## 11. What is the value of x?

\*\*Given these variable definitions:\*\*  
**```python  
lst1 = ['x', 'y', 'z']  
lst2 = ['<', '>']  
x = [a+b+c for a in lst1 for b in lst2 for c in ['@', '!']]  
```**  
\*\*Options:\*\*  
- ['x<@', 'y>@']  
- ['x<@', 'x<!', 'x>@', 'x>!', 'y<@', 'y<!', 'y>@', 'y>!', 'z<@', 'z<!', 'z>@', 'z>!']  
- ['@x', '@y', '@z']  
- ['<x!', '>x!']  
  
\*\*Explanation:\*\*  
This code creates a list by combining \*\*letters\*\* (`lst1`), \*\*symbols\*\* (`lst2`), and special characters (`['@', '!']`).  
- For each `a` in `lst1` (`'x', 'y', 'z'`):  
 - For each `b` in `lst2` (`'<', '>'`):  
 - For each `c` in `['@', '!']`:  
 Combine `a + b + c`.  
  
\*\*Step-by-step:\*\*  
- When `a = 'x'`: Combine `'x<@', 'x<!', 'x>@', 'x>!'`.  
- When `a = 'y'`: Combine `'y<@', 'y<!', 'y>@', 'y>!'`.  
- When `a = 'z'`: Combine `'z<@', 'z<!', 'z>@', 'z>!'`.

**Answer:**

\*\*Final list:\*\* `['x<@', 'x<!', 'x>@', 'x>!', 'y<@', 'y<!', 'y>@', 'y>!', 'z<@', 'z<!', 'z>@', 'z>!']`.

## 12. Study the code and decide which are TRUE.

```python  
class Animal:  
 def \_\_init\_\_(self, name):  
 self.name = name  
 def speak(self):  
 return 'Some sound'  
  
class Dog(Animal):  
 def \_\_init\_\_(self, name, breed):  
 super().\_\_init\_\_(name)  
 self.breed = breed  
 def speak(self):  
 return 'Bark'  
```  
\*\*Options:\*\*  
- Dog inherits from Animal.  
- The `speak()` method in Dog overrides the `speak()` method in Animal.  
- Dog does not inherit from Animal.  
- The `speak()` method in Dog and Animal are the same.  
  
\*\*Explanation:\*\*  
- \*\*Inheritance:\*\* Dog gets all the features of Animal.  
- \*\*Override:\*\* The Dog class changes the `speak()` method to say `'Bark'` instead of `'Some sound'`.  
Imagine Animal is a template, and Dog adds specific features (like breed and sound).

**Answer:**

- Dog inherits from Animal.  
- The `speak()` method in Dog overrides the `speak()` method in Animal.

## 13. What is the predicate logic in the truth table?

\*\*Options:\*\*  
- OR  
- AND  
- XOR  
- NOT  
  
\*\*Explanation:\*\*  
- The \*\*OR\*\* logic means 'true if at least one input is true.'  
 - `True OR False = True`.  
 - `False OR False = False`.  
  
\*\*Logic Diagram Example:\*\*  
```  
 A B A OR B  
 0 0 0  
 0 1 1  
 1 0 1  
 1 1 1  
```

**Answer:**

**OR**

## 14. Which algorithms can traverse a graph?

\*\*Options:\*\*  
- Breadth-First Search (BFS)  
- Depth-First Search (DFS)  
- Bubble Sort  
- Quick Sort  
  
\*\*Explanation:\*\*  
- \*\*BFS:\*\* Visits all nodes one level at a time.  
- \*\*DFS:\*\* Goes as deep as possible before backtracking.  
Sorting algorithms like Bubble Sort and Quick Sort are not for graphs.

**Answer:**

- Correct: Breadth-First Search (BFS)  
- Correct: Depth-First Search (DFS)

## 15. What is the output of this code?

**```python  
result = {}  
for i in range(5):  
 for j in range(10):  
 result[i] = j  
print(result)  
```**  
\*\*Options:\*\*  
- {0: 9, 1: 9, 2: 9, 3: 9, 4: 9}  
- {0: 0, 1: 1, 2: 2, 3: 3, 4: 4}  
- {0: 10, 1: 10, 2: 10, 3: 10, 4: 10}  
- {0: 1, 1: 2, 2: 3, 3: 4, 4: 5}  
  
\*\*Explanation:\*\*  
The last value of `j` (9) overwrites all previous values for each `i`.  
\*\*Final result:\*\* `{0: 9, 1: 9, 2: 9, 3: 9, 4: 9}`.

**Answer:**

- Correct: {0: 9, 1: 9, 2: 9, 3: 9, 4: 9}

## 16. Which control structure is used for repetition?

\*\*Options:\*\*  
- While loops  
- For loops  
- If-else  
- Try-except  
  
\*\*Explanation:\*\*  
- \*\*While loops:\*\* Repeat until a condition is false.  
- \*\*For loops:\*\* Repeat for a specific number of times.

**Answer:**

- Correct: While loops  
- Correct: For loops

## 18. What does this code do?

**```python  
def calc\_factorial(x):  
 if x == 1:  
 return 1  
 else:  
 return x \* calc\_factorial(x-1)  
print(calc\_factorial(4))  
```**  
\*\*Options:\*\*  
-Calculates factorial using recursion.  
- Loops through numbers.  
- Adds numbers.  
- Uses iteration.  
  
  
Imagine you’re building a tower with blocks. To finish the tower, you need to stack blocks one at a time, starting from the bottom (block 1) and working your way up to block 4.  
  
\*\*This code is like saying:\*\*  
1. 'If I only have 1 block, I’m done!'  
2. 'If I have more than 1 block, I need to put one block on top and then ask myself: How many blocks do I need for the rest of the tower?'  
  
\*\*Step-by-step for `calc\_factorial(4):\*\*  
- For 4 blocks: `4 \* calc\_factorial(3)`.  
- For 3 blocks: `3 \* calc\_factorial(2)`.  
- For 2 blocks: `2 \* calc\_factorial(1)`.  
- For 1 block: 'I’m done!' Result = 1.  
  
Now stack them together: `4 \* 3 \* 2 \* 1 = 24`.

**Answer:**

- Correct: Calculates factorial using recursion.

## 20. What are the contents of this list?

*```python  
lst = [(s, len(s)) for s in ('apple', 'banana', 'fish')]  
```*  
\*\*Options:\*\*  
- [('apple', 5), ('banana', 6), ('fish', 4)]  
- [('apple', 4), ('banana', 6), ('fish', 3)]  
- [('apple', 'banana', 'fish'), (5, 6, 4)]  
- [('apple', 5), ('fish', 4)]  
  
\*\*Explanation:\*\*  
This code creates a list of \*\*pairs\*\*, where each pair contains:  
1. A word from `('apple', 'banana', 'fish')`.  
2. The number of letters in the word, calculated by `len(s)`.  
  
\*\*What is `len(s)`?\*\*  
- `len()` means 'length' and tells us how many characters (letters, spaces, or symbols) are in a string.  
 - `'apple'` has \*\*5 letters\*\*.  
 - `'banana'` has \*\*6 letters\*\*.  
 - `'fish'` has \*\*4 letters\*\*.  
  
\*\*Step-by-step:\*\*  
- For `'apple'`: Pair is `('apple', 5)`.  
- For `'banana'`: Pair is `('banana', 6)`.  
- For `'fish'`: Pair is `('fish', 4)`.  
  
**Answer:**

\*\*Final list:\*\* `[('apple', 5), ('banana', 6), ('fish', 4)]`.