Question 1. [4 MARKS]

For each block of code in this question, write its output in the box below it. If it would generate an error, say so, and give the reason for the error.

Part (a) [1 MARK]

```
D = {}
D['apple'] = 4
D['apple'] = 8
print D
```

```
{'apple' : 8}
```

Part (b) [1 MARK]

```
D = {2 : ['d'], 0 : ['c']}
D[0].append('a')
print D[0]
```

```
['c', 'a']
```

Part (c) [1 MARK]

```
D = {4 : 3, 3 : 2, 2 : 1}
for x in D.values():
    print D[x]
```

Error

Part (d) [1 MARK]

```
D1 = {'a' : 1, 'b' : 2, 'c' : 3}
D2 = {'b' : 2, 'c' : 3, 'a' : 1}
print D1 == D2
```

True

Question 2. [4 MARKS]

For each block of code in this question, write its output in the box below it. If it would generate an error, say so, and give the reason for the error.

Part (a) [1 MARK]

```
L1 = ['yesterday', 'today']
L2 = L1.append('tomorrow')
print L2
```

```
None
```

Part (b) [1 MARK]

```
L = [['on', 'off'], \
     ['go', 'slow', 'stop'], \
     ['up']]
print L[1][2][3]
```

```
p
```

Part (c) [1 MARK]

```
s1 = 'hello'
s2 = s1
s2 = s2 + '-bye'
print s1
```

```
hello
```

Part (d) [1 MARK]

```
L1 = ['hello']
L2 = L1
L2[0] = 'bye'
print L1
```

```
['bye']
```

Question 3. [4 MARKS]

Part (a) [3 MARKS]

Write a good docstring for this function.

```
def mystery(a, b):
    ```(str, str) -> bool
    Return True if every character in b appears in a, and return False otherwise.
    '''

    i = 0
    while b != '' and i < len(a):
        if a[i] in b:
            b = b.replace(a[i], '')
        i += 1

    return b == ''
```

### Part (b) [1 MARK]

Provide sample arguments for a call to function `mystery(a, b)` and give the value returned.

| Value of `a` | Value of `b` | Return value |
|---|---|---|
| | | |

There are many possible answers. Here are two possibilities:

```
'apple'   'abc'   False
'apple'   'pal'   True
```
```

Question 4. [8 MARKS]

Complete the following functions according to their docstring descriptions.

Part (a) [4 MARKS] **Do not use the `str` method `swapcase`.**

```
def our_swapcase(s):
    '''(str) -> str
    Return a copy of the string s with uppercase letters converted to
    lowercase, and lowercase letters converted to uppercase.'''

    s = ''

    for char in s:
        if char.islower():
            s += char.upper()
        elif char.isupper():
            s += char.lower()
        else:
            s += char
    return s
```

Part (b) [4 MARKS]

```
def length_dict(word_list):
    '''(list of strs) -> dict of {int : list of strs}
    Return a dictionary in which each key is a word length and each value is a
    list of the words from word_list with that length.  Only the lengths of the
    words that appear in word_list should be included as keys in the dictionary.'''

    word_dict = {}

    for word in word_list:
        if len(word) in word_dict:
            word_dict[len(word)].append(word)
        else:
            word_dict[len(word)] = [word]

    return word_dict
```

Question 5. [8 MARKS]

The `media` module has a function called `media.copy` that takes a `Picture` and returns a new `Picture` that is a copy of the original. Implement your own version of `media.copy` without using `media.copy`. You will be marked for efficiency, as well as for correctness.

```
def our_copy(pic):
    ```(Picture) -> Picture
    Return a new picture that is a copy of pic.'''

    pic_copy = media.create_picture(pic.get_width(), pic.get_height())

    for pixel in pic:
        x = media.get_x(pixel)
        y = media.get_y(pixel)
        color = media.get_color(pixel)
        sister_pic = media.get_pixel(pic_copy, x, y)
        media.set_color(sister_pic, color)

    return pic_copy
```
```

Question 6. [4 MARKS]

Recall the following function from Exercise 3:

```
def get_letter_counts(letter_dict):
    '''(dict of {str : list of strs}) -> dict of {str : int}
    In letter_dict, each key is a single lowercase letter and each value is a
    list of lowercase words that start with that letter. Based on letter_dict,
    return a new dictionary in which each key is a single lowercase letter and
    each value is the number of lowercase words that start with that letter.'''
```

Suppose that we want to test `get_letter_counts`. Describe four test cases that each test different "categories" of inputs. To describe each test case, give the `dict` that you would pass to `get_letter_counts`, the return value you expect, and the purpose of the test case. Do **not** write any code. We have given you one test case as an example; add four more.

Solution:

Value of `letter_dict`	Return value	Purpose of this test case
`{}`	`{}`	empty dictionary

```
Lots of options, including:
list of length 1
list longer than length 1
all elements equal -1
one element is -1
position of min at beginning, middle, end
```

Question 7. [8 MARKS]

A **"address file"** contains one street address per line, such as:

```
100 Main Street
23 Spring Park Road
2012 Sunny Lane
4 Martin Luther King Drive
```

An **"address list"** is a `list of lists of strs`, where each inner list is a 3-element list of street number, name, and type. For example, the "address list" for the file above is:

```
[['100', 'Main', 'Street'], ['23', 'Spring Park', 'Road'], \
['2012', 'Sunny', 'Lane'], ['4', 'Martin Luther King', 'Drive']]
```

In the file, a single space separates the number, name and type. The format of each address is:

- the street number is an integer
- the street name is one or more words (with words separated by a single space)
- the street type is one word

Complete the following funtion according to its docstring description.

```
def load_addresses(numbers_file):
    '''(file) -> list of lists of strs
    Return an "address list" based on the contents of the given "address file".'''

    addresses = []

    for line in numbers_file:
        address = []
        parts = line.split()
        address.append(parts[0])

        name = ''
        for i in range(1, len(parts) - 1):
            name += parts[i] + ' '
        address.append(name.strip())

        address.append(parts[-1])

        addresses.append(address)

    return addresses
```

Question 8. [12 MARKS]

In this question, you will implement part of a word search game. A word search board is a rectangular board of lowercase letters. For example:

r	b	t	x	m	k
e	a	t	x	d	s
x	b	a	t	o	p
q	y	v	x	d	l

A **"word search file"** contains a word search board. For example:

```
rbtxmk
eatxds
xbatop
qyvxdl
```

A **"word search list"** (a `list of strs`) represents a word search board. For example:

```
['rbtxmk', 'eatxds', 'xbatop', 'qyvxdl']
```

The object of this game is to identify words (sequences of characters) that appear in the board. A word can appear from top to bottom in a column, for example ***baby*** or ***spl***:

r	***b***	t	x	m	k
e	***a***	t	x	d	***s***
x	***b***	a	t	o	***p***
q	***y***	v	x	d	***l***

A word can also appear from left to right in a row, for example ***eat*** or ***vx***:

r	b	t	x	m	k
e	***a***	***t***	x	d	s
x	b	a	t	o	p
q	y	***v***	***x***	d	l

In the examples shown above, ***spl*** and ***vxd*** are considered words on the board, even though they are not actual English words. For the purpose of this program, a word in the word search board is any sequence of 1 or more character(s) that appears in a row (from left to right) or column (from top to bottom) of the word search board, regardless of whether it is an actual English word. We will not consider words that appear on a diagonal.

The next two pages contain starter code for a program that continually prompts the user to guess a word, until they guess a word that appears on the word search board.

The functions and main on the next two pages are in the same module. Complete the missing code according to the docstring descriptions and inline comments. Avoid duplicate code by calling functions from this module as helpers. You should do this even if the helpers are incomplete/incorrect. Each function will be marked as though any helper it relies on works correctly.

```
def load_board(ws_file):
    '''(file) -> list of strs
    ws_file is a "word search file".  Each line of the file contains
    lowercase letters and all lines have the same length.
    Return a "word search list" based on the contents of ws_file.'''

    # main passed a str not a file, so some may have included code to open the file:
    # open(word_search_file)

    board = []
    for line in file:
        board.append(line.strip())
    return board


def in_row(ws_board, word):
    '''(list of strs, str) -> bool
    Return True iff word appears in at least one row of "word search board" ws_board.'''

    return in_list(board, word)
```

```
def in_column(ws_board, word):
    '''(list of strs, str) -> bool
    Return True iff word appears in at least one column of "word search board" ws_board.'''

    columns = []

    for i in range(len(board[0])):
        columns.append('')

    for row in board:
        for i in range(len(row)):
            columns[i] += row[i]
    print columns

    return in_list(columns, word)


def is_found(ws_board, word):
    '''(list of strs, str) -> bool
    Return True iff word appears in at least one row or column of "word search board"
    ws_board.'''

    return in_row(board, word) or in_column(board, word)


if __name__ == '__main__':

    board = load_board('words.txt')  # should have been a file
    display_board(board)

    word = raw_input("Enter a word that appears in the board: ")

    while not is_found(board, word):
        word = raw_input("Enter a word that appears in the board: ")

    print "The word %s appears in the board" % (word)
```

Question 9. [12 MARKS]

This question involves working with dictionaries of the following types:

"book to people dictionary"

- **key:** a book (a `str`), **value:** a list of people who have read that book (a `list of strs`)
- **example:**
```
{'Treasure Island': ['Joanna', 'Jonathan', 'Hui', 'Ali'],\
 'Wuthering Heights': ['Hui', 'Monia', 'Ali'],\
 'Middlemarch': ['Vlad', 'Ali']}
```

"person to books dictionary"

- **key:** a person (a `str`), **value:** a list of books that person has read (a `list of strs`)
- **example:**
```
{'Vlad': ['Middlemarch'], 'Monia': ['Wuthering Heights'],\
 'Ali': ['Treasure Island', 'Wuthering Heights', 'Middlemarch'],\
 'Hui': ['Treasure Island', 'Wuthering Heights'],\
 'Joanna': ['Treasure Island'], 'Jonathan': ['Treasure Island']}
```

Complete the following functions according to their docstring descriptions.

Part (a) [5 MARKS]

```
def invert_book_to_people(book_to_people):
    '''(dict of {str: list of strs}) -> dict of {str: list of strs}
    book_to_people is a "book to people dictionary".
    Return a "person to books dictionary" based on book_to_people.'''

    person_to_books = {}

    for (book, people) in book_to_people.items():
        for person in people:
            if person in person_to_books:
                person_to_books[person].append(book)
            else:
                person_to_books[person] = [book]

    return person_to_books
```

Part (b) [7 MARKS]

This part of the question involves another type of dictionary that is used to make recommendations.

"recommendations dictionary" for book B

- **key:** a book (a `str`) that people who read book B have also read,
  value: the number of people (an `int`) who read both this book (the key) and book B
- **example:** for 'Wuthering Heights'

  ```
  {'Treasure Island': 2, 'Middlemarch': 1}
  ```

```
def make_recommendations(book_to_people, book):
    '''(dict of {str: list of strs}) -> dict of {str: int}
    book_to_people is a "book to people dictionary" and book is a book title
    that occurs as a key in book_to_people.
    Return a new "recommendations dictionary" for book.'''

    recommendations = {}
    people = book_to_people[book]
    person_to_books = invert_book_to_people(book_to_people)

    for person in people:
        books = person_to_books[person]
        for possible_book in books:
            if book != possible_book:
                if possible_book in recommendations:
                    recommendations[possible_book] += 1
                else:
                    recommendations[possible_book] = 1

    return recommendations
```

Question 10. [8 MARKS]

Throughout this question, assume that we are sorting lists into non-descending order. **Do not guess.** There is a one-mark deduction for incorrect answers.

Part (a) [2 MARKS]

We are partly through sorting a list, and have completed 3 passes through the data. The list currently contains `[4, 6, 7, 8, 5, 9, 7, 15]`. Which sorting technique are we definitely *not* using? Circle one.

bubble sort **insertion sort**

Part (b) [2 MARKS]

Suppose you have a list of 100 distinct numbers in order from largest to smallest, in other words, backwards to the order we want. Which sorting technique would require the most comparisons? Circle one.

insertion sort selection sort **they would require an equal number**

Part (c) [2 MARKS]

Suppose you have a list of 100 distinct numbers in order from largest to smallest, in other words, backwards to the order we want. Which sorting technique would require the most swaps? Circle one.

insertion sort selection sort they would require an equal number

Part (d) [2 MARKS]

Consider the following lists: `L1 = [9, 8, 7, 6, 5, 4, 3, 2, 1]` and `L2 = [8, 1, 3, 6, 9, 7, 4, 2, 5]` Which list would cause insertion sort to do more swaps? Circle one.

L1 `L2` they would require an equal number

Question 11. [4 MARKS]

Do not guess. There is a half-mark deduction for incorrect answers.

Part (a) [1 MARK]

```
for i in range(len(L)):
    L[i] = L[i] ** 2
```

Let `n` be the size of the list `L`. Which of the following most accurately describes how the runtime of this function grow as `n` grows? Circle one.

(a) It grows linearly, like n does. (b) It grows quadratically, like n^2 does.

(c) It grows less than linearly. (d) It grows more than quadratically.

Part (b) [1 MARK]

```
i = 0
while i != len(L):
    L[i] = L[i] ** 2
    i += 1
```

Let `n` be the size of the list `L`. Which of the following most accurately describes how the runtime of this function grow as `n` grows? Circle one.

(a) It grows linearly, like n does. (b) It grows quadratically, like n^2 does.

(c) It grows less than linearly. (d) It grows more than quadratically.

Part (c) [1 MARK]

```
for x in range(len(L)):
    for y in range(len(L[0])):
        if y == y1:
            L[x][y1] = L[x][y1] ** 2
```

Solution: The question did not specify how L[0] grows, so accepting two answers as correct (**linearly**: assuming L[0] is constant; **quadratically**: assuming L[0] grows proportionally to L).

(a) It grows linearly, like n does. **(b) It grows quadratically, like n^2 does.**

(c) It grows less than linearly. (d) It grows more than quadratically.

Part (d) [1 MARK]

```
for x in range(len(L)):
    L[x][y1] = L[x][y1] ** 2
```

Let `n` be the size of the list `L`. Which of the following most accurately describes how the runtime of this function grow as `n` grows? Circle one.

(a) It grows linearly, like n does. (b) It grows quadratically, like n^2 does.

(c) It grows less than linearly. (d) It grows more than quadratically.