Question 1. [10 marks]

Below are five segments of code. Each one runs without error. To the right of each segment, show the output it generates.

```
L = [1, 2, 3, 4]
for item in L:
    item = item * 5
print L
```

Output:

```
L = [["a", "b"], ["c", "d"]]
for item in L:
    item.append("0")
print L
```

Output:

```
s = "pizzapizza"
count = 0
i = 1
while i < len(s):
    if s[i] > "m":
        count = count + 1
    i = i + 1
print count
```

Output:

```
def change(s):
    s = s + "!"

word = "holiday"
change(word)
print word
```

Output:

```
def alter(d):
    d["Dec"] = "vacation!"

months = {"Jan": "school"}
alter(months)
print months
```

Output:

Solution:

```
[1, 2, 3, 4]
[['a', 'b', '0'], ['c', 'd', '0']]
5
holiday
{'Jan': 'school', 'Dec': 'vacation!'}
```

Question 2. [6 MARKS]

Consider the following code fragments:

```
# Code fragment 1
for char in s:
    print char
for char in s:
    print char + char
for char in s:
    print char + char + char
for char in s:
    print char + char + char + char
for char in s:
    print char + char + char + char + char
```

```
# Code fragment 2
for char_1 in s:
    for char_2 in s:
        for char_3 in s:
            print char_1 + char_2 + char_3
```

Part (a) [1 MARK]

If we assign the value `"bumble"` to `s` and execute code fragment 1, how many lines of output will be produced?

Part (b) [2 MARKS]

In general, for a string of length n, how many lines of output will be produced by code fragment 1?

Part (c) [1 MARK]

If we assign the value `"bumble"` to `s` and execute code fragment 2, how many lines of output will be produced?

Part (d) [2 MARKS]

In general, for a string of length n, how many lines of output will be produced by code fragment 2?

Solution:

(a) 5×6 or 30 (either answer is fine). (b) $5n$. (c) 6^3 or 216 (either answer is fine). (d) n^3.

Question 3. [8 MARKS]

Part (a) [5 MARKS]

Write the function `unique_values` according to its docstring:

```
def unique_values(d):
    '''Return a sorted list containing all the values (not keys) in dict d.
    Even if a value occurs more than once in d, include it only once in the
    list.'''
```

Solution:

```
    L = []
    for k in d:
        if d[k] not in L:
            L.append(d[k])
    L.sort()
    return L
```

Part (b) [3 MARKS]

Complete the table below by adding three distinct test cases for `unique_values`. For each, provide a specific value for `d`, the expected result, and the purpose of the test case. We have provided one test case as an example. All of the test cases should be different from each other, and each should test something significant. You will receive no credit for repetitive test cases. Do not include tests that check for invalid input.

Value of `d`	Expected result	Purpose of this test case
`{}`	`[]`	empty dictionary

Solution:

Value of `d`	Expected result	Purpose of this test case
{}	`[]`	empty dictionary
`{'apple' : 'mac'}`	['mac']	one entry
`{1 : 2, 3 : 2, 4 : 1, -1 : -1, 76 : 8}`	`[-1, 1, 2, 8]`	several entries, with dups
`{1 : 1, 2 : 2, 3 : 3, 4 : 4}`	`[1, 2, 3, 4]`	several entries with no dups

Question 4. [8 MARKS]

One way to put a list of people into teams is to "count them off". For example, if you want 4 teams, you go through the list of people, assigning them, in order, to teams 0, 1, 2, 3, 0, 1, 2, 3, etc. The function below forms teams in this manner. For example, if the list of people is:

```
["paul", "francois", "andrew", "sue", "steve", "arnold", "tom",
"danny", "nick", "anna", "dan", "diane", "michelle", "jeremy",
"karen"]
```

and the number of teams desired is four, it should return:

```
[['paul', 'steve', 'nick', 'michelle'], ['francois', 'arnold', 'anna', 'jeremy'],
['andrew', 'tom', 'dan', 'karen'], ['sue', 'danny', 'diane']]
```

Write the body of the function. Do **not** assume that the list of people will divide evenly into the desired number of teams. If the list of people does not divide evenly into the desired number of teams, the teams will simply have unequal sizes. Do **not** assume that the number of people is at least as large as the number of teams. If there aren't enough people to cover the teams, some (or even all) teams will simply be empty. **Hint:** The `range` function will be very helpful.

```
def form_teams(people, num_teams):
    '''Make num_teams teams out of the names in list people by counting off.
    people is a list of people's names (strs) and num_teams (an int >= 1) is
    the desired number of teams. Return a list of lists of names, each sublist
    representing a team.'''
```

Solution:

```
    L = []
    for i in range(num_teams):
        L.append([])
    for team in range(num_teams):
        for i in range(0, len(people), num_teams):
            if (team + i) < len(people):
                L[team].append(people[team + i])
    return L
```

Question 5. [12 MARKS]

A "birthday month dictionary" is a dictionary of birthday information with the following structure: the keys are month names (strings, with capitalization) and the values are themselves dictionaries. The keys in each of these sub-dictionaries are integers from 1 to 31 representing days of the month, and the values are lists of strings, which are the names of people with their birthday on that day. For example, the following birthday month calendar says that Catherine's birthday is February 13th, Katie's is May 3rd, and both Peter and Ed were born on May 8th:

```
{"February" : {13 : ["Catherine"]}, "May": {3 : ["Katie"], 8 : ["Peter", "Ed"]}}
```

Part (a) [6 MARKS]

Complete the following function according to its docstring description.

```
def most_covered(bdm):
    '''bdm is a birthday months calendar with at least one key in it. Return
    the name of the month (a str) that is most "covered" in the dictionary.
    I.e., the month with the most days on which someone has a birthday. If
    there is a tie, return any month with that maximum coverage.'''
```

Solution:

```
    months = bdm.keys()
    biggest_month = months[0]
    size_of_biggest_month = len(bdm[biggest_month].keys())

    for month, d in bdm.items():
        if len(d.keys()) > size_of_biggest_month:
            biggest_month = month
            size_of_biggest_month = len(d.keys())

    return biggest_month
```

Part (b) [6 marks]

A "birthday dictionary" also contains information about birthdays, but in a different format. The keys are people's names and the values are 2-item lists describing those people's birthdays. The first item in each list is the name of a month (a str) and the second is the day of a month (an int). For example, the following birthday dictionary has the same content, although in a **different format**, as the birthday month dictionary in part (a). It is **equivalent** to the birthday month dictionary in part (a).

```
{'Ed': ['May', 8], 'Peter': ['May', 8], 'Catherine': ['February', 13], 'Katie': ['May', 3]}
```

Complete the following function according to its docstring description. You may assume that each name string in the dictionary is unique.

```
def invert_birthdays(bdm):
    '''bmd is a birthday month dictionary. Return the equivalent birthday
    dictionary.'''
```

Solution:

```
    answer = {}
    for (month, month_dict) in bdm.items():
        for (day, day_list) in month_dict.items():
            for name in day_list:
                answer[name] = [month, day]
    return answer
```

Question 6. [10 marks]

Part (a) [6 marks]

Write the following function according to its docstring.

```
def read_weather_data(reader):
    '''Return a list containing (int, int) tuples where each tuple is of the
    form (LOW_TEMPERATURE, HIGH_TEMPERATURE). The temperature data is obtained
    from the open reader, which contains lines of the form:
    DATE,PRECIPITATION,HIGH_TEMPERATURE,LOW_TEMPERATURE
    The tuple corresponding to the Nth line in the reader (counting from zero)
    should be located at index N in the list.'''
```

Solution:

```
    temp_data = []
    for line in reader:
        dl = line.strip().split(",")
        temp_data.append((int(dl[3]), int(dl[2])))
    return temp_data
```

Part (b) [4 marks]

Write the following function according to its docstring.

```
def greatest_range(temps):
    '''temps is a list of (int, int) tuples.  Find the tuple with the greatest
    difference between its two elements and return this difference. Assume that
    temps contains at least one element and that the first element of each
    tuple is less than or equal to the second element.'''
```

Solution:

```
    g_r = temps[0][1] - temps[0][0]
    for tup in temps:
        diff = tup[1] - tup[0]
        if diff > g_r:
            g_r = diff
    return g_r
```

Question 7. [12 MARKS]

In this problem, you should call functions from earlier parts of the question in your solutions for later parts. When calling a function, assume that it has been implemented correctly.

Part (a) [4 MARKS]

Complete the following function according to its docstring.

```
def count_chars(s):
    '''Return a dict that contains each character in str s as a key. The
    value associated with each key is the number of times that character
    occurs in s.'''
```

Solution:

```
    d = {}
    for ch in s:
        if ch in d:
            d[ch] += 1
        else:
            d[ch] = 1
    return d
```

Part (b) [5 MARKS]

Complete the following function according to its docstring.
Definition: Two words are *anagrams* if one can be formed by rearranging the letters of the other. A word is considered an anagram of itself.

```
def find_anagrams(word, r):
    '''Return a list containing every str in open reader r that is an anagram of
    str word. Each line in r contains a string with no whitespace.'''
```

Solution:

```
    anagrams = []
    chars = count_chars(word)
    for line in r:
        line = line.strip()
        line_count = count_chars(line)
        if line_count == chars:
            anagrams.append(line)
    return anagrams
```

Part (c) [3 MARKS]

Each line in file `word_list.txt` contains a string with no whitespace. Complete the program below so that it prints the list of all words in file `word_list.txt` after the first word that are anagrams of the first word in the file.

Note that you can pass a reader that has been partially read to a function. The function will simply continue reading from that point.

```
if __name__ == '__main__':
```

Solution:

```
    f = open("word_list.txt", "r")
    comparison_word = f.readline().strip()
    print find_anagrams(comparison_word, f)
```

Question 8. [3 MARKS]

Write the following function according to its docstring. Consider only exponents that are non-negative integers. Recall that $2^0 = 1$.

You must not use a for loop in this question. You must not use `break`, and you must have only one `return` statement.

```
def next_power(n):
    '''Return the next power of two that is greater than n. n is an int >= 0.'''
```

Solution:

```
    power = 1
    while power <= n:
        power = power * 2

    return power
```

Question 9. [10 MARKS]

Consider the following `Card` and `Deck` classes:

```
class Card(__builtin__.object)
 |  A card with a rank and a suit that can be either face up or face down.
 |  The rank is one of deck.Deck.RANKS and the suit is one of
 |  deck.Deck.SUITS.
 |
 |  Methods defined here:
 |
 |  __cmp__(self, other)
 |      Return -1 if this Card is less than Card other, 0 if they are equal,
 |      and 1 if this Card is greater. The comparison is done based on the rank
 |      of the cards.
 |
 |  __init__(self, s, r)
 |      A new face-down Card with suit str s and rank str r.
 |
 |  __str__(self)
 |      Return this Card's rank and suit together as a str, or 'XX' if this
 |      card is face down.
 |
 |  turn_down(self)
 |      Make this card be face down.
 |
 |  turn_over(self)
 |      Flip this card over.

class Deck(__builtin__.object)
 |  A deck of Cards.
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      A new deck with 52 face-down cards.
 |
 |  empty(self)
 |      Return True iff this deck has no cards.
 |
 |  get_next_card(self)
 |      Remove and return this deck's next Card, face down.
 |
 |  shuffle(self)
 |      Randomly rearrange the cards.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  RANKS = ['2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K', 'A']
 |
 |  SUITS = ['C', 'D', 'H', 'S']
```

Part (a) [4 MARKS]

1. Write code to create a `Deck` called `my_deck` and shuffle it.

2. Which method gets called when you create a `Deck`?

Now suppose you have two `Card`s called `my_card` and `your_card`. Without creating any new variables, do the following:

3. Write a line of code that will cause `Card.__cmp__` to be called.

4. Write a line of code that will cause `Card.__str__` to be called.

Solution:

```
    # 1
    d = Deck()
    d.shuffle()

    # 2
    # __init__ is called

    # 3
    print my_card < your_card

    # 4
    print my_card
```

Part (b) [6 MARKS] Complete the following function according to its docstring description.

```
def deal_solitaire(d):
    '''Deal out and return a solitaire game from deck d. A solitaire game is a
    list of 7 lists of cards. The first list has one card in it, the second
    list has two, ..., and the last list has 7 cards in it. In each sublist of
    cards, the last card is face up. All other cards are face down. Assume
    that Deck d has at least enough cards in it to deal out a solitaire
    game.'''
```

Solution:

```
solitaire = []
for p in range(7):
    # Pile p needs p + 1 cards
    pile = []
    for i in range(p + 1):
        pile.append(d.get_next_card())
    pile[p].turn_over()
    solitaire.append(pile)
return solitaire
```

Question 10. [4 marks]

Below are two statements to be executed in the shell. For each one, write Python statements to go before it that will guarantee that the statement will run without error. It doesn't matter what the code does, just that it runs. There are many correct answers for each. Below is an example.

Code:

```
a = p(b % c)
```

Statements to go before this one to guarantee that it will run without error:

```
def p(n):
    return n + 1
b = 45
c = 13
```

Part (a) [2 marks]

Code:

```
n = d[x](96) + 7
```

Statements to go before this one to guarantee that it will run without error:

Part (b) [2 marks]

Code:

```
print blah()[y][z]
```

Statements to go before this one to guarantee that it will run without error:

Solution (there are many correct ones!):

```
# (a)
def hello(n):
    return n + 5
d = {1:hello}
x = 1

# (b)
def blah():
    return [[1,2,3], [2,3,4], [3,4,5]]
y = 2
z = 1
```

Question 11. [5 MARKS]

Don't guess. There is a 1-mark **deduction** for wrong answers on this question. Your minimum total mark on this question is zero.

A local elementary school has library card IDs of the following form: IDs either begin with "s" (for "student") and are exactly 6 characters long, including the "s"; or they begin with "t" (for "teacher") and have no length requirement. The following function is supposed to get a valid ID, but it is missing its while condition.

```
def get_valid_id():
    '''Prompt the user for and return a valid library card ID.'''

    s = raw_input("Enter ID: ")
    while ?????:
        s = raw_input("Try again: ")
    return s
```

Below are some possible conditions for the while loop. For each, indicate whether it will make the function work correctly.

1. Will this condition make the function work correctly?

   `(s[0] != "s") and (len(s) != 6) and (s[0] != "t")`

   Circle one: yes no

2. Will this condition make the function work correctly?

   `not( (s[0] == "s" and len(s) == 6) or (s[0] == "t") )`

   Circle one: yes no

3. Will this condition make the function work correctly?

   `(s[0] != "s" and len(s) != 6) or (s[0] != "t")`

   Circle one: yes no

4. Will this condition make the function work correctly?

   `(s[0] == "s" or len(s) == 6) and (s[0] != "t")`

   Circle one: yes no

5. Will this condition make the function work correctly?

   `not( s[0] == "s" and len(s) == 6 ) and (s[0] != "t")`

Circle one: yes no

Solution:

```
(2) and (5) are ‘‘yes".
The rest are ‘‘no".
```

Question 12. [6 MARKS]

Don't guess. There is a 1-mark **deduction** for wrong answers on this question.

Part (a) [2 MARKS] Suppose that

- we are partly through sorting a list,
- the list currently contains the following: [3, 6, 7, 9, 15, 21, 21, 38, 20, 8, 10, 9, 1, 9]
- and our algorithm guarantees that, at this point, the sublist from position 0 to 7 inclusive is sorted, and will not change again.

Could we be using insertion sort? Circle one.

yes no

Could we be using selection sort? Circle one.

yes no

Part (b) [2 MARKS] Below is a list before sorting, and then after each of several major iterations of a sorting algorithm.

before sorting:	[9, 4, 5, 8, 1, 7, 2, 6, 4]
after 1 major iteration:	[1, 9, 4, 5, 8, 2, 7, 4, 6]
after 2 major iterations:	[1, 2, 9, 4, 5, 8, 4, 7, 6]
after 3 major iterations:	[1, 2, 4, 9, 4, 5, 8, 6, 7]

Which algorithm is being used? Circle one.

bubblesort insertion sort selection sort

Part (c) [2 MARKS] Below is a list before sorting, and then after each of several major iterations of a sorting algorithm.

before sorting:	[9, 4, 5, 8, 1, 7, 2, 6, 4]
after 1 major iteration:	[4, 9, 5, 8, 1, 7, 2, 6, 4]
after 2 major iterations:	[4, 5, 9, 8, 1, 7, 2, 6, 4]
after 3 major iterations:	[4, 5, 8, 9, 1, 7, 2, 6, 4]

Which algorithm is being used? Circle one.

bubblesort insertion sort selection sort

Solution:

```
(a) has a problem in the question.  It does not count towards the exam.

(b) bubblesort
    It can't be insertion sort.  2 could not have advanced so far.
    It can't be selection sort.  Other elements near the end would not have been shuffled.

(c) insertion sort
    It can't be bubblesort.  4 and 6 would have swapped.
    It can't be selection sort.  1 would be at the front after one major iteration.
```

Short Python function/method descriptions:

```
__builtins__:
  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.
dict:
  D[k] --> value
    Return the value associated with the key k in D.
  k in d --> boolean
    Return True if k is a key in D and False otherwise.
  D.get(k) -> value
    Return D[k] if k in D, otherwise return None.
  D.keys() -> list of keys
    Return the keys of D.
  D.values() -> list of values
    Return the values associated with the keys of D.
  D.items() -> list of (key, value) pairs
    Return the (key, value) pairs of D, as 2-tuples.
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:
  x in L --> boolean
    Return True if x is in L and False otherwise.
  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.
str:
  x in s --> boolean
    Return True if x is in s and False otherwise.
  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.lower() -> string
    Return a copy of the string S converted to lowercase.
  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.
  S.upper() -> string
    Return a copy of the string S converted to uppercase.
```