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}?

   We could use a nested list structure:

```
A = [[1, 2], [3, 4], [-7, 4]]
i = 2
j = 1
# access Aij
Aij = A[i][j]
# assign some value to Aij
A[i][j] = 4
```

   (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?

   We can create an adjacency list using a dictionary and lists:

```
graph = {'A': ['B', 'F'],
         'B': ['C'],
         'C': ['B'],
         'D': [],
         'E': ['D'],
         'F': []}

def add_edge(i, j):
    graph[i] += [j]

add_edge('F', 'B')
```

   (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.

   Instead of using a list of node keys for the adjacency list, we can use a list of (node key, weight) tuples:

```
# Assume the following edge weights:
# (A, B): 3; (A, F): 10
# (B, C): -4
# (C, B): 7
# (E, D): 8

graph = {'A': [('B', 3), ('F', 10)],
         'B': [('C', -4)],
         'C': [('B', 7)],
         'D': [],
         'E': [('D', 8)],
         'F': []}

def add_edge(i, j, weight):
    graph[i] += [(j, weight)]

# Assume (F, B) has weight -1
add_edge('F', 'B', -1)
```

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

   powers = [2 ** i for i in range(20)]
   ```

   (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

   approximations = [round(math.pi, i) for i in range(1, 15)].
   ```

   (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))

   points = [(x, y, z) for x in xpoints for y in ypoints for z in zpoints]
   ```

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()
```

Line 3 (syntax error): no parentheses; need `def func1():`
Line 6 (indentation error): else does not align with if

(b)

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

Line 2 (syntax error): should be `while i < 10:`
Line 5 (indentation error): `i += 1` does not align with the rest of the code block

(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
```

No errors. However, mixing all of these coding styles is frowned upon.

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]
```

On the second line, `arr2` points to the same object as `arr1`. Thus, in line 3, the assignment affects both `arr1` and `arr2` (they point to the same object). The first print statement prints 13.

The slicing operator creates a copy, so `arr3` points to a newly created object. Therefore, the second print statement prints 2.

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

The first print statement just prints 3, the length of `my_arr`. When the function is called, the reference is passed, so 2 gets appended to `my_arr`. Thus, the second print statement prints 4.

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

`dict1` and `dict2` are references to the same dictionary. Adding the bananas key to `dict2` also adds the key to `dict1`. Thus, "bananas is there!" gets printed.

(d)
```
x = 'hello'
y = x
y = 'hi'
print x
```

Strings are immutable, so we can't change them. In line 2, we just make a copy of "hello" that gets stored in the variable y. In line 3, we store "hi" in the variable y. Thus, the print statement prints "hi". The same holds for ints and tuples, which are also immutable.