### **Python Control Structures**

1. **If-Else Statement**: Write a program to check if a number is positive, negative, or zero.

**Ans**:-

def check\_number(number):

if number > 0:

return "The number is positive."

elif number < 0:

return "The number is negative."

else:

return "The number is zero."

number = float(input("Enter a number: "))

result = check\_number(number)

print(result)

1. **For Loop**: Write a program to print the first 10 natural numbers using a for loop.

**Ans**:-

print("the first natural numbers=")

for i in range(1,11):

print(i)

1. **While Loop**: Write a program to print the Fibonacci sequence up to the nth term using a while loop.

**Ans**:-

n=7

num1 = 0

num2 = 1

next\_number = num2

count = 1

print("Fibonacci sequence:", end=" ")

while count <= n:

print(next\_number, end=" ")

count += 1

num1, num2 = num2, next\_number

next\_number = num1 + num2

print()

1. **Nested Loop**: Write a program to print a multiplication table from 1 to 10.

**Ans**:-

print("Tables:")

for i in range(1, 11):

for j in range(1, 11):

print(i, '\*', j, '=', i \* j)

1. **List Comprehension**: Write a program to generate a list of squares of numbers from 1 to 10 using list comprehension.

**Ans**:-

squares = [x \*\* 2 for x in range(1, 11)]

print("List of squares of numbers from 1 to 10:")

print(squares)

1. **Dictionary Comprehension**: Write a program to create a dictionary from two lists: one of keys and one of values.

**Ans**:-

def Dictionary\_Comprehension(keys, values):

return {keys[i]: values[i] for i in range(len(keys))}

keys = ["sn", "id", "pin"]

values = [1, 222, 515701]

result = Dictionary\_Comprehension(keys, values)

print (result)

1. **Break and Continue**: Write a program to iterate over a list of numbers and print each number. If you encounter the number 5, break the loop. If you encounter the number 3, skip to the next iteration.

**Ans:-**

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for num in numbers:

if num == 5:

print("Encountered 5. Breaking the loop.")

break

if num == 3:

print("Encountered 3. Skipping to the next iteration.")

continue

print("Current number:", num)

print("Loop finished.")

1. **Functions**: Write a function to calculate the factorial of a number using recursion.

**Ans:-**

def factorial(n):

if n == 1:

return 1

else:

return n \* factorial(n - 1)

num = 6

print("number : ",num)

print("Factorial : ",factorial(num))

1. **Lambda Function**: Write a program to sort a list of tuples based on the second element using a lambda function.

**Ans:-**

list\_of\_tuples = [(2, 10), (2, 5), (3, 8), (4, 2), (5, 6)]

sorted\_list = sorted(list\_of\_tuples, key=lambda x: x[1])

print("Sorted list of tuples based on the second element:")

print(sorted\_list)

1. **Exception Handling**: Write a program to handle a division by zero exception.

**Ans**:-

a = 10

b = 0

if b != 0:

result = a / b

else:

print("Error: Division by zero!")

### **Python Data Structures**

1. **Lists**: Create a list of the first 10 prime numbers. Write functions to add an element, remove an element, and find an element in the list.

**Ans:-**

ef is\_prime(n):

if n <= 1:

return False

elif n == 2:

return True

elif n % 2 == 0:

return False

else:

for i in range(3, int(n \*\* 0.5) + 1, 2):

if n % i == 0:

return False

return True

def first\_n\_primes(n):

primes = []

num = 2 # Start checking from 2

while len(primes) < n:

if is\_prime(num):

primes.append(num)

num += 1

return primes

prime\_numbers = first\_n\_primes(10)

print("List of the first 10 prime numbers:", prime\_numbers)

def add\_element(lst, element):

if is\_prime(element):

lst.append(element)

print("Element", element, "added successfully.")

else:

print("Error: Element is not a prime number.")

def remove\_element(lst, element):

if element in lst:

lst.remove(element)

print("Element", element, "removed successfully.")

else:

print("Error: Element not found in the list.")

def find\_element(lst, element):

if element in lst:

print("Element", element, "found in the list at index", lst.index(element))

else:

print("Element", element, "not found in the list.")

add\_element(prime\_numbers, 31)

add\_element(prime\_numbers, 33)

remove\_element(prime\_numbers, 13)

remove\_element(prime\_numbers, 15)

find\_element(prime\_numbers, 7)

find\_element(prime\_numbers, 15)

1. **Tuples**: Create a tuple with 5 different elements. Write a program to access and print each element in the tuple.

**Ans**:-

A= (12, 'nageswari', 3.14, True, 'hello')

for x in A:

print (x)

1. **Dictionaries**: Create a dictionary with keys as student names and values as their scores. Write a program to find the student with the highest score.

**Ans**:-

student\_scores = {'sasi': 95,'manu': 94,'bindu': 88,'anu': 90,'nag': 60}

highest\_score\_student = max(student\_scores, key=student\_scores.get)

highest\_score = student\_scores[highest\_score\_student]

print("Student with the highest score:", highest\_score\_student)

print("Score:", highest\_score)

1. **Sets**: Create two sets of integers. Write a program to find their union, intersection, and difference.

**Ans**:-

set1 = {10, 20, 30, 40, 50}

set2 = {40, 50, 60, 70, 80}

union\_result = set1.union(set2)

intersection\_result = set1.intersection(set2)

difference\_set1\_set2 = set1.difference(set2)

difference\_set2\_set1 = set2.difference(set1)

print("Union of the sets:", union\_result)

print("Intersection of the sets:", intersection\_result)

print("Difference of set1 - set2:", difference\_set1\_set2)

print("Difference of set2 - set1:", difference\_set2\_set1)

1. **Stacks**: Implement a stack using a list. Write functions for push, pop, and peek operations.

**Ans**:-

stack = []

stack.append(10)

stack.append(15)

stack.append(20)

print('Initial stack')

print(stack)

print('\nElements popped from stack:')

print(stack.pop())

print(stack.pop())

print(stack.pop())

print('\nStack after elements are popped:')

print(stack)

1. **Queues**: Implement a queue using a list. Write functions for enqueue and dequeue operations.

**Ans**:-

lass Queue:

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

self.queue = []

def enqueue(self, item):

self.queue.append(item)

def dequeue(self):

if self.is\_empty():

print("Error: Queue is empty. Cannot dequeue.")

return None

return self.queue.pop(0)

def is\_empty(self):

return len(self.queue) == 0

queue = Queue()

queue.enqueue(1)

queue.enqueue(2)

queue.enqueue(3)

print("Current queue:", queue.queue)

dequeued\_item = queue.dequeue()

print("Dequeue")

1. **Linked List**: Implement a singly linked list with methods to insert, delete, and display elements.

**Ans**:-

class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def insert(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

return

last\_node = self.head

while last\_node.next:

last\_node = last\_node.next

last\_node.next = new\_node

def delete(self, data):

if self.head is None:

print("Error: Linked list is empty. Cannot delete.")

return

if self.head.data == data:

self.head = self.head.next

return

prev\_node = self.head

current\_node = self.head.next

while current\_node:

if current\_node.data == data:

prev\_node.next = current\_node.next

return

prev\_node = current\_node

current\_node = current\_node.next

print("Error: Node with data", data, "not found.")

def display(self):

"""Display the elements of the linked list."""

if self.head is None:

print("Linked list is empty.")

return

current\_node = self.head

while current\_node:

print(current\_node.data, end=" ")

current\_node = current\_node.next

print()

linked\_list = LinkedList()

linked\_list.insert(10)

linked\_list.insert(20)

linked\_list.insert(30)

linked\_list.insert(40)

print("Linked list after insertions:")

linked\_list.display()

linked\_list.delete(30)

print("Linked list after deletion of 3:")

linked\_list.display()

linked\_list.delete(2)

linked\_list.delete(10)

print("Linked list after deletion of 1:")

linked\_list.display()

1. **Binary Tree**: Implement a binary tree with methods for in-order, pre-order, and post-order traversal.

**Ans**:-

class Node:

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

self.left = None

self.right = None

self.val = key

class BinaryTree:

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

self.root = None

def in\_order\_traversal(self, node):

result = []

if node:

result = self.in\_order\_traversal(node.left)

result.append(node.val)

result = result + self.in\_order\_traversal(node.right)

return result

def pre\_order\_traversal(self, node):

result = []

if node:

result.append(node.val)

result = result + self.pre\_order\_traversal(node.left)

result = result + self.pre\_order\_traversal(node.right)

return result

def post\_order\_traversal(self, node):

result = []

if node:

result = self.post\_order\_traversal(node.left)

result = result + self.post\_order\_traversal(node.right)

result.append(node.val)

return result

if \_\_name\_\_ == "\_\_main\_\_":

tree = BinaryTree()

tree.root = Node(1)

tree.root.left = Node(2)

tree.root.right = Node(3)

tree.root.left.left = Node(4)

tree.root.left.right = Node(5)

tree.root.right.left = Node(6)

tree.root.right.right = Node(7)

print("In-order traversal:", tree.in\_order\_traversal(tree.root))

print("Pre-order traversal:", tree.pre\_order\_traversal(tree.root))

print("Post-order traversal:", tree.post\_order\_traversal(tree.root))

1. **Graphs**: Represent a graph using an adjacency list and write a program to perform a breadth-first search (BFS).

**Ans**:-

from collections import defaultdict, deque

class Graph:

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

self.adjacency\_list = defaultdict(list)

def add\_edge(self, u, v):

self.adjacency\_list[u].append(v)

self.adjacency\_list[v].append(u)

def bfs(self, start\_node):

visited = set()

queue = deque([start\_node])

while queue:

node = queue.popleft()

if node not in visited:

print(node, end=" ")

visited.add(node)

for neighbor in self.adjacency\_list[node]:

if neighbor not in visited:

queue.append(neighbor)

graph = Graph()

graph.add\_edge(0, 1)

graph.add\_edge(0, 2)

graph.add\_edge(1, 2)

graph.add\_edge(2, 3)

graph.add\_edge(3, 3)

print("BFS starting from node 2:")

graph.bfs(2)

1. **Hash Table**: Implement a basic hash table with functions for inserting, deleting, and searching elements.

**Ans**:-

class HashTable:  
    def \_\_init\_\_(self, size):  
        self.size = size  
        self.table = [None] \* size  
    def \_hash(self, key):  
        return hash(key) % self.size  
    def insert(self, key, value):  
        index = self.\_hash(key)  
        if self.table[index] is None:  
            self.table[index] = [(key, value)]  
        else:  
            self.table[index].append((key, value))  
    def delete(self, key):  
        index = self.\_hash(key)  
        if self.table[index] is not None:  
            for i, (existing\_key, existing\_value) in enumerate(self.table[index]):  
                if existing\_key == key:  
                    del self.table[index][i]  
                    break  
    def search(self, key):  
        index = self.\_hash(key)  
        if self.table[index] is not None:  
            for existing\_key, existing\_value in self.table[index]:  
                if existing\_key == key:  
                    return existing\_value  
        return None  
hash\_table = HashTable(10)  
hash\_table.insert("apple", 20)  
hash\_table.insert("banana", 30)  
hash\_table.insert("cherry", 40)  
print("Search 'banana':", hash\_table.search("banana"))  
print("Search 'grape':", hash\_table.search("grape"))  
hash\_table.delete("banana")  
print("Search 'banana' after deletion:", hash\_table.search("banana"))

### **SQL DDL (Data Definition Language)**

1. **Create Table**: Write a SQL statement to create a table named Books with columns id, title, author, and price.

**Ans**:-

mysql>create table books (id int,title varchar(20),price int,author varchar (20));

Query OK, 0 rows affected (0.04 sec)

1. **Drop Table**: Write a SQL statement to drop the Books table.

**Ans**:-

mysql> drop table books;

Query OK, 0 rows affected (0.02 sec)

1. **Alter Table**: Write a SQL statement to add a new column published\_date to the Books table.

**Ans**:

mysql> alter table books add published\_date date;

Query OK, 0 rows affected (0.03 sec)

Records: 0 Duplicates: 0 Warnings: 0

1. **Rename Table**: Write a SQL statement to rename the Books table to LibraryBooks.

**Ans**:-

alter table books rename to LibraryBooks;

Query OK, 0 rows affected (0.02 sec)

1. **Create Index**: Write a SQL statement to create an index on the author column of the Books table.

**Ans**:-

### mysql> CREATE INDEX idx\_author ON Books(author);

### Query OK, 0 rows affected (0.03 sec)

### Records: 0 Duplicates: 0 Warnings: 0

### **SQL DML (Data Manipulation Language)**

1. **Insert Data**: Write a SQL statement to insert a new record into the Books table.

**Ans**:-

mysql> insert into books values (111,'Frankenstein',1200,'1818-02-01','Mary Shelley');

Query OK, 1 row affected (0.01 sec)

mysql> insert into books values (222,'Carmilla',1300,'1872-02-01','Sheridan Le Fanu'),(333,'Dracula',1500,'1897-03-03','Bram Stoker'),(444,'Rebecca',1400,'1938-04-05','Daphne'),(555,'Legend',1600,'1982-06-07','Richard');

Query OK, 4 rows affected (0.01 sec)

Records: 4 Duplicates: 0 Warnings: 0

mysql> select \*from books;

+------+--------------+-------+----------------+------------------+

| id | title | price | published\_date | author |

+------+--------------+-------+----------------+------------------+

| 111 | Frankenstein | 1200 | 1818-02-01 | Mary Shelley |

| 222 | Carmilla | 1300 | 1872-02-01 | Sheridan Le Fanu |

| 333 | Dracula | 1500 | 1897-03-03 | Bram Stoker |

| 444 | Rebecca | 1400 | 1938-04-05 | Daphne |

| 555 | Legend | 1600 | 1982-06-07 | Richard |

+------+--------------+-------+----------------+------------------+

1. **Update Data**: Write a SQL statement to update the price of a book in the Books table based on its id.

**Ans**:-

update books set price =2000 where id =111;

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select \*from books;

+------+--------------+-------+----------------+------------------+

| id | title | price | published\_date | author |

+------+--------------+-------+----------------+------------------+

| 111 | Frankenstein | 2000 | 1818-02-01 | Mary Shelley |

| 222 | Carmilla | 1300 | 1872-02-01 | Sheridan Le Fanu |

| 333 | Dracula | 1500 | 1897-03-03 | Bram Stoker |

| 444 | Rebecca | 1400 | 1938-04-05 | Daphne |

| 555 | Legend | 1600 | 1982-06-07 | Richard |

+------+--------------+-------+----------------+------------------+

1. **Delete Data**: Write a SQL statement to delete a book from the Books table based on its id.

**Ans**:-

mysql> delete from books where id =222;

Query OK, 1 row affected (0.00 sec)

mysql> select \*from books;

+------+--------------+-------+----------------+--------------+

| id | title | price | published\_date | author |

+------+--------------+-------+----------------+--------------+

| 111 | Frankenstein | 2000 | 1818-02-01 | Mary Shelley |

| 333 | Dracula | 1500 | 1897-03-03 | Bram Stoker |

| 444 | Rebecca | 1400 | 1938-04-05 | Daphne |

| 555 | Legend | 1600 | 1982-06-07 | Richard |

+------+--------------+-------+----------------+--------------+

1. **Select Data**: Write a SQL statement to select all records from the Books table.

**Ans**:-

mysql> select \*from books;

+------+--------------+-------+----------------+--------------+

| id | title | price | published\_date | author |

+------+--------------+-------+----------------+--------------+

| 111 | Frankenstein | 2000 | 1818-02-01 | Mary Shelley |

| 333 | Dracula | 1500 | 1897-03-03 | Bram Stoker |

| 444 | Rebecca | 1400 | 1938-04-05 | Daphne |

| 555 | Legend | 1600 | 1982-06-07 | Richard |

+------+--------------+-------+----------------+--------------+

4 rows in set (0.00 sec)

1. **Join Tables**: Write a SQL statement to join two tables, Books and Authors, on a common column and select relevant data.

**Ans**:-

mysql> select \*from books;

+------+--------------+-------+----------------+--------------+

| id | title | price | published\_date | author |

+------+--------------+-------+----------------+--------------+

| 111 | Frankenstein | 2000 | 1818-02-01 | Mary Shelley |

| 333 | Dracula | 1500 | 1897-03-03 | Bram Stoker |

| 444 | Rebecca | 1400 | 1938-04-05 | Daphne |

| 555 | Legend | 1600 | 1982-06-07 | Richard |

+------+--------------+-------+----------------+--------------+

4 rows in set (0.00 sec)

mysql> select \*from authors;

+------+------------+-------+---------------+------------------+

| id | title | price | released\_date | author |

+------+------------+-------+---------------+------------------+

| 666 | Psycho | 10000 | 1959-06-01 | Robert |

| 777 | Hell House | 3000 | 1971-01-01 | Richard Matheson |

+------+------------+-------+---------------+------------------+

2 rows in set (0.00 sec)

mysql> insert into books (id,title,price, published\_date,author) select id,title,price,released\_date,author from authors;

Query OK, 2 rows affected (0.01 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> select \*from books;

+------+--------------+-------+----------------+------------------+

| id | title | price | published\_date | author |

+------+--------------+-------+----------------+------------------+

| 111 | Frankenstein | 2000 | 1818-02-01 | Mary Shelley |

| 333 | Dracula | 1500 | 1897-03-03 | Bram Stoker |

| 444 | Rebecca | 1400 | 1938-04-05 | Daphne |

| 555 | Legend | 1600 | 1982-06-07 | Richard |

| 666 | Psycho | 10000 | 1959-06-01 | Robert |

| 777 | Hell House | 3000 | 1971-01-01 | Richard Matheson |

+------+--------------+-------+----------------+------------------+

6 rows in set (0.00 sec)