**Assignment 3:-**

**Q1**.

List comprehensions are a **compact and elegant way** to create lists in Python.

They allow you to generate a new list by **iterating over a sequence**, applying **transformations** and **filters** in a **single line of code**.

**Difference from Traditional for Loops:**

| **Feature** | **Traditional for Loop** | **List Comprehension** |
| --- | --- | --- |
| **Syntax** | Multi-line, explicit append() | One-liner with clean syntax |
| **Readability** | More verbose | More concise |
| **Performance** | Slightly slower (function call overhead) | Generally faster (internally optimized) |
| **Flexibility** | Can do more complex logic | Best for simple transformations |
| **squares = []**  **for i in range(10):**  **squares.append(i \*\*2)**  **squares = [i \*\* 2 for i in range(10)]**  **Q2**.  **Two Main Forms:**  **1. Filtering only:**  python  CopyEdit  [expression for item in iterable if condition]   * This **includes** items **only if** the condition is True.   **2. With if-else (conditional expression):**  [true\_expr if condition else false\_expr for item in iterable]   * This evaluates **every item**, but **transforms** them based on the condition.   **Example 1 – *Filter only*:**  # Get odd numbers from 1 to 10  odds = [x for x in range(1, 11) if x % 2 != 0]  # ➞ [1, 3, 5, 7, 9]  **Example 2 – *Conditional expression*:**  # Label numbers from 1 to 5 as 'even' or 'odd'  labels = ['even' if x % 2 == 0 else 'odd' for x in range(1, 6)]  # ➞ ['odd', 'even', 'odd', 'even', 'odd']  **Coding Challenge**  **Create a list comprehension that generates a list of all even numbers between 1 and 50.**  **Solution:**  evens = [x for x in range(1, 51) if x % 2 == 0]  **Output:**  print(evens)  print(evens)  # ➞ [2, 4, 6, 8, ..., 50]  Q3.  **. if (Filtering Only)**  This is used to **include only elements** that meet a condition.  python  CopyEdit  [x for x in iterable if condition]  It **filters** elements.  **2. if-else (Conditional Expression)**  This is used when you want to **transform elements** based on a condition — keep all elements, but **change their values**.  [true\_expr if condition else false\_expr for x in iterable]  It **modifies** each item based on the condition.  **Example Comparison:**  **Filtering (if only)**  # Keep only even numbers  [x for x in range(10) if x % 2 == 0]  # ➞ [0, 2, 4, 6, 8]  **Conditional transformation (if-else):**  # Label numbers as 'even' or 'odd'  ['even' if x % 2 == 0 else 'odd' for x in range(5)]  # ➞ ['even', 'odd', 'even', 'odd', 'even']  **Coding Challenge:**  **Write a list comprehension that replaces every negative number in a list with 0, leaving positive numbers unchanged.**  **Solution:**  def replace\_negatives(lst):  return [x if x >= 0 else 0 for x in lst]  **Example:**  nums = [4, -3, 0, -1, 7, -9]  print(replace\_negatives(nums))  # ➞ [4, 0, 0, 0, 7, 0]  Q4.  matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  flattened = []  for row in matrix:  for item in row:  flattened.append(item)  flattened = [item for row in matrix for item in row]  matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  flattened = [item for row in matrix for item in row]  print(flattened)  Q5.  **Limitations:**   | **Limitation** | **Explanation** | | --- | --- | | **Readability** | Too many nested for and if clauses can make them hard to read. | | **Debugging Difficulty** | Harder to insert breakpoints, print() statements, or comments. | | **One-liner Pressure** | Encourages packing complex logic into one line, which can be confusing. | | **Side Effects** | Not ideal if your loop has side effects (like logging or I/O). | | **Not for all logic** | If logic is conditional, multi-step, or needs exception handling — prefer a loop. |   result = [x \* y for x in range(1, 10) for y in range(1, 10) if x \* y % 2 == 0]  result = []  for x in range(1, 10):  for y in range(1, 10):  product = x \* y  if product % 2 == 0:  result.append(product)  Q6.  [function(x) for x in iterable]  [function(x) for x in iterable if condition]  def square(n):  return n \* n  numbers = [1, 2, 3, 4]  squares = [square(x) for x in numbers]  print(squares) # ➞ [1, 4, 9, 16]  nums = [-5, -1, 0, 3, -8, 7]  absolute\_values = [abs(x) for x in nums]  print(absolute\_values)  output:  [5, 1, 0, 3, 8, 7]  Q7.  if you reuse the **same variable name**, it can **shadow or override** variables in the outer scope within the list comprehension, though it won't change the outer variable *after* the comprehension.  **Key Points:**   | **Behavior** |  | | --- | --- | | Variables inside list comprehension **do not leak** into outer scope |  | | If outer variable has same name, it’s **shadowed temporarily** |  | | Outer variable **retains its original value** |  |   x = 10 # outer variable  # List comprehension uses the same variable name  squares = [x \* x for x in range(5)]  print("List comprehension result:", squares)  print("Value of x after comprehension:", x)  output:-  List comprehension result: [0, 1, 4, 9, 16]  Value of x after comprehension: 10  Q8.  **Difference Between List Comprehensions and Generator Expressions**   | **Feature** | **List Comprehension** | **Generator Expression** | | --- | --- | --- | | Syntax | [x for x in iterable] | (x for x in iterable) | | Output | **List** (stored in memory) | **Generator object** (lazy evaluation) | | Memory usage | Stores **entire list** in memory | Generates **items one at a time** (efficient) | | Performance | Fast for small/medium data | Better for **large datasets** | | Use case | When you need **all results at once** | When you want to **iterate** or **stream** data |   multiples\_of\_three = [x for x in range(100) if x % 3 == 0]  multiples\_of\_three = (x for x in range(100) if x % 3 == 0)  Q9.  [transformation for item in iterable if condition]  data = [1, 'hello', 3, 'world', 5, '42', 7]  squared\_integers = [x\*\*2 for x in data if isinstance(x, int)]  print(squared\_integers)  output:-  [1, 9, 25, 49]  Q10.  **Syntax Differences:**   | **Comprehension Type** | **Syntax Format** | | --- | --- | | **List Comprehension** | [expression for item in iterable if condition] | | **Set Comprehension** | {expression for item in iterable if condition} | | **Dict Comprehension** | {key\_expr: value\_expr for item in iterable if condition} |   s = "hello"  ascii\_map = {char: ord(char) for char in s}  print(ascii\_map)  output:-  {'h': 104, 'e': 101, 'l': 108, 'o': 111} |  |  |