

青少年人工智能编程水平测试八年级试题 3

一、单项选择题（每题 2 分，共 15 题，共 30 分）

1. 在 Python 中，什么是类的实例化？

- A. 创建类的方法
- B. 创建类的一个对象
- C. 定义一个新类
- D. 删除类的实例

正确答案：B

解析：实例化是根据类创建对象的过程，即类的具体实现。

2. 以下哪种方法需要由对象进行调用？

- A. 实例方法
- B. 静态方法
- C. 类方法
- D. 主要方法

正确答案：A

解析：实例方法需要在实例化对象后再由对象进行调用。

3. 以下代码定义了一个类，请问正确描述该类的选项是哪一个？

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

```
    def get_name(self):
        return self.name
```

- A. 类 Student 没有任何方法。
- B. `__init__` 方法初始化类的属性。
- C. `get_name` 是类方法。
- D. 类 Student 不能被实例化。

正确答案：B

解析：`__init__` 方法用于在类实例化时初始化对象的属性。

4. 在 Python 中，当 try 块未发生异常时，以下哪个块会被执行？

- A. `except`
- B. `else`
- C. `raise`
- D. 都不执行

正确答案：B

解析：`else` 块在 `try` 块成功执行且无异常时执行。

5. 在 Python 异常处理中，`finally` 块的作用通常是什么？

- A. 清理操作
- B. 捕获异常
- C. 处理错误
- D. 只在无异常时执行

正确答案: A

解析: `finally` 块用于执行清理操作, 无论异常是否发生都会执行。

6. 如何用 NumPy 库创建一个包含 1 到 20 的数组?

- A. `np.arange(1, 21)`
- B. `np.arange(1, 20)`
- C. `np.linspace(1, 20)`
- D. `np.zeros(20)`

正确答案: A

解析: `np.arange(1, 21)` 生成一个包含 1 到 20 的整数数组。

7. 以下哪种 NumPy 函数用于生成在给定范围内的均匀间隔的数值?

- A. `arange()`
- B. `linspace()`
- C. `zeros()`
- D. `ones()`

正确答案: B

解析: `linspace()` 生成在指定范围内的均匀间隔的数值。

8. 以下哪种操作可以用于计算数组的均值?

- A. `mean()`
- B. `sum()`
- C. `prod()`
- D. `max()`

正确答案: A

解析: `mean()` 函数用于计算数组中所有元素的平均值。

9. 在 Pandas 中, 读取 HTML 文件数据的方法是?

- A. `read_csv()`
- B. `read_html()`
- C. `to_html()`
- D. `read_excel()`

正确答案: B

解析: `read_html()` 用于从 HTML 文件加载数据到 `DataFrame`。

10. 在 Pandas 中, 以下哪个方法用于根据多列进行排序?

- A. `sort()`
- B. `sort_values(by=['col1', 'col2'])`
- C. `order()`
- D. `rank()`

正确答案：B

解析：sort_values() 方法可以通过指定多个列进行排序。

11. 如何在 Pandas 中对数据分组后计算每个组的计数？

- A. df.groupby('column').count()
- B. df.groupby('column').sum()
- C. df.count('column')
- D. df.group_count('column')

正确答案：A

解析：groupby().count() 用于对每组数据计数。

12. 在 Pandas 中，如何筛选出 DataFrame 中所有 'Age' 列的值小于 18 的行？

- A. df[df['Age'] < 18]
- B. df['Age'] < 18
- C. df.filter('Age < 18')
- D. df.query(Age < 18)

正确答案：A

解析：A 选项正确，通过布尔索引筛选满足条件的行。

13. 动态规划算法适用于以下哪类问题？

- A. 简单决策问题
- B. 复杂分步决策问题
- C. 所有路径遍历问题
- D. 无需优化的问题

正确答案：B

解析：动态规划用于通过分步优化决策来求解复杂问题。

14. 以下哪种算法适用于大规模数据的排序？

- A. 插入排序
- B. 分治排序
- C. 选择排序
- D. 冒泡排序

正确答案：B

解析：分治排序（如快速排序、归并排序）适用于大规模数据的高效排序。

15. 广度优先搜索（BFS）适合解决以下哪类问题？

- A. 深度路径查找
- B. 最短路径查找
- C. 随机路径探索
- D. 有序列表排序

正确答案：B

解析：BFS 适合寻找最短路径和层级搜索问题。

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 进行数据分析时，以下哪些方法可用于数据清理？

- A. dropna()
- B. fillna()
- C. drop_duplicates()
- D. merge()

答案：A, B, C

解析：dropna()、fillna()、drop_duplicates() 都用于数据清理，而 merge() 用于合并数据。

2. 在面向对象编程中，关于继承，下列哪些描述是正确的？

- A. 子类可以扩展父类的功能
- B. 子类不能访问父类的私有方法
- C. 继承使代码更易于维护
- D. 所有子类必须实现父类的所有方法

答案：A, B, C

解析：继承使得子类可以扩展父类功能，访问父类私有方法受限，同时提高代码的可维护性。D 选项不正确，子类不必实现所有父类方法。

3. 以下代码演示了 NumPy 数组的常用操作，哪些描述是正确的？

```
import numpy as np
arr = np.array([5, 10, 15])
arr += 5
print(arr)
```

- A. 原数组被修改
- B. 数组元素被逐一加上 5
- C. 操作创建了一个新数组
- D. 操作对原数组无影响

答案：A, B

解析：代码直接修改了原数组，并将每个元素加上 5，而没有创建新的数组。

4. 以下代码展示了 Python 异常处理的结构，哪些描述是正确的？

```
try:
    x = 1 / 0
except ZeroDivisionError as e:
    print('Error:', e)
finally:
    print('Cleanup')
```

- A. 代码捕获了 ZeroDivisionError
- B. finally 块在无论是否有异常时都会执行
- C. 异常未被捕获
- D. 捕获异常后，程序停止执行

答案：A, B

解析：代码捕获了 `ZeroDivisionError`，`finally` 块在无论是否有异常时都会执行。程序在捕获异常后继续执行 `finally` 块。

5. 关于贪心算法，下列哪些描述是正确的？

- A. 贪心算法通过每一步选择当前最优解
- B. 贪心算法适用于所有的最优化问题
- C. 贪心算法通常不能保证全局最优解
- D. 贪心算法效率高且实现简单

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，贪心算法不适用于所有最优问题。

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）

小华是一名忙碌的程序员，他有很多任务需要完成，每个任务都有一个截止时间和完成该任务的奖励。小华每天只能完成一个任务，而且他希望通过合理的任务安排，尽可能获得最高的奖励。请帮助小华设计出一个最优的任务安排方案，使得他能获得的总奖励最大。

输入描述

第一行包含一个整数 n ，表示任务的数量。（ $1 \leq n \leq 100$ ）

接下来的 n 行，每行包含两个整数 d 和 r ，分别表示第 i 个任务的截止时间和奖励。（ $1 \leq d \leq 100, 1 \leq r \leq 1000$ ）

输出描述

输出小华通过合理安排任务能够获得的最大总奖励。

样例输入

5

2 50

1 10

2 20

1 30

3 40

样例输出

120

示例题解程序：

```
def max_reward(n, tasks):
    # 按照奖励从高到低排序
    tasks.sort(key=lambda x: x[1], reverse=True)

    # 用来记录每天的任务安排情况
    days = [False] * 101 # 最多有 100 天

    total_reward = 0

    # 遍历每个任务，尽量安排在其截止时间之前的最后一天
    for deadline, reward in tasks:
        # 从该任务的截止时间往前找一个可以安排的位置
        for day in range(deadline, 0, -1):
            if not days[day]: # 如果这天还没有安排任务
                days[day] = True
                total_reward += reward
                break

    return total_reward

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

# 输出结果
print(max_reward(n, tasks))
```

2. （本题共 5 组测试数据，每个测试点 2 分，共 10 分）

小明正在组织一个生日派对，他打算用不同颜色的气球来装饰会场。小明有 n 种不同颜色的气球，每种颜色的气球都可以选择使用或不使用，并且最多只能使用一次。他想要确保每个装饰的气球排列都很漂亮，为此，他需要满足以下规则：

1. 气球的排列中不能有相邻的两只气球颜色相同。
2. 整个排列中，至少使用两种不同的颜色。

请帮助小明计算，在所有可能的气球排列中，有多少种符合要求的排列方式。

输入描述

一个正整数 n ，表示气球的颜色种类， $2 \leq n \leq 10$ 。

输出描述

一个整数，表示所有符合要求的排列方式数量。

样例输入

3

样例输出

12

示例题解程序：

```
from itertools import permutations
```

```
def count_valid_arrangements(n):
```

```
    # 创建颜色标记，例如 ['C1', 'C2', ..., 'Cn']
```

```
    colors = [f'C{i + 1}' for i in range(n)]
```

```
    count = 0
```

```
    # 生成所有可能的非空排列
```

```
    for i in range(2, n + 1): # 至少使用两种颜色
```

```
        for perm in permutations(colors, i):
```

```
            # 检查相邻气球颜色是否不同
```

```
            if all(perm[j] != perm[j + 1] for j in range(len(perm) - 1)):
```

```
                count += 1
```

```
    return count
```

```
# 读取输入
```

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

```
# 计算并输出结果
```

```
print(count_valid_arrangements(n))
```

3. （本题共 5 组测试数据，每个测试点 3 分，共 15 分）

小明是一位探险家，他喜欢在各种迷宫中寻找宝藏。今天他来到了一座神秘的迷宫，迷宫由一个二维网格组成，其中有空地可以行走，有障碍物无法通过，还有一个宝藏所在的位置。小明需要从起点出发，尽快找到迷宫中的宝藏。

迷宫中的每个格子有以下几种类型：

- S：小明的起点，迷宫中唯一一个。
- T：宝藏的位置，迷宫中唯一一个。
- .：可以行走的空地。
- #：障碍物，不可通过。

小明只能上下左右移动，请帮小明计算从起点到达宝藏的最短步数。如果无法到达宝藏，输出 -1。

输入描述

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

接下来的 n 行，每行包含 m 个字符，表示迷宫的格局。

输出描述

输出一个整数，表示小明从起点到宝藏的最短步数。如果无法到达宝藏，输出 -1。

样例输入

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

样例输出

```
5
```

示例题解程序：

```
def dfs(maze, x, y, steps):
    global min_steps, n, m, visited
    # 如果到达宝藏位置，更新最小步数
    if maze[x][y] == 'T':
        min_steps = min(min_steps, steps)
        return

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

    # 标记当前格子为已访问
    visited[x][y] = True

    # 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] and maze[nx][ny] !=
'##':
            dfs(maze, nx, ny, steps + 1)

    # 回溯，取消当前格子的访问标记
    visited[x][y] = False

def find_shortest_path(maze):
    global min_steps, n, m, visited
    min_steps = float('inf')
    n = len(maze)
    m = len(maze[0])
    visited = [[False] * m for _ in range(n)]

    # 找到起点的位置
    start_x, start_y = 0, 0
    for i in range(n):
        for j in range(m):
            if maze[i][j] == 'S':
                start_x, start_y = i, j

    # 从起点开始深度优先搜索
    dfs(maze, start_x, start_y, 0)

    # 返回最小步数，如果未找到则返回 -1
    return min_steps if min_steps != float('inf') else -1
```

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

# 输出结果
print(find_shortest_path(maze))
```

4. （本题共 10 组测试数据，每个测试点 2 分，共 20 分）

小美是一名快递员，她负责将快递包裹送到居民楼的各个住户中。小美所在的小区是一个迷宫式的楼房结构，每个格子代表一个房间，其中有些房间之间有门可以通行，而有些房间则无法通过。小美每天从快递中心出发，需要尽快到达目标住户的房间，将包裹送到。

小美只能通过相邻的房间进行移动，每次可以向上下左右四个方向移动一格。请帮小美计算从快递中心到目标住户的最短路径，如果无法到达目标住户，请输出 -1。

输入描述

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

接下来的 n 行，每行包含 m 个字符，表示小区的结构：

- S: 快递中心的起点位置，唯一。
- E: 目标住户的位置，唯一。
- .: 可以通行的房间。
- #: 障碍物，不可通过。

输出描述

输出一个整数，表示从快递中心到目标住户的最短路径步数。如果无法到达目标住户，输出 -1。

样例输入

```
5 5
S..#.
.#.##.
..#..
.#..E
.....
```

样例输出

9

示例题解程序：

```
from collections import deque
```

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

    # 找到起点和终点的位置
    start = None
    end = None
    for i in range(n):
        for j in range(m):
            if grid[i][j] == 'S':
                start = (i, j)
            elif grid[i][j] == 'E':
                end = (i, j)

    # BFS 初始化
    queue = deque([start])
    visited = [[False] * m for _ in range(n)]
    visited[start[0]][start[1]] = True
    steps = 0

    # 四个方向的移动
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    # 开始广度优先搜索
    while queue:
        for _ in range(len(queue)):
            x, y = queue.popleft()

            # 如果到达终点，返回步数
            if (x, y) == end:
                return steps

            # 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] and
grid[nx][ny] != '#':
                    visited[nx][ny] = True
```

```
        queue.append((nx, ny))

    steps += 1

    # 如果无法到达终点, 返回 -1
    return -1


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

# 输出结果
print(bfs_shortest_path(grid))
```