

```
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"]
```

```
1  """
2  Simulate games of poker to track different statistics
   on likelihood of hands
3
4  Author: Nick DeBaise
5  Note: I affirm that I have carried out the attached
   academic endeavors with full academic honesty,
6      in accordance with the Union College Honor
   Code and the course syllabus.
7  Note on refactor:
8      I gave a lot of thought to functions, behavior,
   and implementation before writing them which
9      ensured that I did not have to refactor a
   significant portion of my code. With that being said,
10     some functions required some refactoring due to
   too much logic in them or poor initial assessment
11     of what the function should do. These are noted at
   the top of each file in a comment.
12 """
13 import deck_utilities as deck
14 import display_utilities as display_utils
15 import poker_utilities as poker_utils
16
17
18 def play_rounds():
19     cards = deck.get_deck()
20     cards = deck.shuffle(cards)
21
22     display_utils.print_headers()
23
24     for j in range(1, 11):
25
26         num_pairs = 0
27         num_2_pairs = 0
28         num_flushes = 0
29         num_high_cards = 0
30
31         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.
48             get_display_data(num_pairs, num_2_pairs, num_flushes,
49             num_high_cards, j * 10000))
50
51 if __name__ == '__main__':
52     play_rounds()
```

```
1  """
2  Functions for creating, getting, and printing a card
3
4  Author: Nick DeBaise
5  Note on Refactor:
6      I did not have to refactor this file. I gave the
       initial storing of the card some thought and decided a
       tuple
7      was the best option. This allowed me easy access
       when creating the getter methods, and it also
8      was an easy implementation. I think this made the
       functions very easy to write. It may have been more
9      readable to use a dictionary and store the card
       like {"value": value, "suit": suit} but it could have
       added
10     more complexity.
11 """
12 from constants import suits
13
14
15 def create(value, suit):
16     """
17     Given a value and a suit, return a representation
       of a card
18     containing the value and the suit
19     :param value: number (integer) of the card (1-13)
20     :param suit: suit (integer) of the card (1-4)
21     :return: a representation of a card with a value
       & suit
22     """
23
24     return (value, suit)
25
26
27 def get_value(card):
28     """
29     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
32     card
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
40     that card
41     :param card: the card object with value & suit
42     :return: the number (integer) of the suit of the
43     card
44     """
45
46     return card[1]
47
48
49 def as_str(card):
50     """
51     Given a card, return a print-ready string
52     :param card: the card object with value & suit
53     :return: a string → the prettified version of the
54     card object
55     """
56
57     return "[{0}{1}].format(card[0], suits[card[1] -
58 1])
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
1  """
2  Functions for creating, shuffling, dealing, sorting,
   and printing a deck/hands in a deck
3  Author: Nick DeBaise
4
5  Note on refactor:
6      The only functions I refactored were put_in_dict
   and sort_by_value. I moved these funcs into this file
7      as I thought that these two functions only handled
   behaviour based on a deck/hand of cards (list)
8      They were better suited for this file.
9  """
10 import random
11
12 import card_utilities as cards
13
14
15 def put_in_dict(hand):
16     """
17     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
18     cards and the values being the num of cards in the
   hand that match that value
19     :param hand: poker hand (list of cards)
20     :return: dict with keys as card values and values
   as number of that card in hand
21     """
22
23     dict = {}
24
25     for card in hand:
26         val = str(cards.get_value(card))
27
28         if val in dict.keys():
29             dict[val] += 1
30         else:
31             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
39     increasing value (1 → 13), disregarding suit
40     :param hand: the hand or deck (list) of cards
41     :return: nothing
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
49     cards
50     :return: a list of cards
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
64     return it
65     :param deck: a list of cards
66     :return: the newly randomized list
67     """
```

```
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
82     it from the deck) and return the card
83     :param deck: the deck (list) of cards
84     :return: a card
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
92     of cards off the top of the deck and return that new
93     list
94     :param deck: the list of cards to be used to deal
95     :param num_of_cards_in_hand: the number of cards
96     to deal
97     :return: a list of the cards from the top of the
98     deck
99     """
100
101     hand = []
102     for _ in range(0, num_of_cards_in_hand):
103         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
107     -ready string format
108     :param deck: the list of cards
109     :return: a prettified print-ready string
110     """
111     return " | ".join([cards.as_str(card) for card in
112                         deck])
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
```

```
1  """
2  Functions for classifying poker hands
3  Author: Nick DeBaise
4
5  Note on refactor:
6      One of my functions (is_continuous_hand) contained
7      logic for sorting, storing, and classifying
8      a flush. I decided to split that logic up into 3
9      different funcs, which proved helpful in the future.
10     The three new funcs are is_continuous_hand,
11     put_in_dict (deck_utilities) and sort_by_value (
12     deck_utilities)
13     These three funcs were useful as I reused them in
14     other classification functions even though I wrote
15     those
16     after refactoring. Initially, I had the 3 funcs
17     all in poker_utils but thought that deck_utilities
18     would be a better place for put_in_dict and
19     sort_by_value, so I refactored those out of this file.
20 """
21 import card_utilities as card_utils
22 import deck_utilities as deck_utils
23
24
25 def is_pair(hand):
26     """
27     Given a standard 5 card poker hand, return whether
28     there is a pair or three of a kind
29     :param hand: Standard 5 card poker hand
30     :return: True if there is one of those pairs,
31     False if not
32     """
33
34     dict = deck_utils.put_in_dict(hand)
35
36     num_pairs = 0
37     is_three_kind = False
38
```

```
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
42     there is a 2 pair, 4 of a kind, or full house
43     :param hand: Standard 5 card poker hand
44     :return: True if there is one of those pairs,
45             False if not
46     """
47
48     dict = deck_utils.put_in_dict(hand)
49
50     num_pairs = 0
51     is_four_kind = False
52     is_three_kind = False
53
54     for key in dict.keys():
55         if dict[key] == 2:
56             num_pairs += 1
57
58         if dict[key] == 4:
59             is_four_kind = True
60
61         if dict[key] == 3:
62             is_three_kind = True
63
64     return (is_three_kind and num_pairs == 1) or
65           num_pairs ≥ 2 or is_four_kind
```

```
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
```

```
1  """
2  Functions for printing card data in a table
3  Author: Nick DeBaise
4
5  Note on refactor:
6      I attempted to modularize many functions such as
7      preparing the data, printing headers, printing rows
8      to ensure that, if in the future, more columns are
9      added, data changes, headers change, etc, it becomes
10     easier to make those changes.
11     I think I could have modularized the print_row
12     function a bit more by printing the pairs in the row
13     (eg. the number of cards paired with the
14     percentage of those cards) to allow for more columns
15     to be added easier
16     but, it also could add more complexity right now
17     considering it may not be used in the future.
18
19 """
20
21
22 def get_display_data(num_pairs, num_2_pairs,
23                     num_flushes, num_high_cards, number_items):
24     """
25     Given card data, return a list with 9 items, each
26     containing the appropriate data for the table
27     :param num_pairs: the number of pairs/three of a
28     kind in the sequence
29     :param num_2_pairs: the number of 2 pairs, four of
30     a kind, & full houses in the sequence
31     :param num_flushes: the number of flushes (normal
32     , royal, straight) in the sequence
33     :param num_high_cards: the number of high card/
34     straights in the sequence
35     :param number_items: the number of times that the
36     poker game was simulated
37     :return: list of card data in order with
38     percentages
```

```

25         """
26
27     return [number_items, num_pairs, num_pairs /
28             number_items * 100, num_2_pairs, num_2_pairs /
29             number_items * 100,
30             num_flushes, num_flushes / number_items *
31             100, num_high_cards, num_high_cards / number_items *
32             100]
33
34 def print_headers():
35     """
36     Print the header text in a designated format
37     """
38
39     print("{:>10}   {:>8}   {:^5}   {:>10}   {:^5}   {:>10}
40           {:^5}   {:>10}   {:^5}".format(
41         "# of hands", "pairs", "%", "2 pairs", "%", "
42         flushes", "%", "high card", "%"
43     ))
44
45 def print_row(data):
46     """
47     Given a list of data for the row, print it in a
48     designated format
49     :param data: the list (length = 9) of data for
50     cards
51     """
52
53     print("{:>10,d}   {:>8}   {:0>5.2f}   {:>10}   {:0>5.
54           2f}   {:>10}   {:0>5.2f}   {:>10}   {:0>5.2f}".format(
55         data[0], data[1], data[2], data[3], data[4],
56         data[5], data[6], data[7], data[8]
57     ))
58
59 if __name__ == "__main__":

```



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