

## 青少年人工智能编程水平测试七级试题

### 一、单项选择题（每题 2 分，共 15 题，共 30 分）

1. 在 Python 中，哪种数据类型不适合用于存储有序数据？

- A. 列表
- B. 元组
- C. 集合
- D. 字典

正确答案：C

解析：集合（set）是无序的，不适合存储有序数据。

2. 以下选项中，创建了二维列表的是？

- A. `list2d = [[1, 2], [3, 4]]`
- B. `list2d = [(1, 2), (3, 4)]`
- C. `list2d = {1: [2, 3], 4: [5, 6]}`
- D. `list2d = [[1, 2], (3, 4)]`

正确答案：A

解析：选项 A 正确地展示了如何在 Python 中创建一个二维列表，即列表的列表。

3. 以下哪个 Python 内置函数不是一个高阶函数？

- A. `map`
- B. `filter`
- C. `sorted`
- D. `print`

正确答案：D

解析：`print` 函数不是高阶函数，它不会接受函数作为参数。

4. 关于 lambda 表达式，以下描述错误的是？

- A. lambda 表达式可以有多个参数
- B. lambda 表达式可以包含复杂的表达式
- C. lambda 表达式可以与 `map` 一起使用
- D. lambda 表达式可以有多个返回值

正确答案：D

解析：lambda 表达式只能有一个返回值，且语法简洁，通常用于简单的操作。

5. 以下哪个 Python 表达式正确地使用了 `map` 和 `lambda` 来将列表中的每个元素乘以 2？

- A. `result = list(map(lambda x: x * 2, [1, 2, 3, 4]))`
- B. `result = map(lambda x: x * 2, [1, 2, 3, 4])`
- C. `result = [x * 2 for x in [1, 2, 3, 4]]`
- D. `result = filter(lambda x: x * 2, [1, 2, 3, 4])`

正确答案：A

解析：选项 A 正确使用了 `map` 和 `lambda` 来对列表中的每个元素进行操作。

6. 以下哪种方法不是 Pillow 库中的图像转换函数？

- A. `convert()`
- B. `resize()`
- C. `transform()`
- D. `rotate()`

正确答案：C

解析：`transform()` 方法不是用于图像转换的函数，其他选项均为图像处理中的常用方法。

7. 使用 Pillow 库保存图像的函数是？

- A. `save()`
- B. `store()`
- C. `export()`
- D. `write()`

正确答案：A

解析：`save()` 函数用于将图像保存到文件中。

8. 在 Pandas 中，以下哪个方法不用于修改数据？

- A. `drop()`
- B. `rename()`
- C. `update()`
- D. `select()`

正确答案：D

解析：`select()` 不是 Pandas 中的数据修改方法，其他选项均用于修改 `DataFrame` 中的数据。

9. 在 Pandas 中，以下哪个选项创建了一个包含四个元素的 Series？

- A. `pd.Series([1, 2, 3, 4])`
- B. `pd.DataFrame([1, 2, 3, 4])`
- C. `pd.Series({1: 'a', 2: 'b', 3: 'c'})`
- D. `pd.Series((1, 2, 3))`

正确答案：A

解析：选项 A 正确地创建了一个包含四个元素的 `Series`。

10. 在 Pandas 中，以下哪个选项正确描述了 `DataFrame`？

- A. `DataFrame` 是一个一维数据结构
- B. `DataFrame` 是一个二维数据结构，可以包含不同类型的数据
- C. `DataFrame` 只能包含数值型数据
- D. `DataFrame` 的列必须是同一种数据类型

正确答案：B

解析：`DataFrame` 是一个二维数据结构，可以包含不同类型的数据。

11. 以下哪个 HTTP 方法用于发送不安全的请求（如提交表单数据）？

- A. GET
- B. POST

C. DELETE

D. PUT

正确答案：B

解析：POST 方法用于发送不安全的请求，通常用于提交表单数据。

12. 以下哪种排序算法的时间复杂度为  $O(n^2)$ ？

A. 冒泡排序

B. 堆排序

C. 快速排序

D. 归并排序

正确答案：A

解析：冒泡排序的时间复杂度为  $O(n^2)$ ，是一种比较简单但效率较低的排序算法。

13. 在下列操作中，哪一个的时间复杂度为  $O(1)$ ？

A. 从列表中随机访问一个元素

B. 遍历列表中的所有元素

C. 在无序列表中查找一个元素

D. 对列表进行排序

正确答案：A

解析：从列表中随机访问一个元素的时间复杂度为  $O(1)$ ，即常数时间复杂度。

14. 以下哪种算法不是稳定的排序算法？

A. 快速排序

B. 冒泡排序

C. 插入排序

D. 归并排序

正确答案：A

解析：快速排序不是稳定的排序算法，它无法确保保持相同元素相对顺序。

15. 关于顺序查找算法，以下描述正确的是？

A. 顺序查找算法的时间复杂度是  $O(\log n)$

B. 顺序查找算法只能用于排序列表

C. 顺序查找算法逐一检查每个元素，直到找到目标值

D. 顺序查找算法要求列表中的元素按升序排列

正确答案：C

解析：顺序查找算法逐一检查每个元素，直到找到目标值或遍历完整个列表。

## 二、多项选择题（每题 3 分，共 5 题，共 15 分，多选或少选不得分）

1. 关于 Python 字典的说法中，哪些是正确的？

A. 字典中的键必须是可哈希的

B. 字典中的值可以是任何数据类型

C. 字典是按插入顺序存储键值对的

D. 字典的键可以重复

正确答案：A, B, C

解析：A 正确：字典中的键必须是可哈希的，这意味着键必须是不可变的类型，如字符串、数字或元组。B 正确：字典中的值可以是任何数据类型，既可以是数字，也可以是列表、字典等。C 正确：从 Python 3.7 开始，字典按照插入顺序存储键值对。D 错误：字典中的键必须是唯一的，不能重复。

2. 以下哪些是 Python 集合（set）的特性？

- A. 集合中的元素是无序的
- B. 集合中的元素可以重复
- C. 集合中的元素必须是可哈希的
- D. 集合可以用来进行数学集合操作，如并集和交集

正确答案：A, C, D

解析：A 正确：集合中的元素是无序的，不能通过索引访问。B 错误：集合中的元素不允许重复。C 正确：集合中的元素必须是可哈希的，类似于字典中的键。D 正确：集合支持数学集合操作，如并集、交集、差集等。

3. 观察以下代码，选择正确的描述：

```
numbers = [10, 20, 30, 40, 50]
result = list(map(lambda x: x / 10, filter(lambda x: x > 20, numbers)))
```

- A. 代码段使用了两个高阶函数。
- B. filter 函数用于筛选出大于 20 的元素。
- C. map 函数用于将每个元素乘以 10。
- D. result 列表将包含元素 [3.0, 4.0, 5.0]。

正确答案：A, B, D

解析：A 正确：代码段使用了 map 和 filter 两个高阶函数。B 正确：filter 函数用于筛选出大于 20 的元素。C 错误：map 函数用于将每个元素除以 10，而不是乘以 10。D 正确：result 列表将包含元素 [3.0, 4.0, 5.0]。

4. 观察以下代码，选择正确的描述：

```
import pandas as pd
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Score': [85, 90, 78, 92],
        'Passed': [True, True, False, True]}
df = pd.DataFrame(data)
high_scores = df[df['Score'] > 80]
passed_students = df[df['Passed']]
```

- A. DataFrame 是通过字典创建的，每列对应字典中的一个键。
- B. high\_scores 是一个 DataFrame 对象，包含得分大于 80 的学生信息。
- C. passed\_students 是一个 Series 对象，包含通过的学生信息。
- D. 可以直接通过列名访问 DataFrame 中的列。

正确答案：A, B, D

解析：A 正确：DataFrame 可以通过字典创建，字典的键对应列名，值对应列的数据。B 正

确:high\_scores 是一个 DataFrame, 包含得分大于 80 的学生信息。C 错误:passed\_students 是一个 DataFrame 对象, 而不是 Series。D 正确: 可以直接通过列名 (如 df['Name']) 访问 DataFrame 中的列。

5. 使用 Pandas 库处理数据时, 下列哪些操作是有效的?

- A. 将 DataFrame 按某列排序
- B. 使用布尔索引筛选数据
- C. 将两个 DataFrame 合并为一个
- D. 直接在 DataFrame 中添加新列

正确答案: A, B, C, D

解析: A 正确: 可以使用 sort\_values() 方法按某列排序。B 正确: 可以使用布尔索引 (如 df[df['Score'] > 80]) 筛选数据。C 正确: 可以使用 concat()、merge() 等方法将两个 DataFrame 合并。D 正确: 可以直接添加新列, 如 df['NewColumn'] = some\_data。

### 三、编程题（共 4 题，共 55 分）

注：试题均不允许在输入函数的括号中写入内容，输出函数仅输出题目要求的信息。所有程序运行时间限制为 3000MS，内存限制为 512MByte。

#### 1. （本题共 5 组测试数据，每个测试点 2 分，共 10 分）

某个村庄的农民们决定在秋收季节举行一个丰收庆典，为此他们需要从仓库中挑选出一些天的水果产量作为庆典的展示品。为了让庆典更加丰盛，村民们希望找到一个连续的 S 天时间段，使得这段时间内收获的水果总量最大，并且在不超过限制重量 W 的情况下，尽可能接近这个限制。

#### 输入描述

第 1 行有三个用空格隔开的整数，分别表示 D、S 和 W ( $1 < S < D < 100$ )。

第 2 行有 D 个用空格隔开的整数，依次表示接下来 D 天中每天的水果产量。

第 3 行有一个整数 W，表示水果的总重量限制。

#### 输出描述

一个整数，表示连续 S 天内不超过限制重量 W 的最大水果总量。如果无法找到满足条件的时间段，则输出 0。

#### 样例输入

```
8 4 20
2 5 6 3 8 2 4 7
```

#### 样例输出

```
19
```

示例题解程序：

```
def max_harvest_within_limit(d, s, w, harvests):
    max_sum_within_limit = 0

    for i in range(d - s + 1):
        current_sum = sum(harvests[i:i+s])
        if current_sum <= w:
            max_sum_within_limit = max(max_sum_within_limit, current_sum)

    return max_sum_within_limit

# 读取第一行的 d、s 和 w
d, s, w = map(int, input().split())
# 读取第二行的水果产量
harvests = list(map(int, input().split()))

# 计算并输出结果
result = max_harvest_within_limit(d, s, w, harvests)
print(result)
```

2. （本题共 5 组测试数据，每个测试点 2 分，共 10 分）

你是一名智能家居系统的开发工程师，负责管理一个家庭的能源消耗。该家庭有多个电器设备（如空调、洗衣机、烤箱等），每个设备在不同时间段内开启或关闭，消耗不同的电量。你的任务是根据一天内设备的使用情况，模拟家庭的总电量消耗。

具体要求如下：

1. 每个电器设备都有一个功率（单位为千瓦），表示该设备每小时的耗电量。
2. 每个电器设备都有一个开启时间和关闭时间（按小时计算，范围为 0 到 23），表示该设备在这些时间段内消耗电力。
3. 每个电器设备在开启期间的电量消耗为其功率乘以开启的小时数。

输入描述：

第 1 行包含一个整数  $n$ ，表示电器设备的数量（至少为 1 个）。

接下来的  $n$  行，每行包含三个整数，分别表示设备的功率（单位为千瓦）、开启时间和关闭时间（闭区间）。

输出描述：

输出一个整数，表示家庭的总电量消耗（单位为千瓦时）。

样例输入：

```
3
2 0 5
3 6 9
4 10 15
```

样例输出：

```
48
```



题目分析：

```
def simulate_energy_consumption(devices):  
    # 每小时的功率消耗数组（24 小时）  
    hourly_consumption = [0] * 24  
  
    # 计算每个设备在每个小时的消耗  
    for power, start, end in devices:  
        for hour in range(start, end + 1):  
            hourly_consumption[hour] += power  
  
    # 累计每小时的消耗  
    total_consumption = sum(hourly_consumption)  
  
    # 输出总消耗  
    print(total_consumption)  
  
# 读取输入  
n = int(input())  
devices = [tuple(map(int, input().split())) for _ in range(n)]  
  
# 模拟能量消耗过程  
simulate_energy_consumption(devices)
```

3. (本题共 5 组测试数据, 每个测试点 3 分, 共 15 分)

小明有一串由数字组成的神秘序列, 这个序列的生成规则如下:

序列的第一个元素是一个整数  $x$ 。

序列的第二个元素是  $y$ 。

从第三个元素开始, 每个元素是前两个元素的 $\text{GCD}$ 加上它们的最小公倍数( $\text{LCM}$ )。

例如, 假设  $x = 4$ ,  $y = 6$ , 那么这个序列的前几项为:

1. 第 1 项: 4
2. 第 2 项: 6
3. 第 3 项:  $\text{GCD}(4, 6) + \text{LCM}(4, 6) = 2 + 12 = 14$
4. 第 4 项:  $\text{GCD}(6, 14) + \text{LCM}(6, 14) = 2 + 42 = 44$
5. 第 5 项:  $\text{GCD}(14, 44) + \text{LCM}(14, 44) = 2 + 308 = 310$

小明想知道, 这个序列的第  $n$  项是什么。

输入描述:

三个整数  $x$ ,  $y$  表示序列的前两项, 和一个正整数  $n$  表示序列的第  $n$  项,  $1 \leq n \leq 15$ 。

输出描述:

一个整数, 表示序列的第  $n$  项的值。

样例输入:

4 6 5

样例输出:

310

示例题解程序（基本递归解法）：

# 计算最大公约数（GCD）

```
def gcd(a, b):  
    while b != 0:  
        a, b = b, a % b  
    return a
```

# 计算最小公倍数（LCM）

```
def lcm(a, b):  
    return abs(a * b) // gcd(a, b)
```

# 递归计算序列的第 n 项

```
def sequence(n, x, y):  
    if n == 1:  
        return x  
    if n == 2:  
        return y  
    prev1 = sequence(n - 1, x, y)  
    prev2 = sequence(n - 2, x, y)  
    gcd_value = gcd(prev1, prev2)  
    lcm_value = lcm(prev1, prev2)  
    return gcd_value + lcm_value
```

# 输入序列的前两项 x, y, 以及 n 的值

```
x, y, n = map(int, input().split())
```

# 输出序列的第 n 项

```
print(sequence(n, x, y))
```

4. (本题共 10 组测试数据, 每个测试点 2 分, 共 20 分)

小红有一张长方形的桌布, 她决定将这张桌布剪成若干个矩形块。每次剪裁时, 她可以选择将桌布沿着长边或宽边剪开, 但每个剪开的部分必须是整数大小的矩形。小红想知道, 如果她从一块初始大小为  $n \times m$  的桌布开始, 有多少种不同的剪裁方式可以将桌布完全剪成若干个矩形块。

例如, 假设桌布的大小为  $2 \times 2$ , 她可以按以下方式进行剪裁:

1. 不剪裁, 保留原样 (1 种方式)
  2. 沿着长度剪成两个  $1 \times 2$  的矩形, 不再继续 (1 种方式)
  3. 沿着宽度剪成两个  $2 \times 1$  的矩形, 不再继续 (1 种方式)
  4. 按照 2 的方式剪裁后, 其中一个矩形继续裁剪成 2 个  $1 \times 1$  矩形 (2 种方式)
  5. 按照 2 的方式剪裁后, 两个矩形都继续裁剪成 2 个  $1 \times 1$  矩形 (1 种方式)
  5. 按照 3 的方式剪裁后, 其中一个矩形继续裁剪成 2 个  $1 \times 1$  矩形 (2 种方式)
  6. 按照 3 的方式剪裁后, 两个矩形都继续裁剪成 2 个  $1 \times 1$  矩形 (1 种方式)
- 总共有 9 种不同的剪裁方式。

输入描述:

两个正整数  $n$  和  $m$ , 表示桌布的长度和宽度,  $1 \leq n, m \leq 10$ 。

输出描述:

一个整数, 表示小红可以剪裁桌布的不同方式数量。

样例输入:

2 2

样例输出:

9

示例题解程序（递归解法）：

```
def count_cutting_ways(n, m):
    if n == 1 and m == 1:
        return 1
    total_ways = 1 # 初始化为 1，表示不剪裁保留原样的方式
    # 尝试沿长边剪裁
    for i in range(1, n):
        total_ways += count_cutting_ways(i, m) * count_cutting_ways(n - i, m)
    # 尝试沿宽边剪裁
    for j in range(1, m):
        total_ways += count_cutting_ways(n, j) * count_cutting_ways(n, m - j)
    return total_ways

# 输入桌布的长度和宽度
n, m = map(int, input().split())
# 输出剪裁方式的数量
print(count_cutting_ways(n, m))
```