

Annexure-I

DSA Self-paced

GeeksForGeeks

A training report

Submitted in partial fulfillment of the requirements for the award of degree of

**Bachelor of Technology
(Computer Science Engineering)**

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB



**L OVELY
P ROFESSIONAL
U NIVERSITY**

From 05/25/22 to 07/10/22

SUBMITTED BY

Name of Student: Aman Raj

Registration Number: 12101946

Signature of the student:

Annexure-II: Student Declaration

To whom so ever it may concern

I, **Aman Raj, 12101946,** hereby declare that the work done by me on “**DSA Self-paced**” from **May, 2022** to **July, 2022**, is a record of original work for the partial fulfillment of the requirements for the award of the degree, **Bachelor of Technology.**

A handwritten signature in blue ink, appearing to read 'Aman Raj', with a horizontal line underneath.

Signature of the student:

CERTIFICATE:



CERTIFICATE OF COURSE COMPLETION

THIS IS TO CERTIFY THAT

Aman Raj

has successfully completed the course on DSA Self paced of duration 8 weeks.

Sandeep Jain

Mr. Sandeep Jain
Founder & CEO, GeeksforGeeks

<https://media.geeksforgeeks.org/courses/certificates/f8491dba6aca56a7069852b00a5c2e5d.pdf>

Link:

<https://media.geeksforgeeks.org/courses/certificates/f8491dba6aca56a7069852b00a5c2e5d.pdf>

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to the teacher and instructor of the course DSA Self-Paced from GeeksForGeeks who provide me the golden opportunity to learn a new technology from home. I would like to also thank Lovely Professional University for offering such a course which not only improve my programming skill but also taught me other new technology. Then I would like to thank my parents and friends who have helped me with their valuable suggestions and guidance for choosing this course. I would also like to thank my all classmates who have helped me a lot.

Table of Content

S. No.	TITLE	
1	Cover Page	
2	Declaration of the student	
3	Training Certificate	
4	Acknowledgement	
5	Table of content	
6	Introduction	
7	Technology learnt	
8	Reason for choosing DSA	
9	Learning Outcome	
10	Project Made	
11	References	
	End	

INTRODUCTION

➤ The course name DSA stands for “Data Structures and Algorithms” and Self-paced.

➤ What is Data Structure? Data Structure is a way of collecting and organizing data

in such a way that we can perform operations on these data in an effective way.

Data Structures is about rendering data elements in terms of some relationship, for

better organization and storage. For example, we have some data which has,

player's name "Virat" and age 26. Here "Virat" is of String data type and 26 is

of integer data type.

➤ What is Algorithm? An algorithm is a finite set of instructions or logic, written in

order, to accomplish a certain predefined task. Algorithm is not the complete code

or program, it is just the core logic(solution) of a problem, which can be expressed

either as an informal high-level description as pseudocode or using a flowchart

➤ This course is a complete package that helped me learn Data Structures and

Algorithms from basic to an advanced level. The course curriculum has been

divided into 8 weeks where one can practice questions & attempt the assessment

tests according to his own pace. The course offers me a wealth of programming

challenges that will help me to prepare for interviews with top-notch companies like Microsoft, Amazon, Adobe etc.

TECHNOLOGY LEARNT

- Learn Data Structures and Algorithms from basic to an advanced level like:
- Learn Topic-wise implementation of different Data Structures & Algorithms as follows:

• Analysis of Algorithm

In this I learned about background analysis through a Program and its functions.

• Order of Growth

- ❖ A mathematical explanation of the growth analysis through limits and functions.
- ❖ A direct way of calculating the order of growth

• Asymptotic Notations

1. Best, Average and Worst-case explanation through a program.
 - 1.1. Big O Notation
 - 1.1.1. Graphical and mathematical explanation.
 - 1.1.2. Calculation
 - 1.1.3. Applications at Linear Search
 - 1.2. Omega Notation
 - 1.2.1. Graphical and mathematical explanation.
 - 1.2.2. Calculation.
 - 1.3. Theta Notation
 - 1.3.1. Graphical and mathematical explanation.
 - 1.3.2. Calculation.
 - 1.4. Analysis of common loops
 - 1.4.1. Single, multiple and nested loops

- **Analysis of Recursion**

Various calculations through Recursion Tree method

- **Space Complexity**

- ❖ Basic Programs
- ❖ Auxiliary Space
- ❖ Space Analysis of Recursion
- ❖ Space Analysis of Fibonacci number

MATHEMATICS

- Finding the number of digits in a number.
- Arithmetic and Geometric Progressions.
- Quadratic Equations.
- Mean and Median.
- Prime Numbers.
- LCM and HCF
- Factorials
- Permutations and Combinations
- Modular Arithmetic

BITMAGIC

- 1.1. Bitwise Operators in C++
 - 1.1.1. Operation of AND, OR, XOR operators
 - 1.1.2. Operation of Left Shift, Right Shift and Bitwise
 - 1.1.3.
- 1.2. Problem (With Video Solutions): Check Kth bit is set or not
 - 1.2.1. Method 1: Using the left Shift.
 - 1.2.2. Method 2: Using the right shift

RECURSION

- Introduction to Recursion
- Applications of Recursion
- Writing base cases in Recursion
 - ✓ Factorial
 - ✓ N-th Fibonacci number

ARRAYS

- Introduction and Advantages
- Types of Arrays
 - Fixed-sized array
 - Dynamic-sized array
- Operations on Arrays
 - Searching
 - Insertions
 - Deletion

Arrays vs other DS

Reversing - Explanation with complexity

SEARCHING

- Binary Search Iterative and Recursive
- Binary Search and various associated problems
- Two Pointer Approach Problems

SORTING

- Implementation of C++ STL sort () function in Arrays and

Vectors

Time Complexities

- Sorting in Java
- Arrays.sort() in Java
- Collection.sort() in Java
- Stability in Sorting Algorithms

Examples of Stable and Unstable Algos

- Insertion Sort
- Merge Sort
- Quick Sort

o Using Lomuto and Hoare

o Time and Space analysis

o Choice of Pivot and Worst case

- Overview of Sorting Algorithms

MATRIX

- Introduction to Matrix in C++ and Java
- Multidimensional Matrix
- Pass Matrix as Argument
- Printing matrix in a snake pattern
- Transposing a matrix
- Rotating a Matrix
- Check if the element is present in a row and column-wise sorted matrix.
- Boundary Traversal
- Spiral Traversal
- Matrix Multiplication

- Search in row-wise and column-wise Sorted Matrix

HASHING

- Introduction and Time complexity analysis
- Application of Hashing
- Discussion on Direct Address Table
- Working and examples on various Hash Functions
- Introduction and Various techniques on Collision Handling
- Chaining and its implementation
- Open Addressing and its Implementation
- Chaining V/S Open Addressing
- Double Hashing
- C++

o Unordered Set

o Unordered Map

- Java

o HashSet

o HashMap

STRINGS

- Discussion of String DS
- Strings in CPP
- Strings in Java
- Rabin Karp Algorithm
- KMP Algorithm

LINKED LIST

- Introduction
 - o Implementation in CPP
 - o Implementation in Java
 - o Comparison with Array DS
- Doubly Linked List
- Circular Linked List
- Loop Problems
 - o Detecting Loops
 - o Detecting loops using Floyd cycle detection
 - o Detecting and Removing Loops in Linked List

STACK

- Understanding the Stack data structure
- Applications of Stack
- Implementation of Stack in Array and Linked List
 - o In C++
 - o In Java

QUEUE

- Introduction and Application
- Implementation of the queue using array and LinkedList
 - o In C++ STL
 - o In Java
 - o Stack using queue

DEQUE

- Introduction and Application
- Implementation
 - o In C++ STL
 - o In Java
- Problems (With Video Solutions)
 - o Maximums of all subarrays of size k
 - o Array Deque in Java
 - o Design a DS with min max operations

TREE

- Introduction
 - o Tree
 - o Application
 - o Binary Tree
 - o Tree Traversal
- Implementation of:
 - o Inorder Traversal
 - o Preorder Traversal
 - o Postorder Traversal
 - o Level Order Traversal (Line by Line)
 - o Tree Traversal in Spiral Form

BINARY SEARCH TREE

- Background, Introduction and Application
- Implementation of Search in BST
- Insertion in BST
- Deletion in BST
- Floor in BST

- Self-Balancing BST
- AVL Tree

HEAP

- Introduction & Implementation
- Binary Heap
 - o Insertion
 - o Heapify and Extract
 - o Decrease Key, Delete and Build Heap
- Heap Sort
- Priority Queue in C++
- Priority Queue in Java

GRAPH

- Introduction to Graph
- Graph Representation
 - o Adjacency Matrix
 - o Adjacency List in CPP and Java
 - o Adjacency Matrix VS List
- Breadth-First Search
 - o Applications
- Depth First Search
 - o Applications
- Shortest Path in Directed Acyclic Graph

- Prim's Algorithm/Minimum Spanning Tree
 - o Implementation in CPP
 - o Implementation in Java
- Dijkstra's Shortest Path Algorithm
 - o Implementation in CPP
 - o Implementation in Java
- Bellman-Ford Shortest Path Algorithm
- Kosaraju's Algorithm
- Articulation Point
- Bridges in Graph
- Tarjan's Algorithm

GREEDY

- Introduction
- Activity Selection Problem
- Fractional Knapsack
- Job Sequencing Problem

BACKTRACKING

- Concepts of Backtracking
- Rat In a Maze
- N Queen Problem

DYNAMIC PROGRAMMING

- Introduction
- Dynamic Programming
 - o Memoization
 - o Tabulation

TREE

- Introduction
 - o Representation
 - o Search
 - o Insert
 - o Delete
- Count Distinct Rows in a Binary Matrix

SEGMENT TREE

- Introduction
- Construction
- Range Query
- Update Query

DISJOINT SET

- Introduction
- Find and Union Operations
- Union by Rank
- Path Compression

- Kruskal's Algorithm

- Improved my problem-solving skills by practicing problems to become a stronger Developer

- Practice problems

This track contains many practice problems for the users which are considered important and must-do as far as Data Structure and Algorithm is concerned.

- Developed my analytical skills on Data Structures to use them efficiently

- Solved problems asked in product-based companies' interviews

- Solved problems in contests similar to coding round for SDE role

Reason for choosing DSA

“If your goal is to build new things and become a computer scientist, you simply can’t reach that level without DSA.”

Data structures and algorithms are foundational to programming. Programming languages come and go but DSA has always been there, providing efficiency to software development.

Data structures are used to store and organize data in a logical manner whereas an algorithm is a process we use to execute a certain task.

Both these combined allow smartphones, computers, and websites to make efficient decisions and function smoothly.

However, many programmers don't value the importance of DSA and skip it entirely as they either find it complex or they don't see much value in modern programming. But, that couldn't be further away from the truth.

Learning DSA goes a long way in making them good programmers and setting them up for top-notch careers.

LEARNING OUTCOME

DSA enables you to solve real-world problems

Understanding data structures and algorithms enables you to understand the problem statements on a deeper level and create logical solutions to solve them. The use of DSA is not just limited to computing.

Maybe you haven't noticed but you've been subconsciously using DSA to perform simple tasks in your lives. Suppose we have to search for the word "Programming" in the dictionary.

One way to do it is to read each word from the first page and see if it matches our word. However, this process can take you hours if not days. This approach of searching in the DS world is called [Linear Search](#), and we humans are smart enough to not use this technique.

Instead, we use another search algorithm - [Binary search](#) - even though we're not aware of the term. We go straight to the middle page of the dictionary and compare the words on that page with 'programming'. If 'programming' is alphabetically lower than the words on that page, we ignore all other pages on the right.

And, if 'programming' is alphabetically higher we will ignore the pages on the left. We will keep repeating the process of filtering from the middle of the range until we find the right word.

Similarly, DSA helps a programmer break down real-world problems into smaller steps, providing them with a framework to reach desired solutions.

Think of how books are stored in different sections in a library. If that wasn't the case, we would just be scratching our heads trying to find one book out of a lot of thousands.

*Since they're kept according to the subject/genre /writer's name in a **structure**, it's easy to find the book we're looking for.*

DSA helps in writing Optimized Code

Data structures and algorithms help you in writing optimized code. If you can choose the best-fit data structure that requires the least space and the algorithm that requires the least amount of time to execute the code, the companies could ask for nothing more.

Let's say you are a company that has millions of rows of customer data and you have to find a particular customer's details:-

- If you store data in an unsorted list and use the linear search algorithm, the program will go through each customer and take forever to get the desired output.
- But if you store data in a sorted array and use a 'binary search algorithm' instead, it'll be a matter of seconds before you get the desired output. (Binary search repeatedly divides the search interval in half to get the required element.

Do you see how using the correct data structure and algorithm to write an optimized code can save hundreds of hours in a

simple instance? Optimized solutions help companies limit the use of resources spent on a particular project.

Thus, knowing DSA inside out is of utmost importance in the process of being a good programmer.

DSA opens up the path to becoming a Computer Scientist

Though 90% of your coding will be concerned with using library functions, you need to understand the logic behind these abstractions if you want to use them properly.

There are two key reasons why a developer chooses Software Engineering - either to get a high-paying job or simply because they love programming.

The second category of people loves innovating new things and building software based on experiments.

Computer scientists build amazing things in the real world using in-depth theories and core concepts of computer science. Without a doubt, they need to be good at data structures and algorithms as it's the core concept in the world of programming.

So, if your goal is to build new things and become a computer scientist, you simply can't reach that level without DSA.

Improves Adaptability to Emerging Tech Stacks

Frameworks will come and go. A few years earlier, EJB was a big thing but Spring has almost replaced it. There are many frameworks available for different languages, and they have a shorter lifespan. But DSA will stay forever.

Being the fundamental concept in programming, DSA stays the same no matter what technology is in use. People with a

stronghold on DSA adapt to different tech stacks easily as they understand the crux of programming better.

Helps you crack the top tech companies

Most apex companies want to hire people who have strong fundamentals in programming. This is the reason why they test candidates on Data structures and algorithms.

Big companies like FAANG (Facebook, Amazon, Apple, Netflix, and Google) and other tech giants quite often have to deal with huge complex problems. These problems need to be solved with minimal attention and time to keep the functions running.

And, it's no surprise that people with proficiency in DSA can solve problems quicker. Hence, these companies keep questions about various Data structures & algorithms in their first-level interviews.

Programming is all about data structures and algorithms. Data structures are used to hold data while algorithms are used to solve the problem using that data. Data structures and algorithms (DSA) goes through solutions to standard problems in detail and gives you an insight into how efficient it is to use each one of them. It also teaches you the science of evaluating the efficiency of an algorithm. This enables you to choose the best of various choices.

For example, you want to search your roll number in 30000 pages of documents, for that you have choices like Linear search, Binary search, etc. So, the more efficient way will be Binary search for searching something in a huge number of

data. So, if you know the DSA, you can solve any problem efficiently. The main use of DSA is to make your code scalable because

- Time is precious
- Memory is expensive

PROJECT MADE

Phone directory application using doubly-linked lists

Phonebook management have three main operations:

1. Searching
2. Sorting
3. Deleting

These three operation can be performed efficiently (among above data structures) with Linked List. Doubly Linked List because while searching first element but the current status of pointer is in between middle and first element so it should traverse backward because then it will take less time.

Time complexities in O

1. Searching $O(n)$
2. Deletion $O(n)$
3. Sorting $O(n * \log n)$

OPERATIONS IMPLEMENTED

- 1) DELETE SAME NUMBER
- 2) DELETE SAME NAME
- 3) SEARCH
- 4) DELETE CONTACT

5)DISPLAY CONTACT-In sorted(bubble sort) display

6)UPDATE DETAILS-

A)NAME

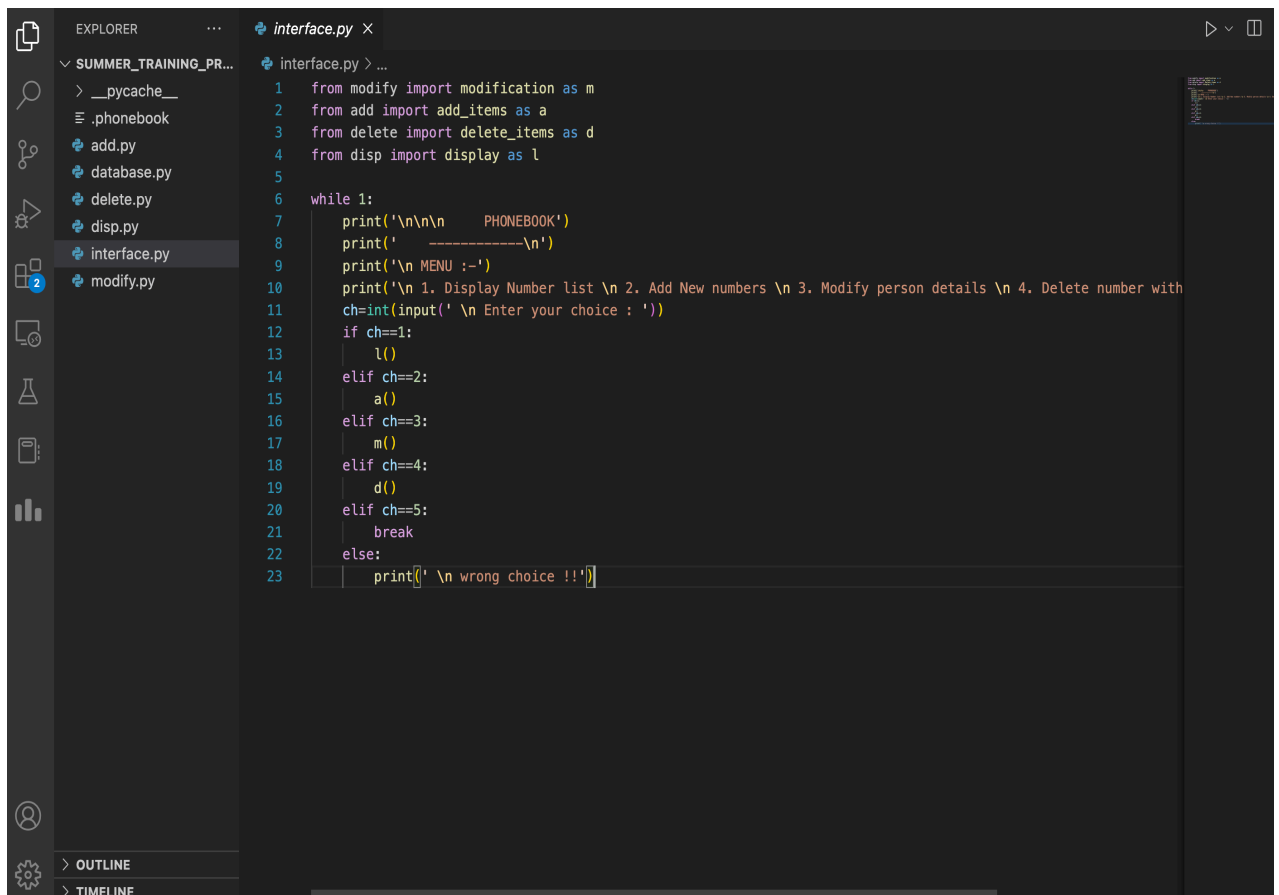
B)NUMBER

C)G-MAIL

7)INSERT CONTACT

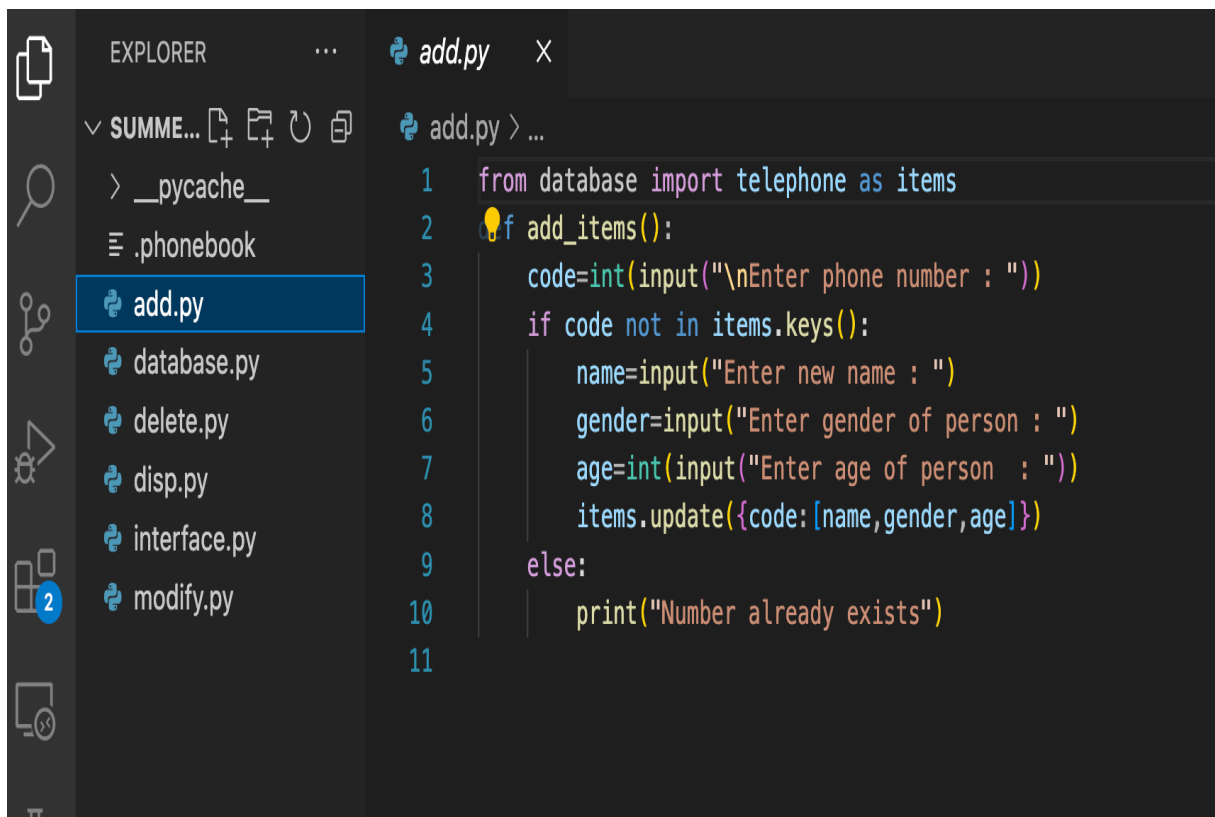
Screenshots of the code

Interface



```
1  from modify import modification as m
2  from add import add_items as a
3  from delete import delete_items as d
4  from disp import display as l
5
6  while 1:
7      print('\n\n\n    PHONEBOOK')
8      print('    -----')
9      print('\n MENU :-')
10     print('\n 1. Display Number list \n 2. Add New numbers \n 3. Modify person details \n 4. Delete number with
11     ch=int(input(' \n Enter your choice : '))
12     if ch==1:
13         l()
14     elif ch==2:
15         a()
16     elif ch==3:
17         m()
18     elif ch==4:
19         d()
20     elif ch==5:
21         break
22     else:
23         print('\n wrong choice !!')
```

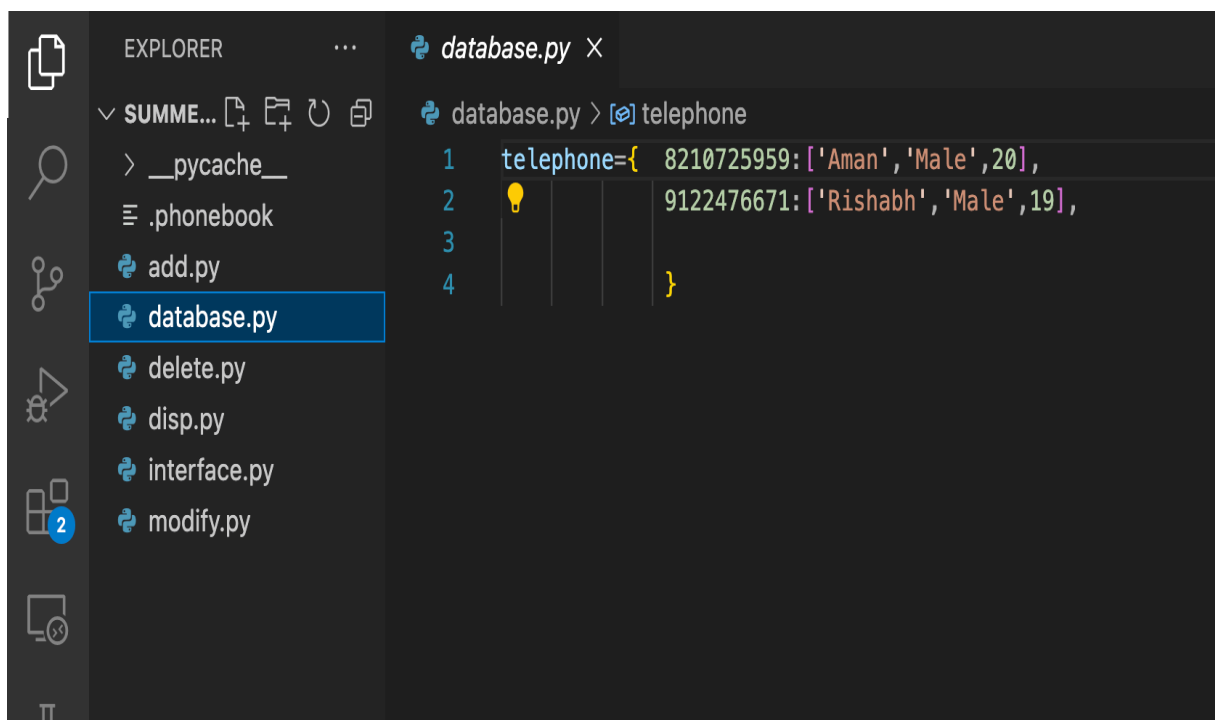
Insertion of contacts



The screenshot shows the Visual Studio Code interface. In the Explorer on the left, the file `add.py` is selected under a folder named `.phonebook`. The main editor displays the code for `add.py`:

```
1 from database import telephone as items
2 def add_items():
3     code=int(input("\nEnter phone number : "))
4     if code not in items.keys():
5         name=input("Enter new name : ")
6         gender=input("Enter gender of person : ")
7         age=int(input("Enter age of person : "))
8         items.update({code:[name,gender,age]})
9     else:
10        print("Number already exists")
11
```

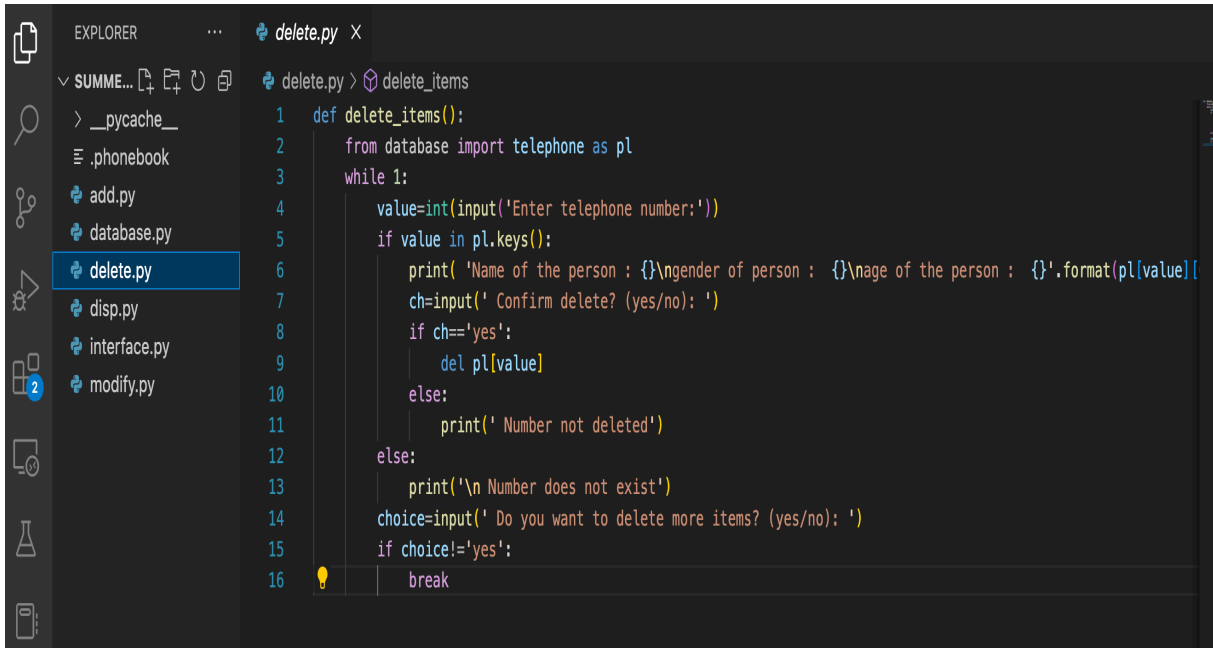
Database



The screenshot shows the Visual Studio Code interface. In the Explorer on the left, the file `database.py` is selected under the same `.phonebook` folder. The main editor displays the code for `database.py`:

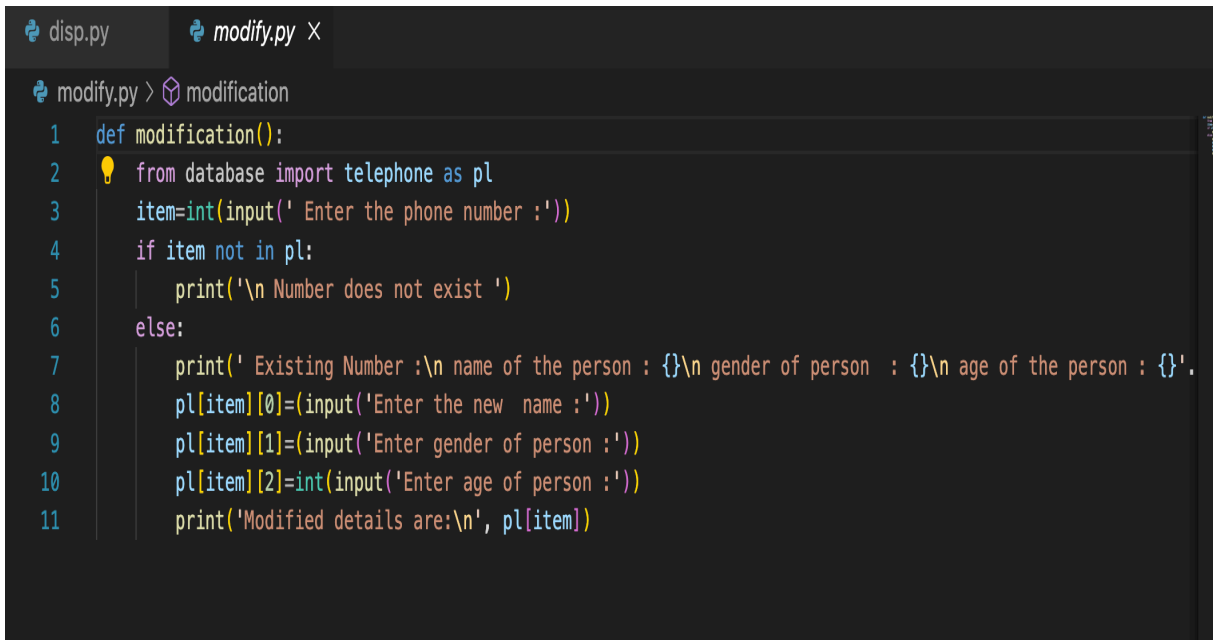
```
1 telephone={ 8210725959:['Aman','Male',20],
2             9122476671:['Rishabh','Male',19],
3             }
4
```

Delete



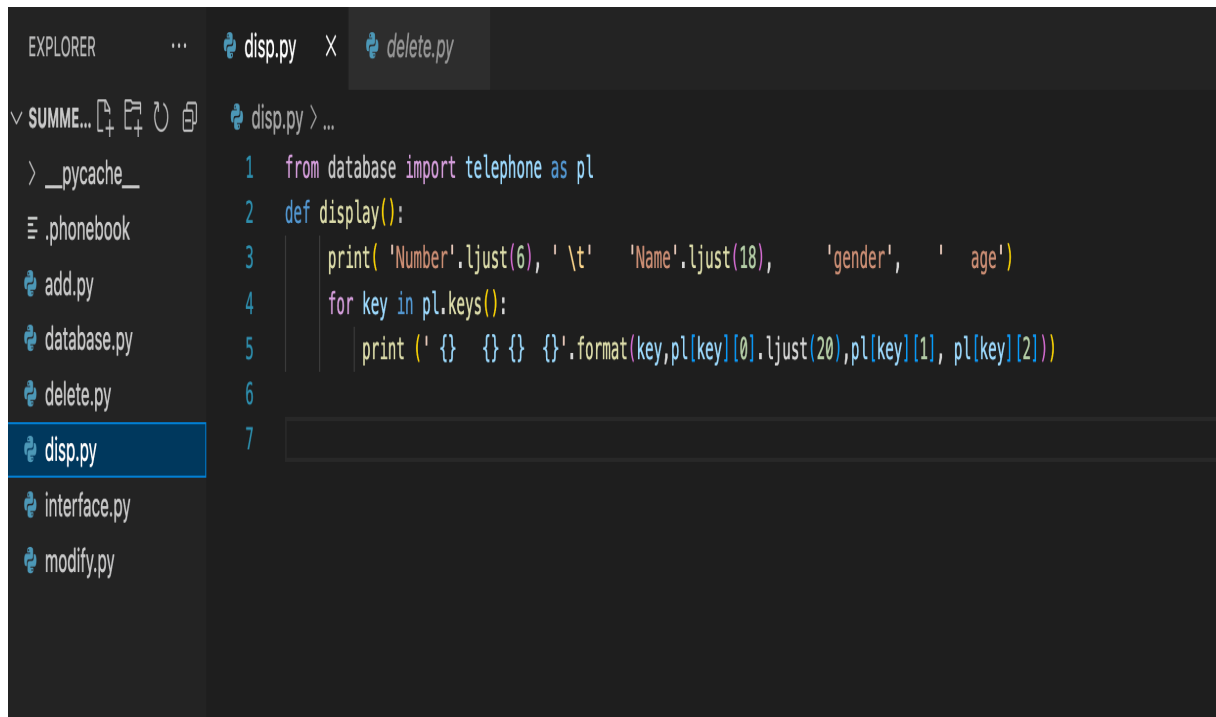
```
1 def delete_items():
2     from database import telephone as pl
3     while 1:
4         value=int(input('Enter telephone number:'))
5         if value in pl.keys():
6             print( 'Name of the person : {} \ngender of person : {} \nage of the person : {}'.format(pl[value][0], pl[value][1], pl[value][2]))
7             ch=input(' Confirm delete? (yes/no): ')
8             if ch=='yes':
9                 del pl[value]
10            else:
11                print(' Number not deleted')
12        else:
13            print('\n Number does not exist')
14        choice=input(' Do you want to delete more items? (yes/no): ')
15        if choice!='yes':
16            break
```

Modify



```
1 def modification():
2     from database import telephone as pl
3     item=int(input(' Enter the phone number :'))
4     if item not in pl:
5         print('\n Number does not exist ')
6     else:
7         print(' Existing Number :\n name of the person : {} \n gender of person : {} \n age of the person : {}'.format(pl[item][0], pl[item][1], pl[item][2]))
8         pl[item][0]=(input('Enter the new name :'))
9         pl[item][1]=(input('Enter gender of person :'))
10        pl[item][2]=int(input('Enter age of person :'))
11        print('Modified details are:\n', pl[item])
```

Display



```
1 from database import telephone as pl
2 def display():
3     print( 'Number'.ljust(6), '\t'   'Name'.ljust(18),    'gender',    ' age')
4     for key in pl.keys():
5         print ( ' {}    {} {} {}'.format(key,pl[key][0].ljust(20),pl[key][1], pl[key][2]))
6
7
```

CODE

Add.py

```
from database import telephone
def add_items():
    code=int(input("\nEnter phone number : "))
    if code not in telephone.keys():
        name=input("Enter new name : ")
        gender=input("Enter gender of person : ")
    )
```

```

        age=int(input("Enter age of person :
"))
telephone.update({code: [name,gender,age]})
    else:
        print("Number already exists")

```

delete.py

```

def delete_items():
    from database import telephone as pl
    while 1:
        value=int(input('Enter telephone
number:'))
        if value in pl.keys():
            print( 'Name of the person :
{}\ngender of person :  {}\nage of the person :
{}'.format(pl[value][0],pl[value][1],pl[value][
2]))
            ch=input(' Confirm delete?
(yes/no): ')
            if ch=='yes':
                del pl[value]
            else:
                print(' Number not deleted')
        else:
            print('\n Number does not exist')
            choice=input(' Do you want to delete
more items? (yes/no): ')
            if choice!='yes':
                break

```

disp.py

```
from database import telephone as pl
def display():
    print( 'Number'.ljust(6), ' \t'
'Name'.ljust(18), 'gender', ' age')
    for key in pl.keys():
        print (' {} {} {}
{}'.format(key,pl[key][0].ljust(20),pl[key][1],
pl[key][2]))
```

modify.py

```
def modification():
    from database import telephone as pl
    item=int(input(' Enter the phone number
:'))
    if item not in pl:
        print('\n Number does not exist ')
    else:
        print(' Existing Number :\n name of the
person : {}\n gender of person : {}\n age of
the person :
{}'.format(pl[item][0].ljust(20),pl[item][1],pl
[item][2]))
        pl[item][0]=(input('Enter the new name
:'))
        pl[item][1]=(input('Enter gender of
person :'))
        pl[item][2]=int(input('Enter age of
person :'))
```

```
        print('Modified details are:\n',  
pl[item])
```

interface.py

```
from modify import modification as m  
from add import add_items as a  
from delete import delete_items as d  
from disp import display as l  
  
while 1:  
    print('\n\n\n      PHONEBOOK')  
    print('      ----- \n')  
    print('\n MENU :-')  
    print('\n 1. Display Number list \n 2. Add  
New numbers \n 3. Modify person details \n 4.  
Delete number with their details \n 5. Exit')  
    ch=int(input(' \n Enter your choice : '))  
    if ch==1:  
        l()  
    elif ch==2:  
        a()  
    elif ch==3:  
        m()  
    elif ch==4:  
        d()  
    elif ch==5:  
        break  
    else:  
        print(' \n wrong choice !!')
```


SCREENSHOTS OF RESULT

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  JUPYTER  TERMINAL

python -u "/Users/amanraj/Desktop/Summer/SUMMER_TRAINING_PROJECT/interface.py"
○ (base) amanraj@Amans-MacBook-Air SUMMER_TRAINING_PROJECT % python -u "/Users/amanraj/Desktop/Summer/SUMMER_TRAINING_PROJECT/interface.py"

PHONEBOOK
-----

MENU :-

1. Display Number list
2. Add New numbers
3. Modify person details
4. Delete number with their details
5. Exit

Enter your choice : 2

Enter phone number : 8210845632
Enter new name : Sid
Enter gender of person : Male
Enter age of person : 19

PHONEBOOK
-----

MENU :-

1. Display Number list
2. Add New numbers
3. Modify person details
4. Delete number with their details
5. Exit
```

PHONEBOOK

MENU :-

1. Display Number list
2. Add New numbers
3. Modify person details
4. Delete number with their details
5. Exit

Enter your choice : 1

Number	Name	gender	age
8210725959	Aman	Male	20
9122476671	Rishabh	Male	19
8210845632	Sid	Male	19

PHONEBOOK

MENU :-

1. Display Number list
2. Add New numbers
3. Modify person details
4. Delete number with their details
5. Exit

Enter your choice : █

REFERENCES

- **GEEKSFORGEEKS website**
- **Google.com**
- **Sample report**

Thank
you