

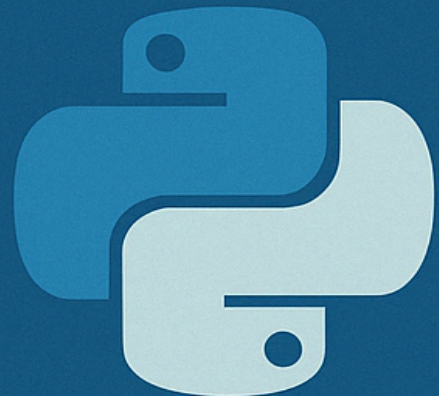
KARNATAKA SECOND PUC

COMPUTER SCIENCE

STUDY MATERIAL

MOHAMMED MATHEEN L R

PYTHON
& MYSQL



Contents

LICENSE	8
CHAPTER 1: EXCEPTION HANDLING IN PYTHON	9
CHAPTER SYNOPSIS	9
Introduction	9
Syntax Errors	9
Exceptions	10
Built-in Exceptions	10
Raising Exceptions	12
The raise Statement	12
The assert Statement	12
Handling Exceptions	13
The try...except Block	15
The try...except...else Clause	16
The finally clause	17
Recovering and continuing with finally clause	17
CHAPTER 2: FILE HANDLING IN PYTHON	19
Introduction to Files	19
Types of Files	19
Text Files	19
Binary Files	19
Opening and Closing a File	19
Opening a File	19
File Access Modes with Examples	20
File Attributes	21
Closing a File	21
Opening a File Using with Clause	21
File Access Modes	22
File Attributes	22
Closing a File	22
Opening a File Using with Clause	22
Writing to a Text File	23
Syntax: write()	23
Example: write()	23
Syntax: writelines()	23

Example: writelines()	23
Reading from a Text File	24
Syntax: read(n)	24
Example: read(n)	24
Syntax: read()	24
Example: read()	24
Syntax: readline(n)	24
Example: readline(n)	24
Syntax: readlines()	25
Example: readlines()	25
Complete Example from Text	25
File Offset Methods	26
Syntax: tell()	26
Syntax: seek(offset, reference_point)	26
Creating and Traversing a Text File	26
Creating File and Writing Data	26
Displaying File Contents	27
Combined Read/Write Program	27
Pickle Module	27
Pickling	27
Unpickling	27
Import Statement	28
Syntax for dump()	28
Syntax for load()	28
Example for dump()	28
Example for load()	28
Complete Program Example	28
CHAPTER 3: STACKS	30
CHAPTER NOTES	30
3.1 Introduction	30
3.2 Stack	30
3.2.1 Applications of Stack	30
3.3 Operations on Stack	31
3.4 Implementation of Stack in Python	31
3.5 Notations for Arithmetic Expressions	33
3.6 Conversion from Infix to Postfix	33
3.7 Evaluation of Postfix Expression	34

Summary Points	34
CHAPTER 4: QUEUE	35
CHAPTER NOTES	35
4.1 Introduction to Queue	35
4.1.1 FIFO Principle	35
Applications of Queue	35
4.2 Operations on Queue	36
4.3 Implementation of Queue using Python	37
4.4 Introduction to Deque (Double-Ended Queue)	38
Applications of Deque (Double-Ended Queue)	38
4.4.2 Operations on Deque	39
Algorithm 4.1: Palindrome using Deque	39
4.5 Implementation of Deque using Python	39
Summary Points	41
CHAPTER 5: SORTING	42
CHAPTER NOTES	42
5.1 Introduction	42
5.2 Bubble Sort	42
5.3 Selection Sort	44
5.4 Insertion Sort	45
5.5 Time Complexity of Algorithms	46
Summary Points	47
CHAPTER 6: SEARCHING	48
CHAPTER NOTES	48
6.1 Introduction	48
6.2 Linear Search (Sequential Search)	48
Algorithm 6.1: Linear Search	48
Example	48
6.3 Binary Search	49
Algorithm 6.2: Binary Search	49
Example	50
Program 6-2: Binary Search in Python	50
6.3.1 Applications of Binary Search	51
6.4 Search by Hashing	51
Example	51
Program 6-3: Hashing in Python	52

6.4.1 Collision	53
Summary	53
CHAPTER 7: UNDERSTANDING DATA	54
CHAPTER NOTES	54
7.1 Introduction to Data	54
7.1.1 Importance of Data	54
7.1.2 Types of Data	55
7.2 Data Collection	55
7.3 Data Storage	56
7.4 Data Processing	56
Figure 7.2: Real-Life Data Processing Examples	58
7.5 Statistical Techniques for Data Processing	58
Glossary of Key Terms	60
CHAPTER 8 DATABASE CONCEPTS	62
CHAPTER NOTES	62
8.1 Introduction	62
8.2 File System	62
8.3 Database Management System (DBMS)	64
8.4 Relational Data Model	68
8.5 Keys in a Relational Database	69
Summary of Chapter 8	71
Summary of Key Terms	71
CHAPTER 9 STRUCTURED QUERY LANGUAGE (SQL)	73
CHAPTER NOTES	73
Introduction	73
9.2 Structured Query Language (SQL)	73
9.2.1 Installing MySQL	74
9.3 Data Types and Constraints in MySQL	75
9.4 SQL for Data Definition	77
9.4.4 — ALTER TABLE	79
9.4.5 — DROP Statement	83
9.5 — SQL for Data Manipulation	84
9.6 — SQL for Data Query	87
9.6.2 — Querying Using Database OFFICE	88
9.7 — Data Updation and Deletion	101
9.8: Functions in SQL	103

9.9 — GROUP BY Clause in SQL	107
9.10 – Operations on Relations	110
CHAPTER 10 COMPUTER NETWORKS	114
CHAPTER NOTES	114
10.1: Introduction to Computer Networks	114
10.2: Evolution of Networking	115
10.3: Types of Networks	116
10.3.1 Personal Area Network (PAN)	116
10.4: Network Devices	117
10.5: Networking Topologies	118
What is a Topology?	118
10.6: Identifying Nodes in a Networked Communication	120
10.7: Internet, Web, and the Internet of Things (IoT)	121
10.8: Domain Name System (DNS)	122
CHAPTER 11 DATA COMMUNICATION	124
CHAPTER NOTES	124
11.1 Concept of Communication	124
11.2: Components of Data Communication	125
11.3: Measuring Capacity of Communication Media	126
11.4: Types of Data Communication	127
11.5: Switching Techniques	128
11.6: Transmission Media	130
11.6.1 Wired (Guided) Transmission Media	130
11.6.2 Wireless (Unguided) Transmission Media	131
11.6.3: Wireless Technologies	133
11.7: Mobile Telecommunication Technologies	134
11.8: Protocol	136
CHAPTER 12 SECURITY ASPECTS	139
CHAPTER NOTES	139
12.1 - Threats and Prevention	139
12.2 - Malware	140
12.3 - Antivirus	143
12.4 - Spam	144
12.5 - HTTP vs HTTPS	144
12.6 - Firewall	145
12.7 - Cookies	146

12.8 - Hackers and Crackers	147
12.9 - Network Security Threats	147

MOHAMMED MATHEEN L R

LICENSE

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

“Karnataka Second PUC Computer Science Study Material / Student Notes” by L R Mohammed Matheen is licensed under CC BY-NC-ND 4.0.



Figure 1: Licence

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International License.

Portions of this work may include material under separate copyright. These materials are not covered by this Creative Commons license and are used by permission or under applicable copyright exceptions.

This book is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International License.

CHAPTER 1: EXCEPTION HANDLING IN PYTHON

CHAPTER SYNOPSIS

Introduction

When there are syntax errors, runtime errors or logical errors in the code, program does not execute at all or the program executes but generates unexpected output or behaves abnormally.

Exception handling in Python allows a programmer to deal with runtime errors, ensuring that the program continues to operate in the presence of unexpected conditions. This process involves identifying, catching, and handling exceptions to maintain the normal flow of the program.

Syntax Errors

Definition Syntax errors occur when the code written does not conform to the rules of the Python language. These errors are detected during the parsing stage and must be fixed before the program can run. Syntax errors are also called *parsing errors*.

Example:

```
marks = 10
if marks>20:
    print "GOOD SCORE!"
```

This code will produce a `SyntaxError` because of the missing closing quote.

Syntax error in interactive/shell mode:

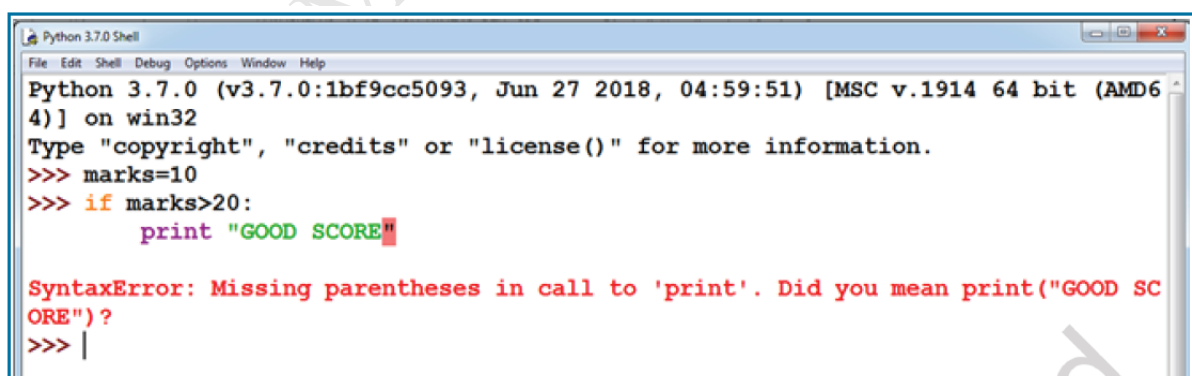
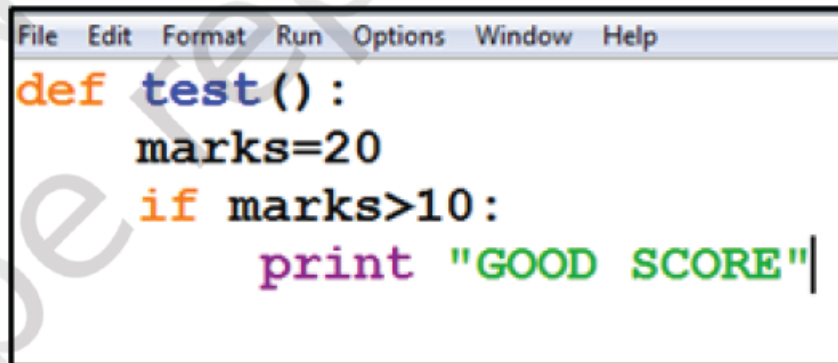


Figure 2: Syntax error in Shell mode

Syntax error detected in script mode:



```
def test():
    marks=20
    if marks>10:
        print 'GOOD SCORE'|
```

Figure 1.2: An error in the script

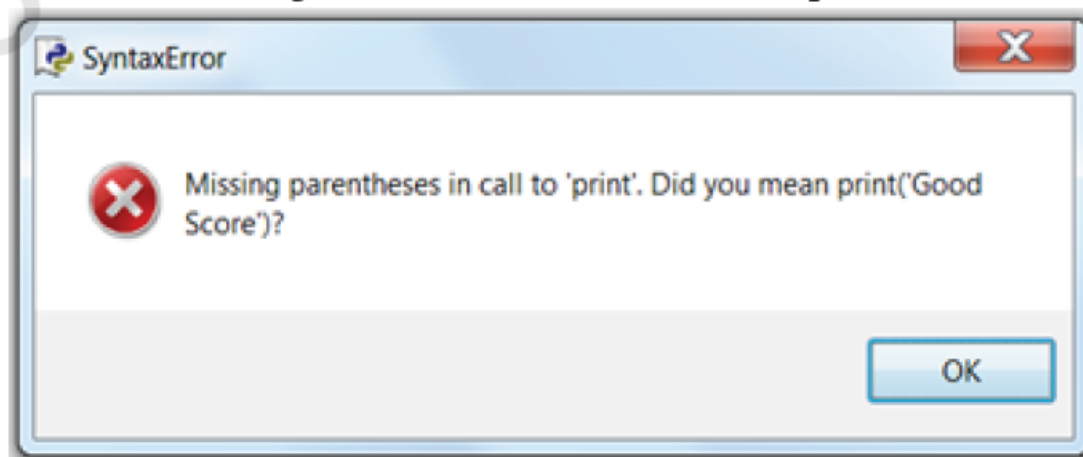


Figure 3: Syntax Error in Script mode

Exceptions

Definition These are errors that occur during the execution of the program. Unlike syntax errors, exceptions are detected during runtime and can be handled using exception handlers.. Examples include trying to open a file that does not exist or dividing a number by zero.

Example:

```
result = 10 / 0
```

Built-in Exceptions

Commonly occurring exceptions are usually defined in the compiler/interpreter. These are called *built-in exceptions*.

Common Built-in Exceptions Some common built-in exceptions in Python include:

Exception	Description
SyntaxError	Raised when there is an error in the syntax of the code.
ValueError	Raised when a built-in method or operation receives an argument with the right data type but an inappropriate value.
IOError	Raised when an input/output operation fails.
KeyboardInterrupt	Raised when the user interrupts the program's execution (usually by pressing Ctrl+C).
ImportError	Raised when an import statement fails to find the module definition.
EOFError	Raised when the <code>input()</code> function hits an end-of-file condition.
ZeroDivisionError	Raised when the second argument of a division or modulo operation is zero.
IndexError	Raised when an index is out of range.
NameError	Raised when a local or global name is not found.
IndentationError	Raised when there is incorrect indentation.
TypeError	Raised when an operation is applied to an object of inappropriate type.
OverflowError	Raised when a calculation exceeds the maximum limit for a numeric data type.

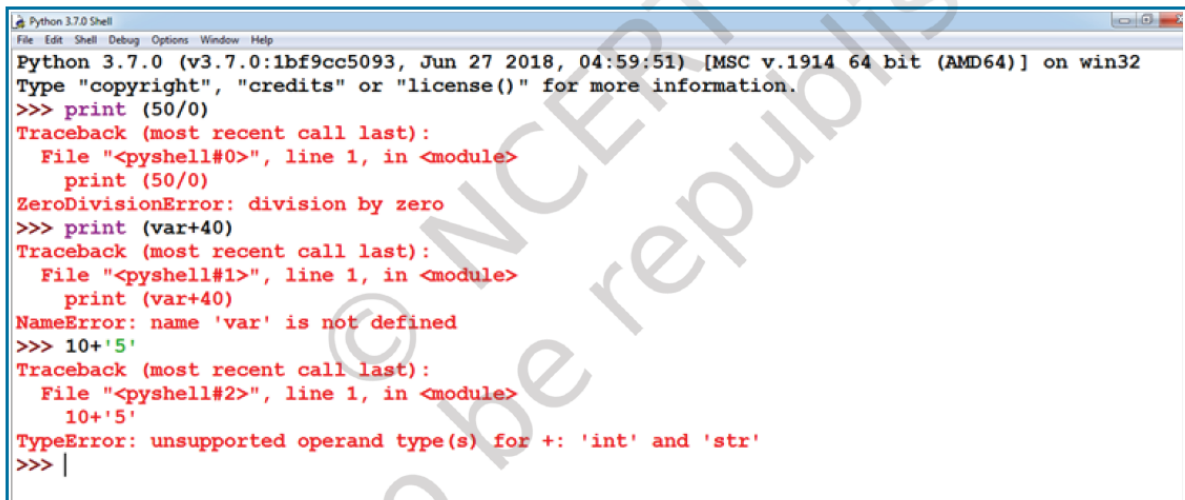
Examples:

```
# Example of NameError
print(var + 40)

# Example of IndexError
lst = [1, 2, 3]
print(lst[5])

# Example of TypeError
print(10 + '5')
```

Built-in Exceptions:



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print (50/0)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print (50/0)
ZeroDivisionError: division by zero
>>> print (var+40)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print (var+40)
NameError: name 'var' is not defined
>>> 10+'5'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    10+'5'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> |
```

A programmer can also create custom exceptions to suit one's requirements. These are called *user-defined exceptions*.

Raising Exceptions

We can forcefully raise an exception using the `raise` or `assert` statement.

The raise Statement

The syntax of raise statement is:

```
raise exception-name[(optional argument)]
raise ValueError("This is a forced exception")
```

Example:

```
numbers = [40,50,60,70]
length = 10
if length>len(numbers):
    raise IndexError
    print ("No Execution")
else:
    print(length)
```

The assert Statement

The `assert` statement is used to test expressions and raise an `AssertionError` if the expression is false. This statement is generally used in the beginning of the function or after a function call to check for

valid input. It is commonly used for debugging purposes. The syntax for assert statement is: `assert Expression[,arguments]`

Example:

```
print("use of assert statement")
def negativecheck(number):
    assert(number>=0), "OOPS... Negative Number"
    print(number*number)
```

```
negativecheck(100)
negativecheck(-350)
```

Handling Exceptions

Exception Handling

Exception handling refers to the process of responding to the occurrence of exceptions during the execution of a program. This is typically done using try, except, finally, and else blocks in Python.

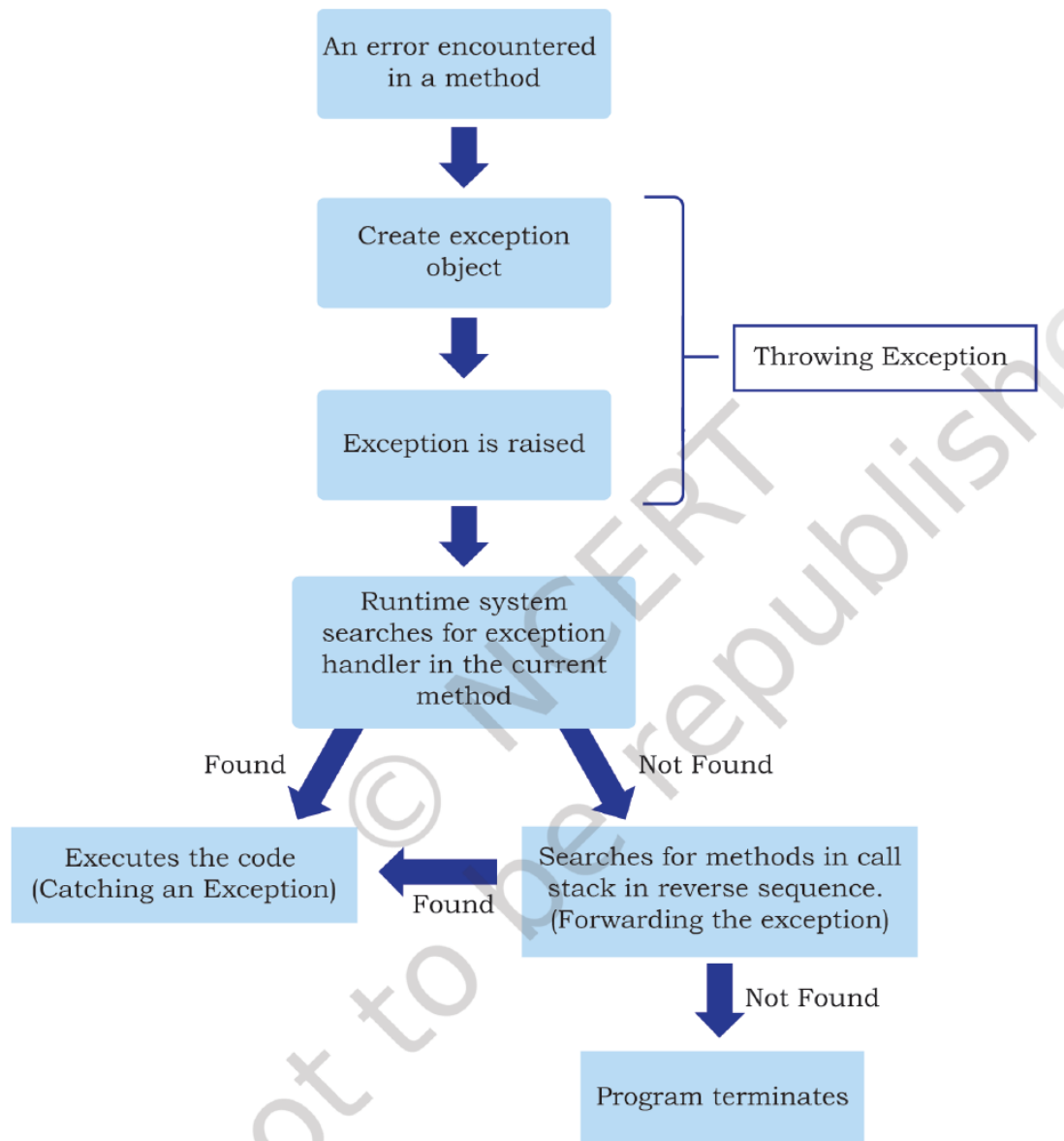
Need for Exception Handling: - Essential to prevent program crashes by capturing and managing runtime errors.

- Separates main program logic from error detection and correction code.
- The compiler/interpreter tracks the exact error location.
- Applicable to both user-defined and built-in exceptions.

Process of Handling Exception

- When an error occurs, Python interpreter creates an object called the *exception object*.
- This object contains information about the error like its type, file name, and position in the program where the error has occurred.
- The object is handed over to the runtime system to find an appropriate code to handle this particular exception.
- This process of creating an exception object and handing it over to the runtime system is called *throwing an exception*.
- When an exception occurs while executing a particular program statement, the control jumps to an exception handler, abandoning the execution of the remaining program statements.

- The runtime system searches the entire program for a block of code, called the *exception handler*, that can handle the raised exception.
- The runtime system first searches the method in which the error occurred and the exception was raised. If not found, it searches the method from which this method was called.
- This hierarchical search in reverse order continues until the exception handler is found.
- This entire list of methods is known as the *call stack*.
- When a suitable handler is found in the call stack, it is executed by the runtime process.
- The process of executing a suitable handler is known as *catching the exception*.
- If the runtime system cannot find an appropriate exception after searching all the methods in the call stack, then the program execution stops.



The try...except Block

The try...except block is used to handle exceptions in Python. Code that might raise an exception is placed inside the try block, and the handling of the exception is done in the except block.

Syntax:

```
try:
    # Code that might raise an exception
```

```
except ExceptionType:
    # Code to handle the exception
```

Example:

```
print("Practicing for try block")
try:
    numerator = 50
    denom = int(input("Enter the denominator: "))
    quotient = (numerator / denom)
    print(quotient)
    print("Division performed successfully")
except ZeroDivisionError:
    print("Denominator as ZERO.... not allowed")
print("OUTSIDE try..except block")
```

Multiple except Blocks Handling different exceptions separately.

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError as e:
    print("Error: Division by zero -",e)
except ValueError as e:
    print("Error: Invalid input - ",e)
```

Catching All Exceptions Using a generic except clause.

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except Exception as e:
    print(f"An error occurred: {e}")
```

The try...except...else Clause

We can put an additional else clause to the normal try...except. Code inside the else block will run if no exceptions are raised in the try block.

```
print("Handling exception using try...except...else...finally")
try:
    numerator = 50
    denom = int(input("Enter the denominator: "))
    quotient = (numerator / denom)
```

```
    print("Division performed successfully")
except ZeroDivisionError:
    print("Denominator as ZERO is not allowed")
except ValueError:
    print("Only INTEGERS should be entered")
else:
    print("The result of the division operation is", quotient)
```

The finally clause

The finally clause is an optional part of the try statement in Python. The code inside the finally block is always executed, regardless of whether an exception occurred or not. It is typically used for cleanup actions, like closing files or releasing resources.

Example:

```
print("Handling exception using try...except...else...finally")
try:
    numerator = 50
    denom = int(input("Enter the denominator: "))
    quotient = (numerator / denom)
    print("Division performed successfully")
except ZeroDivisionError:
    print("Denominator as ZERO is not allowed")
except ValueError:
    print("Only INTEGERS should be entered")
else:
    print("The result of the division operation is", quotient)
finally:
    print("OVER AND OUT")
```

In this example, the message “OVER AND OUT” will be displayed regardless of whether an exception is raised or not.

Recovering and continuing with finally clause

If an error has been detected in the try block and the exception has been thrown, the appropriate except block will be executed to handle the error. But if the exception is not handled by any of the except clauses, then it is re-raised after the execution of the finally block.

Example:

```
print("Handling exception using try...except...else...finally")
try:
    numerator = 50
    denom = int(input("Enter the denominator: "))
    quotient = (numerator / denom)
    print("Division performed successfully")
except ZeroDivisionError:
    print("Denominator as ZERO is not allowed")
else:
    print("The result of the division operation is", quotient)
finally:
    print("OVER AND OUT")
```

MOHAMMED MATHEEN L R

CHAPTER 2: FILE HANDLING IN PYTHON

Introduction to Files

- In Python, data stored in variables is temporary and disappears after the program ends.
- To store data permanently, we use **files**, which are saved on secondary storage (e.g., hard disks).
- Files allow us to store inputs, outputs, and objects for later use.
- A Python file (source code) is saved with a `.py` extension.

Types of Files

Files can be classified into two major types:

Text Files

- Human-readable and stored using characters like letters, digits, symbols.
- Examples: `.txt`, `.py`, `.csv`, etc.
- Stored using encoding formats like **ASCII**, **Unicode**.
- Each character is represented by its byte equivalent.
- **End of Line (EOL)** character (`\n`) is used to separate lines.

Example:

The ASCII value of 'A' is 65 → Binary: 1000001

Binary Files

- Not human-readable (appear as symbols or garbage in text editors).
- Store data such as images, audio, video, compressed or executable files.
- Readable only with appropriate programs.

Opening and Closing a File

Opening a File

- Files are opened using the `open()` function which returns a file object.

Syntax:

```
file_object = open("filename", "mode")
```

File Access Modes with Examples**Read Mode - r**

- Opens file for reading only.
- File must exist.
- File pointer at the beginning.

```
f = open("myfile.txt", "r")
```

Binary Read Mode - rb

- Opens file in binary mode for reading.

```
f = open("myfile.txt", "rb")
```

Read and Write Mode - r+

- Opens file for reading and writing.
- File must exist.

```
f = open("myfile.txt", "r+")
```

Write Mode - w

- Opens file for writing only.
- Overwrites file if it exists or creates a new one.

```
f = open("myfile.txt", "w")
```

Write Binary and Read Mode - wb+

- Opens file in binary mode for writing and reading.
- Overwrites or creates a new file.

```
f = open("myfile.txt", "wb+")
```


Append Mode - a

- Opens file for appending data.
- Creates file if it doesn't exist.
- File pointer at the end.

```
f = open("myfile.txt", "a")
```

Append and Read Mode - a+

- Opens file for both appending and reading.

```
f = open("myfile.txt", "a+")
```

Example from textbook:

```
myObject = open("myfile.txt", "a+")
```

File Attributes

```
file.closed    # True if file is closed  
file.mode      # Access mode of file  
file.name      # Name of the file
```

Closing a File

- Always close files after use to free system resources. **Syntax:**

```
file_object.close()
```

Opening a File Using with Clause

- Preferred method for file operations as it auto-closes the file. **Syntax:**

```
with open("myfile.txt", "r+") as myObject:  
    content = myObject.read()
```

Opening a File Syntax:

```
file_object = open("filename", "mode")
```

File Access Modes

Mode	Description	File Pointer Position
r	Read only	Beginning
rb	Read binary	Beginning
r+	Read and write	Beginning
w	Write only (overwrites if exists)	Beginning
wb+	Write and read binary (overwrites)	Beginning
a	Append (creates file if it doesn't exist)	End
a+	Append and read	End

Example:

```
myObject = open("myfile.txt", "a+")
```

File Attributes

```
file.closed    # True if file is closed
file.mode      # Access mode of file
file.name      # Name of the file
```

Closing a File

Syntax:

```
file_object.close()
```

Opening a File Using with Clause

Syntax:

```
with open("myfile.txt", "r+") as myObject:
    content = myObject.read()
```

- Automatically closes the file when the block is exited.

Writing to a Text File

Syntax: write()

```
file_object.write(string)
```

- Writes a single string to the file.
- Returns the number of characters written.

Example: write()

- Writes a single string to the file.

```
myobject = open("myfile.txt", 'w')
myobject.write("Hey I have started using files in Python\n")
myobject.close()
```

Writing numbers:

```
marks = 58
myobject.write(str(marks))
```

Syntax: writelines()

```
file_object.writelines(list_of_strings)
```

- Writes multiple strings to the file.
- Takes an iterable like a list or tuple.

Example: writelines()

- Writes a sequence (like a list or tuple) of strings.

```
lines = ["Hello everyone\n", "Writing multiline strings\n", "This is the third line"]
myobject.writelines(lines)
```

Reading from a Text File

Syntax: read(n)

```
file_object.read(n)
```

- Reads n bytes from the file.

Example: read(n)

- Reads n bytes from the file.

```
myobject = open("myfile.txt", 'r')
print(myobject.read(10))
myobject.close()
```

Syntax: read()

```
file_object.read()
```

- Reads the entire content of the file.

Example: read()

- Reads the entire file content.

```
myobject = open("myfile.txt", 'r')
print(myobject.read())
myobject.close()
```

Syntax: readline(n)

```
file_object.readline(n)
```

- Reads one line or up to n bytes until newline character.

Example: readline(n)

- Reads one line or n bytes until newline.

```
myobject = open("myfile.txt", 'r')
print(myobject.readline(10))
myobject.close()
```

Syntax: readlines()

```
file_object.readlines()
```

- Reads all lines and returns them as a list.

Example: readlines()

- Reads all lines and returns a list.

```
myobject = open("myfile.txt", 'r')
print(myobject.readlines())
myobject.close()
```

Splitting into words:

```
for line in lines:
    words = line.split()
    print(words)
```

Using splitlines():

```
for line in lines:
    print(line.splitlines())
```

Complete Example from Text

```
fobject = open("testfile.txt", "w")
sentence = input("Enter the contents to be written in the file: ")
fobject.write(sentence)
fobject.close()

print("Now reading the contents of the file: ")
fobject = open("testfile.txt", "r")
for str in fobject:
    print(str)
fobject.close()
```

File Offset Methods

Syntax: tell()

```
file_object.tell()
```

- Returns current position of file pointer.
- Returns current file pointer position.

```
file_object.tell()
```

Syntax: seek(offset, reference_point)

```
file_object.seek(offset, reference_point)
```

- Moves pointer to specified position from reference point (0 = start, 1 = current, 2 = end).
- Moves pointer to specified position.

```
file_object.seek(5, 0) # Move to 5th byte from start
```

Program Example:

```
fileobject = open("testfile.txt", "r+")
str = fileobject.read()
print(str)
print("Initially, position:", fileobject.tell())
fileobject.seek(0)
print("Now at beginning:", fileobject.tell())
fileobject.seek(10)
print("Pointer at:", fileobject.tell())
print(fileobject.read())
fileobject.close()
```

Creating and Traversing a Text File

Creating File and Writing Data

```
fileobject = open("practice.txt", "w+")
while True:
    data = input("Enter data to save in the text file: ")
```



```
fileobject.write(data)
ans = input("Do you wish to enter more data?(y/n): ")
if ans == 'n': break
fileobject.close()
```

Displaying File Contents

```
fileobject = open("practice.txt", "r")
str = fileobject.readline()
while str:
    print(str)
    str = fileobject.readline()
fileobject.close()
```

Combined Read/Write Program

```
fileobject = open("report.txt", "w+")
while True:
    line = input("Enter a sentence ")
    fileobject.write(line + "\n")
    choice = input("Do you wish to enter more data? (y/n): ")
    if choice.lower() == 'n': break

print("File position:", fileobject.tell())
fileobject.seek(0)
print("Reading contents:")
print(fileobject.read())
fileobject.close()
```

Pickle Module

Pickling

- Process of converting Python objects to a byte stream (serialization).
- Can store complex objects like lists, dictionaries.

Unpickling

- Restores the byte stream back into original object (deserialization).

Import Statement

```
import pickle
```

Syntax for dump()

```
pickle.dump(object, file_object)
```

Syntax for load()

```
variable = pickle.load(file_object)
```

Example for dump()

```
listvalues = [1, "Geetika", 'F', 26]
fileobject = open("mybinary.dat", "wb")
pickle.dump(listvalues, fileobject)
fileobject.close()
```

Example for load()

```
fileobject = open("mybinary.dat", "rb")
objectvar = pickle.load(fileobject)
fileobject.close()
print(objectvar)
```

Complete Program Example

```
import pickle
bfile = open("empfile.dat", "ab")
recno = 1
while True:
    eno = int(input("Employee number: "))
    ename = input("Name: ")
    ebasic = int(input("Basic Salary: "))
    allow = int(input("Allowances: "))
    totalsal = ebasic + allow
    edata = [eno, ename, ebasic, allow, totalsal]
    pickle.dump(edata, bfile)
    if input("More records? (y/n): ").lower() == 'n': break
```

```
print("File size:", bfile.tell())
bfile.close()

print("Reading employee records")
try:
    with open("empfile.dat", "rb") as bfile:
        while True:
            edata = pickle.load(bfile)
            print(edata)
except EOFError:
    pass
```

MOHAMMED MATHEEN L R

CHAPTER 3: STACKS

CHAPTER NOTES

3.1 Introduction

- **Data Structures:** Ways to store, organize, and access data efficiently.
 - Examples: String, List, Set, Tuple, Array, Linked List, Stack, Queue, Trees, Graphs, etc.
 - Stack and Queue are not built-in Python structures but can be implemented using lists.
-

3.2 Stack

- **Definition:** A linear data structure where elements are added/removed from one end (TOP).
 - **LIFO Principle:** Last In, First Out – the last element added is the first to be removed.
 - **Analogy:** Stack of plates or books – add/remove only from the top.
-

3.2.1 Applications of Stack

Real-life Scenarios

1. **Pile of clothes in an almirah** – Clothes are added or removed from the top of the pile, following the LIFO order.
2. **Multiple chairs in a vertical pile** – Chairs are stacked one on top of the other, and removed in reverse order.
3. **Bangles worn on the wrist** – The last bangle worn is the first to be removed.
4. **Boxes of eatables in pantry** – Boxes are added and removed from the top for convenience.

Programming Applications

1. String Reversal

- Characters are added to a stack and popped in reverse order to reverse the string.

2. Undo/Redo in Editors

- Each edit (text/image) is pushed onto a stack. Undo removes the last change, redo re-applies it.
-

3. Back Function in Web Browsers

- Navigated pages are pushed onto a stack. Pressing 'Back' pops the last visited page and returns to the previous one.

4. Parentheses Matching in Expressions

- During program execution, a stack is used to ensure every opening parenthesis has a matching closing one. It helps identify syntax errors like unmatched or misnested parentheses.
-

3.3 Operations on Stack

- **PUSH:** Add an element to the TOP of the stack.
 - **POP:** Remove the topmost element.
 - **Overflow:** Trying to PUSH to a full stack (not typically an issue in Python).
 - **Underflow:** Trying to POP from an empty stack.
-

3.4 Implementation of Stack in Python

Functions:

```
# Create an empty stack
glassStack = list()

# Check if stack is empty
def isEmpty(glassStack):
    return len(glassStack) == 0

# PUSH operation
def opPush(glassStack, element):
    glassStack.append(element)

# POP operation
def opPop(glassStack):
    if isEmpty(glassStack):
        print("underflow")
        return None
    else:
```

```
        return glassStack.pop()

# Return stack size
def size(glassStack):
    return len(glassStack)

# Return top element
def top(glassStack):
    if isEmpty(glassStack):
        print("Stack is empty")
        return None
    else:
        return glassStack[-1]

# Display all elements (from TOP to bottom)
def display(glassStack):
    print("Current elements in the stack are:")
    for i in range(len(glassStack)-1, -1, -1):
        print(glassStack[i])
```

Sample Program:

```
glassStack = list() # create empty stack

element = 'glass1'
print("Pushing element", element)
opPush(glassStack, element)

element = 'glass2'
print("Pushing element", element)
opPush(glassStack, element)

print("Current number of elements in stack is", size(glassStack))

element = opPop(glassStack)
print("Popped element is", element)

element = 'glass3'
print("Pushing element", element)
opPush(glassStack, element)

print("Top element is", top(glassStack))
display(glassStack)
```



```
# delete all elements
while True:
    item = opPop(glassStack)
    if item is None:
        break
    print("Popped element is", item)

print("Stack is empty now")
```

3.5 Notations for Arithmetic Expressions

Notation	Description	Example
Infix	Operator between operands	$(x + y) / (z * 5)$
Prefix	Operator before operands	$/+xy*z5$
Postfix	Operator after operands	$xy+z5*/$

- **Prefix** (Polish) and **Postfix** (Reverse Polish) do not require parentheses.
- **Infix** needs BODMAS and parentheses for clarity.

3.6 Conversion from Infix to Postfix

Why? Computers struggle with operator precedence in infix; postfix avoids this.

Algorithm 3.1: Infix to Postfix

1. Create empty string postExp and stack stack.
2. For each character in infix expression:
 - If (\rightarrow PUSH to stack
 - If) \rightarrow POP and add to postExp until (is found
 - If operator:
 - POP higher or equal precedence operators and append to postExp

- PUSH current operator
 - If operand \rightarrow Append to postExp
3. POP remaining operators to postExp.

Example: $(x + y)/(z * 8) \rightarrow xy+z8/$

3.7 Evaluation of Postfix Expression

Algorithm 3.2:

1. For each character in postfix expression:
 - If operand \rightarrow PUSH to stack
 - If operator \rightarrow POP two operands, apply operation, PUSH result
2. At the end, if one item in stack \rightarrow Result
3. Else \rightarrow Invalid expression

Example: $7\ 8\ 2\ *\ 4\ /\ + \rightarrow \text{Result: } 11$

Summary Points

- Stack = Linear structure, LIFO principle.
- Core ops: PUSH, POP; special cases: Overflow, Underflow.
- Python `list + append()` and `pop()` can simulate stack.
- Notations: **Infix**, **Prefix**, **Postfix**
- Stacks are essential in converting and evaluating arithmetic expressions.

CHAPTER 4: QUEUE

CHAPTER NOTES

4.1 Introduction to Queue

- A **Queue** is an ordered linear data structure that follows the **FIFO (First In First Out)** principle.
- Items are inserted at the **rear (tail)** and removed from the **front (head)**.
- **Real-life Examples:** Bank queues, toll booths, printer tasks, IVRS systems.

4.1.1 FIFO Principle

- **First-In-First-Out:** The element added first is the one removed first.
- Implemented using two ends:
 - **REAR (TAIL)** → for insertion
 - **FRONT (HEAD)** → for deletion

Applications of Queue

Real-Life Applications

1. Train Ticket Waiting List

- When you book a ticket and it shows **W/L1**, it means your request is added to a **queue**.
- If someone cancels their confirmed ticket, the one at the **front of the queue** (like W/L1) gets confirmed.
- This strictly follows the **First-In-First-Out (FIFO)** principle.

2. Customer Care Calls (IVRS Systems)

- When you call a customer care number, you might hear: > “Please wait while your call is transferred...”
- Your call is placed in a **queue**.
- As soon as a customer care executive is free, the **first caller** in the queue is attended.

3. Traffic Management (One-Way Road or Toll Booth)

- Vehicles on a one-way street or at a fuel pump/toll plaza form a queue.
- The **vehicle that arrived first** gets to move first.

Applications in Computer Science

1. Web Server Request Handling

- A server can handle only a limited number of simultaneous requests (say 50).
- If 1000 students try to check results at the same time, the server keeps extra requests in a **queue**.
- These requests are served **one by one** as per **FIFO**.

2. CPU Task Scheduling in Operating Systems

- When multiple programs (called **jobs**) request CPU time, only one can run at a time.
- Jobs are **queued** and the CPU executes them in the order they arrive (simple **job scheduling**).

3. Printer Queue Management

- If multiple print requests are sent to a shared printer:
 - The **first print command** sent is executed first.
 - Remaining ones wait in the **print queue**.

4.2 Operations on Queue

Operation	Description
enqueue()	Insert an element at rear. Exception: Overflow if full (static case).
dequeue()	Remove from front. Exception: Underflow if empty.
isEmpty()	Check if queue is empty.
peek()	View the front item without removing.
isFull()	Check if queue is full (not required in Python list-based queue).

Visual Example (Queue of alphabets):

```
enqueue('Z') → Z
enqueue('X') → Z X
enqueue('C') → Z X C
dequeue()    → X C
enqueue('V') → X C V
```

4.3 Implementation of Queue using Python

```
myQueue = list()

def enqueue(myQueue, element):
    myQueue.append(element)

def isEmpty(myQueue):
    return len(myQueue) == 0

def dequeue(myQueue):
    if not isEmpty(myQueue):
        return myQueue.pop(0)
    else:
        print("Queue is empty")

def size(myQueue):
    return len(myQueue)

def peek(myQueue):
    if isEmpty(myQueue):
        print("Queue is empty")
        return None
    else:
        return myQueue[0]
```

Program 4-1: Simulation of a Queue in a Bank

```
myQueue = list()

element = input("Enter person's code to enter in queue: ")
enqueue(myQueue, element)
element = input("Enter person's code for insertion in queue: ")
enqueue(myQueue, element)

print("Person removed from queue is:", dequeue(myQueue))
print("Number of people in the queue is:", size(myQueue))

for _ in range(3):
    element = input("Enter person's code to enter in queue: ")
    enqueue(myQueue, element)

print("Now removing remaining people from queue:")
```

```
while not isEmpty(myQueue):  
    print("Person removed from queue is", dequeue(myQueue))
```

4.4 Introduction to Deque (Double-Ended Queue)

- **Deque** allows **insertion and deletion** from **both ends** – front and rear.
- Can behave as both:
 - **Stack** (LIFO): insertion/deletion at same end.
 - **Queue** (FIFO): insertion/deletion at opposite ends.

Applications of Deque (Double-Ended Queue)

Real-Life Applications

1. Re-entry at Ticket Counter

- A person buys a ticket, leaves, and then comes back with a query.
- Since they've already been served, they might get the privilege to **rejoin from the front**, not the rear.
- This needs a data structure where you can insert from **either end** — a **deque**.

2. Toll Booth Queue Redirection

- Suppose multiple toll booths exist.
- If one booth clears early, vehicles from the rear of other queues might **shift to the front** of the vacant booth's queue.
- This needs both **front and rear insertions/deletions**, which is **deque-like behavior**.

Applications in Computer Science

1. Browser History Navigation

- URLs are added to a **stack-like structure**, but if the memory is limited, older entries (from the rear) must be removed.
- Deque can store history and remove **least-recently-used** entries when full.

2. Undo/Redo in Text Editors

- Operations are stored so that the **last action can be undone first**, like a **stack**.
- But to manage multiple undo/redo and limited memory, **deque** provides efficient handling.

3. Palindrome Checking

- A **string** is inserted character by character into a deque.
- Then characters are compared from **front and rear** simultaneously.
- If they match all the way, the string is a **palindrome**.
- Deque allows accessing **both ends** easily.

4.4.2 Operations on Deque

Operation	Description
insertFront()	Insert at front
insertRear()	Insert at rear
deletionFront()	Delete from front
deletionRear()	Delete from rear
isEmpty()	Check if deque is empty
getFront()	View front without removing
getRear()	View rear without removing

Algorithm 4.1: Palindrome using Deque

1. Traverse string left to right.
2. Insert each character at rear.
3. Repeat: delete character from both front and rear, compare.
4. If all pairs match → it's a **palindrome**.

4.5 Implementation of Deque using Python

```
def insertFront(myDeque, element):
    myDeque.insert(0, element)

def insertRear(myDeque, element):
    myDeque.append(element)

def isEmpty(myDeque):
    return len(myDeque) == 0

def deletionRear(myDeque):
    if not isEmpty(myDeque):
        return myDeque.pop()
    else:
```

```
    print("Deque empty")

def deletionFront(myDeque):
    if not isEmpty(myDeque):
        return myDeque.pop(0)
    else:
        print("Deque empty")

def getFront(myDeque):
    if not isEmpty(myDeque):
        return myDeque[0]
    else:
        print("Queue empty")

def getRear(myDeque):
    if not isEmpty(myDeque):
        return myDeque[-1]
    else:
        print("Deque empty")
```

Program 4-2: Menu-driven Deque Program

```
def main():
    dQu = list()
    choice = int(input("Enter 1 to use as queue, 2 otherwise: "))

    if choice == 1:
        element = input("Data for insertion at rear: ")
        insertRear(dQu, element)
        print("Data at front:", getFront(dQu))
        element = input("Data for insertion at rear: ")
        insertRear(dQu, element)
        print("Removed:", deletionFront(dQu))
        print("Removed:", deletionFront(dQu))
    else:
        element = input("Data for insertion at front: ")
        insertFront(dQu, element)
        print("Data at rear:", getRear(dQu))
        element = input("Data for insertion at front: ")
        insertFront(dQu, element)
        print("Removed:", deletionRear(dQu))
        print("Removed:", deletionRear(dQu))
```


Summary Points

- **Queue:** FIFO, insertion at rear, deletion from front.
- **Deque:** Insertion & deletion possible from both ends.
- Python lists can implement both structures easily.
- Additional operations: isEmpty, peek, size, getFront, getRear.

MOHAMMED MATHEEN L R

CHAPTER 5: SORTING

CHAPTER NOTES

5.1 Introduction

- **Sorting** is the process of arranging elements in a particular order (ascending, descending, alphabetical, etc.).
- It is essential for easy data retrieval and efficient searching.
- Common applications:
 - Dictionaries (alphabetical order)
 - Exam seating plans (roll number order)
 - Sorting by height, weight, etc.

5.2 Bubble Sort

Concept

- Compares **adjacent elements** and **swaps** them if they are in the wrong order.
- After each **pass**, the largest unsorted element “bubbles up” to its correct position.
- Requires **$n - 1$ passes** for a list of size **n** .

Optimization Tip If no swaps occur in a pass, the list is already sorted — the algorithm can be terminated early.

Diagram: Bubble Sort Passes We will sort:

[8, 7, 13, 1, -9, 4]

Each pass compares adjacent elements and swaps if needed.

Pass 1

[8, 7, 13, 1, -9, 4]

→ Swap 8 and 7 → [7, 8, 13, 1, -9, 4]

→ No Swap (8,13)

→ Swap 13 and 1 → [7, 8, 1, 13, -9, 4]

→ Swap 13 and -9 → [7, 8, 1, -9, 13, 4]

→ Swap 13 and 4 → [7, 8, 1, -9, 4, 13]

Pass 2

[7, 8, 1, -9, 4, 13]

→ No Swap (7,8)

→ Swap 8 and 1 → [7, 1, 8, -9, 4, 13]

→ Swap 8 and -9 → [7, 1, -9, 8, 4, 13]

→ Swap 8 and 4 → [7, 1, -9, 4, 8, 13]

Pass 3

[7, 1, -9, 4, 8, 13]

→ Swap 7 and 1 → [1, 7, -9, 4, 8, 13]

→ Swap 7 and -9 → [1, -9, 7, 4, 8, 13]

→ Swap 7 and 4 → [1, -9, 4, 7, 8, 13]

Pass 4

[1, -9, 4, 7, 8, 13]

→ Swap 1 and -9 → [-9, 1, 4, 7, 8, 13]

Pass 5

No swaps → Sorting complete.

Final list: [-9, 1, 4, 7, 8, 13]

Algorithm 5.1: Bubble Sort

BUBBLESORT(numList, n)

1. Set $i = 0$

2. While $i < n$ repeat:

3. Set $j = 0$

4. While $j < n - i - 1$ repeat:

5. If $\text{numList}[j] > \text{numList}[j+1]$, then

6. Swap $\text{numList}[j]$ and $\text{numList}[j+1]$

7. $j = j + 1$

8. $i = i + 1$

Python Program

```
def bubble_Sort(list1):  
    n = len(list1)  
    for i in range(n):
```

```
    for j in range(0, n-i-1):
        if list1[j] > list1[j+1]:
            list1[j], list1[j+1] = list1[j+1], list1[j]

numList = [8, 7, 13, 1, -9, 4]
bubble_Sort(numList)
print("The sorted list is:")
for i in range(len(numList)):
    print(numList[i], end=" ")
```

5.3 Selection Sort

Concept

- The list is divided into **sorted** and **unsorted** parts.
- The **smallest** element from the unsorted list is selected and **swapped** with the first unsorted element.
- Requires **n – 1 passes** for n elements.

Diagram: Selection Sort Passes Starting List: [8, 7, 13, 1, -9, 4]

Pass 1 - Smallest is -9 → Swap with 8

[-9, 7, 13, 1, 8, 4]

Pass 2 - Smallest in [7,13,1,8,4] is 1 → Swap with 7

[-9, 1, 13, 7, 8, 4]

Pass 3 - Smallest in [13,7,8,4] is 4 → Swap with 13

[-9, 1, 4, 7, 8, 13]

Pass 4 - Smallest in [7,8,13] is 7 (already in place)

[-9, 1, 4, 7, 8, 13]

Pass 5 - Already sorted

[-9, 1, 4, 7, 8, 13]

Algorithm 5.2: Selection Sort

SELECTIONSORT(numList, n)

1. Set i = 0
2. While i < n repeat:
3. Set min = i, flag = 0
4. Set j = i + 1
5. While j < n repeat:

```
6.     If numList[j] < numList[min]:
7.         min = j
8.         flag = 1
9.     If flag == 1:
10.        Swap numList[i], numList[min]
11.    i = i + 1
```

Python Program

```
def selection_Sort(list2):
    n = len(list2)
    for i in range(n):
        min = i
        for j in range(i + 1, n):
            if list2[j] < list2[min]:
                min = j
        list2[i], list2[min] = list2[min], list2[i]

numList = [8, 7, 13, 1, -9, 4]
selection_Sort(numList)
print("The sorted list is:")
for i in range(len(numList)):
    print(numList[i], end=" ")
```

5.4 Insertion Sort

Concept

- Elements from the unsorted part are picked and **inserted** into the correct position of the sorted part.
- Like arranging cards in hand.

Diagram: Insertion Sort Passes Initial List: [8, 7, 13, 1, -9, 4]

Pass 1 Insert 7 into sorted [8]

[7, 8, 13, 1, -9, 4]

Pass 2 Insert 13 into sorted [7,8]

[7, 8, 13, 1, -9, 4]

Pass 3 Insert 1 into sorted [7,8,13]

[1, 7, 8, 13, -9, 4]

Pass 4 Insert -9 into sorted [1,7,8,13]

[-9, 1, 7, 8, 13, 4]

Pass 5 Insert 4 into sorted [-9,1,7,8,13]

[-9, 1, 4, 7, 8, 13]

Algorithm 5.3: Insertion Sort

INSERTIONSORT(numList, n)

```
1. Set i = 1
2. While i < n repeat:
3.   temp = numList[i]
4.   Set j = i - 1
5.   While j >= 0 and numList[j] > temp:
6.     numList[j+1] = numList[j]
7.     j = j - 1
8.   numList[j+1] = temp
9.   i = i + 1
```

Python Program

```
def insertion_Sort(list3):
    n = len(list3)
    for i in range(n):
        temp = list3[i]
        j = i-1
        while j >= 0 and temp < list3[j]:
            list3[j+1] = list3[j]
            j -= 1
        list3[j+1] = temp

numList = [8, 7, 13, 1, -9, 4]
insertion_Sort(numList)
print("The sorted list is:")
for i in range(len(numList)):
    print(numList[i], end=" ")
```

5.5 Time Complexity of Algorithms

Basic Complexity Concepts

- **Constant Time ($O(1)$):** No loops.
- **Linear Time ($O(n)$):** Single loop.
- **Quadratic Time ($O(n^2)$):** Nested loops.

All three sorting algorithms in this chapter (bubble, selection, insertion) have **$O(n^2)$** complexity due to nested loops.

Summary Points

- **Sorting:** Rearranging elements for efficient access.
- **Bubble Sort:** Repeated adjacent swaps, $n-1$ passes.
- **Selection Sort:** Select and place smallest, reduce unsorted list.
- **Insertion Sort:** Insert each element into sorted part at the correct position.
- **Time Complexity:** Helps choose suitable algorithms for large datasets.

MOHAMMED MATHEEN L R

CHAPTER 6: SEARCHING

CHAPTER NOTES

6.1 Introduction

- Searching is the process of locating a specific element (called the **key**) in a collection.
- It determines **whether the key is present** and, if so, **its position**.
- Three main techniques:
 1. **Linear Search**
 2. **Binary Search**
 3. **Search by Hashing**

6.2 Linear Search (Sequential Search)

- It compares each element of the list with the key until a match is found or the list ends.
- Useful for **unsorted** or **small-sized** lists.

Algorithm 6.1: Linear Search

```
LinearSearch(numList, key, n)
Step 1: SET index = 0
Step 2: WHILE index < n, REPEAT Step 3
Step 3: IF numList[index] = key THEN
    PRINT "Element found at position", index+1
    STOP
ELSE
    index = index + 1
Step 4: PRINT "Search unsuccessful"
```

Example

List: numList = [8, -4, 7, 17, 0, 2, 19], Key: 17

Index	Comparison	Result
0	$8 == 17$	not found

Index	Comparison	Result
1	-4 == 17	not found
2	7 == 17	not found
3	17 == 17	Found at index 3

```
def linearSearch(list, key):
    for index in range(len(list)):
        if list[index] == key:
            return index + 1
    return None

list1 = []
maximum = int(input("How many elements in your list? "))
for i in range(maximum):
    n = int(input())
    list1.append(n)

key = int(input("Enter the number to be searched:"))
position = linearSearch(list1, key)

if position is None:
    print("Number", key, "is not present in the list")
else:
    print("Number", key, "is present at position", position)
```

6.3 Binary Search

- Requires a **sorted** list.
- More efficient than linear search.
- Repeatedly divides list into halves and compares key with the **middle element**.

Algorithm 6.2: Binary Search

```
BinarySearch(numList, key)
Step 1: SET first = 0, last = n - 1
Step 2: WHILE first <= last, REPEAT Step 3
Step 3: SET mid = (first + last)//2
        IF numList[mid] == key THEN
```

```

    PRINT "Element found at position", mid + 1
    STOP
ELSE IF numList[mid] > key THEN
    last = mid - 1
ELSE
    first = mid + 1
Step 4: PRINT "Search unsuccessful"

```

Example

List: numList = [2,3,5,7,10,11,12,17,19,23,29,31,37,41,43], Key: 2 Here's the corrected and neatly formatted table in Markdown:

Step	Low	High	Mid	Mid Value	Comparison
1	0	14	7	17	$2 < 17 \rightarrow$ left half
2	0	6	3	7	$2 < 7 \rightarrow$ left half
3	0	2	1	3	$2 < 3 \rightarrow$ left half
4	0	0	0	2	Match found

Minimum iterations: 1 (if key is in middle)

Maximum iterations: $\lfloor \log_2(n) \rfloor$

Program 6-2: Binary Search in Python

```

def binarySearch(list, key):
    first = 0
    last = len(list) - 1
    while first <= last:
        mid = (first + last)//2
        if list[mid] == key:
            return mid
        elif key > list[mid]:
            first = mid + 1
        else:
            last = mid - 1
    return -1

```

```
numList = []
print("Enter elements in ascending order (-999 to stop):")
num = int(input())
while num != -999:
    numList.append(num)
    num = int(input())

key = int(input("Enter the number to be searched: "))
pos = binarySearch(numList, key)
if pos != -1:
    print(key, "is found at position", pos + 1)
else:
    print(key, "is not found in the list")
```

6.3.1 Applications of Binary Search

- Dictionary/telephone directory search.
- Database indexing.
- Data compression, routing tables, etc.

6.4 Search by Hashing

- Direct access method using a **hash function**.
- Searches key in constant time $O(1)$.
- Hash function computes index:
$$h(\text{element}) = \text{element} \% \text{size_of_table}$$

Example

List: [34, 16, 2, 93, 80, 77, 51], Hash Table size = 10

Element	Hash Value	Index in Table
34	4	4
16	6	6
2	2	2
93	3	3
80	0	0

Element	Hash Value	Index in Table
77	7	7
51	1	1

Final Hash Table:

Index	Value
0	80
1	51
2	2
3	93
4	34
5	None
6	16
7	77
8	None
9	None

Program 6-3: Hashing in Python

```
def hashFind(key, hashTable):
    if hashTable[key % 10] == key:
        return (key % 10) + 1
    else:
        return None

hashTable = [None]*10
L = [34, 16, 2, 93, 80, 77, 51]

for i in L:
    hashTable[i % 10] = i

key = int(input("Enter the number to be searched: "))
```

```

position = hashFind(key, hashTable)
if position:
    print("Number", key, "present at", position, "position")
else:
    print("Number", key, "is not present in the hash table")

```

6.4.1 Collision

- Occurs when multiple elements hash to the same index.
- Requires **collision resolution** techniques (not covered here).
- A **perfect hash function** prevents collision by mapping each element uniquely.

Summary

Technique	Best For	List Requirement	Time Complexity	Notes
Linear Search	Small/Unordered lists	None	$O(n)$	Simple but slow for large n
Binary Search	Large/Sorted lists	Sorted	$O(\log n)$	Fast, requires sorted input
Hashing	Fastest lookups	Hash Function	$O(1)$	Needs collision handling

CHAPTER 7: UNDERSTANDING DATA

CHAPTER NOTES

7.1 Introduction to Data

Data is essential for human decision-making in nearly every field of life. Whether we are selecting a college, making business plans, monitoring health, or predicting weather, we rely on data.

Data are raw facts and figures which, when processed, yield **information** that helps in making decisions. For instance: - Colleges maintain **placement data** of students to attract new students. - Governments conduct **census** to gather demographic data. - **Banks** maintain account-related data for every customer. - **Sports teams** analyze opponent data to form strategies.

Definition: Data is a collection of characters, numbers, or symbols that represent values of some variables.

Examples of Common Data Types:

- Personal information: Name, Age, Gender, Contact
- Transactional data: Online or offline purchases
- Multimedia: Images, videos, graphics, audio
- Documents and web content
- Social media posts
- Sensor signals (like IoT data)
- Satellite data (weather, earth observation, etc.)

7.1.1 Importance of Data

Humans and computers both rely heavily on data. Computers can process large volumes of data quickly and reveal patterns or traits not easily visible.

Real-life Examples:

- **ATM transactions:** The system deducts the withdrawn amount from your bank account.
- **Weather monitoring:** Meteorological departments analyze satellite data for warnings.
- **Businesses:** Analyze market trends, customer feedback, and dynamic pricing.
- **Transportation:** Cab apps adjust pricing based on demand and supply.
- **Restaurants:** Offer discounts during “happy hours” by analyzing customer visits.

Other Scenarios:

- **EVMs:** Record and count votes digitally.
- **Scientific research:** Involves collecting experimental data.
- **Pharmaceuticals:** Analyze medicine effectiveness through trials.
- **Libraries:** Store book and membership data.
- **Search Engines:** Analyze web content to show relevant results.
- **Weather Systems:** Use satellite data to generate alerts.

7.1.2 Types of Data

(A) Structured Data: Organized in a tabular format (rows and columns), structured data can be easily processed using spreadsheets or databases. Each row is a record, and each column is an attribute.

Table 7.1: Structured Inventory Data

ModelNo	ProductName	Unit Price	Discount(%)	Items_in_Inventory
ABC1	Water Bottle	126	8	13
ABC2	Melamine Plates	320	5	45
ABC3	Dinner Set	4200	10	8
GH67	Jug	80	0	10

With structured data, you can calculate: - Total items in inventory - Total inventory value

Table 7.2: More Structured Examples - Books: Title, Author, Price, Year - Fee Payments: Student Name, Roll No., Amount - ATM Withdrawals: Account No., Amount, ATM ID, Date & Time

(B) Unstructured Data: Lacks a fixed format or structure. Examples: - News articles - Emails - Social media posts - Documents with images, audio, video

Example: A newspaper page can have a mix of text, ads, images with no fixed layout.

Metadata: Data about data

E.g., For a photo → size, type (JPEG/PNG), resolution. For an email → subject, sender, attachments.

7.2 Data Collection

Before data processing, you must **collect or identify** relevant data.

3 Scenarios:

1. **Manual Records** (e.g., diaries): Must be converted to digital format.
2. **Digital Files**: Like CSV files, can be directly processed.
3. **No existing data**: Need to create software (e.g., using Python) to store and retrieve new data.

Our daily digital activities constantly generate data (e.g., browsing, transactions, healthcare records).

Example Applications:

- **Hospitals**: Use patient data for service improvements.
- **Shopping malls**: Analyze co-purchased items to optimize product placement.
- **Social Media**: Public sentiment analysis before elections.
- **Global Institutions**: Use economic data for forecasting.

7.3 Data Storage

After collection, data needs to be stored safely for current and future use. The **volume of data** is increasing rapidly, making storage challenging.

Common Storage Devices:

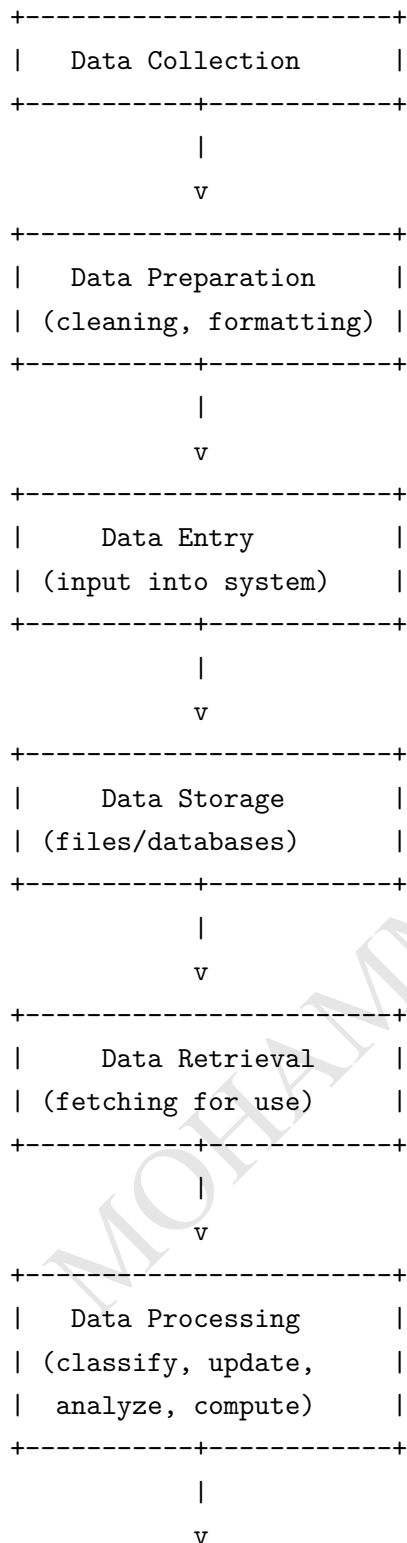
- Hard Disk Drive (HDD)
- Solid State Drive (SSD)
- CD/DVD
- Tape Drives
- Pen Drives
- Memory Cards

Data such as documents, images, audio, video, etc., are stored as **files**. For complex requirements, we use a **Database Management System (DBMS)** instead of simple file storage.

7.4 Data Processing

Just storing data is not enough. Data needs to be **processed** to extract information. Raw data alone cannot be used for analysis or decision-making.

Here is the **Data Processing Cycle** diagram from the textbook, **properly redrawn and explained** for clarity and accuracy:

Figure 7.1: Data Processing Cycle

```

+-----+
|   Data Output   |
| (Reports, results, |
| visualizations)  |
+-----+

```

Explanation of Each Step:

1. Data Collection

Gathering raw data from various sources like forms, sensors, logs, or digital inputs.

2. Data Preparation

Cleaning data (removing errors, duplicates), converting formats, and making it consistent.

3. Data Entry

Inputting the cleaned data into digital systems (manually or via automated processes).

4. Data Storage

Saving data in files, spreadsheets, or databases for persistent use.

5. Data Retrieval

Accessing stored data when needed for processing.

6. Data Processing

Analyzing, classifying, computing, and transforming data to generate meaningful results.

7. Data Output

Presenting the results in the form of charts, tables, summaries, or reports.

Figure 7.2: Real-Life Data Processing Examples

Problem Statement	Inputs	Processing	Output
Exam registration	Student details, payment info	Validate data, assign roll no.	Admit card
ATM withdrawal	Account info, PIN	Verify & deduct amount	Currency, receipt
Train ticket booking	Journey info, payment	Check availability, assign berth	E-ticket

7.5 Statistical Techniques for Data Processing

7.5.1 Measures of Central Tendency

(A) Mean (Average)

$$[\text{Mean} = \frac{x_1 + x_2 + \dots + x_n}{n}]$$

- Gives the average value. - Not reliable if data has **outliers** (extreme values).

Example: [90, 102, 110, 115, 85, 90, 100, 110, 110]

→ Mean = 101.33

(B) Median

- Middle value in **sorted** data.
- If odd count → middle item
- If even count → average of two middle items

Sorted Example: [85, 90, 90, 100, 102, 110, 110, 115]

→ Median = 102

(C) Mode

- Most frequent value.
- Can be used for both numeric and non-numeric data.
- May have one, multiple, or no mode.

Example: Mode = 110 (occurs 3 times)

7.5.2 Measures of Variability**(A) Range**

$$[\text{Range} = \text{Max} - \text{Min}]$$

- Shows spread of data. - Sensitive to outliers.

Example: Heights → Max = 115, Min = 85 → Range = 30 cm

(B) Standard Deviation (σ) Measures how values differ from the **mean**.

$$[\sigma = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}}]$$

Table 7.3: SD Calculation for Heights

- Mean = 101.33
- $SD \approx \sqrt{938/9} = approx.10.21$

Lower SD → tightly clustered data

Higher SD → widely spread data

Important Statistical Technique Selection

Task	Best Technique
Disparity in salaries	Standard Deviation
Average class performance	Mean
Compare height in 2 cities	Mean or SD
Popular car color	Mode
Find dominant data value	Mode
Compare incomes	Mean + SD

Glossary of Key Terms

Term	Explanation
Data	Raw, unorganised facts and figures
Information	Processed data that is meaningful
Structured Data	Tabular data with rows and columns
Unstructured Data	Data without a predefined format
Metadata	Data about other data
Data Collection	Gathering data from various sources
Data Storage	Saving data on physical/digital devices
Data Processing	Converting raw data into useful output
Mean	Average of numerical data
Median	Middle value in sorted data

Term	Explanation
Mode	Most frequently occurring value
Range	Difference between max and min values
Standard Deviation (σ)	Measures the spread of values around the mean
Outlier	Data point that is far from other values
DBMS	Software to manage large datasets efficiently

CHAPTER 8 DATABASE CONCEPTS

CHAPTER NOTES

8.1 Introduction

Manual record-keeping systems are inefficient and prone to errors. Consider a teacher manually recording attendance for 50 students over 26 working days: that results in 1,300 manual entries monthly!

Limitations of Manual Record-Keeping

1. Rewriting student details across records introduces chances of error.
2. Data redundancy and inconsistency due to repeated entries.
3. Difficulty in modifying or retrieving information.
4. Risk of physical damage/loss.
5. Error-prone calculations.

Why Move to Electronic Storage?

- Easy copying and promotion of records.
- Efficient search, update, and deletion.
- Better management via computerised data files.

8.2 File System

A **file** is a container on a computer to store data like text, code, images, videos, CSVs, etc.

Example: Two Data Files

- STUDENT
- ATTENDANCE

Table 8.1 — STUDENT File Maintained by Office Staff

Roll Number	SName	SDateofBirth	GName	GPhone	GAddress
1	Atharv Ahuja	2003-05-15	Amit Ahuja	5711492685	G-35, Ashok Vihar, Delhi
2	Daizy Bhutia	2002-02-28	Baichung Bhutia	7110047139	Flat no. 5, Darjeeling Appt., Shimla
3	Taleem Shah	2002-02-28	Himanshu Shah	9818184855	26/77, West Patel Nagar, Ahmedabad
4	John Dsouza	2003-08-18	Danny Dsouza	—	S -13, Ashok Village, Daman
5	Ali Shah	2003-07-05	Himanshu Shah	9818184855	26/77, West Patel Nagar, Ahmedabad
6	Manika P.	2002-03-10	Sujata P.	7802983674	HNO-13, B- block, Preet Vihar, Madurai

Table 8.2 — ATTENDANCE File Maintained by Class Teacher

AttendanceDate	RollNumber	SName	AttendanceStatus
2018-09-01	1	Atharv Ahuja	P
2018-09-01	2	Daizy Bhutia	P
2018-09-01	3	Taleem Shah	A
2018-09-01	4	John Dsouza	P
2018-09-01	5	Ali Shah	A
2018-09-01	6	Manika P.	P
2018-09-02	1	Atharv Ahuja	P
2018-09-02	2	Daizy Bhutia	P
2018-09-02	3	Taleem Shah	A
2018-09-02	4	John Dsouza	A
2018-09-02	5	Ali Shah	P
2018-09-02	6	Manika P.	P

8.2.1 Limitations of File System

Problem	Description
Difficulty in Access	Application programs required for data retrieval.
Data Redundancy	Same data (e.g., guardian names) repeated.
Data Inconsistency	Inconsistent updates across files.
Data Isolation	No link/mapping between files.
Data Dependence	Program changes needed when structure changes.
Controlled Data Sharing	Difficult to manage user access levels.

8.3 Database Management System (DBMS)

A **DBMS** is software to store, manage, and retrieve logically related data efficiently.

Common DBMS Software:

- MySQL
- Oracle
- PostgreSQL
- SQL Server
- MS Access
- MongoDB

Table 8.3 — Use of Database in Real-Life Applications

Application	Database Maintains Information About
Banking	Customer, Account, Loans, Transactions
Crop Loan	Farmer info, land, credit card, repayment
Inventory Management	Products, orders, customers
Organisation Resources	Employee, salary, department, branches
Online Shopping	Items, users, preferences

File System to DBMS: Key Transformations

- Remove duplicated columns.
- Split STUDENT table and creating another GUARDIAN table.
- Introduce GUID (Guardian ID).
- Originally, the STUDENT file contained guardian information, which was repeated for students having the same guardian (e.g., siblings).
- By splitting the STUDENT file and creating a separate GUARDIAN table (linked using GUID), the DBMS design removed redundancy.
- Similarly, removing the student name from ATTENDANCE and linking it via RollNumber established a clean reference.

Figure 8.1 — Record Structure in DBMS

STUDENT	GUARDIAN	ATTENDANCE
-----	-----	-----
RollNumber	GUID	AttendanceDate
SName	GName	RollNumber
SDateofBirth	GPhone	AttendanceStatus
GUID	GAddress	

Table 8.4 — Snapshot of STUDENT Table

RollNumber	SName	SDateofBirth	GUID
1	Atharv Ahuja	2003-05-15	444444444444
2	Daizy Bhutia	2002-02-28	111111111111
3	Taleem Shah	2002-02-28	
4	John Dsouza	2003-08-18	333333333333
5	Ali Shah	2003-07-05	101010101010
6	Manika P.	2002-03-10	466444444666

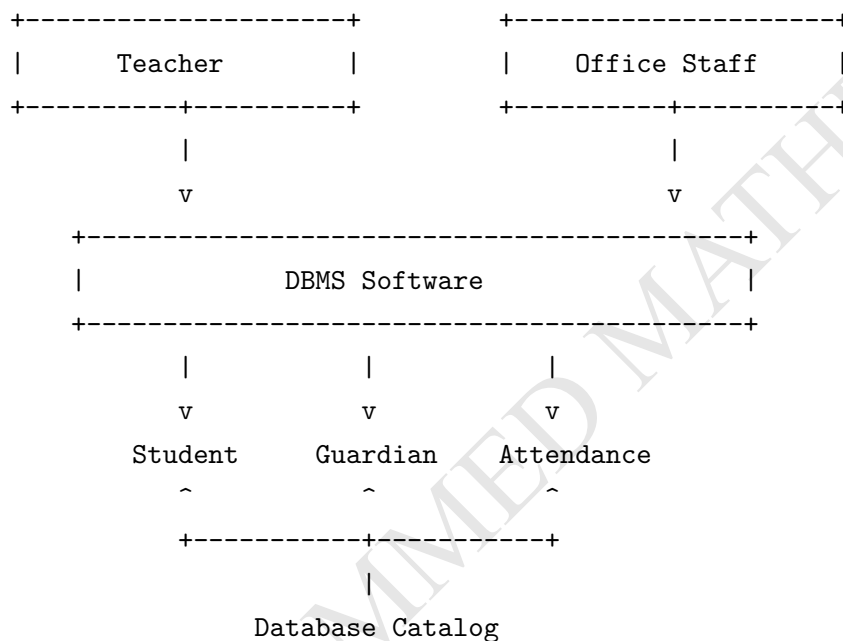
Table 8.5 — Snapshot of GUARDIAN Table

GUID	GName	GPhone	GAddress
444444444444	Amit Ahuja	5711492685	G-35, Ashok Vihar, Delhi
111111111111	Baichung Bhutia	7110047139	Flat no. 5, Darjeeling Appt., Shimla
101010101010	Himanshu Shah	9818184855	26/77, West Patel Nagar, Ahmedabad
333333333333	Danny Dsouza	—	S -13, Ashok Village, Daman
466444444666	Sujata P.	7802983674	HNO-13, B- block, Preet Vihar, Madurai

Table 8.6 — Snapshot of ATTENDANCE Table

Date	RollNumber	Status
2018-09-01	1	P
2018-09-01	2	P
2018-09-01	3	A
2018-09-01	4	P
2018-09-01	5	A
2018-09-01	6	P
2018-09-02	1	P
2018-09-02	2	P

Date	RollNumber	Status
2018-09-02	3	A
2018-09-02	4	A
2018-09-02	5	P
2018-09-02	6	P

Figure 8.2 — STUDENTATTENDANCE DBMS Environment**Key Concepts in DBMS**

Concept	Description
Database Schema	Structure of the database including table names, attributes, and constraints
Data Constraint	Conditions imposed on values of attributes (e.g., NOT NULL, UNIQUE)
Meta-data	Data about the data; includes schema and constraints
Database Instance	Current snapshot of the data in the database
Query	Request for accessing/manipulating data in the DBMS

Concept	Description
Data Manipulation	Operations such as insert, update, delete
Database Engine	Internal component that processes queries and manages storage

8.4 Relational Data Model

The **relational model** stores data in the form of **tables (relations)** made up of **attributes (columns)** and **tuples (rows)**.

Table 8.7 — Relation Schemas

Relation	Attributes
STUDENT	RollNumber, SName, SDateofBirth, GUID
ATTENDANCE	AttendanceDate, RollNumber, AttendanceStatus
GUARDIAN	GUID, GName, GPhone, GAddress

Common Terminologies

Term	Meaning
Attribute	A column in a relation
Tuple	A row in a relation
Domain	Set of permissible values for an attribute
Degree	Number of attributes (columns) in a relation
Cardinality	Number of tuples (rows) in a relation

Three Important Properties of a Relation

1. Properties of Attributes:

- Attribute names must be **distinct**.
- The **order of attributes** in a relation does not matter.

2. Properties of Tuples:

- All tuples (rows) in a relation must be **unique**.
- The **order of tuples** in a relation does not matter.

3. Properties of Values:

- Every value in a tuple must be **atomic** (indivisible).
- Values may be **NULL** to represent missing or unknown information.
- All values in a column must be from the **same domain**.

Figure 8.4 — Relation GUARDIAN Example

GUID	GName	GPhone	GAddress
4444444444444	Amit Ahuja	5711492685	G-35, Ashok Vihar, Delhi
1111111111111	Baichung Bhutia	7110047139	Flat no. 5, Darjeeling Appt., Shimla
101010101010	Himanshu Shah	9818184855	26/77, West Patel Nagar, Ahmedabad
3333333333333	Danny Dsouza	—	S -13, Ashok Village, Daman
4664444444666	Sujata P.	7802983674	HNO-13, B- block, Preet Vihar, Madurai

Degree = 4 (columns), **Cardinality** = 5 (rows)

8.5 Keys in a Relational Database

Keys help uniquely identify records and maintain relationships.

8.5.1 Candidate Key

- Any attribute (or combination) that can uniquely identify a tuple.
- Example: GUID, GPhone in GUARDIAN.

8.5.2 Primary Key

- Chosen candidate key to uniquely identify rows.
- Example: GUID in GUARDIAN table.

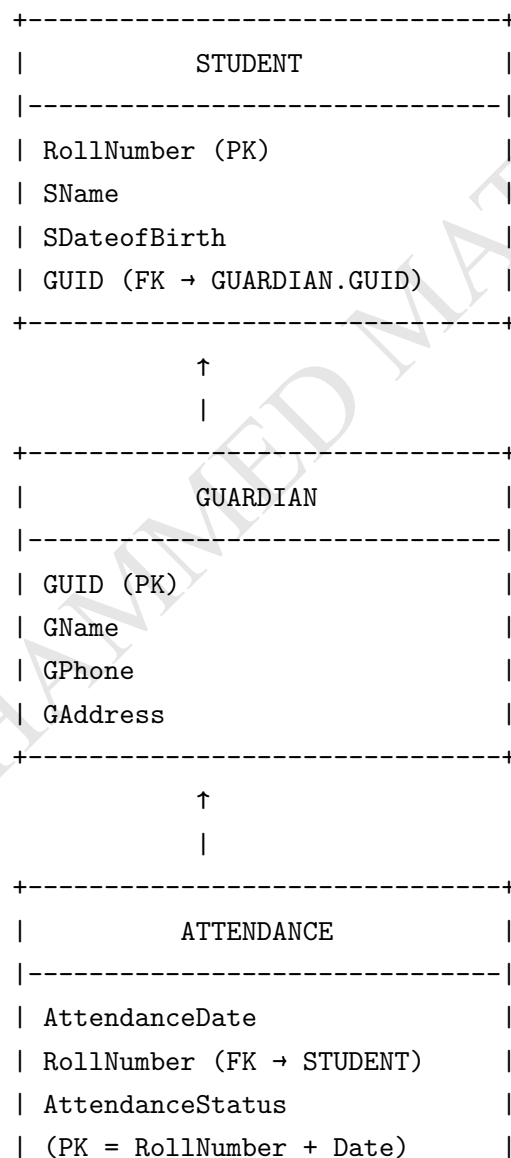
8.5.3 Composite Primary Key

- A key formed by combining two or more attributes.
- Example: {RollNumber, AttendanceDate} in ATTENDANCE.

8.5.4 Foreign Key

- Attribute that refers to a primary key in another table.
- Used to link two tables.

Figure 8.5 — STUDENTATTENDANCE Schema Diagram



+-----+

Summary of Chapter 8

- Manual file systems are inefficient and lead to redundancy.
- DBMS manages data efficiently, with support for querying and concurrent access.
- The relational model uses tables linked via keys.
- Primary keys uniquely identify rows; foreign keys link tables.
- All data is stored in atomic form with defined schemas and constraints.

Summary of Key Terms

Term	Explanation
File	Container to store digital data like text, audio, code, etc.
DBMS	Software that manages storage, access, manipulation, and retrieval of data.
Database	Organized collection of logically related data.
Schema	Structure of a database including tables, attributes, and constraints.
Instance	Current snapshot or state of data in a database.
Attribute	A column in a table representing a data field.
Tuple	A row in a table representing a single record.
Domain	Permissible set of values for an attribute.

Term	Explanation
Degree	Number of attributes (columns) in a table.
Cardinality	Number of records (rows) in a table.
Primary Key	Attribute(s) that uniquely identify each tuple in a relation.
Candidate Key	Attributes that qualify to be primary key.
Alternate Key	Candidate key not chosen as primary key.
Composite Key	A primary key made up of multiple attributes.
Foreign Key	Attribute referring to a primary key in another table.
Query	Request to retrieve or manipulate data in the database.
Constraint	Condition or rule enforced on data to maintain integrity.
Meta-data	Data that describes other data (e.g., schema, constraints).
Atomic Value	Single indivisible value in a data field.
NULL	Special value indicating missing or unknown data.
Database Engine	Core service that processes database queries and manages storage.
Data Manipulation	Operations such as INSERT, UPDATE, DELETE performed on data.

CHAPTER 9 STRUCTURED QUERY LANGUAGE (SQL)

CHAPTER NOTES

Introduction

Overview of RDBMS and SQL

- In the **previous chapter**, you learned about **Relational Database Management Systems (RDBMS)** — systems that organize data in tables called **relations**.
- Examples of RDBMS:
 - MySQL
 - Microsoft SQL Server
 - PostgreSQL
 - Oracle

Purpose of RDBMS

- Allows us to:
 - **Create a database** using **relations** (tables).
 - **Store, retrieve, and manipulate** data using **queries**.
- The tool used to perform all these actions is called **SQL (Structured Query Language)**.

9.2 Structured Query Language (SQL)

What is SQL?

- **SQL (Structured Query Language)** is a special-purpose language used to **access and manipulate data** in a database.
- It is used **instead of writing application programs**, especially in the context of **Database Management Systems (DBMS)**.
- SQL works with popular RDBMS like:
 - **MySQL**
 - **ORACLE**
 - **SQL Server**

Key Features of SQL

Feature	Description
Easy to Learn	SQL uses descriptive English words ; it's readable and intuitive.
Case-Insensitive	Commands like SELECT and select are treated the same.
Declarative	You tell SQL what you want, not how to do it.
Powerful	SQL does more than just queries. It includes commands to define data, insert data, update, and manage constraints.

Capabilities of SQL

- **Define data structures** (tables, attributes, constraints)
- **Manipulate data** (insert, delete, update)
- **Query data** (retrieve specific or filtered data)
- **Declare constraints** (rules on what kind of data can go into tables)

SQL with MySQL Example

- The textbook uses the **StudentAttendance** database (introduced in Chapter 8) to explain SQL queries.
- Students will **create**, **populate**, and **query** this database using SQL.

9.2.1 Installing MySQL

What is MySQL?

- **MySQL** is a popular, open-source **Relational Database Management System (RDBMS)**.
- It supports **SQL** and is widely used to create and manage databases.

How to Install MySQL?

- **Download** from the official site:
<https://dev.mysql.com/downloads>
- **Install the software** on your computer.
- **Start MySQL service.**
 - Once started, the terminal (or command-line interface) will show the prompt:

```
mysql>
```

- This indicates that **MySQL is ready** to accept SQL commands.

Key Points to Remember While Using SQL in MySQL

Point	Description
Case Insensitivity	SQL keywords, table names, and column names are not case-sensitive.
End Statements with ;	Always terminate SQL commands with a semicolon .
Multiline Statements	If an SQL statement is long: → Write it over multiple lines → Don't put ; until the last line → mysql> prompt changes to -> for continuation

Activity 9.1 Q: Find and list other types of databases besides RDBMS.

Example Answers:

- NoSQL databases (e.g., MongoDB, CouchDB)
- Hierarchical databases
- Object-oriented databases
- Network databases

9.3 Data Types and Constraints in MySQL

What are Data Types and Constraints?

- Every table (**relation**) is made up of **attributes** (columns).
- Each attribute:
 - Has a **data type** — defines the kind of values it can store.
 - May have **constraints** — rules that restrict the values entered into that column.

9.3.1 Data Type of Attribute

Definition: The **data type** of an attribute defines:

- What kind of **values** can be stored.
- What kind of **operations** can be performed on them.

Data Type	Description
CHAR(n)	Fixed-length character string. - Length n can be from 0 to 255. - e.g., CHAR(10) stores exactly 10 characters; unused space is padded with spaces.
VARCHAR(n)	Variable-length character string. - Length n up to 65535. - Only actual input length occupies space.
INT	Integer values. - Occupies 4 bytes. - Use BIGINT (8 bytes) for larger numbers.
FLOAT	Decimal (floating-point) numbers. - Occupies 4 bytes.
DATE	Stores dates in 'YYYY-MM-DD' format. - Range: '1000-01-01' to '9999-12-31'.

Activity 9.2: What other data types are supported in MySQL? Are there other variants of integer and float?

Examples:

- BIGINT, DECIMAL, DOUBLE
- TEXT, TINYTEXT, BLOB
- DATETIME, TIMESTAMP, TIME, YEAR

9.3.2 Constraints

Definition: **Constraints** are rules or restrictions placed on data in a table to ensure **accuracy** and **reliability**.

Constraint	Description
NOT NULL	Value must be provided; cannot be left empty (NULL).
UNIQUE	All values in the column must be different .

Constraint	Description
DEFAULT	Assigns a default value if no value is entered.
PRIMARY KEY	Uniquely identifies each row in a table. Must be unique and not null .
FOREIGN KEY	Refers to a primary key in another table. Used to create relationships.

Concept Check: Q: Which two constraints, when applied together, produce a **Primary Key**? **NOT NULL + UNIQUE**

9.4 SQL for Data Definition

Purpose This section explains how to:

- Create a **database**
- Create **tables** inside a database
- Define **attributes, data types, and constraints**
- View or **modify** table structure

1. STUDENT Table Attributes & Description

Attribute	Data Type	Constraint	Description
RollNumber	INT	PRIMARY KEY	Unique roll number of student (1–100)
SName	VARCHAR(20)	NOT NULL	Student name
SDateofBirth	DATE	NOT NULL	Date of birth
GUID	CHAR(12)	FOREIGN KEY	Guardian's Aadhaar (linked to GUARDIAN)

SQL Command

```
CREATE TABLE STUDENT (
    RollNumber INT,
    SName VARCHAR(20),
```

```

SDateofBirth DATE,
GUID CHAR(12),
PRIMARY KEY (RollNumber)
);

```

DESCRIBE Output

```
mysql> DESCRIBE STUDENT;
```

Field	Type	Null	Key	Default	Extra
RollNumber	int	NO	PRI	NULL	
SName	varchar(20)	YES		NULL	
SDateofBirth	date	YES		NULL	
GUID	char(12)	YES		NULL	

2. GUARDIAN Table Attributes & Description

Attribute	Data Type	Constraint	Description
GUID	CHAR(12)	PRIMARY KEY	Unique Aadhaar number of guardian
GName	VARCHAR(20)	NOT NULL	Guardian name
GPhone	CHAR(10)	NULL, UNIQUE	Phone number (optional but unique)
GAddress	VARCHAR(30)	NOT NULL	Address of guardian

3. ATTENDANCE Table Attributes & Description

Attribute	Data Type	Constraint	Description
AttendanceDate	DATE	Composite Primary Key	Date of attendance
RollNumber	INT	Composite PK, FOREIGN KEY	Student's roll number
AttendanceStatus	CHAR(1)	NOT NULL	'P' for present, 'A' for absent

SHOW TABLES Command in SQL Purpose:

The SHOW TABLES; command is used to **list all tables** present in the **currently selected database**.

Syntax:

```
SHOW TABLES;
```

Example:

Assume you've created and selected the database named StudentAttendance.

```
USE StudentAttendance;
```

```
SHOW TABLES;
```

Output

```
+-----+
| Tables_in_studentattendance |
+-----+
| STUDENT                     |
| GUARDIAN                    |
| ATTENDANCE                  |
+-----+
3 rows in set (0.00 sec)
```

This shows that the database contains **three tables**: STUDENT, GUARDIAN, and ATTENDANCE.

9.4.4 — ALTER TABLE

Purpose of ALTER TABLE Once a table is created, you might need to:

- Add or remove **attributes (columns)**
- Modify **data types** or **constraints**
- Add **primary/foreign/unique** keys

All these structural changes are done using the ALTER TABLE command.

General Syntax:

```
ALTER TABLE table_name
    <alteration_action>;
```

(A) Add Primary Key to a Table

Example:

```
ALTER TABLE GUARDIAN
ADD PRIMARY KEY (GUID);

ALTER TABLE ATTENDANCE
ADD PRIMARY KEY (AttendanceDate, RollNumber);
```

Composite Primary Key: Combines multiple columns to uniquely identify a record.

(B) Add Foreign Key to a Table

Ensure:

- The referenced table exists.
- The referenced column is a primary key.
- Data types must match.

Syntax:

```
ALTER TABLE table_name
ADD FOREIGN KEY (column_name)
REFERENCES referenced_table (referenced_column);
```

Example:

```
ALTER TABLE STUDENT
ADD FOREIGN KEY (GUID) REFERENCES GUARDIAN(GUID);
```

(C) Add UNIQUE Constraint Ensures all values in a column are distinct.

Example:

```
ALTER TABLE GUARDIAN
ADD UNIQUE (GPhone);
```

(D) Add New Attribute

Syntax:

```
ALTER TABLE table_name  
ADD column_name datatype;
```

Example:

```
ALTER TABLE GUARDIAN  
ADD Income INT;
```

(E) Modify Data Type of Attribute**Syntax:**

```
ALTER TABLE table_name  
MODIFY column_name NEW_DATATYPE;
```

Example:

```
ALTER TABLE GUARDIAN  
MODIFY GAddress VARCHAR(40);
```

(F) Modify Constraint (e.g., Add NOT NULL)**Example:**

```
ALTER TABLE STUDENT  
MODIFY SName VARCHAR(20) NOT NULL;
```

You must **re-specify the datatype** when changing constraints.

(G) Add DEFAULT Value**Syntax:**

```
ALTER TABLE table_name  
MODIFY column_name datatype DEFAULT default_value;
```

Example:

```
ALTER TABLE STUDENT  
MODIFY SDateofBirth DATE DEFAULT '2000-05-15';
```

(H) Remove an Attribute**Syntax:**

```
ALTER TABLE table_name  
DROP column_name;
```

Example:

```
ALTER TABLE GUARDIAN  
DROP Income;
```

(I) Remove Primary Key**Syntax:**

```
ALTER TABLE table_name  
DROP PRIMARY KEY;
```

Example:

```
ALTER TABLE GUARDIAN  
DROP PRIMARY KEY;
```

Note: Each table **should** have a primary key for data integrity. You can re-add it using ADD PRIMARY KEY.

ALTER TABLE — Summary Table

Operation	Purpose	Syntax / Example
Add Primary Key	To define a column(s) as a primary key	ALTER TABLE table_name ADD PRIMARY KEY (column); ALTER TABLE GUARDIAN ADD PRIMARY KEY (GUID);
Add Composite Primary Key	Use multiple columns as a single primary key	ALTER TABLE ATTENDANCE ADD PRIMARY KEY (AttendanceDate, RollNumber);
Add Foreign Key	To create a relationship to another table's primary key	ALTER TABLE STUDENT ADD FOREIGN KEY (GUID) REFERENCES GUARDIAN(GUID);

Operation	Purpose	Syntax / Example
Add UNIQUE Constraint	To ensure all values in a column are unique	ALTER TABLE GUARDIAN ADD UNIQUE (GPhone);
Add New Attribute	To add a new column to an existing table	ALTER TABLE GUARDIAN ADD Income INT;
Modify Data Type	To change data type of an existing column	ALTER TABLE GUARDIAN MODIFY GAddress VARCHAR(40);
Modify Constraint	To enforce NOT NULL or other constraints	ALTER TABLE STUDENT MODIFY SName VARCHAR(20) NOT NULL;
Add DEFAULT Value	To provide a default value for a column	ALTER TABLE STUDENT MODIFY SDateofBirth DATE DEFAULT '2000-05-15';
Remove Attribute	To delete a column from a table	ALTER TABLE GUARDIAN DROP Income;
Remove Primary Key	To drop an existing primary key	ALTER TABLE GUARDIAN DROP PRIMARY KEY;

9.4.5 — DROP Statement

Purpose of the DROP Statement The DROP statement in SQL is used to **permanently delete** a:

- **Table**, or
- **Entire database**

Once dropped:

- The data **cannot be recovered** (use with caution).
- All associated data, structure, and constraints are **lost permanently**.

A. DROP a Table

Syntax:

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE STUDENT;
```

This command will **delete the STUDENT table**, including all its data and structure from the database.

B. DROP a Database**Syntax:**

```
DROP DATABASE database_name;
```

Example:

```
DROP DATABASE StudentAttendance;
```

This will **delete the entire database** and all tables (STUDENT, GUARDIAN, ATTENDANCE, etc.) within it.

Important Notes:

- There is **no undo** for DROP. Once executed, the data is gone permanently.
- Always **double-check** before dropping tables or databases.
- If you're testing, it's good practice to back up data or use temporary test tables.

9.5 — SQL for Data Manipulation**What is Data Manipulation?**

- After defining tables (DDL), we use **Data Manipulation Language (DML)** commands to:
 - **Insert**
 - **Update**
 - **Delete**

9.5.1 INSERTION of Records Syntax 1: Insert into all columns

```
INSERT INTO table_name  
VALUES (value1, value2, ..., valueN);
```

Use this when inserting values for **all attributes** in the same order as the table.

Syntax 2: Insert into selected columns

```
INSERT INTO table_name (column1, column2, ...)  
VALUES (value1, value2, ...);
```

Use when **skipping** some columns (e.g., default or NULL values).

Example 1: Insert a Guardian Record

```
INSERT INTO GUARDIAN  
VALUES (444444444444, 'Amit Ahuja', 5711492685, 'G-35, Ashok Vihar, Delhi');
```

Result:

Query OK, 1 row affected (0.01 sec)

View the Inserted Record:

```
SELECT * FROM GUARDIAN;
```

GUID	GName	GPhone	GAddress
444444444444	Amit Ahuja	5711492685	G-35, Ashok Vihar, Delhi

Example 2: Insert Guardian with NULL Phone

```
INSERT INTO GUARDIAN (GUID, GName, GAddress)  
VALUES (333333333333, 'Danny Dsouza', 'S -13, Ashok Village, Daman');
```

Explanation: Skips GPhone. It will be stored as NULL.

Query OK, 1 row affected (0.03 sec)

Example 3: Insert into STUDENT with All Fields

```
INSERT INTO STUDENT  
VALUES (1, 'Atharv Ahuja', '2003-05-15', 444444444444);
```

Or

```
INSERT INTO STUDENT (RollNumber, SName, SDateofBirth, GUID)  
VALUES (1, 'Atharv Ahuja', '2003-05-15', 444444444444);
```

View Student Record:

```
SELECT * FROM STUDENT;
```

RollNumber	SName	SDateofBirth	GUID
1	Atharv Ahuja	2003-05-15	44444444444444

Example 4: Insert Student with NULL GUID (Allowed)

```
INSERT INTO STUDENT
VALUES (3, 'Taleem Shah', '2002-02-28', NULL);
```

Or using selected columns:

```
INSERT INTO STUDENT (RollNumber, SName, SDateofBirth)
VALUES (3, 'Taleem Shah', '2002-02-28');
```

GUID is a foreign key, so it can be NULL, but must match if present.

View Student Table:

```
SELECT * FROM STUDENT;
```

RollNumber	SName	SDateofBirth	GUID
1	Atharv Ahuja	2003-05-15	44444444444444
3	Taleem Shah	2002-02-28	NULL

Tip:

If the order of attributes is **not known**, always use **Syntax 2** (with column names).

Q: Can we insert two records with the same RollNumber? No, because RollNumber is a **PRIMARY KEY**, it must be **unique**.

Great! Let's now explore the next major section.

9.6 — SQL for Data Query

What is Data Querying? Once data is inserted into tables, we use SQL to **retrieve specific information**. This process is called **querying**.

SQL provides the SELECT command to:

- Display entire tables
- Fetch specific columns
- Apply conditions to filter rows

9.6.1 SELECT Statement Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example 1: Select All Columns

```
SELECT * FROM STUDENT;
```

Explanation:

* selects **all attributes** from the table.

Output:

RollNumber	SName	SDateofBirth	GUID
1	Atharv Ahuja	2003-05-15	44444444444444
3	Taleem Shah	2002-02-28	NULL

Example 2: Select Specific Columns

```
SELECT RollNumber, SName FROM STUDENT;
```

Output:

RollNumber	SName
------------	-------

1	Atharv Ahuja
3	Taleem Shah

Example 3: Apply a WHERE Clause

```
SELECT * FROM STUDENT
WHERE GUID IS NOT NULL;
```

Output:

RollNumber	SName	SDateofBirth	GUID
1	Atharv Ahuja	2003-05-15	44444444444444

9.6.2 — Querying Using Database OFFICE

Context Many organizations manage data using databases like OFFICE, which include **related tables** such as EMPLOYEE and DEPARTMENT. Each employee belongs to a department, and the **DeptId in EMPLOYEE is a foreign key** referencing DEPARTMENT.

EMPLOYEE Table (Sample Data — Table 9.8)

EmpNo	Ename	Salary	Bonus	DeptId
101	Aaliya	10000	234	D02
102	Kritika	60000	123	D01
103	Shabbir	45000	566	D01
104	Gurpreet	19000	565	D04
105	Joseph	34000	875	D03
106	Sanya	48000	695	D02
107	Vergese	15000	NULL	D01
108	Nachaobi	29000	NULL	D05

EmpNo	Ename	Salary	Bonus	DeptId
109	Daribha	42000	NULL	D04
110	Tanya	50000	467	D05

(A) Retrieve Selected Columns Query 1:

```
SELECT EmpNo FROM EMPLOYEE;
```

Output:

```
+-----+
| EmpNo |
+-----+
| 101 |
| 102 |
| 103 |
| 104 |
| 105 |
| 106 |
| 107 |
| 108 |
| 109 |
| 110 |
+-----+
```

Query 2:

```
SELECT EmpNo, Ename FROM EMPLOYEE;
```

Output:

```
+-----+-----+
| EmpNo | Ename   |
+-----+-----+
| 101 | Aaliya  |
| 102 | Kritika |
| 103 | Shabbir |
```

	104		Gurpreet	
	105		Joseph	
	106		Sanya	
	107		Vergese	
	108		Nachaobi	
	109		Daribha	
	110		Tanya	
+-----+-----+				

(B) Renaming Columns (Using Alias) Query:

```
SELECT Ename AS Name FROM EMPLOYEE;
```

Output:

+-----+		
	Name	
+-----+		
	Aaliya	
	Kritika	
	Shabbir	
	Gurpreet	
	Joseph	
	Sanya	
	Vergese	
	Nachaobi	
	Daribha	
	Tanya	
+-----+		

(C) Calculated Columns (Annual Income) Query:

```
SELECT Ename AS Name, Salary*12 FROM EMPLOYEE;
```

Output:

+-----+-----+		
	Name	Salary*12
+-----+-----+		

Aaliya		120000	
Kritika		720000	
Shabbir		540000	
Gurpreet		228000	
Joseph		408000	
Sanya		576000	
Vergese		180000	
Nachaobi		348000	
Daribha		504000	
Tanya		600000	
+-----+-----+			

To rename the calculated column:

Query (Using Alias with Space):

```
SELECT Ename AS Name, Salary*12 AS 'Annual Income' FROM EMPLOYEE;
```

Output:

+-----+-----+			
Name		Annual Income	
+-----+-----+			
Aaliya		120000	
Kritika		720000	
Shabbir		540000	
Gurpreet		228000	
Joseph		408000	
Sanya		576000	
Vergese		180000	
Nachaobi		348000	
Daribha		504000	
Tanya		600000	
+-----+-----+			

Note: Aliased column names with **spaces** should be enclosed in **single quotes** ('Annual Income').

(D) Using DISTINCT — To Avoid Duplicates Purpose: The DISTINCT clause is used to retrieve **unique values only**, avoiding repetitions.

Query: Display Unique Department IDs

```
SELECT DISTINCT DeptId FROM EMPLOYEE;
```

Output:

```
+-----+
| DeptId |
+-----+
| D01    |
| D02    |
| D04    |
| D03    |
| D05    |
+-----+
```

Without DISTINCT, repeated DeptIds would be shown for each employee.

(E) Using WHERE Clause — To Filter Rows Purpose: The WHERE clause helps select only those rows that satisfy a **specific condition**.

Query 1: Employees from Department D01

```
SELECT * FROM EMPLOYEE
WHERE DeptId = 'D01';
```

Output:

```
+-----+-----+-----+-----+-----+
| EmpNo | Ename   | Salary | Bonus | DeptId |
+-----+-----+-----+-----+-----+
| 102   | Kritika | 60000  | 123   | D01    |
| 103   | Shabbir | 45000  | 566   | D01    |
| 107   | Vergese | 15000  | NULL  | D01    |
+-----+-----+-----+-----+-----+
```

Query 2: Employees from Department D02

```
SELECT * FROM EMPLOYEE
WHERE DeptId = 'D02';
```

Output:

EmpNo	Ename	Salary	Bonus	DeptId
101	Aaliya	10000	234	D02
106	Sanya	48000	695	D02

Query 3: Employees with Salary > 25000

```
SELECT EmpNo, Ename, Salary
FROM EMPLOYEE
WHERE Salary > 25000;
```

Output:

EmpNo	Ename	Salary
102	Kritika	60000
103	Shabbir	45000
105	Joseph	34000
106	Sanya	48000
108	Nachaobi	29000
109	Daribha	42000
110	Tanya	50000

Query 4 Query: Display all details of employees from department D04 who earn more than 5000.

```
SELECT * FROM EMPLOYEE
WHERE Salary > 5000 AND DeptId = 'D04';
```

Output:

EmpNo	Ename	Salary	Bonus	DeptId
104	Gurpreet	19000	565	D04
109	Daribha	42000	NULL	D04

Query 5

Query: Display all employee records **except** Aaliya.

```
SELECT * FROM EMPLOYEE
WHERE NOT Ename = 'Aaliya';
```

Output:

EmpNo	Ename	Salary	Bonus	DeptId
102	Kritika	60000	123	D01
103	Shabbir	45000	566	D01
104	Gurpreet	19000	565	D04
105	Joseph	34000	875	D03
106	Sanya	48000	695	D02
107	Vergese	15000	NULL	D01
108	Nachaobi	29000	NULL	D05
109	Daribha	42000	NULL	D04
110	Tanya	50000	467	D05

Query 6 Query: Employees earning salary **between 20000 and 50000**.

```
SELECT Ename, DeptId
FROM EMPLOYEE
WHERE Salary BETWEEN 20000 AND 50000;
```

Output:

Ename	DeptId
Shabbir	D01
Joseph	D03
Sanya	D02
Nachaobi	D05
Daribha	D04
Tanya	D05

You can also write:

```
WHERE Salary >= 20000 AND Salary <= 50000;
```

Query 7 Query: Employees from departments **D01**, **D02**, or **D04** using OR:

```
SELECT * FROM EMPLOYEE
```

```
WHERE DeptId = 'D01' OR DeptId = 'D02' OR DeptId = 'D04';
```

Output:

EmpNo	Ename	Salary	Bonus	DeptId
101	Aaliya	10000	234	D02
102	Kritika	60000	123	D01
103	Shabbir	45000	566	D01
104	Gurpreet	19000	565	D04
106	Sanya	48000	695	D02
107	Vergese	15000	NULL	D01
109	Daribha	42000	NULL	D04

(F) Using IN — Multiple Matching Values Purpose: The IN operator allows checking if a value matches any value in a list.

Query: Employees from departments **D01**, **D02**, **D04**

```
SELECT * FROM EMPLOYEE
```

```
WHERE DeptId IN ('D01', 'D02', 'D04');
```

Output:

EmpNo	Ename	Salary	Bonus	DeptId
101	Aaliya	10000	234	D02
102	Kritika	60000	123	D01
103	Shabbir	45000	566	D01
104	Gurpreet	19000	565	D04
106	Sanya	48000	695	D02

107	Vergese	15000	NULL	D01
109	Daribha	42000	NULL	D04

IN is more concise than using multiple OR conditions.

Query: Select all employees **except** those working in departments D01 or D02.

```
SELECT * FROM EMPLOYEE
WHERE DeptId NOT IN ('D01', 'D02');
```

Output:

EmpNo	Ename	Salary	Bonus	DeptId
104	Gurpreet	19000	565	D04
105	Joseph	34000	875	D03
108	Nachaobi	29000	NULL	D05
109	Daribha	42000	NULL	D04
110	Tanya	50000	467	D05

(G) Using IS NULL — Check for Missing Data Purpose: To check whether a column contains a NULL (empty) value.

Query: Employees with no bonus

```
SELECT EmpNo, Ename, Bonus
FROM EMPLOYEE
WHERE Bonus IS NULL;
```

Output:

EmpNo	Ename	Bonus
107	Vergese	NULL
108	Nachaobi	NULL
109	Daribha	NULL

You **cannot** use = NULL. Always use IS NULL or IS NOT NULL.

Query: Select names of employees **with bonus** and working in department **D01**.

```
SELECT EName FROM EMPLOYEE
WHERE Bonus IS NOT NULL
AND DeptId = 'D01';
```

Output:

```
+-----+
| EName |
+-----+
| Kritika |
| Shabbir |
+-----+
```

(H) Using LIKE — Pattern Matching Purpose: The LIKE operator is used to match **text patterns** using wildcards:

- % → zero or more characters
- _ → exactly one character

Query: Names that start with 'S'

```
SELECT Ename FROM EMPLOYEE
WHERE Ename LIKE 'S%';
```

Output:

```
+-----+
| Ename |
+-----+
| Shabbir |
| Sanya |
+-----+
```

Query: Names that end with 'a'

```
SELECT Ename FROM EMPLOYEE
WHERE Ename LIKE '%a';
```

Output:

```

+-----+
| Ename  |
+-----+
| Aaliya  |
| Kritika |
| Sanya   |
| Daribha |
| Tanya   |
+-----+

```

Query: Names containing 'ya'

```

SELECT Ename FROM EMPLOYEE
WHERE Ename LIKE '%ya%';

```

Output:

```

+-----+
| Ename  |
+-----+
| Sanya   |
| Tanya   |
+-----+

```

Query: Select employees whose name starts with 'K'.

```

SELECT * FROM EMPLOYEE
WHERE Ename LIKE 'K%';

```

Output:

```

+-----+-----+-----+-----+-----+
| EmpNo | Ename  | Salary | Bonus | DeptId |
+-----+-----+-----+-----+-----+
| 102   | Kritika | 60000  | 123   | D01    |
+-----+-----+-----+-----+-----+

```

Query: Select employees whose name ends with 'a' and earn salary > 45000.

```
SELECT * FROM EMPLOYEE
WHERE Ename LIKE '%a'
AND Salary > 45000;
```

Output:

EmpNo	Ename	Salary	Bonus	DeptId
102	Kritika	60000	123	D01
106	Sanya	48000	695	D02
110	Tanya	50000	467	D05

Query: Select employees whose names are **exactly 5 letters** and have **'ANYA'** starting from second character.

```
SELECT * FROM EMPLOYEE
WHERE Ename LIKE '_ANYA';
```

Output:

EmpNo	Ename	Salary	Bonus	DeptId
106	Sanya	48000	695	D02
110	Tanya	50000	467	D05

Purpose of ORDER BY The ORDER BY clause in SQL is used to **sort the result set** of a query by:

- **One or more columns**
- In **ascending (ASC)** or **descending (DESC)** order

If no order is specified, it defaults to **ascending**.

Basic Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column_name [ASC|DESC];
```

Query – Sort by Salary (Ascending)

```
SELECT Ename, Salary
FROM EMPLOYEE
ORDER BY Salary;
```

Output:

Ename	Salary
Aaliya	10000
Vergese	15000
Gurpreet	19000
Joseph	34000
Nachaobi	29000
Shabbir	45000
Daribha	42000
Sanya	48000
Tanya	50000
Kritika	60000

ORDER BY Salary sorts rows by salary **increasingly**.

Query – Sort by Salary (Descending)

```
SELECT Ename, Salary
FROM EMPLOYEE
ORDER BY Salary DESC;
```

Output:

Ename	Salary
Kritika	60000
Tanya	50000
Sanya	48000
Shabbir	45000
Daribha	42000

Joseph	34000
Nachaobi	29000
Gurpreet	19000
Vergese	15000
Aaliya	10000

+-----+-----+

DESC keyword sorts values **from highest to lowest**.

9.7 — Data Updation and Deletion

9.7.1 Data Updation Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Caution: Missing the WHERE clause will update **all rows**.

Query — Update GUID for Roll Number 3

The student with roll number 3 is a sibling of student 5, so assign same GUID.

```
mysql> UPDATE STUDENT
-> SET GUID = 101010101010
-> WHERE RollNumber = 3;
```

Output:

```
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

You can verify with:

```
SELECT * FROM STUDENT;
```

Query — Update Multiple Columns in GUARDIAN Table

Change address and phone of guardian with GUID = 466444444666.

```
mysql> UPDATE GUARDIAN
-> SET GAddress = 'WZ - 68, Azad Avenue, Bijnour, MP',
-> GPhone = 9010810547
-> WHERE GUID = 466444444666;
```

Output:

Query OK, 1 row affected (0.06 sec)

Rows matched: 1 Changed: 1 Warnings: 0

After Update:

```
mysql> SELECT * FROM GUARDIAN;
```

GUID	GName	GPhone	GAddress
4444444444444	Amit Ahuja	5711492685	G-35, Ashok Vihar, Delhi
1111111111111	Baichung B.	7110047139	Flat 5, Darjeeling Appt., Shimla
101010101010	Himanshu Shah	9818184855	26/77, West Patel Nagar, Ahmedabad
3333333333333	Danny Dsouza	NULL	S -13, Ashok Village, Daman
4664444444666	Sujata P.	9010810547	WZ - 68, Azad Avenue, Bijnour, MP

9.7.2 Data Deletion Syntax:

```
DELETE FROM table_name
WHERE condition;
```

Caution: Omitting WHERE will delete **all records** from the table.

Query — Delete Student with Roll Number 2

```
mysql> DELETE FROM STUDENT
-> WHERE RollNumber = 2;
```

Output:

Query OK, 1 row affected (0.06 sec)

After Deletion:

```
SELECT * FROM STUDENT;
```

RollNumber	SName	SDateofBirth	GUID
1	Atharv Ahuja	2003-05-15	444444444444
3	Taleem Shah	2002-02-28	101010101010
4	John Dsouza	2003-08-18	333333333333
5	Ali Shah	2003-07-05	101010101010
6	Manika P.	2002-03-10	466444444666

9.8: Functions in SQL

Introduction Functions in SQL are predefined commands that perform operations on data. They are used for:

- Performing calculations
- Modifying strings
- Extracting date components
- Summarizing data

SQL functions are categorized into:

1. **Single Row Functions (Scalar Functions):** Operate on individual row values (return one result per row)
2. **Multiple Row Functions (Aggregate Functions):** Work on groups of rows (return one result per group)

Differences Between Single Row and Multiple Row Functions

Feature	Single Row Functions	Multiple Row Functions (Aggregate)
Operate On	Individual rows	Groups of rows
Return	One result per row	One result per group
Used In	SELECT, WHERE, ORDER BY	SELECT, HAVING
Examples	UCASE(), POWER(), ROUND(), DAY()	COUNT(), SUM(), AVG(), MAX(), MIN()

Database Used: CARSHOWROOM Tables:

- CUSTOMER
- EMPLOYEE
- INVENTORY
- SALE

1. Single Row (Scalar) Functions**A. Math Functions**

Function	Description	Example	Output
POWER(x,y)	x raised to the power y	SELECT POWER(2,3);	8
ROUND(n,d)	Rounds n to d decimals	SELECT ROUND(2912.564,1);	2912.6
MOD(a,b)	Remainder of a ÷ b	SELECT MOD(21, 2);	1

Examples with INVENTORY table:**1. GST Calculation:**

```
SELECT ROUND(12/100 * Price, 1) AS GST FROM INVENTORY;
```

2. EMI and Remaining Amount:

```
SELECT CarId, FinalPrice,
ROUND(FinalPrice - MOD(FinalPrice,1000)/10, 0) AS EMI,
MOD(FinalPrice, 10000) AS "Remaining Amount"
FROM INVENTORY;
```

3. Rounded Commission:

```
SELECT InvoiceNo, SalePrice, ROUND(Commission, 0)
FROM SALE;
```

B. String Functions

Function	Description	Example	Output
LCASE()	Converts to lowercase	SELECT LCASE('SQL');	sql
UCASE()	Converts to uppercase	SELECT UCASE('sql');	SQL
MID(str,p,n)	Extract substring from position p, length n	SELECT MID('Informatics', 3, 4);	form
LENGTH(str)	Returns string length	SELECT LENGTH('SQL');	3
LEFT(str,n)	Leftmost n characters	SELECT LEFT('Email', 4);	Emai
INSTR(s,sub)	Position of substring in string	SELECT INSTR('Informatics', 'ma');	6
TRIM()	Remove leading/trailing/specific chars	SELECT TRIM('.com' FROM 'abc@gmail.com');	abc@gmail

Examples with CUSTOMER table:

1. Convert name and email:

```
SELECT LOWER(CustName), UPPER(Email) FROM CUSTOMER;
```

2. Extract name from email:

```
SELECT LENGTH(Email), LEFT(Email, INSTR(Email, "@") - 1) FROM CUSTOMER;
```

3. MID() to extract area code:

```
SELECT MID(Phone, 3, 4) FROM CUSTOMER WHERE CustAdd LIKE '%Rohini%';
```

4. TRIM domain suffix:

```
SELECT TRIM('.com' FROM Email) FROM CUSTOMER;
```

C. Date Functions

Function	Description	Example	Output
NOW()	Returns system date and time	SELECT NOW();	2019-07-11
DATE()	Extracts date from datetime	SELECT DATE(NOW());	2019-07-11
DAY(date)	Returns day of month	SELECT DAY('2003-03-24');	24

Function	Description	Example	Output
MONTH(date)	Returns numeric month	SELECT MONTH('2003-11-28');	11
MONTHNAME(date)	Returns name of the month	SELECT MONTHNAME('2003-11-28');	November
YEAR(date)	Returns year	SELECT YEAR('2003-11-28');	2003
DAYNAME(date)	Returns weekday name	SELECT DAYNAME('2019-07-11');	Thursday

Examples with EMPLOYEE table:

1. DOJ components:

```
SELECT DAY(DOJ), MONTH(DOJ), YEAR(DOJ) FROM EMPLOYEE;
```

2. Formatted joining date:

```
SELECT DAYNAME(DOJ), DAY(DOJ), MONTHNAME(DOJ), YEAR(DOJ)
FROM EMPLOYEE WHERE DAYNAME(DOJ) != 'Sunday';
```

3. Day of birth:

```
SELECT EmpName, DAYNAME(DOB) FROM EMPLOYEE WHERE Salary > 25000;
```

2. Multiple Row (Aggregate) Functions

Function	Description	Example	Output
COUNT(*)	Total row count	SELECT COUNT(*) FROM MANAGER;	4
COUNT(col)	Count of non-NULL values	SELECT COUNT(MEMNAME) FROM MANAGER;	3
COUNT(DISTINCT col)	Unique values	SELECT COUNT(DISTINCT Model) FROM INVENTORY;	6
SUM()	Total of numeric column	SELECT SUM(Price) FROM INVENTORY;	4608733.00
AVG()	Average of numeric column	SELECT AVG(Price) FROM INVENTORY;	576091.625

Function	Description	Example	Output
MAX()	Highest value in column	SELECT MAX(Price) FROM INVENTORY;	673112.00
MIN()	Lowest value in column	SELECT MIN(Price) FROM INVENTORY;	355205.00

Examples with INVENTORY/SALE tables:

1. Count cars with model VXI:

```
SELECT COUNT(*) FROM INVENTORY WHERE Model = 'VXI';
```

2. Distinct models:

```
SELECT COUNT(DISTINCT Model) FROM INVENTORY;
```

3. Average price for LXI:

```
SELECT AVG(Price) FROM INVENTORY WHERE Model = 'LXI';
```

9.9 — GROUP BY Clause in SQL

Purpose of GROUP BY The GROUP BY clause is used to:

- Aggregate rows with the same values in specified columns
- Perform operations like COUNT, SUM, AVG, etc. on groups

Basic Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name;
```

Each column in the SELECT clause must either:

- Appear in the GROUP BY clause, or
- Be used inside an aggregate function

9.9.1 Examples Using GROUP BY Count Cars Sold Per Customer

```
SELECT CustID, COUNT(*) AS "Number of Cars"
FROM SALE
GROUP BY CustID;
```

Output:

CustID	Number of Cars
C0001	2
C0002	2
C0003	1
C0004	1

Displays how many cars each customer bought

Count Distinct Employees Handling Sales Per Model

```
SELECT Model, COUNT(DISTINCT EmpID)
FROM SALE, INVENTORY
WHERE SALE.CarID = INVENTORY.CarID
GROUP BY Model;
```

Output:

Model	COUNT(DISTINCT EmpID)
LXI	2
VXI	1

Combines two tables using JOIN condition (implicit syntax), then groups

Count Customers by Payment Mode

```
SELECT PaymentMode, COUNT(*)
FROM SALE
GROUP BY PaymentMode;
```

Output:

PaymentMode	COUNT(*)
Cheque	1
Credit Card	1
Online	1

Finds frequency of each payment type

Display Customers Who Bought More Than 1 Car

```
SELECT CustID, COUNT(*)
FROM SALE
GROUP BY CustID
HAVING COUNT(*) > 1;
```

Output:

CustID	COUNT(*)
C0001	2
C0002	2

Displays customers and number of cars bought, but only if more than 1 car was purchased.

Payment Modes Used More Than Once

```
SELECT PaymentMode, COUNT(PaymentMode)
FROM SALE
GROUP BY PaymentMode
HAVING COUNT(*) > 1
ORDER BY PaymentMode;
```

Output:

PaymentMode	Count(PaymentMode)
-------------	--------------------

Bank Finance	2
Credit Card	2

Filters only those payment modes used more than once, after grouping all SALE records by Payment-Mode.

9.10 – Operations on Relations

Introduction SQL allows operations that involve **multiple tables** (relations). These are called **set operations** and work when:

- Both tables have the **same number of columns**
- Corresponding columns have the **same data types** (domain compatibility)

Types of Relational Operations

Operation	Description
UNION (\cup)	Combines rows from two tables, removing duplicates
INTERSECT (\cap)	Returns rows common to both tables
MINUS (-)	Returns rows in the first table that are not in the second
CARTESIAN PRODUCT (\times)	All combinations of rows from two tables

Tables Used for Examples Table: DANCE

SNo	Name	Class
1	Aastha	7A
2	Mahira	6A
3	Mohit	7B
4	Sanjay	7A

Table: MUSIC

SNo	Name	Class
1	Mehak	8A
2	Mahira	6A
3	Lavanya	7A
4	Sanjay	7A
5	Abhay	8A

9.10.1 – UNION (\cup) Query:

```
SELECT * FROM DANCE
UNION
SELECT * FROM MUSIC;
```

Output (DANCE \cup MUSIC):

SNo	Name	Class
1	Aastha	7A
2	Mahira	6A
3	Mohit	7B
4	Sanjay	7A
1	Mehak	8A
3	Lavanya	7A
5	Abhay	8A

Combines students from both events without repeating common rows

9.10.2 – INTERSECT (\cap) Query:

```
SELECT * FROM DANCE
INTERSECT
SELECT * FROM MUSIC;
```

Output (DANCE \cap MUSIC):

SNo	Name	Class
2	Mahira	6A
4	Sanjay	7A

Finds students common to both DANCE and MUSIC

9.10.3 – MINUS (-) Query:

```
SELECT * FROM MUSIC
MINUS
SELECT * FROM DANCE;
```

Output (MUSIC - DANCE):

SNo	Name	Class
1	Mehak	8A
3	Lavanya	7A
5	Abhay	8A

Shows students participating only in MUSIC

9.10.4 – CARTESIAN PRODUCT (\times) Query:

```
SELECT * FROM DANCE, MUSIC;
```

Output (DANCE \times MUSIC):

- **Degree** (columns): 6
- **Cardinality** (rows): $4 \times 5 = 20$ rows

SNo	Name	Class	SNo	Name	Class
1	Aastha	7A	1	Mehak	8A
1	Aastha	7A	2	Mahira	6A
...
4	Sanjay	7A	5	Abhay	8A

Generates all possible combinations of students from both tables

CHAPTER 10 COMPUTER NETWORKS

CHAPTER NOTES

10.1: Introduction to Computer Networks

What is a Network? A **network** is a group of interconnected entities. These can be: - People (like in social networks) - Systems (like in mobile networks) - Devices (like in computer networks)

What is a Computer Network? A **Computer Network** is: - A system where two or more computers or computing devices are interconnected. - Allows devices to **share data and resources**.

Size and Scale

- **Small networks** may connect just a few computers in one room.
- **Large networks** can span buildings, cities, or countries.

What Makes Up a Network?

- **Hosts (nodes):** Computers, laptops, mobile phones, servers, etc.
- **Networking Devices:** Switches, routers, modems, etc.
- **Communication Media:**
 - **Wired** (like cables)
 - **Wireless** (like air/waves)

Data Transfer

- Data is broken into **packets** for transmission.
- Devices that **send, receive, store, or forward** data are called **nodes**.
- Examples of nodes include:
 - Modem, hub, switch, router
 - Computers, printers, phones

Benefits of Networks

- **Information exchange** through emails, calls, websites
- **Resource sharing:** Printers, files, and storage
- **Personal Networks:** Hotspots on mobile phones forming PANs (Personal Area Networks)

10.2: Evolution of Networking

Origin of Networking: ARPANET

- In the **1960s**, the **U.S. Department of Defence** initiated a project called **ARPANET** through its Advanced Research Projects Agency (ARPA).
- Objective: Connect **academic and research institutions** for **scientific collaboration**.
- **First message** was transmitted between:
 - **University of California, Los Angeles (UCLA)**
 - **Stanford Research Institute (SRI)**

Key Milestones in Networking Evolution

Here is a timeline of major events:

Year	Event
1961	Concept of ARPANET was proposed.
1969	ARPANET became functional (UCLA ↔ SRI).
1971	Email (E-mail) developed by Ray Tomlinson; introduced **@** symbol.
1974	Telenet launched as the first commercial version of ARPANET .
1982	The term “ Internet ” was coined.
1983	TCP/IP protocol introduced as the standard on ARPANET.
1986	NSFNET (National Science Foundation Network) extended Internet access to more users.
1990	WWW (World Wide Web) invented by Tim Berners-Lee using HTML, URL.
1997	First version of Wi-Fi (802.11 standard) introduced.

Try It Yourself Activity Suggestion:

Create a hotspot using a smartphone and connect other devices to experience a Personal Area Network.

10.3: Types of Networks

Computer networks vary in **size**, **coverage area**, and **data transfer speed**. Based on these characteristics, networks are broadly categorized as:

10.3.1 Personal Area Network (PAN)

- A **smallest-sized network** used for connecting **personal devices**.
- Range: **Up to 10 meters**.
- Can be **wired** (e.g., USB cable between phone and laptop) or **wireless** (e.g., Bluetooth between smartphones).

Examples: - Smartphone connected to a printer via Bluetooth. - Laptop connected to phone using a USB cable.

10.3.2 Local Area Network (LAN)

- Connects devices like computers, phones, tablets, and printers **within a limited area**.
- Common in **homes, schools, labs, offices, or campuses**.
- Uses **Ethernet cables, fiber optics, or Wi-Fi**.

Features: - Range: **Up to 1 km**. - High-speed data transfer: - **Ethernet:** 10 Mbps - **Gigabit Ethernet:** 1000 Mbps - **Secure:** Only authorized users can access shared resources. - Devices can **share printers, access servers, and transfer files**.

10.3.3 Metropolitan Area Network (MAN)

- Covers a **city or a town**.
- **Data transfer speed:** Comparable to LAN but slightly lower.
- Connects multiple LANs together.

Examples: - **Cable TV networks** - **Broadband services** in a city

Range: **Up to 30–40 km**

10.3.4 Wide Area Network (WAN)

- Largest network that spans **countries and continents**.
- Connects multiple **LANs and MANs** using **wired/wireless** media.

Examples: - **The Internet:** Largest example of a WAN - Large business or government networks linking offices globally

Think About This: If you're accessing your **bank account** from another city, what type of network is enabling that access?

It's a **WAN**.

10.4: Network Devices

To establish communication and functionality in a network, various **hardware devices** are used. Here's a breakdown of key devices:

10.4.1 Modem

- **Full Form:** *MOdulator-DEModulator*
- **Purpose:** Converts digital data (used by computers) to analog signals (used for transmission) and vice versa.
- **Usage:**
 - At sender's end: Converts digital → analog (modulation)
 - At receiver's end: Converts analog → digital (demodulation)

Used in: Internet connections via telephone lines

10.4.2 Ethernet Card (NIC – Network Interface Card)

- **Installed in computers** to enable wired networking.
- Each NIC has a **unique MAC address** to identify the device on the network.
- Supports data transfer from **10 Mbps to 1 Gbps**.
- Connected to the network using an **Ethernet cable**.

10.4.3 RJ-45 Connector

- **Full Form:** Registered Jack 45
- An **8-pin connector** used with Ethernet cables.
- Standard interface used to plug network cables into devices.

10.4.4 Repeater

- A device used to **regenerate weak or lost signals**.
- Signals weaken after traveling ~100 meters.
- **Repeaters** amplify and resend these signals to extend the transmission range.

10.4.5 Hub

- A **basic device** used to connect multiple computers in a network.
- **Broadcasts** incoming data to **all connected devices**.
- **Limitation**: If multiple data packets arrive simultaneously, they **collide**.

10.4.6 Switch

- **Smarter than a hub**.
- Sends data **only to the intended recipient** using the destination address in the packet.
- **More secure and efficient** than hubs.
- Can **filter corrupted data** and ask for retransmission.
- Commonly used in **homes and offices** to create LANs.

10.4.7 Router

- Connects **different networks**, including **LAN to the Internet**.
- Can analyze, alter, and **repackage data** as per the network type.
- May be **wired or wireless**.
- Home routers often work as a **modem + router + switch**.

10.4.8 Gateway

- Acts as a **gate** between a local network and the outside world (e.g., the Internet).
- All incoming/outgoing data **passes through the gateway**.
- May be implemented using **software, hardware, or both**.
- Often combined with **firewalls** for security.

Example: Your home Internet provider acts as the **gateway** to the Internet.

10.5: Networking Topologies

What is a Topology?

- A **topology** is the **arrangement or layout** of computers and devices in a network.
- It determines how devices are **interconnected** and how **data flows**.

There are **five common types** of network topologies:

10.5.1 Mesh Topology

- **Every device is connected to every other device** in the network.
- High **reliability** and **security**.
- Multiple devices can communicate **simultaneously**.
- **Disadvantage:** Complex and expensive due to the high number of cables and connections.

To connect n devices fully: [Number of connections = $\frac{n(n-1)}{2}$]

10.5.2 Ring Topology

- Each node is connected to **two other nodes**, forming a **ring**.
- Data travels in **one direction** (either clockwise or counterclockwise).
- **Limitation:** If one node fails, it can affect the whole network.

10.5.3 Bus Topology

- All nodes are connected to a **single communication line** (bus or backbone).
- **Cost-effective** and **easy to install**.
- Data travels along the bus and is available to **all connected devices**.
- **Limitation:** If the bus fails, the **entire network goes down**.
- Less secure and less reliable compared to mesh or star.

10.5.4 Star Topology

- Each device is connected to a **central hub or switch**.
- Data is routed **through the central node**.
- **Advantages:**
 - Easy to troubleshoot and expand.
 - Failure of one device does **not affect others**.
- **Limitation:** If the **central hub fails**, the **whole network is down**.

10.5.5 Tree or Hybrid Topology

- A **hierarchical** structure combining two or more topologies (e.g., star + bus).
- Common in **WANs** where **multiple LANs** are connected.
- Suitable for **large and scalable networks**.
- Data reaches the centralized device first, then travels to branches and nodes.

Reflection:

How do **bus** and **ring** topologies behave when a node goes down? - In **bus**: The whole network may fail.
- In **ring**: Data cannot be passed beyond the failed node.

10.6: Identifying Nodes in a Networked Communication

To ensure data reaches the correct destination in a network, each **node (device)** must have a **unique identity**. Two primary identifiers are used:

10.6.1 MAC Address (Media Access Control Address)

- **Definition:** A **permanent, unique** hardware address assigned to a device's **Network Interface Card (NIC)**.
- Also called: **Physical Address** or **Hardware Address**.
- **Burned into the NIC** during manufacturing and **cannot be changed**.

Structure:

- **12-digit hexadecimal** number (48 bits)
- First 6 digits: **Organisational Unique Identifier (OUI)** – identifies the manufacturer.
- Last 6 digits: **Serial Number** – uniquely identifies the device.
- **Example:** 00:A0:C9:14:C8:29

Use: Physically identifies a device within a **local network**.

10.6.2 IP Address (Internet Protocol Address)

- **Definition:** A **logical address** assigned to a device on a network that uses the **Internet Protocol**.
- Can **change** if the device connects to a different network (unlike MAC).
- **Used globally** to identify devices over the Internet.

Versions:**1. IPv4 (Internet Protocol Version 4)**

- 32-bit address
- Written as four decimal numbers separated by dots (e.g., 192.168.0.178)
- Each part ranges from 0–255
- Limited to ~4.3 billion addresses

2. IPv6 (Internet Protocol Version 6)

- 128-bit address
- Written in **8 groups of hexadecimal numbers** separated by colons
- Example: 2001:CDBA:0000:0000:0000:0000:3257:9652
- Designed to support the **growing number of devices** on the Internet

Activity Suggestion: Explore how to find the **MAC address** of your computer system (e.g., using `ipconfig /all` in Windows or `ifconfig` in Linux/macOS).

10.7: Internet, Web, and the Internet of Things (IoT)

10.7.1 The Internet

- The **Internet** is a **global network** connecting billions of devices—like desktops, laptops, smart-phones, tablets, routers, and smart appliances.
- These devices communicate using **standardized protocols**, allowing data to travel around the world.

Examples of Internet-enabled devices: - Computers, printers, scanners - Smart devices: TVs, ACs, refrigerators, lights, fans, security cameras - Modern IoT devices: drones, smart vehicles, door locks

How the Internet is Structured

1. Your device connects to a **modem/router**.
2. The modem connects to a **local ISP (Internet Service Provider)**.
3. The ISP connects to a **national or regional network**.
4. These regional networks connect together to form the **Internet backbone**.

So, the Internet is a **multi-layered structure** of interconnected networks.

10.7.2 The World Wide Web (WWW)

- The **WWW** is a **collection of interlinked web pages and resources**, stored on web servers and accessed over the Internet.
- Invented by **Sir Tim Berners-Lee** in **1990**.
- It allows users to **access, share, and view content** globally.

Key Technologies Behind the Web:

1. **HTML (HyperText Markup Language):**
 - Language used to create **webpages**.

- Ensures standard formatting and readability across systems.

2. URI (Uniform Resource Identifier):

- A **unique address** for each web resource.
- Often called **URL (Uniform Resource Locator)**.
- Example: `https://www.ncert.nic.in/textbook/textbook.htm`

3. HTTP/HTTPS (HyperText Transfer Protocol):

- A set of **rules** for retrieving linked pages from the web.
- **HTTPS** is the **secure** version of HTTP.

Internet ≠ Web: - **Internet:** The underlying network infrastructure (hardware + communication). - **Web:** The **content** (documents, multimedia, etc.) shared and retrieved over the Internet.

Discussion Prompt: What areas of life (like traffic management or health monitoring) do you think **IoT** can improve?

10.8: Domain Name System (DNS)

Why DNS is Needed Every **website or server** on the Internet is stored on a device with an **IP address**. But IP addresses are strings of numbers, which are **difficult to remember**.

Problem: Would you rather type `164.100.60.233` or `ncert.nic.in`?

That's where **Domain Names** come in!

What is a Domain Name?

- A **user-friendly name** assigned to an IP address.
- Makes it easier to **access websites**.
- Works like a **phonebook**:
 - Instead of remembering phone numbers, we remember names.

Example:

Domain Name	IP Address
<code>ncert.nic.in</code>	<code>164.100.60.233</code>
<code>cbse.nic.in</code>	<code>164.100.107.32</code>

Domain Name	IP Address
mhrd.gov.in	164.100.163.45
wikipedia.org	198.35.26.96

10.8.1 DNS Server What it does:

- Converts a **domain name** → **IP address**.
- This process is called **Domain Name Resolution**.

How it works:

1. You type a URL in your browser (e.g., `www.ncert.nic.in`).
2. Your browser contacts a **DNS server** to find the IP address.
3. The DNS server replies with the correct IP (e.g., `164.100.60.233`).
4. Your browser uses the IP to retrieve the webpage.

Illustration:

User → HTTP request to DNS Server → Receives IP address → Connects to server

DNS Server Hierarchy

- The DNS system is **hierarchical**.
- At the top are **13 root servers** named **A to M**.
- These root servers are distributed globally:
 - 10 in the **US**
 - 1 in **London**
 - 1 in **Stockholm**
 - 1 in **Japan**

Maintained by: Internet Assigned Numbers Authority (**IANA**)

CHAPTER 11 DATA COMMUNICATION

CHAPTER NOTES

11.1 Concept of Communication

What is Data Communication?

- **Data Communication** = Data + Communication.
- **Data** can be:
 - Text
 - Image
 - Audio
 - Video
 - Multimedia
- **Communication** refers to the act of sending or receiving this data.
- Therefore, **Data Communication** means the **exchange of data between two or more devices** that are connected via a network.

Requirements for Data Communication

- Devices must be:
 - Networked/Connected
 - Capable of **sending** and **receiving** data
- Data is transferred over a **communication medium** (wired or wireless).

Examples of Devices

- Personal computers
- Laptops
- Mobile phones
- Printers
- Servers
- Switches

11.2: Components of Data Communication

Any successful data communication involves the following **five essential components**:

1. Sender

- Device that **sends data**.
- Can be: Computer, mobile phone, smartwatch, walkie-talkie, video camera, etc.

2. Receiver

- Device that **receives data**.
- Can be: Printer, laptop, mobile phone, TV, etc.
- Both sender and receiver are also called **nodes** in a network.

3. Message

- The **data or information** being exchanged.
- Can be in the form of:
 - Text
 - Numbers
 - Images
 - Audio
 - Video
 - Multimedia

4. Communication Medium

- The **path** through which the message travels.
- Also known as **transmission media**.
- Can be:
 - Wired: Ethernet cable, TV cable, telephone cable
 - Wireless: Satellite, microwaves, etc.

5. Protocol

- A set of **rules** followed during communication.
- Ensures **successful and reliable** data transmission.
- Examples: Ethernet, HTTP

11.3: Measuring Capacity of Communication Media

In data communication, the **transmission medium** (also called a **channel**) has a **capacity**, which determines how much data it can carry.

This capacity is measured using two important concepts:

11.3.1 Bandwidth

- **Definition:** Bandwidth is the **range of frequencies** available for data transmission through a channel.
- It is the **difference between the maximum and minimum frequencies** that a channel can carry.
- **Higher bandwidth = higher data transfer rate**
- **Unit of measurement: Hertz (Hz)**

Conversion Units:

- 1 Kilohertz (KHz) = 1,000 Hz
- 1 Megahertz (MHz) = 1,000 KHz = 1,000,000 Hz

11.3.2 Data Transfer Rate

- **Definition:** Data Transfer Rate (also called **bit rate**) is the **number of bits transmitted per second** between the source and destination.
- **Measured in: Bits per second (bps)**

Common Units:

Unit	Value
1 Kbps	2^{10} bps = 1,024 bps
1 Mbps	2^{20} bps = 1,024 Kbps
1 Gbps	2^{30} bps = 1,024 Mbps
1 Tbps	2^{40} bps = 1,024 Gbps

NOTE: - **MBps** (Megabytes per second) \neq **Mbps** (Megabits per second)

- 1 Byte = 8 bits
- MBps is 8 times larger than Mbps.

Example 11.1 Q: A user uploads 10 pages in 20 seconds.
Each page has 1600 characters, and each character = 8 bits.
What is the required data rate?

Solution:

- Total bits = $10 \times 1600 \times 8 = 128,000$ bits
- Time = 20 seconds
- Data rate = $128,000 / 20 = 6,400$ bps

11.4: Types of Data Communication

Data is exchanged as **signals** between two or more **devices/nodes** over communication channels. Depending on the direction of data flow, communication can be of three types:

11.4.1 Simplex Communication

- **One-way (Unidirectional)** communication.
- Only the **sender** transmits; the **receiver** only receives.
- The channel is **fully used** by the sender only.

Analogy:

- Like a **one-way street** — vehicles (data) can move in only one direction.

Examples:

- **Keyboard to computer**
- **Speaker output**
- **IoT remote control** of home appliances (e.g. turning on AC from office)

11.4.2 Half-Duplex Communication

- **Two-way (Bidirectional)** communication — **but not at the same time**.
- Both devices can send and receive data, **but one at a time**.
- The direction alternates — like a **narrow one-way bridge** used by vehicles from both sides **one at a time**.

Examples:

- **Walkie-talkie:** Press-to-talk — while one speaks, others listen.

11.4.3 Full-Duplex Communication

- **Two-way (Bidirectional)** communication where **both devices can send and receive simultaneously**.
- Channel capacity is **shared** or **duplicated**.

How It Works:

- Either:
 - Two separate channels: One for sending, one for receiving
 - OR**
 - One channel split to handle both directions at the same time

Examples:

- **Mobile phones**
- **Landline telephones**
- **Online video conferencing**

Analogy:

- Like a **two-lane road** — traffic moves in both directions **at the same time**.

11.5: Switching Techniques

When multiple devices exist in a network, we need a way to establish communication between **sender and receiver** efficiently. Instead of creating permanent links (which are expensive in large networks), **switching techniques** are used.

Switching involves temporarily routing data through various network nodes to reach the destination.

There are **two major types**:

11.5.1 Circuit Switching

- A **dedicated path** is created **before communication starts**.
- The entire communication session uses this **same path**.
- Once the call or data transfer ends, the path is released.

Real-life Example:

- **Old landline phone systems**

When you make a call, a **physical path** is set from your phone to the receiver's phone.

Features:

- Reliable
- All data follows the same path
- Useful for **real-time communication**

Limitation:

- Path is **reserved**, even if no data is being transferred — **wastes resources**

11.5.2 Packet Switching

- Data/message is **divided into small packets**.
- Each packet may take **different routes** to the destination.
- At the destination, packets are **reassembled** in the correct order.

Packet Structure:

- **Header**: Contains destination address and other control information
- **Payload**: The actual data/message

Features:

- **Efficient** — channel used only during transmission
- **Flexible** — can reroute packets if a path is busy
- Suitable for **data-heavy applications** like the Internet

Examples:

- **Internet**
- **Email**
- **Streaming services**

Comparison: Circuit vs Packet Switching

Feature	Circuit Switching	Packet Switching
Path Setup	Before communication	Not required
Data Path	Fixed	Variable
Resource Usage	Reserved throughout session	Shared, released after use
Delay	Low after setup	Can vary per packet
Suitability	Voice calls	Data communication (Internet, VoIP)

11.6: Transmission Media

What is a Transmission Medium? A **transmission medium** is **anything that can carry signals or data** from a sender (source) to a receiver (destination).

Examples:

- **Wires, cables** (in case of fans, bulbs, etc.)
- **Air** (when two people talk or devices communicate wirelessly)

Types of Transmission Media Transmission media are broadly classified into:

1. **Guided (Wired) Media**
2. **Unguided (Wireless) Media**

11.6.1 Wired (Guided) Transmission Media

In guided media, data travels through a **physical path** like wires or cables.

(A) Twisted Pair Cable

- Made of **two insulated copper wires** twisted together like a DNA helix.
- Often bundled into larger cables with many such pairs.
- **Twisting reduces electrical interference** from nearby cables.
- Used in **telephones** and **LANs**.

Types:

- **UTP (Unshielded Twisted Pair)**
- **STP (Shielded Twisted Pair)** — has extra shielding for noise protection

(B) Coaxial Cable

- Has a **central copper conductor** surrounded by:
 - **Insulating material**
 - **Outer conductor (copper mesh)** for shielding
 - **Plastic covering**
- **More bandwidth** and better shielding than twisted pair
- Used in:
 - **Cable TV**
 - **High-frequency signals over longer distances**

(C) Optical Fiber Cable

- Transmits data using **light signals**.
- Structure:
 - **Core**: Thin glass strand that carries light
 - **Cladding**: Less dense glass to reflect light
 - **Outer jacket**: PVC or Teflon
- Uses **refraction** to guide light through the fiber

Advantages:

- High **bandwidth**
- Long-distance transmission
- Immune to **electromagnetic interference**

Disadvantages:

- **Expensive**
- **Unidirectional** (requires two cables for full-duplex communication)

11.6.2 Wireless (Unguided) Transmission Media

Here, data is transmitted using **electromagnetic waves** through the **air** via antennas.

Frequency Range:

- 3 KHz to 900 THz
- Divided into:
 - **Radio Waves**
 - **Microwaves**
 - **Infrared Waves**
 - **Light Waves** (Visible spectrum)

Electromagnetic Waves Overview**Radio Waves (3 KHz – 1 GHz)**

- **Omnidirectional**
- Can penetrate walls
- Used in:
 - AM/FM Radio
 - TV
 - Cordless phones

Microwaves (1 GHz – 300 GHz)

- **Unidirectional**
- Needs **line-of-sight**
- Used in:
 - Satellite communication
 - Radar
 - Mobile phones

Infrared Waves (300 GHz – 400 THz)

- Very high frequency
- Can't pass through walls
- Used in:
 - TV remotes
 - Bluetooth devices
 - Mobile-to-printer or mobile-to-mobile connections

11.6.3: Wireless Technologies

Wireless technologies enable **devices to communicate without physical cables**, using electromagnetic waves over the air. Here are the major types discussed:

(A) Bluetooth

- **Short-range** wireless technology.
- Connects devices like:
 - Mobile phones
 - Headphones
 - Printers
 - Keyboards
 - Mice
 - Computers

Key Features:

- Uses **2.4 GHz** unlicensed frequency band.
- Speed: **1 to 2 Mbps**
- Range: **Up to 10 meters**
- Devices must have a **Bluetooth transceiver chip**.

Network Type:

- Bluetooth devices form a **Personal Area Network (PAN)** called a **Piconet**.
- Works in **master-slave configuration**:
 - 1 Master + up to 7 active slave devices.
 - Total up to **255 devices** can be part of a piconet, but only 8 are active at a time.

(B) Wireless LAN (Wi-Fi)

- A **Local Area Network (LAN)** that operates wirelessly.
- Based on IEEE standard **802.11** (commonly called Wi-Fi).

Key Components:

- **Access Point (AP)**: Connects wireless devices to a wired network.
- Devices (laptops, phones, tablets, etc.) connect to the AP to:

- Access the Internet
- Communicate with each other wirelessly

Advantages:

- **Mobility:** Users can move around while staying connected.
- Can be used to **extend or replace wired networks**.
- Popular in **homes, schools, offices, and public areas** (cafes, libraries).

(C) WiMAX (Worldwide Interoperability for Microwave Access)

- Similar to Wi-Fi but with a **larger coverage area** and **higher data rates**.
- Used in **MAN (Metropolitan Area Network)** applications.

Key Features:

- Provides **high-speed broadband** over large areas.
- Suitable for connecting **multiple locations** across a city or town.
- Covers distances **beyond the range of Wi-Fi**.

11.7: Mobile Telecommunication Technologies

Mobile telecommunication allows **wireless communication while on the move** — this includes calls, text messaging, and Internet access. With time, mobile network technologies have progressed through various **generations (G)**.

1G (First Generation) – 1982

- **Analog signals** used to transmit **only voice calls**.
- Very basic — poor voice quality and no data transmission.
- Limited capacity and security.

2G (Second Generation) – 1991

- Switched from analog to **digital signal transmission**.
- **Improved call quality** and **better security**.
- Introduced **SMS (Short Message Service)** and **MMS (Multimedia Messaging Service)**.
- More users could communicate simultaneously.

3G (Third Generation) – Introduced ~2001

- Supported **both voice and data** services.
- Enabled:
 - Internet browsing
 - Video calling
 - Mobile apps
- Voice and Internet used **the same radio towers**.

4G (Fourth Generation)

- Much **faster** than 3G.
- Enhanced Internet experiences with:
 - HD streaming
 - Video conferencing
 - Online gaming
 - Interactive multimedia services
- Introduced technologies like **VoIP**, **VoLTE**, and **WiMAX**.
- Revolutionized mobile Internet by offering **broadband-like speeds**.

5G (Fifth Generation) – Current / Emerging

- **Under development** and being rolled out in stages.
- Aimed at ultra-fast Internet: **speeds in Gbps**.
- Will support:
 - **IoT (Internet of Things)**
 - **M2M (Machine-to-Machine)** communication
 - **Smart cities**
 - **Connected vehicles**
- Designed to connect **billions of smart devices** with **minimal latency**.

Key Concepts:

- **IoT**: Enables devices (like smart bulbs, ACs, cars) to communicate over the Internet.
- **M2M**: Direct device-to-device communication without human intervention.

11.8: Protocol

What is a Protocol? A **protocol** is a set of **rules and standards** that define how devices communicate over a network.

It ensures proper: - **Data transmission - Error handling - Authentication - Synchronization**

In essence, **without protocols**, communication would be chaotic and unreliable.

11.8.1 Need for Protocols

Protocols handle critical functions like:

Flow Control

- Regulates data speed between sender and receiver.
- Example: If sender is faster than the receiver, data may get lost unless sender slows down.
 - **Example case:**
 - * Sender: 1024 Mbps
 - * Receiver: 512 Mbps
 - * Receiver needs to inform sender to adjust speed.

Access Control

- Decides **which device gets to send data** on a shared medium to avoid data collisions.

Addressing

- Ensures that data reaches the **correct device** (node) on a network.

Error Checking & Packet Handling

- Ensures **complete and correct** data reaches the receiver.
- Helps with:
 - Packet order
 - Missing data
 - Data reassembly

11.8.2 Common Protocols in Use

(A) HTTP – HyperText Transfer Protocol

- Used for **accessing web pages** on the Internet.
- **Client-server model**:
 - Browser (client) sends request
 - Web server responds
- Developed by **Tim Berners-Lee** at CERN in 1989.
- Works over **TCP (Transmission Control Protocol)**.
- Handles web commands like loading a webpage or clicking a link.

(B) FTP – File Transfer Protocol

- Used to **transfer files** between two systems.
- Also follows the **client-server model**.
- Optionally requires **username & password**.
- Handles:
 - Differences in file formats
 - Directory structures
 - File naming convention

(C) PPP – Point-to-Point Protocol

- Used for **direct link** between two devices (like routers or modems).
- Establishes a **duplex connection** (two-way).
- Handles:
 - Device **authentication**
 - **Packet order**
 - **Error reporting and recovery**

(D) SMTP – Simple Mail Transfer Protocol

- Used for **sending emails**.
- Uses **email headers** to identify recipients.
- Adds outgoing emails to a **queue**, sends them, and removes recipients as delivery is confirmed.
- Delivers emails to **receiver's mailbox** via an **SMTP receiver program**.

(E) TCP/IP – Transmission Control Protocol / Internet Protocol

- The **backbone protocol of the Internet**.
- Uses **client-server communication** model.

IP (Internet Protocol) - Assigns **unique addresses** (IP addresses) to each device. - Routes packets through the Internet.

TCP (Transmission Control Protocol) - **Breaks data** into packets. - **Ensures delivery** and correct order.
- Handles **packet reassembly** at the receiver.

Recap:

- **HTTP**: Web browsing
- **FTP**: File transfer
- **PPP**: Direct device link
- **SMTP**: Email sending
- **TCP/IP**: General internet communication (all types)

CHAPTER 12 SECURITY ASPECTS

CHAPTER NOTES

12.1 - Threats and Prevention

1. Ideal vs. Practical Security

- **Isolation as Ideal**
 - A computer with no external links cannot suffer network threats.
- **Connectedness as Reality**
 - Modern computing requires connectivity; complete isolation is not feasible.

2. Definition of Network Security

- **Goal:** Protect devices **and** data from illegitimate access or misuse.
- **Threats:** Any method exploiting vulnerabilities or weaknesses in a network or communication system to cause harm or damage one's reputation.

3. Key Terminology

- **Threat:** Ways to exploit network weaknesses.
- **Vulnerability:** A weakness in a system that can be exploited.
- **Prevention:** Measures taken to guard against threats.

4. Why Prevention Matters

- **Connected devices introduce risk.**
- **Threats exploit vulnerabilities** to damage hardware, steal data, or harm reputation.
- **Prevention** focuses on identifying and mitigating these threats before they succeed.

12.2 - Malware

Definition:

- Malware = MALicious softWARE.
- Software developed to damage hardware, steal data, or cause trouble for the user.
- Causes financial damages worth billions annually.

Common Types:

- Viruses
- Worms
- Ransomware
- Trojans
- Spyware
- Adware
- Keyloggers

12.2.1 Virus

- Coined by Fred Cohen in 1985.
- Behaves like a biological virus: infects by copying/inserting its code into executable files.
- Remains dormant until the infected file is executed.
- Hampering resources: CPU time, memory, files, sensitive data.
- Common motives: steal passwords/data, corrupt files, spam contacts, control machines.
- Examples: CryptoLocker, ILOVEYOU, MyDoom, Sasser, Netsky, Slammer, Stuxnet.

12.2.2 Worms

- Standalone programs; no host program needed.
- Replicates and spreads automatically through networks (no human triggering).
- Causes unexpected or damaging behavior on infected systems.
- Examples: Storm Worm, Sobig, MSBlast, Code Red, Nimda, Morris Worm.

12.2.3 Ransomware

- Targets user data by blocking access or threatening publication.
- May encrypt data, demanding ransom (often in Bitcoin).
- Example: WannaCry (May 2017) infected ~200,000 computers across 150 countries; encrypted data and demanded Bitcoin ransom.

12.2.4 Trojan

- Named after the wooden horse of Troy.
- Appears as legitimate software to trick users into installing.
- Does **not** self-replicate or infect other files; spreads via user action (e.g., email attachments).
- Often creates backdoors for remote access.

12.2.5 Spyware

- Spies on users/organizations by gathering information without knowledge.
- Records and sends data (e.g., browsing habits, credentials) to external entities.
- Used by advertisers or attackers to capture sensitive information (credit cards, passwords).

12.2.6 Adware

- Designed to generate revenue via online advertisements.
- Displays pop-ups, webpages, or installation-screen ads.
- Uses pay-per-click or impression models.
- Generally annoying but can pave the way for more dangerous malware via unsafe links.

12.2.7 Keyloggers

- Can be software or hardware.
- Records keystrokes to create logs of keyboard usage.
- Sends logs externally, exposing passwords, emails, private conversations.
- **Countermeasure:** Use an online virtual keyboard (randomized layout) instead of fixed on-screen keyboard to thwart software keyloggers.
 - **On-screen keyboard:** fixed QWERTY layout (vulnerable).
 - **Online virtual keyboard:** randomizes keys each use (more secure).

12.2.8 Modes of Malware Distribution

1. **Downloaded from the Internet:** disguised as free software/files.
2. **Spam Email:** unsolicited emails with malicious links or attachments.
3. **Removable Storage Devices:** infects USB drives, SSD cards, etc., then transfers to other systems.
4. **Network Propagation:** some malware (e.g., worms) spread through network connections.

12.2.9 Combating Malware Common Infection Signs:

- Frequent pop-ups prompting downloads.
- Changed browser homepage.
- Mass emails sent from your account.
- Slow performance, crashes.
- Unknown startup programs.
- Programs opening/closing automatically.
- Sudden storage-space loss, random messages or sounds.
- Files appearing/disappearing without knowledge.

Preventive Measures:

- Use and regularly update antivirus/anti-malware software.
- Configure browser security settings.
- Check for lock icon (HTTPS) when making payments.
- Avoid pirated or unlicensed software; prefer FOSS.

- Apply manufacturer updates/patches.
- Regularly back up important data.
- Enforce firewall protection.
- Avoid entering sensitive data on unknown/public computers or networks.
- Don't click links or download attachments from unsolicited emails.
- Scan removable media with antivirus before use.
- Never share passwords/PINs.
- Remove unrecognized programs.
- Close pop-ups via the window's "X" rather than in-ad buttons; cancel installations immediately if launched.

12.3 - Antivirus

Definition:

- **Antivirus** (also called anti-malware) is software initially developed to detect and remove viruses.
- Evolved to include prevention, detection, and removal of a wide range of malware.

12.3.1 Methods of Malware Identification Antivirus programs employ multiple techniques to identify malicious software:

1. Signature-Based Detection

- Uses a database of known virus signatures called the **Virus Definition File (VDF)**.
- VDF must be updated continuously; outdated VDF renders antivirus ineffective against new threats.
- Limitations: Cannot detect polymorphic malware (changes its signature) or code-encrypted malware.

2. Sandbox Detection

- Executes suspect applications/files in a virtual "sandbox" environment.
- Observes behavioral "fingerprint" for malicious actions.
- Safer but slower, as it isolates unknown code from actual system resources.

3. Data Mining Techniques

- Applies data mining and machine learning to classify files as benign or malicious.

4. Heuristics

- Compares source code of suspect programs against known malware patterns in a heuristic database.
- Flags code as malicious if it shares a significant pattern match with known malware.

5. Real-Time Protection

- Runs in the background, monitoring active memory for suspicious behavior.
- Detects malware that is dormant at startup but activates later.

12.4 - Spam

- **Definition:**

- Broad term for unwanted digital content across email, messaging, forums, chat, and advertising.
- Most recognized form is **email spam**.

- **How It Works:**

- Individuals or organizations compile or purchase mailing lists.
- Send unsolicited advertisement links or invitations en masse.
- Fills recipients' inboxes with junk, aiming to trick users into purchases or malware downloads.

- **Countermeasures in Email Services:**

- Modern services (e.g., Gmail, Hotmail) use automatic spam-detection algorithms to filter unsolicited mail.
- Users can manually mark undetected spam as "spam," training the filter to block similar future messages.

12.5 - HTTP vs HTTPS

- **HTTP (Hyper Text Transfer Protocol):**

- Protocol governing data transmission over the World Wide Web.
- Sends information “in the clear” (unencrypted), making it vulnerable to interception by attackers.
- **HTTPS (Hyper Text Transfer Protocol Secure):**
 - Variant of HTTP that **encrypts** data before transmission.
 - Decrypts data at the receiver’s end to recover the original content.
 - Requires an SSL digital certificate on the server.
- **Use Cases:**
 - **HTTP:** Suitable for websites with purely public information (e.g., news portals, blogs).
 - **HTTPS:** Essential for sites handling personal data, banking credentials, passwords.
 - Always look for “https://” (and a lock icon) in the address bar when entering sensitive information.

12.6 - Firewall

- **Definition:**
 - A network security system designed to protect a trusted private network from unauthorized access or traffic originating from an untrusted outside network (e.g., the Internet or other network segments).
 - Can be hardware, software, or a combination of both.
- **Function:**
 - Acts as a barrier between networks (e.g., LAN and WAN).
 - Continuously monitors and controls incoming and outgoing traffic based on predefined security rules.
 - Example rule: School LAN prevents students from accessing finance server, while allowing accountants to connect.

12.6.1 Types of Firewall

1. Network Firewall

- Placed between two or more networks.
- Monitors and filters traffic flowing between networks.

2. Host-based Firewall

- Installed on individual computers.
- Monitors and controls traffic to and from that single host.

12.7 - Cookies

- **Origin of Term:**

- Derived from the Unix “magic cookie”: a data packet received and returned unchanged by a program.

- **Definition:**

- A small file or data packet stored by a website on the client’s computer.
- Edited only by the creating website; stored on the client’s device.
- Used to store browsing information to enhance user experience.

- **Common Uses:**

- **Shopping carts:** Remember items added to cart.
- **Login credentials:** Authentication cookies to keep users logged in across pages.
- **Preferences:** Language choice, search queries, recently viewed pages, autofill data (name, address, D.O.B., etc.).

12.7.1 Threats due to Cookies

- **Supercookies/Zombie Cookies:**

- Malware disguised as cookies; zombie cookies get recreated after deletion.

- **Third-Party Tracking:**

- Third-party cookies may share user data without consent for advertising/tracking.
- E.g., search for an item—later see ads for similar items on other websites.

- **Precautions:**

- Be careful granting sites permission to create/store cookies on your device.

12.8 - Hackers and Crackers

- **Definitions:**

- **Hackers/Crackers:** Individuals with deep knowledge of computer systems, operating systems, networks, and programming who exploit vulnerabilities to gain unauthorized access.

- **Categories by Intent:**

1. **White Hats (Ethical Hackers)**

- Use skills to find and help fix security flaws.
- Employed by organizations to secure systems.

2. **Black Hats (Crackers)**

- Use knowledge unethically to break laws, exploit flaws, and disrupt security.

3. **Grey Hats**

- Sit between white and black hats.
- Hack for challenge or curiosity, without clear monetary or political motives.

12.9 - Network Security Threats

12.9.1 Denial of Service (DoS)

- **DoS Attack:** Attacker floods a resource with illegitimate requests, making it unavailable to legitimate users.
 - E.g., overloading a web server with massive packets so it can't respond to real traffic.

- **Recovery:**

- Simple DoS: Restarting the server/resource may restore service.
- Flooding DoS: Harder to recover, as malicious and legitimate traffic mix.
- **Distributed Denial of Service (DDoS):**
 - Uses a network of compromised “zombie” machines (Bot-net) worldwide.
 - More difficult to mitigate because attack traffic originates from many sources.

12.9.2 Intrusion Problems

- **Network Intrusion:** Any unauthorized activity on a network (e.g., DoS, Trojans, worms).
- **Other Intrusion Attacks:**
 1. **Asymmetric Routing:**
 - Attacker sends intrusion packets via multiple paths to bypass intrusion sensors.
 2. **Buffer Overflow Attacks:**
 - Overwrites memory areas with malicious code that executes on overflow.
 3. **Traffic Flooding:**
 - Overwhelms intrusion detection systems with packets, preventing proper monitoring.

12.9.3 Snooping (Sniffing)

- **Definition:** Secretly capturing and analyzing network traffic.
- **Method:**
 - Taps into communication channel, reads packets, then re-injects them so network appears normal.
- **Use Cases:**
 - Malicious data theft if traffic is unencrypted.
 - Legitimate network troubleshooting by administrators.

- **Tools:**

- Dedicated sniffing software or SPAN (Sniffer Port Analyzer) on hubs/switches.

12.9.4 Eavesdropping

- **Definition:** Real-time interception or monitoring of private communication (e.g., VoIP, instant messages, video calls).

- **Difference from Snooping:**

- **Eavesdropping:** Live interception; doesn't store for later.
- **Snooping:** Captures traffic for later analysis.

- **Examples:**

- Wiretapping traditional phone lines.
- Hacking devices with rootkits to access built-in microphones/cameras.

For more information Visit:

<https://matheenhere.blogspot.com>