**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:** Lists are ordered collections of items which can be of any data type. Elements can be accessed using indexes (both positive and negative).

**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:** You can add elements using append() or insert(), and remove them with pop() or remove().

**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:** Lists can be iterated, sorted, reversed, and modified easily.

**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 immutable sequences. Once created, their contents cannot be changed.

**Lab:**

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

#### **Accessing Tuples**

**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 are key-value pairs. Keys must be unique.

**Lab:**

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

#### **Working with Dictionaries**

**Lab:**

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

#### **Functions**

**Theory:** Functions are reusable blocks of code. They can take arguments and return values. Lambda functions are anonymous one-liners.

**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 files containing Python code. Built-in modules like math and random provide useful utilities.

**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**

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

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

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

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

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

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

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**

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

#### **7. Inheritance**

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**

# 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)**

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**

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]**