**Python Backend Development Assignment (Humanized)**

### **Module 1: Overview of IT Industry**

…[Module 1 content as-is]…

### **Module 14: Python Collections, Functions and Modules**

#### **Accessing Lists**

**Theory:**  
In Python, a list is a collection of ordered items which can be of different data types like integers, strings, or even other lists. Lists are mutable, meaning you can change their contents. You access items using index numbers, where the first index is 0. Negative indexing allows access from the end of the list.

**Lab:**

# Create a list with multiple data types  
my\_list = [10, "hello", 3.14, True]  
print("List:", my\_list)  
  
# Access elements  
print("First element:", my\_list[0])  
print("Last element:", my\_list[-1])

#### **List Operations**

**Theory:**  
Python lists come with many built-in operations. You can add new elements using append() to the end or insert() at a specific index. You can remove elements using pop() (removes by index) or remove() (removes by value). These operations are useful for dynamic list management.

**Lab:**

# Add elements  
my\_list.append("new")  
my\_list.insert(1, "inserted")  
  
# Remove elements  
my\_list.pop() # removes last  
my\_list.remove("hello") # removes value "hello"

#### **Working with Lists**

**Theory:**  
You can loop through lists using for loops. Sorting can be done using the sort() method or the built-in sorted() function. Lists are flexible and used heavily in data storage, processing, and iteration tasks.

**Lab:**

# Iterate and sort  
numbers = [5, 2, 9, 1]  
for num in numbers:  
 print(num)  
  
print("Sorted:", sorted(numbers))  
numbers.sort()  
print("In-place sorted:", numbers)

#### **Tuples**

**Theory:**  
Tuples are like lists but immutable, which means you can’t change their values once defined. They use parentheses () instead of square brackets []. Tuples are often used for fixed collections of items.

**Lab:**

t1 = (1, "apple", 3.14)  
t2 = (4, 5)  
concatenated = t1 + t2  
print(concatenated)

#### **Accessing Tuples**

**Theory:**  
Just like lists, you can access tuple elements using indexes. You can also use slicing to retrieve ranges of items, and negative indexing is supported too.

**Lab:**

tuple1 = (10, 20, 30, 40, 50, 60)  
print("Between index 1 and 5:", tuple1[1:6])  
print("Alternate values:", tuple1[1:6:2])

#### **Dictionaries**

**Theory:**  
Dictionaries store data in key-value pairs. Keys must be unique and can be of any immutable type. They’re perfect for situations where you want to associate one piece of data (value) with another (key).

**Lab:**

d = {"name": "Alice", "age": 25, "city": "Rajkot"}  
print("Name:", d["name"])  
d["age"] = 26 # update

#### **Working with Dictionaries**

**Theory:**  
You can loop through dictionaries using for loops to access keys and values. The get() method helps avoid errors when accessing non-existing keys. Dictionaries are excellent for structured data.

**Lab:**

# Count character frequency  
text = "banana"  
freq = {}  
for char in text:  
 freq[char] = freq.get(char, 0) + 1  
print(freq)

#### **Functions**

**Theory:**  
Functions in Python help group reusable code. You define them with def, and they can take parameters and return results using return. Lambda functions are small anonymous functions for quick operations.

**Lab:**

def greet(name):  
 print(f"Hello, {name}!")  
  
greet("John")  
  
# Lambda function  
add = lambda x, y: x + y  
print(add(5, 3))

#### **Modules**

**Theory:**  
Modules are Python files with functions or classes you can reuse in other programs. Python provides many built-in modules like math, and you can import custom or external modules too.

**Lab:**

import math  
print("Square root:", math.sqrt(16))  
  
import random  
print("Random number:", random.randint(1, 100))

### **Module 15: Advanced Python Programming**

#### **1. Printing on Screen**

**Theory:**  
print() is used to show output. You can format strings using f-strings for clarity and readability.

name = "Alice"  
age = 25  
print(f"My name is {name} and I am {age} years old.")

#### **2. Reading Data from Keyboard**

**Theory:**  
You can take input using input(). Convert it to other types like int() or float() when needed.

name = input("Enter your name: ")  
age = int(input("Enter your age: "))  
print(f"Name: {name}, Age: {age}")

#### **3. Opening and Closing Files**

**Theory:**  
Use open() with modes like 'r', 'w', 'a' to read or write files. Always close the file or use with for safety.

f = open("example.txt", "w")  
f.write("Hello, file!")  
f.close()

#### **4. Reading and Writing Files**

**Theory:**  
read(), readline(), and readlines() are used to read files. write() writes string content to files.

# Write  
with open("data.txt", "w") as f:  
 f.write("Python is fun!\n")  
  
# Read  
with open("data.txt", "r") as f:  
 content = f.read()  
 print(content)

#### **5. Exception Handling**

**Theory:**  
Exceptions are handled using try, except, finally blocks. This prevents crashes and allows graceful error messages.

try:  
 num = int(input("Enter a number: "))  
 result = 100 / num  
except ZeroDivisionError:  
 print("Can't divide by zero!")  
except ValueError:  
 print("Invalid input!")  
finally:  
 print("Program completed.")

#### **6. Class and Object**

**Theory:**  
A class is a blueprint for creating objects. Objects are instances of a class with their own data and methods.

class Person:  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
p = Person("Alice")  
print(p.name)

#### **7. Inheritance**

**Theory:**  
Inheritance allows one class to use the properties and methods of another. It’s useful for code reuse.

class Animal:  
 def speak(self):  
 print("Animal speaks")  
  
class Dog(Animal):  
 def speak(self):  
 print("Dog barks")  
  
pet = Dog()  
pet.speak()

#### **8. Method Overloading and Overriding**

**Theory:**  
Overriding means redefining a method in a child class. Overloading (not directly supported like C++) is done using default or variable arguments.

# Overriding  
class A:  
 def show(self):  
 print("A")  
  
class B(A):  
 def show(self):  
 print("B")  
  
B().show()

#### **9. SQLite3 and PyMySQL (only SQLite3 shown)**

**Theory:**  
SQLite3 is a lightweight database embedded in Python. You can use SQL commands to create and manage databases.

import sqlite3  
conn = sqlite3.connect("test.db")  
cursor = conn.cursor()  
cursor.execute("CREATE TABLE IF NOT EXISTS student (id INTEGER, name TEXT)")  
cursor.execute("INSERT INTO student VALUES (1, 'Alice')")  
conn.commit()  
for row in cursor.execute("SELECT \* FROM student"):  
 print(row)  
conn.close()

#### **10. Search and Match Functions**

**Theory:**  
Python’s re module provides search() (anywhere) and match() (only beginning). They’re useful in pattern matching.

import re  
  
# Search  
text = "Hello Python"  
match = re.search("Python", text)  
print("Found!" if match else "Not Found")  
  
# Match  
match = re.match("Hello", text)  
print("Matched!" if match else "Not Matched")

**[Next: Django + REST Framework modules if needed]**