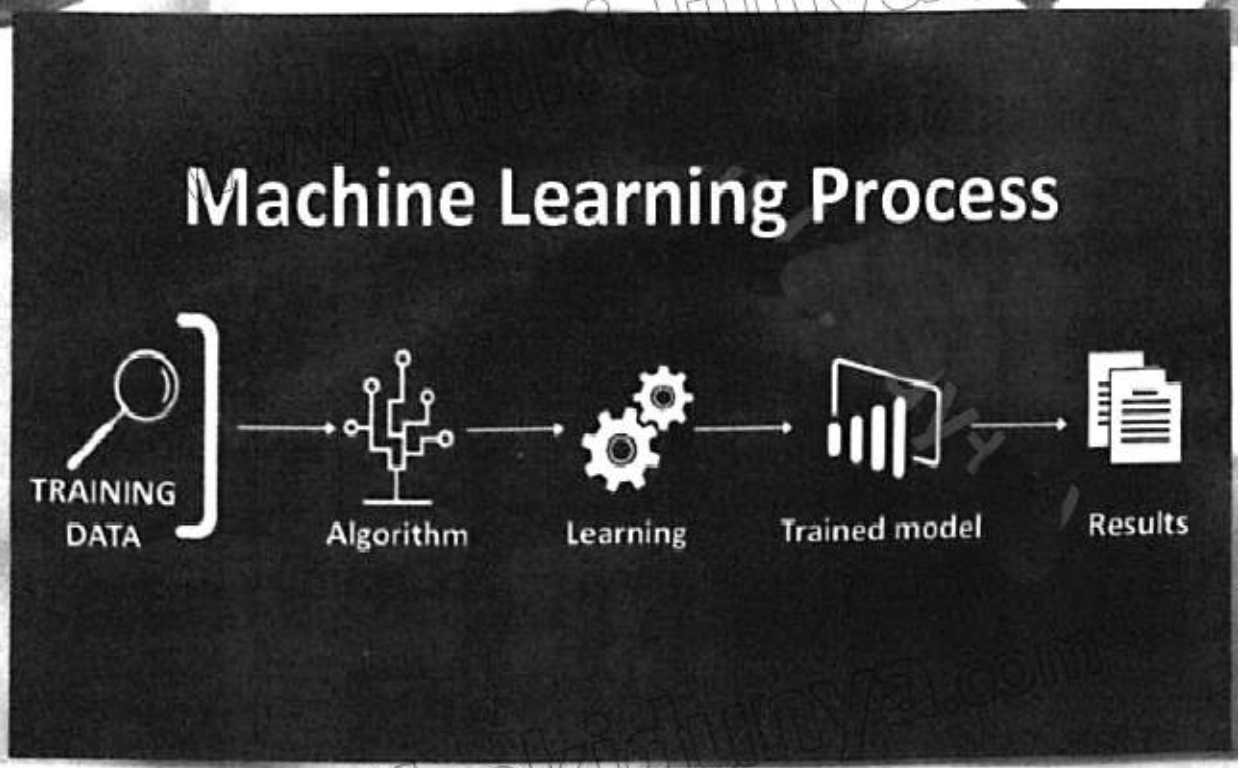
4 Data and Analysis

Learning Outcomes

At the end of this unit students will be able to:

- analyze data and identify key model performance metrics of real-world machine learning models.
- explain and create a data visualization using data visualization software (for example, Structured Query Language (SQL), Python, or R)
- learn how to form hypotheses and perform hypothesis testing. Students will learn to communicate findings using advanced data visuals and tie them back to hypotheses.



```
Hello, What's your name?Abrar
okay! Abrar I am Guessing a number between 1 and 100:
50
Your guess is too low
75
Your guess is too high
60
Your guess is too high
55
Your guess is too low
58
Yourguessed the number in 5 tries!
```

```
Hello, What's your name?Azam
okay! Azam I am Guessing a number between 1 and 100:
50
Your guess is too low
75
Your guess is too low
90
Your guess is too high
85
Your guess is too high
80
Your guess is too high
You did not guess the number, The number was 78
```

13. The following code is supposed to open a file 'abc.txt' and counts the number of occurrences of the character 's' and displays output like:

```
1   file_name = 'abc.txt'
2   target_letter = 'S'
3   try:
4       file = open(file_name, "r")
5       text = file.read()
6       file.close
7   except FileNotFoundError:
8       print(f"Error: File '{file_name}' not found.")
9       exit()
10  count = 0
11  for char in text:
12      if char = target_letter:
13          count += 1
14  print("The letter '{target_letter}' appears " + count + " times in the file '{file_name}'.")
```

- Sample text in file 'abc.txt': I scream, you scream, we all scream for ice cream.
- Sample Output: The letter 's' appears 3 times in the file 'abc.txt'.

However, the code does not execute. Debug and remove any bug(s) from the code.

Mini Project 1

Design a game where multiple objects are falling from the top. The objective is to "Catch The Falling Objects" using arrow keys. For every catch, a score increases to 1 point. The game ends whenever an object is missed.

Mini Project 2

Design two textboxes such that the first is for entering text while the second shows the text converted to uppercase. The user can click on the first textbox and type a sentence. As the user types, the second textbox automatically displays the uppercase version. The second textbox is not editable.

Give long answers to the following Extended Response Questions (ERQs).

1. Write a Python function that calculates the factorial of a given number. Test the function with 5 different inputs.
2. Implement a Python class 'Rectangle' with methods to calculate the area and perimeter of a rectangle. Create instances of the class and print the perimeter of rectangle.
3. Write a Python program highlighting the use of functions to simplify a problem. For example, create a program that calculates the average of a list of numbers and by using separate functions to add all numbers, , marking repeated values and calculate the average of the list with and without repeating values.
4. Create a Python dictionary to store student names as keys and their grades as values. Write a program to add new students, update existing students' grades, and remove students from the dictionary.
5. Write a Python program to create a file named data.txt, write a string to it and then read the content of the file and display the output.
6. Create a Python program that appends a new line of text to an existing file. If the file does not exist, it should create a new file.
7. Write Python code using Pygame that creates a simple window with a button. When the button is clicked, it should display a message on the window.
8. Develop a Python program that calculates and prints the transpose of a 3x3 matrix represented using a nested list.
9. Write a Python function that counts the number of occurrences of each letter in a text file and prints the results.
10. Write a Python program to connect to a database and demonstrate how to insert, update and retrieve data from the database. Explain the importance of database connectivity in Python applications.
11. Explain unit testing and its importance in Python programming. How does it help in catching errors early during the development process? Include an example of a unit test for a simple Python function.
12. Write a Python code that selects a number between 1 and 10 and the player will try to guess the number in 5 tries. For every incorrect guess, the program provides a hint whether the last guess was on higher or lower side. If the number is successfully guessed then a Congratulating message should be displayed along with in how many attempts the number was guessed correctly. Otherwise another message should be displayed along with the chosen number.

The outputs should look like:

14. The primary goal of debugging is _____.

a. Writing new code b. Fixing errors in the code

c. Increasing program speed d. Creating a graphical interface

15. Python function used to print debugging messages to the console _____.

a. print() b. debug() c. trace() d. log()

16. Output of the following code will be:

```
1  my_dict = {"a": [1, 2], "b": [3, 4]}
2  print(my_dict["a"][1])
```

a. 1 b. 2 c. [1, 2] d. [3, 4]

17. The code will print "Inner" _____ times.

```
1  for i in range(3):
2      for j in range(2):
3          print("Inner")
```

a. 2 b. 3 c. 5 d. 6

Give short answers to the following Short Response Questions (SRQs).

1. Elaborate programming language paradigm and why is it important for software development?
2. How do you interpret the concept of complexity reduction in programming? How does dividing code into modules help manage complexity?
3. Compare Functional Programming and Object-Oriented Programming (OOP) paradigms. Enlist main differences in their approach to problem-solving.
4. What is Disk I/O and why is it important for system performance?
5. Describe the difference modes in file handling in Python.
6. What is the benefit of using the 'with' keyword while working with files in Python?
7. How can you create a simple GUI application using Pygame? Enlist the steps involved.
8. Describe the use of a nested list in Python and how can it be used to represent a matrix?
9. What is data normalization and why is it used in database management?
10. What is debugging and why is it important in software development?
11. What seems to be the problem in:

```
1  Sample_dict = {"a": 1, "b": 2}
2  print(sample_dict["c"])
```

12. Final value of my_list after the following lines would be:

```
1  own_list = [1, 2, 3]:
2  own_list.append(4, 5)
```

Exercise

Select the best answer for the following Multiple-Choice Questions (MCQs).

1. Main purpose of the programming paradigm is _____.
   a. Increase code length
   b. Complexity reduction
   c. Eliminate bugs
   d. Improve hardware performance
2. _____ programming paradigm emphasizes the use of functions without changing state.
   a. Procedural b. Functional c. Object-oriented d. Structural
3. Object-Oriented Programming primarily revolves around _____ of the following.
   a. Data structures b. Functions c. Objects d. Variables
4. In Python _____ data structure is used to store a collection of items.
   a. List b. Dictionary c. Tuple d. Set
5. A dictionary stores _____ in Python.
   a. Keys only b. Values only c. Key-value pairs d. Sequential data
6. An example of disk I/O _____.
   a. Storing a variable in memory
   b. Reading a file from the hard disk
   c. Defining a function
   d. Looping through a list
7. To write content to a file in Python _____ method is used.
   a. file.write() b. file.insert() c. file.add() d. file.modify()
8. A primary key in a database is _____.
   a. A key that has a default value
   b. A unique identifier for records
   c. A key for linking multiple databases
   d. A key used for temporary tables
9. A common database normalization goal is _____.
   a. Reducing redundancy
   b. Maximizing data size
   c. Increasing the number of tables
   d. Storing duplicate records
10. _____ key is used to uniquely identify a record in a related table.
    a. Primary b. Secondary c. Foreign d. Unique
11. To create a simple GUI in Python, _____ library is used.
    a. Pygame b. OS c. Sys d. math
12. A nested list in Python is referred as _____.
    a. A list containing another list
    b. A list of dictionaries
    c. A dictionary of lists
    d. A set containing multiple lists
13. The purpose of unit testing in Python is _____.
    a. Test individual units of code
    b. Test the entire program
    c. Debug the entire program
    d. Optimize the code for speed

Summary

- **Programming Paradigm:** Refers to different styles and methodologies of programming used to solve problems, making code more structured and easier to manage.
- **Complexity Reduction:** Helps to break down complex problems into simpler parts resulting in easier to understand and solve.
- **Functional Programming:** programs are developed in the form of functions.
- **Object-Oriented Programming (OOP):** Revolves around the use of objects and classes to represent real-world entities.
- **List in Python:** A data structure that stores an ordered collection of items which can be accessed by their index. Lists are mutable i.e. their contents can be changed.
- **Dictionary in Python:** A key-value pair data structure used for mapping keys to values such that every key in a dictionary is unique and values can be accessed using their corresponding keys.
- **Files in Python:** File handling allows Python programs to read from and write to files and store data permanently. Common operations include opening, reading, writing and closing files.
- **Databases:** A structured collection of data used for efficient storage and retrieval. Tools like MySQL, SQLite, and PostgreSQL are commonly used for managing databases.
- **Data Normalization:** A database design technique aimed at reducing redundancy and dependency by organizing fields and tables of data.
- **Primary Key:** Is a unique identifier for records in a database table emphasizing each record can be distinctly retrieved or updated.
- **GUI in Python:** A graphical user interface that allows users to interact with applications visually, using elements like buttons, text fields and windows, etc. Python's pygame library is commonly used to create GUIs.
- **Nested Lists:** Lists within lists are nested lists that are used for storing and organizing complex data structures like matrices.
- **List as Value in Dictionary:** Python allows the use of lists as values in dictionaries such that multiple items can be stored under a single key.
- **Testing in Python:** Testing ensures the correctness of code.
- **Unit Test:** Unit testing focuses on testing small, isolated units of functionality, such as individual functions or methods.
- **Debugging:** The process of identifying and fixing bugs or errors in a program. Techniques include using print messages, where messages are printed to the console to help track the flow of a program, and more sophisticated tools like debuggers that allow stepping through the code line by line.

However, for the third breakpoint the result variable is flagged in the code and watchpoint shows the mismatched value as compared to the expected value to track the bug.

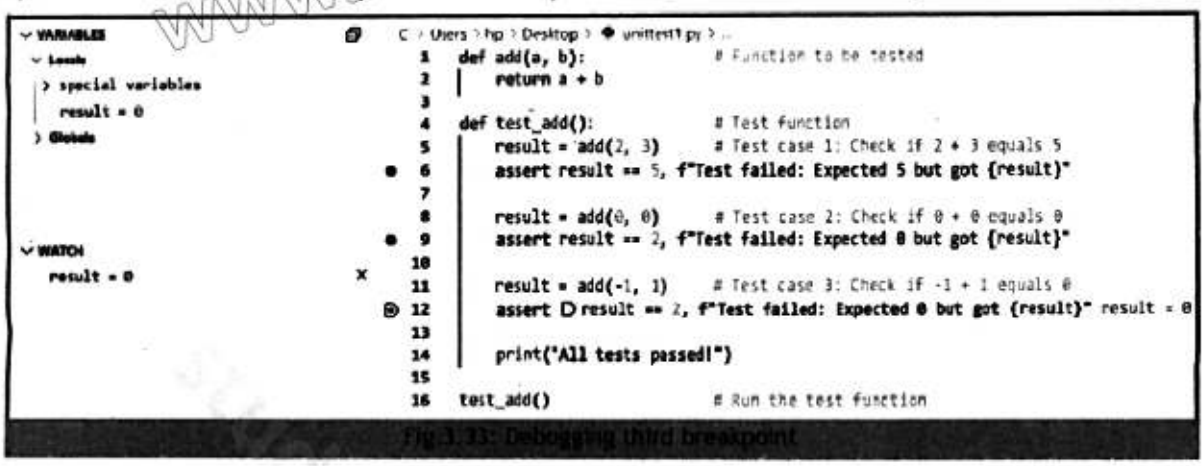


Fig 3.33: Debugging third breakpoint

The same code is changed for multiplication and tests are adjusted accordingly. Will all tests be passed? If not, which one will fail and why? Try debugging the code.

```
1   def multiply(a, b):
2       return a * b
3
4   def test_multiply():
5       result = multiply(2, 3)
6       assert result == 6, f"Test failed: Expected 6 but got {result}"
7
8       result = multiply(4, 0)
9       assert result == 1, f"Test failed: Expected 1 but got {result}"
10
11      result = multiply(-1, -1)
12      assert result == 1, f"Test failed: Expected 1 but got {result}"
13
14      print("All tests passed!")
15
16  test_multiply()
```

Fig 3.34: Function to multiply 2 numbers with tests

```
1   def add(a, b):                  # Function to be tested
2       return a + b
3
4   def test_add():                 # Test function
5       result = add(2, 3)          # Test case 1: Check if 2 + 3 equals 5
6       assert result == 5, f"Test failed: Expected 5 but got {result}"
7
8       result = add(0, 0)          # Test case 2: Check if 0 + 0 equals 0
9       assert result == 0, f"Test failed: Expected 0 but got {result}"
10
11      result = add(-1, 1)         # Test case 3: Check if -1 + 1 equals 0
12      assert result == 0, f"Test failed: Expected 0 but got {result}"
13
14      print("All tests passed!")
15
16  test_add()                      # Run the test function
```

Fig.3.30: Adding Breakpoints

Let's execute the code and debug it. We deployed 3 breakpoints on line 6, 9 and 12 to track the values to be checked. As watchpoint we added 'result' variable. As the debugging starts and it stops on first breakpoint, the result is as expected.

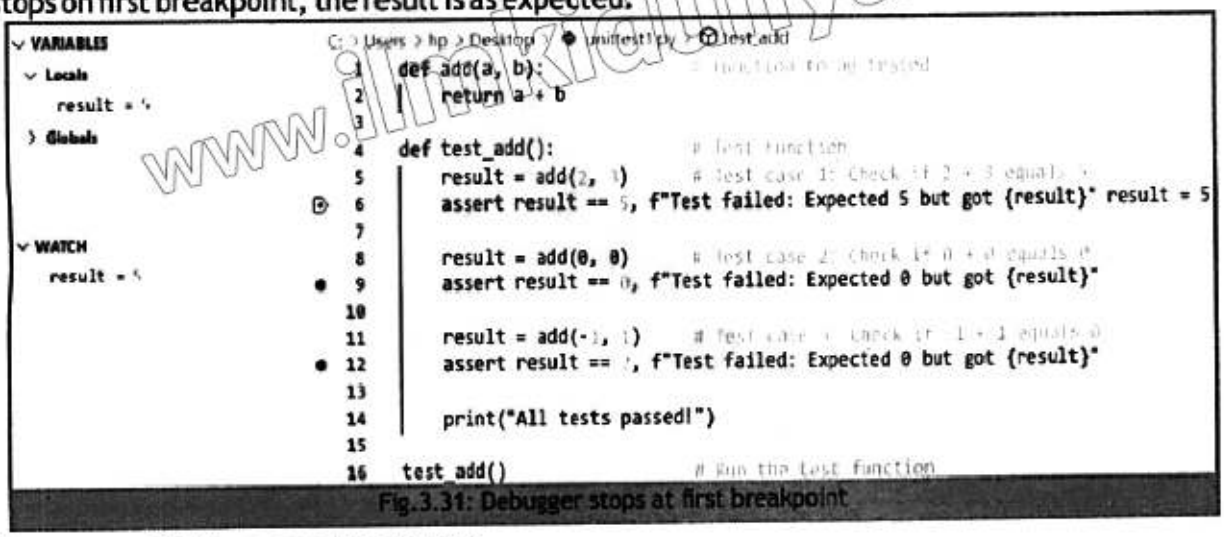


Fig.3.31: Debugger stops at first breakpoint

Same goes with the second breakpoint.

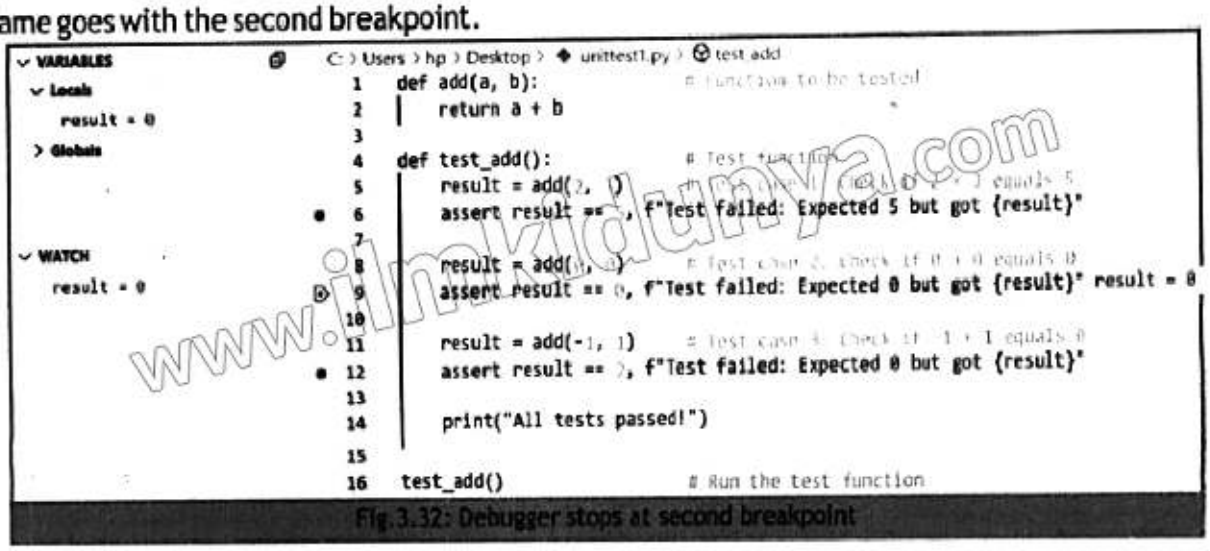


Fig.3.32: Debugger stops at second breakpoint

3.3.2 Python Unit test

Python's built-in unittest framework is a tool that helps you check if your code works in the desired way. Rather than manually testing your code by running it, unittest lets you write small tests that check specific parts of your program. You may check it for some values to check whether it is returning the desired and correct result. But as the lines of code increase the code may become compilcated to test it manually. unittest is quite useful in such scenarios to write a set of tests to ensure the function is executing correctly for various inputs.

So, for a written function that performs some operation, you would write another function (a test function) to check if the first function gives the correct result. Use assertions to compare the actual output with the expected output. For example, you have a function that adds two numbers, like:

```
1   def add(a, b):                  # Function to be tested
2       return a + b
3
4   def test_add():                 # Test function
5       result = add(2, 3)          # Test case 1: Check if 2 + 3 equals 5
6       assert result == 5, f"Test failed: Expected 5 but got {result}"
7
8       result = add(0, 0)          # Test case 2: Check if 0 + 0 equals 0
9       assert result == 0, f"Test failed: Expected 0 but got {result}"
10
11      result = add(-1, 1)         # Test case 3: Check if -1 + 1 equals 0
12      assert result == 0, f"Test failed: Expected 0 but got {result}"
13
14      print("All tests passed!")
15
16  test_add()                      # Run the test function
```

Fig.3.29: Function to add 2 numbers with tests

On line 1 the add() function adds two numbers. The test_add() function on line 4 tests the add() function with different inputs, like as input parameters two positive numbers are provided on line 5, two null values on line 8 and a positive and negative number on line 11. The assert statement checks if the output of add() matches the expected value. If not, it raises an error with a custom message. If all assertions pass, it prints "All tests passed!"

To have a better understanding you may check if for different values and results. As a next step, we assume that expected result was 2 on line 12 and we oversaw the negative value on line 11. So, definitely we would get the 'Test failed: ...' message. To investigate it we can debug the code by applying breakpoints in the code:

This code performs the transpose of a 3x3 matrix (swapping rows with columns). Here's a simple explanation:

- Line 2-3: Initializes a matrix and an empty list transpose to store the result of the transposed matrix.
- Line 5-8: The outer loop goes through each column in the original matrix. Inside this loop, a new row (new_row) is created.
- Line 9-12: The inner loop iterates through each row of the matrix, appending the corresponding column value to new_row.
- Line 14-15: Once the inner loop finishes, the new_row (a transposed row) is added to transpose.
- Line 16-17: Finally, the code prints the transposed matrix by printing each row in transpose.

Advanced debugging involves more than basic print commands by using techniques like breakpoints and watchpoints to identify and fix bugs and errors. Breakpoints are placed on the line of code while watchpoints are set on variables and expressions or formulae. Multiple breakpoints and watchpoints can be placed in a program. These methods are especially useful for handling complex issues.

The same code of Fig.3.27 can be placed in a debug mode in an IDE like Visual Studio.Net or an online IDE like REPLIT which provide a better GUI interface like selecting the line and setting the breakpoint on the said line. When run in debug mode step by step, the debugger not only stops at the breakpoint but also provides the values of the different variables and data structures. This helps in analyzing the data and overall behavior of the program. Fig. 3.28 shows the code while the debugger is on, reflecting the different variable values and the corresponding breakpoint set at line 6.

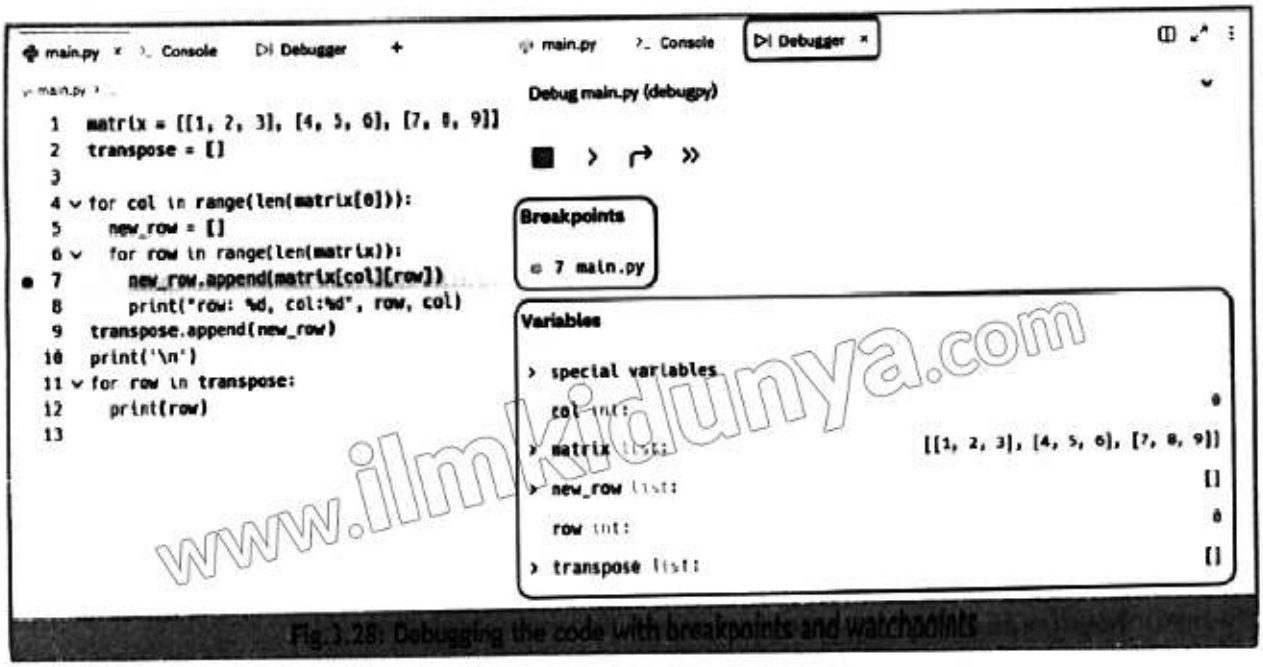


Fig.3.28: Debugging the code with breakpoints and watchpoints

3.3 Techniques for Testing & Debugging

3.3.1. Debugging

Debugging is the process to analyze the code for bugs and errors and help in locating them, such that when fixed error-free program can run smoothly and deliver desired results. The simplest of debugging involves adding multiple print statements in a program to get the idea whether the program is behaving as expected or not. This does not mean that we should add print statements on each line or every second line. The idea is that wherever some formulae, expression or a critical value update is in the code, using print statements it should be validated that particular variable, line or block of code is behaving as desired.

To make the point clearer, let us refer the matrix transpose example as shown in Fig. 3.27. Here on line 8, we have intentionally placed a bug, by switching the positions of the matrix indices. Once we run it the output shows that matrix and transpose matrix are same. So, within the nested loop, we have placed multiple print statements which we not only label the print statement for identification purposes, but also print the values of different variables and data structures. For example, 'Print-1' on line 6 ensures that the 'new_row' gets always initialized whenever the outer loop index updates. Similarly, 'Print-3' on line 10 prints the values which will be appended in the transpose matrix after completion of the inner loop. On line 9, 'Print-2' is placed inside the nested for-loop to print the values of matrix indices and points out that the values of matrix are as it is assigned to transpose and needs to be changed.

```
1  matrix = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]
2  transpose = []
3
4  for col in range(len(matrix[0])):
5      new_row = []
6      print('Print1-Outer Loop: ',new_row)
7      for row in range(len(matrix)):
8          new_row.append(matrix[col][row])
9          print('Print2-Inner Loop: row: ',row,'- col: ', col)
10     print('Print3-Outer Loop: ',new_row)
11     transpose.append(new_row)
12 for row in transpose:
13     print(row)
```

Output

```
Print1-Outer Loop:  []
Print2-Inner Loop: row:  0 - col:  0
Print2-Inner Loop: row:  1 - col:  0
Print2-Inner Loop: row:  2 - col:  0
Print3-Outer Loop:  [1, 2, 3]
Print1-Outer Loop:  []
Print2-Inner Loop: row:  0 - col:  1
Print2-Inner Loop: row:  1 - col:  1
Print2-Inner Loop: row:  2 - col:  1
Print3-Outer Loop:  [4, 5, 6]
Print1-Outer Loop:  []
Print2-Inner Loop: row:  0 - col:  2
Print2-Inner Loop: row:  1 - col:  2
Print2-Inner Loop: row:  2 - col:  2
Print3-Outer Loop:  [7, 8, 9]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

Fig. 3.27: Debugging with Print Statements

6. Update Record

For updating a record or a particular column, 'update' command is used. A good practice is to use select statement with where clause so that the desired record is retrieved. Thereafter for the same condition use the update command.

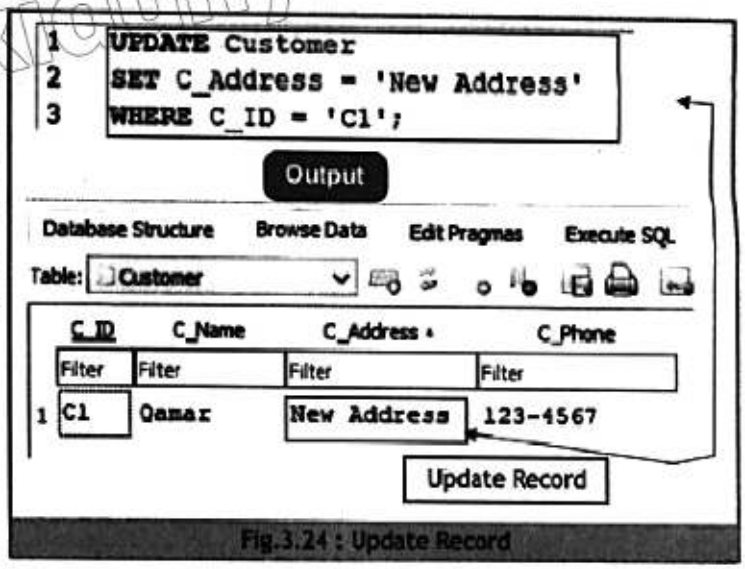


Fig.3.24 : Update Record

7. Use of Order By Clause

There are scenarios where records should be displayed in a sorted order. For that reason 'order by' clause is used in the query and the result will be displayed in ascending order, by default. An additional 'DESC' keywords is used with 'order by' clause to display the results in descending order.

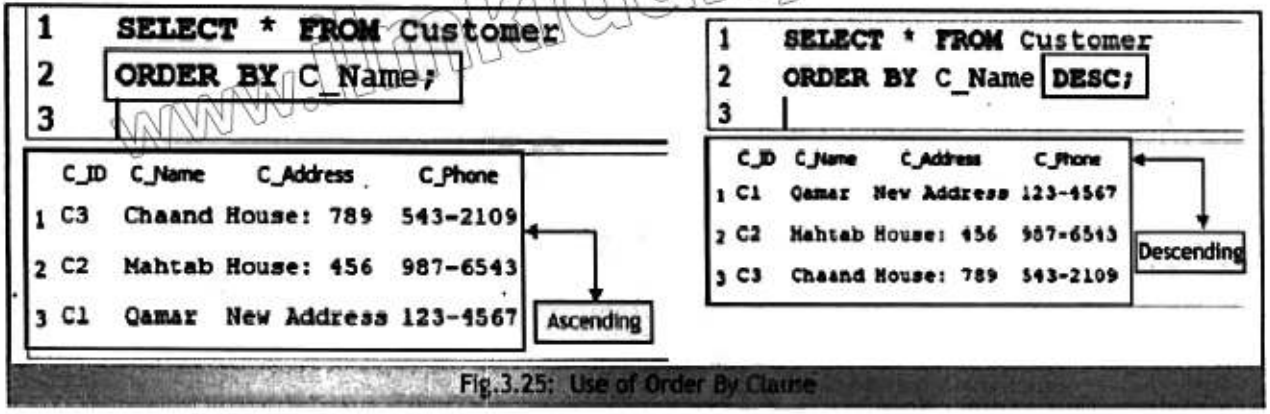


Fig.3.25: Use of Order By Clause

8. Deletion of Record(s)

You may delete a single or multiple entries from a table using 'Delete' command. However the table structure will remain in the database. It works just like select statement with an '*' all records will be deleted while using 'where' clause only the selected record(s) will be deleted.

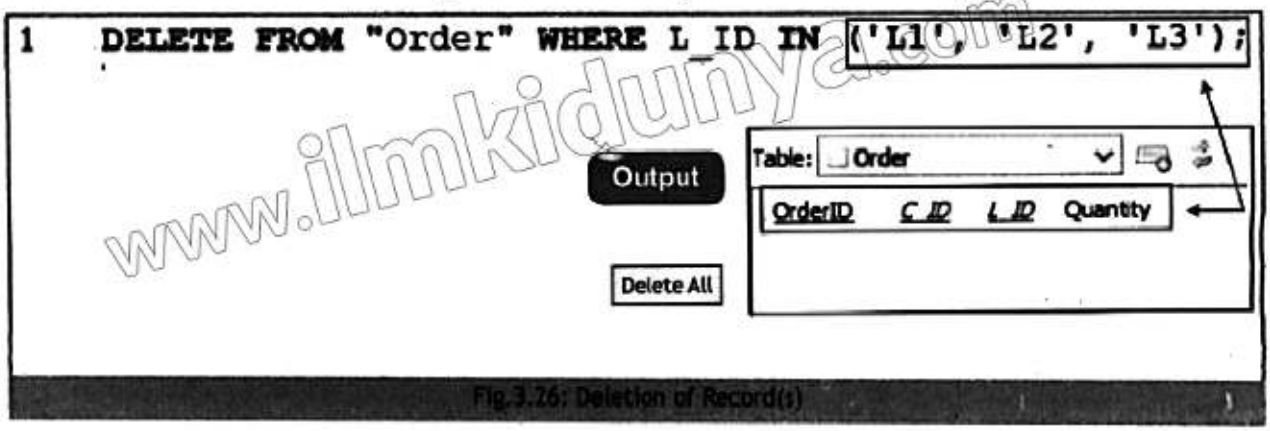


Fig.3.26: Deletion of Record(s)

Rather than adding single record, we can add multiple records as well using a single 'Insert Into' command as shown in Fig 3.21.

```
1   INSERT INTO Laptop (L_ID, L_Name, L_Price, L_Brand)
2   VALUES
3       ('L2', 'MacBook Air', 90000, 'Apple'),
4       ('L3', 'HP Spectre x360', 120000, 'HP');
5
```

Output

Database Structure | Browse Data | Edit Pragmas

Table: Laptop

	L_ID	L_Name	L_Price	L_Brand
	Filter	Filter	Filter	Filter
1	L1	Dell XPS 13	100000	Dell
2	L2	MacBook Air	90000	Apple
3	L3	HP Spectre x360	120000	HP

Multiple Record Added

Fig.3.21 :Adding multiple records

In this fashion, we can create additional tables and populate the data.

5. Querying Records from Table using Select

'Select' is the mostly used query to select records from the table. An '*' means all records in that particular table. Alternatively, we can mention the column names and only the specified columns will be displayed for all records.

Though select query fetches all records from the table but we can apply condition using where clause to get matching records. The matching can be a single or multiple records satisfying the condition.

```
1   SELECT * FROM Customer;
2
```

Output

	C_ID	C_Name	C_Address	C_Phone
1	C1	Qamar	New Address	123-4567
2	C2	Mahtab	House: 456	987-6543
3	C3	Chaand	House: 789	543-2109

Select All Records

Fig.3.22 : Querying Records from Table using Select

```
1   SELECT * FROM Customer
2   WHERE C_ID = 'C2';
3
```

	C_ID	C_Name	C_Address	C_Phone
1	C2	Mahtab	House: 456	987-6543

Record with ID 'C2'

Output

Fig.3.23 : Select query fetches all records from the table

Activity 7

Populate data for tables 'Customer', 'Order' and 'OrdreDetails'.

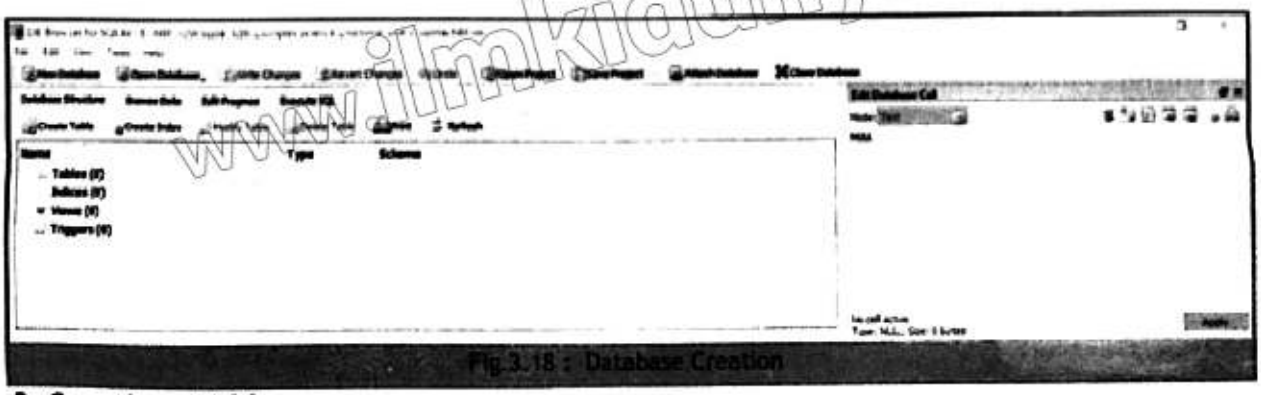


Fig 3.18 : Database Creation

3. Creating a Table

As a next step, use the 'CREATE TABLE' command to construct a table in the database. This command takes in the table's name, column names and data types. For example, we can create a table called 'Laptop' with four columns: 'L_ID' (an integer primary key), 'L_Name' (text), 'L_Price' (integer) and 'L_Brand' (text). So far only the schema is created and it does not contain any record.

```
1  CREATE TABLE Laptop (
2      L_ID TEXT PRIMARY KEY,
3      L_Name TEXT NOT NULL,
4      L_Price INTEGER NOT NULL,
5      L_Brand TEXT NOT NULL
6  );
```

Output

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Laptop

L_ID | L_Name | L_Price | L_Brand

Table Created

Fig 3.19 : Creating a Table

4. Adding Record(s)

Once the table is created we can use the 'INSERT INTO' command to enter a single row in the table. This command requires the table name, column names and values for the record we wish to add in the table. Be careful to use the values according to the schema, sequence matters.

```
1  INSERT INTO Laptop (L_ID, L_Name, L_Price, L_Brand)
2  VALUES ('L1', 'Dell XPS 13', 100000, 'Dell');
```

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Laptop

	L_ID	L_Name	L_Price	L_Brand
	Filter	Filter	Filter	Filter
1	L1	Dell XPS 13	100000	Dell

Single Record Added

Fig 3.20 : Adding single record

3.2.10 Create a Database

1. Installation of SQLite

To download and install SQLite on your Windows system, first go to the official SQLite website (https://sqlitebrowser.org/dl/). Look for a "Download" button and select the right download for your system. SQLite normally comes with already compiled files. Select the version that matches your Windows operating system and click the download link. The browser will prompt you to save the file. Choose a suitable location on your computer, such as the 'Downloads' folder. After the download is finished, you will notice that the file is a compressed (e.g., a ZIP) file. To access the SQLite files, right-click the downloaded file and select "Extract All". This will create a new folder containing the SQLite files which are ready to be used.

To check whether the SQLite was successfully installed, launch the 'Command Prompt' by typing "cmd" into the Windows search bar and click it. After opening the Command Prompt, use the 'cd' command to navigate to the folder where you extracted the SQLite files (for example, 'cd C:\Downloads\sqlite\'). Next type'sqlite3' and click Enter. If the installation was successful, SQLite's command prompt will appear showing that it is ready for use.

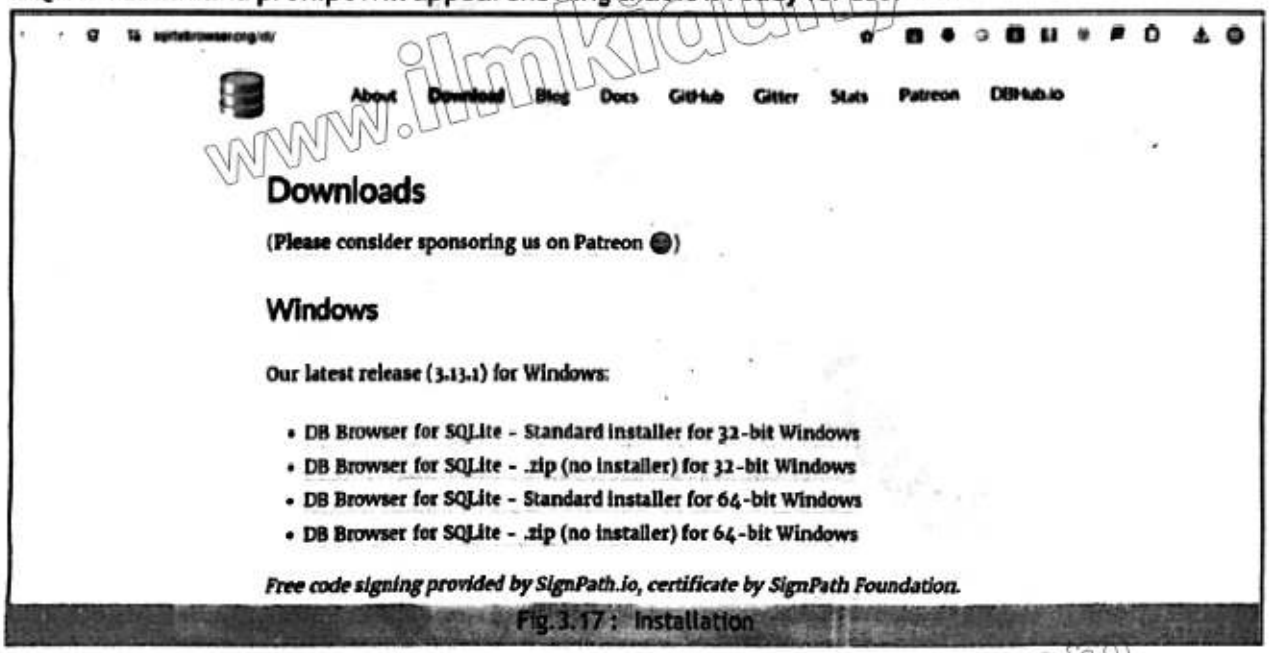


Fig. 3.17 : Installation

2. Database Creation

To create a new SQLite database, launch the SQLite Command-Line Interface (CLI). To access the interactive SQLite shell:

- Open your terminal or command prompt and type'sqlite3' and press Enter.
- Once inside the SQLite shell, use the command '.open NBF.db' to create a new database namely 'NBF'.

This command creates a new file, 'NBF.db' in the current directory to store your database. And we have successfully created a new SQLite database.

Step 5:

Lastly, to normalize to 3NF we ought to identify transitive dependencies, i.e.:

- L_ID in the Order table depends on L_Name, L_Price, and L_Brand, which are attributes of the Laptop table.

To remove transitive dependencies:

- The Order table now only references the L_ID.

Thus, the 3NF Tables are:

OrderDetails

OrderID	OrderDate
1	20-11-2024
2	21-11-2024
3	22-11-2024
4	23-11-2024
5	24-11-2024

Customer

OrderID	C_ID	L_ID	Quantity
1	C1	L1	1
2	C2	L2	2
3	C1	L3	1
4	C2	L1	1
5	C3	L2	2

Order

C_ID	C_Name	C_Address	C_Phone
C1	Qamar	House:123	123-4567
C2	Mahtab	House:456	987-6543
C3	Chaand	House:789	543-2109

Laptop

L_ID	L_Name	L_Price	L_Brand
L1	Dell XPS 13	100000	Dell
L2	MacBook Air	90000	Apple
L3	HP Spectre x360	120000	HP

Database Tools

Various database tools are available for data management, with Microsoft Access and SQLite being two popular options. MS Access, part of the MS Office suite, is simple to install, easy to use, and offers a user-friendly drag-and-drop interface. It is commonly used by individuals and small organizations for tasks like inventory, project management, and event planning. MS Access also integrates well with other MS Office programs like Excel and Project. SQLite is a lightweight database system that stores data in a single file. It is ideal for applications needing local storage and is widely used in mobile apps and scenarios that do not require a large database server. SQLite is simple, efficient, and easy to use.

Advantages of SQLite:

- SQLite occupies small space making it suitable for applications with limited resources.
- It is easily integrated into applications, hence becomes easier for use and deployment.
- SQLite has a serverless architecture which eliminates unnecessary dependencies.
- The complete database is saved in a single file.

Customer

C_ID	C_Name	C_Address	C_Phone
C1	Qamar	House:123	123-4567
C2	Mahtab	House: 456	987-6543
C3	Chaand	House: 789	543-2109

Laptop

L_ID	C_Name	L_Price	L_Brand
L1	Dell XPS 13	100000	Dell
L2	MacBook Air	90000	Apple
L3	HP Spectre x360	120000	HP

Order

OrderID	OrderDate	C_ID	L_ID	Quantity
1	20-11-2024	C1	L1	1
2	21-11-2024	C2	L2	2
3	22-11-2024	C1	L3	1
4	23-11-2024	C2	L1	1
5	24-11-2024	C3	L2	2

Step 4:

To normalize to 2NF we need to detect partial dependencies, which come out to be:

OrderID depends on OrderDate, but not on C_ID or L_ID.

As a result, we need to create a new table:

OrderDetails (OrderID, OrderDate)

As a result, 2NF Tables are:

OrderDetails

OrderID	OrderDate
1	20-11-2024
2	21-11-2024
3	22-11-2024
4	23-11-2024
5	24-11-2024

Order

OrderID	C_ID	L_ID	Quantity
1	C1	L1	1
2	C2	L2	2
3	C1	L3	1
4	C2	L1	1
5	C3	L2	2

Normalization steps for a database are:

- **First Normal Form (1NF):**

In 1NF tables have atomic values along with repeating groupings / multi-valued cells are removed.

- **Second Normal Form (2NF):**

2NF ensures that every non-key attribute is dependent on the whole Primary key.

- **Third Normal Form (3NF):**

In 3NF transitive dependencies, in which one non-key attribute relies upon another non-key attribute, are eliminated. For example, while developing a small database for a laptop shop the initial information may look like:

For a laptop shop if we want to create a database, the steps will be as follows:

Step 1:

Classify entities along with the attributes, which are:

- Customer: C_ID, C_Name, C_Address, C_Phone
- Laptop: L_ID, L_Name, L_Price, L_Brand
- Order: OrderID, OrderDate, C_ID, L_ID, Quantity

Step 2:

Let us initially create a single table to represent all the data:

OrderID	OrderDate	C_ID	C_Name	C_Address	C_Phone	L_ID	L_Name	L_Price	L_Brand	Quantity
1	20-11-2024	C1	Qamar	House:123	123-4567	L1	Dell XPS 13	100000	Dell	1
2	21-11-2024	C2	Mahtab	House: 456	987-6543	L2	MacBook Air	90000	Apple	2
3	22-11-2024	C3	Qamar	House:123	123-4567	L3	HP Spectre x360	120000	HP	1
4	23-11-2024	C4	Mahtab	House: 456	987-6543	L4	Dell XPS 13	100000	Dell	1
5	24-11-2024	C5	Chaand	House: 789	543-2109	L5	MacBook Air	90000	Apple	2

Step 3:

To normalize to 1NF we need to find repeating groups, which are:

Customer information that is repeated for each order.

Laptop information that is repeated for each order.

So, let us create separate tables as follows:

- Customer (C_ID, C_Name, C_Address, C_Phone)
- Laptop (L_ID, L_Name, L_Price, L_Brand)
- Order (OrderID, OrderDate, C_ID, L_ID, Quantity)

Hence, 1NF Tables are:

- **Field:**

A field is a piece of data within a record that defines the information placed in a certain column. For example, a Name field would store text data, whereas an Age field would store numerical data.

- **Data Integrity**

Data integrity refers to the accuracy and consistency of data within a database. It ensures that the data is accurate, dependable and appropriate for usage.

Primary & Secondary Keys

Keys are essential in creating database to uniquely identify records and create links among tables. The most common types of keys used in normalization are:

- **Primary Key:**

A primary key is a minimal set-of-attributes that is exclusively able to identify a record. It can be a single attribute or a combination of attributes.

- **Candidate Key:**

A candidate key is a single or a set-of-attributes that allow identifying of a record uniquely. Though there may be several candidate keys, only one is chosen to be the primary key. The rest are called secondary keys.

- **Foreign Key:**

A foreign key is a field in one table that creates a relationship with another table by referring to its primary key.

- **Composite Key:**

A composite key is a combination of two or more attributes that distinctively identify a row in a database table.

Data Normalization

A key component of relational database management is data normalization, which focuses on minimizing duplicate data. To ensure effective and consistent data storage and speed-up database access, the contents are divided into smaller interlinked tables. Data redundancy or duplicate storage of the same information is a common problem in databases which causes problems in maintaining consistency and increase chances of inaccuracies during updating of data. By dividing the data into more manageable and inter-linked tables, normalization solves these problems. Every table represents a separate entity with keys defining the relationships among the entities.

- Without normalization databases frequently show inconsistencies and abnormalities during updates or deletions. For example, a database containing customer information with several addresses would require changes in many locations for every address change, increasing the chances of mistakes. Normalized data addresses this by creating various tables for customers and addresses, which are connected by a key like customer-ID.

```
35
36    score_label = tk.Label(window, text="Score: 0")
37    score_label.pack()
38
39  ⑥ new_round()
40    window.mainloop()
```

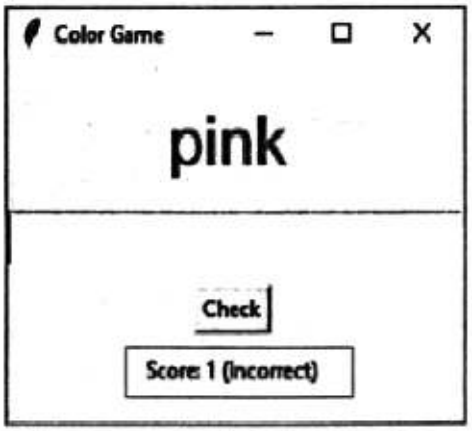


Fig.3.16 : tkinter

3.2.9 Databases

A set of data organized in a systematic manner, usually in a computer system, is called a database. A database stores, retrieves and handles large number of records for multiple users. Databases allow users to locate specific information quickly and accurately. The central source of data for every software program is a database. For instance, when a user logs in, shops online or makes a transaction, a database system saves the relevant data so that it may be retrieved later, i.e. what time the user logged in, which items the user purchased or the transaction to which account took place, etc.

Databases remove redundancies and inconsistencies that are possible in manual record keeping techniques by organizing data and ensuring its reliability. Databases also provide concurrent access and changes which allows several users to work on the same data-set at the same time. They also include security measures to safeguard sensitive information from unwanted access. Databases provide effective information retrieval and help in data analysis.

Types of Databases

- Most databases are relational. They organize data into rows and columns of a table such that each table represents an individual entity (for example customers or items).
- NoSQL databases provide flexibility and can manage vast amounts of unstructured or semi-structured data. Unlike relational databases, they do not have a fixed table structure.

Key Concepts involved in databases are:

- **Table:**

A table is the fundamental structure of a relational database and is made up of rows and columns. Each row denotes a record, while each column represents a field.

- **Record:**

A record is a single table item with information about a specific entity. For example a client record could include information like his or her name, address and telephone number.

```
1  import tkinter as tk
2  import random
3
4  window = tk.Tk()
5  window.title("Color Game")
6  score = 0
7  colors = ["red", "blue", "green", "yellow", "pink", "black",
   "orange", "purple", "brown"]
8
9  def new_round():
10     global score
11     global color_to_guess, random_color
12     color_to_guess = random.choice(colors)
13     random_color = random.choice(colors)
14     label.config(text=color_to_guess, fg=random_color)
15     entry.delete(0, tk.END)
16
17 def check_answer():
18     global score
19     user_input = entry.get().lower()
20     if user_input == random_color:
21         score += 1
22         score_label.config(text="Score: " + str(score))
23     else:
24         score_label.config(text="Score: " + str(score) + "
           (Incorrect)")
25     new_round()
26
27 label = tk.Label(window, font=("Arial", 24))
28 label.pack(pady=20)
29
30 entry = tk.Entry(window, font=("Arial", 18))
31 entry.pack()
32
33 check_button = tk.Button(window, text="Check",
   command=check_answer)
34 check_button.pack(pady=10)
```

① ② ③ ④ ⑤

- Align the label with the 'color_to_guess' text and 'random_color'.
- Clear player's input from the entry field, if any.

- For 'check_answer()' function:
  - Acquire the user_input from the entry field and check if it matches the random_color.
  - If correct, increment the score and update the score_label.
  - If incorrect, display a message showing incorrect answer.
  - Call 'new_round()' to start the next round.

4. GUI

- Create the main window:

```
1. window = tk.Tk()
2. window.title("Color Game")
```

- GUI elements that we need to place:

a) Add label 'Display the word to be guessed'.

```
27    label = tk.Label(window, font=("Arial", 24))
28    label.pack(pady=20)
```

b) Input field for entering answer.

```
30    entry = tk.Entry(window, font=("Arial", 18))
31    entry.pack()
```

c) Add a button to check player's answer when pressed.

```
33    check_button = tk.Button(window, text="Check", command=check_answer)
34    check_button.pack(pady=10)
```

d) Displays the player's current score.

```
36    score_label = tk.Label(window, text="Score: 0")
37    score_label.pack()
```

5. Starting the game

- Initially 'new_round()' function will be called to start the first game round.
- Call 'window.mainloop()' to run the game.

The complete code is shown in Fig 3.16.

```
21
22    entry2 = tk.Entry(root, width=10)
23    entry2.grid(row=0, column=1, padx=5, pady=5)
24
25    add_button = tk.Button(root, text="Add", command=add_numbers)
26    add_button.grid(row=1, column=0, padx=5, pady=5)
27
28    clear_button = tk.Button(root, text="Clear", command=clear_fields)
29    clear_button.grid(row=1, column=1, padx=5, pady=5)
30
31    result_label = tk.Label(root, text="Result will be displayed here.")
32    result_label.grid(row=2, column=0, columnspan=2, padx=5, pady=5)
33
34    root.mainloop()
```

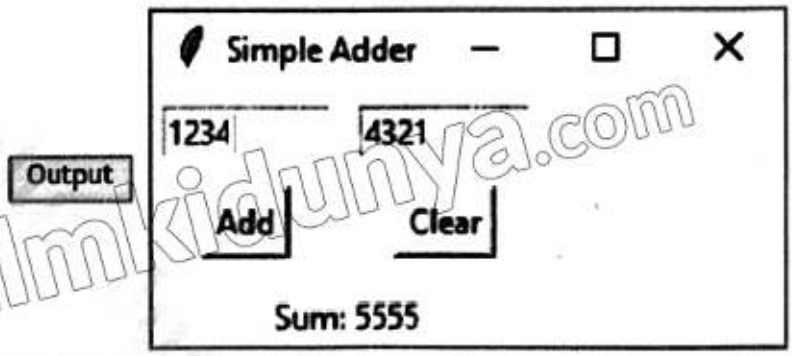


Fig.3.15 : tkinter

Developing Game using tkinter

In terms of GUI application, even games can be developed using tkinter. Let us create a simple game of correctly identification of color such that the text color needs to be identfied and written with correct spellings. The twist however is that the text show name of a different color to confuse the player. Only right color input with correct spellings will be counted.

Steps to create the color game:

1. Create a new Python file and import necessary libraries:
    1. import tkinter as tk
    2. import random
2. Set a list of colors:

colors = ["red", "blue", "green", "yellow", "pink", "black", "orange", "purple", "brown"]

3. Sketch game logic

- Initialize variables like score = 0.
- Outline 'new_round()' function:
    - Select a random color from the colors list for 'color_to_guess'.
    - Select another random color for the text's display color 'random_color'.

6.Subsequently buttons are added in the screen.

```
add_button = tk.Button(root, text="Add", command=add_numbers)
add_button.grid(row=1, column=0, padx=5, pady=5)

clear_button = tk.Button(root, text="Clear", command=clear_fields)
clear_button.grid(row=1, column=1, padx=5, pady=5)
```

We create an "Add" button which when clicked will call "add_numbers()" function. This button is placed in the second row and the first column of our window with a little bit of space around it as in the last step. We also have a "Clear" button that when clicked will call "clear_fields()" function.

7.To display result, a label needs to be added. So, a label is created to show the results. Initially, it says "Result will be displayed here." This label is positioned on the grid in row 2, stretching across both columns using the 'columnspan' attribute. We also add a little bit of padding around the label to make it look nicer.

```
result_label = tk.Label(root, text="Result will be displayed here.")
result_label.grid(row=2, column=0, columnspan=2, padx=5, pady=5)
```

8.Lastly we run the GUI in a continuous manner. Without it, the window will just flash and disappear. This loop ensures that the windows stays open and interactive.

```
root.mainloop()
```

The complete code will look like as shown in Fig 3.15.

```
import tkinter as tk

def add_numbers():
    try:
        num1 = float(entry1.get())
        num2 = float(entry2.get())
        result_label.config(text=f"Sum: {int(num1 + num2)}")
    except ValueError:
        result_label.config(text="Invalid input. Please enter numbers.")

def clear_fields():
    entry1.delete(0, tk.END)
    entry2.delete(0, tk.END)
    result_label.config(text="Result will be displayed here.")

root = tk.Tk()
root.title("Simple Adder")

entry1 = tk.Entry(root, width=10)
entry1.grid(row=0, column=1, padx=5, pady=5)
```

components and is created using tkinter.Tk().

2.Next we define a function to add numbers on the screen.

```
1   import tkinter as tk
2
3   def add_numbers():
4       try:
5           num1 = float(entry1.get())
6           num2 = float(entry2.get())
7           result_label.config(text=f"Sum: {int(num1 + num2)}")
8       except ValueError:
9           result_label.config(text="Invalid input. Please enter numbers."
```

We take in the user input into the first and second boxes using get() function. Next we add the numbers and update the result label to display the sum. If the user types something that's not a number like letters, etc. an error message is displayed.

3.Thereafter we defined another function to clear fields.

```
11  def clear_fields():
12      entry1.delete(0, tk.END)
13      entry2.delete(0, tk.END)
14      result_label.config(text="Result will be displayed here.")
```

To clear the contents of the first text entry box, we use the delete command, starting at the 'position 0' and going all the way to the end. Next, we do the same thing for the second text entry box, clearing its text as well. Finally, we reset the result label to its default message by configuring its text to "...".

4.Create main window along with the title of the program should be the next step using:

```
16  root = tk.Tk()
17  root.title("Simple Adder")
```

5.As a next step we setup input fields:

```
19  entry1 = tk.Entry(root, width=10)
20  entry1.grid(row=0, column=0, padx=5, pady=5)
21
22  entry2 = tk.Entry(root, width=10)
23  entry2.grid(row=0, column=1, padx=5, pady=5)
```

We create a little text box for the user to type in. The 'tk.Entry()' creates the box itself and we have set it to be 10 characters wide. Then using 'grid()' we put it in the top left corner of our window (row 0 and column 0 to be specific). We also added a little bit of space around it i.e. 5 pixels on all sides using padx and pady. Similarly for the second text box next to the first one, in the same row (row 0) but one column over (column 1).

```
try:
    with open("abc.txt", "r") as file:
        for char in file.read():
            print(char, end="")
    print("\nFile read successfully")
except FileNotFoundError:
    print("Error: The file does not exist.")
except IOError:
    print("Error: An issue occurred while reading the file.")
```

Fig.3.14: try-except block

Activity 6

Write a program to open the file 'abc.txt' in read-mode. Create another file 'xyz.txt' and write the contents of abc.txt by reading character by character.

3.2.8 GUI in Python

Graphical User Iinterface (GUI) improves the usability and accessibility of applications as compared to Command-Line Interface (CLI) systems which require users to write commands. In GUI through components like menus, text fields, buttons etc., users may interact with the program more smoothly. Python provides a library for developing GUIs and games namely 'Tkinter to handle common tasks like user input i.e. the steps to be taken when a button is clicked. Tkinter is a built-in Python library used to create GUIs. It allows programmers to create interactive application components, like desktop Applications including word processors, media players and basic utilities where all their interfaces can be made with Tkinter. Tkinter also can be used for creating 2D games with simple graphics. Game elements and windows can be created and user inputs like keyboard and mouse clicks can be controlled using functions provided by Tkinter. Options provided by Tkinter, include:

- The basic components of the interface are called widgets which include checkboxes, buttons, labels and text boxes. We can organize and modify widgets to make the layout as desired.
- Tkinter also is used for event handling like mouse clicks, key presses and window resizing. We can use code to create responses to these events for interactive and dynamic applications.

The arrangement of GUI components and the underlying logic are generally kept separate in the code.

Simple Number Adder with GUI

Before moving on with the code, it is quite helpful to understand the steps involved in creating a GUI using Tkinter library. Let us create a simple Adder application which takes in two numbers as input and when the 'Add' button is pressed, the result gets displayed. Major steps are:

1. First of all, import the tkinter library in Python code and create the main application window which will hold all

```
import tkinter as tk
```

```
1 with open("abc.txt", "w") as file:
2   file.write("The quick brown fox jumps over the lazy dog")
3
4 print("File created and written to successfully!")
```

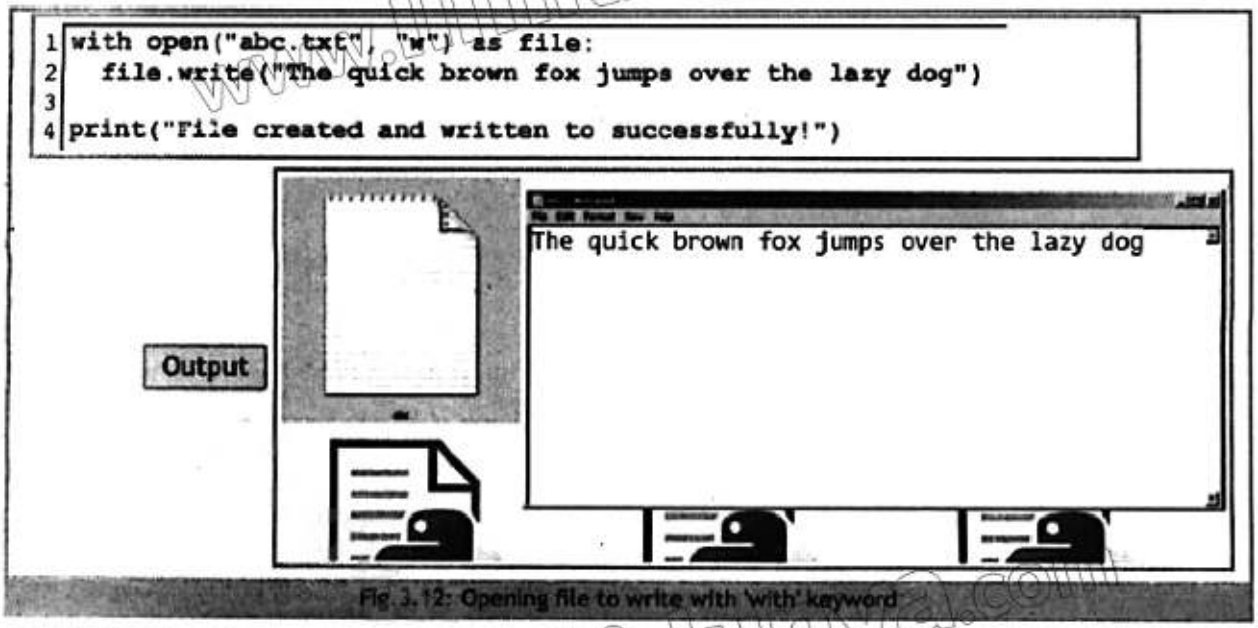


Fig. 3.12: Opening file to write with 'with' keyword

Read Existing Files

Next, let's read the contents of an existing file, like the file 'abc.txt' that we just created and saved. The file "abc.txt" is opened in read mode ("r") in the first line of Fig.3.13. An error message will be displayed if the file is missing. As can be seen in line 2, using a for-loop the whole file is read character-by-character and printed on the screen without adding a new line after each character. Lastly, a confirmation message is displayed on line 4 that the file is successfully read, as shown in the output.

```
1 with open("abc.txt", "r") as file:
2   for char in file.read():
3     print(char, end="")
4 print("\nFile read character by character successfully!")
```

Output

```
The quick brown fox jumps over the lazy dog
File read character by character successfully!
```

Fig.3.13: Program to read file

Errors may arise while dealing with files and may result in abnormal termination of the program. To avoid this, 'Try-except' blocks are used to handle possible problems while accessing and working with files. Typical issues include trying to open a nonexistent file, reading from a file or running into issues when writing data. Alternative actions can be implemented by enclosing file operations in a try block and addressing particular exceptions such as 'FileNotFoundError', 'PermissionError' or 'IOError' in the except block as shown in Fig 3.14.

(with complete path) while the other argument is the desired mode of operation (reading, writing, appending, etc.).

- After the file is opened in read mode, its contents can be extracted by using **read()** function and the complete contents of the file is obtained as a string.
- Write mode allows you to add new material or edit already-existing information to the file using **write()** function.
- You may add new data at the end of an already-existing file without erasing its contents, by using the append mode using the **add()** technique.
- The **close()** function must be used to close a file once you have completed working with it. This guarantees data integrity and releases system memory that was in use by the file.

Example: Program To Create An Empty File And Write Content In It.

For the code in Fig.3.11, in the first line a file named 'abc.txt' is created (if it already does not exist) with **open()** function and returns a file handler. Thereafter, every operation uses this file handler to operate on the file. For example, in the very next line a sentence is written in the file, using **write()** function using 'new_file' file handler. In line 4, a message for the user is displayed such that the contents of the file can be checked by opening it, as shown in the output. Lastly, for every file that is opened, it needs to be explicitly closed before the program terminates, and that is generally the last line of the program.

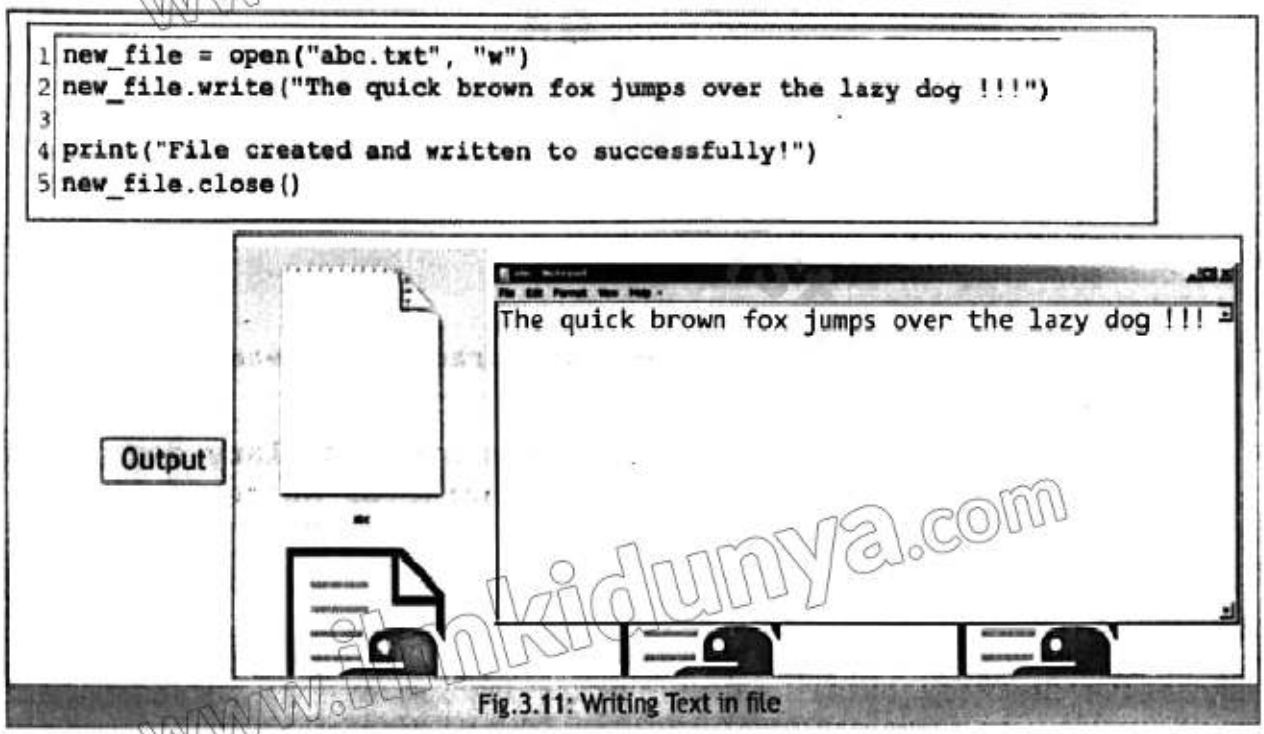


Fig.3.11: Writing Text in file

Alternatively, as shown in Fig.3.12, we can use the 'with' keyword. The benefit of using 'with' keyword for opening file is that the file does not need to be closed explicitly and specially in case of an error, it will automatically be closed. For the sake of clarity, as we are writing the same sentence as of Fig 3.11, but without '!!!' at the end of the sentence, just to distinguish that new text has been updated. When executed, the updated output of the same file is shown in the output.

Fig 3.10 shows the code how to create a list in a dictionary and initialize it, such that the list comprises of all the data related to student in every row. Line 7 shows how to extract the registration number from the list while line 9 depicts how to select marks from the list. Furthermore, on line 12 we insert a new record in the dictionary and print the whole dictionary in the next line which shows all the records in the dictionary. To understand how to access the list elements in the dictionary, let's assume that the second subject marks of the record that we just inserted needs to be updated. In line 19, based on the key being the student's name, we update the marks of the second subject. Lastly, on line 24, using a for loop in terms of the key-value pair we print all the items of the dictionary.

Activity 5

A primary school has subject teachers such that a teacher teaches only 2 subjects to a class. Assuming, each class studies 8 subjects in total. Write a program using list as a value in dictionary, so that 'class' becomes the key and respective 'teachers' are stored as values.

3.2.7 Files in Python

Importance of Disk I/O

Disk input/output (I/O) is vital for efficient data handling and optimal system performance in computers. Disk I/O bridges between the CPU and HDD for information retrieval, such that relevant data is passed to the CPU for processing and later is stored back on permanent storage devices. Almost all computer operations are supported by this continuous data interchange, from launching of the programs, opening files and save changes made by user.

In Python, files are mainly of two types:

Text File:

In a text file, data is stored through characters like .txt, .csv, etc. Point to note here is that a special marker called 'End Of Line' (EOL) generally '\n' is used to highlight completion of a line. This way, Python knows that a new line is starting after EOL marker.

Binary File:

In a binary file data is stored in combination of 0s and 1s, i.e. binary data. Binary files are not meant for humans but rather for other programs to read information. Images (having extension .jpg, .png), audio (.mp3), video (.mp4) and executable programs (.exe) are all examples of binary files.

File Handling Methods and Operations

Data in files from storage devices is brought up in RAM, where functions like creation of new file, reading and updating of already existing file, closing the file, etc. can be performed on the files. Python has file handling methods that let you work with files and you may interact with files in an organized manner and create, read, write and modify their contents programmatically, using:

- The **open()** method is used to access a file such that it takes in 2 arguments, one is file name

3.2.6 List as a Value in Dictionary

A dictionary comprises of a key-value pair and for multiple values against a key can be stored in the dictionary in the form of a list. For example, every record of a student can be stored in the dictionary such that the student's name will act as the key while the registration number and marks will be the values. To store multiple values, a list can be used.

```
students = {
    "Sara": ["345", 85, 90, 78],
    "Jamal": ["678", 75, 88, 92],
    "Zaid": ["912", 80, 79, 95]
}

sara_reg_num = students["Sara"][0]
print("Sara's Reg. No.: ",sara_reg_num)
jamal_marks = students["Jamal"][1]
print("Jamal's Marks for first subject are: ",jamal_marks)

students["Azra"] = ["234", 78, 85, 90]
print(students, "\n")

student_name = "Azra"
subject_index = 2
new_mark = 100
if student_name in students:
      students[student_name][subject_index] = new_mark
      print(f"*** Marks updated for {student_name} at index {subject_index} \n")
else:
    print("Student not found \n")

for name, details in students.items():
    print(f"Name: {name}")
    print(f"Registration Number: ", {details[0]})
    print(f"Marks: [", end='')
    print(details[1],',',details[2],',',details[3], "]\n")
```

Output

```
Sara's Reg. No.:  345
Jamal's Marks for first subject are:  75
{'Sara': ['345', 85, 90, 78], 'Jamal': ['678', 75, 88, 92],
 'Zaid': ['912', 80, 79, 95], 'Azra': ['234', 78, 85, 90]}

*** Marks updated for Azra at index 2

Name: Sara
Registration Number:  {'345'}
Marks: [85 , 90 , 78 ]

Name: Jamal
Registration Number:  {'678'}
Marks: [75 , 88 , 92 ]

Name: Zaid
Registration Number:  {'912'}
Marks: [80 , 79 , 95 ]

Name: Azra
Registration Number:  {'234'}
Marks: [78 , 100 , 90 ]
```

Fig. 3.10: A dictionary with a list.

Activity 3

Tailor the code of Fig 3.8, such that for all odd values of the matrix the sign should be inverted while for all even values multiply it by 2.

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print('Matrix is: ', matrix)
for i in matrix:
    print(i)

trans = [
    [0,0,0],
    [0,0,0],
    [0,0,0]
    ]

for row in range(len(matrix)):
    for col in range(len(matrix)):
        trans[col][row] = matrix[row][col]

print('Transpose of the Matrix is: ',trans)
for j in trans:
    print(j)
```

Be careful about the indices as it may return the original matrix and not the transpose of the matrix.

```
Matrix is:  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
Transpose of the Matrix is:  [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]
```

Output

```
Matrix is:  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
Transpose of the Matrix is:  [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

Fig.3.9: Program to create Transpose of a matrix

Activity 4

A matrix 'X' is symmetric matrix, if X = (Transpose of X). Write code to determine whether a given matrix is symmetric matrix or not.

3.2.5 Nested List

A nested list is a list of lists such that the inner lists or sub-lists can further contain list as elements. Nested list is quite useful in handling complex data like matrices. So, to explore nested list let's try to create a matrix and print it. First, we define a 3x3 matrix. This is achieved using a nested-list such that every element of the list is itself a list and contains row-wise data. As shown in Fig.3.8, we initialized a 3x3 matrix and thereafter on line 6 just by passing the matrix name as an argument to the print() function, we can get it printed. The first line of the output clearly distinguishes the multiple lists that exist in the list. Next, we print the matrix on line 8 using a for-loop. Since, the list has only 3 members, the loop will execute thrice and in every iteration a row gets printed, hence in the form of a matrix. Lastly, on line 11 we use nested loop, such that the outer loop index is 'row' while the inner loop index is 'col'. This way, for the first row, all columns will be printed and then the same will be processed for second and third rows and the whole matrix gets printed in a single line. To modify the code and get it printed in the form of a matrix, is left as an activity for you.

```
1  matrix = [
2      [1, 2, 3],
3      [4, 5, 6],
4      [7, 8, 9],
5  ]
6  print(matrix)
7
8  for i in matrix:
9      print(i)
10
11 for row in range(len(matrix)):
12     for col in range(len(matrix)):
13         print(matrix[row][col], end=' ')
```

Output

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
1 2 3 4 5 6 7 8 9
```

Fig.3.8: Printing a Matrix using Nested List

In this code, a 3x3 matrix is defined, and its transpose is calculated and printed. In line 2-4, the original matrix is given with numbers. In line 7, the code initializes an empty list to store the transpose. Lines 9-12 iterate through each column of the original matrix and create rows for the transpose. In line 13, each new row is appended to the transpose list. Finally, in lines 16-17, the code prints the transposed matrix. The transpose swaps the rows and columns of the original matrix.

Thereafter, we extend the program to calculate the transpose of the matrix and print it. For this purpose, on line 10 of Fig.3.9 we initialized another matrix. On line 16, just like in the last example, we placed a nested loop as we know how to access the nested list contents one-by-one. But, while assigning these values to the transpose matrix the rows should be converted into columns and vice versa. Therefore, note that the indices of the transpose matrix are reversed, which will result in the element at row 1 and column 2 to be placed at column 1 and row 2, which was the desired outcome. In lines 17-19, the output reflects successful creation of transpose matrix of the given matrix.

```
student_test_number = {}
while True:
    name = input("Enter student name (or 'q' to quit): ")
    if name.lower() == 'q':
        break
    grade = float(input("Enter Test-Number for {}: ".format(name)))
    student_test_number[name] = grade

name = input("Enter student name to view test-number: ")
if name in student_test_number:
    print(f"{name}'s test-number: {student_test_number[name]}")
else:
    print(f"Student '{name}' not found.")
```

Output

```
Enter student name (or 'q' to quit): Amir
Enter Test-Number for Amir: 90
Enter Student name (or 'q' to quit): Bilal
Enter Test-Number for Bilal: 87
Enter Student name (or 'q' to quit): Dilawar
Enter Test-Number for Dilawar: 92
Enter student name (or 'q' to quit): Zaid
Enter Test-Number for Zaid: 88
Enter student name (or 'q' to quit): q
Enter student name to view test-number: Bilal
Bilal's test-number: 87.0
```

Fig.3.6: Search an entry in dictionary

Note that, it takes significantly less time to find a value in a dictionary than it does to search a list in Python. This is the reason because in lists we need to go through each item individually in order to locate what you're searching for whereas in dictionaries using the key-value pair, we can locate the searching item quickly.

Finding a Value in a Dictionary v/s List

For the sake of better understanding, a program was written where a list and dictionary were randomly populated with 1 lakh entries. Such that a list of strings of 4-characters was setup while for every 4-character in the dictionary a corresponding key value was randomly arranged. Next 1 random element was searched in both the programming constructs and their times were noted. The difference of time clearly shows the searching in dictionary is faster as compared to the list, as shown in Fig. 3.7. Sample of populated data in dictionary and list are shown in first two lines.

```
Search results (Dictionary): {20145: 'dlei'}
Search results (List): ['uiib']
Dictionary search time: 1.000166 msec
List search time: 3.000259 msec
Difference of time: Dictionary Time - List Time: -2.000093 msec
```

Fig.3.7: Difference of time to Search in dictionary v/s list.

students and their respective test numbers from the user are taken in, as shown in Fig.3.6. Lastly, the student name is asked to be searched in the dictionary and corresponding search result is displayed.

```
1  student_grades = {
2      'Amir': 90,
3      'Bilal': 85,
4      'Dilawar': 92
5  }
6
7  for student, grade in student_grades.items():
8      print(f"{student}: {grade}")
9
10 student_grades['Zaid'] = 88
11 student_grades['Bilal'] = 87
12
13 del student_grades['Dilawar']
14
15 if 'Dilawar' in student_grades:
16     print(f"\n Dilawar's grade:", student_grades['Dilawar'], "\n")
17 else:
18     print("Student Dilawar not found \n")
19
20 for student, grade in student_grades.items():
21     print(f"{student}: {grade}")
```

Output

output without line 13

```
Amir: 90
Bilal: 85
Dilawar: 92

 Dilawar's grade: 92

Amir: 90
Bilal: 87
Dilawar: 92
Zaid: 88
```

Confirmation via printing

Output with line 13

```
Amir: 90
Bilal: 85
Dilawar: 92
Student Dilawar not found

Amir: 90
Bilal: 87
Zaid: 88
```

Confirmation via printing

Fig.3.5: Program to create dictionary for storing students' grades

Activity 2

Extend the program of Fig 3.5 in such a way that if the searched item is not found in the dictionary, ask the user if it needs to be added. If user presses 'y' for yes, then the key and respective value (to be taken input) are to be added.

```
1 grades = {85, 92, 78, 92, 85, 95}
2 print(grades)
3 grades.add(88)
4 print(grades)
5 if 92 in grades:
6   print("student has 92 among the grades.")
```

Output

```
{92, 85, 78, 95}
{78, 85, 88, 92, 95}
Student has 92 among the grades.
```

Fig.3.4: Sets

3.2.4 Dictionary

Dictionary Store Key-Value Pairs

Dictionary is an essential data structure in Python that is used to store and arrange data in an organized way using key-value pairs such that each key represents a distinct value, i.e. each key is associated with a particular value. List have index while in dictionary it can be of any data type. For example, in a phone book where the phone number (value) and the person's name (key) are associated with each other. By searching for the matching name, we may quickly get the phone number. Because of this fundamental idea, dictionaries may be used for storing and retrieving information by associating unique keys with values.

In order for keys to be used for searching, they must be unchangeable types of data like strings, integers or tuples. Values may store complicated data structures with flexibility since they can be any data type including texts, integers, lists and even other dictionaries. By design dictionaries avoid duplicate keys such that every key acts as a unique label hence avoiding reshuffling of data. Dictionaries maintain an ordered link among keys and their corresponding values, like keeping track of passwords and usernames, etc.

The Python code in Fig.3.5 shows how to add, create and retrieve entries from a dictionary. Initially a dictionary is initialized with 3 student records where student names being the key and their respective marks are the values associated with the keys. Immediately after that on line 7 and 8 using for loop, we print the dictionary. On line 10, we add another student 'Zaid' in the dictionary, which is added at the end of the list. A similar statement for 'Bilal' is written, since it is already in the dictionary, its value will be updated. On line 13, we removed 'Dilawar' from the dictionary. For confirmation, in the next 4 lines using conditional statements (if-else) we checked for 'Dilawar' in the dictionary. Lastly, final dictionary is printed again. Point to note here is that if we comment line 13, record of 'Dilawar' will not be deleted. The corresponding outputs of the said program and after commenting line 13, both are shown for better understanding.

Next, we modify the code such that an empty dictionary is created and thereafter, name of

```
1 grades = (85, 92, 78, 92, 85, 95)
2 print(grades)
3 print("First grade:", grades[0])
4 print("Third grade:", grades[2])
5 if 92 in grades:
6   print("Student got 92")
7 print(grades.count(92))
8 new_grades = grades + (88, 90)
9 print(new_grades)
```

Output

```
(85, 92, 78, 92, 85, 95)
First grade: 85
Third grade: 78
Student got 92
2
(85, 92, 78, 92, 85, 95, 88. 90)
```

Fig.3.3: Tuples

On line 1 we create a tuple of student grades namely 'grades' and initialize it with some sample grades and print it in the next line which shows duplicate values as well. However, individual grades are accessed by specifying the index (which starts from 0) on line 3 and 4. To check if a specific grade exists in the tuple the `in` operator can also be used with tuples as we used in line 5. Additionally, to find the number of occurrences of a grade `count()` method returns the number of times a specific value appears in the tuple as shown in line 7. Line 8 shows that we can create a new tuple by concatenating existing tuples using the `+` operator.

3.2.3 Sets

Sets in Python are represented using '{_,_,_}' and are collections of unique and unordered elements i.e. a set cannot comprise of duplicate values. A set will only hold one instance even if the same value is added to it more than once. Sets can be used to store a list of unique usernames, product IDs, etc. Using the 'in' operator it is quite easy to check whether a particular element exists in a set or not.

Initially on line 1 a set of student grades is created namely 'grades' and is initialized with some sample grades. Notice that duplicate grades (85 and 92) exist in the initial assigned values. Since sets only store unique values, the duplicates will be automatically removed. Line 2 prints the set 'grades', demonstrating that the duplicates have been removed.

Next, we add a new grade to the set i.e. 88 on line 3 using the `add()` method. Line 4 prints the set again showing the updated values. To check if a specific grade exists in the set, the `in` operator is used. In line 5, if 92 is found in the set, then a message is printed.

```
1  student_grades = [85, 92, 78, 95, 82]
2
3  student_grades.insert(2, 90)
4  print("Grades after adding a new grade:", student_grades)
5  student_grades.append(100)
6  print("Grades after adding another new grade:", student_grades)
7
8  student_grades.remove(78)
9  print("Grades after removing a grade:", student_grades)
10 lowest_grade = student_grades.pop(2)
11 print("Grades after removing the grade on index 2:", student_grades)
12
13 if 88 in student_grades:
14     student_grades.remove(88)
15 else:
16     print("Grades88 not found in the list")
17
18 average_grade = sum(student_grades) / len(student_grades)
19 print("Average grade:", {average_grade})
```

Output

```
Grades after adding a new grade: [85, 92, 90, 78, 95, 82]
Grades after adding another new grade: [85, 92, 90, 78, 95, 82, 100]
Grades after removing a grade: [85, 92, 90, 95, 82, 100]
Grades after removing the grade on index 2: [85, 92, 95, 82, 100]
Grade 88 not found in the list
Average grade: {90.8}
```

Fig.3.2: Add and remove items from a list

Activity 1

Extend the program of Fig.3.2 to search for all odd numbers in the list and remove them.

Moreover, there are scenarios, where a particular value is not in the list but you intend to remove it and results in an error. So, a good programming practice is to use conditional (if-else) statement as we have used in lines 13-16. This way, if the number is in the list, it will be deleted otherwise it won't abnormally terminate the program rather a message is printed that the said number is not found, or something like this. Lastly, using the **sum()** and **len()** functions, we calculated the average grade of the student without using any loop.

3.2.2 Tuples

In Python, tuples are represented using '(_,_,_)' and are a special type of list having one feature that they are immutable i.e. a tuple's contents cannot be updated once it has been created. We can view the objects and their arrangements but we cannot add, remove or rearrange them. Tuples are used when a set of values must remain in order and unchanging throughout the program. For example the days of the week (("Monday", "Tuesday", "Wednesday")) or geographical coordinates (latitude and longitude), etc.

terms, a 'Vehicle' may have attributes such as speed, color, wheels and operations like accelerate() and brake(). If we specify it further for a 'Car' which is a type of 'Vehicle', so we do not need to define the above stated attributes and operations again because 'Car' inherited these from 'Vehicle' being a derived/sub class and automatically owns these properties and functions of 'Vehicle'. However, 'Car' may have its own additional properties like 'seatingCapacity' and operation like openSunroof().

Polymorphism allows objects from various classes to be considered as instances of a common type, though the implementation of the common type differs among the various classes in their own unique way. This allows writing generic code that can deal with a wide variety of objects. Polymorphism makes it easier to modify object behavior.

For example, the common type 'Shape' can be a triangle, rectangle or circle. Though all three are shapes but have their own distinct properties and operations like draw, resize, calculate-area, etc. Polymorphism is useful in such scenarios where a common operation applies for all but implementation differs based on the type of shape. In context of specific shapes, triangle needs base and height, rectangle uses length and width while circle requires radius to calculate the area. Instead of writing three separate specific calculate-area methods (which may increase the more shapes we take into consideration), via polymorphism we can send a simple instruction to different objects such as 'draw-yourself' and each object will perform the operation in its own prescribed way on the basis of type of shape.

3.2 Programming Constructs in Python

3.2.1 List

List is a basic data structure in Python that holds groups of elements in a sequential manner. This flexibility allows list to store elements of various data types such as boolean, integers, etc. The order that things are added is maintained by lists. Index 0 will have the first item inserted, index 1 will have the second, and so on. This arrangement is essential for retrieving elements based on their location. Different data types can be stored as elements in the same list. List sizes are dynamic, i.e. elements can be added or removed as required.

In Fig.3.2, let's utilize the common functions of adding and removing items from the list. Initially, on line 1 we defined a list namely 'student_grades' with 5 elements in it, to store grades of a student for instance. On line 3, we inserted an additional grade of 90 in the list on index 2, using the **insert()** function. Alternative to insert any other value is by using the **append()** function, which we used in line 5 and inserted another value of 100 which was added at the end of the list. Thereafter, on line 8 the value of 78 from the list was deleted using **remove()** function. Another way to remove any element from the list is using the **pop()** function, such that index number needs to be mentioned. This way, we can remove an element from the list either if the element or its index number is known.

properties, such as size, color or number of rooms. All objects have the same basic structure as defined by the class, apart its own specific characteristics. Objects in a program interact with one another to imitate real-world situations and improve the organization and modularity of the code.

The 'StationaryItem' class (as shown in table) acts as the stencil for all stationary items in the bookstore. Individual goods developed by this session include pens, pencils and notebooks. Each item has unique values for attributes such as name, price and color. The methods of the class allow you to retrieve or change these attributes which allows managing of information about various stationary items easily.

Class: StationaryItem		
Attributes:	Object 1: A blue XYZ pen.	Object 2: A red ABC pencil.
name (e.g., "Pen", "Pencil", "Notebook")	name: "Pen"	name: "Pencil"
price	price: 200	price: 100
color (e.g., "Blue", "Black", "Red", "Green")	color: "Blue"	color: "Red"
brand (e.g., "XYZ", "ABC", "DEF")	brand: "XYZ"	brand: "ABC"
It can have methods like get_price() to obtain the price of the item, set_price() to adjust the price of the said item, etc.		

Encapsulation is the process of information hiding by combining data (attributes) and the methods that operate on the data into a single entity, i.e. a class. The main concept is to keep together the data and methods that are relevant. Encapsulation does not allow direct access to some components but via some designated methods provided by the object. This way internal working of an object is hidden. Encapsulation protects data from unwanted access or changes and also improves code to be written in modules and easy to read and understand.

For instance, in a bank, the account holder's name, account number and balance are critical information. Rather than changing the balance directly, deposit(amount) method will add the amount to the balance while withdraw(amount) method will subtract the said amount from the balance. Further, get_balance() method will retrieve current balance without changing the balance figures. This way, critical information are encapsulated and cannot be modified directly.

Abstraction simplifies creating complicated scenarios by showing only the necessary information and usefulness. It provides only essential information about the data to others, hiding the background details or implementation. Consider a vehicle for example, where we are mainly concerned with common operations such as start, stop and accelerate. But the underlying mechanism of how engine burns fuel and shifting of gears, etc. are kept hidden.

Inheritance allows you to create new classes (subclasses or derived classes) from existing parent classes. Subclasses inherit the attributes and methods of their parent class, helping in reducing repetition of codes. This idea promotes the efficient extension of existing structures. In general

3.1.2 Object Oriented Programming

Object Oriented Programming (OOP) is software development paradigm based on the data and objects, which include functions (methods) that define the behavior and data (attributes) to indicate the properties of objects. These are the program's building blocks to regenerate the actual objects and their functionalities, in code. The programs designed using OOP paradigm represents the physical attributes of the objects from real world and therefore developers have a more natural understanding of the problem to be solved by devising the program around objects and their actions.

The main advantage of using OOP paradigm is that the emphasis in problem solving is on objects and not the logic. The program reflects real-world objects like car, fruit, etc. This approach makes it easier to understand the problem, design solution and change the code if needed. OOP is implemented through classes which allow defining initially the object and their functionalities and later physical instances are created on spot when need to be used in the program. This is quite useful for complex systems where developers can map objects and their associations to be realistic and handle them more efficiently. OOP limits direct access to data for safety and reliability of data and programs.

Table.3.2: Pros & Cons of Object Oriented Programming

Pros	Cons
Simplifies complex systems by breaking them into smaller reusable components.	May result in complex designs if not implemented thoughtfully.
Hides implementation details, hence improving code maintainability and security.	At times code is harder to understand and debug due to encapsulation.
Establishes hierarchical relationships between classes.	Rigid design due to inheritance may reduce flexibility.
Objects of different types are treated as if they were of the same type.	Polymorphism increases complexity, making code harder to understand and reason about.
Represents real-world objects and their relationships effectively.	May not be well-suited for certain problems, such as computationally intensive tasks.

Key Concepts

A class is an object creation stencil or template. It specifies a collection of methods (functions) and attributes (variables or properties) that its objects will hold. You are creating an instance of a class when you create an object from it. Although each object has a distinct set of data (attribute values), they all use the same class-defined methods. You may effectively express and control complicated data structures and behaviors in your program with this paradigm.

An object is an instance of a class that denotes a physical thing, such a person, a car or a bank account. Every object has distinct properties (attributes) and behaviors (methods). For instance, each home (object) constructed from a class that specifies the structure of a house has particular

3.1 The Programming Paradigm

The term "programming paradigm" describes a vital process for creating and organizing computer program code. It helps in the structure of code, program flow and managing of data for programmers. In other words it can be termed as a particular methodology to use programming language for problem solving. Although programmers can design their own unique structures and functions inside a language, these can't be termed as a paradigm, as it is the programming language that specifies and provides guidelines and functions in terms of what and how to develop. Thus, a programming language paradigm offers a particular way to arrange various functionalities in order to accomplish specified objectives. Additionally, it facilitates the developers with standard tools and implementation techniques that are used commonly for effective problem-solving which helps developers to focus on the specific task in hand and design solution effectively. The most commonly used programming languages include Python, Java, SQL, etc.

3.1.1 Functional Programming

Functional programming is the paradigm of software development which was and is followed by many, especially for small applications where the code is divided into functions based on functionality. The developers in functional programming focus more on 'what' is to be achieved rather than 'how' to address the problem. The solution of the problem in hand is divided into segments and each segment is typically addressed in functions. This way, every step of the solution is mapped in a group of functions as shown in Fig. 3.1.

Fig.3.1: Functional programming

Table.3.1: Pros & Cons of Functional Programming

Pros	Cons
Simpler code.	Not easy for beginners. Needs practice
Code is easier to track for logic and debug.	May involve with overhead especially when handling large data structures.
Functions are easier to test and run.	Complex algorithms may be harder to express, often requiring higher-order functions.
Naturally supports parallel and concurrent programming.	Requires careful handling of data sharing and synchronization for efficient execution.
Emphasizes on what to compute rather than how, hence more concise code.	May be less efficient in performance-critical scenarios.

So, if the flow is clear how to achieve the solution, it becomes easier in functional programming. This way, updating the code for any changes becomes simpler as only that particular functions(s) need to be modified and whole program does not need to be read and understood