Lecture 11: Review 1

COMP90059 Introduction to Programming

Wally Smith

School of Computing & Information Systems

© University of Melbourne 2020

Lecture Overview

A review of MOCK EXAM - Sample 2

A detailed review of the MOCK exam questions Sample2 on Canvas, and thinking about how best to prepare for questions like these.

The card game challenge continued

As a continued exploration of computational thinking in larger programs with an applied purpose, we will extend the card game program from Week 10 to evaluate the poker hands dealt.

COMP90059 Introduction to Programming - Semester 1, 2020

MOCK EXAMINATION QUESTIONS - SAMPLE 1

- **DURATION** : 3 hours
- **THIS EXAM IS WORTH**: 70 marks and 70% of the subject grade
- **INDIVIDUAL ASSESSMENT**. Your answers to this examination must be your own work and you must not consult with others when completing it.

INSTRUCTIONS

- **Attempt EVERY question in this exam.**
- **Be SURE to press SUBMIT when you have finished entering your answers.**
- **OPEN-BOOK**. You can consult notes and sources of information during the examination. However, this will take up time and you are advised to prepare your own set of notes that are likely to be most relevant and quick to access.
- **IDLE**. For Section D, you can use Python's IDLE to write and check your program code. Paste your code from IDLE into the answer box of the exam and check that the formatting is correct. You will still get marks for correct elements in code that does not run or has errors, so enter best your best answer.

SECTIONS, MARKS & RECOMMENDED TIME ALLOCATION

There are four Sections of questions as follows:

- **Section A. Programming Concepts I**; 20 marks, spend approx 50 minutes

Multiple choice questions about the elements of programs, techniques of programming,; and computational thinking.

- **Section B. Programming Concepts II**; 15 marks, spend approx 40 minutes

Short answer questions about the elements of programs, techniques of programming, and computational thinking.

- **Section C. Reading Code**; 15 marks, spend approx 40 minutes

Presenting pieces of code and asking you to interpret their purpose, outputs, any errors, and other aspects.

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

(This MOCK EXAM is INCOMPLETE and shows ONLY some EXAMPLE QUESTIONS)

Programming Concepts and Techniques

COMP90059 Introduction to Programming - Semester 1, 2020

WHAT IS PROGRAMMING?

- comparison with natural languages
- problem analysis, problem decomposition
- modular program design
- algorithms
- code quality: correct syntax, effectiveness, efficiency, readability, traceability, extensibility, elegance
- programming environments: IDLE, the shell, the script
- interpreted vs compiled languages

CODING RULES, CONVENTIONS & STYLE

- variable naming rules
- commenting the code
- reserved words
- PEP8 style guidelines

SCALES OF PROGRAM CODE

- expressions
- statements
- functions
- modules (ie a program script)
- libraries

COMMON BUILT-IN FUNCTIONS

- print()
- input()
- abs()

... more

FORMATTING OUTPUT

- f-string

DATA TYPES & STRUCTURES

- literals
- variables: string, integer, real numbers, boolean
- collections: lists, tuples, sets, dictionaries
- dynamic typing in Python
- type casting: int(), str(), float(), list()
- type identification: type()

CHARACTER OPERATIONS

- escape sequences
- UniCode character set
- ord(), chr()

EXPRESSIONS

- assignment, simultaneous assignment
- strings: concatenation, repetition
- arithmetic expressions: operators and precedence rules (**, * , / , // , % + , -)
- boolean expressions: comparison operators (==, !=, >, <, >=, <=, and, or, not, in)

in: substring in string, item in list, key in dict
not: the inverse of a boolean, eg. 'a' not in 'castle'

SEQUENCE OPERATIONS

- len()
- indexing: string[index], list[index]
- slicing: string[start, stop, step], list[start, stop, step],
- multidimensional indexing & slicing
- string methods: center(), count(), endswith(), find(), isalpha(), isdigit(), join(), lower(), replace(), split(), startswith(), strip(), upper()
- list methods: append(), sort(), reverse(), index(), insert(), count(), remove(), pop()
- dictionary methods: keys(), values(), items(), get(key, <default>), del dict[key], clear()
- min(), max() - for lists

CONTROL FLOW

- default line by line
- conditionals: if, elif, else; nested conditionals
- iteration: for loops, while loops, nested loops
- break

ITERABLES - The things we iterate through

- range(start, stop, step), range(len(string))
- strings
- lists, tuples, dictionaries

DEFINED FUNCTIONS & MODULAR PROGRAM DESIGN

- function definition
- parameters, arguments
- local and global scope
- return
- positional vs keyword arguments
- modular program design using main( )

MUTABILITY

- mutable vs immutable data structures
- aliasing
- mutables as parameters

FILES

- reading from a data file: read(), readline(), readlines()
- write to a data file: print('entry', file =<file_object>)

ERRORS

- types of error: syntax, run-time, logic
- debugging techniques

CODING PATTERNS

- sentinel while loop
- building a string
- building a list - using append()
- working through a temporary list - using pop()
- accumulators
- boolean flags
- converting a string to a list - using list() and split()

useful methods

- sort()
- count()
- upper(), lower()
- isalpha(), isdigit()

useful functions that convert a string to a list

- list()
- sorted()

... and from a list to a string

- join()

Tips for coding questions in the exam

- You will get a good mark for the **right approach**, even if your code does not work perfectly.
- Always make a **plan showing each step of your program.**
  (don't include it in your answer unless asked to do so)
- You do **NOT need to comment** your code (unless asked to do so)
- There are **many different solutions to a question**, try to think of the simplest approach first ... but don't worry if yours is not the perfect solution.
- Learn the **coding patterns** and be ready to use them.
- Study the **methods and functions** that we have looked at and be ready to apply them.
- Be clear about **strings and lists:** the difference between their methods and functions, when and how to convert between them.
- Be sure about **string indexing & slicing, and range()**
  remember: count from zero, stopping one short of stop.
- Be sure about **Boolean expressions and operators.**

- **Section A. Programming Concepts I**; 20 marks, spend approx 50 minutes

Multiple choice questions about the elements of programs, techniques of programming,; and computational thinking.

Question 1

1 pts

Which of the following is a major benefit of modular program design using functions?

- The code runs much faster.
- There is a less linear flow from the beginning to the end of the code.
- The code has shorter statements.
- The code is less repetitive.
- Data is read from files more efficiently.

LECTURE 05

```python
#Days of Life Calculation V4
#V4 uses functions for readability, to reduce repetition and to improve

def main():
    dobDay, dobMonth, dobYear = getUsersBirthDate()
    todayDay,todayMonth,todayYear = gettoday()
    daysOfLife =  daysFirstYear(dobDay, dobMonth, dobYear)\
                + daysWholeYears(dobYear, todayYear)\
                + daysCurrentYear(todayDay, todayMonth, todayYear)
    print('You have been alive for', daysOfLife,'days')
```

```python
def daysFirstYear
    daysOfMonths
    if leapYear(
        daysOfMo
    days = daysO
    for month in
        days +=
    return days
```

```python
def daysWholeYears
    days = (y2-y1-1
    for year in range(y1,y2+1):
        if leapYear(year):
            days += 1
    return days
```

```python
# Calculate number of leap years (leapYears)
leapYears = 0
for year in range (dobYear+1, todayYear):
    if year%400==0 or (year%400 != 0 and year%100 != 0 and year%4 == 0):
        leapyears = leapYears + 1
if dobYear%400==0 or (dobYear%400 != 0 and dobYear%100 != 0 and dobYear%4 == 0):
    if dobMonth <= 2:
        leapYears = leapYears + 1
if todayYear%400==0 or (todayYear%400 != 0 and todayYear%100 != 0 and todayYear%
    if todayMonth > 2:
        leapYears = leapYears + 1
```

earlier code to calculate leap year in Version 2

```python
def leapYear(year):
    if year%400==0 or (year%400 != 0 and year%100 != 0 and year%4 == 0):
        return True
    else:
        return
```

- **Section A. Programming Concepts I**; 20 marks, spend approx 50 minutes

Multiple choice questions about the elements of programs, techniques of programming,; and computational thinking.

Question 2

1 pts

Which of the following is NOT true about using lists in Python?

- Lists have more flexible naming rules than integer and string variables.
- Work done on a list inside a function takes effect elsewhere in a program.
- Lists can be built dynamically at run-time using append().
- Lists can contain a variety of data types.
- Lists are iterables.

Variable Names - Rules in Python

- Reserved words cannot be used as variable names
  Examples: if, for, while and import

- Don't use built-in functions as variable names
  Examples: enumerate, print, input

- Names must begin with a letter or underscore _

- The rest of the name can contain zero or more occurrences of the following things:
  digits (0 to 9)
  alphabetic letters
  underscores

- Names are case sensitive
  Example: **WEIGHT** is different from **weight**

Lists

A list is an **ordered** and **mutable** collection of values.

square brackets

```
customers = ['Johnson', 'Wang', 'Zhu', 'Agarwal', 'Williams', 'Nguyen']
```

* assorted values: can contain a mixture of types of data including collections

```
profile = [ 'Wilson', 45, True, [2, 4, 56, 23], 'Sydney', 3078]
```

* ordered: the list can be indexed and sliced

```
print(profile[1])          45
print(profile[2:4])        True, [2,4,56,23]
```

* mutable: items can be added or changed *in place*

```
profile[2] = 'hello'       ['Wilson', 45, 'hello', [2, 4, 56, 23], 'Sydney', 3078]
```

Lists are the most flexible and useful collections in Python.

Mutability allows lists to be built and modified dynamically during run-time.

- **Section A. Programming Concepts I**; 20 marks, spend approx 50 minutes

Multiple choice questions about the elements of programs, techniques of programming,; and computational thinking.

LECTURE 03
Conditionals:
Boolean expressions

LECTURE 04
Strings:
indexing & slicing

Question 3

1 pts

Which of the following Boolean expressions is true? Assume that the following assignments have been made: first = 'The quick brown fox jumped over the lazy dog'; second = 'Who told them that?'

- first[0] == second[4]
- second[3] in first[0:15]
- len(first) == len(second) * 2
- first + first == second
- not(len(second) < 100)

indexing & slicing

'	i	w	e	Y	d	U	h	O	p	L	k	j	h	U	A	U	'
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

```
password = 'iweYdUhOpLkjhUAU'
```

indexing

index

slice

slicing

```
print( password[3] )                    Y
print( password[-1] )                   U

print( password[4:8] )                  dUhO
print( password[:5] )                   iweYd
print( password[-3:] )                  UAU
print( password[-7:-1] )                LkjhUA
print( password[-7:len(password)] )     LkjhUAU
print( password[:] )                    iweYdUhOpLkjhUAU
```

- **Section B. Programming Concepts II**; 15 marks, spend approx 40 minutes

Short answer questions about the elements of programs, techniques of programming, and computational thinking.

Question 4 3 pts

In a nested for-loop (containing an inner and an outer loop), one loop cycles at a faster rate than the other. Name the loop that cycles more quickly, and write ONE sentence to explain why this is true.

12pt Paragraph

inner loop
The inner loop cycles faster because for each cycle of the outer loop, the inner loop goes through all of its cycles.

```
words = ['hello', 'why', 'goodbye', 'not']
for word in words:
    for ch in word:
        print(ch)
```

h
e
l
l
o
w
h
y
g
o
o
d
b
y
e
n
o
t

Question 5 3 pts

Name THREE techniques that change the default line-by-line flow of control in a Python program.

12pt Paragraph

- conditionals - if, elif, else
- iteration - for loops, while loops
- functions

0 words

try ... except

break
continue
return - in the middle of a function

- **Section C. Reading Code**; 15 marks, spend approx 40 minutes

Presenting pieces of code and asking you to interpret their purpose, outp
other aspects.

Question 6 4 pts

Write out ONE statement from the program in Figure 1 for which BOTH the following are true: (i) it is an assignment statement, and (ii) it contains a list of lists.

12pt Paragraph B I U A T²

chartypes = defineCharTypes()

Question 7 2 pts

If the code in Figure 1 was put into IDLE exactly in this form, it would not produce any output on the screen. Write out the SINGLE statement that needs to be added to the code to make it produce an output to the screen.

12pt Paragraph B I U A T²

main()

0 words

```
def main():
    password = input('Enter your proposed password: ')
    print(password,'is', status(password))

def status(password):
    chartypes = defineCharTypes()
    if lengthOK(password) and countCategories(password, chartypes)>2:
        return 'secure'
    else:
        return 'not secure'

def defineCharTypes():
    special = ['specials', False, 33, 47]
    digit = ['digits', False, 49, 57]
    lowerCase = ['lower case', False, 97, 122]
    upperCase = ['upper case', False, 65, 90]
    return [special, digit, lowerCase, upperCase]

def lengthOK(password):
    if len(password)>= 8 and len(password)<=20:
        return True
    else:
        return False

def countCategories(password, chartypes):
    for char in password:
        for chartype in chartypes:
            if ord(char) in range(chartype[2], chartype[3]+1):
                chartype[1]= True
    numberCategories = 0
    for chartype in chartypes:
        if chartype[1]:
            numberCategories +=1
    return numberCategories
```

- **Section C. Reading Code**; 15 marks, spend approx 40 minutes

Presenting pieces of code and asking you to interpret their purpose, outp
other aspects.

Write ONE sentence to explain what the expression ord(char) in Figure 1 means.

12pt Paragraph B I U A T²

It is the numeric value of the character char in the ASCII or UniCode character set.

Question 9 2 pts

The code in Figure 1 uses an accumulator code pattern in one of its functions. Write down ONLY the name of this function.

12pt Paragraph B I U A T²

countCategories

0 words

```
def main():
    password = input('Enter your proposed password: ')
    print(password,'is', status(password))

def status(password):
    chartypes = defineCharTypes()
    if lengthOK(password) and countCategories(password, chartypes)>2:
        return 'secure'
    else:
        return 'not secure'

def defineCharTypes():
    special = ['specials', False, 33, 47]
    digit = ['digits', False, 49, 57]
    lowerCase = ['lower case', False, 97, 122]
    upperCase = ['upper case', False, 65, 90]
    return [special, digit, lowerCase, upperCase]

def lengthOK(password):
    if len(password)>= 8 and len(password)<=20:
        return True
    else:
        return False

def countCategories(password, chartypes):
    for char in password:
        for chartype in chartypes:
            if ord(char) in range(chartype[2], chartype[3]+1):
                chartype[1]= True
    numberCategories = 0
    for chartype in chartypes:
        if chartype[1]:
            numberCategories +=1
    return numberCategories
```

Common Python errors

- equality (==) vs. assignment (=)
- failing to return a value from functions
- incorrect use of types, e.g, forgetting that **input()** returns a string
- incorrect use of a function or method
- mis-spelling a variable name
- indexing and slicing - forgetting to count from zero
- loops and incrementing (out by one!)
- wrong indentation in function definitions, conditionals, loops: shifting lines of code into the wrong block
- forgetting to put () on a function call
- forgetting to put main() in the top-area of the program

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 10

3pts

Write a program that takes a word from the user and prints that word with the letters in reverse order. Do NOT use the function **reversed()** or the method **list.reverse()**.

12pt Paragraph

```
word = input('Enter a word')
letters = list(word)
letters.reverse()
print(''.join(letters))
```

0 words

What if we could use **list.reverse()**?

Note that **list.reverse()** works with lists but not strings.

converting a string to a list

* input word
* convert word to a list of letters
* use list.reverse()
* convert the list back to a string using join()
* print answer

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 10

3pts

Write a program that takes a word from the user and prints that word with the letters in reverse order. Do NOT use the function **reversed()** or the method **list.reverse()** .

12pt Paragraph

```
word = input('Enter a word')
new_word = ''
for c in range(len(word)-1,-1,-1):
    new_word += word[c]
print(new_word)
```

0 words

building a string

* create new_word as empty string
* iterate backwards through the letters of word
  * add each letter to new_word
* print new_word

0 1 2 3 4 5 6

'program'

len(word) = 7

range(start, stop, step)

range(len(word)-1, -1, -1)

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 10

3pts

Write a program that takes a word from the user and prints that word with the letters in reverse order. Do NOT use the function **reversed()** or the method **list.reverse()** .

12pt Paragraph

```
word = input('Enter a word: ')
new_word = ''
i = len(word)
while i >= 1:
    new_word += word[i-1]
    i -= 1
print(new_word)
```

0 words

building a string

* create new_word as empty string
* set counter variable to len word
* iterate backwards using a while loop
  and decreasing counter variable
  * add each letter to new_word
* print new_word

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 10

3pts

Write a program that takes a word from the user and prints that word with the letters in reverse order. Do NOT use the function **reversed()** or the method **list.reverse()** .

12pt Paragraph

```
word = input('Enter a word')
new_word = ''
for ch in word:
    new_word = ch + new_word
print(new_word)
```

0 words

building a string

* create new_word as empty string
* iterate through the letters of word
  * add each letter to the front of new_word
* print new_word

Slicing: Step Size and Direction

Slicing can specify a third number which indicates how much to step through the sequence by:


```
sentence = "the quick brown fox jumped over the lazy dog"
print( sentence[0:16:2] )
```

slice step

teqikbon

```
print( sentence[16:0:-2] )
```

fnobkiqe

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 10

3pts

Write a program that takes a word from the user and prints that word with the letters in reverse order. Do NOT use the function **reversed()** or the method **list.reverse()** .

12pt Paragraph

```
word = input('Enter a word')
for ch in word[::-1]:
    print(ch, end='')
```

0 words

slicing

* iterate through a reverse slice of word
* print each letter without a return character

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 10

3pts

Write a program that takes a word from the user and prints that word with the letters in reverse order. Do NOT use the function **reversed()** or the method **list.reverse()** .

12pt Paragraph

```
print(input('Enter a word: ')[::-1])
```

0 words

slicing

* print a reverse slice of an inputted word

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 11

5 pts

Write a function definition that takes a list that contains playing cards and returns a Boolean variable (called flush) which indicates if the playing cards are all the same suit, or not. Each playing card in the list is represented by a string that combines its value (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, T, J, Q, K, A) with its suit (C, H, D, S). For example, a hand of cards might be: ['3C', 'kH', '10S', 'JS', 'AD']. Your function must determine if all of the cards are the same suit. Your function should accept a list of any length.

12pt Paragraph

```
def detect_flush(hand):
    flush = True
    for card in range(len(hand)-1):
        if hand[card][1] != hand[card+1][1]:
            flush = False
    return flush
```

0 words

Boolean flag

['3C', 'KC', '10C', '8C', '9C']

['2D', 'JD', '7S', '2D', '7H']

* set Boolean flag, flush = True
* iterate through the cards of the hand
    * if card's suit is NOT the same as the next card, set flush to False
* return flush

doesn't work for '10C' or any 10

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 11

5 pts

Write a function definition that takes a list that contains playing cards and returns a Boolean variable (called flush) which indicates if the playing cards are all the same suit, or not. Each playing card in the list is represented by a string that combines its value (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, T, J, Q, K, A) with its suit (C, H, D, S). For example, a hand of cards might be: ['3C', 'kH', '10S', 'JS', 'AD']. Your function must determine if all of the cards are the same suit. Your function should accept a list of any length.

12pt Paragraph

```
def detect_flush(hand):
    flush = True
    for card in range(len(hand)-1):
        if hand[card][-1] != hand[card+1][-1]:
            flush = False
    return flush
```

0 words

Boolean flag

['3C', 'KC', '10C', '8C', '9C']

['2D', 'JD', '7S', '2D', '7H']

* set Boolean flag, flush = True
* iterate through the cards of the hand
    * if card's suit is NOT the same as the next card, set flush to False
* return flush

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 11

5 pts

Write a function definition that takes a list that contains playing cards and returns a Boolean variable (called flush) which indicates if the playing cards are all the same suit, or not. Each playing card in the list is represented by a string that combines its value (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, T, J, Q, K, A) with its suit (C, H, D, S). For example, a hand of cards might be: ['3C', 'kH', '10S', 'JS', 'AD']. Your function must determine if all of the cards are the same suit. Your function should accept a list of any length.

12pt Paragraph

```
def check_flush(hand):
    suit = hand[0][-1]
    for card in hand:
        if card[-1] != suit:
            return False
    return True
```

0 words

Boolean flag

['3C', 'KC', '10C', '8C', '9C']

['2D', 'JD', '7S', '2D', '7H']

* set Boolean flag, flush = True
* iterate through the cards of the hand
    * if card's suit is NOT the same as the first card, set flush to False
* return flush

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requirin ... blem.

'the quick brown fox'

['the', 'quick', 'brown', 'fox']

Question 12

Write a function definition that takes a sentence and capitalizes (i.e., makes upper case) the first letter of every word. Do not use the Python **capitalize()** function as part of your answer. Your function should take the sentence as a string parameter, and return the transformed sentence. Assume that in the initial sentence there is a single space between every word and no spaces or other characters at the beginning or end.

12pt Paragraph B I U A T² ⋮

0 words

converting a string to a list

building a string - twice

* split sentence into list of words
* create an empty new sentence
* iterate through each word
  * create an empty new word
  * iterate through the letter of each word
    * convert the first letter to upper case
    * add each letter to new word
  * add each new word to new sentence
* return new sentence

```
def capital(sentence):
    words = sentence.split()
    new_sentence = ''
    for word in words:
        new_word = ''
        for ch in range(len(word)):
            if ch == 0:
                letter = word[ch].upper()
            else:
                letter = word[ch]
            new_word += letter
        new_sentence += new_word + ' '
    return new_sentence[:-1]
```

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requirin ... blem.

'the quick brown fox'

['the', 'quick', 'brown', 'fox']

Question 12

Write a function definition that takes a sentence and capitalizes (i.e., makes upper case) the first letter of every word. Do not use the Python **capitalize()** function as part of your answer. Your function should take the sentence as a string parameter, and return the transformed sentence. Assume that in the initial sentence there is a single space between every word and no spaces or other characters at the beginning or end.

12pt Paragraph B I U A T²

0 words

building a string

* create an empty new sentence
* iterate through each letter
  * if previous letter was a space ... convert the letter to upper case
  * add each letter to new sentence
* return new sentence

```
def capital(sentence):
    new_sentence = ''
    space = True
    for ch in sentence:
        if space:
            new_sentence += ch.upper()
        else:
            new_sentence += ch
        if ch == ' ':
            space = True
        else:
            space = False
    return new_sentence
```

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 13

6 pts

Write a program that takes in two words from a user and reports back if they are anagrams of each other. If two words are anagrams, they have exactly the same letter but in a different order. Your program should disregard the case of letters, so 'Cat' and 'tac' are anagrams.

12pt Paragraph

0 words

Boolean flag

converting a string to a list

* input word1 and word2
* set flag: anagram is False
* make words lower case
* convert words to sorted list of letters
* if sorted lists are identical, and word1 is not identical to word 2, then its an anagram
* return result

```
word1 = input('Enter first word: ')
word2 = input('Enter second word: ')
anagram = False

word1 = word1.lower()
word2 = word2.lower()

letters1 = sorted(word1)
letters2 = sorted(word2)

if letters1 == letters2 and word1 != word2:
    anagram = True

if anagram:
   print('The words are anagrams')
else:
   print('The words are not anagrams')
```

- **Section D. Analysis and Coding**; 20 marks, spend approx 50 minutes

Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Question 13

6 pts

Write a program that takes in two words from a user and reports back if they are anagrams of each other. If two words are anagrams, they have exactly the same letter but in a different order. Your program should disregard the case of letters, so 'Cat' and 'tac' are anagrams.

12pt Paragraph

0 words

Boolean flag

converting a string to a list

* input word1 and word2
* set flag: anagram is False
* make words lower case
* convert words to sorted list of letters
* if sorted lists are identical, and word1 is not identical to word 2, then its an anagram
* return result

```
word1 = input('Enter first word: ').lower()
word2 = input('Enter second word: ').lower()
anagram = False

letters1 = sorted(word1)
letters2 = sorted(word2)

if letters1 == letters2 and word1 != word2:
    anagram = True

if anagram:
   print('The words are anagrams')
else:
   print('The words are not anagrams')
```

The card game challenge: evaluating poker hands

Challenge

Write a program that can print out hands of cards from a 'shuffled' pack of playing cards.

```
********* Round 1 ********
Player 1 : 9♥  4♥  3♥  2♦  10♠
Player 2 : J♣  7♦  K♦  6♠  K♣
Player 3 : 7♠  7♣  9♦  K♠  5♣
Player 4 : 8♦  10♣ J♦  Q♠  8♠
Player 5 : 5♠  4♣  5♦  8♥  2♣
Player 6 : 6♦  A♣  Q♥  8♣  J♠
Player 7 : 4♠  4♦  3♣  Q♣  5♥
Player 8 : Q♦  10♥ A♦  9♣  2♠
```

```
suitNames = ('♣','♦','♥','♠')
valueNames = ('A','2','3','4','5','6','7','8','9','10','J','Q','K')
```

```
def main():
    players = 8
    cardsPerPlayer = 5
    rounds = 3
    for round in range(rounds):
        show_round(round)
        deal_round(players, cardsPerPlayer)
```

```
def show_round(round):
    print('********* Round',round+1,'********')

def deal_round(players, cards):
    deck = [ ]
    manufacture(deck)
    shuffle(deck)
    for player in range(players):
        hand = deal_hand(deck, cards)
        report(player, hand)
```

```
********* Round 1 ********
Player 0 : K♠ K♥ 6♠ 7♠ Q♣
Player 1 : 7♦ 3♦ 8♣ 4♦ 9♠
Player 2 : 4♥ 4♣ A♥ J♥ 2♥
Player 3 : 7♣ 10♣ K♣ K♦ J♣
Player 4 : 8♦ 5♠ 2♠ Q♠ A♠
Player 5 : 3♠ 8♠ 5♦ 4♠ 6♣
Player 6 : 7♥ 6♥ 9♦ 10♠ 10♥
Player 7 : 2♦ A♣ 8♥ Q♦ 2♣

********* Round 2 ********
Player 0 : 9♥ 4♥ 3♥ 2♦ 10♠
Player 1 : J♣ 7♦ K♦ 6♠ K♣
Player 2 : 7♠ 7♣ 9♦ K♠ 5♣
Player 3 : 8♦ 10♣ J♦ Q♠ 8♠
Player 4 : 5♠ 4♣ 5♦ 8♥ 2♣
Player 5 : 6♦ A♣ Q♥ 8♣ J♠
Player 6 : 4♠ 4♦ 3♣ Q♣ 5♥
Player 7 : Q♦ 10♥ A♦ 9♣ 2♠

********* Round 3 ********
Player 0 : J♦ A♣ 6♠ 2♠ J♠
Player 1 : 9♠ Q♦ 3♥ J♣ 8♣
Player 2 : 4♠ 7♣ 3♦ K♦ 5♦
Player 3 : 9♣ 8♦ 4♥ 2♥ 10♥
Player 4 : J♥ 6♦ 10♣ 4♣ 3♣
Player 5 : 5♠ 9♦ 2♣ 5♣ 7♥
Player 6 : 8♥ A♥ 10♠ 6♥ 10♦
Player 7 : 4♦ A♦ 7♠ 8♠ 3♠
```

```
def deal_round(players, cards):
    deck = [ ]
    manufacture(deck)
    shuffle(deck)
    for player in range(players):
        hand = deal_hand(deck, cards)
        report(player, hand)
```

```
def manufacture(deck):
    suitNames = ('♣','♦','❤','♠')
    valueNames = ('A','2','3','4','5','6','7','8','9','10','J','Q','K')
    for suit in range(4):
        for value in range(13):
            deck.append(valueNames[value]+suitNames[suit])
```

```
def shuffle(deck):
    for swap in range(10000):
        x = random.randint(0,51)
        y = random.randint(0,51)
        deck[x], deck[y] = deck[y], deck[x]
```

deck

['A♣', '2♣', '3♣', '4♣', '5♣', '6♣', '7♣', '8♣', '9♣', '10♣', 'J♣', 'Q♣', 'K♣', 'A♦', '2♦', '3♦', '4♦', '5♦', '6♦', '7♦', '8♦', '9♦', '10♦', 'J♦', 'Q♦', 'K♦', 'A❤', '2❤', '3❤', '4❤', '5❤', '6❤', '7❤', '8❤', '9❤', '10❤', 'J❤', 'Q❤', 'K❤', 'A♠', '2♠', '3♠', '4♠', '5♠', '6♠', '7♠', '8♠', '9♠', '10♠', 'J♠', 'Q♠', 'K♠']

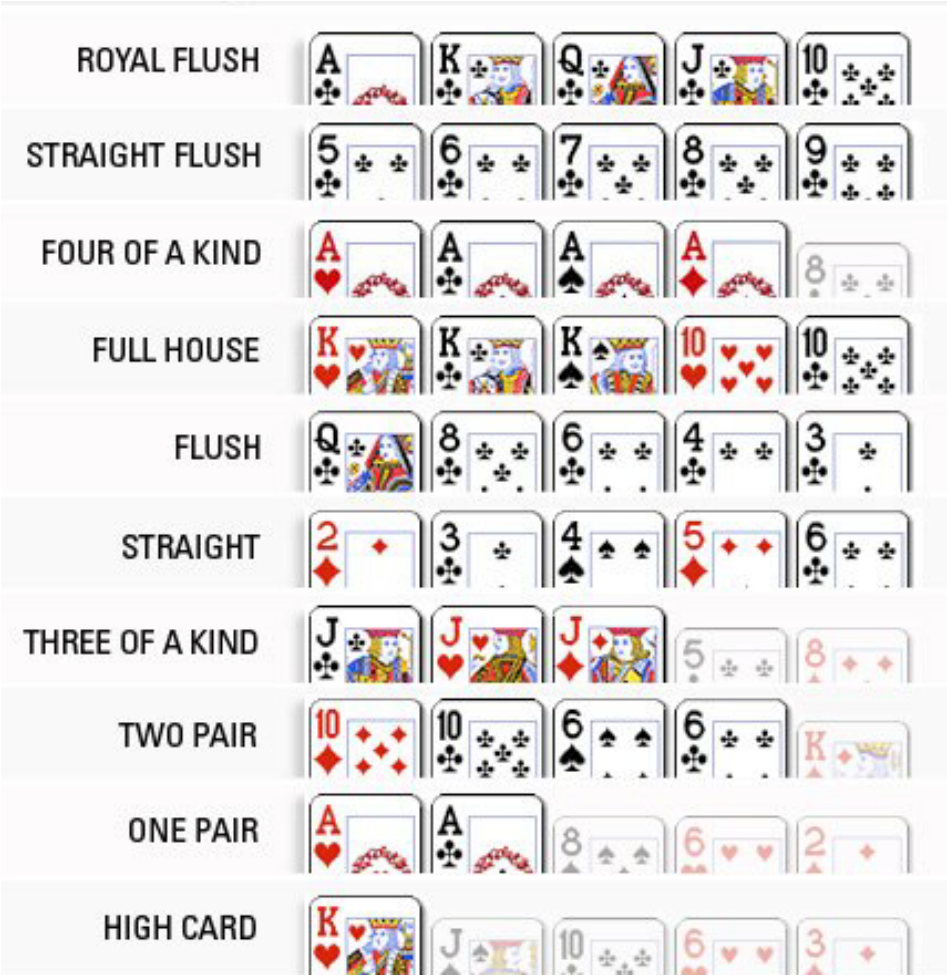
['K♠', 'K❤', '6♠', '7♠', 'Q♣', '7♦', '3♦', '8♣', '4♦', '9♠', '4❤', '4♣', 'A❤', 'J❤', '2❤', '7♣', '10♣', 'K♣', 'K♦', 'J♣', '8♦', '5♠', '2♠', 'Q♠', 'A♠', '3♠', '8♠', '5♦', '4♠', '6♣', '7❤', '6❤', '9♦', '10♠', '10❤', '2♦', 'A♣', '8❤', 'Q♦', '2♣', '10♦', '6♦', '9♣', 'Q❤', '5❤', '9❤', 'A♦', '3♣', '5♣', '3❤', 'J♠', 'J♦']

```
def deal_round(players, cards):
    deck = [ ]
    manufacture(deck)
    shuffle(deck)
    for player in range(players):
        hand = deal_hand(deck, cards)
        report(player, hand)
```

```
def deal_hand(deck, cards):
    hand =[ ]
    for card in range(cards):
        top_card = deck.pop(0)
        hand.append(top_card)
    return hand

def report(player, hand):
    print('Player',player,': ',end='')
    for card in range(len(hand)):
        print(f'{hand[card]:4}',end='')
    print()
```

```
********* Round 1 ********
Player 0 : K♠  K♥  6♠  7♠  Q♣
Player 1 : 7♦  3♦  8♣  4♦  9♠
Player 2 : 4♥  4♣  A♥  J♥  2♥
Player 3 : 7♣  10♣ K♣  K♦  J♣
Player 4 : 8♦  5♠  2♠  Q♠  A♠
Player 5 : 3♠  8♠  5♦  4♠  6♣
Player 6 : 7♥  6♥  9♦  10♠ 10♥
Player 7 : 2♦  A♣  8♥  Q♦  2♣

********* Round 2 ********
Player 0 : 9♥  4♥  3♥  2♦  10♠
Player 1 : J♣  7♦  K♦  6♠  K♣
Player 2 : 7♠  7♣  9♦  K♠  5♣
Player 3 : 8♦  10♣ J♦  Q♠  8♠
Player 4 : 5♠  4♣  5♦  8♥  2♣
Player 5 : 6♦  A♣  Q♥  8♣  J♠
Player 6 : 4♠  4♦  3♣  Q♣  5♥
Player 7 : Q♦  10♥ A♦  9♣  2♠

********* Round 3 ********
Player 0 : J♦  A♣  6♠  2♠  J♠
Player 1 : 9♠  Q♦  3♥  J♣  8♣
Player 2 : 4♠  7♣  3♦  K♦  5♦
Player 3 : 9♣  8♦  4♥  2♥  10♥
Player 4 : J♥  6♦  10♣ 4♣  3♣
Player 5 : 5♠  9♦  2♣  5♣  7♥
Player 6 : 8♥  A♥  10♠ 6♥  10♦
Player 7 : 4♦  A♦  7♠  8♠  3♠
```

What if we want the program to score each hand of cards as follows ... ?



```
def deal_round(players, cards):
    deck = [ ]
    manufacture(deck)
    shuffle(deck)
    for player in range(players):
        hand = deal_hand(deck, cards)
        report(player, hand)
```

How can we add a scoring of each hand?

```
def deal_hand(deck, cards):
    hand =[ ]
    for card in range(cards):
        top_card = deck.pop(0)
        hand.append(top_card)
    return hand

def report(player, hand):
    print('Player',player,': ',end='')
    for card in range(len(hand)):
        print(f'{hand[card]:4}',end='')
    print()
```

```
********* Round 1 ********
Player 0 :  J♥  9♠  5♦  A♦  5♣      Pair
Player 1 : 10♠  8♦  2♦  Q♣  6♥      High Card
Player 2 :  3♣  3♥  5♥  2♥  3♠      Three of a Kind
Player 3 :  5♠  4♦  7♠  9♣  4♠      Pair
Player 4 :  Q♥  J♦  Q♦  A♣  4♥      Pair
Player 5 :  J♣  6♦  6♠  K♥ 10♣      Pair
Player 6 :  2♣  8♣  K♣  2♣  6♣      Flush
Player 7 :  8♠  9♦  8♣  7♥  K♦      Pair
********* Round 2 ********
Player 0 :  5♣  4♥ 10♠  A♥  2♥      High Card
Player 1 :  J♣  3♣  Q♦  K♥  Q♣      Pair
Player 2 : 10♥  2♦ 10♣  K♠  2♣      Two Pair
Player 3 :  Q♥  7♣  4♠  5♥  J♠      High Card
Player 4 :  4♣  9♦  8♠  8♥  3♥      Pair
Player 5 :  A♣ 10♦  3♦  9♠  A♦      Pair
Player 6 :  A♠  7♥  7♠  J♥  8♣      Pair
Player 7 :  6♠  7♣  9♥  8♠ 10♠      Straight
********* Round 3 ********
Player 0 :  3♠  9♥  8♦  Q♦  8♣      Pair
Player 1 :  J♦  4♥  J♣  6♦ 10♠      Pair
Player 2 :  Q♠  9♦  7♦  J♥  2♦      High Card
Player 3 :  2♣  4♠  Q♣  3♥  5♣      High Card
Player 4 :  6♣  3♣  5♦  3♦ 10♦      Pair
Player 5 :  A♠  2♥  9♣  5♠  7♠      High Card
Player 6 :  6♥ 10♥  K♦  K♠  K♥      Three of a Kind
Player 7 :  4♦  J♠  6♠  4♣  A♣      Pair
```

```
def deal_round(players, cards):
    deck = [ ]
    manufacture(deck)
    shuffle(deck)
    for player in range(players):
        hand = deal_hand(deck, cards)
        report(player, hand)
```

How can we add a scoring of each hand?

deck

['A♣', '2♣', '3♣', '4♣', '5♣', '6♣', '7♣', '8♣', '9♣', '10♣', 'J♣', 'Q♣', 'K♣', 'A♦', '2♦', '3♦', '4♦', '5♦', '6♦', '7♦', '8♦', '9♦', '10♦', 'J♦', 'Q♦', 'K♦', 'A♥', '2♥', '3♥', '4♥', '5♥', '6♥', '7♥', '8♥', '9♥', '10♥', 'J♥', 'Q♥', 'K♥', 'A♠', '2♠', '3♠', '4♠', '5♠', '6♠', '7♠', '8♠', '9♠', '10♠', 'J♠', 'Q♠', 'K♠']

```
def manufacture(deck):
    suitNames = ('♣','♦','♥','♠')
    valueNames = ('A','2','3','4','5','6','7','8','9','10','J','Q','K')
    for suit in range(4):
        for value in range(13):
            deck.append(value, suit, valueNames[value]+suitNames[suit])
```

```
def shuffle(deck):
    for swap in range(10000):
        x = random.randint(0,51)
        y = random.randint(0,51)
        deck[x], deck[y] = deck[y], deck[x]
```

['K♠', 'K♥', '6♠', '7♠', 'Q♣', '7♦', '3♦', '8♣', '4♦', '9♠', '4♥', '4♣', 'A♥', 'J♥', '2♥', '7♣', '10♣', 'K♣', 'K♦', 'J♣', '8♦', '5♠', '2♠', 'Q♠', 'A♠', '3♠', '8♠', '5♦', '4♠', '6♣', '7♥', '6♥', '9♦', '10♠', '10♥', '2♦', 'A♣', '8♥', 'Q♦', '2♣', '10♦', '6♦', '9♣', 'Q♥', '5♥', '9♥', 'A♦', '3♣', '5♣', '3♥', 'J♠', 'J♦']

```
def deal_round(players, cards):
    deck = [ ]
    manufacture(deck)
    shuffle(deck)
    for player in range(players):
        hand = deal_hand(deck, cards)
        report(player, hand)
```

How can we add a scoring of each hand?

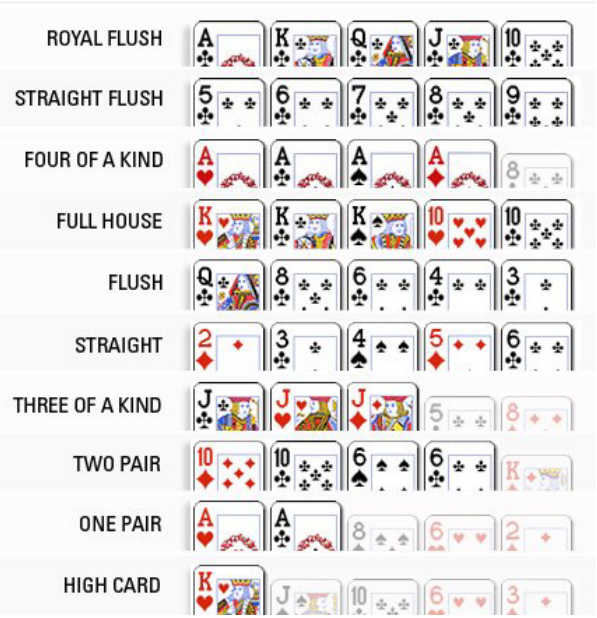
```
def deal_hand(deck, cards):
    hand =[ ]
    for card in range(cards):
        top_card = deck.pop(0)
        hand.append(top_card)
    return hand

def report(player, hand):
    print('Player',player,': ',end='')
    for card in range(len(hand)):
        print(f'{hand[card][2]:4}',end='')
    print(f'{score(hand):>18}')
```

```
********* Round 1 ********
Player 0 :  J♥  9♠  5♦  A♦  5♣        Pair
Player 1 : 10♠  8♦  2♦  Q♣  6♥        High Card
Player 2 :  3♣  3♥  5♥  2♥  3♠        Three of a Kind
Player 3 :  5♠  4♦  7♠  9♣  4♠        Pair
Player 4 :  Q♥  J♦  Q♦  A♣  4♥        Pair
Player 5 :  J♣  6♦  6♠  K♥ 10♣        Pair
Player 6 :  2♣  8♣  K♣  2♣  6♣        Flush
Player 7 :  8♠  9♦  8♣  7♥  K♦        Pair
********* Round 2 ********
Player 0 :  5♣  4♥ 10♠  A♥  2♥        High Card
Player 1 :  J♣  3♣  Q♦  K♥  Q♣        Pair
Player 2 : 10♥  2♦ 10♣  K♠  2♣        Two Pair
Player 3 :  Q♥  7♣  4♠  5♥  J♠        High Card
Player 4 :  4♣  9♦  8♠  8♥  3♥        Pair
Player 5 :  A♣ 10♦  3♦  9♠  A♦        Pair
Player 6 :  A♠  7♥  7♠  J♥  8♣        Pair
Player 7 :  6♠  7♣  9♥  8♠ 10♠        Straight
********* Round 3 ********
Player 0 :  3♠  9♥  8♦  Q♦  8♣        Pair
Player 1 :  J♦  4♥  J♣  6♦ 10♠        Pair
Player 2 :  Q♠  9♦  7♦  J♥  2♦        High Card
Player 3 :  2♣  4♠  Q♣  3♥  5♣        High Card
Player 4 :  6♣  3♣  5♦  3♦ 10♦        Pair
Player 5 :  A♠  2♥  9♣  5♠  7♠        High Card
Player 6 :  6♥ 10♥  K♦  K♠  K♥        Three of a Kind
Player 7 :  4♦  J♠  6♠  4♣  A♣        Pair
```

ordhand

[[0, 0, 'A♣'], [0, 2, 'A♥'], [3, 3, '4♠'], [5, 0, '6♣'], [12, 2, 'K♥']]

```
def score(hand):
    ordhand = sorted(hand)
    ace_straight =    test_ace_straight(ordhand)
    straight = test_straight(ordhand)
    flush = test_flush(ordhand)

    if ace_straight and flush:
        score = 'Royal Flush'
    elif straight and flush:
        score = 'Straight Flush'
    elif straight or ace_straight:
        score = 'Straight'
    elif flush:
        score = 'Flush'
...
```

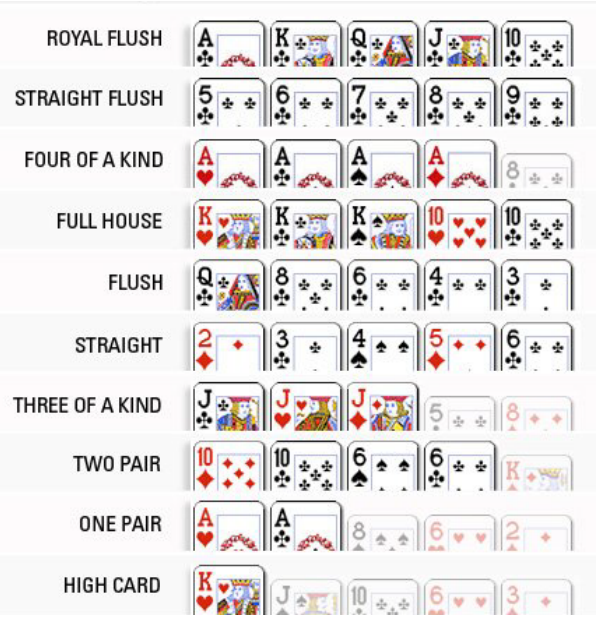


```
def test_flush(hand):
    flush = True
    for card in range(len(hand)-1):
        if hand[card][1] != hand[card+1][1] :
            flush = False
    return flush
```

ordhand

[[0, 0, 'A♣'], [0, 2, 'A♥'], [3, 3, '4♠'], [5, 0, '6♣'], [12, 2, 'K♥']]

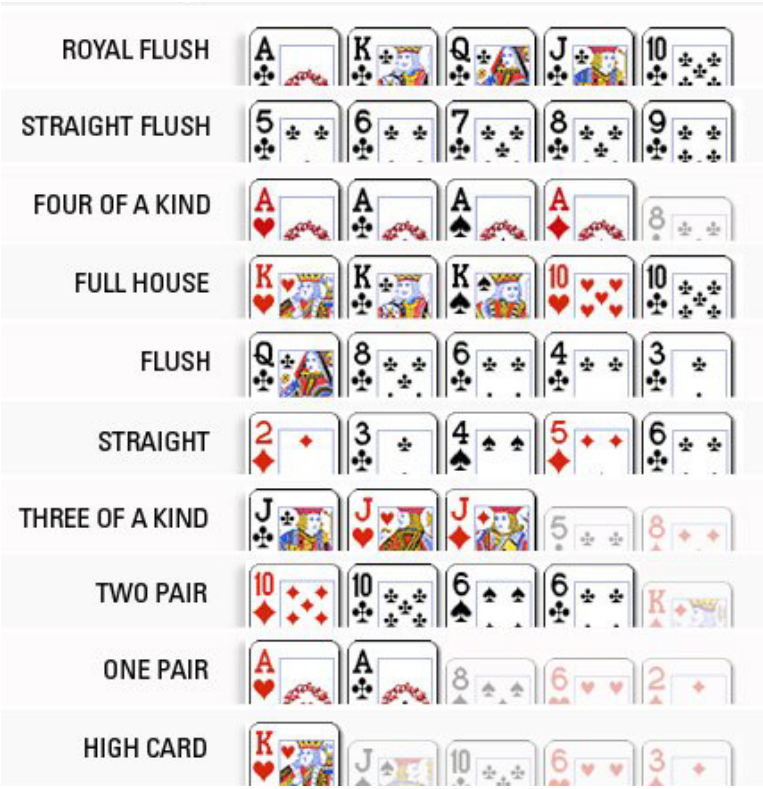
```
def score(hand):
    ordhand = sorted(hand)
    ace_straight =    test_ace_straight(ordhand)
    straight = test_straight(ordhand)
    flush = test_flush(ordhand)

    if ace_straight and flush:
        score = 'Royal Flush'
    elif straight and flush:
        score = 'Straight Flush'
    elif straight or ace_straight:
        score = 'Straight'
    elif flush:
        score = 'Flush'
...
```




```
def test_ace_straight(hand):
    return [hand[0][0],hand[1][0],hand[2][0],hand[3][0],hand[4][0]] == [0,9,10,11,12]

def test_straight(hand):
    straight = True
    for card in range(len(hand)-1):
        if hand[card+1][0] - hand[card][0] != 1:
            straight = False
    return straight
```

How to score four of a kind, three of a kind, full house, two pair, pair ?

Compare each card with every other card and count matche of same values.



6 matches

4 matches

3 matches

2 matches

1 match

0 matches

```
def count_matches(hand):
    count = 0
    for c1 in hand:
        for c2 in hand:
            if c1[0] == c2[0]:
                count += 1
    return (count - len(hand))/2
```

c2

c1	10	10	6	6	K
10	○	●			
10	●	○			
6			○	●	
6			●	○	
k					○

```
def score(hand):
    ordhand = sorted(hand)
    ace_straight = test_ace_straight(ordhand)
    straight = test_straight(ordhand)
    flush = test_flush(ordhand)
    matches = count_matches(ordhand)

    if ace_straight and flush:
        score = 'Royal Flush'
    elif straight and flush:
        score = 'Straight Flush'
    elif straight or ace_straight:
        score = 'Straight'
    elif flush:
        score = 'Flush'
    elif matches == 6:
        score = 'Four of a Kind'
    elif matches == 4:
        score = 'Full House'
    elif matches == 3:
        score = 'Three of a Kind'
    elif matches == 2:
        score = 'Two Pair'
    elif matches == 1:
        score = 'Pair'
    else:
        score = 'High Card'
    return score
```