University of St. Gallen

Blackjack – Python

Introduction to Programming

Dr. Mario Silic

22.5.2020

Corsin Dosch (15-731-185), Deniz Süzen (17-609-066), Dimitri de Witt (15-412-299), Samuel Beyene (17-612-037)

User names from left to right: Corsin, CoderDen, Cobra, SamuelB

As a team, we decided to create a game, a Blackjack game that will allow the player to join a table or to play alone. In order to code efficiently, we imported two libraries, random and time. They respectively support our code to shuffle the decks and to control the speed of execution.

Firstly, we created two functions that respectively slow the execution of a regular print of strings and the print of a list, each character will be executed with a delay of 0.01 second. This function will constantly be executed to make sure that the speed of execution stays fluid. To separate some phases of the game, we will also use a time.sleep() functionality. Afterwards, we created a class with two functions which will control the player's input. Each time the player will be asked to answer a question or to insert a number, we will make sure that only specific inputs can be entered. In order to adequately guide the player, we needed to add several parameters for each function as they will react differently. Those two functions share similar aspects, they use if statements as well as a while loop. IF-ELIF-ELSE statements are efficient in directing python in the answer that should be given for the player to understand why his response was rejected. The while loop was required to make sure that the question will continue to be asked until an accepted input is provided. This is followed by a try – except function which blocks the player from inserting wrong inputs and for the while loop to stop when the system reaches a break statement.

Secondly, the player is asked whether he wants to learn something about Blackjack and with how many decks he would want to play with. The answer to the latter question will then be used in the function, which creates the deck. The creation of the deck is initiated by several FOR statements with values that are added to an empty list with the function append. Both the heads and the numbers are added to the deck. Then, a nested FOR-loop combined those values in the list deck to the names of the suites in another empty list. The first for loop will stop at the end of the list suites whereas the second one runs until the end of the first deck.
After the creation of the decks, the code needs to know with how many CPUs, excluding the dealer, the player wants to play with and where the player would want to sit. In fact, when playing Blackjack, players carefully choose where to sit in order to make their decisions depending on the other players' hands. At this point, it was important to create a function which can add the value of the cards as the players need to be guided and the dealer has to say whether they can continue playing or if they lost. In order to create such a function, a comprehensive dictionary was added. Every card except for the Ace has a specific value attached to it, indeed, the Ace can either take the value of 1 or 11 depending on the player's strategic decision.
The function check_cards calculates the sum of the cards with the dictionary, which translates the cards to their specific numeric values. However, as the Ace can either take the value of a 1 or an 11, previous sums as well as the turn number will have to be considered. The Ace's value has to be stored and used again for the player not to have the opportunity to change his mind. Also, one can note that the function reacts differently for CPUs and players. The player will have to choose the value of the Ace whereas the CPU will always follow a best response strategy.

Then, we initiate the game and distribute the first cards. However, as a specific card from a deck cannot be distributed twice, a function had to be implemented for the dealer to always select the first card in the deck. In order to deal the cards efficiently, we remove the card withdrawn from the deck using the pop function each time a card is withdrawn which ensures that we always give the first card of the deck as in real life if someone draws a card. This will diminish the list decks each time a card is provided to a player.

Afterwards, cards need to be distributed to players and in the specific order depending on the player's position at the table. Therefore, we create empty lists, a total of seven CPUs as well as the player_cards and the dealer_cards. As the player can choose to play with 6 other people excluding the dealer and can be positioned anywhere, a total of 7 people will be at the table. Consequently, one list will always stay empty but needs to be created for the player to sit anywhere at the table. Moreover, during the code the number of players was counted but always excluded both the dealer and the player. Thus, if the player plays alone against the dealer, only two will need to be added to the number of players. The range in FOR statements does not take into account the last value, the second parenthesis is not inclusive. For example, for I in range(0,2) will only compute 0 and 1. If the player wanted to play with CPUs, a total of three will need to be added instead of two because of the position at the table. As it starts at 1, when i equals 0 nothing will happen. The dealer is recognized as z-1 as the statement is not inclusive, the dealer will receive its card when the value of i equates z-1. This and the player's position had to be on the top in order to block the program from giving the card to a CPU as the i would have been recognized in one of the succeeding lines (the player takes the place of one of the CPU's).

For the second round, we created functions for steering the behavior of the CPU players and the dealer. We made an "amateur" strategy, a "better" strategy and a strategy for the dealer. That means that these players will request another card if their total sum of cards is under the value of 19, 21 and 17, respectively.

In addition, the player has the option to split his cards, if he got two cards with the equal value (for number-cards) or the same picture (for Jacks, Queens, Kings and Aces). To check if the cards are equal, we created a function, which compares the first letter (or number) of the two elements in the list "player_cards". If they are equal, the question appears, if the player wants to split or not and the answer will be saved for the second round.

In the second round of the game, where players can decide whether they want to hit another card or stay with what they already received, we used the same code like in the first round but added the function with the different CPU decisions to give them a behavior. For the player of the game, we made a way with split and one without split. If the answer to the split question is yes, we created a left and a right hand. The first element in the list of the player cards is appended to the new list hand1. To make it clearer for the player, the dealer concentrates just on the first card remaining after the split until the player doesn't want another card and stands with his score or has exceeded a value of 21, before taking care of the other card. He gets asked if he wants to hit or stand. We created a while loop for the answer yes of if he wants another card or not, so that every time he sees his new hand, the same question, whether to hit or stand, appears until he decides to stay with his current value. He can choose another card, as long as his score is under 22, for that we used an if statement. After the process for the first side of the split is done, we used the same code for the other side. The only difference is that we used the name previous_sum_1 instead of previous_sum because of the results in the end, where we need both scores to determine the outcomes.

If the player doesn't want or doesn't even have the opportunity to split, there is the way of the second round without split. We let python check if the split question has any other answer than yes and if this is true, the game continues with the two cards received in the beginning in one list. After the question if he wants to hit or stand, we used again the while loop like before in the split part, where another card is distributed as long as the player decides to hit and the value of his hand is under 22. After he wants to stand or exceeded a value of 21, the game continues with the CPU players at a later position on the table than him, if there are any. Additionally, if the player busts (value of the cards over 21), a message appears that tells the player that he lost at this point. The sum of the cards is saved as previous_sum. In the end, the cards of the dealer are shown. At this point the part of the gameplay is terminated, and the results have to be presented in a last step.

For the results we created a function, which compares the values of the cards of the player and the dealer. For that we used the check_cards function for the dealer. For the player we took the previous_sum (and in case of a split previous_sum1 additionally). Each scenario is checked through an if statement and contains a message as well as the calculation of the amount of money the player receives. Again, we made a way with and without split. If the split question was answered with yes, the player receives a result for each hand.

After the player received the results and the information about whether one won or not, he is asked if he wants to play again. If the player wants to play again, the main() function will be executed again. Therefore, to be able play again without the need to stop and run the program again, the whole code for the game is inserted into the main() function. As long as the player inserts “yes” to the play again question, the player can play again as much as he wants. In case the player should insert “no” the program stops running.