```
1 """
2 This file holds any constants that may need to be used
   throughout the entire file.
3
4 Author: Nick DeBaise
5 """
6
7 suits = ["H", "D", "S", "C"]
```

```
"""
Simulate games of poker to track different statistics on likelihood of hands

Author: Nick DeBaise
Note: I affirm that I have carried out the attached academic endeavors with full academic honesty,
        in accordance with the Union College Honor Code and the course syllabus.
Note on refactor:
    I gave a lot of thought to functions, behavior, and implementation before writing them which
    ensured that I did not have to refactor a significant portion of my code. With that being said,
    some functions required some refactoring due to too much logic in them or poor initial assessment
    of what the function should do. These are noted at the top of each file in a comment.
"""
import deck_utilities as deck
import display_utilities as display_utils
import poker_utilities as poker_utils


def play_rounds():
    cards = deck.get_deck()
    cards = deck.shuffle(cards)

    display_utils.print_headers()

    for j in range(1, 11):

        num_pairs = 0
        num_2_pairs = 0
        num_flushes = 0
        num_high_cards = 0

        for i in range(0, j * 10000):
```

```
32              if len(cards) ⩽ 5:
33                  cards = deck.get_deck()
34                  cards = deck.shuffle(cards)
35
36              hand = deck.deal_hand(cards)
37
38              if poker_utils.classify_flush(hand):
39                  num_flushes += 1
40              elif poker_utils.is_two_pair(hand):
41                  num_2_pairs += 1
42              elif poker_utils.is_pair(hand):
43                  num_pairs += 1
44              else:
45                  num_high_cards += 1
46
47          display_utils.print_row(display_utils.
    get_display_data(num_pairs, num_2_pairs, num_flushes,
    num_high_cards, j * 10000))
48
49
50  if __name__ == '__main__':
51      play_rounds()
52
```

```
"""
Functions for creating, getting, and printing a card

Author: Nick DeBaise
Note on Refactor:
    I did not have to refactor this file. I gave the initial storing of the card some thought and decided a tuple
    was the best option. This allowed me easy access when creating the getter methods, and it also
    was an easy implementation. I think this made the functions very easy to write. It may have been more
    readable to use a dictionary and store the card like {"value": value, "suit": suit} but it could have added
    more complexity.
"""
from constants import suits


def create(value, suit):
    """
    Given a value and a suit, return a representation of a card
    containing the value and the suit
    :param value: number (integer) of the card (1-13)
    :param suit: suit (integer) of the card (1-4)
    :return: a representation of a card with a value & suit
    """

    return (value, suit)


def get_value(card):
    """
    Given a card, return the number value (1-13) of that card
```

```
30      :param card: the card object with value & suit
31      :return: the number (integer) of the value of the
    card
32      """
33
34      return card[0]
35
36
37  def get_suit(card):
38      """
39      Given a card, return the suit as a number (1-4) of
     that card
40      :param card: the card object with value & suit
41      :return: the number (integer) of the suit of the
    card
42      """
43
44      return card[1]
45
46
47  def as_str(card):
48      """
49      Given a card, return a print-ready string
50      :param card: the card object with value & suit
51      :return: a string → the prettified version of the
     card object
52      """
53
54      return "[{0}{1}]".format(card[0], suits[card[1] -
    1])
55
56
57  if __name__ == "__main__":
58      print(create(13, 3))
59      print(as_str(create(10, 2)))
60
```

```
"""
Functions for creating, shuffling, dealing, sorting, and printing a deck/hands in a deck
Author: Nick DeBaise

Note on refactor:
    The only functions I refactored were put_in_dict and sort_by_value. I moved these funcs into this file
    as I thought that these two functions only handled behaviour based on a deck/hand of cards (list)
    They were better suited for this file.
"""
import random

import card_utilities as cards


def put_in_dict(hand):
    """
    Given a list of cards (usually a standard 5 card hand), put all items into a dict with the keys as the values of the
    cards and the values being the num of cards in the hand that match that value
    :param hand: poker hand (list of cards)
    :return: dict with keys as card values and values as number of that card in hand
    """

    dict = {}

    for card in hand:
        val = str(cards.get_value(card))

        if val in dict.keys():
            dict[val] += 1
        else:
            dict[val] = 1
```

```
32
33      return dict
34
35
36  def sort_by_value(hand):
37      """
38      Given a hand/deck (list of cards), sort it by
    increasing value (1 → 13), disregarding suit
39      :param hand: the hand or deck (list) of cards
40      :return: nothing
41      """
42
43      hand.sort(key=lambda card: cards.get_value(card))
44
45
46  def get_deck():
47      """
48      Create and return a list of 52 standard playing
    cards
49      :return: a list of cards
50      """
51
52      deck = []
53
54      for i in range(1, 5):
55          for j in range(1, 14):
56              card = cards.create(j, i)
57              deck.append(card)
58      return deck
59
60
61  def shuffle(deck):
62      """
63      Given a deck of cards, randomize the order and
    return it
64      :param deck: a list of cards
65      :return: the newly randomized list
66      """
```

```
67
68      for i in range(0, 52):
69          ind = random.randint(0, 51)
70
71          el1 = deck.pop(i)
72          el2 = deck.pop(ind - 1)
73
74          deck.insert(i, el2)
75          deck.insert(ind, el1)
76      return deck
77
78
79  def deal(deck):
80      """
81      Given a deck, deal one card from the top (removes
     it from the deck) and return the card
82      :param deck: the deck (list) of cards
83      :return: a card
84      """
85
86      return deck.pop(random.randint(0, len(deck) - 1))
87
88
89  def deal_hand(deck, num_of_cards_in_hand=5):
90      """
91      given a deck and a number of cards, deal the num
    of cards off the top of the deck and return that new
    list
92      :param deck: the list of cards to be used to deal
93      :param num_of_cards_in_hand: the number of cards
    to deal
94      :return: a list of the cards from the top of the
    deck
95      """
96
97      hand = []
98      for _ in range(0, num_of_cards_in_hand):
99          hand.append(deal(deck))
```

```
100
101     return hand
102
103
104 def print_as_str(deck):
105     """
106     Given a list of cards, return the list in a print
    -ready string format
107     :param deck: the list of cards
108     :return: a prettified print-ready string
109     """
110
111     return " | ".join([cards.as_str(card) for card in
     deck])
112
113
114 if __name__ == "__main__":
115     deck = get_deck()
116
117     print(shuffle(deck))
118     print(print_as_str(deck))
119
120     for i in range(0, 52):
121         print(deal(deck))
122     print(len(deck))
123
124     deck2 = get_deck()
125
126     print(deal_hand(deck2))
127
```

```
"""
Functions for classifying poker hands
Author: Nick DeBaise

Note on refactor:
    One of my functions (is_continuous_hand) contained logic for sorting, storing, and classifying
    a flush. I decided to split that logic up into 3 different funcs, which proved helpful in the future.
    The three new funcs are is_continuous_hand, put_in_dict (deck_utilities) and sort_by_value (deck_utilities)
    These three funcs were useful as I reused them in other classification functions even though I wrote those
    after refactoring. Initially, I had the 3 funcs all in poker_utils but thought that deck_utilities
    would be a better place for put_in_dict and sort_by_value, so I refactored those out of this file.
"""
import card_utilities as card_utils
import deck_utilities as deck_utils


def is_pair(hand):
    """
    Given a standard 5 card poker hand, return whether there is a pair or three of a kind
    :param hand: Standard 5 card poker hand
    :return: True if there is one of those pairs, False if not
    """

    dict = deck_utils.put_in_dict(hand)

    num_pairs = 0
    is_three_kind = False

```

```
29      for key in dict.keys():
30          if dict[key] == 2:
31              num_pairs += 1
32
33          if dict[key] == 3:
34              is_three_kind = True
35
36      return is_three_kind or num_pairs == 1
37
38
39  def is_two_pair(hand):
40      """
41      Given a standard 5 card poker hand, return whether
     there is a 2 pair, 4 of a kind, or full house
42      :param hand: Standard 5 card poker hand
43      :return: True if there is one of those pairs,
    False if not
44      """
45
46      dict = deck_utils.put_in_dict(hand)
47
48      num_pairs = 0
49      is_four_kind = False
50      is_three_kind = False
51
52      for key in dict.keys():
53          if dict[key] == 2:
54              num_pairs += 1
55
56          if dict[key] == 4:
57              is_four_kind = True
58
59          if dict[key] == 3:
60              is_three_kind = True
61
62      return (is_three_kind and num_pairs == 1) or
    num_pairs >= 2 or is_four_kind
63
```

```
64
65 def classify_flush(hand):
66     """
67     Given a hand, classify it as a flush or not a
   flush
68     :param hand:
69     :return:
70     """
71
72     possible_flush = is_continuous_hand(hand)
73     if possible_flush is None:
74         return False
75     return "flush" in possible_flush
76
77
78 def is_continuous_hand(hand):
79     """
80     Given a standard 5 card poker hand, classify it
   if there are continuous cards
81     :param hand: a standard 5 card poker hand
82     :return: "straight" "flush" "royal flush" "
   straight flush" or None
83     """
84
85     # sort the list by value to make it easier
86     deck_utils.sort_by_value(hand)
87
88     is_same_suit = True
89     possible_flush = True
90
91     suit = card_utils.get_suit(hand[0])
92     val = card_utils.get_value(hand[0])
93
94     for i in range(1, len(hand)):
95         card = hand[i]
96         card_val = card_utils.get_value(card)
97         card_suit = card_utils.get_suit(card)
98
```

```
 99          # check royal flush
100          if suit ≠ card_suit:
101              is_same_suit = False
102
103          if card_val ≠ val + 1:
104              # it's possible it could be royal flush (
    ace) and ace is seen as value = 1
105              if not (card_val == 10 and val == 1):
106                  possible_flush = False
107          val = card_val
108
109      if possible_flush and is_same_suit and card_utils
    .get_value(hand[4]) == 13:
110          return "royal flush"
111      elif is_same_suit and possible_flush:
112          return "straight flush"
113      elif is_same_suit and not possible_flush:
114          return "flush"
115      elif possible_flush:
116          return "straight"
117      else:
118          return None
119
120
121 if __name__ == "__main__":
122      deck = deck_utils.shuffle(deck_utils.get_deck())
123      hand = deck_utils.deal_hand(deck)
124
125      print(hand)
126
127      print(classify_flush(hand))
128      print(classify_flush([(1, 1), (2, 1), (3, 1), (4
    , 1), (5, 1)]))  # true
129      print(classify_flush([(1, 1), (12, 1), (11, 1), (
    10, 1), (13, 1)]))  # true
130      print(classify_flush([(5, 1), (12, 1), (9, 1), (
    10, 1), (3, 1)]))  # true
131      print(classify_flush([(5, 1), (3, 3), (6, 2), (7
```

```
131 , 1), (4, 1)]))  # false
132     print("----")
133     print(is_two_pair(hand))
134     print(is_two_pair([(1, 1), (1, 2), (1, 3), (1, 4
    ), (4, 2)]))  # true
135     print(is_two_pair([(2, 1), (2, 2), (1, 3), (1, 4
    ), (4, 2)]))  # true
136     print(is_two_pair([(1, 1), (4, 2), (1, 3), (5, 4
    ), (9, 2)]))  # false
137     print("----")
138     print(is_pair(hand))
139     print(is_pair([(1, 1), (1, 2), (1, 3), (1, 4), (4
    , 2)]))  # false
140     print(is_pair([(2, 1), (2, 2), (1, 3), (1, 4), (4
    , 2)]))  # false
141     print(is_pair([(1, 1), (4, 2), (1, 3), (5, 4), (9
    , 2)]))  # true
142
```

```
"""
Functions for printing card data in a table
Author: Nick DeBaise

Note on refactor:
    I attempted to modularize many functions such as preparing the data, printing headers, printing rows
    to ensure that, if in the future, more columns are added, data changes, headers change, etc, it becomes
    easier to make those changes.
    I think I could have modularized the print_row function a bit more by printing the pairs in the row
    (eg. the number of cards paired with the percentage of those cards) to allow for more columns to be added easier
    but, it also could add more complexity right now considering it may not be used in the future.

"""


def get_display_data(num_pairs, num_2_pairs, num_flushes, num_high_cards, number_items):
    """
    Given card data, return a list with 9 items, each containing the appropriate data for the table
    :param num_pairs: the number of pairs/three of a kind in the sequence
    :param num_2_pairs: the number of 2 pairs, four of a kind, & full houses in the sequence
    :param num_flushes: the number of flushes (normal, royal, straight) in the sequence
    :param num_high_cards: the number of high card/straights in the sequence
    :param number_items: the number of times that the poker game was simulated
    :return: list of card data in order with percentages
```

```
25      """
26
27      return [number_items, num_pairs, num_pairs /
    number_items * 100, num_2_pairs, num_2_pairs /
    number_items * 100,
28              num_flushes, num_flushes / number_items *
    100, num_high_cards, num_high_cards / number_items *
    100]
29
30
31  def print_headers():
32      """
33      Print the header text in a designated format
34      """
35
36      print("{:>10}  {:>8}  {:^5}  {:>10}  {:^5}  {:>10
    }  {:^5}  {:>10}  {:^5}".format(
37          "# of hands", "pairs", "%", "2 pairs", "%", "
    flushes", "%", "high card", "%"
38      ))
39
40
41  def print_row(data):
42      """
43      Given a list of data for the row, print it in a
    designated format
44      :param data: the list (length = 9) of data for
    cards
45      """
46
47      print("{:>10,d}  {:>8}  {:0>5.2f}  {:>10}  {:0>5.
    2f}  {:>10}  {:0>5.2f}  {:>10}  {:0>5.2f}".format(
48          data[0], data[1], data[2], data[3], data[4],
    data[5], data[6], data[7], data[8]
49      ))
50
51
52  if __name__ == "__main__":
```

```
53      print_headers()
54      print_row([10000, 5061, 51.92231, 4800, 47.02, 451
    , 2.001, 21, 0.04])
55      print_row([50000, 5061, 51.92231, 4800, 47.02, 451
    , 2.001, 21, 0.04])
56      print_row([10000, 5061, 51.92231, 4800, 47.02, 451
    , 2.001, 21, 0.04])
57      print_row([10000, 5061, 51.92231, 4800, 47.02, 451
    , 2.001, 21, 0.04])
58      print_row([100000, 5061, 51.92231, 4800, 47.02,
    451, 2.001, 21, 0.04])
59
```