Homework 1: Yahtzee!

Introduction

The goal of this exercise is to implement a simplified version of the Yahtzee game in Python. You will complete the implementation of various functions that handle game logic, including rolling dice, scoring, and managing game rounds.

Submission Guidelines

1. This is an assignment to be done in pairs of exactly 2 students. Students that are caught cheating will obtain a score of 0 points. This homework is worth 20% of your final grade.
2. Comment your code properly, which includes naming your variables in a meaningful manner. Badly documented code will be penalized.
3. There is a 25% penalty for deliveries at most 2 days late. Homeworks submitted after this will be graded with 0 points.
4. Submit the completed `yahtzee.py` and `main.py` files in a single, zipped folder that respects the following naming convention `student1id_student2id.zip`. Make sure that other than the files listed above, no other files are included in the archive and that files are zipped; `.tar` or other compression archives are not accepted.
5. The submission deadline is **Saturday, 4th of October 2025, at 23:59**.
6. One group member shall submit the `.zip` file to the corresponding Moodle activity. It is not necessary for both group members to submit.

Setup

Before starting, ensure you have a working Python installation on your computer. This assignment covers functions, loops, and dictionaries as taught in the labs. You may develop the functions in a Jupyter notebook as usually done in the labs. For the final submission, however, the functions developed need to be put in `yahtzee.py` and called in `main.py`.

Restrictions

You are not allowed to use third-party libraries. Only use libraries that are part of the Python standard library (i.e. that do not require a `pip install` or `conda install`).

Files Provided

You are given two files:

1. `main.py`: Contains the main game loop and setup.
2. `yahtzee.py`: Contains the core game logic functions that you need to implement.

Task Breakdown

Your task is to complete the implementation of functions in the `yahtzee.py` file and the game loop in `main.py`. Below is a breakdown of each task:

Functions to Implement

1. **roll_dice(n)**:
   - Implement a function to simulate rolling `n` dice, each with a value between 1 and 6.
   - Use the `random` module to generate random integers between 1 and 6.
2. **create_empty_scorecard()**:
   - Initialize and return an empty scorecard as a dictionary.
   - The scorecard should have keys for each category: '1', '2', '3', '4', '5', '6', 'three_of_a_kind', 'four_of_a_kind', 'full_house', 'four_straight', 'five_straight', 'yahtzee', and 'chance'.
   - Each key should be initialized to `None`.
3. **select_keep(dice)**:
   - Prompt the user to select which dice to keep.
   - Use `input()` to read user input from the terminal/console. Users are expected to enter a string like "333" to select the dice they want to keep.
   - Ensure the input only contains valid dice numbers from the current roll.
   - Return the selected dice as a list.
4. **reroll(dice, kept)**:
   - Reroll the dice that were not kept.
   - Calculate the number of dice to reroll, generate new random dice, and return the combined dice roll.
5. **has_straight(dice, length)**:
   - Determine if the dice contain a straight of a specified length.
   - Sort the unique dice values and check if there is a sequence of 'length' consecutive numbers.
6. **evaluate(dice)**:
   - Calculate and return scores for all possible categories based on the current dice roll.
   - For each number category (1-6), sum the dice showing that number.
   - For 'three_of_a_kind', check if at least three dice show the same number and sum all dice if true.
   - For 'four_of_a_kind', check if at least four dice show the same number and sum all dice if true.
   - For 'full_house', check if there are three of one number and two of another, and return 25 if true.
   - For 'four_straight' and 'five_straight', check for sequences of four or five consecutive numbers, respectively, and return 30 or 40 if true.

- For 'yahtzee', check if all five dice show the same number and return 50 if true.
- For 'chance', sum all dice.

7. **choose(scores, used)**:
   - Present the available scoring options to the user and get their selection.
   - The code should reject the user choice if a category has already been chosen.
   - Display a list of available scoring options and prompt the user to choose one by entering the corresponding number.
8. **display_scorecard(card)**:
   - Display the current state of the scorecard.
   - Iterate through the scorecard dictionary and print each category and its score.
   - Calculate and display the upper section total, bonus, and total score.
9. **play_round(card)**:
   - Simulate a single round of Yahtzee with up to three dice rolls.
   - Implement logic for rolling dice up to three times per round, keeping dice between rolls, and selecting a category to score.

Implementation Steps

1. Open `yahtzee.py`.
2. Implement each function one by one according to the provided docstrings and hints.
3. Test each function individually to ensure it works as expected before moving on to the next.
4. Once all functions are implemented, edit the `main.py` file accordingly to create the complete game flow. You can test the game by running `python main.py` from your current location using the terminal/console. If using a Jupyter notebook, you can launch the terminal from there.

Important Notes

- **Show the scorecard after every round** to keep the player informed of their progress.
- Use an **iterative approach** to play the game for its 13 rounds.
- Implement logic for when the upper section of the game (i.e., the numbers 1 to 6) has at least 63 points, a bonus of 35 points should be applied.

Testing

- Test your functions with different inputs to ensure they handle edge cases.
- Play through several rounds of the game manually to verify that scoring and dice rolling work correctly.

What's in it for you?

By completing this exercise, you will have a working Yahtzee game for your gamenights with friends and a better understanding of implementing game logic in Python.