Question 1. [6 marks]

A deck of cards has 52 cards. Each card has a suit and a rank. The suits are Clubs, Diamonds, Hearts, and Spades. The ranks are Ace, 2, 3, ..., 9, 10, Jack, Queen, and King.

Blackjack is a game that starts off with each player getting (being dealt) one card face down and one card face up. In this question, you will be reading the help for classes `Card` and `Deck` (which are in modules `card` and `deck`, respectively), and writing code that uses those two classes.

```
class Card:
 |  A card with a rank and a suit that can be
 |  either face up or face down.
 |
 |  Methods defined here:
 |
 |  __init__(self, s, r)
 |      A new face-down Card with suit str s
 |      and rank str r.
 |
 |  turn_over(self)
 |      Flip this card over.
 |
 |  value(self)
 |      Return this card's rank and suit as a
 |      str, or "XX" if this card is face down.

class Deck:
 |  A deck of cards.
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      A new deck with 52 cards.
 |
 |  get_next_card(self)
 |      Remove and return this Deck's next card.
 |
 |  shuffle(self)
 |      Randomly rearrange the cards.
 |
 |  ----------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  RANKS = ['A', '2', '3', '4', '5', '6', '7', '8',
 |      '9', 'T', 'J', 'Q', 'K']
 |
 |  SUITS = ['C', 'D', 'H', 'S']
```

Fill in the missing code in the spaces provided.

```
import deck
import card

def print_cards(p):
    '''Print player p's cards.'''

    for card in p:
        print card.value(),
    print

def deal(deck, p):
    '''Deal one Card face-down and once Card face-up from the deck into list of
    Cards p. The face-up card should be the last item in p.'''

    p.append(deck.get_next_card())
    p.append(deck.get_next_card())
    p[-1].turn_over()


if __name__ == '__main__':
    # Create a deck and shuffle it.

    deck = deck.Deck()
    deck.shuffle()

    # The cards for the two players.  The top card is the last card in the list.
    p1_cards = []
    p2_cards = []

    # Put one face-down and one face-up card into each player's list.

    deal(deck, p1_cards)
    deal(deck, p2_cards)

    # Print the cards for each player.
    print_cards(p1_cards)
    print_cards(p2_cards)
```

Question 2. [12 MARKS]

Complete the following functions according to their docstring descriptions.

Part (a) [6 MARKS]

```
def input_to_dict(r):
    '''Given a reader r that contains lines of the form
    INSTRUCTOR:COURSE:YEAR
    return a dictionary in which each key is an INSTRUCTOR (str) and each value
    is a list of (str, int) tuples where each tuple is of the form (COURSE,
    YEAR) representing a course the instructor has taught and the year they
    taught the course.'''

    teaching_dict = {}

    for line in r:
        line_list = line.strip().split(':')
        if not line_list[0] in teaching_dict:
            teaching_dict[line_list[0]] = []
        teaching_dict[line_list[0]].append((line_list[1], line_list[2]))

    return teaching_dict
```

Part (b) [6 marks]

```
def dict_to_file(fname, d):
    '''Create a file named fname with the contents of dictionary d.  Each key is
    a float and each value is a list of strings.  For each key in d, the file
    should contain this:

    key:
      str1
      str2
      ...
      strN

    N is the number of strings in the list of values corresponding with that
    key. Values are indented 2 spaces.'''

    f = open(fname, 'w')

    for key in d:
        f.write(str(key) + ':\n')
        for string in d[key]:
            f.write('  ' + string + '\n')

    f.close()
```

Question 3. [12 marks]

Read the docstring for function `update_list_with_dict`.

```
def update_list_with_dict(d, L):
    '''Modify list L by replacing each element in L that appears as a key in
    dict d with the value associated with that element. If the list element is
    not a key in d, it is unchanged.'''
```

Part (a) [4 marks] For each of the following values for `d` and `L`, what is the value of `L` after `update_list_with_dict` returns?

d	original L	modified L
{}	[]	
{1:2}	[7]	
{1:2}	[1]	
{1:2}	[2]	
{1:2, 3:4, 5:6}	[1, 5, 7, 5]	

Part (b) [4 marks] Write the body of function `update_list_with_dict`:

```
    i = 0
    while (i < len(L)):
        if L[i] in d:
            L[i] = d[L[i]]
        i = i + 1
```

Part (c) [4 marks]

Modify your function to return a new, updated list rather than updating the input list in place. Write a complete solution, including the function header and docstring, below.

```
newL = []
for elem in L:
    if elem in d:
        newL.append(d[elem])
    else:
        newL.append(elem)
return newL
```

Question 4. [9 marks]

In the table below, trace the variable values during execution of the function call:

```
mystery('erie')
```

For each blank in the table, fill in the specified variable's value **after** the corresponding line has executed. Write NR ("not reached") if that line was not executed. **The loop will iterate between 1 and 4 times; fill in only as many iterations as necessary.** On the last line, write the value returned by the function call.

```
def mystery(x):
  i = 0

  while i < len(x) / 2:
    a = x[:len(x)-i]
    b = x[len(x)-i:]
    if x[i] != a[-1]:
      x = a + x[i] + b
    i = i + 1
  return x
```

Show variable values after each line has executed:			
`x = 'erie'`			
i: 0			
During 1st iteration	During 2nd iteration	During 3rd iteration	During 4th iteration
x: 'erie' i: 0	x: 'erie' i: 1	x: 'erire' i: 2	x: NR i: NR
a: 'erie'	a: 'eri'	a: NR	a: NR
b: ''	b: 'e'	b: NR	b: NR
x[i]: 'e' a[-1]: 'e'	x[i]: 'r' a[-1]: 'i'	x[i]: NR a[-1]: NR	x[i]: NR a[-1]: NR
x: NR	x: 'erire'	x: NR	x: NR
i: 1	i: 2	i: NR	i: NR
value returned: 'erire'			

Question 5. [8 MARKS]

Complete the following function according to its docstring description. *Hint:* You may find it useful to draw a diagram that describes the relationship between the center point (x, y) and the parameters h and w.

```
def spread_green(pic, x, y, h, w):
    '''Modify Picture pic so that all pixels in the specified rectangle have the
    same green value as the green value of the pixel at (x, y).  The rectangle is
    centred on the point (x, y), and h and w respectively represent the height
    and width of the rectangle.  h and w will always be odd.'''

    this_green = get_green(get_pixel(pic, x, y))

    # start at r pixels to the left of x and t pixels above y,
    # and go to r pixels to the right of x and t below y:

    for i in range(x - (h / 2), x + (h / 2) + 1):
        for j in range(y - (t / 2), y + (t / 2) + 1):
            set_green(get_pixel(pic, i, j), this_green)
```

Question 6. [10 marks]

Online driving directions, such as what Google Maps and Mapquest produce, give directions that look like this (these directions are from the Bahen Centre to Mezzetta):

```
1. Head south on St George St toward College St  0.2 km
2. Turn right at College St                       1.0 km
3. Turn right at Bathurst St                      2.9 km
4. Turn left toward Vaughan Rd                    0.0 km
5. Slight right at Vaughan Rd                     0.2 km
6. Turn left at St Clair Ave W                    0.4 km
```

Write a function `total_distance(r)` that reads directions in this format from reader `r`, adds up the kilometers, and returns that sum. Do not include a docstring.

```
def total_distance(r):
sum = 0
for line in r:
    parts = line.split()
    sum += float(parts[-2])
return sum
```

Question 7. [5 MARKS]

Consider the following function which contains **one** bug. The numbers on the left are line numbers to be used to answer the questions below.

```
1 def count_chars(s, L):
2    '''Given a string s and a list of distinct characters L, return a dictionary whose
3    keys are the characters in L and whose values are the frequencies, in s, of each key.
4    If the character does not appear in s, its frequency will be 0.'''
5
6    freq_dict = {}
7    for c in s:
8        freq_dict[c] = 0
9
10    for c in s:
11        if freq_dict.has_key(c):
12            freq_dict[c] += 1
13
14    return freq_dict
```

The test case below fails, giving you some information about the error. **Note:** Be sure to read the `docstring` to learn exactly how the function is supposed to behave.

```
def test_count_1():
    test_string = 'ccccccccc'
    test_list = ['a', 'b', 'c']
    test_dict = count_chars(test_string, test_list)
    assert test_dict['a'] == 0, "char in L not in string s."
```

Part (a) [2 MARKS] What is the result of running the test case?

The dictionary has no key for `'a'`, so there will be an exception (KeyError...).

Part (b) [3 MARKS]

How would you change the function to fix the bug? Please **describe precisely** which line(s) you wish to delete or replace and where you would insert new code. (For example: "delete line 4", "insert the following lines after line 17", etc.) In line 7, change the "s" to "L".

Question 8. [7 MARKS]

The following questions ask you to write code that you might need while developing a Tkinter GUI.

Part (a) [2 MARKS] Complete the following function according to its docstring description.

```
def set_Tkvar_list(tkvar_list, value_list):
    '''Set the contents of each StringVar in tkvar_list to the item at the same
    index in value_list.  Assume tkvar_list and value_list have the same
    length and that they contain only StringVars and strs, respectively.'''

    for i in range(len(value_list)):
        tkvar_list[i].set(value_list[i])
```

Part (b) [3 MARKS]

Write code that creates a Tkinter `Button` labeled with the text ``Set Record'' into the `Frame` named `frame`. When pressed, the button you create should set the values in the list of `StringVars` `record_names` to the values in the list `record_data`. You may use the function `set_Tkvar_list` from above.

```
    set_vals_cmd = lambda: set_Tkvar_list(label_list, record_data)
    button = Tkinter.Button(frame, text="Set Record", command=set_vals_cmd)
    button.pack()
```

Part (c) [2 MARKS]

Write code that creates a Tkinter `Label` that displays the contents of the `StringVar` `record_name`. Place the label at row 3 and column 4 of the `Frame` named `record_frame`.

```
    record_label = Tkinter.Label(record_frame, textvariable=record_name)
    record_label.grid(row=3, column=4)
```

Question 9. [8 marks]

A *palindrome* is a string that reads the same both forwards and backwards. Examples are `'toot'`, `'radar'`, and `'moogah hagoom'`.

Complete the following function according to its docstring. For full marks, use only one return statement, zero if statements, and write a while loop that exits as soon as you find that the string is not a palindrome.

```
def is_palindrome(s):
    '''Return True if s is a palindrome, and return False otherwise.'''

    i = 0
    j = len(s) - 1
    while i < j and s[i] == s[j]:
        i += 1
        j -= 1

    return i >= j
```

Question 10. [4 MARKS]

Complete the function `is_ancestor` on the following page.

This question is tricky. You have these choices:

- Leave the question blank for zero marks,
- Write "I do not know how to answer this question." in the space provided and earn 1 of the 4 marks.
- Answer the question and understand that we will be strict with the marking (you'll need to get it mostly right to get any marks).

Consider a dictionary that has names of parents as keys and lists of child names as values. For example, if Barbara is the mother of Matt and Holly, Matt has no children, and Holly and Tom are the mother and father of Chris and Bekkah, then this dictionary captures that information:

```
{'Holly': ['Bekkah', 'Chris'],
 'Matt': [],
 'Barbara': ['Matt', 'Holly'],
 'Tom': ['Bekkah', 'Chris'],
 'Chris': [],
 'Bekkah': []}
```

Notice that Matt, Chris, and Bekkah are keys in the dictionary even though they don't have children.

For this question, we say that person `p` is the *ancestor* of these people:

- `p`
- `p`'s children
- `p`'s children's children
- `p`'s children's children's children
- etc.

Hint: keep a list of people that you've seen (starting with `p1`), and add their descendants to it.

```
def is_ancestor(p1, p2, d):
    '''Return True if p1 is an ancestor of p2 in family tree dictionary d.

    p1 and p2 are names of people (strs).'''

    L = [p1]
    found = False
    while not found and len(L) != 0:
        next = L.pop()
        for child in d[next]:
            L.append(child)

        found = next == p2

    return next == p2
```

Short Python function/method descriptions:

```
__builtins__:
  lambda: expr -> function
    Returns a function that evaluates the Python expression expr.
  len(x) -> integer
    Return the length of the list, tuple, dict, or string x.
  max(L) -> value
    Return the largest value in L.
  min(L) -> value
    Return the smallest value in L.
  open(name[, mode]) -> file object
    Open a file.  Legal modes are "r" (read), "w" (write), and "a" (append).
  range([start], stop, [step]) -> list of integers
    Return a list containing the integers starting with start and ending with
    stop - 1 with step specifying the amount to increment (or decrement).
    If start is not specified, the list starts at 0.  If step is not specified,
    the values are incremented by 1.
Color:
  black
    RGB: 0, 0, 0
  Color(red, green, blue) -> Color
    Define a Color with the specified red, green, and blue components.
  blue
    RGB: 0, 0, 255
  green
    RGB: 0, 255, 0
  red
    RGB: 255, 0, 0
  white
    RGB: 255, 255, 255
dict:
  D.get(k) -> value
    Return the value associated with the key k in D.
  D.has_key(k) -> boolean
    Return True if k is a key in D and False otherwise.
  D.keys() -> list of keys
    Return the keys of D.
  D.values() -> list of values
    Return the values associated with the keys of D.
file (also called a "reader"):
  F.close()
    Close the file.
  F.read([size]) -> read at most size bytes, returned as a string.
    If the size argument is negative or omitted, read until EOF (End
    of File) is reached.
  F.readline([size]) -> next line from the file, as a string. Retain newline.
    A non-negative size argument limits the maximum number of bytes to return (an incomplete
    line may be returned then).  Return an empty string at EOF.
float:
  float(x) -> floating point number
    Convert a string or number to a floating point number, if possible.
int:
  int(x) -> integer
    Convert a string or number to an integer, if possible.  A floating point
    argument will be truncated towards zero.
list:
  L.append(x)
    Append x to the end of the list L.
  L.index(value) -> integer
    Returns the lowest index of value in L.
  L.insert(index, x)
    Insert x at position index.
  L.remove(value)
    Removes the first occurrence of value from L.
  L.sort()
    Sorts the list in ascending order.
media:
  choose_file() -> str
    Prompt user to pick a file. Return the path to that file.
  create_picture(int, int) -> Picture
    Given a width and a height, return a Picture with that width and height.  All pixels are white.
  get_blue(Pixel) -> int
    Return the blue value of the given Pixel.
  get_color(Pixel) -> Color
    Return the Color object with the given Pixel's RGB values.
  get_green(Pixel) -> int
    Return the green value of the given Pixel.
  get_pixel(Picture, int, int) -> Pixel
    Given x and y coordinates, return the Pixel at (x, y) in the given Picture.
  get_red(Pixel) -> int
    Return the red value of the given Pixel.
  load_picture(str) -> Picture
    Return a Picture object from file with the given filename.
  set_blue(Pixel, int)
    Set the blue value of the given Pixel to the given int value.
  set_color(Pixel, Color)
    Set the RGB values of the given Pixel to those of the given Color.
  set_green(Pixel, int)
      Set the green value of the given Pixel to the given int value.
  set_red(Pixel, int)
    Set the red value of the given Pixel to the given int value.
  show(Picture)
    Display the given Picture.
str:
  str(x) -> string
    Convert an object into its string representation, if possible.
  S.find(sub[,i]) -> integer
    Return the lowest index in S (starting at S[i], if i is given) where the
    string sub is found or -1 if sub does not occur in S.
  S.index(sub) -> integer
    Like find but raises an exception if sub does not occur in S.
  S.isdigit() -> boolean
    Return True if all characters in S are digits and False otherwise.
  S.replace(old, new) -> string
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
  S.rstrip([chars]) -> string
    Return a copy of the string S with trailing whitespace removed.
    If chars is given and not None, remove characters in chars instead.
  S.split([sep]) -> list of strings
    Return a list of the words in S, using string sep as the separator and
    any whitespace string if sep is not specified.
  S.strip() -> string
    Return a copy of S with leading and trailing whitespace removed.
Tkinter:
  Button:
    Button(parent, [text=],[textvariable=], [command=]) -> button object
      Construct a button with the given parent.
  Entry:
    Entry(parent) -> entry object
      Construct an entry field with the given parent.
    E.get() -> string
      Return the text in the entry field E.
  Frame:
    Frame(parent) -> frame object
      Construct a frame widget with the given parent.
  Label:
    Label(parent, [text=], [textvariable=]) -> label object
      Construct a label with the given parent.
  TkVariables:
    DoubleVar() -> Tk float variable object
      Construct a Tkinter float variable.
    IntVar() -> Tk int variable object
      Construct a Tk int variable.
    StringVar() -> Tk string variable object
      Construct a Tk string variable.
    tkvar.get() -> int
      Return the value of tkvar.
    tkvar.set(value)
      Set tkvar to value.
  Window:
    Tk() -> window
      Return a new window widget.
  All Widgets:
    A.grid(row=r, column=c)
      Place widget A in row r and column c.
    A.pack()
      Place widget A below the last widget that was packed.
```