

## 青少年人工智能编程水平测试八年级试题 2

### 一、单项选择题（每题 2 分，共 15 题，共 30 分）

1. 在 Python 中，实例是以下哪种定义的具体实现？

- A. 方法
- B. 类
- C. 模块
- D. 对象

正确答案：B

解析：实例是类的具体实现，类定义了实例的结构和行为。

2. 以下哪种方法不需要实例化对象即可由类调用？

- A. 静态方法
- B. 实例方法
- C. 普通方法
- D. 特殊方法

正确答案：A

解析：静态方法不需要类的实例化，可以直接由类调用。

3. 以下代码定义了一个类，哪个选项正确描述了该类？

```
class Book:
    def __init__(self, title):
        self.title = title

    def show_title(self):
        print(self.title)
```

- A. 类 Book 没有任何属性。
- B. `__init__` 方法初始化类的属性。
- C. `show_title` 是类方法。
- D. 类 Book 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，B 选项正确。

4. 在 Python 中，当 try 块发生异常时，哪个代码块会被执行？

- A. else
- B. except
- C. finally
- D. 都不会执行

正确答案：B

解析：except 块用于处理 try 块中的异常，当发生异常时执行。

5. 在异常处理过程中，finally 块何时会被执行？

- A. 只有在发生异常时
- B. 没有发生异常时
- C. 不论是否发生异常
- D. 在 except 块之后

正确答案: C

解析: finally 块总会执行, 无论 try 块中是否有异常发生。

6. 如何用 NumPy 库创建一个包含 5 到 15 的数组?

- A. np.arange(5, 15)
- B. np.arange(5, 16)
- C. np.linspace(5, 15, 10)
- D. np.zeros(10)

正确答案: B

解析: np.arange(5, 16) 生成从 5 到 15 的整数数组。

7. 以下哪种 NumPy 函数可用于创建特定数量的等间距值?

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

正确答案: A

解析: linspace() 用于在指定范围内生成等间距的数值。

8. 以下哪种方法可以用于计算数组元素的乘积?

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

正确答案: B

解析: prod() 函数用于计算数组中所有元素的乘积。

9. 在 Pandas 中, 读取 JSON 文件数据的方法是?

- A. read\_csv()
- B. to\_json()
- C. read\_json()
- D. to\_csv()

正确答案: C

解析: read\_json() 用于从 JSON 文件加载数据到 DataFrame。

10. 在 Pandas 中, 以下哪个方法用于根据列值降序排序?

- A. sort\_index()
- B. sort\_values(ascending=False)
- C. order()
- D. rank()

正确答案：B

解析：sort\_values() 可以使用 ascending=False 选项来进行降序排序。

11. 如何在 Pandas 中对数据进行分组后计算每个组的总和？

- A. df.groupby('column').sum()
- B. df.groupby('column').max()
- C. df.sum('column')
- D. df.group\_sum('column')

正确答案：A

解析：groupby().sum() 用于对每组数据进行求和操作。

12. 在 Pandas 中，以下哪个选项可筛选出 DataFrame 中 'Salary' 列大于 50000 的行？

- A. df[df['Salary'] > 50000]
- B. df['Salary'] > 50000
- C. df.filter('Salary > 50000')
- D. df.query(Salary > 50000)

正确答案：A

解析：A 选项通过布尔索引筛选满足条件的行。

13. 以下哪种算法适用于需要作出局部最优决策的问题？

- A. 动态规划
- B. 贪心算法
- C. 深度优先搜索
- D. 广度优先搜索

正确答案：B

解析：贪心算法通过每一步选择局部最优希望达到全局最优。

14. 以下哪种算法适用于递归分解问题并解决子问题以获得整体解？

- A. 动态规划
- B. 分治算法
- C. 贪心算法
- D. 图算法

正确答案：B

解析：分治算法将问题递归分解为子问题，再合并子问题的解。

15. 深度优先搜索（DFS）适合于解决以下哪类问题？

- A. 查找最短路径
- B. 在图中找到所有可能路径
- C. 按照层次遍历节点
- D. 排序数组中的元素

正确答案：B

解析：DFS 用于图中探索所有可能的路径，适合路径探索类问题。

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 进行数据操作时，以下哪些方法可用于数据过滤？

- A. query()
- B. loc[]
- C. iloc[]
- D. merge()

答案：A, B, C

解析：query()、loc[]、iloc[] 都可以用于过滤数据，merge() 用于合并数据。

2. 在面向对象编程中，关于封装性，下列哪些描述是正确的？

- A. 封装将数据和方法绑定在一起
- B. 封装可以限制对类内部数据的直接访问
- C. 封装性要求所有成员都必须是私有的
- D. 封装可以提高代码的可维护性

答案：A, B, D

解析：封装将数据和方法绑定并隐藏内部实现，C 选项不正确，封装不要求所有成员都为私有。

3. 以下代码展示了一个 NumPy 数组的基本操作，哪些描述是正确的？

```
import numpy as np
arr = np.array([1, 2, 3, 4])
arr *= 2
print(arr)
```

- A. 代码修改了原数组
- B. 代码生成了一个新的数组
- C. 数组元素被逐一乘以 2
- D. 数组操作不影响原数组

答案：A, C

解析：代码修改了原数组，数组元素逐一被乘以 2，而不是生成新的数组。

4. 在 Python 异常处理结构中，以下哪些语法是合法的？

- A. try-except-as
- B. try-except-finally
- C. try-finally
- D. try-catch

答案：B, C

解析：合法的组合包括 try-except-finally 和 try-finally, try-catch 是其他语言的用法。

5. 关于递归算法，下列哪些描述是正确的？

- A. 递归函数必须有一个终止条件
- B. 递归可以替代循环解决问题
- C. 递归更适合解决大规模问题
- D. 递归总是比迭代更高效

答案：A, B

解析：A 和 B 正确，递归必须有终止条件，并可以替代循环，但 C 和 D 不一定成立，递归的效率取决于问题类型。

### 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）

小丽是一位图书管理员，她负责将新进的书籍摆放到书架上。每本书都有一个固定的厚度，书架的每一层都有一定的容量，无法超出这个容量限制。现在给定书架的容量和每本书的厚度，请帮助小丽计算使用最少的书架层数来放下所有的书。如果无法将所有书放下，请输出 -1。

输入描述

第一行包含两个整数  $c$  和  $n$ ，分别表示书架每一层的容量和书的数量。（ $1 \leq c \leq 1000$ ， $1 \leq n \leq 100$ ）

第二行包含  $n$  个整数，表示每本书的厚度。（ $1 \leq \text{厚度} \leq 100$ ）

输出描述

输出放下所有书所需的最少层数。如果无法放下所有书，输出 -1。

样例输入

10 5  
4 8 3 5 6

样例输出

3

示例题解程序：

```
def min_shelves(c, books):
    # 按书的厚度从大到小排序
    books.sort(reverse=True)

    shelves = []

    # 尝试将每本书放入现有的书架层中
    for book in books:
        placed = False
        for i in range(len(shelves)):
            # 如果当前书可以放入书架的某一层
            if shelves[i] + book <= c:
                shelves[i] += book
                placed = True
                break
        # 如果不能放入现有的任何一层，创建新的层
        if not placed:
            shelves.append(book)

    # 返回所需的最少层数
    return len(shelves)

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

# 输出结果
print(min_shelves(c, books))
```

2. （本题共 5 组测试数据，每个测试点 2 分，共 10 分）

小美是一位烘焙爱好者，她正在为朋友们制作一盘美味的曲奇饼干。她有  $n$  种不同口味的曲奇饼，每种口味的饼干可以选择使用或不使用，并且每种口味的饼干最多只能使用一次。小美希望所有选出的饼干按特定的排列方式放在盘子里，并遵循以下规则：

1. 相同口味的饼干不能放在相邻的位置。
2. 整个排列中必须至少包含 3 块饼干。
3. 如果存在编号为 3 的倍数的饼干，则不能放在最左边。

请帮小美计算，在所有可能的饼干排列中，有多少种符合要求的排列方式。

输入描述

一个正整数  $n$ ，表示饼干的口味种类数量， $3 \leq n \leq 10$ 。

第二行包含  $n$  个整数，分别表示每种口味的编号。

输出描述

一个整数，表示所有符合要求的饼干排列方式数量。如果无法按照要求排列，则输出-1。

样例输入

4

1 2 3 4

样例输出

36



示例题解程序：

```
from itertools import permutations

def count_valid_cookies(n, flavors):
    count = 0

    # 生成所有可能的排列，长度从 3 到 n
    for i in range(3, n + 1):
        for perm in permutations(flavors, i):
            # 检查相邻饼干是否不同
            if all(perm[j] != perm[j + 1] for j in range(len(perm) - 1)):
                # 检查最左边是否为编号为 3 的倍数的饼干
                if perm[0] % 3 != 0:
                    count += 1

    # 如果没有符合要求的排列方案，输出 -1
    return count if count > 0 else -1

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

# 计算并输出结果
print(count_valid_cookies(n, flavors))
```

3. （本题共 5 组测试数据，每个测试点 3 分，共 15 分）

小美是一名探险爱好者，她正在探索一片神秘的地下洞穴。这个洞穴由多个相连的房间组成，每个房间之间有可能存在一条通道。每次小美可以沿着通道从一个房间移动到另一个房间。小美的目标是从起点房间出发，尽可能多地探索不同的房间，但她不能走回头路，因为洞穴非常复杂，迷失方向会非常危险。

现在，请帮小美计算，她从起点房间出发，最多可以探索到多少个不同的房间。

输入描述

第一行包含两个整数  $n$  和  $m$ ，分别表示房间的数量和通道的数量， $2 \leq n \leq 50$ ， $1 \leq m \leq 100$ 。

接下来的  $m$  行，每行包含两个整数  $u$  和  $v$ ，表示房间  $u$  和房间  $v$  之间有一条双向通道。

输出描述

输出一个整数，表示小美从起点房间（房间编号为 1）出发，最多可以探索到的不同房间数量。

样例输入

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

样例输出

```
5
```

示例题解程序：

```
def dfs(graph, visited, node):
    """深度优先搜索探索房间。"""
    visited[node] = True
    count = 1 # 计数当前房间

    # 遍历相邻的房间
    for neighbor in graph[node]:
        if not visited[neighbor]:
            count += dfs(graph, visited, neighbor)

    return count

def max_explored_rooms(n, connections):
    # 构建图的邻接表
    graph = [[] for _ in range(n + 1)]
    for u, v in connections:
        graph[u].append(v)
        graph[v].append(u)

    # 初始化访问数组
    visited = [False] * (n + 1)

    # 从房间 1 开始探索
    return dfs(graph, visited, 1)

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

# 计算并输出结果
print(max_explored_rooms(n, connections))
```

4. (本题共 10 组测试数据, 每个测试点 2 分, 共 20 分)

小明是一位病毒研究员, 他正在研究一片森林中的病毒传播情况。森林可以看作是一个二维网格, 每个格子代表一个区域。病毒会从感染源开始传播, 每次传播一步, 可以传播到相邻的四个方向的格子 (上下左右)。小明希望尽快统计出病毒完全感染整个森林需要的时间。

森林中的每个格子可能是以下几种状态:

- I: 感染源 (病毒传播的起点), 可以有多个。
- T: 树木, 健康状态, 可能被感染。
- .: 空地, 可以传播病毒但不会被感染。
- #: 岩石, 无法感染且阻挡病毒传播。

请帮小明计算, 病毒完全感染整个森林的所有树木需要的最短时间。如果某些树木永远无法被感染, 请输出 -1。

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, 表示森林的结构:

- I: 感染源, 初始感染位置。
- T: 树木, 健康的可感染区域。
- .: 空地, 可以传播病毒。
- #: 岩石, 无法感染的障碍。

输出描述

输出一个整数, 表示病毒完全感染整个森林需要的最短时间 (步数)。如果有无法感染的树木, 输出 -1。

样例输入

```
5 5
I.T..
.#T#.
.TI.T
.T.#T
T..#I
```

样例输出

```
4
```

示例题解程序：

```
from collections import deque
```

```
def bfs_infection(grid):
    n = len(grid)
    m = len(grid[0])
    queue = deque()
    visited = [[False] * m for _ in range(n)]

    # 初始化：将所有感染源加入队列
    for i in range(n):
        for j in range(m):
            if grid[i][j] == 'I':
                queue.append((i, j))
                visited[i][j] = True

    steps = 0
    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()

            # 尝试向四个方向传播
            for dx, dy in directions:
                nx, ny = x + dx, y + dy
                if 0 <= nx < n and 0 <= ny < m and not visited[nx][ny]:
                    # 传播到空地或树木
                    if grid[nx][ny] == '.' or grid[nx][ny] == 'T':
                        visited[nx][ny] = True
                        queue.append((nx, ny))

            # 每完成一层扩展，增加一步
            if queue:
                steps += 1

    # 检查是否还有未感染的树木
    for i in range(n):
        for j in range(m):
            if grid[i][j] == 'T' and not visited[i][j]:
                return -1
```

```
    return steps
```

```
# 读取输入
```

```
n, m = map(int, input().split())
```

```
grid = [input().strip() for _ in range(n)]
```

```
# 输出结果
```

```
print(bfs_infection(grid))
```