Your Name:

1. Write a Python method that takes an integer \( n \geq 0 \) as a parameter and returns a Python list of length \( n \), all of whose elements are zeros. If \( n = 0 \), it should return a list of length 0.

   *Here are two possibilities:*

   ```python
def createList(n):
    return [0] * n

def createList(n):
    return [0 for i in range(n)]
```

2. Here is the start of the definition of a method that someone has added to the `Graph` class you wrote for Homework 3:

   ```python
def IndependentSet(self, i, j, k):
```

   The parameters \( i, j, k \) are vertices. It returns `True` if \( i, j, \) and \( k \) are an independent set. Suppose you have written a program that has a variable \( G \) that references an instance of `Graph` that has at least five vertices. Write a call to the method on \( G \) for vertices 1,3,4:

   `Solution: G.IndependentSet(1,3,4)`

3. Disprove this by counterexample: When BFS is called on the source of a DAG that has only one source, the order in which vertices are extracted from the queue is always a topological sort.

   ![Graph Diagram](image)

   *The order in which the vertices come off the stack could be (a,b,c).*

4. For each of the following pairs, tell whether \( f(n) = o(g(n)) \), \( f(n) = \omega(g(n)) \), or \( f(n) = \Theta(g(n)) \). (In other words, tell whether \( f(n) \) is slower growing than \( g(n) \), faster growing than \( g(n) \), or tied with \( g(n) \) in the big-O sense.) Show just enough work that gives evidence that you didn’t just guess, but err on the side of saving time. For example, set up the appropriate ratio and simplify, or give big-\( \Theta \) bounds for \( f(n) \) and \( g(n) \).

   (a) \( f(n) = 2^n; g(n) = 2^{2n} \)

   \[
   f(n) = o(g(n)): \quad g(n)/f(n) = 2^n, \text{ which goes to infinity as } n \text{ does.}
   \]
(b) \( f(n) = \log_2 2^n; \ g(n) = \log_2 2^{2n} \)
\[ f(n) = \Theta(g(n)); \ f(n) = n, \ g(n) = 2n \]
(c) \( f(n) = 9^n; \ g(n) = 10^n. \)
\[ g(n)/f(n) = (10/9)^n, \text{ which goes to infinity as } n \text{ does} \]
(d) \( f(n) = 20n^2 + 100n + 400; \ g(n) = n^{2.1}/\log_2 n. \)
\[ f(n) = o(g(n)); \ g(n) = \Omega(n^{2.05}), \text{ since a } logn \text{ is smaller in the big-O sense than } n^\epsilon \text{ for any } \epsilon > 0, \text{ such as 0.05.} \]
(e) \( f(n) = n!; \ g(n) = (2n)! \)
\[ f(n) = o(g(n)); \ g(n)/f(n) = (n + 1)(n + 2)...2n, \text{ which grows to infinity as } n \text{ does.} \]
(f) \[ f(n) = \sum_{i=1}^{n} i; \ g(n) = \sum_{i=1}^{2n} i. \]
\[ f(n) = \Theta(g(n)); \ f(n) = n(n + 1)/2 = \Theta(n^2), \ g(n) = 2n(2n + 1)/2 = \Theta(n^2). \]

5. Show by induction on \( n \) that \( \sum_{i=1}^{n} (2i - 1) = n^2 \) for \( n \geq 1. \)

(a) Base case:
\[ \sum_{i=1}^{1} (2i - 1) = 1 = 1^2. \]
(b) Suppose \( n \geq 2 \) and the claim is true for \( n - 1. \) That is:
\[ \text{(write an equality involving a summation that expresses this assumption)} \]
\[ \sum_{i=1}^{n-1} (2i - 1) = (n - 1)^2. \]
(c) Using algebra, complete the proof:
\[ \sum_{i=1}^{n} (2i - 1) = [\sum_{i=1}^{n-1} (2i - 1)] + 2n - 1 = (n - 1)^2 + 2n - 1 = n^2 - 2n + 1 + 2n - 1 = n^2. \]

6. Write a method that tells whether an undirected graph is connected. Assume you are adding it to the `Graph` class you built as part of Homework 3. Therefore, you can call any of the methods in the solution to that homework. The precondition is that the graph \( G, \) the parameter, is undirected. The postcondition is that it returns `False` if the graph is not connected and `True` if it is connected.

Recall that you have the following method available to call:
def dfs(self, i, colored):
    colored[i] = True
    for j in self.neighbors(i):
        if not colored[j]:
            self.dfs(j, colored)

One solution: By the White Path Theorem, it is connected if all vertices get colored by a call to DFS on a vertex, such as vertex 0:

def connected(self):
    colored = [False] * self.getn()
    self.dfs(0, colored)
    return not False in colored

(False in colored) is True if False occurs somewhere in Colored. The negation of this gives the answer.

After class, someone pointed out that the strongly-connected components of an undirected graph are just its connected components, so it suffices to just call scc. That is also a correct solution.

7. Suppose we have already shown that Gale-Shapley always produces a stable matching and that it always assigns every man his highest attainable partner. Let us now use these facts to show that it always assigns every woman her lowest attainable partner.

We can show this by contradiction. Let us suppose the negation:

Suppose that in some run of the algorithm on some instance of stable matching, there is a woman $w$ who gets a man $m'$ who she likes more than her lowest attainable man $m$.

(a) By the definition of attainable, there exists a stable matching $S'$ such that (fill out what logically goes in this part of the proof): $w$ is married to $m$.

(b) Let $w'$ be the spouse of $m'$ in $S'$. We have said that $w$ likes $m'$ more than $m$. Also, $m'$ likes $w$ more than $w'$ because: Gale-Shapley gives every man his highest attainable, so $w$ is the highest attainable for $m'$. Since $w'$ is also attainable for $m'$ (they are married in $S'$), she must be lower on his list.

(c) This gives a contradiction because: $S'$ can't be both stable and unstable.

(d) We can conclude that every woman gets her lowest attainable man because: The assumption that this was false led to a contradiction.

8. Nobody knows a polynomial-time algorithm for finding a longest path in a graph. (If you can find one, you will go down in history as one of the greatest computer scientists ever.)

If the graph is a DAG, however, we can get a good time bound by modifying DFS and calling blacken with the modified version of DFS. The trick is to label each vertex $v$
with the a **length** label that tells length of the longest path that begins at \( v \) when it is time to color \( v \) black.

(a) If \( v \) is a vertex in a DAG, then what colors can the neighbors of \( v \) be at the moment when \( v \) is blackened?

*T hey must all be black. None can be white by the White Path Theorem, and none can be gray, since that would imply a directed cycle.*

(b) How can you label \( v \) with its correct **length** label at this time? By induction, we can assume that all other vertices that are black at this time have been correctly labeled. You should design this step so that it adds \( O(n+m) \) to the running time of **blacken**, so that your total running time is \( O(n+m) \) to assign the labeling of

> Since they are black, the neighbors are all correctly labeled. Let \( w \) be a neighbor with a largest label, \( k \). Assign a label of \( k+1 \) to \( v \).

*(The edge \((v,w)\) followed by a path of length \( k \) from \( w \) is a path of length \( k+1 \) from \( v \).)*

(c) Tell how you can now construct a longest path in \( O(n+m) \) time.

*Find a vertex with largest label in \( O(n) \) time. Iteratively hop from vertex to vertex, each time to a vertex whose label is one smaller, until you reach a vertex whose label is 0.*

*An alternative is to install a parent pointer from \( x \) to a neighbor \( y \) that has a largest label at the time \( x \) is labeled. To construct a longest path from any vertex, follow the parent pointers, just as you did for shortest paths.*