**Week 7 In-Class Exercises (Review)**

**Q1: Code Tracing [ \*\* ]**

Given the following pieces of code, what are the outputs? Use memory state diagram to help you.

**(a)**

def do\_trick(a\_list):

a\_list = a\_list + a\_list[1:3] + [a\_list[3]]  
 print(a\_list)

my\_list = ['a', 'b', 'c', 'd']  
do\_trick(my\_list)  
print(my\_list)

**(b)**

def do\_trick(a\_list):

a\_list.append(a\_list[1:3] + [a\_list[3]])

print(a\_list)

my\_list = ['a', 'b', 'c', 'd']

do\_trick(my\_list)

print(my\_list)

**(c)**

my\_str = '123'

for index in range(0, len(my\_str)–1):  
 n = int(my\_str[index] + my\_str[index+1])  
 print(n)  
 m = int(my\_str[index:index+2])  
 print(m)

**Q2: Print Shapes**

1. [ \*\* ] Inside a file called **q2.py**, write a function called print\_triangle(ch, num\_rows) that takes in a character and a number as its parameters. The number indicates the number of rows. The function ***prints out*** a triangle using the character.

For example, if ch is '\*', num\_rows is 3, the following output is to be generated:

\*  
 \*\*\*  
\*\*\*\*\*

If ch is '#', num\_rows is 5, the following output is to be generated:

#  
 ###  
 #####  
 #######  
#########

You can assume that ch is always a string with a single character, and num\_rows is an integer that is at least 1.

Hint:

* For each row of the output, you need to figure out how many spaces to print in front of the characters. Note that this number is dependent on both the row number (i.e., 1st row, 2nd row, etc.) and the total number of rows (i.e., the second parameter of the function).
* For each row of the output, you also need to figure out how many times the character needs to be printed. Note that this number is dependent on the row number.

1. [ \*\* ] Inside **q2.py**, write another function called print\_frame(ch, num\_rows, num\_cols). The function ***prints out*** a frame as shown below. For example, if ch is '\*', num\_rows is 3 and num\_cols is 4, the following output is to be generated:

\*\*\*\*  
\* \*  
\*\*\*\*

If ch is '#', num\_rows is 5 and num\_cols is 6, the following output is to be generated:

######  
# #  
# #  
# #  
######

You can assume that ch is a string with a single character, and both num\_rows and num\_cols are integers that are at least 2.

1. [ \*\*\* ] Inside **q2.py**, write another function called print\_diamond(ch, n). The function takes in a character ch and an integer n as its parameters. You can assume that ch is a string with a single character and that n is an integer that is at least 2. The function prints a diamond shape where each side of the diamond has exactly n of ch.

For example,

* print\_diamond('\*', 4) prints

\*  
 \* \*  
 \* \*  
\* \*  
 \* \*  
 \* \*  
 \*

* print\_diamond('&', 2) prints

&  
& &  
 &

You can use **q2\_test.py** to test your code.

**Q3: Pangram [ \*\*\* ]**

Inside a file **q3.py**, write a function called is\_pangram to determine if a sentence is a pangram. A pangram is a sentence using every letter of the English alphabet at least once (either in uppercase or in lowercase). The best-known pangram is:

"The quick brown fox jumps over the lazy dog."

The function ***returns*** True if the given sentence is a pangram or False if it is not.

You can use **q3\_test.py** to test your code.

**Q4: Sort Strings [ \*\*\* ]**

In **q4.py**, write a function call sort\_strings().This function takes in a list of strings as its parameter. The function returns a new list of strings that contains all the strings in the original list but in **increasing order of their lengths**. The original list should remain unchanged. For strings of equal length, they should appear in the same order as they are in the original list.

For example, sort\_strings(['abc', 'a', 'xy', '12', 'x']) should return the list ['a', 'x', 'xy', '12', 'abc'].

**Hint:**

* You can gradually add strings in the original list to the new list, but you need to be careful about where to add each string to the new list.
* Instead of always appending a string to the end of the new list, you will need to insert a string into the new list. Suppose you want to insert string my\_str into the 4th position of my\_list, you can do the following:

my\_list = my\_list[0:4] + [my\_str] + my\_list[4:]

You can use **q4\_test.py** to test your code.