**Robo Manipal Coding Task phase**

Q1

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

if number % 2 == 0:

print("Even")

else:

print("Odd")

--------------------------------------------------------------------------------------------------------------------------------------

Q2

number = 1234

sum = 0

while number > 0:

sum += number % 10

number //= 10

print(sum)

--------------------------------------------------------------------------------------------------------------------------------------

Q3

num = [4, 7, 1, 8, 5]

largest\_number = num[0]

for i in num:

if i > largest\_number:

largest\_number = i

print(largest\_number)

Q4

c = ['H', 'e', 'l', 'l', 'o']

new = ""

for i in c:

new += i

print(new)

--------------------------------------------------------------------------------------------------------------------------------------

Q5

is\_prime = True

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

if num % i == 0:

is\_prime = False

break

if is\_prime:

print("Prime")

else:

print("Not Prime")

--------------------------------------------------------------------------------------------------------------------------------------

Q6

nested\_list = [1, [2, [3, 4], 5], 6]

flattened\_list = []

stack = nested\_list.copy()

while stack:

element = stack.pop(0) # Take the first element from the stack

if isinstance(element, list):

stack = element + stack

else:

flattened\_list.append(element)

print(flattened\_list)

--------------------------------------------------------------------------------------------------------------------------------------

Q7

set1 = {1, 2, 3, 4}

set2 = {3, 4, 5, 6}

common = set()

for i in set1:

if i in set2:

common.add(i)

print(common)

--------------------------------------------------------------------------------------------------------------------------------------

Q8

str1 = "Listen"

str2 = "Silent"

new\_str1 = sorted(str1.lower())

new\_str2 = sorted(str2.lower())

if new\_str1 == new\_str2:

print("Anagram")

else:

print("Not an Anagram")

--------------------------------------------------------------------------------------------------------------------------------------

Q9

list1 = [1, 2, 3, 4, 5]

list2 = [4, 5, 6, 7]

intersection = []

for element in list1:

if element in list2:

intersection.append(element)

print(intersection)

--------------------------------------------------------------------------------------------------------------------------------------

Q10

data = [

{"name": "John", "age": 30},

{"name": "Jane", "age": 25},

{"name": "Doe", "age": 40}

]

def get\_age(dictionary):

return dictionary['age']

sorted\_data = sorted(data, key=get\_age)

print(sorted\_data)

--------------------------------------------------------------------------------------------------------------------------------------

Q11

keys = ['name', 'age', 'city']

values = ['Alice', 30, 'New York']

dictionary = {}

for i in range(len(keys)):

dictionary[keys[i]] = values[i] # Assign the value from values list to the corresponding key

print(dictionary)

--------------------------------------------------------------------------------------------------------------------------------------

Q12

x = "Python is fun"

words = x.split()

length = [len(word) for i in words]

print(length)

--------------------------------------------------------------------------------------------------------------------------------------

Q13

arr = [10, 22, 9, 33, 21, 50, 41, 60, 80]

dp = [1] \* len(arr)

for i in range(1, len(arr)):

for j in range(i):

if arr[i] > arr[j]:

dp[i] = max(dp[i], dp[j] + 1)

length\_of\_lis = max(dp)

print(length\_of\_lis)

--------------------------------------------------------------------------------------------------------------------------------------

Q13

class Node:

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

self.data = data

self.next = None

class SinglyLinkedList:

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

self.head = None

def insert(self, data):

new\_node = Node(data)

if not self.head:

self.head = new\_node

else:

current = self.head

while current.next:

current = current.next

current.next = new\_node

def delete(self, data):

current = self.head

previous = None

while current and current.data != data:

previous = current

current = current.next

if current:

if previous:

previous.next = current.next

else:

self.head = current.next

--------------------------------------------------------------------------------------------------------------------------------------

Q14

def traverse(self):

elements = []

current = self.head

while current:

elements.append(current.data)

current = current.next

return elements

linked\_list = SinglyLinkedList()

# Insert elements

linked\_list.insert(10)

linked\_list.insert(20)

linked\_list.insert(30)

# Delete an element

linked\_list.delete(20)

# Traverse the linked list and print the result

elements = linked\_list.traverse()

print("Linked List:", " -> ".join(map(str, elements)))

--------------------------------------------------------------------------------------------------------------------------------------

Q15

class Stack:

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

self.stack = []

def push(self, item):

self.stack.append(item)

def pop(self):

if not self.is\_empty():

return self.stack.pop()

else:

raise IndexError("Pop from an empty stack")

def is\_empty(self):

return len(self.stack) == 0

def \_\_str\_\_(self):

return str(self.stack)

# Create a stack and perform operations

stack = Stack()

# Push elements

stack.push(5)

stack.push(10)

# Pop an element

stack.pop()

# Check if stack is empty

is\_empty = stack.is\_empty()

# Output the result

print("Stack after operations:", stack)

print("Is stack empty?", is\_empty)

--------------------------------------------------------------------------------------------------------------------------------------