

```
1 """
2 Analyzes a card
3
4 :reflection: Defined functions called get_suit and
   get_num_char for information hiding.
5 """
6
7 def get_suit(hands, card_order):
8     """
9     Gets the card suit with the card order from hands
10
11     :param hands: a list for hands
12     :param card_order: an integer for the index of specific
   card
13     :return: a string for the card suit
14     """
15     return hands[card_order][len(hands[card_order]) - 1]
16
17 def get_num_char(hands, card_order):
18     """
19     Gets the card number or character with the card order from
   hands
20
21     :param hands: a list for hands
22     :param card_order: an integer for the index of specific
   card
23     :return: a string for the card number or character
24     """
25     return hands[card_order][0: len(hands[card_order]) - 1]
26
27 if __name__ == "__main__":
28     print("expected suit: C, actual suit: ", get_suit(["AC", "
   2D", "3H", "4S", "5S"], 0))
29     print("expected number: A, actual number: ", get_num_char
   (["AC", "2D", "3H", "4S", "5S"], 0))
```

```
1  """
2  Models a deck
3
4  :reflection: Used named constants like MAX_NUMBER because it
   is convenient to change their values throughout the code.
5  Used string for cards like "2C" (two clover) because it is
   easy to compare their numbers, characters, or suits
6  if their lengths and index are similar.
7  Defined functions called shuffle, draw, and card_remaining for
   the modularity.
8  """
9
10 import random
11
12 MAX_NUMBER = 10
13 SUITS = ["C", "D", "H", "S"]
14 CHARACTERS = ["A", "J", "Q", "K"]
15
16 def create_number():
17     """
18     Creates a deck of number cards
19
20     :return: a list for a deck of number cards
21     """
22     number_deck = []
23
24     for i in range (2, MAX_NUMBER + 1):
25         for j in SUITS:
26             number_deck.append(str(i) + j)
27
28     return number_deck
29
30 def create_standard():
31     """
32     Creates a standard deck of 52 cards
33
34     :return: a list for a deck of cards
35     """
36     standard_deck = create_number()
37
38     for i in CHARACTERS:
```

```
39         for j in SUITS:
40             standard_deck.append(i + j)
41
42     return standard_deck
43
44 def shuffle(deck):
45     """
46     Shuffles the deck
47
48     :param deck: a list for a deck of cards
49     :return: a list for a shuffled deck of cards
50     """
51     return random.shuffle(deck)
52
53 def draw(deck):
54     """
55     Draw a card from the standard deck
56
57     :param deck: a list for a deck of cards
58     :return: a string drawn randomly
59     """
60     return deck.pop(random.randrange(0, len(deck)))
61
62 def cards_remaining(deck):
63     """
64     Returns a number of cards left in the deck
65
66     :param deck: a list for a deck of cards
67     :return: an integer for cards left
68     """
69     return len(deck)
70
71 if __name__ == "__main__":
72     deck = create_standard()
73
74     print("deck: ", deck)
75     print("card: ", draw(deck))
76     print("cards remaining: ", cards_remaining(deck))
```

```
1  """
2  Models hands and analyzes cards
3
4  :reflection: Imported files called deck and card for the
    modularity.
5  Used named constants like MAX_HANDS because it is convenient
    to change their values throughout the code.
6  Used helper functions like d.draw(deck) for the modularity.
7  Defined functions called hands_extend and card_remove for the
    modularity and information hiding.
8  Defined functions starts with "is_..." for the modularity.
9  Used helper functions like get_suit(hands, i) for the
    modularity and information hiding.
10 """
11
12 import deck as d
13 import card as c
14
15 MAX_HANDS = 5
16 ATTEMPTS = 10000
17
18 def create_hands(deck):
19     """
20     Creates a list for poker hands with 5 cards
21     :return: a list for poker hands
22     """
23     hands = []
24
25     for hand in range (MAX_HANDS):
26         hands.append(d.draw(deck))
27
28     return hands
29
30 def hands_extend(current_hands, given_hands):
31     """
32     Extends current hands with given hands
33
34     :param current_hands: a list for current hands
35     :param given_hands: a list for given hands
36     :return: a list for extended hands
37     """
```

```
38     return current_hands.extend(given_hands)
39
40 def card_remove(hands, card_order):
41     """
42     Removes the card with the card order from hands
43
44     :param hands: a list for hands
45     :param card_order: an integer for card number
46     :return: a list for modified hands
47     """
48     return hands.remove(hands[card_order])
49
50 def is_flush(hands):
51     """
52     Checks whether hands are flush
53
54     :param hands: a list for hands
55     :return: True if all cards have the same shape
56     """
57     for i in range(1, MAX_HANDS):
58         if c.get_suit(hands, i) != c.get_suit(hands, i - 1):
59             return False
60
61     return True
62
63 def is_two_pair(hands):
64     """
65     Checks whether hands are two pair
66
67     :param current_hands: a list for hands
68     :return: True if hands have 2 pairs of the same number or
69     character
70     """
71     current_hands = []
72     hands_extend(current_hands, hands)
73
74     total_pair = 0
75     i = 1
76
77     while len(current_hands) > i:
78         if c.get_num_char(current_hands, 0) == c.get_num_char(
```

```
77     current_hands, i):
78         total_pair += 1
79
80         card_remove(current_hands, i)
81         card_remove(current_hands, 0)
82
83         i = 1
84
85     else:
86         i += 1
87
88     if i == len(current_hands):
89         card_remove(current_hands, 0)
90
91         i = 1
92
93     if total_pair == 2:
94         return True
95
96     return False
97
98 def is_pair(hands):
99     """
100     Checks whether hands are two pair
101
102     :param current_hands: a list for hands
103     :return: True if hands have a pairs of the same number or
104             character
105     """
106     current_hands = []
107     hands_extend(current_hands, hands)
108
109     total_pair = 0
110     i = 1
111
112     while len(current_hands) > i:
113         if c.get_num_char(current_hands, 0) == c.get_num_char
114         (current_hands, i):
115             total_pair += 1
116
117             card_remove(current_hands, i)
```

```
116         card_remove(current_hands, 0)
117
118         i = 1
119
120     else:
121         i += 1
122
123     if i == len(current_hands):
124         card_remove(current_hands, 0)
125
126         i = 1
127
128     if total_pair == 1:
129         return True
130
131     return False
132
133 def confirm_result(hands):
134     """
135     Confirms the actual result of hands
136
137     :param hands: a list for hands
138     :return:
139     """
140     if is_flush(hands):
141         print(hands, " is flush")
142
143     elif is_two_pair(hands):
144         print(hands, " is two pair")
145
146     elif is_pair(hands):
147         print(hands, " is pair")
148
149     else:
150         print(hands, "is high card")
151
152 if __name__ == "__main__":
153     deck = d.create_standard()
154
155     for i in range (10):
156         hands = create_hands(deck)
```

157

158 confirm_result(hands)


```
1  """
2  Computes a table
3
4  :reflection: Imported files called deck and hands for the
    modularity.
5  Used named constant like ATTEMPTS because it is convenient to
    change its value throughout the code.
6  Defined functions called first_row and get_result for the
    modularity.
7  Defined a function called get_percentage and get_content for
    the modularity and information hiding.
8  Used helper functions like d.create_standard() for the
    modularity.
9  """
10
11 import deck as d
12 import hands as h
13
14 ATTEMPTS = 10000
15
16 def first_row():
17     """
18     Outputs the first row of the table
19
20     :return:
21     """
22     print('{ } {:>7} {:^7} {:>9} {:^7} {:>9} {:^7} {:>11} {:^7}
23           }'.format('# of hands', 'pairs', '%', '2 pairs', '%',
24                     'flushes', '%', 'high card', '%'))
25
26 def get_percentage(number, total_attempt):
27     """
28     Gets percentage with the given number and total attempt
29
30     :param number: an integer for the division
31     :param total_attempt: an integer for the division
32     :return: an integer converted into a percentage
33     """
34     return number / total_attempt * 100
35
36 def get_result(total_attempt):
```

```
35     """
36     Gets each total number of flush, two pair, pair, and high
    card and computes each percentage
37
38     :return: a list of total numbers and percentages
39     """
40     total_flush = 0
41     total_two_pair = 0
42     total_pair = 0
43     total_high_card = 0
44
45     deck = d.create_standard()
46     d.random.shuffle(deck)
47
48     for i in range(total_attempt):
49         if d.cards_remaining(deck) < 5:
50             deck = d.create_standard()
51             d.shuffle(deck)
52
53         hands = h.create_hands(deck)
54
55         if h.is_flush(hands):
56             total_flush += 1
57
58         elif h.is_two_pair(hands):
59             total_two_pair += 1
60
61         elif h.is_pair(hands):
62             total_pair += 1
63
64         else:
65             total_high_card += 1
66
67     return [total_attempt, total_pair, get_percentage(
    total_pair, total_attempt), total_two_pair,
68             get_percentage(total_two_pair, total_attempt),
    total_flush, get_percentage(total_flush, total_attempt),
69             total_high_card, get_percentage(total_high_card,
    total_attempt)]
70
71 def get_content(total_result, content_order):
```

```

72     """
73     Gets content from the given total result
74
75     :param total_result: a list of total numbers and
percentages
76     :param content_order: an integer for the index of
specific content
77     :return: an integer or float for the specific content
78     """
79     return total_result[content_order]
80
81 def output_table(total_result):
82     """
83     Outputs the table with total numbers and percentages
84
85     :param total_list: a list of total numbers
86     :return:
87     """
88     print('{:>10}, {:>7} {: 06.2f} {:>10} {: 06.2f} {:>10}
90     {:>12} {: 06.2f}'.
91     format(get_content(total_result, 0), get_content(
total_result, 1), get_content(total_result, 2),
92     get_content(total_result, 3), get_content(
total_result, 4), get_content(total_result, 5),
93     get_content(total_result, 6), get_content(
total_result, 7), get_content(total_result, 8)))
94
95 if __name__ == "__main__":
96     first_row()
97     output_table(get_result(ATTEMPTS))

```

```
1 """
2 A simple poker game
3
4 :author: Chris Hegang Kim
5 :note: I affirm that I have carried out the attached academic
6       endeavors with full academic honesty,
7       in accordance with the Union College Honor Code and the course
8       syllabus.
9
10      :reflection: Imported a file called table for the modularity.
11      Used named constants like ATTEMPTS because it is convenient to
12      change their values throughout the code.
13      Used helper functions like t.first_row() for the modularity.
14 """
15
16 import table as t
17
18 ATTEMPTS = 10000
19 MAX_COLUMNS = 10
20
21 def play_rounds():
22     """
23     Starts the entire program running and prints the output
24     table
25
26     :return:
27     """
28     total_attempt = 0
29
30     t.first_row()
31
32     for i in range (MAX_COLUMNS):
33         total_attempt += ATTEMPTS
34
35         t.output_table(t.get_result(total_attempt))
36
37 if __name__ == "__main__":
38     play_rounds()
```