# CS212 Unit 1

37

Gundega

# Contents

**1 CS212 - Unit 1: Winning Poker Hands ..... 1/15**
1.1 1. Outlining the Problem ..... 1/15
1.1.1 1.1. Quiz: Representing Hands ..... 2/15
1.2 2. Wild West Poker ..... 2/15
1.2.1 2.1. Quiz: Poker Function ..... 4/15
1.2.2 2.2. Quiz: Understanding Max ..... 4/15
1.2.3 2.3. Quiz: Using Max ..... 4/15
1.2.4 2.4. Quiz: Testing ..... 4/15
1.2.5 2.5. Quiz: Extreme Values ..... 5/15
1.2.6 2.6. Quiz: Hand Rank Attempt ..... 5/15
1.2.7 2.7. Quiz: Representing Rank ..... 5/15
1.3 3. Back to Hand Rank ..... 5/15
1.3.1 3.1. Quiz: Testing Hand Rank ..... 5/15
1.3.2 3.2. Quiz: Writing Hand Rank ..... 5/15
1.3.3 3.3. Quiz: Testing Card Rank ..... 5/15
1.3.4 3.4. Quiz: Fixing Card Rank ..... 5/15
1.3.5 3.5. Quiz: Straight and Flush ..... 5/15
1.3.6 3.6. Quiz: Kind Function ..... 6/15
1.3.7 3.7. Quiz: Two Pair Function ..... 6/15
1.3.8 3.8. Quiz: Making Changes ..... 6/15
1.3.9 3.9. Quiz: What to Change ..... 6/15
1.3.10 3.10. Ace Low Straight ..... 6/15
1.3.11 3.11. Quiz: Handling Ties ..... 6/15
1.3.12 3.12. Quiz: Allmax ..... 6/15
1.4 4. Deal ..... 6/15
1.4.1 4.1. Quiz: Hand Frequencies ..... 6/15
1.5 5. Dimensions of Programming ..... 6/15
1.6 6. Refactoring ..... 6/15
1.7 7. Summary ..... 7/15
1.8 8. Bonus - Shuffling ..... 7/15
1.9 9. Complete Code For Poker Problem ..... 8/15
1.10 10. Complete Code For Homeworks (Warning: Refer this only after submitting homework) ..... 12/15

# 1 CS212 - Unit 1: Winning Poker Hands

Contents


1. Outlining the Problem
    1. Quiz: Representing Hands
2. Wild West Poker
    1. Quiz: Poker Function
    2. Quiz: Understanding Max
    3. Quiz: Using Max
    4. Quiz: Testing
    5. Quiz: Extreme Values
    6. Quiz: Hand Rank Attempt
    7. Quiz: Representing Rank
3. Back to Hand Rank
    1. Quiz: Testing Hand Rank
    2. Quiz: Writing Hand Rank
    3. Quiz: Testing Card Rank
    4. Quiz: Fixing Card Rank
    5. Quiz: Straight and Flush
    6. Quiz: Kind Function
    7. Quiz: Two Pair Function
    8. Quiz: Making Changes
    9. Quiz: What to Change
    10. Ace Low Straight
    11. Quiz: Handling Ties
    12. Quiz: Allmax
4. Deal
    1. Quiz: Hand Frequencies
5. Dimensions of Programming
6. Refactoring
7. Summary
8. Bonus - Shuffling
9. Complete Code For Poker Problem
10. Complete Code For Homeworks (Warning: Refer this only after submitting homework)


## 1.1 1. Outlining the Problem

Unit1- 2

Writing a poker program is an example of a general process with three steps; understand, specify and design. The process involves starting with a vague understanding that you refine into a formal specification of a problem. You then further specify your understanding into something that is amenable to being coded. Then, after the design process you end up with working code.

- Step 1: **Understand**
- Start with a vague understanding that you refine into a problem. In this step you want to take inventory of the concepts you are dealing with. With respect to writing a poker problem, begin with the notion of a "hand," which is five cards. Additionally, note that each card has a "rank" and a "suit."

For example, a five of diamonds card has a rank of five and the suit is diamonds.

Another concept to identify is the "hand rank," which takes a hand and maps to details about the hand.

- Step 2: **Specify**
- Specify how this problem can be made amenable to being coded. The main program you are trying to specify is called poker and it takes a list of hands as input and outputs the best hand. The best hands are described here:

http://en.wikipedia.org/wiki/List_of_poker_hands

These rules dictate which hands beat which, that is, how each hand ranks relative to one another.

There are three concepts that make up the hand rank:

1. Kind means that there are cards of the same rank. “n-kind,” where n can be one of a kind, two of a kind or three of a kind.
2. Straight means that there are five consecutive ranks and the suit does not matter. For example: a **five** of clubs, a **six** of spades, a **seven** of diamonds, an **eight** of spades and a **nine** of hearts. This is a straight because the rank of the cards is sequential.
3. Flush means that all of the cards are the same suit and the ranks do not matter.

For example: a ten of **diamonds**, a seven of **diamonds**, five of **diamonds**, four of **diamonds** and two of **diamonds**.

Now you know about the types of data you are dealing with: hands, cards, ranks and suits. You also now know about the functions for them: n-kind, straight, flush. From here you can move on to step 3, the design phase.

Step 3: Design working code

- That's what this course is all about!

### 1.1.1 1.1. Quiz: Representing Hands

Unit1-3

Which of the possible representations for a hand makes the most sense. There may be more than one.

- a. [‘JS’, ‘JD’, ‘2S’, ‘2C’, ‘7H’]
- b. [(11, ‘S’), (11, ‘D’), (2, ‘S’), (2, ‘C’), (7, ‘H’)]
- c. set ([‘JJ’, ‘JD’, ‘2S’, ‘2C’, ‘7H’)])
- d. “JS JD 2S 2C 7H”

## 1.2 2. Wild West Poker

Unit1-11

A poker player will call out his hand when he has to reveal it. With this information you will know how to rank and assign the hand.

- *"Straight flush, Jack high!"*

This is all you have to know about the hand; a straight flush and the highest ranking-Jack.

Here is a ranking table of the possible hands:

- 0- High Card
- 1- One Pair
- 2- Two Pair
- 3- Three of a Kind
- 4- Straight
- 5- Flush
- 6- Full House
- 7- Four of a Kind
- 8- Straight Flush

Use the numbers associated with the hand when writing your program.

This hand can be written:

```
·· 1 (8, 11)¶
¶
```

Here the eight stands for straight flush and the number 11 stands for the Jack.

Here are a few examples of what a player might say and how you would write their hand:

1. *Four aces and a queen kicker?*

```
·· 1 (7, 14, 12)¶
¶
```

Notice in this case, and when you have one pair, two pair, three of a kind and four of a kind, that there is an extra string in the set that tells you what kind you have.

1. *Full house, eights over kings?*

```
·· 1 (6, 8, 13)¶
¶
```

Even though a king is higher than an eight, because there are three eights in this full house, the eight is what matters most so it is included first, followed by 13, which indicates the kings.

1. *Flush, 10-8!*

Usually, declaring the two highest cards in the flush is enough to distinguish the hand from all other flushes. However, you might need all of the cards in the hand to break a tie, although it is unlikely that someone else would have the same exact hand.

When you write this, start with the highest card and work your way down.

```
·· 1 (5, [10, 8, 7, 5, 3])¶
¶
```

1. *Straight, Jack high*

This is all you need to know, because if Jack is the high card, than the rest of the cards in the hand, since it is a straight, have to be ten, nine, eight and seven.

```
·· 1 (4, 11)¶
¶
```

1. *Three sevens!*

Usually, this is enough information to distinguish a hand, but if you really need to break the ties, you can write the complete list of the five cards.

```
·· 1 (3, 7, [7, 7, 7, 5, 2])¶
¶
```

1. *Two pairs, Jacks and threes*

While this describes most of the hand, you also need to compare all of the cards, including the two pairs. Write the highest ranking pair first and include a set of the entire hand, just in case there is a tie – if someone else has a hand with two pairs, Jacks and threes.

```
·· 1 (2, 11, 3, [13, 11, 11, 3, 3])¶
¶
```

1. *Pair of twos, Jack high*

This partially describes a hand, but your player is not impressed with this hand. Here is how you represent this dismally dealt hand, make sure to include the set of the entire hand as a pair of two does not completely disambiguate this hand:

```
·· 1 (1, 2, [11, 6, 3, 2, 2])¶
¶
```

1. *Got nothing*

Sometimes, nobody gets dealt a good hand. How do you decide who wins? Go in order of the ranks of the cards:

```
·· 1 (0, 7, 5, 4, 3, 2)¶
¶
```

### 1.2.1 2.1. Quiz: Poker Function

Unit1-4

Out of the list of hands, you want poker to return the highest -ranking hand. Do you know of a built in function in Python that will allow you to return the highest-ranking item from a list?

Given:

```
·· 1 def poker(hands):¶
·· 2········ "return the best hand: poker([hand,…]) => hand"¶
·· 3········ return ???¶
¶
```

### 1.2.2 2.2. Quiz: Understanding Max

Unit1-5

What will the two max calls return?

```
·· 1 def poker(hands):¶
·· 2···· "Return the best hand: poker ([hand, ...]) => hand"¶
·· 3···· return max¶
¶
```

### 1.2.3 2.3. Quiz: Using Max

Unit1-6

Assume that you have defined a function **hand_rank**, which takes a hand as input and returns some sort of a rank. Given this, how would you write the definition of the function **poker** to return the maximum hand according to the highest ranked?

```
·· 1 def poker(hands):¶
·· 2···· "Return the best hand: poker([hand, …]) => hand"¶
·· 3···· return max¶
·· 4 ¶
·· 5 def hand_rank(hand):¶
·· 6···· return ???¶
·· 7 ¶
·· 8 print max([3, 4, 5, 0]), max ([3, 4, -5, 0], key = abs)¶
¶
```

### 1.2.4 2.4. Quiz: Testing

Unit1-7

Modify the test() function to include two new test cases:

- 1) four of a kind (fk) vs. full house (fh) returns fk.
- 2) full house (fh) vs. full house (fh) returns fh.

```
·· 1 def poker(hands):¶
·· 2···· "Return the best hand: poker([hand,...]) => hand"¶
·· 3···· return max(hands, key=hand_rank)¶
·· 4 ¶
·· 5 def test():¶
·· 6···· "Test cases for the functions in poker program"¶
·· 7···· sf = "6C 7C 8C 9C TC".split() # => ['6C', '7C', '8C', '9C', 'TC']¶
·· 8···· fk = "9D 9H 9S 9C 7D".split()¶
·· 9···· fh = "TD TC TH 7C 7D".split()¶
· 10···· assert poker([sf, fk, fh]) == sf¶
· 11···· assert poker([fk, fh]) == fk¶
· 12···· assert poker([fh, fh]) == fh¶
· 13 ¶
· 14···· # Add 2 new assert statements here. The first¶
· 15···· # should check that when fk plays fh, fk¶
· 16···· # is the winner. The second should confirm that¶
· 17···· # fh playing against fh returns fh.¶
· 18 ¶
· 19 print test()¶
¶
```

### 1.2.5 2.5. Quiz: Extreme Values

Unit1-8

### 1.2.6 2.6. Quiz: Hand Rank Attempt

Unit1-9

### 1.2.7 2.7. Quiz: Representing Rank

Unit1-10

## 1.3 3. Back to Hand Rank

Unit1-12

### 1.3.1 3.1. Quiz: Testing Hand Rank

Unit1-13

### 1.3.2 3.2. Quiz: Writing Hand Rank

Unit1-14

### 1.3.3 3.3. Quiz: Testing Card Rank

Unit1-15

### 1.3.4 3.4. Quiz: Fixing Card Rank

Unit1-16

### 1.3.5 3.5. Quiz: Straight and Flush

Unit1-17

### 1.3.6 3.6. Quiz: Kind Function

Unit1-18

### 1.3.7 3.7. Quiz: Two Pair Function

Unit1-19

### 1.3.8 3.8. Quiz: Making Changes

Unit1-20

### 1.3.9 3.9. Quiz: What to Change

Unit1-21

### 1.3.10 3.10. Ace Low Straight

Unit1-22

### 1.3.11 3.11. Quiz: Handling Ties

Unit1-23

### 1.3.12 3.12. Quiz: Allmax

Unit1-24

## 1.4 4. Deal

Unit1-25

### 1.4.1 4.1. Quiz: Hand Frequencies

Unit1-26

This is the procedure Peter gave us to calculate hand frequencies:

```
·· 1 def hand_percentages(n=700*1000):¶
·· 2···· counts = [0]*9¶
·· 3···· for i in range(n/10):¶
·· 4········ for hand in deal(10):¶
·· 5············ ranking = hand_rank(hand)[0]¶
·· 6············ counts[ranking] += 1¶
·· 7···· for i in reversed(range(9)):¶
·· 8········ print "%15s: %6.3f %%" % (hand_names[i], 100.*counts[i]/n)¶
¶
```

## 1.5 5. Dimensions of Programming

Unit1-27

## 1.6 6. Refactoring

Unit1-28

The two alternative versions of hand_rank that Peter gave in the refactoring class are:

```
·· 1 def hand_rank_alt(hand):¶
·· 2···· "Return a value indicating how high the hand ranks."¶
·· 3···· # count is the count of each rank; ranks lists corresponding
ranks¶
```

```
·· 4···· # E.g. '7 T 7 9 7' => counts = (3, 1, 1) ranks = (7, 10, 9)¶
·· 5···· groups = group(['--23456789TJQKA'.index(r) for r,s in hand])¶
·· 6···· counts, ranks = unzip(groups)¶
·· 7···· if ranks == (14, 5, 4, 3, 2):·· # Ace low straight¶
·· 8········ ranks = (5, 4, 3, 2, 1)¶
·· 9···· straight = len(ranks) == 5 and max(ranks) - min(ranks) == 4¶
· 10···· flush = len(set([s for r,s in hand])) == 1¶
· 11···· return (9 if (5,) == counts else¶
· 12·············· 8 if straight and flush else¶
· 13·············· 7 if (4, 1) == counts else¶
· 14·············· 6 if (3, 2) == counts else¶
· 15·············· 5 if flush else¶
· 16·············· 4 if straight else¶
· 17·············· 3 if (3, 1, 1) == counts else¶
· 18·············· 2 if (2, 2, 1) == counts else¶
· 19·············· 1 if (2, 1, 1, 1) == counts else¶
· 20·············· 0), ranks¶
· 21 ¶
· 22 def group(items):¶
· 23···· "Return a list of [(count, x), ...], highest count first, then
highest x first"¶
· 24···· groups = [(items.count(x), x) for x in set(items)]¶
· 25···· return sorted(groups, reverse = True)¶
· 26 ¶
· 27 def unzip(iterable):¶
· 28···· "Return a tuple of lists from a list of tuples : e.g. [(2, 9), (2,
7)] => ([2, 2], [9, 7])"¶
· 29···· return zip(*iterable)¶
¶
```

The table-based lookup version:

```
·· 1 count_rankings = {(5,): 10, (4, 1): 7, (3, 2): 6, (3, 1, 1): 3, (2, 2,
1): 2,¶
·· 2···················· (2, 1, 1, 1): 1, (1, 1, 1, 1, 1): 0}¶
·· 3 ¶
·· 4 def hand_rank_table(hand):¶
·· 5···· "Return a value indicating how high the hand ranks."¶
·· 6···· # count is the count of each rank; ranks lists corresponding
ranks¶
·· 7···· # E.g. '7 T 7 9 7' => counts = (3, 1, 1) ranks = (7, 10, 9)¶
·· 8···· groups = group(['--23456789TJQKA'.index(r) for r,s in hand])¶
·· 9···· counts, ranks = unzip(groups)¶
· 10···· if ranks == (14, 5, 4, 3, 2):·· # Ace low straight¶
· 11········ ranks = (5, 4, 3, 2, 1)¶
· 12···· straight = len(ranks) == 5 and max(ranks) - min(ranks) == 4¶
· 13···· flush = len(set([s for r,s in hand])) == 1¶
· 14···· return max(count_rankings[counts], 4*straight + 5*flush), ranks¶
¶
```

## 1.7 7. Summary

Unit1-29

## 1.8 8. Bonus - Shuffling

- Bad Shuffle
- Shuffle Runtime
- Good Shuffle
- Is it Random
- Testing Shuffles
- Comparing Shuffles
- Computing or Doing

The shuffling procedures from the bonus videos are:

```
·· 1 def shuffle1(p):¶
·· 2···· n = len(p)¶
·· 3···· swapped = [False]*n¶
·· 4···· while not all(swapped):¶
·· 5········ i, j = random.randrange(n), random.randrange(n)¶
·· 6········ swap(p, i, j)¶
·· 7········ swapped[i] = swapped[j] = True¶
·· 8 ¶
·· 9 def shuffle2(p):¶
· 10···· n = len(p)¶
· 11···· swapped = [False]*n¶
· 12···· while not all(swapped):¶
· 13········ i, j = random.randrange(n), random.randrange(n)¶
· 14········ swap(p, i, j)¶
· 15········ swapped[i] = True¶
· 16 ¶
· 17 def shuffle3(p):¶
· 18···· n = len(p)¶
· 19···· for i in range(n):¶
· 20········ swap(p, i, random.randrange(n))¶
· 21 ¶
· 22 def knuth(p):¶
· 23···· n = len(p)¶
· 24···· for i in range(n-1):¶
· 25········ swap(p, i, random.randrange(i, n))¶
· 26 def swap(p, i, j):¶
· 27···· p[i], p[j] = p[j], p[i]¶
¶
```

The procedures for testing the different shuffles were:

```
·· 1 def test_shuffle(shuffler, deck = 'abcd', n = 10000):¶
·· 2···· counts = defaultdict(int)¶
·· 3···· for _ in range(n):¶
·· 4········ input = list(deck)¶
·· 5········ shuffler(input)¶
·· 6········ counts[''.join(input)] += 1¶
·· 7···· e = n * 1./factorial(len(deck))¶
·· 8···· ok = all((0.9 <= counts[item]/e <= 1.1) for item in counts)¶
·· 9···· name = shuffler.__name__¶
· 10···· print '%s(%s) %s' % (name,· deck, ('ok' if ok else '*** BAD
***'))¶
· 11···· print '·· ',¶
· 12···· for item, count in sorted(counts.items()):¶
· 13········ print "%s:%4.1f" % (item, count * 100. / n),¶
· 14···· print¶
· 15 ¶
· 16 def test_shufflers(shufflers = [knuth, shuffle1, shuffle2, shuffle3],
decks = ['abc', 'ab']):¶
· 17···· for deck in decks:¶
· 18········ print¶
· 19········ for f in shufflers:¶
· 20············ test_shuffle(f, deck)¶
· 21 def factorial(n):¶
· 22···· return 1 if n<= 1 else n * factorial(n-1)¶
¶
```

---

## 1.9 9. Complete Code For Poker Problem

The complete code given by Peter in this unit including some additional test cases:

```
·· 1 #! /usr/bin/env python¶
·· 2 ¶
·· 3 import random¶
·· 4 ¶
·· 5 def poker(hands):¶
·· 6···· "Return a list of winning hands: poker([hand,...]) => [hand,...]"¶
·· 7···· return allmax(hands, key=hand_rank)¶
·· 8 ¶
·· 9 def allmax(iterable, key=None):¶
· 10···· "Return a list of all items equal to the max of the iterable."¶
· 11···· iterable.sort(key=key,reverse=True)¶
· 12···· result = [iterable[0]]¶
· 13···· maxValue = key(iterable[0]) if key else iterable[0]¶
· 14···· for value in iterable[1:]:¶
· 15········ v = key(value) if key else value¶
· 16········ if v == maxValue: result.append(value)¶
· 17········ else: break¶
· 18···· return result¶
· 19 ¶
· 20 def card_ranks(hand):¶
· 21···· "Return a list of the ranks, sorted with higher first."¶
· 22···· ranks = ['--23456789TJQKA'.index(r) for r, s in hand]¶
· 23···· ranks.sort(reverse = True)¶
· 24···· return [5, 4, 3, 2, 1] if (ranks == [14, 5, 4, 3, 2]) else ranks¶
· 25 ¶
· 26 def flush(hand):¶
· 27···· "Return True if all the cards have the same suit."¶
· 28···· suits = [s for r,s in hand]¶
· 29···· return len(set(suits)) == 1¶
· 30 ¶
· 31 def straight(ranks):¶
· 32···· "Return True if the ordered ranks form a 5-card straight."¶
· 33···· return (max(ranks)-min(ranks) == 4) and len(set(ranks)) == 5¶
· 34 ¶
· 35 def kind(n, ranks):¶
· 36···· """Return the first rank that this hand has exactly n-of-a-kind
of.¶
· 37···· Return None if there is no n-of-a-kind in the hand."""¶
· 38···· for r in ranks:¶
· 39········ if ranks.count(r) == n: return r¶
· 40···· return None¶
· 41 ¶
· 42 def two_pair(ranks):¶
· 43···· "If there are two pair here, return the two ranks of the two
pairs, else None."¶
· 44···· pair = kind(2, ranks)¶
· 45···· lowpair = kind(2, list(reversed(ranks)))¶
· 46···· if pair and lowpair != pair:¶
· 47········ return (pair, lowpair)¶
· 48···· else:¶
· 49········ return None¶
· 50 ¶
· 51 ¶
· 52 def hand_rank(hand):¶
· 53···· "Return a value indicating the ranking of a hand."¶
· 54···· ranks = card_ranks(hand)¶
· 55···· if straight(ranks) and flush(hand):¶
· 56········ return (8, max(ranks))¶
· 57···· elif kind(4, ranks):¶
· 58········ return (7, kind(4, ranks), kind(1, ranks))¶
· 59···· elif kind(3, ranks) and kind(2, ranks):¶
· 60········ return (6, kind(3, ranks), kind(2, ranks))¶
· 61···· elif flush(hand):¶
· 62········ return (5, ranks)¶
· 63···· elif straight(ranks):¶
· 64········ return (4, max(ranks))¶
```

```
· 65···· elif kind(3, ranks):¶
· 66········ return (3, kind(3, ranks), ranks)¶
· 67···· elif two_pair(ranks):¶
· 68········ return (2, two_pair(ranks), ranks)¶
· 69···· elif kind(2, ranks):¶
· 70········ return (1, kind(2, ranks), ranks)¶
· 71···· else:¶
· 72········ return (0, ranks)¶
· 73 ¶
· 74 def hand_rank_alt(hand):¶
· 75···· "Return a value indicating how high the hand ranks."¶
· 76···· # count is the count of each rank; ranks lists corresponding
ranks¶
· 77···· # E.g. '7 T 7 9 7' => counts = (3, 1, 1) ranks = (7, 10, 9)¶
· 78···· groups = group(['--23456789TJQKA'.index(r) for r,s in hand])¶
· 79···· counts, ranks = unzip(groups)¶
· 80···· if ranks == (14, 5, 4, 3, 2):·· # Ace low straight¶
· 81········ ranks = (5, 4, 3, 2, 1)¶
· 82···· straight = len(ranks) == 5 and max(ranks) - min(ranks) == 4¶
· 83···· flush = len(set([s for r,s in hand])) == 1¶
· 84···· return (9 if (5,) == counts else¶
· 85············ 8 if straight and flush else¶
· 86············ 7 if (4, 1) == counts else¶
· 87············ 6 if (3, 2) == counts else¶
· 88············ 5 if flush else¶
· 89············ 4 if straight else¶
· 90············ 3 if (3, 1, 1) == counts else¶
· 91············ 2 if (2, 2, 1) == counts else¶
· 92············ 1 if (2, 1, 1, 1) == counts else¶
· 93············ 0), ranks¶
· 94 ¶
· 95 ¶
· 96 count_rankings = {(5,): 10, (4, 1): 7, (3, 2): 6, (3, 1, 1): 3, (2, 2,
1): 2,¶
· 97················ (2, 1, 1, 1): 1, (1, 1, 1, 1, 1): 0}¶
· 98 ¶
· 99 def hand_rank_table(hand):¶
 100···· "Return a value indicating how high the hand ranks."¶
 101···· # count is the count of each rank; ranks lists corresponding
ranks¶
 102···· # E.g. '7 T 7 9 7' => counts = (3, 1, 1) ranks = (7, 10, 9)¶
 103···· groups = group(['--23456789TJQKA'.index(r) for r,s in hand])¶
 104···· counts, ranks = unzip(groups)¶
 105···· if ranks == (14, 5, 4, 3, 2):·· # Ace low straight¶
 106········ ranks = (5, 4, 3, 2, 1)¶
 107···· straight = len(ranks) == 5 and max(ranks) - min(ranks) == 4¶
 108···· flush = len(set([s for r,s in hand])) == 1¶
 109···· return max(count_rankings[counts], 4*straight + 5*flush), ranks¶
 110 ¶
 111 def group(items):¶
 112···· "Return a list of [(count, x), ...], highest count first, then
highest x first"¶
 113···· groups = [(items.count(x), x) for x in set(items)]¶
 114···· return sorted(groups, reverse = True)¶
 115 ¶
 116 def unzip(iterable):¶
 117···· "Return a list of tuples from a list of tuples : e.g. [(2, 9), (2,
7)] => [(2, 2), (9, 7)]"¶
 118···· return zip(*iterable)¶
 119 ¶
 120 mydeck = [r+s for r in '23456789TJQKA' for s in 'SHDC']¶
 121 ¶
 122 def deal(numhands, n=5, deck=mydeck):¶
 123···· random.shuffle(mydeck)¶
 124···· return [mydeck[n*i:n*(i+1)] for i in range(numhands)]¶
 125 ¶
```

```
 126 ¶
 127 hand_names = ["Straight flush", "Four of a kind", "Full house",
"Flush", "Straight",¶
 128·············· "Three of a kind", "Two pair", "One pair", "High card"]¶
 129 ¶
 130 def hand_percentages(n=700*1000):¶
 131···· counts = [0]*9¶
 132···· for i in range(n/10):¶
 133········ for hand in deal(10):¶
 134············ ranking = hand_rank(hand)[0]¶
 135············ counts[ranking] += 1¶
 136···· for i in reversed(range(9)):¶
 137········ print "%15s: %6.3f %%" % (hand_names[i], 100.*counts[i]/n)¶
 138 ¶
 139 def test():¶
 140···· "Test cases for the functions in poker program."¶
 141···· sf1 = "6C 7C 8C 9C TC".split() # Straight Flush¶
 142···· sf2 = "6D 7D 8D 9D TD".split() # Straight Flush¶
 143···· fk = "9D 9H 9S 9C 7D".split() # Four of a Kind¶
 144···· fh = "TD TC TH 7C 7D".split() # Full House¶
 145···· tp = "5D 2C 2H 9H 5C".split() # Two Pair¶
 146 ¶
 147···· # Testing allmax¶
 148···· assert allmax([2,4,7,5,1]) == [7]¶
 149···· assert allmax([2,4,7,5,7]) == [7,7]¶
 150···· assert allmax([2]) == [2]¶
 151···· assert allmax([0,0,0]) == [0,0,0]¶
 152 ¶
 153···· # Testing card_ranks¶
 154···· assert card_ranks(sf1) == [10, 9, 8, 7, 6]¶
 155···· assert card_ranks(fk) == [9, 9, 9, 9, 7]¶
 156···· assert card_ranks(fh) == [10, 10, 10, 7, 7]¶
 157 ¶
 158···· # Testing flush¶
 159···· assert flush([]) == False¶
 160···· assert flush(sf1) == True¶
 161···· assert flush(fh) == False¶
 162 ¶
 163···· # Testing straight¶
 164···· assert straight(card_ranks(sf1)) == True¶
 165···· assert straight(card_ranks(fk)) == False¶
 166 ¶
 167···· # Testing kind¶
 168···· assert kind(3, card_ranks(sf1)) == None¶
 169···· assert kind(4, card_ranks(fk)) == 9¶
 170 ¶
 171···· # Tesing two pair¶
 172···· assert two_pair(card_ranks(sf1)) == None¶
 173···· assert two_pair(card_ranks(tp)) == (5,2)¶
 174 ¶
 175···· # Testing group¶
 176···· assert group([2,3,4,6,2,1,9]) ==
[(2,2),(1,9),(1,6),(1,4),(1,3),(1,1)]¶
 177···· assert group([8,8,8,8]) == [(4,8)]¶
 178···· assert group([2,6,1]) == [(1,6),(1,2),(1,1)]¶
 179 ¶
 180···· # Testing unzip¶
 181···· assert unzip([(2,2),(1,9),(1,6),(1,4),(1,3),(1,1)]) ==
[(2,1,1,1,1,1),(2,9,6,4,3,1)]¶
 182···· assert unzip([(1,6),(1,2),(1,1)]) == [(1,1,1),(6,2,1)]¶
 183···· assert unzip([(2, 9), (2, 7)]) == [(2, 2), (9, 7)]¶
 184 ¶
 185···· # Testing hand rank¶
 186···· assert hand_rank(sf1) == (8,10)¶
 187···· assert hand_rank(fk) == (7,9,7)¶
 188···· assert hand_rank(fh) == (6,10,7)¶
```

```
 189 ¶
 190···· # Testing hand rank alt¶
 191···· assert hand_rank_alt(sf1) == (8, (10,9,8,7,6))¶
 192···· assert hand_rank_alt(fk) == (7,(9,7))¶
 193···· assert hand_rank_alt(fh) == (6,(10,7))¶
 194 ¶
 195···· # Testing hand rank table¶
 196···· assert hand_rank_table(sf1) == (9, (10,9,8,7,6))¶
 197···· assert hand_rank_table(fk) == (7,(9,7))¶
 198···· assert hand_rank_table(fh) == (6,(10,7))¶
 199 ¶
 200···· # Testing poker¶
 201···· assert poker([sf1, fk, fh]) == [sf1]¶
 202···· assert poker([fk, fh]) == [fk]¶
 203···· assert poker([fh, fh]) == [fh, fh]¶
 204···· assert poker([fh]) == [fh]¶
 205···· assert poker([sf2] + 99*[fh]) == [sf2]¶
 206···· assert poker([sf1, sf2, fk, fh]) == [sf1, sf2]¶
 207 ¶
 208···· return 'tests pass'¶
¶
```

## 1.10 10. Complete Code For Homeworks (Warning: Refer this only after submitting homework)

The complete homework solution code given by Peter in this unit.

Homework 1:

```
·· 1 # CS 212, hw1-1: 7-card stud¶
·· 2 #¶
·· 3 # -----------------¶
·· 4 # User Instructions¶
·· 5 #¶
·· 6 # Write a function best_hand(hand) that takes a seven¶
·· 7 # card hand as input and returns the best possible 5¶
·· 8 # card hand. The itertools library has some functions¶
·· 9 # that may help you solve this problem.¶
· 10 #¶
· 11 # -----------------¶
· 12 # Grading Notes¶
· 13 #¶
· 14 # Muliple correct answers will be accepted in cases¶
· 15 # where the best hand is ambiguous (for example, if¶
· 16 # you have 4 kings and 3 queens, there are three best¶
· 17 # hands: 4 kings along with any of the three queens).¶
· 18 ¶
· 19 import itertools¶
· 20 ¶
· 21 def best_hand(hand):¶
· 22···· "From a 7-card hand, return the best 5 card hand."¶
· 23···· return max(itertools.combinations(hand, 5), key=hand_rank)¶
· 24 ¶
· 25 # -----------------¶
· 26 # Provided Functions¶
· 27 #¶
· 28 # You may want to use some of the functions which¶
· 29 # you have already defined in the unit to write¶
· 30 # your best_hand function.¶
· 31 ¶
· 32 def hand_rank(hand):¶
· 33···· "Return a value indicating the ranking of a hand."¶
· 34···· ranks = card_ranks(hand)¶
· 35···· if straight(ranks) and flush(hand):¶
```

```
· 36········ return (8, max(ranks))¶
· 37···· elif kind(4, ranks):¶
· 38········ return (7, kind(4, ranks), kind(1, ranks))¶
· 39···· elif kind(3, ranks) and kind(2, ranks):¶
· 40········ return (6, kind(3, ranks), kind(2, ranks))¶
· 41···· elif flush(hand):¶
· 42········ return (5, ranks)¶
· 43···· elif straight(ranks):¶
· 44········ return (4, max(ranks))¶
· 45···· elif kind(3, ranks):¶
· 46········ return (3, kind(3, ranks), ranks)¶
· 47···· elif two_pair(ranks):¶
· 48········ return (2, two_pair(ranks), ranks)¶
· 49···· elif kind(2, ranks):¶
· 50········ return (1, kind(2, ranks), ranks)¶
· 51···· else:¶
· 52········ return (0, ranks)¶
· 53 ¶
· 54 def card_ranks(hand):¶
· 55···· "Return a list of the ranks, sorted with higher first."¶
· 56···· ranks = ['--23456789TJQKA'.index(r) for r, s in hand]¶
· 57···· ranks.sort(reverse = True)¶
· 58···· return [5, 4, 3, 2, 1] if (ranks == [14, 5, 4, 3, 2]) else ranks¶
· 59 ¶
· 60 def flush(hand):¶
· 61···· "Return True if all the cards have the same suit."¶
· 62···· suits = [s for r,s in hand]¶
· 63···· return len(set(suits)) == 1¶
· 64 ¶
· 65 def straight(ranks):¶
· 66···· """Return True if the ordered¶
· 67···· ranks form a 5-card straight."""¶
· 68···· return (max(ranks)-min(ranks) == 4) and len(set(ranks)) == 5¶
· 69 ¶
· 70 def kind(n, ranks):¶
· 71···· """Return the first rank that this hand has¶
· 72···· exactly n-of-a-kind of. Return None if there¶
· 73···· is no n-of-a-kind in the hand."""¶
· 74···· for r in ranks:¶
· 75········ if ranks.count(r) == n: return r¶
· 76···· return None¶
· 77 ¶
· 78 def two_pair(ranks):¶
· 79···· """If there are two pair here, return the two¶
· 80···· ranks of the two pairs, else None."""¶
· 81···· pair = kind(2, ranks)¶
· 82···· lowpair = kind(2, list(reversed(ranks)))¶
· 83···· if pair and lowpair != pair:¶
· 84········ return (pair, lowpair)¶
· 85···· else:¶
· 86········ return None¶
· 87 ¶
· 88 def test_best_hand():¶
· 89···· assert (sorted(best_hand("6C 7C 8C 9C TC 5C JS".split()))¶
· 90·············· == ['6C', '7C', '8C', '9C', 'TC'])¶
· 91···· assert (sorted(best_hand("TD TC TH 7C 7D 8C 8S".split()))¶
· 92·············· == ['8C', '8S', 'TC', 'TD', 'TH'])¶
· 93···· assert (sorted(best_hand("JD TC TH 7C 7D 7S 7H".split()))¶
· 94·············· == ['7C', '7D', '7H', '7S', 'JD'])¶
· 95···· return 'test_best_hand passes'¶
· 96 ¶
· 97 print test_best_hand()¶
¶
```

Homework 2:

```
# CS 212, hw1-2: Jokers Wild¶
#¶
# -----------------¶
# User Instructions¶
#¶
# Write a function best_wild_hand(hand) that takes as¶
# input a 7-card hand and returns the best 5 card hand.¶
# In this problem, it is possible for a hand to include¶
# jokers. Jokers will be treated as 'wild cards' which¶
# can take any rank or suit of the same color. The¶
# black joker, '?B', can be used as any spade or club¶
# and the red joker, '?R', can be used as any heart¶
# or diamond.¶
#¶
# The itertools library may be helpful. Feel free to¶
# define multiple functions if it helps you solve the¶
# problem.¶
#¶
# -----------------¶
# Grading Notes¶
#¶
# Muliple correct answers will be accepted in cases¶
# where the best hand is ambiguous (for example, if¶
# you have 4 kings and 3 queens, there are three best¶
# hands: 4 kings along with any of the three queens).¶
¶
import itertools¶
¶
## Deck adds two cards:¶
## '?B': black joker; can be used as any black card (S or C)¶
## '?R': red joker; can be used as any red card (H or D)¶
¶
allranks = '23456789TJQKA'¶
redcards = [r+s for r in allranks for s in 'DH']¶
blackcards = [r+s for r in allranks for s in 'SC']¶
¶
def best_wild_hand(hand):¶
····"Try all values for jokers in all 5-card selections."¶
····hands = set(best_hand(h)¶
·····for h in itertools.product(*map(replacements, hand)))¶
····return max(hands, key=hand_rank)¶
¶
def replacements(card):¶
····"""Return a list of the possible replacements for a card.¶
····There will be more than 1 only for wild cards."""¶
····if card == '?B': return blackcards¶
····elif card == '?R': return redcards¶
····else: return [card]¶
¶
def best_hand(hand):¶
····"From a 7-card hand, return the best 5 card hand."¶
····return max(itertools.combinations(hand, 5), key=hand_rank)¶
¶
def test_best_wild_hand():¶
····assert (sorted(best_wild_hand("6C 7C 8C 9C TC 5C ?B".split()))¶
···········== ['7C', '8C', '9C', 'JC', 'TC'])¶
····assert (sorted(best_wild_hand("TD TC 5H 5C 7C ?R ?B".split()))¶
···········== ['7C', 'TC', 'TD', 'TH', 'TS'])¶
····assert (sorted(best_wild_hand("JD TC TH 7C 7D 7S 7H".split()))¶
···········== ['7C', '7D', '7H', '7S', 'JD'])¶
····return 'test_best_wild_hand passes'¶
¶
# ------------------¶
# Provided Functions¶
#¶
# You may want to use some of the functions which¶
```

```
 67 # you have already defined in the unit to write¶
 68 # your best_hand function.¶
 69 ¶
 70 def hand_rank(hand):¶
 71····  "Return a value indicating the ranking of a hand."¶
 72····  ranks = card_ranks(hand)¶
 73····  if straight(ranks) and flush(hand):¶
 74····  ····  return (8, max(ranks))¶
 75····  elif kind(4, ranks):¶
 76····  ····  return (7, kind(4, ranks), kind(1, ranks))¶
 77····  elif kind(3, ranks) and kind(2, ranks):¶
 78····  ····  return (6, kind(3, ranks), kind(2, ranks))¶
 79····  elif flush(hand):¶
 80····  ····  return (5, ranks)¶
 81····  elif straight(ranks):¶
 82····  ····  return (4, max(ranks))¶
 83····  elif kind(3, ranks):¶
 84····  ····  return (3, kind(3, ranks), ranks)¶
 85····  elif two_pair(ranks):¶
 86····  ····  return (2, two_pair(ranks), ranks)¶
 87····  elif kind(2, ranks):¶
 88····  ····  return (1, kind(2, ranks), ranks)¶
 89····  else:¶
 90····  ····  return (0, ranks)¶
 91 ¶
 92 def card_ranks(hand):¶
 93····  "Return a list of the ranks, sorted with higher first."¶
 94····  ranks = ['--23456789TJQKA'.index(r) for r, s in hand]¶
 95····  ranks.sort(reverse = True)¶
 96····  return [5, 4, 3, 2, 1] if (ranks == [14, 5, 4, 3, 2]) else ranks¶
 97 ¶
 98 def flush(hand):¶
 99····  "Return True if all the cards have the same suit."¶
100····  suits = [s for r,s in hand]¶
101····  return len(set(suits)) == 1¶
102 ¶
103 def straight(ranks):¶
104····  """Return True if the ordered¶
105····  ranks form a 5-card straight."""¶
106····  return (max(ranks)-min(ranks) == 4) and len(set(ranks)) == 5¶
107 ¶
108 def kind(n, ranks):¶
109····  """Return the first rank that this hand has¶
110····  exactly n-of-a-kind of. Return None if there¶
111····  is no n-of-a-kind in the hand."""¶
112····  for r in ranks:¶
113····  ····  if ranks.count(r) == n: return r¶
114····  return None¶
115 ¶
116 def two_pair(ranks):¶
117····  """If there are two pair here, return the two¶
118····  ranks of the two pairs, else None."""¶
119····  pair = kind(2, ranks)¶
120····  lowpair = kind(2, list(reversed(ranks)))¶
121····  if pair and lowpair != pair:¶
122····  ····  return (pair, lowpair)¶
123····  else:¶
124····  ····  return None¶
125 ¶
126 print test_best_wild_hand()¶
¶
```