

青少年人工智能编程水平测试八年级试题 1

一、单项选择题（每题 2 分，共 15 题，共 30 分）

1. 在 Python 中，对象指的是什么？

- A. 类的模板
- B. 类的实例
- C. 方法的集合
- D. 数据的封装

正确答案：B

解析：对象是类的实例，类定义了对象的结构和行为。

2. 下列哪种方法不依赖类的实例化而直接被调用？

- A. 实例方法
- B. 类方法
- C. 静态方法
- D. 私有方法

正确答案：C

解析：静态方法可以不依赖于实例而直接被类调用，通常用于工具函数。

3. 以下哪个选项正确描述了下面程序中定义的类？

```
class Animal:
    def __init__(self, species):
        self.species = species

    def speak(self):
        print('This animal speaks.')
```

- A. 类 Animal 没有定义任何方法。
- B. 类 Animal 既有属性，也有方法。
- C. 方法__init__不是必须的。
- D. speak 方法是类方法。

正确答案：B

解析：类 Animal 包含属性 species 和方法 speak；__init__是构造方法，而 speak 是实例方法。

4. 在 Python 中，若 try 块执行时发生异常，下面哪个代码块会被执行？

- A. finally
- B. except
- C. else
- D. 没有代码块会执行

正确答案：B

解析：except 块在 try 块执行时发生异常时执行。

5. 在 Python 的异常处理中，else 块的执行条件是什么？

- A. 发生异常时执行
- B. 没有发生异常时执行
- C. 不论是否发生异常都会执行
- D. 在 except 块之后执行

正确答案：B

解析：else 块仅在 try 块没有发生异常的情况下执行。

6. 以下哪种方法可以用 NumPy 库创建一个全为零的 10 个元素的数组？

- A. np.zeros(9)
- B. np.arange(10)
- C. np.zeros(10)
- D. np.linspace(0, 9, 10)

正确答案：C

解析：np.zeros(10) 生成一个包含 10 个零的数组。

7. 以下哪个 NumPy 函数用于生成对数间隔的数字序列？

- A. linspace()
- B. logspace()
- C. arange()
- D. array()

正确答案：B

解析：logspace() 用于创建在对数间隔内的均匀分布数字序列。

8. 以下哪种操作可以用来计算数组所有元素的累积和？

- A. cumsum()
- B. sum()
- C. prod()
- D. mean()

正确答案：A

解析：cumsum() 计算数组元素的累积和。

9. 使用 Pandas 读取 Excel 文件数据的方法是？

- A. read_csv()
- B. to_csv()
- C. read_excel()
- D. to_excel()

正确答案：C

解析：read_excel() 用于从 Excel 文件中加载数据到 DataFrame。

10. 在 Pandas 中，如果需要根据索引进行排序，以下哪个方法是正确的？

- A. sort()
- B. order()
- C. sort_values()

D. `sort_index()`

正确答案: D

解析: `sort_index()` 用于根据 DataFrame 的索引进行排序。

11. 在 Pandas 中, 以下哪个选项可以用来计算分组后每个组的最大值?

A. `df.groupby('column').max()`

B. `df.groupby('column').min()`

C. `df.max('column')`

D. `df.groupby('column').mean()`

正确答案: A

解析: `groupby().max()` 用于计算每个组的最大值。

12. 使用 Pandas 过滤 DataFrame 中所有 'Score' 列值小于 50 的行, 以下哪种方法正确?

A. `df.query('Score < 50')`

B. `df[df['Score'] < 50]`

C. `df['Score'] < 50`

D. `df.filter('Score < 50')`

正确答案: B

解析: B 选项正确, 通过布尔索引筛选满足条件的行。

13. 贪心算法通常适用于以下哪类问题?

A. 需要全局最优解的复杂问题

B. 可以通过局部最优决策达到全局最优的问题

C. 所有类型的图问题

D. 需要回溯的问题

正确答案: B

解析: 贪心算法通过在每一步选择局部最优解, 希望能导致全局最优解。

14. 以下哪种算法适用于需要分步解决问题并得到最优解的情况?

A. 贪心算法

B. 动态规划

C. 深度优先搜索

D. 分治算法

正确答案: B

解析: 动态规划通过将问题分解为子问题, 逐步求解各子问题的最优解来得到整体最优解。

15. 广度优先搜索 (BFS) 适合解决以下哪类问题?

A. 查找最深层节点

B. 按照层次结构寻找路径

C. 所有路径都必须遍历

D. 排序已知的节点

正确答案: B

解析: BFS 适合按层次结构搜索, 尤其适用于寻找最短路径。

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 进行数据分析时，哪些方法可用于数据合并？

- A. `merge()`
- B. `join()`
- C. `concat()`
- D. `split()`

答案：A, B, C

解析：`merge()`、`join()`、`concat()` 都是用于合并数据的方法，而 `split()` 用于字符串分割。

2. 在面向对象编程中，关于多态性，下列哪些描述是正确的？

- A. 多态允许对象调用相同方法时表现不同的行为
- B. 多态性要求子类必须重写父类的所有方法
- C. 接口实现是多态的常见形式
- D. 多态性可以通过方法重载实现

答案：A, C, D

解析：A、C、D 描述多态的特性，而 B 不正确，多态并不要求子类必须重写父类的所有方法。

3. 关于 NumPy 数组的操作，以下哪些是正确的？

- A. 数组可以进行逐元素的比较
- B. 数组的形状可以通过 `reshape()` 方法修改
- C. 数组只能包含相同数据类型的元素
- D. 数组不支持矩阵运算

答案：A, B, C

解析：NumPy 数组支持逐元素比较、形状修改和单一数据类型元素，但支持矩阵运算。

4. 在 Python 异常处理的结构中，以下哪些组合是合法的？

- A. `try-except-finally`
- B. `try-except-else`
- C. `try-except-except`
- D. `try-else-finally`

答案：A, B

解析：合法的组合包括 `try-except-finally` 和 `try-except-else`，而重复的 `except` 和独立 `else-finally` 是不合法的。

5. 关于贪心算法，下列哪些描述是正确的？

- A. 贪心算法通过局部最优选择试图达到全局最优
- B. 贪心算法适用于所有最优解问题
- C. 贪心算法在某些情况下比动态规划更高效
- D. 贪心算法通常不保留已做选择的状态

答案：A, C, D

解析：A、C、D 正确描述了贪心算法的特性，而 B 不正确，因为贪心算法并不总是能找到全局最优解。

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）

小明是一位园丁，他负责为一条长长的花坛浇水。花坛可以看作是一条直线，被分成了若干个小区间，每个区间的长度是 1 米。由于水资源有限，小明只能使用最少数量的洒水器来覆盖花坛上的所有区间。

洒水器有不同的覆盖范围，一个洒水器可以覆盖其中心点左右的多个区间。给定花坛的长度 n 和 m 个洒水器的位置及其覆盖范围，请帮助小明计算使用最少数量的洒水器是否能覆盖花坛的所有区间。如果可以，输出最少需要使用的洒水器数量；如果不行，输出 -1。

输入描述

第一行包含两个整数 n 和 m ，表示花坛的长度和洒水器的数量。（ $1 \leq n \leq 1000$ ， $1 \leq m \leq 100$ ）

接下来的 m 行，每行包含两个整数 p 和 r ，分别表示第 i 个洒水器的位置和它的覆盖半径。（ $1 \leq p \leq n$ ， $0 \leq r \leq 100$ ）

输出描述

如果能覆盖所有区间，输出使用的最少洒水器数量；否则输出 -1。

样例输入

```
10 3
4 1
8 2
2 3
```

样例输出

```
2
```

示例题解程序：

```
def min_sprinklers(n, m, sprinklers):
    # 根据洒水器的位置和覆盖范围，计算每个洒水器的覆盖区间
    intervals = []
    for p, r in sprinklers:
        left = max(1, p - r) # 起始区间不能小于 1
        right = min(n, p + r) # 结束区间不能大于 n
        intervals.append((left, right))

    # 按照覆盖区间的起始位置排序，如果相同则按结束位置排序
    intervals.sort(key=lambda x: (x[0], x[1]))

    # 使用贪心算法寻找最少数量的洒水器
    count = 0
    max_reach = 0
    i = 0

    # 从 1 开始直到 n
    while max_reach < n:
        # 当前最大能到达的最右位置
        current_max = max_reach
        # 尽可能扩展覆盖范围
        while i < m and intervals[i][0] <= max_reach + 1:
            current_max = max(current_max, intervals[i][1])
            i += 1
        # 如果无法扩展，表示无法完全覆盖
        if current_max == max_reach:
            return -1
        # 更新最大覆盖位置，并增加洒水器使用数量
        max_reach = current_max
        count += 1

    return count

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

# 输出结果
print(min_sprinklers(n, m, sprinklers))
```

2. (本题共 5 组测试数据, 每个测试点 2 分, 共 10 分)

小美喜欢喝果汁, 她准备制作一杯混合果汁。在她的厨房里, 有 n 种不同口味的果汁, 每种果汁的瓶子容量都是有限的, 每种果汁只能倒一次或不倒。小美希望从这些果汁中选出若干种进行混合, 但她有以下要求:

1. 至少选 2 种不同的果汁进行混合。
2. 如果混合的果汁中包含编号为 5 的倍数的果汁, 则必须至少包含 3 种果汁。

请帮小美计算出有多少种符合要求的果汁混合方案。

输入描述

第一行包含一个正整数 n , 表示果汁的种类数量, $2 \leq n \leq 10$ 。

第二行包含 n 个整数, 分别表示每种果汁的编号。

输出描述

一个整数, 表示所有符合要求的果汁混合方案数量。如果没有符合要求的方案, 则输出 -1。

样例输入

4

1 5 6 7

样例输出

8

示例题解程序：

```
from itertools import combinations

def count_valid_combinations(n, juices):
    count = 0

    # 生成所有可能的组合，长度从 2 到 n
    for i in range(2, n + 1):
        for combo in combinations(juices, i):
            # 检查是否符合组合规则
            if any(x % 5 == 0 for x in combo):
                # 如果包含 5 的倍数，则必须至少选择 3 种
                if len(combo) >= 3:
                    count += 1
            else:
                count += 1

    # 如果没有符合要求的组合方案，输出 -1
    return count if count > 0 else -1

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

# 计算并输出结果
print(count_valid_combinations(n, juices))
```


3. （本题共 5 组测试数据，每个测试点 3 分，共 15 分）

小李是一名城市规划师，他正在设计一个新城市的电网布局。城市中有 n 个重要的电力站点，这些站点之间有一些已经建好的电缆线路。每个站点可以通过一条或多条电缆连接到其他站点，但由于成本原因，小李希望减少电缆的使用次数。小李的目标是找到一种方法，使得所有的站点可以互相连接，并且在选定的电缆中没有形成循环路径，以确保电力线路的稳定和安全。

每条电缆连接两个站点，并有一定的维护成本。小李需要选取一些电缆，使得所有站点都能够连通且不会形成环路，并且总的维护成本最低。请帮助小李找出这条最低成本的电网布局，并输出总的最小维护成本。如果无法使所有站点连通，请输出 -1 。

输入描述

第一行包含两个整数 n 和 m ，分别表示城市中站点的数量和已有的电缆数量， $2 \leq n \leq 100$ ， $1 \leq m \leq 200$ 。

接下来的 m 行，每行包含三个整数 u ， v 和 c ，表示站点 u 和站点 v 之间有一条维护成本为 c 的电缆。

输出描述

输出一个整数，表示使所有站点互相连通且总维护成本最低的方案的成本。如果无法连通所有站点，则输出 -1 。

样例输入

```
4 5
1 2 3
2 3 4
3 4 5
1 4 10
1 3 2
```

样例输出

```
10
```

示例题解程序：

```
class UnionFind:
    """并查集实现，辅助判断环路并进行站点连接。"""

    def __init__(self, n):
        self.parent = list(range(n))

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        rootX = self.find(x)
        rootY = self.find(y)
        if rootX != rootY:
            self.parent[rootX] = rootY
            return True
        return False

def min_cost(n, cables):
    # 按照电缆的维护成本从小到大排序
    cables.sort(key=lambda x: x[2])

    # 初始化并查集
    uf = UnionFind(n)
    total_cost = 0
    edges_used = 0

    # 遍历所有电缆
    for u, v, cost in cables:
        # 连接站点，确保不形成环
        if uf.union(u - 1, v - 1):
            total_cost += cost
            edges_used += 1
            # 如果使用的电缆数量达到了 n-1，说明已经成功连接所有站点
            if edges_used == n - 1:
                return total_cost

    # 如果无法连接所有站点，返回 -1
    return -1
```

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

# 计算并输出结果
print(min_cost(n, cables))
```

4. （本题共 10 组测试数据，每个测试点 2 分，共 20 分）

小明是一名快递员，他每天需要在城市中穿梭，将快递包裹送到不同的目标地点。城市可以看作一个二维网格，每个格子代表一个区域，其中有些区域是道路，有些区域是障碍物，还有一些是目标住户的家。小明每天从快递站出发，需要在最短时间内将所有包裹送达所有目标住户。

小明可以上下左右四个方向移动，每次移动一步。小明必须找到一个路线，将包裹送到所有目标住户的家。他必须依次访问每个目标点，并最终返回快递站。

请帮小明计算，从快递站出发，依次访问所有目标住户并返回的最短路径。如果无法访问所有住户，输出 -1。

输入描述

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

接下来的 n 行，每行包含 m 个字符，表示城市的结构：

- S: 快递站的起点位置，唯一。
- T: 目标住户的家，可以有多个。
- .: 可以通行的道路。
- #: 障碍物，不可通行。

输出描述

输出一个整数，表示从快递站出发，依次访问所有目标住户并返回的最短路径步数。如果无法访问所有住户，输出 -1。

样例输入

```
5 5
S..#T
.##.#
..#..
.##..T
T..#.
```

样例输出

```
24
```

示例题解程序：

```
from collections import deque
from itertools import permutations

def bfs(start, target, grid):
    """广度优先搜索，计算从 start 到 target 的最短路径"""
    n, m = len(grid), len(grid[0])
    queue = deque([start])
    visited = [[False] * m for _ in range(n)]
    visited[start[0]][start[1]] = True
    steps = 0

    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    while queue:
        for _ in range(len(queue)):
            x, y = queue.popleft()
            if (x, y) == target:
                return steps

            for dx, dy in directions:
                nx, ny = x + dx, y + dy
                if 0 <= nx < n and 0 <= ny < m and not visited[nx][ny] and
grid[nx][ny] != '#':
                    visited[nx][ny] = True
                    queue.append((nx, ny))

            steps += 1
    return float('inf')

def find_shortest_route(grid):
    n, m = len(grid), len(grid[0])
    start = None
    targets = []

    # 找到起点和所有目标住户的位置
    for i in range(n):
        for j in range(m):
            if grid[i][j] == 'S':
                start = (i, j)
            elif grid[i][j] == 'T':
                targets.append((i, j))
```

```

# 将起点和目标点放入列表，计算所有点之间的最短距离
all_points = [start] + targets
dist = [[0] * len(all_points) for _ in range(len(all_points))]

# 计算每对点之间的最短距离
for i in range(len(all_points)):
    for j in range(i + 1, len(all_points)):
        d = bfs(all_points[i], all_points[j], grid)
        dist[i][j] = dist[j][i] = d

# 遍历所有访问顺序，找到最短的总路径
min_path = float('inf')
for perm in permutations(range(1, len(all_points))):
    total_dist = dist[0][perm[0]] # 从起点到第一个目标点
    for i in range(len(perm) - 1):
        total_dist += dist[perm[i]][perm[i + 1]]
    total_dist += dist[perm[-1]][0] # 返回起点

    min_path = min(min_path, total_dist)

return min_path if min_path != float('inf') else -1

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

# 输出结果
print(find_shortest_route(grid))

```