1. **Using data structures**

   (a) An $m \times n$ matrix A is a mathematical structure with $m * n$ entries of data. Typically, A is partitioned into m rows and n columns, so that A_{ij} is the entry in the jth column of the ith row. For example, if $m = 3$, $n = 2$, and $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ -7 & 4 \end{pmatrix}$, $A_{21} = 3$.

   Using the data structures from class, describe how you could construct an $m \times n$ matrix. How can you access element A_{ij}? Can you change the element A_{ij}?

   (b) A directed graph is a set of vertices V and a set of edges $E \subset V \times V$. Suppose we have a graph with vertices $V = \{A, B, C, D, E, F\}$ and edges $E = \{(A, B), (B, C), (A, F), (E, D), (C, B)\}$. Using the data structures from class, describe a way to represent this graph.

   How would you add the edge (F, B) to your structure?

   (c) Sometimes, the edges in a graph can have weights. For example, if the set of vertices represent cities and the set of edges represent roads between those cities, then the weights could be the distance of the roads between the edges. Update your data structure from part (b) to have information about edge weights.

2. **List comprehensions**
   The following pieces of code are kludgy. Rewrite each of them using list comprehensions. *Hint: for parts (a) and (b), the range() function may be useful.*

   (a)
```
# compute first 20 powers of 2
i = 0
powers = []
while i < 20:
  p = 2 ** i
  powers.append(p)
  i = i + 1
```

   (b)
```
import math
# first 14 k-digit approximations of pi
approximations = []
i = 1
while i <= 14:
  approx = round(math.pi, i)
  approximations.append(approx)
  i += 1
```

   (c)
```
# Generate all (x, y, z) coordinates from three lists
xpoints = [1, 2, -1]
ypoints = [8, 4, 3, 0]
zpoints = [0, -1]
points = []
for x in xpoints:
  for y in ypoints:
    for z in zpoints:
      points.append((x, y, z))
```

3. **Syntax and indentation errors**
   Identify any syntax or indentation errors in the following Python scripts.

(a)

```
1  x = [1, 2, 3]
2
3  def func1:
4      if len(x) > 2:
5          x.append(4)
6    else:
7      x.pop()
```

(b)

```
1  def func(i, x):
2      while i < 10
3        print x * i
4        print i
5          i += 1
```

(c)

```
1  numbers = [1, 2] * 4
2  for i, j in enumerate(numbers): print i, j
3  for i, j in enumerate(numbers):
4      print i, j
5  for i, j in enumerate(numbers):
6    print i, j
```

4. **Copying**
Python does not use copy on assignment for objects. We will explore this in the following examples. What gets printed in the following Python scripts? Explain.

(a)

```
arr1 = [2, 3, 5, 7]
arr2 = arr1
arr2[2] = 13
print arr1[2]
arr3 = arr1[0:3]
arr3[0] = 17
print arr1[0]
```

(b)

```
def func(arr):
  arr.append(2)
my_arr = [1, 2, 3]
print len(my_arr)
func(my_arr)
print len(my_arr)
```

(c)

```
dict1 = {'apples': 3, 'oranges': 2}
dict2 = dict1
dict2['bananas'] = 5
if 'bananas' in dict1:
    print 'bananas is there!'
else:
    print 'no bananas'
```

(d)

```
x = 'hello'
y = x
y = 'hi'
print x
```