```python
class card:
    """
    Class for the card object

    """

    def __init__(self, value, suit):
        """
        initializes card object

        :param self: instance of card
        :param value: cards number or face value
        :param suit: suit value of card (heart,diamond, club, or spade)
        :return:
        """
        self.value = value
        self.suit = suit



    def printCard(self):

        print("[" + str(self.value) + " , " + self.suit + "]")

```

```python
import random

from card import card


class deck:
    """
    model of a deck
    """

    def __init__(self):
        """
        initialize deck object
        """
        self.cardList = []
        deck.generate(self)



    def generate(self):
        """
        generates a deck of 52 cards
        :return: none
        """

        for i in range(0,4):
            for j in range(2,15):

                if i == 0: #Heart suit
                    suit = "heart"

                if i == 1: #Diamond suit
                    suit = "diamond"

                if i == 2: #spade suit
                    suit = "spade"

                if i == 3: #club suit
                    suit = "club"


                self.cardList.append(card(j, suit))

        random.shuffle(self.cardList)



    def drawCard(self):
        """
        draws a card from the deck
        :return: card
        """

        if self.cardList.__len__() != 0:
            return self.cardList.pop()

        else:
            self = deck()
            return self.cardList.pop()

    def printDeck(self):
        """
        prints deck of cards
        :return: none
        """

        for card in self.cardList:
            card.printCard()





```

```python
class hand:
    """
    models a poker hand
    """

    def __init__(self):
        self.hand = []

    def deal_poker_hand(self, deck):
        """
        this function adds 5 cards from the deck to the hand

        :param deck: deck that cards are being drawn from
        :return:
        """

        for i in range(5):
            self.hand.append(deck.drawCard())

    @property
    def what_is_it(self):
        """
        evaluates the hand

        :return: value of hand
        """
        pairs = []
        triples = []

        values = sorted([card.value for card in self.hand])
        suits = [card.suit for card in self.hand]

        for v in set(values):
            if values.count(v) == 4:
                return "4 of a kind"
            if values.count(v) == 3:
                triples.append(v)
            if values.count(v) == 2:
                pairs.append(v)

        if all(s == suits[0] for s in suits):
            return "Flush"

        if len(triples) == 1 and len(pairs) == 1:
            return "Full House"

        if len(triples) == 1 and len(pairs) == 0:
            return "3 of a kind"

        if len(pairs) == 2:
            return "2 pair"

        if len(pairs) == 1:
            return "1 pair"
        else:
            return "High card"

    def print_hand(self):
        """
        prints all cards in hand
        :return: none
        """

        for card in self.hand:
            card.printCard()

```

```python
from deck import deck
from hand import hand
from accumulator import accumulator

"""
By: Ian Sulley

Honor Code:
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.

Refactoring:
I eliminated the use of global variables by building an accumulator object that stores and tracks all the statistics you
may want to know about the results of the current test

I eliminated my prints result function by overriding the __str__() property of my accumulator object.
This way just by printing the accumulator you automatically have it in correct formatting with all data fields displayed
"""



def count_hand(accum, my_hand):
    """
    counts instances of each type of hand
    :param my_hand:  hand that is being counted
    :return: none
    """

    hand_value = my_hand.what_is_it

    if hand_value == "Flush":
        accum.flush_count += 1
    if hand_value == "4 of a kind" or hand_value == "Full House" or hand_value == "2 pair":
        accum.two_pair_count += 1
    if hand_value == "1 pair" or hand_value == "3 of a kind":
        accum.pair_count += 1
    else:
        accum.high_card_count += 1


def generate_hand(deck):
    """
    generates a hand from the given deck
    :param deck: deck used to form hand
    :return: hand
    """
    new_hand = hand()
    new_hand.deal_poker_hand(deck)
    return new_hand


def generate_deck():
    """
    creates a new deck
    :return: deck
    """

    new_deck = deck()
    return new_deck


def tester(accum):
    """
    runs necessary processedures to create and count hands
    :param number_of_hands: how many hands you want to create
    :return: none
    """

    hands_made = 0  # keeps track of hands made
    my_deck = generate_deck()

    while hands_made < accum.number_of_hands:

        if (len(my_deck.cardList) >= 5):  # if we can make a full hand then do so

            my_hand = generate_hand(my_deck)
            count_hand(accum, my_hand)
            hands_made += 1  # we can still make hands

        if (len(my_deck.cardList) < 5):  # if the cards in the deck drops below 5

             new_deck = generate_deck()  # generate a new deck

             new_deck.cardList += my_deck.cardList
             my_deck = new_deck


 def main():

     num_of_hands = [10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000]
     print("# of hands     pairs    %      2 pairs      %      flushes    %      high card    %")
     for num in num_of_hands:
         x = accumulator(num)
         tester(x)
         x.calculate_percents()
         print(x)


 if __name__ == '__main__':
     main()

```

```python
class accumulator:
    """
    class for accumulating my results

    Refactoring:
    I added the percentage counts as accumulator properties so it is easier to keep track of them
    and keep them with the test they were computed from
    """

    def __init__(self, number_of_hands):

        self.number_of_hands = number_of_hands

        self.flush_count = 0

        self.two_pair_count = 0

        self.pair_count = 0

        self.high_card_count = 0

        self.flush_percent = 0

        self.two_pair_percent = 0

        self.pair_percent = 0

        self.high_card_percent = 0


    def __str__(self):
        """
        builtin in method so that when you call print on an object it does this
        :return: values of each count and percentage
        """
        x = " "

        if(self.number_of_hands != 100000):
            hand_x = 4 * x
        else:
            hand_x = 3 * x


        return(hand_x + format(self.number_of_hands, ","
) + 5 * x
              + format(self.pair_count, ">5") + 2 * x + format(self.pair_percent, "0>5.2f") + 6 * x
              + format(self.two_pair_count, ">5") + 2 * x + format(self.two_pair_percent, "0>5.2f") + 6 * x
              + format(self.flush_count, ">5") + 2 * x + format(self.flush_percent, "0>5.2f") + 8 * x
              + format(self.high_card_count, ">5") + 2 * x + format(self.high_card_percent, "0>5.2f"))





    def calculate_percents(self):
        """
        calculates the percentage value of each hand type based on number of hands delt
        :return: none
        """


        self.flush_percent = (self.flush_count / self.number_of_hands) * 100

        self.two_pair_percent = (self.two_pair_count / self.number_of_hands) * 100

        self.pair_percent = (self.pair_count / self.number_of_hands) * 100

        self.high_card_percent = (self.high_card_count / self.number_of_hands) * 100

```