```
#################################################
######################
# Computer project 06
# play poker game of simplified texas poker
# find the highest catergory without consider
the value
# Only for two players
#################################################
######################

import cards

rank_list = ["High card", "one pair", "two
pairs", "three of a kind", "straight",
"flush",
"full house", "four of a kind", "straight
flush"]


def less_than(c1,c2):
    '''Return
           True if c1 is smaller in rank,
           True if ranks are equal and c1 has
a 'smaller' suit
           False otherwise'''
    if c1.rank() < c2.rank():
        return True
    elif c1.rank() == c2.rank() and c1.suit()
< c2.suit():
        return True
    return False

def min_in_list(L):
    '''Return the index of the mininmum card
in L'''
    min_card = L[0]  # first card
    min_index = 0
    for i,c in enumerate(L):
        if less_than(c,min_card):  # found a
smaller card, c
            min_card = c
            min_index = i
    return min_index

def cannonical(H):
    ''' Selection Sort: find smallest and swap
with first in H,
        then find second smallest (smallest of
rest) and swap with second in H,
        and so on...'''
    for i,c in enumerate(H):
        # get smallest of rest; +i to account
for indexing within slice
        min_index = min_in_list(H[i:]) + i
        H[i], H[min_index] = H[min_index], c
# swap

    return H

def flush_7(H):
    '''Return a list of 5 cards forming a
flush,
       if at least 5 of 7 cards form a flush
in H, a list of 7 cards,
       False otherwise.'''
    color = []
    for i,c in enumerate(H):
        color.append(c.suit())

    return_list = []
    for s in color:

        if color.count(s) >= 5:
            for c in H:
                if c.suit() == s:
                    return_list.append(c)   #
add to return list
                if len(return_list) == 5:   #
if there collected all five cards, halt for
loop
                    break
        if len(return_list) == 5:
            break
    if return_list:

        return return_list
    else:
        return False



def straight_7(H):
    '''Return a list of 5 cards forming a
straight,
       if at least 5 of 7 cards form a
straight in H, a list of 7 cards,
       False otherwise.'''

    H = cannonical(H)

    return_list = []
    returned_list = []

    number  = []
    for i,c in enumerate(H):
        number.append(c.rank())
    number.sort()
    min_rank = min(number)
    while True:
        # add to return list
        if min_rank in number and (min_rank +
1) in number and (min_rank + 2) in number and
(min_rank + 3) in number and (min_rank + 4) in
number :
            for h in H:
                if h.rank() == min_rank and
h.rank() not in returned_list:
                    return_list.append(h)

returned_list.append(h.rank())
                if h.rank() == min_rank + 1
and h.rank() not in returned_list:
                    return_list.append(h)

returned_list.append(h.rank())
                if h.rank() == min_rank + 2
and h.rank() not in returned_list:
                    return_list.append(h)

returned_list.append(h.rank())
                if h.rank() == min_rank + 3
and h.rank() not in returned_list:
                    return_list.append(h)

returned_list.append(h.rank())
                if h.rank() == min_rank + 4
and h.rank() not in returned_list:
                    return_list.append(h)

returned_list.append(h.rank())
        if min_rank + 4 == max(number) or
len(return_list) == 5:
            break
        else:
            min_rank += 1





    if return_list:

        return return_list
    else:
        return False

def straight_flush_7(H):
    '''Return a list of 5 cards forming a
straight flush,
       if at least 5 of 7 cards form a
straight flush in H, a list of 7 cards,
       False otherwise.'''
    if straight_7(H) == flush_7(H) :
        return flush_7(H)
    else:
        return False

def four_7(H):
    '''Return a list of 4 cards with the same
rank,
       if 4 of the 7 cards have the same rank
in H, a list of 7 cards,
       False otherwise.'''
    number = []
    for i,c in enumerate(H):
        number.append(c.rank())        # get
the all possible rank in the H

    return_list = []
    for s in number:

        if number.count(s) == 4:
            for c in H:
                if c.rank() == s:
                    return_list.append(c)   #
add to return list
                if len(return_list) == 4:
                    break
        if len(return_list) == 4:
            break
    if return_list:

        return return_list
    else:
        return False

def three_7(H):
    '''Return a list of 3 cards with the same
rank,
       if 3 of the 7 cards have the same rank
in H, a list of 7 cards,
       False otherwise.
       You may assume that four_7(H) is
False.'''
    color = []
    for i,c in enumerate(H):
        color.append(c.rank())

    return_list = []
    for s in color:

        if color.count(s) == 3:
            for c in H:
                if c.rank() == s:
                    return_list.append(c)  #
add to return list
                if len(return_list) == 3:
                    break
        if len(return_list) == 3:    # after
add all three card, stop iterate the list
            break
    if return_list:

        return return_list
    else:
        return False

def two_pair_7(H):
    '''Return a list of 4 cards that form 2
pairs,
       if there exist two pairs in H, a list
of 7 cards,
       False otherwise.
       You may assume that four_7(H) and
three_7(H) are both False.'''
    dictionary_for_value = {}
    for i in H:
        if i.rank() not in
dictionary_for_value:
            dictionary_for_value[i.rank()] = 1
        else:
            dictionary_for_value[i.rank()] +=
1

    n_list = []
    for d,v in dictionary_for_value.items():
        if v == 2:
            n_list.append(d)    # for pair,
add twice of same value
            n_list.append(d)
    return_list = []
    if len(n_list) == 4:
        i = 0
        for n in n_list:
            for h in H:
                if h.rank() == n and h not in
return_list:
                    return_list.append(h)   #
add to return list

        return return_list
    else:
        return False





def one_pair_7(H):
    '''Return a list of 2 cards that form a
pair,
       if there exists exactly one pair in H,
a list of 7 cards,
       False otherwise.
       You may assume that four_7(H),
three_7(H) and two_pair(H) are False.'''
    dictionary_for_value = {}
    for i in H:
        if i.rank() not in
dictionary_for_value:      # for the rank
shows at the first time
            dictionary_for_value[i.rank()] = 1
        else:
            dictionary_for_value[i.rank()] +=
1
    n_list = []
    for d,v in dictionary_for_value.items():
        if v == 2:
            n_list.append(d)
            n_list.append(d)
    return_list = []
    if len(n_list) == 2:
        i = 0
        for h in H:
            if h.rank() == n_list[i]:
                i+=1
                return_list.append(h)   # add
to return list
            if i == 2:
                break

        return return_list
    else:
        return False

def full_house_7(H):
    '''Return a list of 5 cards forming a full
house,
       if 5 of the 7 cards form a full house
in H, a list of 7 cards,
       False otherwise.
       You may assume that four_7(H) is
False.'''
    H_copy = H.copy()   # create a shallow
copy
    color = []
    for i,c in enumerate(H_copy):
        color.append(c.rank())

    return_list = []
    for s in color:

        if color.count(s) == 3:
            for c in H_copy:
                if c.rank() == s:
                    return_list.append(c) #
add to return list
                    H_copy.remove(c)      #
remove the added card from the shallow copy
                if len(return_list) == 3:
                    break
        if len(return_list) == 3:
            break


    if len(return_list) == 3:

        list2 = []
        for o in H_copy:

            list2.append(o.rank())

        for l in list2:
            if list2.count(l) == 2:
                for h in H_copy:
                    if h.rank() == l:
                        return_list.append(h)
# add to return list
                    if len(return_list) == 5:
                        break
            if len(return_list) == 5:
                        break
    if len(return_list) == 5:
        return return_list
    else:
        return False





def main():
    D = cards.Deck()
    D.shuffle()

    while True:
#        if len(D) < 9:
#            print("Deck has too few cards so
game is done.")
#            break
        community_list = []
        for i in range(5):
            community_list.append(D.deal()) #
create community cards

        hand_1_list = []
        hand_2_list = []
        for i in range(2):
            hand_1_list.append(D.deal())
        for i in range(2):
            hand_2_list.append(D.deal())
        # create Player 1 hand
        # create Player 2 hand
        game_hand1 = hand_1_list +
community_list # get 7 cards list
#        print('Game hand1',game_hand1)
        game_hand2 = hand_2_list +
community_list # get 7 cards list
#        print('Game hand2',game_hand2)

        # These are the all indetify for the
two gamne_hand's categories.
        # By calling function by if elif. Try
to get the highest return list
        rank_level_hand1 = 10
        rank_level_hand2 = 10
        if straight_flush_7(game_hand1) :
            return_list1 =
straight_flush_7(game_hand1)
            rank_level_hand1 = 9
        elif four_7(game_hand1):
            return_list1 = four_7(game_hand1)
            rank_level_hand1 = 8
        elif full_house_7(game_hand1):
            return_list1 =
full_house_7(game_hand1)
            rank_level_hand1 = 7
        elif flush_7(game_hand1):
            return_list1 = flush_7(game_hand1)
            rank_level_hand1 = 6
        elif straight_7(game_hand1):
            return_list1 =
straight_7(game_hand1)
            rank_level_hand1 = 5
        elif three_7(game_hand1):
            return_list1 = three_7(game_hand1)
            rank_level_hand1 = 4
        elif two_pair_7(game_hand1):
            return_list1 =
two_pair_7(game_hand1)
            rank_level_hand1 = 3
        elif one_pair_7(game_hand1):
            return_list1 =
one_pair_7(game_hand1)
            rank_level_hand1 = 2
        else:
            rank_level_hand1 = 1
        # These are the all indetify for the
two gamne_hand's categories.
        # By calling function by if elif. Try
to get the highest return list
        if straight_flush_7(game_hand2) :
            return_list2 =
straight_flush_7(game_hand2)
            rank_level_hand2 = 9
        elif four_7(game_hand2):
            return_list2 = four_7(game_hand2)
            rank_level_hand2 = 8
        elif full_house_7(game_hand2):
            return_list2 =
full_house_7(game_hand2)
            rank_level_hand2 = 7
        elif flush_7(game_hand2):
            return_list2 = flush_7(game_hand2)
            rank_level_hand2 = 6
        elif straight_7(game_hand2):
            return_list2 =
straight_7(game_hand2)
            rank_level_hand2 = 5
        elif three_7(game_hand2):
            return_list2 = three_7(game_hand2)
            rank_level_hand2 = 4
        elif two_pair_7(game_hand2):
            return_list2 =
two_pair_7(game_hand2)
            rank_level_hand2 = 3
        elif one_pair_7(game_hand2):
            return_list2 =
one_pair_7(game_hand2)
            rank_level_hand2 = 2
        else:
            rank_level_hand2 = 1

        # show hands
        print("-"*40)
        print("Let's play poker!\n")
        print("Community
cards:",community_list)

        print("Player 1:",hand_1_list)
        print("Player 2:",hand_2_list)
        print()

        # Comparing who is higher or they are
in the same category
        if rank_level_hand1 ==
rank_level_hand2 == 1:
            print('TIE with high card')
        elif rank_level_hand1 ==
rank_level_hand2:
            print('TIE with
'+str(rank_list[rank_level_hand1 -
1])+':',cannonical(return_list1))
        elif rank_level_hand1 >
rank_level_hand2:
            print('Player 1 wins with
'+str(rank_list[rank_level_hand1 -
1])+':',cannonical(return_list1))
        elif rank_level_hand1 <
rank_level_hand2:
            print('Player 2 wins with
'+str(rank_list[rank_level_hand2 -
1])+':',cannonical(return_list2))

        if len(D) < 9:
            print("Deck has too few cards so
game is done.")
            break

        input_str = input('Do you wish to play
another hand?(Y or N) ')
        if input_str.lower() == 'y':
            pass
        else:
            break



if __name__ == "__main__":
    main()
```