```python
"""
For this project, I first thought of using list of strings as card representation, because I thought that it would save me more time to write but then as i work more into this project
I realized that string isn't really the best choice. I thought of using a dictionary, but that would've been a hassle for me, so the best option I had was to go with tuples.
And I now believe tuples is a better option to go with for this project, it could clearly separate the rank and suits, it is also easy for me access the rank and suits at any time.

One part of the code that required significant refactoring was in hands.py. At first, I tried to write just four functions to handle Flush, Two Pair, One Pair, and High Card.
But this approach quickly became complicated, especially since Flush includes multiple variations like regular Flush, Straight Flush, and Royal Flush. So instead of forcing
those four functions, I wrote a bunch of different functions to identify the hands and evaluate them into these 4 groups.
"""

import random

ranks = ["2","3","4","5","6","7","8","9","10","J","Q","K","A"]
suits = ["Spades","Clubs","Hearts","Diamonds"]

def create():
    """
    Create a list of cards in tuple form and returns them.
    :return: a list of tuples that represent cards
    """
    deck = []
    for rank in ranks:
        for suit in suits:
            deck.append((rank, suit))
    return deck


def shuffle(deck):
    """
    Shuffles the deck.
    :param deck: a list of tuples that represent cards
    """
    random.shuffle(deck)

def deal(deck, n):
    """
    Deals n cards from the deck.
    :param deck: a list of tuples that represent cards
    :param n: the number of cards to deal
    :return: the dealed cards
    """
    hands = []
    for _ in range(n):
        hands.append(deck.pop(0))
    return hands

```

```python
#  Poker Hands
# • Flush (includes normal, royal, and straight flushes)
# • Two pair (includes two pair, four-of-a-kind, and full house)
# • Pair (includes pair and three-of-a-kind)
# • High card (includes high card and straight). Ace has the highest rank and Two has the lowest.

def is_flush(hands):
    """
    Check if a hand is a flush, all cards same suit.
    :param hands: list of 5 cards as tuples (rank, suit)
    :return: a boolean value
    """
    suits = []
    for card in hands:
        suits.append(card[1])
    removed = set(suits)
    return len(removed) == 1


def is_straight(hands):
    """
    Check if a hand is a straight, consecutive cards.
    :param hands: list of 5 cards as tuples (rank, suit)
    :return: a boolean value
    """
    rank_order = {"2": 2, "3": 3, "4": 4, "5": 5, "6": 6, "7": 7, "8": 8, "9": 9, "10": 10, "J": 11, "Q": 12, "K": 13, "A": 14}
    ranks = []
    for card in hands:
        rank = card[0]
        ranks.append(rank_order[rank])

    ranks.sort()
    for i in range(4):
        if ranks[i] + 1 != ranks[i + 1]:
            return False
    return True


def rank_counts(hands):
    """
    Count how many cards of each rank are in the hand.
    :param hands: list of 5 cards as tuples (rank, suit
)
    :return: dictionary with rank counts
    {rank: count}
    {key: value}
    """
    counts = {}
    for card in hands:
        rank = card[0]
        if rank in counts:
            counts[rank] += 1
        else:
            counts[rank] = 1
    return counts


def is_four_of_a_kind(hands):
    """
    Check if a hand has four cards of the same rank.
    :param hands: list of 5 cards as tuples (rank, suit
)
    :return: a boolean value
    """
    counts = rank_counts(hands)
    if 4 in counts.values():
        return True
    return False

```

```python

def is_full_house(hands):
    """
    Check if a hand has three of one rank and two of another.
    :param hands: list of 5 cards as tuples (rank, suit)
    :return: a boolean value
    """
    counts = rank_counts(hands).values()
    three = False
    two = False

    for count in counts:
        if count == 3:
            three = True
        elif count == 2:
            two = True

    return three and two


def is_three_of_a_kind(hands):
    """
    Check if a hand has exactly three cards of the same rank.
    :param hands: list of 5 cards as tuples (rank, suit)
    :return: a boolean value
    """
    counts = rank_counts(hands).values()
    if 3 in counts and not is_full_house(hands):
        return True
    return False


def has_pairs(hands, n):
    """
    Check if a hand has exactly n pairs.

    precondition: n is a positive integer and n <= 2
    :param hands: list of 5 cards as tuples (rank, suit)
    :param n: number of pairs to check
    :return: a boolean value
    """
    counts = rank_counts(hands)
    pair_count = 0
    for value in counts.values():
        if value == 2:
            pair_count += 1
    return pair_count == n


def evaluate(hands):
    """
    Evaluate a hand and return its category.
    Grouped into: flush, two pair, pair, or high card.
    :param hands: list of 5 cards as tuples (rank, suit)
    :return: string representing the hand category
    """
    flush = is_flush(hands)
    straight = is_straight(hands)

    if flush and straight:
        return "flush"
    elif flush:
        return "flush"
    elif is_four_of_a_kind(hands) or is_full_house(hands) or has_pairs(hands, 2):
        return "two pair"
    elif is_three_of_a_kind(hands) or has_pairs(hands, 1):
        return "pair"
    else:
        return "high card"
```

```python
# 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.


import cards
import hands

def play_rounds():
    """
    Play poker simulation rounds and print out results in a table.
    """
    print("# of hands  pairs  %    2 pairs  %   flushes  %   high card  %")
    for i in range(10000, 100001, 10000):
        pair = 0
        two_pair = 0
        flush = 0
        high_card = 0
        hands_dealt = 0

        while hands_dealt < i:
            deck = cards.create()
            cards.shuffle(deck)

            while hands_dealt < i and len(deck) >= 5:
                hand = cards.deal(deck, 5)
                category = hands.evaluate(hand)
                if category == "pair":
                    pair += 1
                elif category == "two pair":
                    two_pair += 1
                elif category == "flush":
                    flush += 1
                elif category == "high card":
                    high_card += 1

                hands_dealt += 1

        total = hands_dealt
        pair_percent = (pair / total) * 100
        two_pair_percent = (two_pair / total) * 100
        flush_percent = (flush / total) * 100
        high_card_percent = (high_card / total) * 100

        print(f"{total:>7,}  {pair:>5}  {pair_percent:05.2f}  {two_pair:>5}  {two_pair_percent:05.2f}  {flush:>5}  {flush_percent:05.2f}  {high_card:>5}  {high_card_percent:05.2f}")

if __name__ == "__main__":
    play_rounds()

```