**Module 14: Python – Collections, Functions, and Modules** .

**1) Accessing List**

**Theory**

* Lists are ordered, mutable collections that can store mixed data types.
* Indexing:
  + Positive indexing starts at 0 from the left.
  + Negative indexing starts at -1 from the right.
* Slicing: list[start🔚step] returns a new list with selected elements (end is exclusive).

**Lab Solutions**

1. Create a list with elements of multiple data types

mixed\_list = [42, "hello", 3.14, True, None, [1, 2], ("a", "b")]  
print(mixed\_list)

1. Access elements at different index positions

lst = ['apple', 'banana', 'mango', 'kiwi', 'grape']  
print(lst[^0]) # first  
print(lst) # third  
print(lst[-1]) # last  
print(lst[-3]) # third from end

**Practical Examples**

1. Create a list of multiple data type elements

data = [10, "python", 2.5, False, {"key": "value"}]  
print(data)

1. Find the length of a list using len()

items = [1, 2, 3, 4, 5, 6]  
print(len(items))

**2) List Operations**

**Theory**

* Common operations:
  + Concatenation: a + b
  + Repetition: a \* n
  + Membership: x in a
* Useful methods: append(), insert(), remove(), pop(), extend(), clear()

**Lab Solutions**

1. Add elements using insert() and append()

nums = [1, 2, 3]  
nums.append(4) # [1, 2, 3, 4]  
nums.insert(1, 99) # [1, 99, 2, 3, 4]  
print(nums)

1. Remove elements using pop() and remove()

fruits = ['apple', 'banana', 'mango', 'banana']  
fruits.remove('banana') # removes first 'banana'  
popped = fruits.pop(1) # removes element at index 1  
print(fruits, popped)

**Practical Examples**

1. Update list using insert() and append()

lst = [10, 20, 30]  
lst.append(40)  
lst.insert(0, 5)  
print(lst)

1. Remove elements using pop() and remove()

lst = ['x', 'y', 'z', 'y']  
lst.remove('y') # first 'y'  
last = lst.pop() # removes last  
print(lst, last)

**3) Working with Lists**

**Theory**

* Iteration: for x in list
* Sorting:
  + list.sort() sorts in place, returns None
  + sorted(list) returns a new sorted list
* Slicing and updates work due to mutability

**Lab Solutions**

1. Iterate over a list using a for loop

languages = ['Python', 'C', 'C++', 'Java']  
for lang in languages:  
 print(lang)

1. Sort using sort() and sorted()

nums = [5, 2, 9, 1]  
nums\_sorted = sorted(nums) # new list  
print(nums\_sorted, nums) # original unchanged  
nums.sort(reverse=True) # in-place descending  
print(nums)

**Practical Examples**

1. Iterate through a list and print each element

colors = ['red', 'green', 'blue']  
for c in colors:  
 print(c)

1. Insert elements into an empty list using a for loop and append()

result = []  
for i in range(5):  
 result.append(i \* i)  
print(result)

**4) Tuple**

**Theory**

* Tuples are ordered, immutable sequences.
* Can contain mixed data types.
* Support indexing, slicing, membership, concatenation, repetition.

**Lab Solutions**

1. Create a tuple with multiple data types

t = (1, "two", 3.0, False, None)  
print(t)

1. Concatenate two tuples

a = (1, 2)  
b = ('x', 'y')  
c = a + b  
print(c)

**Practical Examples**

1. Convert a list into a tuple

lst = [1, 2, 3]  
tup = tuple(lst)  
print(tup)

1. Create a tuple with multiple data types

t = ("python", 3.11, True)  
print(t)

1. Concatenate two tuples into one

t1 = (10, 20)  
t2 = (30, 40)  
t3 = t1 + t2  
print(t3)

1. Access the value of the first index in a tuple

t = ('a', 'b', 'c')  
print(t[^0])

**5) Accessing Tuples**

**Theory**

* Positive/negative indexing works like lists.
* Slicing: t[start🔚step] returns a new tuple.

**Lab Solutions**

1. Access values between index 1 and 5 in a tuple

t = (0, 1, 2, 3, 4, 5, 6)  
print(t[1:6]) # elements at 1,2,3,4,5

1. Access alternate values between index 1 and 5

t = (0, 1, 2, 3, 4, 5, 6)  
print(t[1:6:2]) # 1,3,5

**Practical Examples**

1. Access values between index 1 and 5

t = ('a', 'b', 'c', 'd', 'e', 'f', 'g')  
print(t[1:6])

1. Access the value from the last index in a tuple

t = (10, 20, 30)  
print(t[-1])

**6) Dictionaries**

**Theory**

* Dicts store key-value pairs.
* Keys must be hashable (immutable types commonly used).
* Core methods: keys(), values(), items(), get(), update(), pop()

**Lab Solutions**

1. Create a dictionary with 6 key-value pairs

student = {  
 "id": 101,  
 "name": "Asha",  
 "age": 20,  
 "dept": "CS",  
 "cgpa": 8.7,  
 "active": True  
}  
print(student)

1. Access values using dictionary keys

print(student["name"])  
print(student.get("cgpa")) # safe access

**Practical Examples**

1. Create a dictionary of 6 key-value pairs

config = {  
 "host": "localhost",  
 "port": 5432,  
 "debug": True,  
 "retries": 3,  
 "timeout": 30.0,  
 "env": "dev"  
}  
print(config)

1. Access values using keys

print(config["host"])  
print(config.get("missing", "default-value"))

**7) Working with Dictionaries**

**Theory**

* Iterate:
  + for key in d
  + for key, value in d.items()
* Merge two lists into dict with zip()
* Frequency counting with dict

**Lab Solutions**

1. Update a value in a dictionary

profile = {"name": "Ravi", "city": "Pune"}  
profile["city"] = "Mumbai"  
profile.update({"age": 25})  
print(profile)

1. Merge two lists into one dictionary using a loop

keys = ["id", "name", "age"]  
vals = [1, "Kiran", 22]  
merged = {}  
for i in range(min(len(keys), len(vals))):  
 merged[keys[i]] = vals[i]  
print(merged)

**Practical Examples**

1. Update a value at a particular key

d = {"a": 1, "b": 2}  
d["b"] = 200  
print(d)

1. Separate keys and values using keys() and values()

d = {"x": 10, "y": 20, "z": 30}  
keys\_list = list(d.keys())  
values\_list = list(d.values())  
print(keys\_list, values\_list)

1. Convert two lists into one dictionary using a for loop

countries = ["IN", "US", "UK"]  
capitals = ["New Delhi", "Washington, D.C.", "London"]  
mapping = {}  
for i in range(len(countries)):  
 mapping[countries[i]] = capitals[i]  
print(mapping)

1. Count how many times each character appears in a string

s = "banana"  
freq = {}  
for ch in s:  
 freq[ch] = freq.get(ch, 0) + 1  
print(freq) # {'b':1,'a':3,'n':2}

**8) Functions**

**Theory**

* Functions are defined with def.
* Types:
  + With/without parameters
  + With/without return value
* Lambda: small anonymous functions for simple expressions.

**Lab Solutions**

1. Function that takes a string and prints it

def echo(text):  
 print(text)  
  
echo("Hello, Functions!")

1. Calculator using functions

def add(a, b): return a + b  
def sub(a, b): return a - b  
def mul(a, b): return a \* b  
def div(a, b): return a / b  
  
op = '+'  
a, b = 10, 5  
ops = {'+': add, '-': sub, '\*': mul, '/': div}  
result = ops[op](a, b)  
print(result)

**Practical Examples**

1. Print a string using a function

def show():  
 print("Hello from function!")  
show()

1. Parameterized function to print sum

def print\_sum(x, y):  
 print(x + y)  
print\_sum(3, 4)

1. Lambda function with one expression

square = lambda x: x \* x  
print(square(6))

1. Lambda function with two expressions (two-argument lambda)

power = lambda a, b: a \*\* b  
print(power(2, 5))

**9) Modules**

**Theory**

* Modules are files containing Python code (functions, variables, classes).
* Importing: import module, from module import name
* Standard library: math, random, datetime, etc.
* Custom modules: any .py file importable via PYTHONPATH.

**Lab Solutions**

1. Import math and use sqrt(), ceil(), floor()

import math  
print(math.sqrt(25))  
print(math.ceil(2.1))  
print(math.floor(2.9))

1. Generate random numbers using random

import random  
print(random.randint(1, 100)) # inclusive  
print(random.random()) # [0.0, 1.0)  
print(random.choice(['a', 'b', 'c']))

**Practical Examples**

1. Demonstrate functions from math module

import math  
values = [3.7, -4.2, 16]  
print(math.fabs(values[^1]))  
print(math.sqrt(values))  
print(math.floor(values), math.ceil(values))

1. Generate random numbers between 1 and 100

import random  
nums = [random.randint(1, 100) for \_ in range(5)]  
print(nums)