1. **Demonstrate Step-by-step Installation Process.**

Step-by-Step Installation Process for Python

1. Installing Python on Windows

Step 1: Download Python

1. Go to the official Python website.
2. Navigate to the "Downloads" section and select "Windows".
3. Click on the latest stable release version of Python.

Step 2: Run the Installer

1. Once the download is complete, open the installer.
2. Check the box that says "Add Python to PATH". This is important for accessing Python from the command line.
3. Choose "Install now" to install Python with default settings or "Customize installation" for advanced options.

Step 3: Verify the Installation

1. Open Command Prompt.
2. Type python --version and press Enter. You should see the version of Python that you installed.

2. Implement a program base on Strings.(slicing, casting, string function)

```
sentence = input("Enter a sentence: ")

# String slicing

first_word = sentence.split()[0]  # Getting the first word

middle_section = sentence[5:15]   # Getting a slice of the string from index 5 to 14

last_five_chars = sentence[-5:]   # Getting the last 5 characters

# String casting

number_str = "123"

number = int(number_str)  # Casting string to integer

# String functions

uppercase_sentence = sentence.upper()  # Converting to uppercase

lowercase_sentence = sentence.lower()   # Converting to lowercase

title_sentence = sentence.title()      # Converting to title case

word_count = sentence.count(" ") + 1   # Counting words based on spaces

print("Original Sentence:", sentence)

print("First Word:", first_word)

print("Middle Section (5 to 14):", middle_section)

print("Last 5 Characters:", last_five_chars)

print("Casted Number:", number)

print("Uppercase Sentence:", uppercase_sentence)

print("Lowercase Sentence:", lowercase_sentence)

print("Title Case Sentence:", title_sentence)

print("Word Count:", word_count)
```

Output:

```
Enter a sentence: bhagyashri patil

Original Sentence: bhagyashri patil

First Word: bhagyashri
```

Middle Section (5 to 14): ashri pati

Last 5 Characters: patil

Casted Number: 123

Uppercase Sentence: BHAGYASHRI PATIL

Lowercase Sentence: bhagyashri patil

Title Case Sentence: Bhagyashri Patil

Word Count: 2

Trimmed Sentence: bhagyashri patil

Replaced Sentence: bhagyashri patil

3. **Implement a program based on Simple statements using Variables, Built-in Data Types, Types of operator, etc.**

```
def simple_statements_demo():
    # Variables and Built-in Data Types
    integer_var = 10  # integer
    float_var = 15.5  # float
    string_var = "Python"  # string
    boolean_var = True  # boolean
    list_var = [1, 2, 3, 4, 5]  # list
    tuple_var = (10, 20, 30)  # tuple
    dict_var = {'name': 'John', 'age': 30}  # dictionary

    # Print variables and their types
    print("Variables and their types:")
    print(f"Integer: {integer_var} (type: {type(integer_var)})")
    print(f"Float: {float_var} (type: {type(float_var)})")
    print(f"String: {string_var} (type: {type(string_var)})")
    print(f"Boolean: {boolean_var} (type: {type(boolean_var)})")
    print(f"List: {list_var} (type: {type(list_var)})")
    print(f"Tuple: {tuple_var} (type: {type(tuple_var)})")
    print(f"Dictionary: {dict_var} (type: {type(dict_var)})")

    # Arithmetic Operators
    print("\nArithmetic Operators:")
    print(f"{integer_var} + {float_var} = {integer_var + float_var}")
    print(f"{integer_var} - {float_var} = {integer_var - float_var}")
    print(f"{integer_var} * {float_var} = {integer_var * float_var}")
    print(f"{integer_var} / {float_var} = {integer_var / float_var}")
    print(f"{integer_var} % 3 = {integer_var % 3}")
    print(f"{integer_var} ** 2 = {integer_var ** 2}")
    print(f"{integer_var} // 3 = {integer_var // 3}")

    # Comparison Operators
    print("\nComparison Operators:")
    print(f"{integer_var} == 10: {integer_var == 10}")
    print(f"{integer_var} != 10: {integer_var != 10}")
    print(f"{integer_var} > 5: {integer_var > 5}")
    print(f"{integer_var} < 15: {integer_var < 15}")
    print(f"{integer_var} >= 10: {integer_var >= 10}")
    print(f"{integer_var} <= 10: {integer_var <= 10}")

    # Logical Operators
    print("\nLogical Operators:")
    print(f"True and False: {True and False}")
    print(f"True or False: {True or False}")
    print(f"not True: {not True}")
    # Assignment Operators
    print("\nAssignment Operators:")
    x = 5
    print(f"x = 5: {x}")
    x += 3
    print(f"x += 3: {x}")
    x -= 2
    print(f"x -= 2: {x}")
    x *= 2
    print(f"x *= 2: {x}")
    x /= 3
    print(f"x /= 3: {x}")
    x %= 2
    print(f"x %= 2: {x}")
    x **= 3
    print(f"x **= 3: {x}")
    x //= 2
    print(f"x //= 2: {x}")

if __name__ == "__main__":
    simple_statements_demo()
```

Output:

```
Arithmetic Operators:
10 + 15.5 = 25.5
10 - 15.5 = -5.5
10 * 15.5 = 155.0
10 / 15.5 = 0.6451612903225806
10 % 3 = 1
10 ** 2 = 100
10 // 3 = 3

Comparison Operators:
10 == 10: True
10 != 10: False
10 > 5: True
10 < 15: True
10 >= 10: True
10 <= 10: True

Logical Operators:
True and False: False
True or False: True
not True: False

Assignment Operators:
x = 5: 5
x += 3: 8
```

```
x -= 2: 6
x *= 2: 12
x /= 3: 4.0
x %= 2: 0.0
x **= 3: 0.0
x //= 2: 0.0
```

Other Program:

```
# Variables and Built-in Data Types
name = "John"       # String
age = 25            # Integer
height = 5.9        # Float
is_student = True   # Boolean
grades = [85, 90, 88] # List (array-like data type)
address = None      # NoneType (for uninitialized variables)

# Printing variables
print("Name:", name)
print("Age:", age)
print("Height:", height)
print("Is Student:", is_student)
print("Grades:", grades)
print("Address:", address)

# Basic Operators
# Arithmetic Operators
addition = age + 5          # Addition
subtraction = age - 3       # Subtraction
multiplication = age * 2    # Multiplication
division = age / 2          # Division (returns float)
integer_division = age // 2  # Integer division (returns int)
modulus = age % 2           # Modulus (remainder)
exponent = age ** 2         # Exponentiation (age^2)

# Comparison Operators
is_adult = age >= 18
is_minor = age < 18

# Logical Operators
is_eligible_for_discount = is_student or age < 18

# String Operators
full_name = name + " Doe"    # String concatenation
repeated_name = name * 3     # String repetition

# List Operations
grades.append(95)           # Adding a grade to the list
average_grade = sum(grades) / len(grades)  # Calculate average grade
```

```
# Output the results
print("\nArithmetic Operations:")
print("Addition (age + 5):", addition)
print("Subtraction (age - 3):", subtraction)
print("Multiplication (age * 2):", multiplication)
print("Division (age / 2):", division)
print("Integer Division (age // 2):", integer_division)
print("Modulus (age % 2):", modulus)
print("Exponent (age ** 2):", exponent)

print("\nComparison & Logical Operations:")
print("Is Adult (age >= 18):", is_adult)
print("Is Minor (age < 18):", is_minor)
print("Is Eligible for Discount:", is_eligible_for_discount)

print("\nString Operations:")
print("Full Name:", full_name)
print("Repeated Name:", repeated_name)

print("\nList Operations:")
print("Updated Grades:", grades)
print("Average Grade:", average_grade)
```

Output:

```
Name: John
Age: 25
Height: 5.9
Is Student: True
Grades: [85, 90, 88]
Address: None

Arithmetic Operations:
Addition (age + 5): 30
Subtraction (age - 3): 22
Multiplication (age * 2): 50
Division (age / 2): 12.5
Integer Division (age // 2): 12
Modulus (age % 2): 1
Exponent (age ** 2): 625

Comparison & Logical Operations:
Is Adult (age >= 18): True
Is Minor (age < 18): False
Is Eligible for Discount: True

String Operations:
Full Name: John Doe
Repeated Name: JohnJohnJohn

List Operations:
Updated Grades: [85, 90, 88, 95]
Average Grade: 89.5
```

4. Implement a program based on decision and looping statements

```
i = 0
# The loop breaks when it reaches 'h'
for letter in 'Python':     # First Example
   if(letter == 'h'):
      print('Yes h is present in string')
      break
   else:
      print ('Current Letter:', letter)
var = 10
while var > 0:
   print('Current variable value: ', var)
   var = var-1
   if var == 5:
      break
print('Good bye!')
```

Output:

```
Current Letter: P
Current Letter: y
Current Letter: t
Yes h is present in string

Current variable value:  10
Current variable value:  9
Current variable value:  8
Current variable value:  7
Current variable value:  6
```

5. **Implement a program based on using Tuples, Lists, Dictionaries, Sets.**

```
# Student Enrollment Program
# List to store students (each student is a tuple)
students = []

# Set to track unique courses
courses = set()

# Dictionary to track enrolled courses for each student
enrollment = {}

# Function to add a student
def add_student(name):
    student = (name,)  # Create a tuple for the student
    students.append(student)  # Add to the list
    enrollment[name] = []  # Initialize their enrolled courses
    print(f"Added student: {name}")

# Function to enroll a student in a course
def enroll_student(name, course):
    if name in enrollment:
        enrollment[name].append(course)
        courses.add(course)  # Add the course to the set
        print(f"{name} enrolled in {course}.")
    else:
        print(f"Student {name} not found.")

# Function to display all students and their courses
def display_students():
    if not students:
        print("No students enrolled.")
    else:
        print("Students and their courses:")
        for student in students:
            name = student[0]
            enrolled_courses = enrollment[name]
            print(f"{name}: {', '.join(enrolled_courses) if enrolled_courses else 'No courses
enrolled'}")

# Main program
def main():
    while True:
        print("\nMenu:")
        print("1. Add Student")
        print("2. Enroll Student in Course")
        print("3. Display Students")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            name = input("Enter student name: ")
            add_student(name)
        elif choice == '2':
            name = input("Enter student name: ")
            course = input("Enter course name: ")
            enroll_student(name, course)
        elif choice == '3':
            display_students()
        elif choice == '4':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

```
Menu:
1. Add Student
2. Enroll Student in Course
3. Display Students
4. Exit
Enter your choice (1-4): 1
Enter student name: bhagyashri
Added student: bhagyashri

Menu:
1. Add Student
2. Enroll Student in Course
3. Display Students
4. Exit
Enter your choice (1-4): 2
Enter student name: bhagyashri
Enter course name: mca
bhagyashri enrolled in mca.
```

```
Menu:
1. Add Student
2. Enroll Student in Course
3. Display Students
4. Exit
Enter your choice (1-4): 3
Students and their courses:
bhagyashri: mca

Menu:
1. Add Student
2. Enroll Student in Course
3. Display Students
4. Exit
Enter your choice (1-4): 1
Enter student name: suchita
Added student: suchita

Menu:
1. Add Student
2. Enroll Student in Course
3. Display Students
4. Exit
Enter your choice (1-4): 3
Students and their courses:
bhagyashri: mca
suchita: No courses enrolled

Menu:
1. Add Student
2. Enroll Student in Course
3. Display Students
4. Exit
```

6. **Implement a program based on using Functions, Function with parameters.**

```
# Function to calculate the sum of two numbers
def add_numbers(a, b):
    """Calculate and return the sum of two numbers."""
    return a + b

# Function to calculate the product of two numbers
def multiply_numbers(a, b):
    """Calculate and return the product of two numbers."""
    return a * b

def main():
    # Get user input
    num1 = float(input("Enter the first number: "))
    num2 = float(input("Enter the second number: "))

    # Call functions and get results
    sum_result = add_numbers(num1, num2)
    product_result = multiply_numbers(num1, num2)

    # Print results
    print(f"The sum of {num1} and {num2} is: {sum_result}")
    print(f"The product of {num1} and {num2} is: {product_result}")

if __name__ == "__main__":
    main()
```

Output:

```
Enter the first number: 10
Enter the second number: 20
The sum of 10.0 and 20.0 is: 30.0
The product of 10.0 and 20.0 is: 200.0
```

7. **Implement a program based on Functions Inside of Functions.**

```
# Basic Arithmetic Operations Program

def arithmetic_operations():
    def add(a, b):
        return a + b

    def multiply(a, b):
        return a * b

    while True:
        print("\nChoose an operation:")
        print("1. Addition")
        print("2. Multiplication")
        print("3. Exit")

        choice = input("Enter your choice (1-3): ")

        if choice == '1':
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
            result = add(num1, num2)
            print(f"The result of addition is: {result}")

        elif choice == '2':
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
            result = multiply(num1, num2)
            print(f"The result of multiplication is: {result}")

        elif choice == '3':
            print("Exiting the program.")
            break

        else:
            print("Invalid choice. Please try again.")

# Run the main function
if __name__ == "__main__":
    arithmetic_operations()
```

Output:

```
Choose an operation:
1. Addition
2. Multiplication
3. Exit
Enter your choice (1-3): 1
Enter first number: 10
Enter second number: 20
The result of addition is: 30.0

Choose an operation:
1. Addition
2. Multiplication
3. Exit
Enter your choice (1-3): 2
Enter first number: 10
Enter second number: 5
The result of multiplication is: 50.0

Choose an operation:
1. Addition
2. Multiplication
3. Exit
Enter your choice (1-3): 3
Exiting the program.
```

8. Implement a program based on built-in functions .

```
def main():

    # Input: Get numbers from the user
    numbers = input("Enter numbers separated by spaces: ")

    # Convert the input string into a list of integers
    number_list = list(map(int, numbers.split()))

    # Calculate the sum using the built-in sum() function
    total_sum = sum(number_list)

    # Find the maximum using the built-in max() function
    max_value = max(number_list)

    # Find the minimum using the built-in min() function
    min_value = min(number_list)

    # Sort the list using the built-in sorted() function
    sorted_list = sorted(number_list)

    # Output the results
    print(f"Sum: {total_sum}")
    print(f"Maximum: {max_value}")
    print(f"Minimum: {min_value}")
    print(f"Sorted List: {sorted_list}")
  # Run the main function
if __name__ == "__main__":
    main()
```

Output:

```
Enter numbers separated by spaces: 20 10 30
Sum: 60
Maximum: 30
Minimum: 10
Sorted List: [10, 20, 30]
```

9. Implement a program using Anonymous Functions - Lambda and Filter.

```
def square(x):
    return x ** 2
numbers = [1, 2, 3, 4]
squared_numbers = map(square, numbers)
print(list(squared_numbers))  # Output: [1, 4, 9, 16]
numbers = [1, 2, 3, 4]
squared_numbers = map(lambda x: x ** 2, numbers)
print(list(squared_numbers))  # Output: [1, 4, 9, 16]
def is_even(x):
    return x % 2 == 0
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = filter(is_even, numbers)
print(list(even_numbers))
```

Output: [2, 4, 6]

10. Implement a program using Maps, List Comprehension, Dictionaries.

```
# Simple Age Conversion Program

def convert_ages_to_months(names, ages):
    # Using map to convert ages to months
    ages_in_months = list(map(lambda age: age * 12, ages))

    # Creating a dictionary using dictionary comprehension

        name_age_dict = {name: age for name, age in zip(names, ages_in_months)}


    return name_age_dict

def main():
    # Input: List of names and corresponding ages
    names = ['Alice', 'Bob', 'Charlie', 'Diana']
    ages = [25, 30, 22, 28]

    # Convert ages to months and create the dictionary
    name_age_dict = convert_ages_to_months(names, ages)

    # Output the results
    print("Name to Age in Months Conversion:")
    for name, age_months in name_age_dict.items():
        print(f"{name}: {age_months} months")

# Run the main function
if __name__ == "__main__":
    main()
```

Output:

```
Name to Age in Months Conversion:

Alice: 300 months

Bob: 360 months

Charlie: 264 months

Diana: 336 months
```

11. Implement a program for Class and Object.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age


    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
    def celebrate_birthday(self):
        self.age += 1
        print(f"Happy Birthday, {self.name}! You are now {self.age} years old.")
person1 = Person("Sanvi", 18)
person2 = Person("Nilansh", 19)
person1.display_info()
print()
person2.display_info()
print()
person1.celebrate_birthday()
person2.celebrate_birthday()
```

Output:

```
Name: Sanvi
Age: 18

Name: Nilansh
Age: 19

Happy Birthday, Sanvi! You are now 19 years old.
Happy Birthday, Nilansh! You are now 20 years old.
```

12. Demonstrate a program using Array in python.

```
# Using Python lists as arrays
def list_example():
    # Create a list (array) of integers
    numbers = [10, 20, 30, 40, 50]
    # Accessing elements
    print("Original list:", numbers)
    print("Element at index 2:", numbers[2])  # Output: 30
    # Modifying an element
    numbers[2] = 35
    print("Modified list:", numbers)
    # Appending a new element
    numbers.append(60)
   print("List after appending 60:", numbers)
    # Removing an element
    numbers.remove(20)
    print("List after removing 20:", numbers)
    # Iterating over the list
    print("Iterating over the list:")
    for num in numbers:
        print(num)
list_example()
```

Output:

```
Original list: [10, 20, 30, 40, 50]
Element at index 2: 30
Modified list: [10, 20, 35, 40, 50]
List after appending 60: [10, 20, 35, 40, 50, 60]
List after removing 20: [10, 35, 40, 50, 60]

Iterating over the list:

10

35

40

50

60
```

13. Implement a program to Create, Import and use Package and Modules.

1. **Create a directory called** my_package1**.**

```
!mkdir my_package1
```

2. **Inside this directory, create a file called __init__.py. This can be empty or contain initialization code. It marks the directory as a package.**

```
%%writefile my_package1/__init__.py
# __init__.py can be empty or contain initialization code
```

Output: Overwriting my_package1/__init__.py

3. **Move the math_operations.py module into the my_package1 directory.**

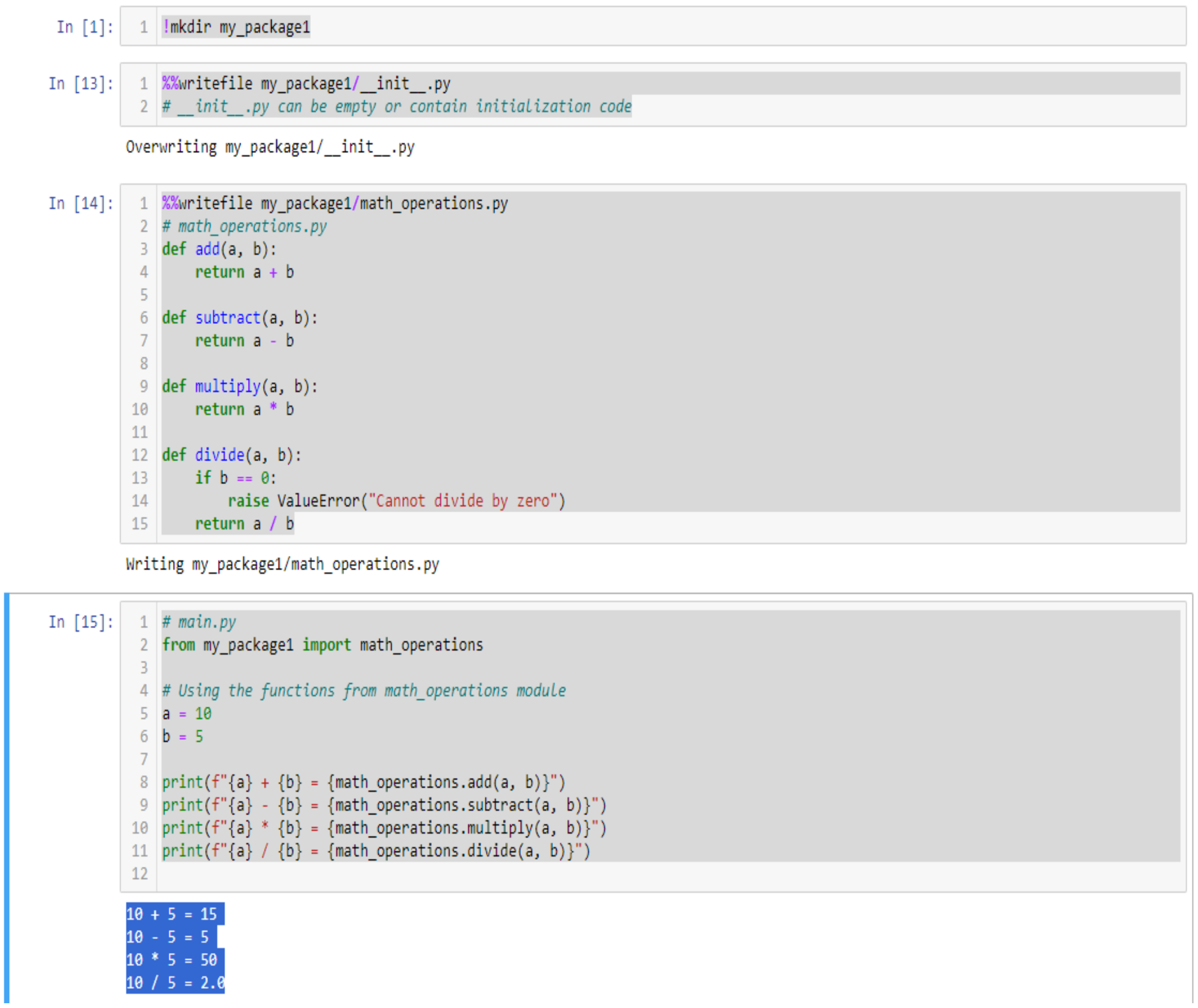
```
%%writefile my_package1/math_operations.py
# math_operations.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
```

4. **# main.py**

```
from my_package1 import math_operations

# Using the functions from math_operations module
a = 10
b = 5
print(f"{a} + {b} = {math_operations.add(a, b)}")
print(f"{a} - {b} = {math_operations.subtract(a, b)}")
print(f"{a} * {b} = {math_operations.multiply(a, b)}")
print(f"{a} / {b} = {math_operations.divide(a, b)}")
```

Output:

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
```



```
In [1]:  !mkdir my_package1
```

```
In [13]: %%writefile my_package1/__init__.py
         # __init__.py can be empty or contain initialization code

Overwriting my_package1/__init__.py
```

```
In [14]: %%writefile my_package1/math_operations.py
         # math_operations.py
         def add(a, b):
             return a + b

         def subtract(a, b):
             return a - b

         def multiply(a, b):
             return a * b

         def divide(a, b):
             if b == 0:
                 raise ValueError("Cannot divide by zero")
             return a / b

Writing my_package1/math_operations.py
```

```
In [15]: # main.py
         from my_package1 import math_operations

         # Using the functions from math_operations module
         a = 10
         b = 5

         print(f"{a} + {b} = {math_operations.add(a, b)}")
         print(f"{a} - {b} = {math_operations.subtract(a, b)}")
         print(f"{a} * {b} = {math_operations.multiply(a, b)}")
         print(f"{a} / {b} = {math_operations.divide(a, b)}")

10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.0
```

This example demonstrates how to create, import, and use a package and modules in Python. The package my_package contains the module math_operations, which is imported and used in the main.py script.

Another Program:

File 1) mymodule.py

```
person1 = {
 "name": "John",
"age": 36,
"country": "Norway"
 }
```

File 2) File2.py

```
import mymodule #mymodule is the first file name
a = mymodule.person1["age"]
print(a)
```

Note: Run File2.py (save File 1 as a name mymodule.py and this name use as a package in File 2)

14. **Implement a program based on Writing Text Files, Appending Text to a File, Reading Text Files.**

```
# Define file path
file_path = 'abc.txt'

# 1. Writing to a Text File
def write_to_file(file_path, text):
    with open(file_path, 'w') as file:
        file.write(text)
    print("Text has been written to the file.")

# 2. Appending Text to a File
def append_to_file(file_path, text):
    with open(file_path, 'a') as file:
        file.write(text)
    print("Text has been appended to the file.")

# 3. Reading from a Text File
def read_from_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
        print("File content read successfully.")
        return content
    except FileNotFoundError:
        print("File not found.")
        return None

# Main program logic
if __name__ == "__main__":
    # Write initial content to the file
    initial_text = "This is the initial text in the file.\n"
    write_to_file(file_path, initial_text)

    # Append additional content to the file
    additional_text = "This is the additional text being appended.\n"
    append_to_file(file_path, additional_text)

    # Read the content of the file
    content = read_from_file(file_path)
    if content is not None:
        print("File Content:")
        print(content)
```

Output:

```
Text has been written to the file.
Text has been appended to the file.
File content read successfully.
File Content:
This is the initial text in the file.
This is the additional text being appended.
```

15. **Demonstrate a program Paths and Directories, File Information, Renaming, Moving, Copying, and Removing Files.**

```
import os
import shutil

# Step 1: Paths and Directories
print("Working with paths and directories")
current_dir = os.getcwd()  # Get the current working directory
print(f"Current Directory: {current_dir}")

# Create a new directory
new_dir = os.path.join(current_dir, "test_dir")
if not os.path.exists(new_dir):
   os.mkdir(new_dir)
   print(f"Directory created: {new_dir}")

# Step 2: File Information
file_name = os.path.join(new_dir, "sample.txt")
with open(file_name, 'w') as f:
   f.write("This is a sample file.")

print("\nFile Information")
print(f"File Path: {file_name}")
print(f"File Exists: {os.path.exists(file_name)}")
print(f"File Size: {os.path.getsize(file_name)} bytes")
print(f"Last Modified: {os.path.getmtime(file_name)}")
print(f"Absolute Path: {os.path.abspath(file_name)}")

# Step 3: Renaming the file
new_file_name = os.path.join(new_dir, "renamed_sample.txt")
os.rename(file_name, new_file_name)
print(f"\nFile Renamed to: {new_file_name}")

# Step 4: Moving the file
moved_file_path = os.path.join(current_dir, "renamed_sample.txt")
shutil.move(new_file_name, moved_file_path)
print(f"File Moved to: {moved_file_path}")

# Step 5: Copying the file
copied_file_path = os.path.join(current_dir, "copied_sample.txt")
shutil.copy(moved_file_path, copied_file_path)
print(f"File Copied to: {copied_file_path}")
```

```
# Step 6: Removing the files
os.remove(moved_file_path)
print(f"File Removed: {moved_file_path}")

os.remove(copied_file_path)
print(f"Copied File Removed: {copied_file_path}")

# Clean up: Remove the directory
os.rmdir(new_dir)
print(f"Directory Removed: {new_dir}")
```

Output:

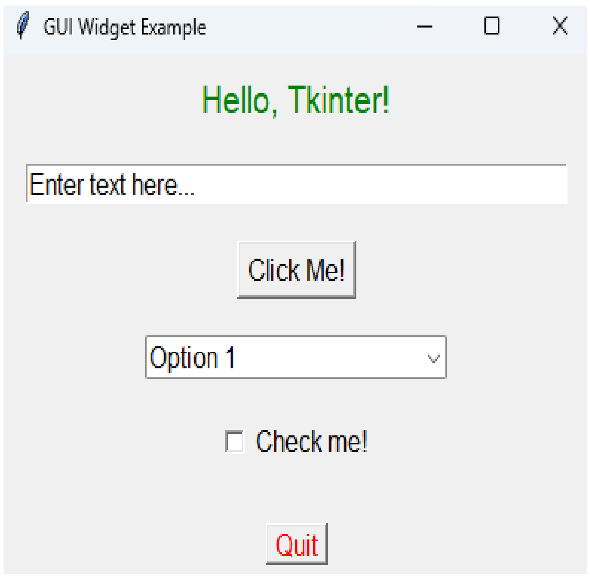
```
Working with paths and directories
Current Directory: C:\Users\imrd

File Information
File Path: C:\Users\imrd\test_dir\sample.txt
File Exists: True
File Size: 22 bytes
Last Modified: 1727155138.016346
Absolute Path: C:\Users\imrd\test_dir\sample.txt

File Renamed to: C:\Users\imrd\test_dir\renamed_sample.txt
File Moved to: C:\Users\imrd\renamed_sample.txt
File Copied to: C:\Users\imrd\copied_sample.txt
File Removed: C:\Users\imrd\renamed_sample.txt
Copied File Removed: C:\Users\imrd\copied_sample.txt
Directory Removed: C:\Users\imrd\test_dir
```

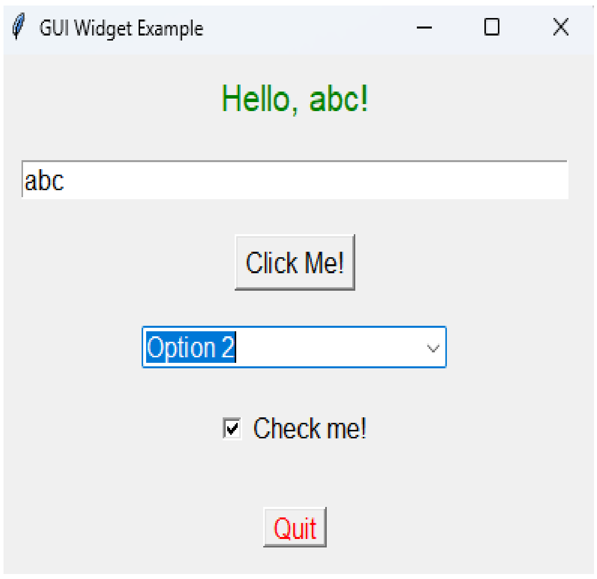
16. **Implement a program to creating Types of GUI Widgets, Resizing, Configuring Options.**

```
import tkinter as tk
from tkinter import ttk

# Initialize the main window
root = tk.Tk()
root.title("GUI Widget Example")

# Resize the window
root.geometry("400x300")

# Step 1: Create GUI Widgets
# Label Widget
label = tk.Label(root, text="Hello, Tkinter!", font=("Arial", 16), fg="blue")
label.pack(pady=10)  # Add some padding for spacing

# Entry Widget (Text box)
entry = tk.Entry(root, width=30, font=("Arial", 12))
entry.insert(0, "Enter text here...")
entry.pack(pady=10)

# Button Widget
def on_button_click():
   label.config(text=f"Hello, {entry.get()}!")  # Update label based on entry text

button = tk.Button(root, text="Click Me!", command=on_button_click, font=("Arial",
12))
button.pack(pady=10)

# Combobox Widget
combo = ttk.Combobox(root, values=["Option 1", "Option 2", "Option 3"],
font=("Arial", 12))
combo.current(0)  # Set default value
combo.pack(pady=10)

# Checkbox Widget
check_var = tk.IntVar()

checkbox = tk.Checkbutton(root, text="Check me!", variable=check_var, font=("Arial",
12))
checkbox.pack(pady=10)
```

```
# Step 2: Resizing and Configuring Options
# Resize widgets using the `pack` method options
# pady and padx are used for vertical and horizontal padding
label.config(fg="green")  # Configure label to change text color
entry.config(width=40)     # Configure entry to change width

# Button to quit the application
quit_button = tk.Button(root, text="Quit", command=root.quit, font=("Arial", 12),
fg="red")
quit_button.pack(pady=20)

# Start the GUI event loop
root.mainloop()
```

Output:



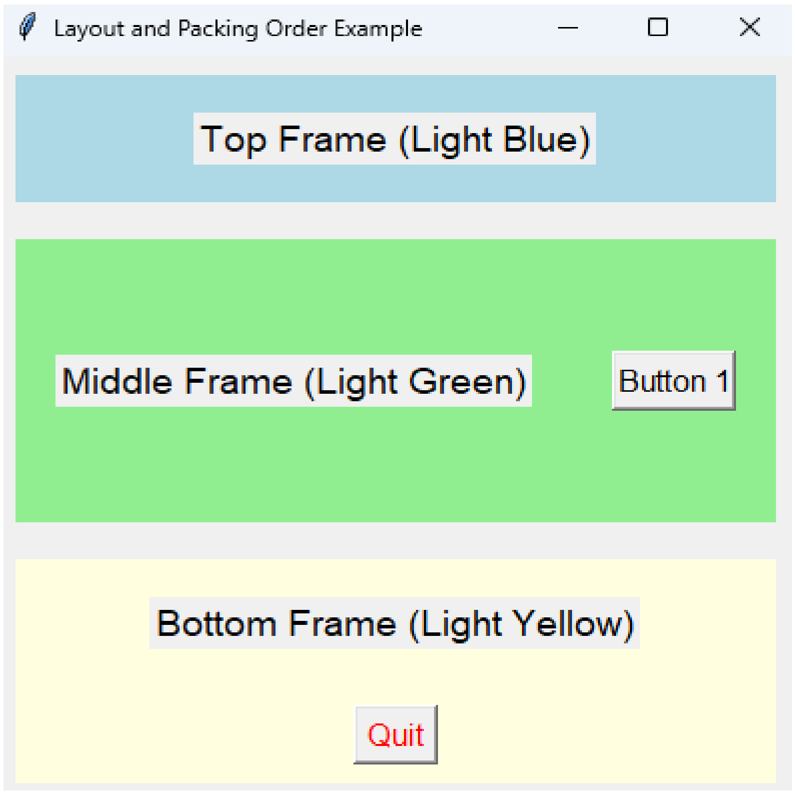


17. **Implement a program to Creating Layouts, Packing Order, Controlling Widget Appearances.**

```
import tkinter as tk

# Initialize the main window
root = tk.Tk()
root.title("Layout and Packing Order Example")
root.geometry("400x400")

# Step 1: Create Frames for Layout Management
# Frame 1 at the top (packing at the top)
frame1 = tk.Frame(root, bg="lightblue", height=100, width=400)
frame1.pack(fill="x", padx=10, pady=10)

# Frame 2 in the middle (packing at the left and right inside this frame)
frame2 = tk.Frame(root, bg="lightgreen", height=100, width=400)
frame2.pack(fill="both", expand=True, padx=10, pady=10)

# Frame 3 at the bottom (packing at the bottom)
frame3 = tk.Frame(root, bg="lightyellow", height=100, width=400)
frame3.pack(fill="x", padx=10, pady=10)

# Step 2: Add Widgets to Frame 1 (Top)
label1 = tk.Label(frame1, text="Top Frame (Light Blue)", font=("Arial", 14))
label1.pack(pady=20)

# Step 3: Add Widgets to Frame 2 (Middle)
label2 = tk.Label(frame2, text="Middle Frame (Light Green)", font=("Arial", 14))
label2.pack(side="left", padx=20, pady=20)

button1 = tk.Button(frame2, text="Button 1", font=("Arial", 12))
button1.pack(side="left", padx=20)

button2 = tk.Button(frame2, text="Button 2", font=("Arial", 12))
button2.pack(side="right", padx=20)

# Step 4: Add Widgets to Frame 3 (Bottom)
label3 = tk.Label(frame3, text="Bottom Frame (Light Yellow)", font=("Arial", 14))
label3.pack(pady=20)
```

```
quit_button = tk.Button(frame3, text="Quit", font=("Arial", 12), command=root.quit,
fg="red")
quit_button.pack(pady=10)

# Start the GUI event loop
root.mainloop()
```

Output:


Layout and Packing Order Example
Top Frame (Light Blue)
Middle Frame (Light Green)
Button 1
Bottom Frame (Light Yellow)
Quit

18. Implement a program based on Text Processing, Searching for Files.

```
import os
import fnmatch

def search_files(directory, pattern):
    matches = []
    # Walk through the directory
    for root, dirs, files in os.walk(directory):
        for filename in fnmatch.filter(files, pattern):
            matches.append(os.path.join(root, filename))
    return matches

if __name__ == "__main__":
    # Specify the directory and file pattern
    search_directory = input("Enter the directory to search in: ")
    file_pattern = input("Enter the file pattern (e.g., *.txt): ")

    # Search for files
    found_files = search_files(search_directory, file_pattern)

    # Display results
    if found_files:
        print("Found files:")
        for file in found_files:
            print(file)
    else:
        print("No files found matching the pattern.")
```

Output:

```
Enter the directory to search in: demo1
Enter the file pattern (e.g., *.txt): ABC.txt
Found files:
demo1\ABC.txt
```

19. Implement a program based on processing XML document, using XML Libraries

1. First create the xml file with .xml extension / Save this file as sample.xml

```
<?xml version="1.0"?>
<company>
  <employee>
    <name>John Doe</name>
    <department>HR</department>
    <salary>50000</salary>
  </employee>
  <employee>
    <name>Jane Smith</name>
    <department>IT</department>
    <salary>75000</salary>
  </employee>
  <employee>
    <name>Mike Johnson</name>
    <department>Finance</department>
    <salary>62000</salary>
  </employee>
</company>
```

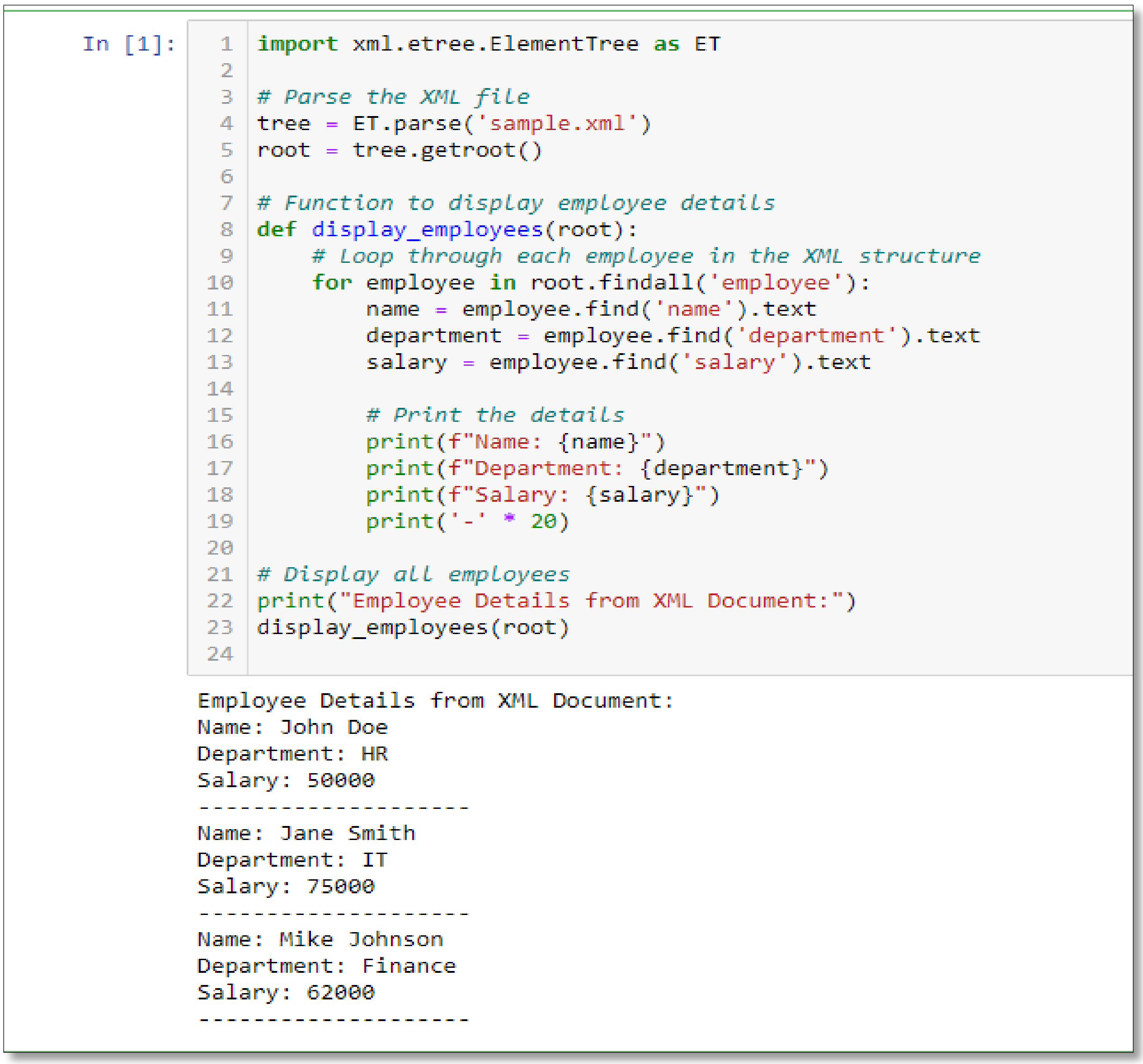
2. In python write this code

```
import xml.etree.ElementTree as ET

# Parse the XML file
tree = ET.parse('sample.xml')
root = tree.getroot()

# Function to display employee details
def display_employees(root):
  # Loop through each employee in the XML structure
  for employee in root.findall('employee'):
    name = employee.find('name').text
    department = employee.find('department').text
    salary = employee.find('salary').text

    # Print the details
    print(f"Name: {name}")
    print(f"Department: {department}")
    print(f"Salary: {salary}")
    print('-' * 20)
```

```
# Display all employees
print("Employee Details from XML Document:")
display_employees(root)
```

Output:

```
Employee Details from XML Document:
Name: John Doe
Department: HR
Salary: 50000
--------------------
Name: Jane Smith
Department: IT
Salary: 75000
--------------------
Name: Mike Johnson
Department: Finance
Salary: 62000
```



In [1]:
```python
import xml.etree.ElementTree as ET

# Parse the XML file
tree = ET.parse('sample.xml')
root = tree.getroot()

# Function to display employee details
def display_employees(root):
    # Loop through each employee in the XML structure
    for employee in root.findall('employee'):
        name = employee.find('name').text
        department = employee.find('department').text
        salary = employee.find('salary').text

        # Print the details
        print(f"Name: {name}")
        print(f"Department: {department}")
        print(f"Salary: {salary}")
        print('-' * 20)

# Display all employees
print("Employee Details from XML Document:")
display_employees(root)

```

```
Employee Details from XML Document:
Name: John Doe
Department: HR
Salary: 50000
--------------------
Name: Jane Smith
Department: IT
Salary: 75000
--------------------
Name: Mike Johnson
Department: Finance
Salary: 62000
--------------------
```

20. Implement a program based on HTML Parsing.

1. First, install `BeautifulSoup` if you haven't already:

```
pip install beautifulsoup4
```

2. Create html file with .html extension

EX**:** sample.html

```
<html>
<head>
   <title>Simple HTML Page</title>
</head>
<body>
   <h1>Main Heading</h1>
   <p>This is the first paragraph of the page.</p>
   <p>This is the second paragraph with <a href="#">a link</a>.</p>
</body>
</html>
```

3. Write Python code

```
from bs4 import BeautifulSoup

# Parse the HTML content with BeautifulSoup
soup = BeautifulSoup(html_content, 'html.parser')

# Extract and print the title
title = soup.title.string
print(f"Title: {title}")

# Extract and print the heading (h1)
heading = soup.h1.string
print(f"Heading: {heading}")

# Extract and print all paragraphs
paragraphs = soup.find_all('p')
for i, paragraph in enumerate(paragraphs, 1):
   print(f"Paragraph {i}: {paragraph.text}")
```

Output:

```
Title: Simple HTML Page
Heading: Main Heading
Paragraph 1: This is the first paragraph of the page.
Paragraph 2: This is the second paragraph with a link.
```

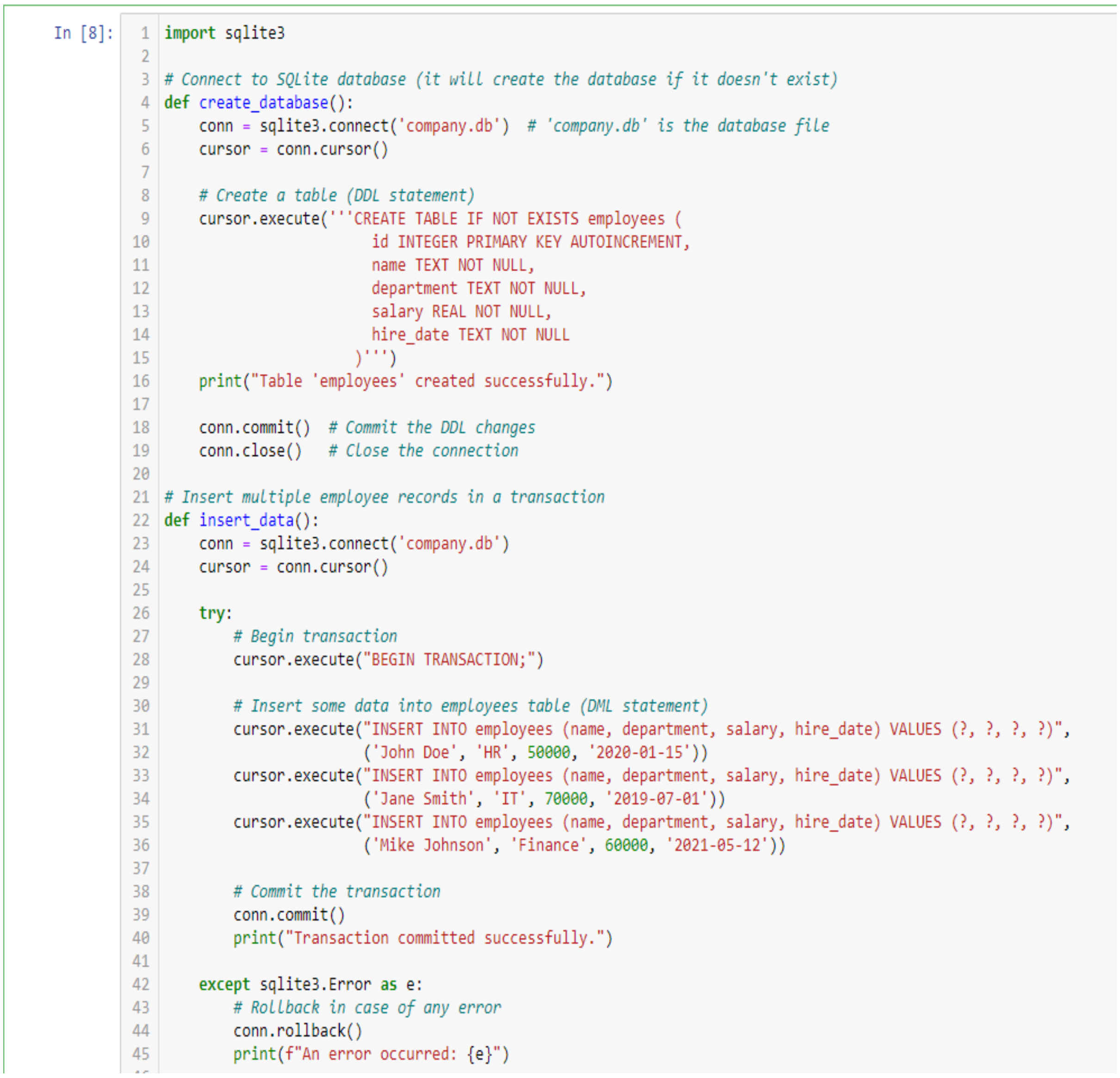
21. **Demonstrate a program using DBM - Creating and Accessing Persistent Dictionaries.**

```
import dbm

# Creating and storing key-value pairs in a DBM database
def create_db():
    with dbm.open('example_db', 'c') as db:  # 'c' mode means create if it doesn't exist
        # Add key-value pairs to the database
        db[b'name'] = b'John Doe'
        db[b'age'] = b'30'
        db[b'city'] = b'New York'
        db[b'occupation'] = b'Software Developer'
        print("Data has been written to the DBM database.")

# Accessing and retrieving key-value pairs from the DBM database
def access_db():
    with dbm.open('example_db', 'r') as db:  # 'r' mode means read-only
        # Access and print values from the database
        name = db.get(b'name', b'Unknown').decode('utf-8')
        age = db.get(b'age', b'Unknown').decode('utf-8')
        city = db.get(b'city', b'Unknown').decode('utf-8')
        occupation = db.get(b'occupation', b'Unknown').decode('utf-8')

        print(f"Name: {name}")
        print(f"Age: {age}")
        print(f"City: {city}")
        print(f"Occupation: {occupation}")

# Delete a key from the DBM database
def delete_from_db():
    with dbm.open('example_db', 'w') as db:  # 'w' mode means read/write (must
exist)
        del db[b'age']  # Delete the 'age' key
        print("Key 'age' has been deleted.")

# Run the functions
create_db()   # Create the DB and store data
access_db()   # Access and print the stored data

delete_from_db()  # Delete an entry
access_db()       # Access the data again to check deletion
```

Output:

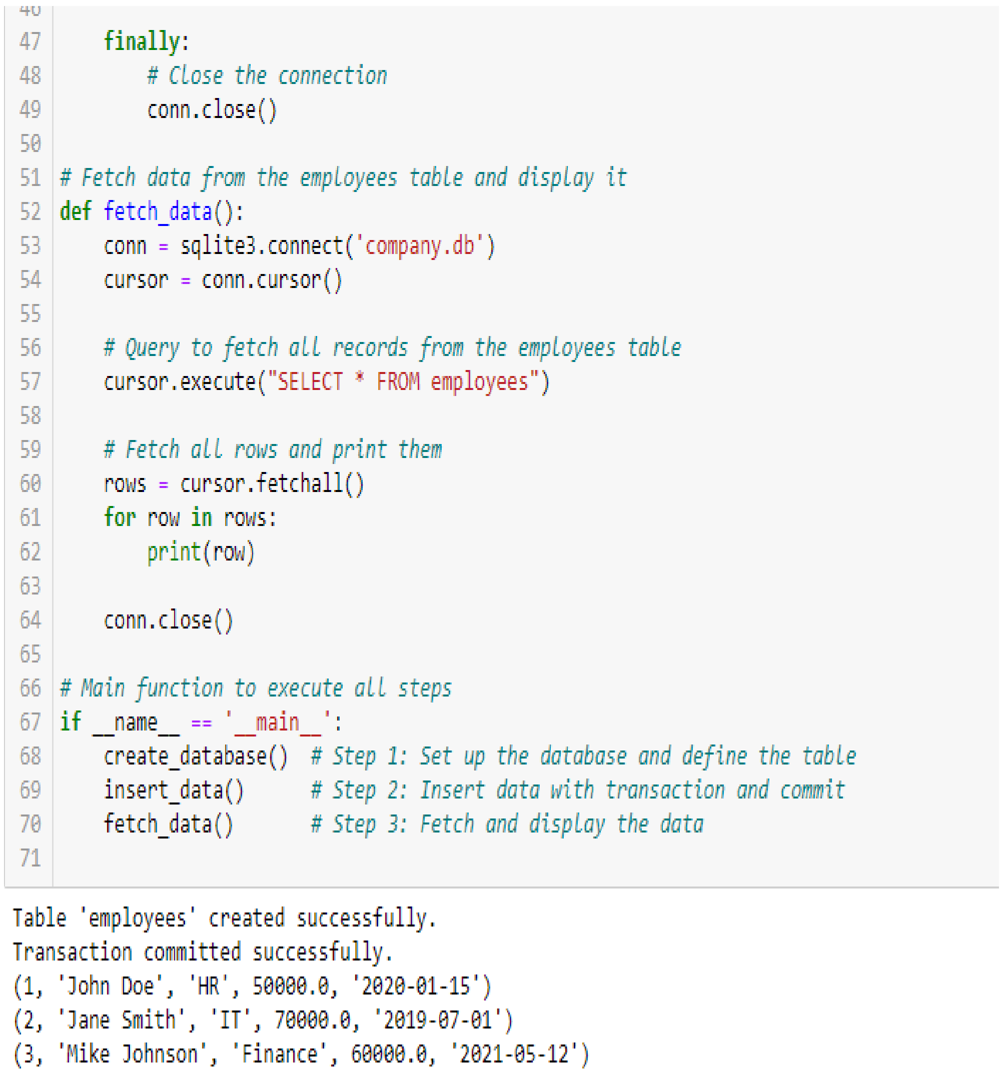
```
Data has been written to the DBM database.
Name: John Doe
Age: 30
City: New York
Occupation: Software Developer
Key 'age' has been deleted.
Name: John Doe
Age: Unknown
City: New York
Occupation: Software Developer
```

22. Demonstrate a program using Relational Database - Writing SQL Statements, Defining Tables, Setting Up a Database.(Transactions and Committing the Results.)



In [8]:

```python
import sqlite3

# Connect to SQLite database (it will create the database if it doesn't exist)
def create_database():
    conn = sqlite3.connect('company.db')  # 'company.db' is the database file
    cursor = conn.cursor()

    # Create a table (DDL statement)
    cursor.execute('''CREATE TABLE IF NOT EXISTS employees (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        name TEXT NOT NULL,
                        department TEXT NOT NULL,
                        salary REAL NOT NULL,
                        hire_date TEXT NOT NULL
                      )''')
    print("Table 'employees' created successfully.")

    conn.commit()  # Commit the DDL changes
    conn.close()   # Close the connection

# Insert multiple employee records in a transaction
def insert_data():
    conn = sqlite3.connect('company.db')
    cursor = conn.cursor()

    try:
        # Begin transaction
        cursor.execute("BEGIN TRANSACTION;")

        # Insert some data into employees table (DML statement)
        cursor.execute("INSERT INTO employees (name, department, salary, hire_date) VALUES (?, ?, ?, ?)",
                       ('John Doe', 'HR', 50000, '2020-01-15'))
        cursor.execute("INSERT INTO employees (name, department, salary, hire_date) VALUES (?, ?, ?, ?)",
                       ('Jane Smith', 'IT', 70000, '2019-07-01'))
        cursor.execute("INSERT INTO employees (name, department, salary, hire_date) VALUES (?, ?, ?, ?)",
                       ('Mike Johnson', 'Finance', 60000, '2021-05-12'))

        # Commit the transaction
        conn.commit()
        print("Transaction committed successfully.")

    except sqlite3.Error as e:
        # Rollback in case of any error
        conn.rollback()
        print(f"An error occurred: {e}")
```



```python
    finally:
        # Close the connection
        conn.close()

# Fetch data from the employees table and display it
def fetch_data():
    conn = sqlite3.connect('company.db')
    cursor = conn.cursor()

    # Query to fetch all records from the employees table
    cursor.execute("SELECT * FROM employees")

    # Fetch all rows and print them
    rows = cursor.fetchall()
    for row in rows:
        print(row)

    conn.close()

# Main function to execute all steps
if __name__ == '__main__':
    create_database()  # Step 1: Set up the database and define the table
    insert_data()      # Step 2: Insert data with transaction and commit
    fetch_data()       # Step 3: Fetch and display the data

```

```
Table 'employees' created successfully.
Transaction committed successfully.
(1, 'John Doe', 'HR', 50000.0, '2020-01-15')
(2, 'Jane Smith', 'IT', 70000.0, '2019-07-01')
(3, 'Mike Johnson', 'Finance', 60000.0, '2021-05-12')
```

```
import sqlite3

# Connect to SQLite database (it will create the database if it doesn't exist)
def create_database():
    conn = sqlite3.connect('company.db')  # 'company.db' is the database file
    cursor = conn.cursor()

    # Create a table (DDL statement)
    cursor.execute('''CREATE TABLE IF NOT EXISTS employees (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    department TEXT NOT NULL,
                    salary REAL NOT NULL,
                    hire_date TEXT NOT NULL
                   )''')
    print("Table 'employees' created successfully.")

    conn.commit()  # Commit the DDL changes
    conn.close()   # Close the connection

# Insert multiple employee records in a transaction
def insert_data():
    conn = sqlite3.connect('company.db')
    cursor = conn.cursor()

    try:
        # Begin transaction
        cursor.execute("BEGIN TRANSACTION;")

        # Insert some data into employees table (DML statement)
        cursor.execute("INSERT INTO employees (name, department, salary, hire_date)
VALUES (?, ?, ?, ?)",
                   ('John Doe', 'HR', 50000, '2020-01-15'))
        cursor.execute("INSERT INTO employees (name, department, salary, hire_date)
VALUES (?, ?, ?, ?)",
                   ('Jane Smith', 'IT', 70000, '2019-07-01'))
        cursor.execute("INSERT INTO employees (name, department, salary, hire_date)
VALUES (?, ?, ?, ?)",
                   ('Mike Johnson', 'Finance', 60000, '2021-05-12'))

        # Commit the transaction
        conn.commit()
        print("Transaction committed successfully.")
    except sqlite3.Error as e:
        # Rollback in case of any error
        conn.rollback()
        print(f"An error occurred: {e}")

    finally:
        # Close the connection
        conn.close()

# Fetch data from the employees table and display it
def fetch_data():
    conn = sqlite3.connect('company.db')
    cursor = conn.cursor()

    # Query to fetch all records from the employees table
    cursor.execute("SELECT * FROM employees")

    # Fetch all rows and print them
    rows = cursor.fetchall()
    for row in rows:
        print(row)

    conn.close()

# Main function to execute all steps
if __name__ == '__main__':
    create_database()  # Step 1: Set up the database and define the table
    insert_data()      # Step 2: Insert data with transaction and commit
    fetch_data()       # Step 3: Fetch and display the data
```

Output:

```
Table 'employees' created successfully.

Transaction committed successfully.

(1, 'John Doe', 'HR', 50000.0, '2020-01-15')

(2, 'Jane Smith', 'IT', 70000.0, '2019-07-01')

(3, 'Mike Johnson', 'Finance', 60000.0, '2021-05-12')
```