DICTIONARY-B

EXERCISE.

A Tic-Tac-Toe Board

A tic-tac-toe board looks like a large hash symbol (#) with nine slots that can each contain an X, an O, or a blank.

To represent the board with a dictionary, you can assign each slot a string-value key, as shown in the Figure.

'top-L'	'top-M'	'top-R'
'mid-L'	'mid-M'	'mid-R'
'low-L'	'low-M'	'low-R'

You can use string values to represent what's in each slot on the board: 'X', 'O', or ' ' (a space character). Thus, you'll need to store nine strings. You can use a dictionary of values for this. The string value with the key 'top-R' can represent the top-right corner, the string value with the key 'low-L' can represent the bottom-left corner, the string value with the key 'mid-M' can represent the middle, and so on.

This dictionary is a data structure that represents a tic-tac-toe board. Store this board-as-a-dictionary in a variable named theBoard. Open a new file editor window, and enter the following source code, saving it as ticTacToe.py:

The data structure stored in the theBoard variable represents the tic-tac-toe board in Figure.

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

Since the value for every key in theBoard is a single-space string, this dictionary represents a completely clear board.

Figure: An empty tic-tac-toe board

If player X went first and chose the middle space, you could represent that board with this dictionary:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

The data structure in theBoard now represents the tic-tac-toe board in Figure.

	X	

A board where player O has won by placing Os across the top might look like this:

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O',
            'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}
```

The data structure in theBoard now represents the tic-tac-toe board in Figure. Player O wins

O	O	O
X	X	
		X

DICTIONARY-B

Let’s create a function to print the board dictionary onto the screen. Make the following addition to ticTacToe.py (new code is in bold):

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ','mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
'low-L': ' ', 'low-M': ' ', 'low-R': ' '}

def printBoard(board):
        print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
        print('-+-+-')

        print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
        print('-+-+-')

        print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])

        printBoard(theBoard)
```

When you run this program, printBoard() will print out a blank tic-tactoe board.

```
| |
-+-+-
| |
-+-+-
| |
```

The printBoard() function can handle any tic-tac-toe data structure you pass it. Try changing the code to the following:

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O', 'mid-L': 'X', 'mid-M':'X', 'mid-R': ' ',
'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}

def printBoard(board):
        print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
        print('-+-+-')
        print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
        print('-+-+-')
        print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
        printBoard(theBoard)
```

Now when you run this program, the new board will be printed to the screen.

```
O|O|O
-+-+-
```

DICTIONARY-B

```
X|X|
-+-+-
| |X
```

Because you created a data structure to represent a tic-tac-toe board and wrote code in printBoard() to interpret that data structure, you now have a program that "models" the tic-tac-toe board. You could have organized your data structure differently (for example, using keys like 'TOP-LEFT' instead of 'top-L'), but as long as the code works with your data structures, you will have a correctly working program. For example, the printBoard() function expects the tic-tac-toe data structure to be a dictionary with keys for all nine slots. *If the dictionary you passed was missing, say, the 'mid-L' key, your program would no longer work.*

```
O|O|O
-+-+-
Traceback (most recent call last):
File "ticTacToe.py", line 10, in <module>
printBoard(theBoard)
File "ticTacToe.py", line 6, in printBoard
print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
KeyError: 'mid-L'
```

DICTIONARY-B

Now let's add code that allows the players to enter their moves.

Modify the ticTacToe.py program to look like this:

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ', 'mid-L': ' ', 'mid-M': '', 'mid-R': ' ',
'low-L': ' ', 'low-M': ' ', 'low-R': ' '}

def printBoard(board):
        print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
        print('-+-+-')

        print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
        print('-+-+-')

        print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])

turn = 'X'
for i in range(9):
        printBoard(theBoard)
        print('Turn for ' + turn + '. Move on which space?')

        move = input()
        theBoard[move] = turn

        if turn == 'X':
                turn = 'O'
        else:
                turn = 'X'

printBoard(theBoard)
```

The new code prints out the board at the start of each new turn u, gets the active player's move v, updates the game board accordingly w, and then swaps the active player x before moving on to the next turn.

When you run this program, it will look something like this:

```
| |
-+-+-
| |
-+-+-
| |
Turn for X. Move on which space?
mid-M
| |
-+-+-
|X|
-+-+-
| |
Turn for O. Move on which space?
low-L
| |
-+-+-
|X|
-+-+-
O| |
------
------
------
------
O|O|X
-+-+-
X|X|O
-+-+-
O| |X
Turn for X. Move on which space?
low-M
O|O|X
-+-+-
X|X|O
-+-+-
O|X|X
```

DICTIONARY-B

Storing Names and Birthdays in a Dictionary

In this exercise we look at a program that keeps your friends' names and birthdays in a dictionary. Each entry in the dictionary uses a friend's name as the key and that friend's birthday as the value. You can use the program to look up your friends' birthdays by entering their names.

The program displays a menu that allows the user to make one of the following choices:

1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

The program initially starts with an empty dictionary, so you have to:

- choose item 2 from the menu to add a new entry.

Once you have added a few entries, you can:

- choose item 1 to look up a specific person's birthday,
- item 3 to change an existing birthday in the dictionary,
- item 4 to delete a birthday from the dictionary, or
- item 5 to quit the program.

The Program shows the program code.
The program is divided into six functions:

- main,
- get_menu_choice,
- look_up,
- add, change, and
- delete.

Rather than presenting the entire program at once, let's first examine the global constants and the main function:

DICTIONARY-B

Program (birthdays.py: main function)

```
1 # This program uses a dictionary to keep friends'
2 # names and birthdays.
3
4 # Global constants for menu choices
5 LOOK_UP = 1
6 ADD = 2
7 CHANGE = 3
8 DELETE = 4
9 QUIT = 5
10
11 # main function
12 def main():
13     # Create an empty dictionary.
14      birthdays = {}
15
16      # Initialize a variable for the user's choice.
17      choice = 0
18
19      while choice != QUIT:
20               # Get the user's menu choice.
21              choice = get_menu_choice()
22
23               # Process the choice.
24              if choice == LOOK_UP:
25                       look_up(birthdays)
26              elif choice == ADD:
27                        add(birthdays)
28               elif choice == CHANGE:
29                        change(birthdays)
30               elif choice == DELETE:
31                       delete(birthdays)
32
```

The global constants that are declared in lines 5 through 9 are used to test the user’s menu selection.

Inside the main function, line 14 creates an empty dictionary referenced by the birthdays variable.

Line 17 initializes the choice variable with the value 0. This variable holds the user’s menu selection.

The while loop that begins in line 19 repeats until the user chooses to quit the program. Inside the loop, line 21 calls the get_menu_choice function. The get_menu_choice function displays the menu and returns the user's selection. The value that is returned is assigned to the choice variable.

The if-elif statement in lines 24 through 31 processes the user's menu choice.

If the user selects item 1, line 25 calls the look_up function.
If the user selects item 2, line 27 calls the add function.
If the user selects item 3, line 29 calls the change function.
If the user selects item 4, line 31 calls the delete function.

The get_menu_choice function is next.
Program (birthdays.py: get_menu_choice function)

```
33 # The get_menu_choice function displays the menu
34 # and gets a validated choice from the user.
35 def get_menu_choice():
36     print()
37     print('Friends and Their Birthdays')
38     print('---------------------------')
39     print('1. Look up a birthday')
40     print('2. Add a new birthday')
41     print('3. Change a birthday')
42     print('4. Delete a birthday')
43     print('5. Quit the program')
44     print()
45
46     # Get the user's choice.
47     choice = int(input('Enter your choice: '))
48
49     # Validate the choice.
50     while choice < LOOK_UP or choice > QUIT:
51         choice = int(input('Enter a valid choice: '))
52
53     # return the user's choice.
54     return choice
55
```

The statements in lines 36 through 44 display the menu on the screen.

Line 47 prompts the user to enter his or her choice. The input is converted to an int and assigned to the choice variable. The while loop in lines 50 through 51 validates the user's input and, if necessary, prompts the user to reenter his or her choice.

Once a valid choice is entered, it is returned from the function in line 54.

The look_up function is next.
Program (birthdays.py: look_up function)

```
56 # The look_up function looks up a name in the
57 # birthdays dictionary.
58 def look_up(birthdays):
59     # Get a name to look up.
60     name = input('Enter a name: ')
61
62     # Look it up in the dictionary.
63     print(birthdays.get(name, 'Not found.'))
64
```

The purpose of the look_up function is to allow the user to look up a friend's birthday. It accepts the dictionary as an argument.

Line 60 prompts the user to enter a name, and line 63 passes that name as an argument to the dictionary's get function. If the name is found, its associated value (the friend's birthday) is returned and displayed. If the name is not found, the string 'Not found.' is displayed.

The add function is next.
Program (birthdays.py: add function)

```
65 # The add function adds a new entry into the
66 # birthdays dictionary.
67 def add(birthdays):
68     # Get a name and birthday.
69     name = input('Enter a name: ')
70     bday = input('Enter a birthday: ')
71
72     # If the name does not exist, add it.
73     if name not in birthdays:
74         birthdays[name] = bday
75     else:
76         print('That entry already exists.')
77
```

The purpose of the add function is to allow the user to add a new birthday to the dictionary. It accepts the dictionary as an argument. Lines 69 and 70 prompt the user to enter a name and a birthday.

The if statement in line 73 determines whether the name is not already in the dictionary. If not, line 74 adds the new name and birthday to the dictionary. Otherwise, a message indicating that the entry already exists is printed in line 76.

The change function is next.
Program (birthdays.py: change function)

```
78 # The change function changes an existing
79 # entry in the birthdays dictionary.
80 def change(birthdays):
81      # Get a name to look up.
82      name = input('Enter a name: ')
83
84      if name in birthdays:
85          # Get a new birthday.
86          bday = input('Enter the new birthday: ')
87
88          # Update the entry.
89          birthdays[name] = bday
90     else:
91         print('That name is not found.')
92
```

The purpose of the change function is to allow the user to change an existing birthday in the dictionary. It accepts the dictionary as an argument.

Line 82 gets a name from the user.

The if statement in line 84 determines whether the name is in the dictionary. If so, line 86 gets the new birthday, and line 89 stores that birthday in the dictionary. If the name is not in the dictionary, line 91 prints a message indicating so.

The delete function is next.
Program (birthdays.py: change function)

```
93 # The delete function deletes an entry from the
94 # birthdays dictionary.
95 def delete(birthdays):
96     # Get a name to look up.
97      name = input('Enter a name: ')
98
99     # If the name is found, delete the entry.
100     if name in birthdays:
```

```
101             del birthdays[name]
102       else:
103             print('That name is not found.')
104
105 # Call the main function.
106 main()
```

The purpose of the delete function is to allow the user to delete an existing birthday from the dictionary. It accepts the dictionary as an argument.

Line 97 gets a name from the user.

The if statement in line 100 determines whether the name is in the dictionary. If so, line 101 deletes it. If the name is not in the dictionary, line 103 prints a message indicating so.

Program Output (with input shown in bold)

```
Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 2        enter
Enter a name: Cameron         enter
Enter a birthday: 10/12/1990    enter

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 2        enter
Enter a name: Kathryn       enter
Enter a birthday: 5/7/1989    enter
```

DICTIONARY-B

```
Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 1 enter
Enter a name: Cameron enter
10/12/1990

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 1 enter
Enter a name: Kathryn enter
5/7/1989

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 3 enter
Enter a name: Kathryn enter
Enter the new birthday: 5/7/1988 enter

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program
```

DICTIONARY-B

```
Enter your choice: 1 [enter]
Enter a name: Kathryn [enter]
5/7/1988

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 4 [enter]
Enter a name: Cameron [enter]


Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 1 [enter]
Enter a name: Cameron [enter]
Not found.

Friends and Their Birthdays
---------------------------
1. Look up a birthday
2. Add a new birthday
3. Change a birthday
4. Delete a birthday
5. Quit the program

Enter your choice: 5 [enter]
```

DICTIONARY-B

ASSIGNMENT:

1. Scrabble Score: In the game of Scrabble™, each letter has points associated with it. The total score of a word is the sum of the scores of its letters. More common letters are worth fewer points while less common letters are worth more points. The points associated with each letter are shown below:

One point	A, E, I, L, N, O, R, S, T and U
Two points	D and G
Three points	B, C, M and P
Four points	F, H, V, W and Y
Five points	K
Eight points	J and X
Ten points	Q and Z

Write a program that computes and displays the Scrabble™ score for a word.
Create a dictionary that maps from letters to point values. Then use the dictionary to compute the score.

2. Text Messaging: On some basic cell phones, text messages can be sent using the numeric keypad. Because each key has multiple letters associated with it, multiple key presses are needed for most letters. Pressing the number once generates the first letter on the key. Pressing the number 2, 3, 4 or 5 times will generate the second, third, fourth or fifth character listed for that key.

Key	Symbols
1	. , ? ! :
2	A B C
3	D E F
4	G H I
5	J K L
6	MNO
7	P Q R S
8	T U V
9	W X Y Z
0	*space*

Write a program that displays the key presses that must be made to enter a text message read from the user. Construct a dictionary that maps from each letter or symbol to the key presses. Then use the dictionary to generate and display the presses for the user's message.
For example, if the user enters "Hello, World!" then your program should output:
4433555555666110966677755531111.