Name- Satyam Singh Virat

Total Experience- 0 (Fresher)

**Answer 1 (Logic Programming) -**

Well, we can simply iterate over the main string(variable1) and check if the current substring from ith index to the i+length of substring(variable2), exists at each position.

So we can do it as follows-

def find\_index(main\_str, sub\_str):

for i in range(len(main\_str) - len(sub\_str) + 1):

if variable1[i:i+ len (variable2)] == variable2:

return i

return -1

variable1 = “gifts are good to hold”

variable2 = “good”

print find\_index(variable1, variable2))

**Answer 2 (Query Optmization)-**

Here we can pusht the filter conditions to subqueries to reduce operations being performed on rows that would improve efficiency -

SELECT \*

FROM emp e

JOIN department d ON e.department\_id = d.id

JOIN cities c ON d.city = c.city

**Answer 3 (File Operations)-**

I believe we can try to get the desired output from the given data like this -

employees = {}

with open('employees.csv') as f:

next(f)

for line in f:

emp\_id, salary, dept = line.strip().split()

salary = int(salary.replace(',', ''))

if dept in employees:

employees[dept]['total'] += 1

employees[dept]['highest'] = max(employees[dept]['highest'], salary)

else:

employees[dept] = {'total': 1, 'highest': salary}

print("Department TotalEmployee HighestSalary")

for dept, data in employees.items():

print(f"{dept} {data['total']} {data['highest']:,}")

**Answer 4 (Recc func) -**

Here, I first created a list and iterated over the numbers from 2 to 30 while still checking it that number is prime or not -  
  
def is\_prime(n, i=2):

if n < 2 or n % i == 0:

return False

elif n == 2 or i \* i > n:

return True

return is\_prime(n, i + 1)

def primes\_upto(n):

return [i for i in range(2, n+1) if is\_prime(i)]

print(primes\_upto(30))

**Answer 5 (DS) -**   
  
I am not exactly sure how to answer this but i believe i need to create a custom data structure that is simialr to a tree type of data structure. So i think we can do it using dicts in a parent-child style analogy like this-   
  
tree = {

'Root': ['Child1', 'Child2'],

'Child1': ['Grandchild1', 'Grandchild2'],

'Child2': ['Grandchild3']

}

**Answer 6 (Memory management)-**

Var1 = [1, 2, 3] - A new list object var1 is created

Var2 = Var1 – Var1 and var2 both referes to the same memory location

Var1.append(4) – the new appended list var1 will also affect var2 as they both point to the same memory locatino

Var2 = (1, 2) – var2 now refers to a new tuple

Var2[0] = 3 – tupples are immutable, hence will rasie an error