**Card Game with Python**

In this article, I’ll walk you through how to make a Card Game with Python. In this card game, each player draws a card from the deck and the player with the highest card wins. I’m going to build this card game by defining classes representing a card, a deck, a player and finally the game itself.

## Card Game with Python: Class Card

Here is a class that models playing cards:

1. class Card:
2. suits = ["spades",
3. "hearts",
4. "diamonds",
5. "clubs"]
7. values = [None, None,"2", "3",
8. "4", "5", "6", "7",
9. "8", "9", "10",
10. "Jack", "Queen",
11. "King", "Ace"]
12. def \_\_init\_\_(self, v, s):
13. """suit + value are ints"""
14. self.value = v
15. self.suit = s
16. def \_\_lt\_\_(self, c2):
17. if self.value < c2.value:
18. return True
19. if self.value == c2.value:
20. if self.suit < c2.suit:
21. return True
22. else:
23. return False
24. return False
25. def \_\_gt\_\_(self, c2):
26. if self.value > c2.value:
27. return True
28. if self.value == c2.value:
29. if self.suit > c2.suit:
30. return True
31. else:
32. return False
33. return False
34. def \_\_repr\_\_(self):
35. v = self.values[self.value] +\
36. " of " + \
37. self.suits[self.suit]
38. return v

The first class in our card game with Python is a Card class, which has two class variables, suits and values. Suits is a tuple of strings representing all the suits a card can be: spades, hearts, diamonds, clubs. value is a tuple of strings representing the different numeric values a card can be: 2-10, Jack, Queen, King, and Ace.

The elements of the first two indexes of the value tuple are None, so the strings in the tuple match the index they represent. So the string “2” in the tuple of values is at index 2.

Card objects have two instance variables: suit and value, each represented by an integer. Together, the instance variables represent the card type of the Card object. For example, you create a 2 of hearts by creating a Card object and passing it the parameters 2 (for color) and 1.

The code for these magic methods can also handle whether the cards have the same value, for example, if both cards have a value of 10. If this happens, the methods use the value of the combinations to break the tie.

The combinations are ranked in order of strength in the combination tuple – with the strongest combination last, and thus assigned to the highest index, and the less powerful combination to the lowest index.

## Card Game with Python: Class Deck

Next, you need to define a class to represent a deck of cards:

1. from random import shuffle

4. **class** Deck:
5. def \_\_init\_\_(self):
6. self.cards = []
7. **for** i in range(2, 15):
8. **for** j in range(4):
9. self.cards\
10. .append(Card(i,
11. j))
12. shuffle(self.cards)
14. def rm\_card(self):
15. **if** len(self.cards) == 0:
16. **return**
17. **return** self.cards.pop()

When you initialize the Deck object, the two for loops of \_\_init\_\_ create Card objects representing all the cards in a 52-card deck and add them to the card list. The first loop goes from 2 to 15 because the first value of a card is 2 and the last value of a card is 14 (the ace).

Each time around the inner loop, a new card is created using the integer from the outer loop as the value (i.e. 14 for an ace) and the integer from the inner loop as the suit. This process creates 52 cards – one card for each combination of suit and value.

After the method creates the cards, the shuffle method of the shuffle module randomly rearranges the items in the card list; imitating the shuffle of a deck of cards.

Our deck has another method called rm\_card which removes and returns a card from the card list, or returns None if it is empty.

#### Card Game with Python: Class Player

You need a class to represent each player in the game to keep track of their cards and the number of tricks won:

1. **class** Player:
2. def \_\_init\_\_(self, name):
3. self.wins = 0
4. self.card = None
5. self.name = name

The Player class has three instance variables: wins to keep track of the number of turns a player has won, card to represent the card a player currently holds, and name to keep track of a player’s name.

#### Card Game with Python: Class Game

Finally, you need a class to represent the game:

1. **class** Game:
2. def \_\_init\_\_(self):
3. name1 = input("p1 name ")
4. name2 = input("p2 name ")
5. self.deck = Deck()
6. self.p1 = Player(name1)
7. self.p2 = Player(name2)
9. def wins(self, winner):
10. w = "{} wins this round"
11. w = w.format(winner)
12. print(w)
14. def draw(self, p1n, p1c, p2n, p2c):
15. d = "{} drew {} {} drew {}"
16. d = d.format(p1n,
17. p1c,
18. p2n,
19. p2c)
20. print(d)
22. def play\_game(self):
23. cards = self.deck.cards
24. print("beginning War!")
25. **while** len(cards) >= 2:
26. m = "q to quit. Any " + \
27. "key to play:"
28. response = input(m)
29. **if** response == 'q':
30. **break**
31. p1c = self.deck.rm\_card()
32. p2c = self.deck.rm\_card()
33. p1n = self.p1.name
34. p2n = self.p2.name
35. self.draw(p1n,
36. p1c,
37. p2n,
38. p2c)
39. **if** p1c > p2c:
40. self.p1.wins += 1
41. self.wins(self.p1.name)
42. **else**:
43. self.p2.wins += 1
44. self.wins(self.p2.name)
46. win = self.winner(self.p1,
47. self.p2)
48. print("War is over.{} wins"
49. .format(win))
51. def winner(self, p1, p2):
52. **if** p1.wins > p2.wins:
53. **return** p1.name
54. **if** p1.wins < p2.wins:
55. **return** p2.name
56. **return** "It was a tie!"

When you create the game object, Python calls the \_\_init\_\_ method and the input function collects the names of the two players in the game and stores them in the variables name1 and name2.

Then you create a new Deck object, store it in the instance variable deck, and create two Player objects using the names in name1 and name2. The play\_game method of the Game class starts the game. There is a loop in the method that maintains the game as long as there are two or more cards left in the deck, and as long as the variable response is not equal to q .

On each loop turn, you assign the variable response to user input. The game continues until the user types “q” or when there are less than two cards left in the deck. Two cards are drawn each time in the loop and the play\_game method assigns the first card to p1 and the second card to p2.

Then he prints each player’s name and the card he drew, compares the two cards to see which card is the bigger one, increments the wins instance variable for the player with the highest card, and prints a message indicating who won. The Game class also has a method called winning which takes two player items, looks at the number of tricks they’ve won, and returns the player who has won the most tricks.

When the Deck object runs out of cards, the play\_game method displays a message that the war is over, calls the winning method (passing both p1 and p2), and displays a message with the result – player name who won.

1. from random import shuffle

4. **class** Card:
5. suits = ["spades",
6. "hearts",
7. "diamonds",
8. "clubs"]
10. values = [None, None,"2", "3",
11. "4", "5", "6", "7",
12. "8", "9", "10",
13. "Jack", "Queen",
14. "King", "Ace"]
16. def \_\_init\_\_(self, v, s):
17. """suit + value are ints"""
18. self.value = v
19. self.suit = s
21. def \_\_lt\_\_(self, c2):
22. **if** self.value < c2.value:
23. **return** True
24. **if** self.value == c2.value:
25. **if** self.suit < c2.suit:
26. **return** True
27. **else**:
28. **return** False
29. **return** False
31. def \_\_gt\_\_(self, c2):
32. **if** self.value > c2.value:
33. **return** True
34. **if** self.value == c2.value:
35. **if** self.suit > c2.suit:
36. **return** True
37. **else**:
38. **return** False
39. **return** False
41. def \_\_repr\_\_(self):
42. v = self.values[self.value] +\
43. " of " + \
44. self.suits[self.suit]
45. **return** v

48. **class** Deck:
49. def \_\_init\_\_(self):
50. self.cards = []
51. **for** i in range(2, 15):
52. **for** j in range(4):
53. self.cards\
54. .append(Card(i,
55. j))
56. shuffle(self.cards)
58. def rm\_card(self):
59. **if** len(self.cards) == 0:
60. **return**
61. **return** self.cards.pop()

64. **class** Player:
65. def \_\_init\_\_(self, name):
66. self.wins = 0
67. self.card = None
68. self.name = name

71. **class** Game:
72. def \_\_init\_\_(self):
73. name1 = input("p1 name ")
74. name2 = input("p2 name ")
75. self.deck = Deck()
76. self.p1 = Player(name1)
77. self.p2 = Player(name2)
79. def wins(self, winner):
80. w = "{} wins this round"
81. w = w.format(winner)
82. print(w)
84. def draw(self, p1n, p1c, p2n, p2c):
85. d = "{} drew {} {} drew {}"
86. d = d.format(p1n,
87. p1c,
88. p2n,
89. p2c)
90. print(d)
92. def play\_game(self):
93. cards = self.deck.cards
94. print("beginning War!")
95. **while** len(cards) >= 2:
96. m = "q to quit. Any " + \
97. "key to play:"
98. response = input(m)
99. **if** response == 'q':
100. **break**
101. p1c = self.deck.rm\_card()
102. p2c = self.deck.rm\_card()
103. p1n = self.p1.name
104. p2n = self.p2.name
105. self.draw(p1n,
106. p1c,
107. p2n,
108. p2c)
109. **if** p1c > p2c:
110. self.p1.wins += 1
111. self.wins(self.p1.name)
112. **else**:
113. self.p2.wins += 1
114. self.wins(self.p2.name)
116. win = self.winner(self.p1,
117. self.p2)
118. print("War is over.{} wins"
119. .format(win))
121. def winner(self, p1, p2):
122. **if** p1.wins > p2.wins:
123. **return** p1.name
124. **if** p1.wins < p2.wins:
125. **return** p2.name
126. **return** "It was a tie!"
128. game = Game()
129. game.play\_game()