Peer Assessments (https://class.coursera.org/interactivepython-004/human_grading/)
/ Mini-project # 6 - Blackjack
Help (https://class.coursera.org/interactivepython-004/help/peergrading?
url=https%3A%2F%2Fclass.coursera.org%2Finteractivepython-
004%2Fhuman_grading%2Fview%2Fcourses%2F972072%2Fassessments%2F33%2Fsubmissions)

due in 6day 24h

Submission Phase

1. Do assignment ☐ (/interactivepython-004/human_grading/view/courses/972072/assessments/33/submissions)

Evaluation Phase

2. Evaluate peers 🔒 (/interactivepython-004/human_grading/view/courses/972072/assessments/33/peerGradingSets)

3. Self-evaluate 🔒 (/interactivepython-004/human_grading/view/courses/972072/assessments/33/selfGradingSets)

Results Phase

4. See results 🔒 (/interactivepython-004/human_grading/view/courses/972072/assessments/33/results/mine)

☐ In accordance with the Honor Code, I certify that my answers here are my own work, and that I have appropriately acknowledged all external sources (if any) that were used in this work.

Save draft   Submit for grading

A reminder about the Honor Code

For previous mini-projects, we have had instances of students submitting solutions that have been copied from the web. Remember, if you can find code on the web for one of the mini-projects, we can also find that code. Submitting copied code violates the Honor Code (../../../../../wiki/view?page=honorcode) for this class as well as Coursera's Terms of Service. Please write your own code and refrain from copying. If, during peer evaluation, you suspect a submitted mini-project includes copied code, please evaluate as usual and email the assignment details to iipphonorcode@online.rice.edu (mailto:iipphonorcode@online.rice.edu). We will investigate and handle as appropriate.

Mini-project description - Blackjack

Blackjack is a simple, popular card game that is played in many casinos. Cards in Blackjack have the following values: an ace may be valued as either 1 or 11 (player's choice), face cards (kings, queens and jacks) are valued at 10 and the value of the remaining cards corresponds to their number. During a round of

Blackjack, the players plays against a dealer with the goal of building a hand (a collection of cards) whose cards have a total value that is higher than the value of the dealer's hand, but not over 21. (A round of Blackjack is also sometimes referred to as a hand.)

The game logic for our *simplified* version of Blackjack is as follows. The player and the dealer are each dealt two cards initially with one of the dealer's cards being dealt faced down (his *hole* card). The player may then ask for the dealer to repeatedly "hit" his hand by dealing him another card. If, at any point, the value of the player's hand exceeds 21, the player is "busted" and loses immediately. At any point prior to busting, the player may "stand" and the dealer will then hit his hand until the value of his hand is 17 or more. (For the dealer, aces count as 11 unless it causes the dealer's hand to bust). If the dealer busts, the player wins. Otherwise, the player and dealer then compare the values of their hands and the hand with the higher value wins. **The dealer wins ties in our version.**

Mini-project development process

We suggest you develop your Blackjack game in two phases. The first phase will concentrate on implementing the basic logic of Blackjack while the second phase will focus on building a more full-featured version. In phase one, you will use buttons to control the game and print the state of the game to the console using print statements. In the second phase, you will replace the print statements by drawing images and text on the canvas and add some extra game logic.

In phase one, we will provide testing templates for four of the steps. The templates are designed to check whether your class implementations work correctly. You should copy your class definition into the testing template and compare the console output generated by running the template with the provided output. If the output matches, it is likely that your implementation of the class is correct. **DO NOT PROCEED TO THE NEXT STEP UNTIL YOUR CODE WORKS WITH THE PROVIDED TESTING TEMPLATE.** Debugging code that uses incorrectly implemented classes is extremely difficult. Avoid this problem by using our provided testing templates.

Phase one

1. Download the **program template** (http://www.codeskulptor.org/#examples-blackjack_template.py) for this mini-project and review the class definition for the `Card` class. This class is already implemented so your task is to familiarize yourself with the code. Start by pasting the `Card` class definition into the provided **testing template** (http://www.codeskulptor.org/#examples-card_template.py) and verifying that our implementation works as expected.

2. Implement the methods `__init__, __str__, add_card` for the `Hand` class. We suggest modeling a hand as a list of cards. For help in implementing the `__str__` method for hands, refer back to practice exercise number four from last week. Remember to use the string method for cards to convert each card object into a string. Once you have implemented the `Hand` class, test it using the provided **testing template** (http://www.codeskulptor.org/#examples-hand_template.py).

3. Implement the methods for the `Deck` class listed in the mini-project template. We suggest modeling a deck of cards as list of cards. You can generate this list using a pair of nested `for` loops or a list comprehension. Remember to use the `Card` initializer to create your cards. Use `random.shuffle()` to shuffle this deck of cards. Once you have implemented the `Deck` class, test your Deck class using the provided **testing template** (http://www.codeskulptor.org/#examples-deck_template.py). Remember that the deck is randomized after shuffling, so the output of the testing template should match the output in the comments in form but not in exact value.

4. Implement the handler for a "Deal" button that shuffles the deck and deals the two cards to both the dealer and the player. The event handler `deal` for this button should shuffle the deck (stored as a global variable), create new player and dealer hands (stored as global variables), and add two cards to each hand. To transfer a card from the deck to a hand, you should use the `deal_card` method of the `Deck` class and the `add_card` method of `Hand` class in combination. The resulting hands should be printed to the console with an appropriate message indicating which hand is which.

5. Implement the `get_value` method for the `Hand` class. You should use the provided `VALUE` dictionary to look up the value of a single card in conjunction with the logic explained in the video lecture for this project to compute the value of a hand. Once you have implemented the `get_value` method, test it using the provided **testing template** (http://www.codeskulptor.org/#examples-getvalue_template.py).

6. Implement the handler for a "Hit" button. If the value of the hand is less than or equal to 21, clicking this button adds an extra card to player's hand. If the value exceeds 21 after being hit, print "You have busted".

7. Implement the handler for a "Stand" button. If the player has busted, remind the player that they have busted. Otherwise, repeatedly hit the dealer until his hand has value 17 or more (using a while loop). If the dealer busts, let the player know. Otherwise, compare the value of the player's and dealer's hands. If the value of the player's hand is less than or equal to the dealer's hand, the dealer wins. Otherwise the player has won. **Remember the dealer wins ties in our version.**

In our version of Blackjack, a hand is automatically dealt to the player and dealer when the program starts. In particular, the program template includes a call to the `deal()` function during initialization. At this point, we would suggest testing your implementation of Blackjack extensively.

Phase two

In the second phase of your implementation, you will add five features. For those involving drawing with global variables, remember to initialize these variables to appropriate values (like creating empty hands for the player and dealer) just before starting the frame.

1. Implement your own `draw` method for the `Hand` class using the `draw` method of the `Card` class. We suggest drawing a hand as a horizontal sequence of cards where the parameter `pos` is the position of the upper left corner of the leftmost card. To simplify your code, you may assume that only the first five cards of a player's hand need to be visible on the canvas.

2. Replace printing in the console by drawing text messages on the canvas. We suggest adding a global `outcome` string that is drawn in the draw handler using `draw_text`. These messages should prompt the player to take some require action and have a form similar to "Hit or stand?" and "New deal?". Also, draw the title of the game, "Blackjack", somewhere on the canvas.

3. Add logic using the global variable `in_play` that keeps track of whether the player's hand is still being played. If the round is still in play, you should draw an image of the back of a card (provided in the template) over the dealer's first (hole) card to hide it. Once the round is over, the dealer's hole card should be displayed.

4. Add a score counter that keeps track of wins and losses for your Blackjack session. In the simplest case (see our demo), the program displays wins minus losses. However, you are welcome to implement a more sophisticated betting/scoring system.

5. Modify the logic for the "Deal" button to create and shuffle a new deck (or restock and shuffle an existing deck) each time the "Deal" button is clicked. This change avoids the situation where the deck becomes empty during play.

6. Finally, modify the `deal` function such that, if the "Deal" button is clicked during the middle of a round, the program reports that the player lost the round and updates the score appropriately.

Congratulations! You have just built Blackjack. To wrap things up, please review the demo of our version of Blackjack in the Blackjack video lecture to ensure that your version has full functionality.

Grading rubric - 18 pts total (scaled to 100)

You must implement the simplified rules of Blackjack specified in this description. Small variations from our demo version are acceptable such as displaying the value of a hand or implementing a betting system. **But, you may not change the logic of the game.** After the submission deadline, you are welcome to post enhanced versions of Blackjack to the Hall of Fame with more realistic game logic such as pushes on ties, splitting pairs and doubling down.

- 1 pt - The program displays the title "Blackjack" on the canvas.

- 1 pt - The program displays 3 buttons ("Deal", "Hit" and "Stand") in the control area.
- 2 pts - The program graphically displays the player's hand using card images. (1 pt if text is displayed in the console instead)
- 2 pts - The program graphically displays the dealer's hand using card images. Displaying both of the dealer's cards face up is allowable when evaluating this bullet. (1 pt if text displayed in the console instead)
- 1 pt - The dealer's hole card is hidden until the current round is over. After the round is over, it is displayed.
- 2 pts - Pressing the "Deal" button deals out two cards each to the player and dealer. (1 pt per player)
- 1 pt - Pressing the "Deal" button in the middle of the round causes the player to lose the current round.
- 1 pt - Pressing the "Hit" button deals another card to the player.
- 1 pt - Pressing the "Stand" button deals cards to the dealer as necessary.
- 1 pt - The program correctly recognizes the player busting.
- 1 pt - The program correctly recognizes the dealer busting.
- 1 pt - The program correctly computes hand values and declares a winner. Evaluate based on messages.
- 2 pts - The program accurately prompts the player for an action with messages similar to "Hit or stand?" and "New deal?". (1 pt per message)
- 1 pt - The program implements a scoring system that correctly reflects wins and losses. Please be generous in evaluating this item.

In the submission phase, cut and paste the URL for your cloud-saved mini-project into the box below. Click the Honor Code box and hit the "Submit for grading" button when you are ready to submit your mini-project. (You may submit as many times as you like before the deadline so we suggest that you use "Submit" instead of "Save".) **IMPORTANT: Please use the "Review your work" link that appears at the top of the subsequent submission page to verify that you submitted a working link for the final version of your mini-project.**

☐ In accordance with the Honor Code, I certify that my answers here are my own work, and that I have appropriately acknowledged all external sources (if any) that were used in this work.

Save draft   Submit for grading