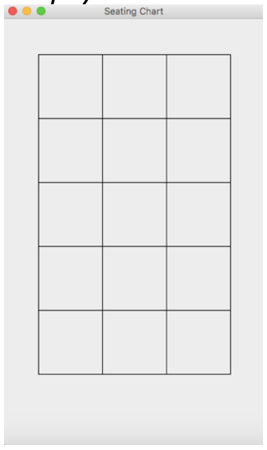
CSCI 141 Problem Set 5

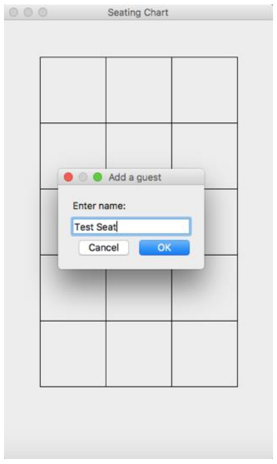
Your solutions to this problem set MUST be submitted online via the Blackboard link on the course page. E-mailed and handwritten solutions will not be graded. Only your last attempt will be graded.

Consider making a seating chart for a classroom or conference room. You have ROWS number of rows, and SEATS seats per row. You want to implement a graphical application which allows the user to click on a seat and then type the name of an individual who will be sitting there into a dialog box. The name then appears on the seating chart. Users are allowed to change their entries (i.e. update information on who is sitting where). If the user enters a name longer than 15 characters, they get an error message. Here is an example of a working implementation:

Empty chart when program starts (ROWS = 5, SEATS = 3):



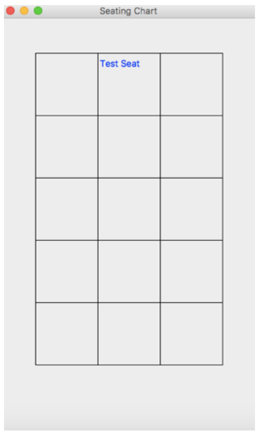
Click on a seat and enter a name:



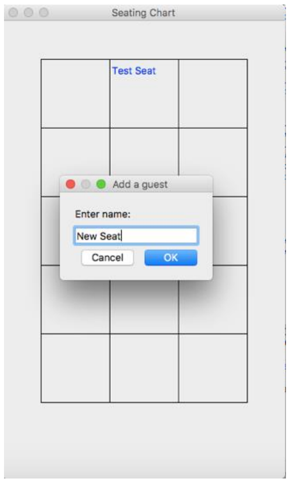
Enter name:

Test Seat

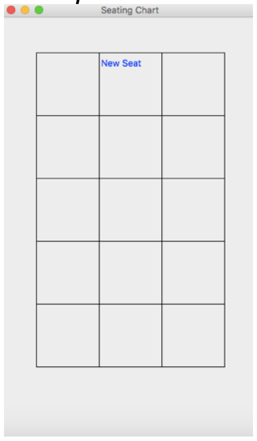
Seat is now filled:



Click on filled seat

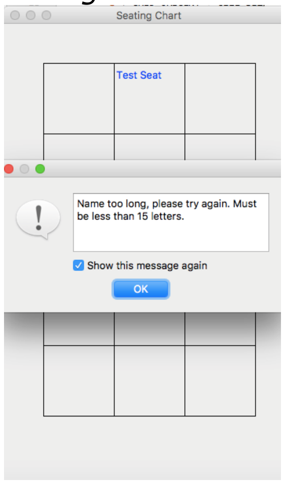


Occupant of seat now changed:



CSCI 141 Problem Set 5

User gets an error message for names that are too long:



Name too long, please try again. Must be less than 15 letters.

1) You are provided with one attempt to implement the chart shown above in the file Seating.py. It does not work correctly. You can fix the implementation by changing a single line of code. Select the line you wish to change and then provide your corrected line which allows the implementation to work properly.

NOTE: Your answers to questions 2 and 3 do NOT rely on your answering question 1 correctly. You can complete all 3 questions independently.

2) In the original implementation, the user is allowed to change the individual who is sitting in a seat at any time by clicking in a filled square. Suppose that we want to remove this functionality once the seating chart is full. That is, as long as there are empty spaces, names can be changed, but once every space is filled in, the user cannot make any more changes. In Seating.py, you have been provided with a blank method definition for a private method called __is_full. Complete the following tasks:

   a) Implement the __is_full method by selecting from the choices below. The method should return a Boolean and will utilize **5 lines of code**. You do not need to indicate indentation, but it may be important in your code running correctly.

```
if r != '':
for place in ROWS:
if c == spot:
for c in r:
return spot != ''
if c != r:
return place != ''
if c == '':
return c
return r == ''
if place == '':
return c != ''
for place in self.__desks:
if place != spot:
for r in range(len(self.__desks)):
if c == r:
```

```
for c in range(r):
for spot in place:
ROWS +=1
return r != ''
SEATS +=1
return place
for r in range(ROWS):
for c in range(self.__desks):
while(spot<SEATS):
for spot in SEATS:
while(r<range(len(self.__desks)):
return spot == ''
r += 1
else:
for c in self.__desks:
return r
if r != spot:
return c == ''
while(place<ROWS):
while(c<ROWS):
if c != '':
if r == spot:
return self.__desks[r][c] == ''
if place == spot:
return True
return False
if spot == '':
return place == ''
for spot in self.__desks:
for place in range(ROWS):
return spot
```

b) Indicate which method in the Seating class should call the __is_full method.

c) Add a single line of code to the method you indicated in order to implement the functionality described using the __is_full method (that is, users can't change names once the chart is full). HINT: the line of code you add may require some indentation changes in order for the method to run correctly – you do not need to indicate these in your answer. In the blanks provided, paste in your line of code, and the line that comes immediately after it in the Seating.py.

NOTE: Your answer to question 3 does NOT rely on your answering questions 1 and 2 correctly. You can complete all 3 questions independently.

3) Instead of only forbidding changes when the seating chart is full, you decide that users can NEVER change their entries. Once a seat has been assigned, the assignment is final. To accomplish this, you need to add a single line of code to the original implementation. Complete the following tasks:

a) Indicate which method you wish to add the line of code to.

b) Paste in your line of code and the line that comes immediately after it in Seating.py into the blanks in Blackboard. You do not need to worry about indentation when submitting your answer.

Many word puzzles can be solved by iterating through a list or file of words while checking for characteristics specified by the puzzle. “dictionary.txt” is a file of words, one per line, to use for the question 4 and 5.

4) The program below aims to find all the words in the dictionary.txt of a particular length containing just a single vowel (defined as a, e, i, o and u) that does not have a particular letter in it. However, it does not work correctly. For example, if I am looking for all the words of length 9 containing just a single vowel that does not have letter ‘t’ in it, the program below tells me “There are no words that fit this criteria”. But there should be two words in the file satisfying the above criteria: “lynchings”, and “lynchpins”.

   a) You can fix the implementation by changing a single line of code. Indicate the line you wish to change and then provide your corrected line which allows the implementation to work properly.

```
1   def onevowel(file):
2       length = int(input("Please enter the word length you are looking for: "))
3       letter = input("Please enter the letter you'd like to exclude: ")
4       wordnum = 0
5       for word in file:
6           word = word.strip()
7           if len(word) == length:
8               count = 0
9               for char in word:
10                  if (char=='a' and char=='e' and char=='i' and char=='o' and char=='u'):
11                      count += 1
12              if count == 1:
13                  flag = 1
14                  word_str = ""
15                  for char in word:
16                      if char == letter:
17                          flag = 0
18                      else:
19                          word_str += char
20                  if flag == 1:
21                      print (word_str)
22                      wordnum += 1
23      if wordnum == 0:
24          print ("There are no words that fit this criteria.")
25
26  if __name__ == "__main__":
27      my_file = open("dictionary.txt","r")
28      onevowel(my_file)
29      my_file.close()
```

5) The program below aims to find all the palindromes of a particular length. A palindrome is a string of characters that is the same forwards and backwards. For example, the words “civic”, “noon”, and “a” are all palindromes.

   a) The function palindrome() below is not complete. Complete the implementation in the provided file “Q13.py” that prints all the palindromes of a particular length. If no words are found that fit the criteria, the message “There are no words that fit this criteria.” should be printed. Don’t modify the existing lines of code in the provided file.

      For example, if you run the file with the complete code from the command line, the output should be as follows:

```
>>> python Q13.py
Enter the length of the palindromes you desire: 5
civic
kayak
level
madam
minim
radar
refer
rotor
sagas
sexes
solos
stats
tenet
There are 13 words that fit this criteria.

>>> python Q13.py
Enter the length of the palindromes you desire: 10
There are no words that fit this criteria.
```

   b) Run your complete code to answer the following questions:
      How many palindromes of length 3 in the dictionary.txt?
      How many palindromes of length 4 in the dictionary.txt?
      How many palindromes of length 6 in the dictionary.txt?

```
def palindrome(file):
    length = int(input("Enter the length of the palindromes you desire: "))
    # Complete the implementation of the function


if __name__ == "__main__":
    my_file = open("dictionary.txt","r")
    palindrome(my_file)
    my_file.close()
```