Python and SQL Interview Questions

# Python Questions on Conditional Statements

## Q1: Dynamic Conditional Expression Builder

Write a Python function `dynamic\_evaluator` that takes a list of tuples as an argument. Each tuple contains a condition as a string (which evaluates to either True or False) and a corresponding action as a callable (function). The function should dynamically evaluate each condition in order, and for the first condition that evaluates to True, it should execute the corresponding action. If none of the conditions evaluate to True, it should return `None`.

def dynamic\_evaluator(conditions\_actions):

for condition, action in conditions\_actions:

if eval(condition):

return action()

return None

# Example usage:

def action1():

print("Condition 1 is True")

def action2():

print("Condition 2 is True")

conditions\_actions = [

("2 == 2", action1),

("3 > 5", action2)

]

dynamic\_evaluator(conditions\_actions)

## Q2: Custom Control Structure

Implement a Python class `SwitchCase` that mimics the switch-case control structure found in other programming languages, as Python does not have a native switch-case statement. This class should allow adding cases with conditions and corresponding actions (functions to execute). It should also allow executing the first matching case's action when a value is checked against the class.

class SwitchCase:

def \_\_init\_\_(self):

self.cases = []

def add\_case(self, condition, action):

self.cases.append((condition, action))

def execute(self, value):

for condition, action in self.cases:

if condition(value):

return action()

return None

# Example usage:

def case1(value):

print("Case 1:", value)

def case2(value):

print("Case 2:", value)

switch = SwitchCase()

switch.add\_case(lambda x: x == 1, lambda: case1(1))

switch.add\_case(lambda x: x == 2, lambda: case2(2))

switch.execute(2)

## Q3: Recursive Conditional Parsing

Given a nested list structure that represents a set of conditions and logical operators (AND, OR) in a recursive manner, write a Python function `evaluate\_conditions` that parses and evaluates this structure. For example, the input `['AND', [True, 'OR', [False, True]]]` should evaluate to True.

def evaluate\_conditions(conditions):

if isinstance(conditions, bool):

return conditions

operator = conditions[0]

operands = conditions[1:]

if operator == 'AND':

return all(evaluate\_conditions(operand) for operand in operands)

elif operator == 'OR':

return any(evaluate\_conditions(operand) for operand in operands)

elif operator in [True, False]: # Handle boolean values within operands

return operator

else:

raise ValueError("Invalid operator: {}".format(operator))

# Example usage

nested\_list = ['AND', ['OR', [False, True], False], True, False]

result = evaluate\_conditions(nested\_list)

print(result) # Output: True

output:

False

## Q4: Conditional Dependency Resolver

Create a Python script that takes a dictionary representing tasks and their dependencies on other tasks' successful completion. Your script should determine an order in which to execute the tasks so that all dependencies are respected. If no such order exists (due to circular dependencies), the script should raise an exception.

def resolve\_dependencies(tasks):

graph = {task: set(dependencies) for task, dependencies in tasks.items()}

result = []

while graph:

independent\_tasks = [task for task, dependencies in graph.items() if not dependencies]

if not independent\_tasks:

raise Exception("Circular dependency detected!")

for task in independent\_tasks:

result.append(task)

graph.pop(task)

for dependencies in graph.values():

dependencies.discard(task)

return result

# Example usage

tasks = {

'A': ['B', 'C'],

'B': ['D'],

'C': ['D'],

'D': []

}

try:

execution\_order = resolve\_dependencies(tasks)

print("Execution order:", execution\_order)

except Exception as e:

print("Error:", e)

Output: Execution order: ['D', 'B', 'C', 'A']

# SQL Questions on Triggers

## Q1: Audit Log Trigger

Write an SQL statement to create a trigger in a PostgreSQL database that automatically inserts a record into an `audit\_log` table every time a row in the `employees` table is updated. The log should record the employee ID, the old salary, the new salary, and the timestamp of the change.

CREATE OR REPLACE FUNCTION audit\_employee\_update()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO audit\_log (employee\_id, old\_salary, new\_salary, change\_timestamp)

VALUES (OLD.employee\_id, OLD.salary, NEW.salary, CURRENT\_TIMESTAMP);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER employee\_update\_trigger

AFTER UPDATE ON employees

FOR EACH ROW

EXECUTE FUNCTION audit\_employee\_update();

## Q2: Referential Integrity Trigger

Assume a `students` table and a `registrations` table in a MySQL database, where each registration is supposed to link to an existing student. Create a trigger that runs before an insert on the `registrations` table to check if the student ID exists in the `students` table. If the student ID does not exist, the trigger should prevent the insert operation and raise an error.

CREATE TRIGGER before\_insert\_registration

BEFORE INSERT ON registrations

FOR EACH ROW

BEGIN

DECLARE student\_count INT;

SELECT COUNT(\*) INTO student\_count FROM students WHERE student\_id = NEW.student\_id;

IF student\_count = 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Error: Student ID does not exist';

END IF;

END;

## Q3: Dynamic Trigger for Soft Delete

In a SQL Server database, design a trigger for the `products` table that intercepts delete operations. Instead of allowing a physical delete, the trigger should convert the operation into an update that sets a `deleted` column (a datetime indicating when the row was deleted) to the current timestamp, effectively implementing a soft delete mechanism.

CREATE TRIGGER soft\_delete\_products

ON products

INSTEAD OF DELETE

AS

BEGIN

UPDATE products

SET deleted = GETDATE()

WHERE product\_id IN (SELECT product\_id FROM deleted);

END;