

青少年人工智能编程水平测试八年级试题 5

一、单项选择题（每题 2 分，共 15 题，共 30 分）

1. 在 Python 中，以下哪项描述了对对象的作用？

- A. 对象用于定义类
- B. 对象是数据和功能的封装
- C. 对象是类的一部分
- D. 对象用于定义方法

正确答案：B

解析：对象是类的实例，是数据和功能的封装。

2. 哪种方法可由类调用并通常用于实现通用功能，而无需访问实例数据？

- A. 实例方法
- B. 静态方法
- C. 类方法
- D. 构造方法

正确答案：B

解析：静态方法可以直接由类调用，并且不访问实例数据。

3. 以下代码定义了一个类，请选出正确描述该类的选项。

```
class Plant:
    def __init__(self, name):
        self.name = name

    def display_name(self):
        print(f'Plant name is {self.name}')
```

- A. 类 Plant 没有任何属性。
- B. display_name 是一个静态方法。
- C. 类 Plant 的 __init__ 方法初始化属性。
- D. 该类不能实例化。

正确答案：C

解析：__init__ 方法用于初始化类的属性，display_name 是实例方法。

4. 在 Python 中，try 块正常执行后会执行哪个代码块？

- A. except
- B. else
- C. raise
- D. catch

正确答案：B

解析：try 块正常执行后执行 else 块。

5. 在 Python 异常处理结构中，finally 块的作用是什么？

- A. 捕获错误
- B. 只在错误发生时执行
- C. 释放资源
- D. 只在没有错误时执行

正确答案: C

解析: `finally` 块通常用于释放资源和执行清理操作, 无论是否发生异常都会执行。

6. 如何使用 NumPy 生成从 20 到 50 的整数数组?

- A. `np.arange(20, 50)`
- B. `np.arange(20, 51)`
- C. `np.linspace(20, 50, 31)`
- D. `np.zeros(30)`

正确答案: B

解析: `np.arange(20, 51)` 生成从 20 到 50 的整数数组。

7. 以下哪种 NumPy 函数用于在指定范围内生成对数间隔的数值?

- A. `arange()`
- B. `linspace()`
- C. `logspace()`
- D. `geomspace()`

正确答案: C

解析: `logspace()` 用于在指定范围内生成对数间隔的数值。

8. 以下哪种操作可以用来计算数组的累积乘积?

- A. `cumsum()`
- B. `cumprod()`
- C. `sum()`
- D. `prod()`

正确答案: B

解析: `cumprod()` 用于计算数组中元素的累积乘积。

9. 在 Pandas 中, 将 DataFrame 数据 df 导出为 1.csv 文件的方法是?

- A. `df.read_csv('1.csv')`
- B. `df.to_csv('1.csv')`
- C. `to_sql(df, '1.csv')`
- D. `read_csv(df, '1.csv')`

正确答案: B

解析: `to_csv()` 用于将 DataFrame 导出为 csv 文件。

10. 在 Pandas 中, 如何根据 DataFrame 的索引降序排列?

- A. `sort_values(ascending=False)`
- B. `sort_index(ascending=False)`
- C. `order_index(ascending=True)`
- D. `rank(ascending=True)`

正确答案：B

解析：sort_index()方法可以根据索引排序，通过设置 ascending=False 实现降序。

11. 如何在Pandas中对数据分组后计算每组的中位数？

- A. df.groupby('column').median()
- B. df.groupby('column').mean()
- C. df.median('column')
- D. df.group_median('column')

正确答案：A

解析：groupby().median()用于对每组数据计算中位数。

12. 在Pandas中，如何筛选DataFrame中‘Score’列的值小于40的行？

- A. df[df['Score'] < 40]
- B. df['Score'] < 40
- C. df.filter('Score < 40')
- D. df.query(Score < 40)

正确答案：A

解析：A选项通过布尔索引筛选出满足条件的行。

13. 动态规划算法适用于以下哪种问题类型？

- A. 局部最优问题
- B. 分步决策问题
- C. 图遍历问题
- D. 线性问题

正确答案：B

解析：动态规划通过分步决策和缓存中间结果解决复杂问题。

14. 以下哪种算法适合解决排序并合并多个已排序数组？

- A. 冒泡排序
- B. 快速排序
- C. 归并排序
- D. 插入排序

正确答案：C

解析：归并排序特别适用于合并多个已排序数组并进行排序。

15. 深度优先搜索（DFS）主要用于解决以下哪类问题？

- A. 按层遍历节点
- B. 最短路径查找
- C. 所有可能路径查找
- D. 数据筛选

正确答案：C

解析：DFS适用于探索所有可能路径，解决路径查找和图遍历问题。

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. 在使用 Pandas 进行数据清理时，以下哪些方法可用于处理缺失值？

- A. fillna()
- B. dropna()
- C. replace()
- D. concat()

答案：A, B, C

解析：fillna()、dropna() 和 replace() 都可以用于处理缺失值，而 concat() 用于合并数据。

2. 在面向对象编程中，关于继承和封装，下列哪些描述是正确的？

- A. 继承可以复用父类的代码
- B. 封装可以保护数据不被外部直接访问
- C. 继承使得所有方法都变成私有的
- D. 封装不影响类的可维护性

答案：A, B

解析：继承用于代码复用，封装保护数据，而继承不会使方法私有，封装提高了代码可维护性。

3. 以下代码展示了 NumPy 数组的多种操作，哪些描述是正确的？

```
import numpy as np
arr = np.array([[1, 2], [3, 4]])
transposed = arr.T
flattened = arr.flatten()
reshaped = arr.reshape(4, 1)
```

- A. transposed 是 arr 的转置
- B. flattened 将数组拉平成一维数组
- C. reshape 操作改变了原数组的结构
- D. reshape 操作可以在不改变数据的情况下调整形状

答案：A, B, D

解析：transposed 是转置，flattened 将数组拉平，reshape 调整形状不改变数据。C 选项描述不准确，reshape 不改变原数组的存储。

4. 以下代码演示了 Python 的异常处理结构，哪些描述是正确的？

```
try:
    value = int('abc')
except ValueError as e:
    print('Value Error:', e)
finally:
    print('This block always executes')
```

- A. 代码捕获了 ValueError
- B. finally 块在无论是否有异常时都会执行
- C. except 块未被执行

D. 捕获异常后，程序停止执行

答案：A, B

解析：代码捕获了 `ValueError`，`finally` 块无论异常是否发生都会执行。`except` 块在捕获异常时才执行。

5. 关于分治算法，下列哪些描述是正确的？

A. 分治算法将问题分成更小的子问题

B. 子问题独立求解，最后合并结果

C. 分治算法适合解决递归问题

D. 分治算法的关键在于每一步都能找到最优解

答案：A, B, C

解析：分治通过递归分解子问题，独立求解并合并结果，适用于递归问题；D 不正确，并非分治算法的特性。

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）

小周是一名志愿者，负责组织社区的物资分发工作。他有若干个不同大小的箱子，每个箱子可以装一定数量的物资。为了提高工作效率，小周希望用最少数量的箱子装下所有物资。请你帮他计算最少需要使用多少个箱子。

输入描述

第一行包含两个整数 m 和 n ，分别表示物资的总数量和箱子的数量。 $(1 \leq m \leq 1000, 1 \leq n \leq 100)$

第二行包含 n 个整数，表示每个箱子的容量。 $(1 \leq \text{容量} \leq 100)$

输出描述

输出最少需要的箱子数量，如果无法装下所有物资，则输出 -1。

样例输入

20 5

5 8 3 6 4

样例输出

3

示例题解程序：

```
def min_boxes(m, boxes):
    # 按箱子的容量从大到小排序
    boxes.sort(reverse=True)

    total_used = 0
    current_capacity = 0

    # 尝试用最少数量的箱子装满所有物资
    for capacity in boxes:
        current_capacity += capacity
        if current_capacity >= m:
            break
        total_used += 1

    # 检查是否装下所有物资
    if current_capacity >= m:
        return total_used
    else:
        return -1

# 读取输入
m, n = map(int, input().split())
boxes = list(map(int, input().split()))

# 输出结果
print(min_boxes(m, boxes))
```

2. （本题共 5 组测试数据，每个测试点 2 分，共 10 分）

小明和朋友们正在组织一个聚会活动，他们计划制作各种颜色的花环来装饰会场。花环是由多个相连的花朵组成，每朵花可以是红色、黄色或蓝色。他们希望每个花环中相邻的两朵花不能是相同的颜色。请帮助小明计算在给定的花朵数量下，一共有多少种不同的花环制作方案。

输入描述

一个正整数 n ，表示花环中的花朵数量， $2 \leq n \leq 15$ 。

输出描述

一个整数，表示所有符合要求的花环制作方案数量。

样例输入

4

样例输出：

18

示例题解程序：

```
from itertools import product
```

```
def count_flower_rings(n):
```

```
    # 三种花的颜色
```

```
    colors = ['R', 'Y', 'B']
```

```
    count = 0
```

```
    # 生成所有可能的颜色组合
```

```
    for ring in product(colors, repeat=n):
```

```
        # 检查相邻花朵和首尾的颜色是否不同
```

```
        if all(ring[i] != ring[i + 1] for i in range(n - 1)) and ring[0] != ring[-1]:
```

```
            count += 1
```

```
    return count
```

```
# 读取输入
```

```
n = int(input())
```

```
# 计算并输出结果
```

```
print(count_flower_rings(n))
```

3. （本题共 5 组测试数据，每个测试点 3 分，共 15 分）

小明是一位生物学家，他正在研究一个神秘的森林生态系统。森林由一个二维网格组成，每个格子可以是树木、灌木、湖泊或草地。小明希望通过考察森林中连通的区域来分析生态系统的稳定性。为了分析连通性，小明需要找到最大的符合以下条件的连通区域：

1. 连通区域只能由树木（T）和灌木（S）组成。
2. 相邻的格子可以是上下左右方向连通。
3. 连通区域中树木和灌木的数量必须接近，即树木数量与灌木数量之差不能超过 1。

请帮助小明计算符合条件的最大连通区域的面积（格子数）。

输入描述

第一行包含两个整数 n 和 m ，分别表示森林的行数和列数， $2 \leq n, m \leq 50$ 。

接下来的 n 行，每行包含 m 个字符，表示森林的格局：

- T：表示树木。
- S：表示灌木。
- W：表示湖泊，不可行走。
- G：表示草地，不参与连通区域。

输出描述

输出一个整数，表示森林中符合条件的最大连通区域面积（格子数）。如果没有符合条件的区域，输出 -1。

样例输入

```
5 5
TTSGW
SSWTG
TSTGG
GWTSW
GGTTS
```

样例输出

```
13
```

示例题解程序：

```
def dfs(grid, x, y):
    global n, m, visited
    # 标记当前格子为已访问
    visited[x][y] = True
    area = 1 # 计数当前区域面积
    trees = 1 if grid[x][y] == 'T' else 0 # 树木计数
    shrubs = 1 if grid[x][y] == 'S' else 0 # 灌木计数

    # 上下左右四个方向
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    # 尝试向四个方向移动
    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        # 检查新位置是否合法且未访问
        if 0 <= nx < n and 0 <= ny < m and not visited[nx][ny]:
            if grid[nx][ny] == 'T' or grid[nx][ny] == 'S':
                sub_area, sub_trees, sub_shrubs = dfs(grid, nx, ny)
                area += sub_area
                trees += sub_trees
                shrubs += sub_shrubs

    return area, trees, shrubs

def find_largest_forest_area(grid):
    global n, m, visited
    n = len(grid)
    m = len(grid[0])
    visited = [[False] * m for _ in range(n)]
    max_area = -1

    # 遍历每个格子，寻找未访问的合格区域
    for i in range(n):
        for j in range(m):
            if (grid[i][j] == 'T' or grid[i][j] == 'S') and not visited[i][j]:
                area, trees, shrubs = dfs(grid, i, j)
                # 检查是否满足树木和灌木数量接近的条件
                if abs(trees - shrubs) <= 1:
                    max_area = max(max_area, area)

    return max_area

# 读取输入
```

```
n, m = map(int, input().split())
grid = [input().strip() for _ in range(n)]
```

```
# 输出结果
```

```
print(find_largest_forest_area(grid))
```

4. (本题共 10 组测试数据, 每个测试点 2 分, 共 20 分)

小红是一名农场主, 她正在规划自己的果园。她的果园由一个二维网格组成, 每个格子中种植着不同数量的果树。小红希望从果园的左上角开始采摘果实, 并走到右下角。她只能向右或向下移动, 每经过一个格子, 她就会采摘格子中的果实数量。

小红希望走出一条采摘最多果实的路径, 请你帮她计算小红最多可以采摘到多少果实。

输入描述

第一行包含两个整数 n 和 m , 分别表示果园的行数和列数, $1 \leq n, m \leq 50$ 。

接下来的 n 行, 每行包含 m 个整数, 表示每个格子中果树的果实数量 ($0 \leq \text{果实数量} \leq 100$)。

输出描述

输出一个整数, 表示小红在果园中可以采摘到的最多果实数量。

样例输入

3 3

1 2 3

4 5 6

7 8 9

样例输出

29

示例题解程序：

```
def max_fruits(grid):
    n = len(grid)
    m = len(grid[0])

    # 创建一个与网格相同大小的 DP 数组，用于记录最大果实数
    dp = [[0] * m for _ in range(n)]

    # 初始化起点的果实数量
    dp[0][0] = grid[0][0]

    # 初始化第一行的果实数量
    for j in range(1, m):
        dp[0][j] = dp[0][j - 1] + grid[0][j]

    # 初始化第一列的果实数量
    for i in range(1, n):
        dp[i][0] = dp[i - 1][0] + grid[i][0]

    # 填充 DP 数组，选择右移或下移的最大果实路径
    for i in range(1, n):
        for j in range(1, m):
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + grid[i][j]

    # 返回到达右下角的最大果实数量
    return dp[-1][-1]

# 读取输入
n, m = map(int, input().split())
grid = [list(map(int, input().split())) for _ in range(n)]

# 输出结果
print(max_fruits(grid))
```