Topic: Review & Practice Problems

Goals: By the end of this topic, we will…

- *review for the test*

Acknowledgements: These class notes build on the content of my previous courses as well as the work of R. Jordan Crouser, and Jeffrey S. Castrucci.

Practice Problems

Question 1. Compare and contrast the sorting algorithms we learned this term.

Question 2. What will this code print?

a)

```
def func1(mult=10, reps=2):
    result_list=[]
    for i in range(1, reps+1):
        result_list.append(i*mult)
    return mult, result_list

print(func1(5, 3))
print(func1(reps=1))
```

b)

```
phonebook = {'Sam':'416-1212',
             'Katie':'647-2122',
             'Joan':'416-9923'}
phonebook['Mike'] = '416-1000'

print('416-1000' in phonebook)
print(phonebook['Sam'])
```

c)

```
number = 10
divisor = 1
list_divisors = []
while divisor < number:
    if number % divisor == 0:
        list_divisors += [divisor]
    divisor += 1

print(list_divisors)
```

d)

```
def fun1(n):        Skip has recursion...
    if n < 10:
        print(n)
    else:
        print(n)
        fun1(n//10)
        print(n)

fun1(1000)
```

Question 3. The Euclidean distance is a measure of the distance between two points in n-dimensional space. For two points: and the Euclidean distance, d, can obtained by applying the following formula:

$$d = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

Write a function which takes lists p = [p1,p2,...,pn] and q=[q1,q2,...,qn], representing two points in n-dimensional space, and returns the Euclidean distance between them.

For example:

```
>>> print(euclidean_distance([1, 1], [4, 5]))
5.0
Assume Math square root function is sqrt(n).
Assume Math to the power j**2
>>> print(euclidean_distance([1.5, -1.1, 4], [4.2, 1, -2.3]))
7.168681887209113

def euclidean_distance(p, q):
    '''(list, list) -> float
    Input: two lists, p and q, of floating point numbers

    Output: the Euclidean distance between p and q as a floating
    point number

    Pre-conditions: p and q have the same size and contain only
    floating point numbers.
    '''
```

Question 4. [4 marks] For an inventory application you must determine if there is enough inventory in the warehouses across the country to fill an order. The main data structure you will use is a dictionary representing the inventory database, with the following format:

```
{<product_name>: [[<amount>, <location>], [<amount>, <location>]]}
```

For example, the following dictionary says there are 120 gloves in New York, 240 gloves in Boston, and 200 buckets in Boston:

```
     {'gloves': [[120, 'New York'], [240, 'Boston']], 'bucket':
[[200, 'Boston']]}
```

On this page and the next page, write the function check_inventory which takes in the inventory database (in the form described above), a product name, and an amount and returns True if the total amount of inventory across all locations is greater than or equal to the amount. Otherwise the function returns False.

If the product does not appear in the database at all, you should return False.

```
>>> print(check_inventory(inventory_dict, 'gloves', 300))
True
```

```
def check_inventory(inventory, product, amount):
    '''(dictionary, str, int) -> bool
    Return True if the total amount of product in inventory
    is greater than or equal to amount. Otherwise return False

    Preconditions: the input arguments are all in the correct format
    '''
```

Question 5. Consider the following problem: `A` is a square of size `n` represented as a list of lists (assume [y][x] order). We want to clear the diagonal path within the square by replacing characters with '0'.

Write a function that corresponds to the following prototype:
def clearDiagonal(A, n):

For example, if A[11][11] looks like this (where the numbers indicate row and column numbers):

```
 0   a b c d e f g h i j k
 1   l m n o p q r s t u v
 2   w x y z a b c d e f g
 3   h i j k l m n o p q r
 4   s t u v w x y z a b c
 5   d e f g h i j k l m n
 6   o p q r s t u v w x y
 7   z a b c d e f g h i j
 8   k l m n o p q r s t u
 9   v w x y z a b c d e f
10   g h i j k l m n o p q

     0 1 2 3 4 5 6 7 8 9 10
```

After `def clearDiagonal(A, n)` is run, it will look like this.

```
10   0 b c d e f g h i j k
 9   l 0 n o p q r s t u v
 8   w x 0 z a b c d e f g
 7   h i j 0 l m n o p q r
 6   s t u v 0 x y z a b c
 5   d e f g h 0 j k l m n
 4   o p q r s t 0 v w x y
 3   z a b c d e f 0 h i j
 2   k l m n o p q r 0 t u
 1   v w x y z a b c d 0 f
 0   g h i j k l m n o p 0

     0 1 2 3 4 5 6 7 8 9 10
```

Self-study: try to clear the reverse diagonal!