Python - Collections, Functions And Modules

Theory Assignment

1. Accessing List

- Q: Understanding how to create and access elements in a list.

A: In Python, lists are ordered collections of items that can contain different data types. A list is created using square brackets []. For example: my_list = [10, 'Hello', 3.14]. Each element can be accessed using an index number. Indexing starts from 0 for the first element. Negative indexing can be used to access elements from the end, where -1 represents the last element. Slicing allows access to a range of elements using the format list[start:end]. Example: my_list[1:3] gives elements from index 1 to 2.

- Q: Indexing in lists (positive and negative indexing).

A: Positive indexing starts from 0, whereas negative indexing starts from -1 from the end. For instance, my_list[-1] gives the last element.

- Q: Slicing a list: accessing a range of elements.

A: Slicing is performed using a colon operator (:). Example: numbers[0:3] returns the first three elements. Omitting start or end defaults to the first or last index respectively.

2. List Operations

- Q: Common list operations: concatenation, repetition, membership.

A: Lists support operations like concatenation (+) to combine two lists, repetition (*) to repeat elements, and membership (in) to check if an element exists. Example: 3 in [1,2,3].

- Q: Understanding list methods like append(), insert(), remove(), pop().

A: These methods modify lists dynamically. append() adds an element to the end, insert() adds at a specific index, remove() deletes a specific element, and pop() removes the last or indexed element.

3. Working with Lists

- Q: Iterating over a list using loops.

A: Lists are iterable; you can use for loops to access each element. Example: for item in my_list: print(item).

- Q: Sorting and reversing a list using sort(), sorted(), and reverse().

A: The sort() method arranges elements in ascending order permanently, while sorted() returns a new sorted list. reverse() changes the order of elements to the opposite direction.

- Q: Basic list manipulations: addition, deletion, updating, and slicing.

A: Lists are mutable. Elements can be added using append() or insert(), removed using remove() or del, updated via direct assignment, and sliced using list[start:end].

4. Tuple

- Q: Introduction to tuples, immutability.

A: Tuples are ordered collections like lists but are immutable, meaning their values cannot be changed once created. They are defined using parentheses (). Example: my_tuple = (1, 2, 3).

- Q: Creating and accessing elements in a tuple.

A: You can access elements via indexing, e.g., my_tuple[0] for the first element.

- Q: Basic operations with tuples: concatenation, repetition, membership.

A: Tuples support +, *, and in operators similar to lists but cannot be modified after creation.

5. Accessing Tuples

- Q: Accessing tuple elements using positive and negative indexing.

A: Like lists, tuples use 0-based indexing for positive and -1-based indexing for negative access. Example: t1[-1] gives the last element.

- Q: Slicing a tuple to access ranges of elements.

A: Tuples support slicing syntax (start:end) to extract a part of the tuple without modifying it.

6. Dictionaries

- Q: Introduction to dictionaries: key-value pairs.

A: Dictionaries are unordered collections of data stored as key-value pairs inside curly braces {}. Example: student = {'name':'John','age':20}. Each key must be unique.

- Q: Accessing, adding, updating, and deleting dictionary elements.

A: Access values via keys using student['name'], add by assignment student['city']='Delhi', update by reassigning existing key, and delete using del student['age'].

- Q: Dictionary methods like keys(), values(), and items().

A: keys() returns all keys, values() returns all values, and items() returns key-value pairs as tuples.

7. Working with Dictionaries

- Q: Iterating over a dictionary using loops.

A: You can iterate over keys, values, or both using loops. Example: for key, value in my_dict.items(): print(key, value).

- Q: Merging two lists into a dictionary using loops or zip().

A: zip() pairs corresponding elements of two lists into key-value pairs. Example: dict(zip(keys, values)).

- Q: Counting occurrences of characters in a string using dictionaries.

A: By iterating through a string, each character can be used as a key and its count as a value. Example: for ch in text: dict[ch] = dict.get(ch,0)+1.

8. Functions

- Q: Defining functions in Python.

A: Functions are reusable blocks of code defined using def keyword. Example: def greet(): print('Hello'). Functions promote modularity and code reuse.

- Q: Different types of functions: with/without parameters, with/without return values.

A: Functions can accept arguments (parameters) and may or may not return values using the return statement. Example: def add(a,b): return a+b.

- Q: Anonymous functions (lambda functions).

A: Lambda functions are single-line anonymous functions defined with lambda keyword. Example: square = lambda x: x*x.

9. Modules

- Q: Introduction to Python modules and importing modules.

A: Modules are files containing Python code (functions, classes, variables) that can be reused. They are imported using import keyword. Example: import math.

- Q: Standard library modules: math, random.

A: The math module provides mathematical functions (sqrt, pow, factorial), while random provides functions for generating random numbers.

- Q: Creating custom modules.

A: A custom module is a Python file created by the user, which can be imported into other scripts. Example: save a file as mymodule.py and import it using import mymodule.