，是“写保护”的，可作为字典的键。
- **创建:** 单元素元组必须有逗号：a = (6,)。
- 🔴 **陷阱 (高频考点):** 元组的“不可变”是指其引用不能变，但如果元组内包含可变对象（如列表），那个可变对象仍可被修改。
  - a = (1, 2, [3, 4])
  - a[2].append(5) -> 允许，a 变为 (1, 2, [3, 4, 5])。

3. 字典 (Dictionary) {}

- **键 (Key):** 必须是不可变类型（int, str, tuple）。
- **优点:** 查找、插入、删除速度极快（平均 $O(1)$）。
- **读取:**
  - d[key]: 键不存在时抛出 **KeyError**（不安全）。
  - d.get(key, default=None): 键不存在时返回 default 值（安全）。
- **遍历:**
  - for k in d: (默认遍历键)。
  - d.keys() (返回所有键)。
  - d.values() (返回所有值)。
  - d.items() (返回所有 (键, 值) 元组)。
- **更新:** d1.update(d2) (将 d2 合并入 d1)。

4. 集合 (Set) {}

- **特性:** 元素不重复，自动去重。
- **创建:** 空集合必须用 set()（{} 创建的是空字典）。
- **功能:** 主要用于去重和数学运算。
- **运算:** | (并集), & (交集), - (差集), ^ (对称差集)。

5. 字符串 (String) ""

- **编码 (高频考点):**
  - .encode('codec'): 字符串 (str) -> 字节 (bytes)。
  - .decode('codec'): 字节 (bytes) -> 字符串 (str)。
  - **乱码原因:** 编码和解码时使用了不一致的 codec（如用 'utf-8' 编码，却用 'gbk' 解码 ）。
- **格式化:**
  1. f-string: (Python 3.6+) f'{name} is {age}' (推荐)。
  2. .format(): '{} is {}'.format(name, age)。
  3. %: '%s is %s' % (name, age) (旧式)。
- **常用方法:**
  - find(s): 查找 s，找不到返回 **-1**（安全）。
  - index(s): 查找 s，找不到抛出 **ValueError**（不安全）。
  - split(sep): 按 sep 分割字符串，返回列表。
  - sep.join(list): 用 sep 连接 list 中的所有字符串。
  - strip(), rstrip(), lstrip(): 删除两端、右端或左端的空白或指定字符。

- 缓存机制 **(考点):**
  - Python 会缓存小整数（-5到256）和短字符串。
  - a = 10, b = 10 -> a is b 为 True。
  - a = 257, b = 257 -> a is b 为 False (在某些环境中)。
  - **is** 比较内存地址，**==** 比较内容。

四、选择与循环结构

1. 条件表达式

- 等价于 **False** 的值**:** False, None (空值), 0 (及 0.0, 0j), "" (空字符串), [] (空列表), () (空元组), {} (空字典/集合)。
- 等价于 **True** 的值**:** 任何非空、非零的值（如 [0], 666）。
- 惰性求值 **(短路):**
  - expr1 and expr2: 如果 expr1 为 False，则不计算 expr2。
  - expr1 or expr2: 如果 expr1 为 True，则不计算 expr2。

2. 选择结构

- **if-elif-else:**

```
if score >= 90:
    print('A')
elif score >= 80:
    print('B')
else:
    print('C')
```

- 三元表达式 **(Ternary Operator):**
  - value_if_true if expression else value_if_false
  - b = 5 if a > 9 else 9

3. 循环结构

- **while** 循环**:**
  - 适用于循环次数难以提前确定的情况。
- **for** 循环**:**
  - 适用于枚举序列或迭代对象。
  - 原理**:** for 循环是基于迭代器协议 (调用 iter() 和 next())，不使用索引。
- **break:** 立即终止并跳出整个循环（else 块不执行）。
- **continue:** 立即终止本次迭代，跳到循环顶部，开始下一次迭代。
- 🔴 **for/while** 的 **else** 子句 **(高频考点):**
  - else 块只在循环正常结束（即没有被 break 中断）时执行。

五、推导式与迭代器 (高阶考点)

1. 推导式 (Comprehensions)

- 列表推导式: [x*x for x in range(10)]，返回一个列表。
- 字典推导式: {key: len(key) for key in ['Jack', 'Tom']}。
- 集合推导式: {x**2 for x in (1,1,2,3) if x % 2 == 0}。

2. 生成器表达式 (Generator Expression)

- 语法: (x*x for x in range(10)) (使用圆括号)。
- 重点: 它返回的不是元组，而是一个生成器 (Generator)。
- 特性 (迭代器):
  1. 惰性 (Lazy): 不会立即计算，只在被迭代时 (如 for 循环, next(), sum()) 才计算。
  2. 一次性 (Exhaustible): 只能被完整遍历一次。遍历后即耗尽。

3. 迭代器 (Iterator)

- for 循环的底层实现。
- 🔴 陷阱：迭代器耗尽 (高频考点):
  - squares = (n**2 for n in [1,2,3])
  - sum(squares) -> 14 (第一次调用，迭代器被消耗)。
  - sum(squares) -> 0 (第二次调用时，迭代器已耗尽，返回 0)。
- 🔴 陷阱：迭代器部分耗尽: in 操作也会消耗迭代器。
  - squares = (n**2 for n in [1,2,3]) # (1, 4, 9)
  - print(9 in squares) -> True (迭代器前进，消耗了 1, 4, 9)。
  - print(4 in squares) -> False (此时迭代器已耗尽，或已越过 4)。

六、必看高频陷阱总结

1. 遍历时删除列表元素:
   - 错误: for i in list_1: list_1.remove(i)
   - 原因: 索引错位，导致漏删。
   - 正确: 遍历副本：for i in list_1[:]。
2. for 循环的 else 何时执行?
   - 当循环没有被 break 语句中断时执行。
3. 循环变量泄露:
   - for i in range(4): pass
   - print(i) -> 输出 3。i 在循环后依然存在并保留最后的值。
4. Lambda 闭包陷阱:
   - funcs = [lambda: i for i in range(4)]
   - [f() for f in funcs] -> 结果是 [3, 3, 3, 3]。
   - 原因: lambda 函数在被调用时 (执行 f() 时) 才查找 i 的值，此时循环早已结束，i 的值是 3。
5. 元组的 "不可变" 陷阱:
   - t = (1, [2, 3])
   - t[1].append(4) -> 允许。元组本身（引用）没变，但它内部的列表变了。

6. **is vs ==:**
    - is 比较内存地址，== 比较内容。
    - 考点：Python 对小整数（-5 到 256）和短字符串的缓存机制。