Assignment 1: Implement a Blackjack game engine

1 Overview

In this assignment, we will put into practice concepts from the first few lessons to implement the core logic of Blackjack. In particular, we will work with:

- if statements
- functions
- loops
- collections

Upon completion of the assignment, we will have a working Blackjack game. Good luck!

2 Rules of (simplified) Blackjack

Before getting to the code, it is important to familiarize ourselves with the rules of Blackjack. This section is relevant even for those that already know how to play Blackjack, because we will be making some simplifications.

2.1 Start of game and scoring

Our simplified game of Blackjack consists of 2 participants, the player and the dealer. The player is whoever will be interacting with our program, and the dealer will be simulated by code that we write. At the beginning of a game, each participant is dealt a hand consisting of 2 cards from a standard deck of 52 cards. A hand is associated with a score; a hand's score is computed as the sum of the values of the cards in the hand. For our purposes, the value of an ace will be 1[1], the value of cards 2-10 will just be that card's number, and the value of any face card is 10. For example, if a hand contains an ace of spades, a 6 of clubs, and queen of diamonds, the score of that hand is $1 + 6 + 10 = 17$. The objective of the game for each participant is to obtain a hand whose score is closest to 21 without exceeding it.

[1] in traditional blackjack, the value of an ace is either 1 or 11 depending on what is most advantageous

2.2 Player's turn

After the initial deal of hands, the player will go first. The player can choose to "Hit" or "Stay". If the player hits, they are dealt another card from the deck, and their hand's score is updated. The player can continue to hit as long as their score is at or below 21. If their score exceeds 21 after a hit, they have "busted" and the dealer wins. At any point prior to busting, the player can choose to stay, and in this case their turn ends and it becomes the dealers turn.

2.3 Dealer's turn

The dealer's turn proceeds identically to the player's turn. They can choose to "Hit" or "Stay". If the dealer hits, they are dealt another card from the deck, and their hand's score is updated. The dealer can continue to hit as long as their score is at or below 21. If their score exceeds 21 after a hit, they have "busted" and the player wins (assuming the player has not busted first). At any point prior to busting, the dealer can choose to stay, and in this case their turn ends and a winner must be determined based on hand scores.

2.4 Determining a winner

If any participant has busted, then the other participant wins. If neither participant has busted, and both have stayed, the winner is the one whose hand is closest to 21. If the player and dealer have the same score, the result is a tie.

3 Setup

The files for this assignment can be obtained by cloning the `learncoding` repository from GitHub. The relevant files can be found under the `learncoding/assignments/assignment1` directory. In particular, there are two python files which we will interact with: `blackjack.py` and `test_blackjack.py`.

3.1 `blackjack.py`

The `blackjack.py` file contains the core logic of the game. It contains several `TODO` comments, which we will be responsible for implementing as part of the assignment. There is also logic to display the game graphically on the command line, but this part has been done for us.

3.2 `test_blackjack.py`

The `test_blackjack.py` file contains several tests for `blackjack.py`. In other words, if we implement `blackjack.py` correctly, all these tests should pass. During our debugging, we may need to refer to the contents of this file to figure out why a certain test is failing.

3.3 Running the game

To run the game, simply run the `blackjack.py` python program. This can be accomplished from the command line by changing into the `assignment1` directory, and executing the command `python3 blackjack.py` (substitute `python3` for whatever alias you have chosen for your python interpreter as appropriate). You can also run the program by opening the `blackjack.py` file in the Python IDLE, and navigating to Run Program>Run Program in the menu bar.

If this was successful, you should see a graphical display of a game of Blackjack show up in your terminal. It may have some slight visual oddities, but that's because we still need to implement the core functionality.

3.4 Testing and debugging the code

Although it is possible to manually test our code by running a new game, we can also test the code by running the `test_blackjack.py` program. When executed, the `test_blackjack.py` program outputs the results of several test cases to the terminal. Initially, most tests will fail as we have not yet implemented the core logic. As we implement more of the logic, more tests should pass.

If you are unsure why a test is failing, note the test name and find its logic within `test_blackjack.py`. Try to run through each line of the test in your head, and hopefully identify what in your implementation is causing the test to fail. This process is called debugging.

4 Design

Before getting to the code, we describe how the existing code represents cards and hands. This will be helpful because the code we write will must interact with these abstract representations.

4.1 Cards

We represent a card as an integer between 0 and 51, inclusive. The first 13 integers correspond to the spades suite, the next 13 correspond to hearts, the next 13 correspond to diamonds, and the last 13 correspond to clubs. Within a suite, the first (lowest) integer corresponds to an ace, the second to a 2, the third to a 3, ..., and the last (highest) to a king. For example, the integer 24 represents the queen of hearts.

4.2 Hands

We represent a hand as a list of cards (i.e. integers). For example, the list [8, 39] represents a hand consisting of a 9 of spades and an ace of clubs.

5 Instructions

Complete the following steps in order to finish the assignment. After each step, more tests should pass. At the end, every test should pass. If for some reason a test is not passing, you will need to debug your code. All code changes should occur only in `blackjack.py`

1. Implement the `score_card` function. Remember to take into account our 0-51 representation of cards.
2. Implement the `score_hand` function. Recall that the score of a hand is just the sum of the scores of each individual card in that hand.
3. Implement the `will_dealer_hit` function. In our version of Blackjack, the dealer should hit as long as their current score is less than 17.
4. Implement the `is_bust_hand` function. Recall that a busted hand is one whose score exceeds 21.
5. Implement the `card_suite_to_str` and `card_name_to_str` functions. Remember to take into account our 0-51 representation of cards.

Once every test passes, make sure to celebrate your success by playing some Blackjack! The graphical interface of the game should have no defects after successful completion of the above steps.