Name: _____ Net ID: _____

Question 1 (20 points)

For each of the following, show the final value of `my_var` at the end of executing the expressions, or write ERROR if there is an error.

	code fragment	value of my_var at end (or ERROR)
a.	`my_var = [ 1, 2, 3]` `my_var[1] = 5`	

b.	`list1 = [1, 2, 3]`
	`list2 = [4, 5]`
	`my_var = list1 + list2`	

c.	`my_var  = [1, 2, 3]`
	`arg = 4`
	`my_var.append( arg )`	

c.	`my_var = 'abc'` `my_var[1] = 'd'`	

e.	`str1 = 'abc'`
	`str2 = 'de'`
	`my_var = str1 + str2`	

c.	`my_var  = 'abc'`
	`arg = 'd'`
	`my_var.append( arg )`	

g.	`str1 = 'abc'`
	`my_var = [4, 5, 6]`
	`my_var.insert(2, str1)`	

h.	`str1 = 'abc'` `my_var = str1 * 3`	
i.	`my_var = [1, 2, 3]` `my_var.append([4, 5])`	
j.	`my_var = 'abc'[1]`	

Question 2 (12 points)
Given the following definition:

```
def funA( x ):
     print ("FunA 1 ", x )
     x = x + 1
     print ("FunA 2 ", x )
     return x


def funB ( y ):
     print ("FunB 1 ", y )
     z = funA( y )
     print ("FunB 2 ", y, z )


def main():
     print ("main 1 ")
     w = 5
     funB( w )
     print ("main 2 ", w )
     v = 10
     z = funA( v )
     print ("main 3 ", v)
     print ("main 4 ", z)
```

What would be printed when calling `main()`?

Output:

Question 3 (11 points)
Given the following definition:

```
def main():
    string = "abcd"
    s = ''
    lst = [1, 3, 0, 2]
    for elem in lst:
        curr_val = string[elem] * lst[elem]
        print(curr_val)
        s = s + curr_val
    print(s)
```

What would be printed when calling `main()`?

Output:

Question 4 (12 points)
Given the following definition:

```
def main():
    orig_lst = [1, 2, 3, 4, 5]
    res_lst = []
    for i in range(len(orig_lst)//2):
        curr_item = orig_lst[i : (len(orig_lst) - i)]
        print(curr_item)
        res_lst.append(curr_item)
    print(res_lst)
```

What would be printed when calling `main()`?

Output:

Question 5 (15 points)

Fill in the blanks to complete the definition of the function `alphabetical_less_than(word1, word2)`, that checks whether `word1` is **alphabetically** less than `word2` in a **case-insensitive** way:

Examples:

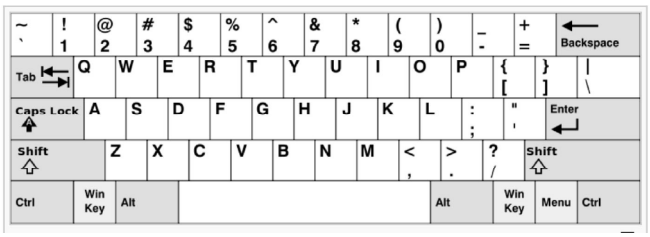
- Calling `alphabetical_less_than("Abc", "abDEf")` should return **`True`**
- Calling `alphabetical_less_than("Abc", "abcd")` should return **`True`**
- Calling `alphabetical_less_than("ABC", "abc")` should return **`False`**

```
def alphabetical_less_than(word1, word2):
    i = 0

    # setting shorter to be the length of the shorter word
    shorter = len(word1)
    if (len(word2) < len(word1)):
        shorter = len(word2)

    same_so_far = True

    # compare letters until a mismatch is found
    # or the end of one of the words is reached
    while (_____________________________________):
        #compare letters in position i
        if (word1[i].lower() < word2[i].lower()):
            same_so_far = ___________
            result = True
        elif (word1[i].lower() > word2[i].lower()):
            same_so_far = ___________
            result = ___________
        else:
            # all the letters match so far
            _____________________________


    if same_so_far:
        # words match, except that one of them is longer
        if (len(word1) < len(word2)):
            result = True
        else:
            result = False
    return result
```

Question 6 (30 points)

Imagine you are part of a team that's developing a system to correct mis-typed words. **Someone else** on your team has implemented a function: `qwerty_letter_distance(char1, char2)` that takes two lower-case letters and returns the qwerty-letter-distance between them. Here, the ***qwerty-letter-distance*** between two characters is a measure of how far apart they are on a standard US keyboard.

For example,
`qwerty_letter_distance('e', 'b') > qwerty_letter_distance('e', 'r')` ,
*in other words the qwerty-letter-distance between 'e' and 'b' is greater than the qwerty-letter-distance between 'e' and 'r' (*because 'b' is further from 'e' on the keyboard than 'r' is).



In this problem you will write and use a function to compute the ***qwerty-word-distance*** between two words of the same length, *word1* and *word2*, as the sum of the qwerty-letter-distances between the corresponding letters in *word1* and *word2* (letters in the same position).
For example, if the *qwerty-letter-distance* between b' and 'c' is 2, between 'a' and 'a' is 0, and between 'y' and 't' is 1, then the *qwerty-word-distance* between "bay" and "cat" is 3, since 2+0+1== 3.

Your task is to implement the following 2 functions:

- `qwerty_word_distance(word1, word2)` that gets two strings of the same length, `word1` and `word2`. Each word contains only lowercase letters.
  The function computes and returns the qwerty-word-distance between `word1` and `word2`.
  For example, under the assumptions above, `qwerty_word_distance("bay", "cat")` should return 3.

  Note: *You do NOT need to know any of the details of how* `qwerty-letter-distance` *is defined or computed -- someone else has implemented that function and you can call it.*

- `find_close_words(word_list, test_word, num)` that gets a list of words, `word_list`, an additional word, `test_word`, and a positive integer `num`.
  The function creates and returns a sub-list of the words from `word_list` such that

their qwerty-word-distance from `test_word` is at most `num`.
Implementation **requirement**: Your function should call `qwerty_word_distance` (It should not directly do the computation that `qwerty_word_distance` does).

Notes:

1. Assume that all words in `word_list` contain only lowercase letters, and so does `test_word`.
2. Assume that each word in `word_list` as well as `test_word` are all of the same length.

For example, given the following `main()` function:

```
def main():
    dist = qwerty_word_distance("bay", "cat")
    print("dist =", dist)

    word_lst = ["bay", "are", "bat"]
    close_lst = find_close_words(word_list, "cat", 4)
    print("close_lst =", close_lst)
```

When calling `main()`, under the assumptions above, the expected output is:

```
dist = 3
close_lst = ["bay", "bat"]
```

Write your functions on the next page. If either function is not around 5 to 10 lines long, think again.

Your functions:

```
def qwerty_word_distance(word1,word2):
```

```
def find_close_words(word_list, test_word, num):
```

EXTRA PAGE IF NEEDED

(Note question numbers of any questions or part of questions that you are answering here. Also write "ANSWER IS ON LAST PAGE" near the space provided for the answer.