Python Blackjack Command-Line Game Development Assignment

I. Basic Assignment Information

- **Course Name**: Fundamentals of Python Programming
- **Assignment Name**: Command-Line Blackjack Game Development
- **Target Audience**: Beginner programming learners
- **Estimated Completion Time**: 2-3 hours

II. Assignment Objectives

1. Proficiency in using conditional statements (`if-elif-else`) to implement real-time state checks and logical branches
2. Mastery of loop structures (`while`) in game flow control, especially turn-switching logic
3. Understanding of random number generation applications in card games to implement deck initialization and dealing logic
4. Cultivation of ability to convert game rules into executable code, enhancing code readability and logical rigor

III. Core Requirements

1. Development Environment and Code Restrictions (Strictly Enforced)

- Development Environment: Python 3.6 or higher
- Only standard libraries are allowed (the `random` module may be used for random functionality)
- **Code Restrictions** (core assessment points):
    - Prohibition of defining any functions (including custom functions and lambda functions)
    - Only basic data types are allowed (lists, strings, integers, booleans, etc.)
    - All game logic must be implemented using **conditional statements** (`if-elif-else`) and **loop structures** (`while`); no other control structures are permitted
    - No use of classes, additional module imports (except `random`), or other advanced syntax

2. Game Flow and Logic (Aided by Flowcharts)

Code logic must be implemented strictly according to the following flowcharts to ensure no omissions in turn switching, real-time checks, and other links.

(1) Player's Turn Flow

```
Start player's turn→Display player's hand→Check if it's a blackjack
  ↓
Each hit→Update hand→Calculate points→Real-time check:
  ├─ Points＞21→Player busts (immediate loss, end player's turn)
  ├─ Points=21→Force end of player's turn (enter dealer's turn)
  └─ Points＜21→Continue asking for hit/stand
  ↓
Player chooses to stand→End player's turn→Enter dealer's turn
```

(2) Dealer's Turn Flow

```
Start dealer's turn→Reveal hidden card→Display full hand→Calculate initial
points→Real-time check:
  ├─ Dealer has blackjack/points=21→Immediate result determination
  ├─ Dealer points＞21→Dealer busts (player wins)
  └─ Dealer points＜17→Enter automatic hitting loop
       ↓
    Each hit→Update hand→Calculate points→Real-time check:
      ├─ Points＞21→Dealer busts (player wins)
      ├─ Points=21→Dealer wins
      └─ Points≥17→Stop hitting, enter final determination
  ↓
Final determination (compare both sides' points)→Display result
```

3. Card Rules and Determination Logic

- **Card Point Calculation**:
  - Numbered cards (2-10) are counted by their face value; J, Q, K all count as 10 points; **Aces can be flexibly counted** (initially as 11 points, automatically converted to 1 point when total exceeds 21).
- **Blackjack Determination**: When the initial 2 cards are **"Ace + 10/J/Q/K"** (total exactly 21 points), it is determined as a "blackjack" (the most efficient winning scenario).
- **Real-time Determination Triggers**:
  - Player's turn: Immediately check "whether bust (＞21)" and "whether exactly 21 points (=21)" after each hit; initial hand must be checked for blackjack.

    - Dealer's turn: Check "whether blackjack", "whether 21 points", and "whether bust" after revealing hidden card; during automatic hitting, check "whether bust", "whether 21 points", and "whether ≥17 points (stop hitting if met)" after each hit.

IV. Input and Output Specifications

1. **Input Handling**:

- Game startup: Prompt `“Please enter a number to select: 1–Start game”`, only accepts input `"1"`.
- Player's turn: Prompt `“Please enter a number to select: 1–Hit, 2–Stand”`, only accepts `"1"` or `"2"`.
- Game end: Prompt `“Please enter a number to select: 1–Restart, 2–End game”`, only accepts `"1"` or `"2"`.
- Invalid input: Prompt `“Invalid input, please enter X or Y”` (X, Y are valid numbers for the current scenario).

2. **Output Requirements**:

- Display welcome interface and rule explanations (including "number selection→operation" reference table) at startup.
- Player's hand must always display **full card faces + real-time points**; dealer's initial hand is displayed as `"[Visible card, *]"` (hidden card remains concealed until player's turn ends).
- Real-time prompts for key states: Blackjack (`“You got a blackjack!”` / `“Dealer got a blackjack!”`), bust (`“You busted!”` / `“Dealer busted!”`), 21 points (`“Your points are exactly 21!”`).
- Clear result prompts: `"You win!"` / `"You lose!"` / `"It's a tie!"`.
- Use separators (e.g., `=`) to divide different stages (e.g., "Start of player's turn", "Start of dealer's turn") to improve readability.

V. Grading Criteria (Total 100 Points)

1. **Compliance with Code Restrictions (20 points)**:

- No functions defined (8 points; 0 points if violated).
- Only basic data types and allowed control structures used (7 points).
- No prohibited advanced syntax used (5 points).

2. **Functional Completeness (40 points)**:

- Correct deck initialization and random dealing logic (5 points).
- Accurate point calculation (especially flexible switching of Ace's "11/1 points") (8 points).
- Correct logic for dealer's hidden card "concealment→reveal" (5 points).
- Complete player's turn flow (including real-time checks: blackjack, bust, 21 points) (7 points).
- Complete dealer's turn flow (including real-time checks: blackjack, bust, 21 points, and "must hit if <17/must stand if ≥17" rules) (8 points).
- No loopholes in number selection operations and loop control (7 points).

3. **Logical Correctness (25 points)**:

- No loopholes in turn-switching logic (smooth transition from player→dealer turn) (10 points).
- Real-time determination covers all scenarios (no omissions or logical contradictions) (10 points).
- No runtime issues like infinite loops or process freezes (5 points).

4. **User Experience (15 points)**:

- Clear input prompts and operation guidance (5 points).
- Intuitive output information and accurate prompts for key states (5 points).
- Neat formatting with separators enhancing readability (5 points).

VI. Submission Requirements

1. Submit a single Python file with the filename format: `studentID_name_blackjack.py`.
2. The code must include comment information at the beginning:

```
# Student ID: XXX
# Name: XXX
# Date: XXXX-XX-XX
# Course Assignment: Blackjack Game Development
```

3. The code must run independently without syntax errors.

VII. Notes

1. **Code restrictions are core assessment points**: Violations will directly deduct corresponding points; serious violations will result in a zero score for the assignment.

2. Plagiarism is prohibited: Any plagiarism found will result in a zero score.
3. Functions and logic must strictly match the flowcharts and rule explanations: Misunderstandings of rules will directly affect the score.

VIII. Extended Tasks (Optional, not included in total score)

1. Implement "multiple decks" functionality (e.g., simulating 6 decks for shuffling and dealing).
2. Add a simple betting system that supports number input for bet amounts.
3. Record player's historical results (win/loss/tie counts, highest points, etc.) and display statistics.

Python 二十一点命令行游戏开发作业

一、作业基本信息

- **课程名称**：Python 程序设计基础
- **作业名称**：命令行二十一点游戏开发
- **适用对象**：编程入门学习者
- **预计完成时间**：2-3小时
-

二、作业目标

1. 熟练运用条件判断（`if-elif-else`）实现实时状态检查与逻辑分支
2. 掌握循环结构（`while`）在游戏流程控制中的应用，尤其是回合切换逻辑
3. 理解随机数生成在卡牌游戏中的应用，实现牌组初始化与发牌逻辑
4. 培养将游戏规则转化为可执行代码的能力，强化代码可读性与逻辑严谨性

三、核心要求

1. 开发环境与代码限制（严格遵守）

- 开发环境：Python 3.6及以上版本
- 仅允许使用标准库（可使用`random`模块实现随机功能）
- **代码限制**（核心考核点）：
  - 禁止定义任何函数（包括自定义函数、lambda函数）

    - 仅允许使用基础数据类型（列表、字符串、整数、布尔值等）
    - 必须通过**条件判断**（`if-elif-else`）和**循环结构**（`while`）实现所有游戏逻辑，不得使用其他控制结构
    - 不得使用类、额外模块导入（除`random`外）及其他高级语法

2. 游戏流程与逻辑（流程图辅助）

需严格按照以下流程图实现代码逻辑，确保回合切换、实时检查等环节无遗漏。

（1）玩家回合流程

```
开始玩家回合→显示玩家手牌→检查是否为黑杰克
   ↓
每要一次牌→更新手牌→计算点数→实时检查：
   ├─ 点数＞21→玩家爆牌（直接判负，结束玩家回合）
   ├─ 点数=21→强制结束玩家回合（进入庄家回合）
   └─ 点数＜21→继续询问要牌/停牌
   ↓
玩家选择停牌→结束玩家回合→进入庄家回合
```

（2）庄家回合流程

```
开始庄家回合→公开暗牌→显示完整手牌→计算初始点数→实时检查：
   ├─ 庄家是黑杰克/点数=21→直接判定胜负
   ├─ 庄家点数＞21→庄家爆牌（玩家胜）
   └─ 庄家点数＜17→进入自动要牌循环
          ↓
     每次要牌→更新手牌→计算点数→实时检查：
        ├─ 点数＞21→庄家爆牌（玩家胜）
        ├─ 点数=21→庄家胜
        └─ 点数≥17→停止要牌，进入最终判定
   ↓
最终判定（对比双方点数）→显示结果
```

3. 牌面规则与判定逻辑

- **牌面点数计算**：
    - 数字牌（2-10）按面值算点数；J、Q、K均算10点；**A可灵活计算**（初始按11点，总点数超21时自动转为1点）。
- **黑杰克判定**：初始2张牌为 **“A + 10/J/Q/K”**（总点数恰好21）时，判定为“黑杰克”（最高效获胜场景）。

- **实时判定触发**：
  - 玩家回合：要牌后立即检查“是否爆牌（＞21）”“是否满21点（=21）”；初始手牌需检查是否为黑杰克。
  - 庄家回合：公开暗牌后检查“是否黑杰克”“是否21点”“是否爆牌”；自动要牌时，每次要牌后检查“是否爆牌”“是否21点”“是否≥17点（满足则停牌）”。

四、输入输出规范

1. **输入处理**：

   - 游戏启动：提示`“请输入数字选择：1–开始游戏”`，仅接受输入`"1"`。
   - 玩家回合：提示`“请输入数字选择：1–要牌，2–停牌”`，仅接受`"1"`或`"2"`。
   - 游戏结束：提示`“请输入数字选择：1–重新开始，2–结束游戏”`，仅接受`"1"`或`"2"`。
   - 无效输入：提示`“无效输入，请输入X或Y”`（`X`、`Y`为当前场景有效数字）。

2. **输出要求**：

   - 启动时显示欢迎界面、规则说明（含“数字选择→操作”对照表）。
   - 玩家手牌始终显示**完整牌面+实时点数**；庄家初始手牌显示为`“[明牌，*]”`（暗牌隐藏，仅玩家回合结束后公开）。
   - 实时提示关键状态：黑杰克（`“你拿到了黑杰克！”`/`“庄家拿到了黑杰克！”`）、爆牌（`“你爆牌了！”`/`“庄家爆牌了！”`）、21点（`“你的点数正好21点！”`）。
   - 胜负结果需明确：`“你赢了！”`/`“你输了！”`/`“平局！”`。
   - 用分隔线（如`=`）分隔不同阶段（如“玩家回合开始”“庄家回合开始”），提升排版可读性。

五、评分标准（总分100分）

1. **代码限制遵守（20分）**：

   - 未定义任何函数（8分，违反则此项得0分）。
   - 仅使用基础数据类型与允许的控制结构（7分）。
   - 未使用禁止的高级语法（5分）。

2. **功能完整性（40分）**：

   - 牌组初始化与随机发牌逻辑正确（5分）。
   - 点数计算准确（尤其是A的“11/1点”灵活切换）（8分）。
   - 庄家暗牌“隐藏→公开”逻辑正确（5分）。

   - 玩家回合流程完整（含实时检查：黑杰克、爆牌、21点）（7分）。
   - 庄家回合流程完整（含实时检查：黑杰克、爆牌、21点、“＜17必须要牌/≥17必须停牌”规则）（8分）。
   - 数字选择操作与循环控制无漏洞（7分）。

3. **逻辑正确性（25分）**：

   - 回合切换逻辑无漏洞（玩家→庄家回合衔接自然）（10分）。
   - 实时判定覆盖所有场景（无遗漏、无逻辑矛盾）（10分）。
   - 无死循环、流程卡死等运行时问题（5分）。

4. **用户体验（15分）**：

   - 输入提示清晰，操作引导明确（5分）。
   - 输出信息直观，关键状态提示准确（5分）。
   - 排版整洁，分隔线等格式增强可读性（5分）。

六、提交要求

1. 提交单个Python文件，文件名格式：`学号_姓名_blackjack.py`。

2. 代码开头需包含注释信息：

```
# 学号：XXX
# 姓名：XXX
# 日期：XXXX年XX月XX日
# 课程作业：二十一点游戏开发
```

3. 代码需可独立运行，无语法错误。

七、注意事项

1. **代码限制为核心考核点**：违反将直接扣除对应分数，严重者作业按零分处理。
2. 禁止抄袭：一经发现，作业按零分处理。
3. 功能与逻辑需严格匹配流程图及规则说明：对规则理解偏差会直接影响评分。

八、扩展任务（选做，不计入总分）

1. 实现“多副牌”功能（如模拟6副牌洗牌发牌）。

2. 增加简单赌注系统，支持输入数字作为赌注金额。
3. 记录玩家历史战绩（胜/负/平次数、最高点数等）并统计展示。