Diploma in
Digital Techniques Application

DDTA

Study Material

Youth Computer Training Centre

Topics Covered:

- Concepts of core programming
- Programming Language (with C)
- Algorithms and Flowchart
- Introduction to OOPs (with JAVA)
- RDBMS(MySQL)
- HTML 5

Warning and Disclaimer

This study material is designed to provide information about 'Diploma in Digital Techniques Application'. While every effort has been made to make this study material as complete, comprehensive and as accurate as possible, no warranty or fitness is to be implied. The information is provided on an "as is" basis. Hewlett Packard Enterprise shall not have any liability or responsibility to any person or entity with respect to any loss or damages arising from the information contained in this study material. Internet websites offered as citations and/ or sources for further information may have changed or disappeared between the time this study material is written and the time when it may be read by the reader.

Contents:

Chapter 1: Concepts of Core Programming

Introduction

Core programming concepts are foundational ideas and principles that underpin all programming languages and techniques. Understanding these concepts is essential for becoming a competent programmer.

Below are the key concepts of core programming:

Core Programming: An Overview

Core programming refers to the foundational principles, concepts, and skills needed to write, understand, and work with computer programs effectively. It forms the basis for software development and is essential for creating functional and efficient software applications.

Key Areas of Core Programming

a. Programming Fundamentals

- Data Types: Understanding integers, floats, strings, booleans, and more.
- Variables: Storing and manipulating data in memory.
- Operators: Arithmetic, relational, logical, and bitwise operations.
- Control Structures:
  - Conditional statements (if-else, switch-case).
  - Loops (for, while, do-while).

b. Algorithms and Logic Development

- Problem-Solving: Breaking problems into smaller, manageable steps.
- Algorithm Design: Writing step-by-step instructions to solve problems.
- Pseudocode and Flowcharts: Visualizing and planning logic.

c. Data Structures

- Arrays, Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables.
- Understanding their usage, implementation, and efficiency.

d. Functions and Modular Programming

- Writing reusable code with functions.
- Understanding parameters, return values, and scope.
- Modular design for easier debugging and maintenance.

e. Object-Oriented Programming (OOP)

- Key concepts: Classes, Objects, Inheritance, Polymorphism, Encapsulation, and Abstraction.

- Designing programs using real-world modeling.

f. File Handling

- Reading from and writing to files.

- Managing data storage and retrieval.

g. Debugging and Testing

- Identifying and fixing errors (syntax, runtime, logical).

- Writing test cases to ensure program correctness.

h. Basic Software Development Tools

- Integrated Development Environments (IDEs): Tools like Visual Studio Code, IntelliJ, Eclipse.

- Version control systems: Git and GitHub for collaboration and tracking changes.

i. Programming Paradigms

- Procedural Programming: Focus on procedures and functions.

- Object-Oriented Programming: Organizing code around objects.

- Functional Programming: Using pure functions and avoiding state changes.

Popular Core Programming Languages

- Python: Great for beginners, versatile, and widely used in AI and web development.
- C: Excellent for understanding low-level programming and memory management.
- Java: Object-oriented, widely used for enterprise applications and Android development.
- C++: Combines procedural and object-oriented paradigms.
- JavaScript: Essential for web development.
- SQL: For database management.

Benefits of Learning Core Programming

- Problem-Solving Skills: Builds logical thinking and analytical abilities.

- Career Opportunities: High demand in software development, data science, and IT.

- Foundation for Advanced Topics: Prepares you for specialized fields like AI, web development, or cybersecurity.

Object-Oriented Programming (OOP):

Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to design and develop software. It is widely used because it helps organize and structure code, making it easier to maintain, understand, and scale.

Concepts in OOP

1. Objects

- Objects are instances of classes and represent real-world entities.
- They encapsulate data (attributes) and behavior (methods).
- Example:

```python
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model
    def display_info(self):
        print(f"Brand: {self.brand}, Model: {self.model}")
my_car = Car("Toyota", "Corolla")
my_car.display_info()
```

2. Classes

- A class is a blueprint for creating objects.
- It defines the structure and behaviour that the objects created from the class will have.

Four Core Principles of OOP

1. Encapsulation

- Encapsulation binds data and methods into a single unit (class) and restricts access to certain components.
- Access modifiers control visibility:
- Public: Accessible everywhere.
- Private: Accessible only within the class.
- Protected: Accessible within the class and its subclasses.

Example:

python

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance   Private variable
    def deposit(self, amount):
        self.__balance += amount
    def get_balance(self):
        return self.__balance
```

2. Inheritance

- Inheritance allows a class to acquire properties and behaviors from another class.
- Parent class: The base class.
- Child class: The derived class.
- Promotes code reuse.

- Example:

```python
class Animal:
    def speak(self):
        print("Animal speaks")
class Dog(Animal):   Dog inherits Animal
    def speak(self):
        print("Dog barks")
```

3. Polymorphism

- Polymorphism allows methods to take different forms depending on the context.

- Achieved through method overloading (same method name, different parameters) or overriding (redefining a method in a subclass).

- Example:

```python
class Shape:
    def area(self):
```

```
        pass
class Circle(Shape):
    def area(self, radius):
        return 3.14  radius  radius
class Rectangle(Shape):
    def area(self, length, width):
        return length  width
```

4. Abstraction

- Abstraction hides complex implementation details and shows only essential features.
- Achieved using abstract classes or interfaces.

- Example (using Python's `ABC` module): python

```
from abc import ABC, abstractmethod
class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass
class Car(Vehicle):
    def start(self):
        print("Car starts")
```

Advantages of OOP

a) Modularity: Code is organized into reusable objects.
b) Code Reusability: Inheritance allows sharing common functionality.
c) Maintainability: Encapsulation and modularity make it easy to maintain and update code.
d) Scalability: Easy to add new features without affecting existing code.
e) Data Security: Encapsulation protects sensitive data.

Common OOP Languages

- Java: Fully object-oriented, widely used in enterprise applications.
- Python: Supports OOP along with other paradigms.
- C++: Combines procedural and OOP paradigms.
- C: Popular for Windows application development.

Chapter 2: Programming Language (with C)

Introduction

C is a high-level and general-purpose programming language developed in the early 1970s by Dennis Ritchie at Bell Labs. Known for its efficiency, simplicity, and flexibility, C has become one of the most widely used languages in computer science and is especially popular in systems programming, embedded systems, and application development.

Key Features of C Language

1. Simple and Efficient: C has a straightforward syntax with a small set of keywords, which makes it easy to learn and use, yet powerful enough for low-level programming.
2. Structured Language: C allows complex programs to be broken down into simpler blocks or functions, enabling structured programming and better code organization.
3. Portability: C code can be compiled and run on various computer platforms with little or no modification, making it highly portable across systems.
4. Low-level Memory Access: C provides direct access to memory through pointers, making it ideal for system-level programming tasks.
5. Rich Library: C comes with a vast library of built-in functions, especially in the C Standard Library, providing utilities for input/output, string manipulation, math, and more.
6. Fast and Efficient: C code can be highly optimized, making it a go-to choice for performance-critical applications.

Applications of C Language

- Operating Systems: Many operating systems, like Unix and Linux, are written in C due to its efficiency and low-level access capabilities.
- Embedded Systems: C is widely used in embedded systems because it provides control over hardware.
- Compilers and Interpreters: Many modern programming language compilers and interpreters are implemented in C.
- Database Systems: Database management systems like MySQL are built using C.
- Graphics and Games: Due to its speed and efficiency, C is used in game engines and graphics libraries

Basic Syntax in C Language

A simple C program consists of functions, with `main()` being the starting point for program execution.

```
#include <stdio.h> // Header file for input/output functions

int main()

{

   printf("Hello, World!\n"); // Prints "Hello, World!" to the console

   return 0; // Returns 0 to indicate successful execution

}
```

Advantages of C Language

- Efficiency and Speed: Ideal for system-level programming where performance is crucial.
- Modularity: Code can be organized into functions and modules.
- Flexibility: Allows low-level programming, which is useful for systems and hardware-level applications.

Disadvantages of C Language

- Manual Memory Management: Requires careful memory management with pointers, which can lead to errors.
- No Object-Oriented Features: C lacks object-oriented concepts like classes and inheritance, which are present in languages like C++.
- No Built-in Exception Handling: C does not have built-in support for exception handling, making error handling more complex.

Tokens in C Language

In C, a token is the smallest element of a program that is meaningful to the compiler. Tokens form the basic building blocks of any C program, and they are used to construct expressions, statements, and functions. There are six types of tokens in C:

1. Keywords

Keywords are reserved words in C that have a special meaning and purpose. They cannot be used as identifiers (like variable or function names).

Examples: `int`, `float`, `return`, `if`, `else`, `while`, `for`, `break`, etc.

2. Identifiers

Identifiers are names used to identify variables, functions, arrays, structures, etc.

An identifier must begin with a letter (uppercase or lowercase) or an underscore (`_`), followed by letters, digits, or underscores.

Examples: `myVariable`, `sum`, `calculateArea`, `MAX_SIZE`

3. Constants

Constants are fixed values that do not change during the execution of a program.

There are different types of constants in C, including:

Integer constants: Whole numbers (e.g., `100`, `-45`)

Floating-point constants: Decimal numbers (e.g., `3.14`, `-0.99`)

Character constants: Single characters enclosed in single quotes

(e.g., `'A'`, `'@'`)

String constants: Strings of characters enclosed in double quotes

(e.g., `"Hello, World!"`)

4. Operators

Operators are symbols that perform operations on variables and values. In C, there are several types of operators:

- Arithmetic operators: `+`, `-`, `*`, `/`, `%`
- Relational operators: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Logical operators: `&&`, `||`, `!`
- Assignment operators: `=`, `+=`, `-=`, `*=`, `/=`, `%=`
- Increment and Decrement operators: `++`, `--`
- Bitwise operators: `&`, `|`, `^`, `~`, `<<`, `>>`

5. Strings

A string in C is a sequence of characters enclosed within double quotes.Strings are technically a sequence of characters stored as arrays in C, with each character taking up one element in the array.

Example: `"Hello, C programming!"`

6. Special Symbols

Special symbols are characters that have specific meanings in C, including:

Punctuation symbols: `;` (end of statement), `,` (comma separator), `:` (label in switch-case)

Braces and Parentheses:

- `{ }` : Used to note at beginning and end of a function
- `( )` : Used for function calls and expressions
- `[ ]` : Used for arrays
- `/* ... */` and `// ...`: Used for comments
- Preprocessor symbols: `#` (used to define macros or include libraries, like `#include` or `#define`)

Example Of Tokens in a C Program

Here's a sample C program to illustrate different tokens:

```
- #include <stdio.h>    // Token: Preprocessor directive
- int main()
- {          // Tokens: int, main, (), {
- int num = 10;     // Tokens: int, num, =, 10, ;
- printf("Number: %d\n", num); // Tokens: printf, (), "Number: %d\n", num, ;
- return 0;         // Tokens: return, 0, ;
- }                  // Token: }
```

In this example:

- ➔ Keywords: `int`, `return`
- ➔ Identifiers: `main`, `num`, `printf`
- ➔ Constants: `10`, `0`
- ➔ Operators: `=`
- ➔ Special Symbols: `{`, `}`, `(`, `)`, `;`
- ➔ String: `"Number: %d\n"`

C Data types

In C programming, data types define the type of data a variable can hold. Choosing the right data type is essential for efficient memory usage and accurate representation of data. C has several data types categorized into four main types:

1. Basic Data Types

Integer (`int`): Used to store whole numbers (without decimal points). The size and range depend on the system, typically 4 bytes, ranging from -2,147,483,648 to 2,147,483,647 on 32-bit systems.

Floating Point (`float`): Used to store single-precision decimal numbers. Occupies 4 bytes in memory.

Double (`double`): Used to store double-precision decimal numbers. Occupies 8 bytes in memory and has a greater precision than `float`.

Character (`char`): Used to store single characters. Occupies 1 byte, with a range of 0 to 255 (if unsigned) or -128 to 127 (if signed).

Data Type	Size	Range
int	4 bytes	2,147,483,648 to 2,147,483,647
float	4 bytes	1.2E-38 to 3.4E+38
Double	8 bytes	2.3E-308 to 1.7E+308
char	1 byte	-128 to 127 (signed) or 0 to 255 (unsigned)

2. Derived Data Types

- Arrays: Used to store multiple values of the same data type in contiguous memory locations.
- Pointers: Variables that store the memory address of another variable.
- Structures (`struct`): Custom data types that group variables of different data types.
- Unions (`union`): Similar to structures but share memory space among all members.

3. Enumeration (`enum`)

Used to assign names to integral constants, making the program more readable.

Example:

enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY };

4. Void Data Type

`void`: Indicates the absence of data. Commonly used in functions to specify that they return no value or take no parameters.

Example: void displayMessage() { /* No return value */ }

MODIFIERS IN C

C also allows type modifiers that adjust the range and size of data types:

- Signed and Unsigned: Specify whether a type can hold negative values.
- Short and Long: Used with integer and floating-point types to alter their size.

Modifier	Example	Size	Range
short int	short int	2 Bytes	-32,768 to 32,767
unsigned int	unsigned int	4 bytes	0 to 4,294,967,295
long int	long int	8 bytes	-2^63 to 2^63 - 1
unsigned char	unsigned char	1 Byte	0 to 255

Variable in C

In C, a variable is a named storage location in memory that can hold a value. Variables are essential for storing data that can be used and manipulated throughout the program. Each variable in C must be declared before it can be used, specifying the type of data it will hold.

Variable Definition in C

A variable definition in C specifies the data type and reserves memory for the variable. The general syntax for defining a variable is: data_type variable_name;

- Data_type: Specifies the type of data the variable will store (e.g., `int`, `float`, `char`).
- variable_name: The name given to the variable (also called the identifier).

Example:

- int age; // Defines an integer variable named 'age'
- float height; // Defines a float variable named 'height'
- char grade; // Defines a char variable named 'grade'

Variable Initialization

A variable can also be initialized (assigned an initial value) when it is defined. This process is called initialization.

Example:

- int age = 25; // Defines and initializes 'age' to 25
- float height = 5.9; // Defines and initializes 'height' to 5.9
- char grade = 'A'; // Defines and initializes 'grade' to 'A'

TYPES OF VARIABLES IN C

Variables in C are categorized based on their scope and lifetime. The main types are:

1. Local Variables: Defined within a function or a block. Accessible only within that function or block.

```
void exampleFunction() {
int localVar = 10; // Local variable
}
```

2. Global Variables: Defined outside of all functions. Accessible from any function within the program.

```
int globalVar = 20; // Global variable
void exampleFunction() {
    // Can access 'globalVar' here
}
```

3. Static Variables: Retain their value between function calls and are limited in scope to the function or file in which they are defined.

```
void exampleFunction() {
    static int count = 0; // Static variable
    count++;
}
```

4. External Variables: Declared using the `extern` keyword, these variables are defined in another file. They allow sharing across multiple files.

```
extern int sharedVar; // Declaration of an external variable
```

Rules for Naming Variables

- Variable names can contain letters, digits, and underscores (`_`).
- Must begin with a letter or underscore.
- Cannot be a keyword (reserved word) in C.
- C is case-sensitive, so `age` and `Age` would be considered different variables.

C Constants and Literals

The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals. Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well. The constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal. An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Floating-point literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form. While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Character constants

Character literals are enclosed in single quotes, e.g., 'x' and can be stored in a simple variable of char type. A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

Escape sequence

Escape Sequence Meaning

- ➔ \\ \character
- ➔ \' ' character
- ➔ \" " character
- ➔ \? ? character
- ➔ \a Alert or bell
- ➔ \b Backspace
- ➔ \n Newline
- ➔ \r Carriage return
- ➔ \t Horizontal tab
- ➔ \v Vertical tab

String literals

String literals or constants are enclosed in double quote "". A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters. You can break a long line into multiple lines using string literals and separating those using white spaces.

C STORAGE CLASSES

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. These specifiers precede the type that they modify. There are the following storage classes, which can be used in a C Program.

- auto
- register
- static
- extern

The auto Storage Class

The auto storage class is the default storage class for all local variables.

The register Storage Class

The register storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location). The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

The static Storage Class

The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The extern Storage Class

The extern storage class is used to give a reference of a global variable that is visible to all the program files. When you use 'extern', the variable cannot be initialized as all it does is point the variable name at a storage location that has been previously defined. When you have multiple files and you define a global variable or function, which will be used in other files also, then extern will be used in another file to give reference of defined variable or function. Just for understanding, extern is used to declare a global variable or function in another file.

C OPERATORS

In C programming, operators are symbols used to perform operations on variables and values. They play a crucial role in manipulating data and creating expressions. Operators in C can be classified into several categories:

1. Arithmetic Operators

Used for basic mathematical operations.

Operators: `+`, `-`, `*`, `/`, `%`

Examples:

- int a = 10, b = 3;
- int sum = a + b; // Addition
- int difference = a - b; // Subtraction
- int product = a * b; // Multiplication
- int quotient = a / b; // Division
- int remainder = a % b; // Modulus (remainder)

2. Relational Operators

Used to compare two values.

These operators return `1` if the comparison is true and `0` if false.

Operators: `==`, `!=`, `>`, `<`, `>=`, `<=`

Examples:

- int x = 5, y = 10;
- (x == y); // False (0)
- (x != y); // True (1)
- (x > y); // False (0)
- (x < y); // True (1)

3. Logical Operators

Used to combine multiple conditions.

Operators: `&&` (logical AND), `||` (logical OR), `!` (logical NOT)

Examples:

- int x = 5, y = 10;
- (x < y && x > 0); // True (1) if both conditions are true
- (x > y || y > 0); // True (1) if at least one condition is true
- !(x < y); // False (0) as NOT inverts the result

4. Assignment Operators

Used to assign values to variables.

The basic assignment operator is `=`, but there are compound assignment operators for shorthand.

Operators: `=`, `+=`, `-=`, `*=`, `/=`, `%=`

Examples:

- int a = 10;
- a += 5; // Equivalent to a = a + 5 (a becomes 15)
- a -= 2; // Equivalent to a = a - 2 (a becomes 13)
- a *= 3; // Equivalent to a = a * 3 (a becomes 39)
- a /= 3; // Equivalent to a = a / 3 (a becomes 13)

5. Bitwise Operators

Used to perform operations on individual bits.

Operators: `&` (AND), `|` (OR), `^` (XOR), `~` (NOT), `<<` (left shift), `>>` (right shift)

Examples:

- int a = 5; // Binary: 0101
- int b = 3; // Binary: 0011
- int c = a & b; // Bitwise AND (Binary: 0001)
- int d = a | b; // Bitwise OR (Binary: 0111)
- int e = a ^ b; // Bitwise XOR (Binary: 0110)
- int f = ~a; // Bitwise NOT (Binary: 1010 for a = 5)

6. Increment and Decrement Operators

Used to increase or decrease the value of a variable by 1.

Operators: `++` (increment), `--` (decrement)

Examples:

- int x = 5;
- x++; // Now x is 6
- x--; // Now x is back to 5

7. Conditional (Ternary) Operator

Shorthand for `if-else` statement, using the syntax `condition ? value_if_true : value_if_false`.

Example:

int x = 10;

int y = (x > 5) ? 100 : 200; // y will be 100 because x > 5 is true

8. Special Operators

sizeof: Returns the size (in bytes) of a data type or variable.

int x = 10;

printf("%lu", sizeof(x)); // Outputs the size of an int (typically 4 bytes)

Comma Operator (,):

Allows multiple expressions to be evaluated in a single statement. Only the last expression is returned.

int a = (5, 10); // a will be 10 (the last expression)

Pointer Operators (`&` and `*`):

`&` is used to get the address of a variable.

`*` is used to dereference a pointer to access the value at the address.

Member Access Operators (`.` and `->`):

`.` is used to access a member of a structure.

`->` is used to access a member of a structure through a pointer.

Example Program with C Operators

```
#include <stdio.h>
int main() {
int a = 5, b = 10;
// Arithmetic Operators
printf("Sum: %d\n", a + b);
// Relational Operators
printf("Is a greater than b? %d\n", a > b);
// Logical Operators
printf("Is a less than b AND b greater than 0? %d\n", (a < b && b > 0));
// Bitwise Operators
printf("Bitwise AND: %d\n", a & b);
// Conditional Operator
int max = (a > b) ? a : b;
printf("Max value: %d\n", max);
return 0;
}
```

Decision Making In C Programming

In C programming, decision-making statements are used to control the flow of execution based on certain conditions. These statements evaluate expressions or conditions and execute specific blocks of code accordingly. C provides several decision-making statements:

1. if Statement

The `if` statement executes a block of code only if a specified condition is true.

Syntax:

```
if (condition) {
    // Code to execute if condition is true
}
```

Example:

```
int x = 10;
if (x > 5) {
    printf("x is greater than 5\n");
}
```

2. if-else Statement

The `if-else` statement provides an alternative set of statements if the `if` condition is false.

Syntax:

```
if (condition) {
    // Code to execute if condition is true
} else {
    // Code to execute if condition is false
}
```

Example:

```
int x = 3;
if (x > 5) {
    printf("x is greater than 5\n");
} else {
    printf("x is less than or equal to 5\n");
}
```

3. if-else if-else Ladder

The `if-else if-else` ladder is used when multiple conditions need to be checked. Only the first true condition block is executed.

Syntax:

```
if (condition1) {
    // Code to execute if condition1 is true
} else if (condition2) {
    // Code to execute if condition2 is true
} else {
    // Code to execute if none of the above conditions are true
}
```

Example:

```
int x = 15;
if (x < 10) {
    printf("x is less than 10\n");
} else if (x == 10) {
    printf("x is 10\n");
} else {
    printf("x is greater than 10\n");
}
```

4. Nested if Statements

`if` statements can be nested inside one another to evaluate more complex conditions.

Syntax:

```
if (condition1) {
    if (condition2) {
        // Code to execute if both condition1 and condition2 are true
    }
}
```

Example:

```
int x = 10, y = 20;
if (x > 5) {
    if (y > 15) {
        printf("Both conditions are true\n");
    }
}
```

5. switch Statement

The `switch` statement allows multiple conditional branches based on the value of a single expression. It's commonly used when there are multiple possible values for a variable.

Syntax:

```
switch (expression) {
    case constant1:
        // Code to execute if expression equals constant1
        break;
    case constant2:
        // Code to execute if expression equals constant2
        break;
    ...
    default:
        // Code to execute if expression doesn't match any case
}
```

Example:

```
int day = 3;
switch (day) {
    case 1:
        printf("Monday\n");
        break;
    case 2:
        printf("Tuesday\n");
```

```
        break;
    case 3:
        printf("Wednesday\n");
        break;
    default:
        printf("Invalid day\n");
}
```

Explanation of `break`:

Each `case` ends with a `break` statement, which exits the `switch` block after a match. Without `break`, execution would continue to the next case, leading to "fall-through."

6. Conditional (Ternary) Operator

The ternary operator (`?:`) is a shorthand way of making decisions. It's a single-line replacement for an `if-else` statement.

Syntax:

```
condition ? expression_if_true : expression_if_false;
```

Example:

```
int x = 10;
int result = (x > 5) ? 100 : 200; // result will be 100 if x > 5, otherwise 200
```

EXAMPLE PROGRAM WITH DECISION-MAKING STATEMENTS

```
#include <stdio.h>          int main() {
    int number = 0;
    printf("Enter a number: ");
    scanf("%d", &number);
  // if-else statement
    if (number > 0) {
        printf("Positive number\n");
    } else if (number < 0) {
        printf("Negative number\n");
    } else {
        printf("Zero\n");
    }
```

```
    // switch statement
    int day = 3;
    switch (day) {
        case 1:
            printf("Monday\n");
            break;
        case 2:
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
        default:
            printf("Other day\n");
    }
 // Ternary operator
    int age = 20;
    printf("You are %s\n", (age >= 18) ? "an adult" : "a minor");
return 0;
}
```

C LOOPS

In C programming, loops allow you to repeat a block of code multiple times until a specified condition is met. C offers several types of loops, each with different use cases and syntax.

1. while Loop

The `while` loop repeats a block of code as long as the specified condition is true. If the condition is false initially, the loop will not execute at all.

Syntax:

```
    while (condition) {
        // Code to execute as long as condition is true
     }
```

Example:

```
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

Output:

0

1

2

3

4

2. do-while Loop

The `do-while` loop is similar to the `while` loop, but it guarantees that the loop's body executes at least once. The condition is checked after the body is executed.

Syntax:

```
do {
    // Code to execute
} while (condition);
```

Example:

```
 int i = 0;
do {
    printf("%d\n", i);
    i++;
} while (i < 5);
```

Output:

0

1

2

3

4

3. for Loop

The `for` loop is commonly used for counting or iterating a fixed number of times. It includes initialization, condition checking, and increment/decrement in one line.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // Code to execute
}
```

Example:

```
for (int i = 0; i < 5; i++) {
    printf("%d\n", i);
}
```

Output:

0

1

2

3

4

4. Nested Loops

You can place one loop inside another to create nested loops. This is useful for working with matrices or generating patterns.

Example:

```
 for (int i = 1; i <= 3; i++) {
for (int j = 1; j <= 3; j++) {
    printf("(%d, %d) ", i, j);
}                    printf("\n");              }
```

Output:

(1, 1) (1, 2) (1, 3)

(2, 1) (2, 2) (2, 3)

(3, 1) (3, 2) (3, 3)

5. Infinite Loops

An infinite loop continues forever because the condition is always true. It's often used for programs that should run continuously, like a server.

Example:

```
while (1) { // Infinite loop
printf("This will print forever.\n");
}
```

6. Loop Control Statements

- break: Exits the loop immediately.
- continue: Skips the current iteration and proceeds to the next iteration.

Examples:

```
// break example
for (int i = 0; i < 5; i++) {
if (i == 3) {
break;
}
printf("%d\n", i);
}
// Output: 0 1 2
// continue example
for (int i = 0; i < 5; i++) {
if (i == 3) {
continue;
} printf("%d\n", i); }
```

// Output: 0 1 2 4

EXAMPLE PROGRAM USING LOOPS

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    // Using a for loop to print multiplication table
    printf("Multiplication Table of %d:\n", n);
    for (int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", n, i, n * i);
    }
    return 0;
}
```

C FUNCTIONS

In C programming, functions are blocks of code designed to perform specific tasks. Functions help in organizing code into smaller parts, promoting reusability and modularity. C programs can have predefined (standard library) functions like `printf()` and `scanf()`, as well as user-defined functions.

Key Components of a Function

1. Function Declaration (Prototype): This tells the compiler about the function's name, return type, and parameters before its actual implementation.

Syntax: return_type function_name(parameter_type1, parameter_type2, ...);

Example: int add(int, int);

2. Function Definition: This is where the actual code for the function resides.

Syntax:

```
return_type function_name(parameter_type1 parameter1, parameter_type2 parameter2, ...) {
    // Code to execute
    return result; // Only if return_type is not void
}
```

Example:

```
int add(int a, int b) {        return a + b;        }
```

3. Function Call: This is where the function is used in the main program to perform its task.

Syntax:

```
function_name(arguments);
```

Example:

```
int sum = add(5, 3);
```

Types of Functions

1. Library Functions: Predefined functions in C, like `printf()`, `scanf()`, `strlen()`, etc., which are included in header files like `stdio.h`, `string.h`, etc.

2. User-Defined Functions: Functions created by the programmer to perform specific tasks.

Function Example: Adding Two Numbers

```
#include <stdio.h>

// Function declaration

int add(int, int);

int main() {

    int a = 5, b = 10;

    int sum = add(a, b); // Function call

    printf("Sum: %d\n", sum);

    return 0;

}
```

// Function definition

```
int add(int x, int y) {

    return x + y;

}
```

Return Type of Functions

- void: Indicates the function does not return a value.
- int, float, char, etc.: The function will return a value of this type.

Parameter Types

- Actual Parameters: The parameters passed to the function during a call.
- Formal Parameters: The parameters in the function definition that receive the values of actual parameters.

Pass by Value vs. Pass by Reference

In C, arguments are generally passed by value, meaning the function works with a copy of the original data.

Changes inside the function do not affect the original values.

3. Recursive Functions

A recursive function is one that calls itself, allowing tasks to be performed repeatedly in smaller pieces. Recursive functions require a base case to terminate.

EXAMPLE: FACTORIAL USING RECURSION

```
#include <stdio.h>
int factorial (int n) {
    if (n == 0) // Base case
        return 1;
    else
        return n * factorial(n - 1); // Recursive call
}
int main() {
    int number = 5;
    printf("Factorial of %d is %d\n", number, factorial(number));
    return 0;
}
```

Advantages of Using Functions

- Code Reusability: Functions allow you to use the same block of code multiple times.
- Modularity: Functions break down large problems into smaller, manageable parts.
- Ease of Maintenance: Functions make the code easier to understand, test, and debug.

C ARRAYS

In C programming, arrays are used to store multiple values of the same data type in a single variable. Arrays allow you to work with a collection of data using a single identifier, which can be very helpful for organizing and manipulating groups of values.

Key Features of Arrays

1. Fixed Size: The size of an array is defined at the time of declaration and cannot be changed during runtime.
2. Same Data Type: All elements in an array are of the same data type.
3. Contiguous Memory Allocation: Elements are stored in consecutive memory locations.

Syntax of Array Declaration

data_type array_name[array_size];

- `data_type`: Type of the elements in the array (e.g., `int`, `float`, `char`).
- `array_name`: Name of the array.
- `array_size`: Number of elements the array can hold.

Example:

int numbers [5]; // Declares an array of 5 integers

Initializing an Array: Arrays can be initialized at the time of declaration.

1. With Specific Values:

int numbers [5] = {1, 2, 3, 4, 5};

2. Partial Initialization: If fewer values are provided, the rest are set to 0 by default.

   int numbers [5] = {1, 2}; // Initializes to {1, 2, 0, 0, 0}

3. Omitting Size: The size can be omitted when initializing, and it will be determined by the number of values provided.

   int numbers [] = {1, 2, 3, 4, 5};

Accessing Array Elements

Array elements are accessed using index numbers. Array indexing starts at `0`, so the first element is `array_name[0]`, the second is `array_name[1]`, and so on.

Example:

int numbers[5] = {10, 20, 30, 40, 50};

printf("%d", numbers[2]); // Output: 30

Array Example: Sum of Elements

```
#include <stdio.h>

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};
    int sum = 0;
    for (int i = 0; i < 5; i++) {
        sum += numbers[i]; // Adds each element to sum
    }
    printf("Sum of array elements: %d\n", sum);        return 0;        }
```

TYPES OF ARRAYS

1. One-Dimensional Array: Stores a list of elements in a single row.

int numbers[5] = {1, 2, 3, 4, 5}

2. Multi-Dimensional Array: Stores data in a table format (e.g., a 2D array is often used to represent matrices).

data_type array_name[rows][columns];

Example of a 2D Array:

```
int matrix[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

C Pointers

As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory. A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address. The general form of a pointer variable declaration is:

type *var-name;

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration: Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

```
int *ip; /* pointer to an integer */
double *dp; /* pointer to a double */
float *fp; /* pointer to a float */
char *ch /* pointer to a character */
```

NULL POINTERS IN C

It is always a good practice to assign a NULL value to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.

POINTER ARITHMETIC

As explained in main chapter, C pointer is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and - To understand pointer arithmetic, let us consider that ptr is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer: ptr++; Now, after the above operation, the ptr will point to the location 1004 because each time ptr is incremented, it will point to the next integer location which is 4 bytes next to the current location. This operation will move the pointer to next memory location without impacting actual value at the memory location. If ptr points to a character whose address is 1000, then above operation will point to the location 1001 because next character will be available at 1001.

POINTER TO POINTER

A pointer to a pointer is a form of multiple indirections, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value. A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, following is the declaration to declare a pointer to a pointer of type int: int **var;

C Strings

The string in C programming language is actually a one-dimensional array of characters which is terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null. The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello".

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

If you follow the rule of array initialization then you can write the above statement as follows:

char greeting[] = "Hello";

Following is the memory presentation of above-defined string in C/C++: Actually, you do not place the null character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array.

C Structures

C arrays allow you to define type of variables that can hold several data items of the same kind but structure is another user defined data type available in C programming, which allows you to combine data items of different kinds. Structures are used to represent a record, suppose you want to keep track of your books in a library. You might want to track the following attributes about each book: Title, Author, Subject, Book ID Here is the way you would declare the Book structure:

```
struct Books

{

char title[50];

char author[50];

char subject[100];

int book_id;

} book;
```

ACCESSING STRUCTURE MEMBERS

To access any member of a structure, we use the member access operator (.). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use struct keyword to define variables of structure type.

Pointers to Structures

You can define pointers to structures in very similar way as you define pointer to any other variable as follows:

```
struct Books *struct_pointer;
```

Now, you can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the & operator before the structure's name as follows:

```
struct_pointer = &Book1;
```

To access the members of a structure using a pointer to that structure, you must use the -> operator as follows:

```
struct_pointer->title;
```

C UNIONS

In C programming, a union is a special data type that allows storing different data types in the same memory location. Unlike structures, where each member has its own memory, a union shares the same memory location for all of its members. This means only one of the union members can hold a value at any given time, which can be memory-efficient when dealing with different types that don’t need to hold values simultaneously.

Key Features of Unions

1. Shared Memory: All members of a union share the same memory. The memory size of a union is equal to the size of its largest member.
2. Single Active Member: Only one member can hold a value at any given time. Assigning a value to one member will overwrite the value of any previously assigned member.
3. Efficient Memory Usage: Unions can be very efficient in memory-constrained environments.

Syntax of a Union

```
union union_name {
    data_type1 member1;
    data_type2 member2;
};
```

- `union_name`: The name of the union.
- `data_type1`, `data_type2`: The data types of the members.
- `member1`, `member2`: The members of the union.

Summary:

C Programming is a powerful, general-purpose programming language that serves as a foundation for many other programming languages. It is widely used for developing system software, embedded systems, and applications requiring high performance.

- **Fundamental Programming Knowledge**

  Core Concepts: C Programming teaches fundamental programming concepts such as loops, functions, arrays, pointers, and memory management. Foundation for Other Languages: Languages like C++, Java, Python, and even modern ones like Rust borrow concepts from C Programming.

- **System-Level Programming**

  Low-Level Access: C Programming allows direct manipulation of memory and hardware, making it ideal for system programming. Operating Systems: C Programming is the backbone of many operating systems, including Linux, Windows, and macOS

Check your Understanding:

Question 1

In which year C language was developed?

a. 1974
b. 1972
c. 1971
d. 1973

Question 2

The C language has been developed at -

a. AT & T Bell Labs
b. IBM
c. Borland International
d. Sun Microsystems

Question 3

C- language is

a. Assembly level Language
b. Low level Language
c. High level Language
d. All of above

Question 4

Who developed the C language

a. Dennis Ritchie
b. Ken Thompson
c. Matrin Richards
d. Patric Naughton

Question 5

Which of the following will not valid expressions in C?

a. a=2+(b=5);
b. a=b=c=5;
c. a=11%3
d. b+5=2

Question 6

The escape sequence '\b" is a

a. back space
b. next line
c. tab
d. none of the above

Question 7

Every C program should compulsorily have a function called as:

a. start()
b. Start()
c. main()
d. Main()

Question 8

Which will be the output of following program?

```
#include<stdio.h>
void main()
{
int a;
printf("%d\n", a);
}
```

a. Error
b. 0
c. -1
d. Garbage Value

Question 9

Which of the following variable declaration is correct?

a. int length
b. char int
c. int long
d. All

Question 10

Find the output of the following c program?

```
#include<stdio.h>
{
int x=10;
printf("%d%d%d", x, x++, ++x) ;
return 0;
}
```

a. 11 11 11
b. 12 10 10
c. 12 11 10
d. 12 11 11

Chapter 3: Algorithms and Flowchart

1. Algorithms

An algorithm is a step-by-step procedure to solve a specific problem. Key aspects of algorithms include:

Types of Algorithms

Sorting Algorithms: Arrange data in a specific order.

Examples: Bubble Sort, Merge Sort, Quick Sort, Heap Sort.

Searching Algorithms: Find specific elements in a dataset.

Examples: Binary Search, Linear Search, Depth-First Search (DFS), Breadth-First Search (BFS).

Dynamic Programming: Breaks problems into smaller overlapping subproblems.

Examples: Fibonacci Sequence, Knapsack Problem, Longest Common Subsequence.

Greedy Algorithms: Make the best choice at each step.

Examples: Dijkstra's Algorithm, Prim's Algorithm, Huffman Encoding.

Divide and Conquer: Divide a problem into smaller subproblems, solve them, and combine results.

Examples: Merge Sort, Quick Sort, Binary Search.

Backtracking: Explore all possibilities to solve a problem.

Examples: Sudoku Solver, N-Queens Problem, Maze Solver.

Algorithm Complexity

Time Complexity: Measures how the runtime of an algorithm grows with input size.

Common classes: O(1), O(log n), O(n), O(n log n), $O(n^2)$, etc.

Space Complexity: Measures the memory usage of an algorithm.

Flowchart

A flowchart is a visual representation of a process, system, or algorithm, using symbols to depict steps and arrows to indicate the sequence of operations. It is widely used for explaining workflows, decision-making processes, or problem-solving strategies.

Key Features of a Flowchart:

1. Simplicity: Breaks down complex processes into clear, sequential steps.

2. Standard Symbols: Uses shapes like ovals, rectangles, diamonds, and arrows to standardize representation.

3. Clarity: Makes processes easier to understand and communicate.

Why Use Flowcharts?

- To visually organize ideas and processes.

- To identify inefficiencies or bottlenecks in workflows.

- To document existing systems for better understanding.

- To simplify programming algorithms or business processes.

Common Uses of Flowcharts:

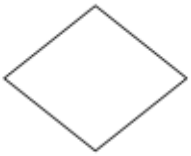
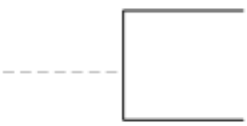
1. Programming: Designing and explaining algorithms.

2. Business: Modelling workflows, such as approval processes.

3. Education: Teaching step-by-step processes in various disciplines.

4. Engineering: Visualizing system operations.

Symbols are used in drawing Flowchart

Flowchart Symbol	Symbol Name	Description
	Terminal (Start or Stop)	Terminals (Oval shapes) are used to represent start and stop of the flowchart.
	Flow Lines or Arrow	Flow lines are used to connect symbols used in flowchart and indicate direction of flow.
	Input / Output	Parallelograms are used to read input data and output or display information
	Process	Rectangles are generally used to represent process. For example, Arithmetic operations, Data movement etc.
	Decision	Diamond shapes are generally used to check any condition or take decision for which there are two answers, they are, yes (true) or no (false).
	Connector	It is used connect or join flow lines.
	Annotation	It is used to provide additional information about another flowchart symbol in the form of comments or remarks.

Some Concepts of Algorithm and Flowchart

	Algorithm	Flowchart
Definition	A set of rules for solving a computational problem	A graphical representation of a program's steps
Complexity	Difficult to create and understand	Easy to design and understand
Geometrical diagrams	Doesn't include geometric patterns	Uses geometric shapes, symbols, and patterns
Scope of usage	Used in computer science and mathematics	Can be used in many disciplines

Algorithm Vs. Flowchart.

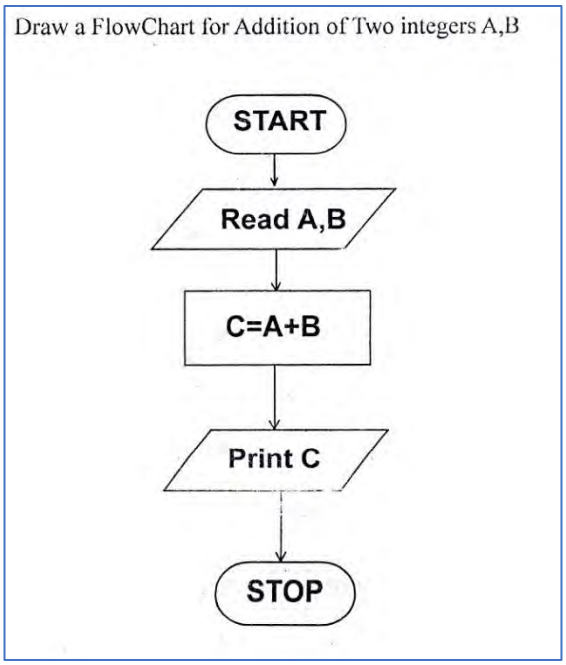
Algorithms and flowcharts are different mechanisms used for designing different programs, particularly in computer programming. An algorithm is a step-by-step summary of the procedure, while on the other hand, a flowchart illustrates the steps of a program graphically.

Example of an algorithm

For finding the Square and Square Root of given 30 number.

- STEP1: Initialize COUNT to zero
- STEP2: Read one number
- STEP3: compute Square and Square Root value
- STEP4: Print the values
- STEP5: Add 1 to COUNT
- STEP6: Is COUNT=30 if no, go to STEP2
- STEP7: STOP

Examples of Flowchart



Applications of Algorithms

1. Programming: Forms the foundation of coding logic.
2. Data Processing: Sorting, searching, and data analysis.
3. Optimization: Solving mathematical and engineering problems efficiently.
4. Automation: Driving decision-making in AI, robotics, and other fields.

Types of Algorithms

1. Brute Force: Tries all possible solutions until the correct one is found.
   Example: Linear search.
2. Divide and Conquer: Breaks the problem into smaller sub problems, solves them independently, and combines the results.
   Example: Merge sort, quicksort.
3. Greedy Algorithm: Makes the most optimal choice at each step.
   Example: Dijkstra's shortest path algorithm.
4. Dynamic Programming: Breaks the problem into overlapping sub problems and solves them efficiently.
   Example: Fibonacci sequence, knapsack problem.
5. Backtracking: Explores all possible solutions and abandons solutions that fail to satisfy the constraints.
   Example: Sudoku solver.

Importance of Algorithms

1. Problem Solving: Helps in breaking down and solving problems logically.
2. Optimization: Ensures tasks are performed in the least time and with the least resources.
3. Scalability: Algorithms form the backbone of systems that handle large-scale data efficiently.

Mastering algorithms is key to efficient programming and effective problem-solving!

Flowchart

A flowchart is a visual representation of a process or algorithm. It uses standardized symbols to describe the flow of steps in a task, making it easier to understand and communicate the logic or sequence of operations.

Features of a Flowchart

1. Visual Representation: Provides a clear and graphical illustration of a process.
2. Logical Sequence: Shows the steps in a process in a logical order.
3. Decision Points: Includes conditional branches for decision-making.

Check your Understanding

Question 1

What is an algorithm?

a) A flowchart

b) Step by step instructions used to solve a problem

c) A flowchart or pseudocode

d) A decision

Question 2

A process box is _____ in shape.

a) Connector

b) Rectangular

c) Oval

d) Circle

Question 3

Part of an algorithm which is repeated for fixed number of times is classified as

a) iteration

b) selection

c) sequence

d) reverse action

Question 4

Diamond shaped symbol is used in flowcharts to show the

e) decision box

f) statement box

g) error box

h) if-statement box

Question 5

What is a list of instructions in a proper order to solve a problem called?

a) Flowchart

b) Sequence

c) Algorithm

d) None of these

Question 6

Symbol used in flowchart such as rectangle with horizontal lines on two sides is used for

a) rhombus

b) parallelogram

c) circle

d) trapezoid

Question 7

In a flowchart a calculation (process) is represented by

a) A rectangle

b) A rhombus

c) A parallelogram

d) A circle

Question 8

A process box is _____ in shape.

a) Connector

b) Rectangular

c) Oval

d) Circle

Question 9

Algorithm and Flow chart help us to

a) Know the memory capacity

b) Identify the base of a number system

c) Direct the output to a printer

d) Specify the problem completely and clearly

Question 10

Diamond shaped symbol is used in flowcharts to show the

a) decision box

b) statement box

c) error box

d) if-statement box

Chapter 4: Introduction to OOPs (with JAVA)

Introduction

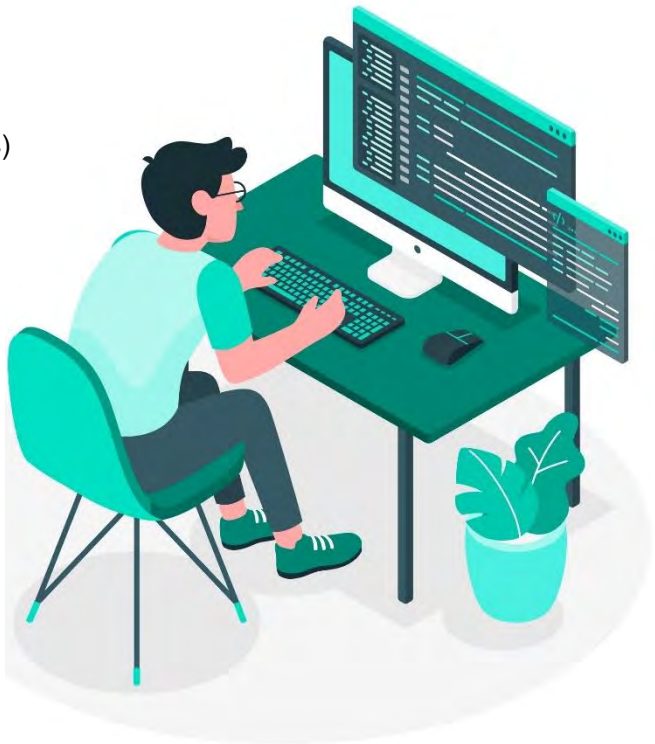
The Java programming language is incredibly useful and continues to be a prominent choice for many developers worldwide due to its versatility, robust ecosystem, and wide range of applications.

- Platform Independence
- Object-Oriented Programming (OOP)
- Rich Ecosystem
- Automatic Memory Management
- Strong Security Features

What is JAVA ?

Java is a high-level, object-oriented, class-based, concurrent, secured and general-purpose computer-programming language that was developed by Sun Microsystems in 1995, and is now owned by Oracle Corporation.

Platform: Any hardware or software environment in which a program runs, is known as a platform. In the context of Java, the term "platform" refers to the combination of the Java Runtime Environment (JRE) and the Java API (Application Programming Interface). This platform allows Java programs to run across different hardware and operating systems without modification.



Application

- Mobile applications (especially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Game Development
- Database connection
- Smart Card
- Robotics
- Embedded Systems

Features of Java

- Simple
- Object-Oriented
- Portable
- Platform independent
- Secured
- Robust
- Architecture neutral
- Interpreted
- High Performance
- Multithreaded
- Distributed
- Dynamic

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Platform Independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into byte code. This byte code is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer
- Java Programs run inside a virtual machine sandbox
- Class loader: Class loader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically.
- Byte code Verifier: It checks the code fragments for illegal code that can violate access rights to objects.

Security Manager:

It determines what resources a class can access such as reading and writing to the local disk.

Robust

Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java.

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. It occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

Java is portable because it facilitates you to carry the Java byte code to any platform. It doesn't require any implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java byte code is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++).

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. Java supports dynamic compilation and automatic memory management (garbage collection).

Creating Hello World Example:

```
class Simple {

public static void main(String args[]) { System.out.println ("Hello Java");

} }
```

Save the above file as Simple. java

To compile: javac Simple.java

To execute: java Simple

Output:

HELLO JAVA COMPILATION FLOW:

When we compile Java program using javac tool, the Java compiler converts the source code into byte code.

PARAMETERS USED IN FIRST JAVA PROGRAM

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- class keyword is used to declare a class in Java.
- public keyword is an access modifier that represents visibility. It means it is visible to all.
- Static is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method.
- The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- void is the return type of the method. It means it doesn't return any value.
- main represents the starting point of the program.
- String[] args or String args[] is used for command line argument. We will discuss it in coming section.
- System.out. println() is used to print statement. Here, System is a class, out is an object of the Print Stream class, println() is a method of the Print Stream class. We will discuss the internal working of System.out.println() statement in the coming section.

BASIC CONCEPTS OF OOPS

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

OBJECT IN JAVA

An entity that has state and behaviour is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- State: represents the data (value) of an object.
- Behaviour: represents the behaviour (functionality) of an object such as deposit, withdraw, etc.
- Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Object Definitions:

- An object is a real-world entity.
- An object is a runtime entity.
- The object is an entity which has state and behaviour.
- The object is an instance of a class.

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs

Terms used in Inheritance:

- Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines. We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

Abstract class: A class which is declared with the abstract keyword is known as an abstract class in Java. A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

A CLASS IN JAVA CAN CONTAIN

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

There are 3 ways to initialize object in Java.

- By reference variable
- By method
- By constructor

There are many ways to create an object in java.

- By new keyword
- By newInstance() method
- By clone() method
- By deserialization
- By factory method etc.

Data: Data field means a defined area of a file or data table used to record an individual piece of standardized data. At its most basic, a Java field is a variable. This means that it represents a value, such as a numerical value or a text.

Data types are divided into two groups:

- Primitive data types -includes byte, short, int, long, float, double, Boolean and char
- Non-primitive data types -such as String, Arrays and Classes.

Declaration of Instance Variables:

Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables. Thus, the data for one object is separate and unique from the data for another. An instance variable can be declared public or private or default (no modifier). When we do not want our variable's value to be changed out-side our class we should declare them private. Public variables can be accessed and changed from outside of the class

Declaration of Instance Method:

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

ACCESSING CLASS MEMBERS IN JAVA

1. Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Understanding Java Access Modifiers

- Access
- Modifier
- Within
- Class
- Within Package Outside Package
- By Subclass
- only
- Outside
- Package

Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

CONSTRUCTOR IN JAVA

A constructor is a special member function of a class with the same name as the class name but has no return type. Whenever an object of a class is created using a new keyword, it invokes a constructor of that class.

Example: class Sample { Sample()

{

//constructor body

}

}

Features of Constructors in Java:

- The constructor's name has the same as the Class Name.
- A constructor must not have any return type. Not even void.
- The constructor automatically gets calls when an object creates for the class.
- Unlike, Constructors in C++, Java doesn't allow the constructor definition outside the class.
- Constructors usually make use to initialize all the data members (variables) of the class.
- In addition, more than one constructor can also declare inside a class with different parameters. It's called "Constructor Overloading."

DESTRUCTOR IN JAVA

Destructor is a method called when the destruction of an object takes place. The main goal of the destructor is to free up the allocated memory and also to clean up resources like the closing of open files, closing of database connections, closing network resources, etc.

Syntax

```
class Object
{
Protected void finalize ()
{
//statements like the closure of database connection
}
}
```

METHOD OVERLOADING IN JAVA

Method overloading in Java means having two or more methods (or functions) in a class with the same name and different arguments (or parameters). It can be with a different number of arguments or different data types of arguments.

```
class Method_Overloading {
double figure(double l, int b) //two parameters with double type
{
return (l*b);
}
float figure(int s) //one parameter with float return type
{
return (s*s);
}
public static void main(String[] args) {
Method_Overloading obj = new Method_Overloading();
System.out.println ("Area of Rectangle: " +obj.figure(5.55, 6));
System.out.println("Area of Square: " +obj.figure(3));       }}
```

This keyword

There can be a lot of usage of Java this keyword. In Java, this is a reference variable that refers to the current object.

Usage of Java this keyword

a) This can be used to refer current class instance variable.
b) This can be used to invoke current class method (implicitly)
c) This () can be used to invoke current class constructor.
d) This can be passed as an argument in the method call.
e) This can be passed as argument in the constructor call.
f) This can be used to return the current class instance from the method.
g) Passing and Returning Objects in Java
h) While creating a variable of a class type, we only create a reference to an object. ...
i) This effectively means that objects act as if they are passed to methods by use of call-by-reference.
j) Changes to the object inside the method do reflect the object used as an argument.

Nested Classes

In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the nested class, and the class that holds the inner class is called the outer class.

Syntax

Write a nested class. Here, the class Outer_Demo is the outer class and the class Inner_Demo is the nested class.

```
class Outer_Demo { class Inner_Demo {

}}
```

Inner Class

Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.

Syntax

```
class Java_Outer_class{

//code

class Java_Inner_class{

//code } }
```

JAVA INHERITANCE

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- subclass (child) - the class that inherits from another class
- superclass (parent) - the class being inherited from To inherit from a class, use the extends keyword.

The syntax of Java Inheritance

class Subclass-name extends Superclass-name

{

//methods and fields

}

- The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
- In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Terms used in Inheritance

- Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

TYPES OF INHERITANCE IN JAVA

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical

MULTILEVEL HIERARCHY IN JAVA

A multi-level hierarchy is a recursive modelling structure where a system, the whole at the level of interest (the macro-level), can be taken apart into a set of sub-systems, the parts, that interact statically or dynamically at the level below (the micro-level).

Example

```
class A {
void funcA(){
        System.out.println("This is class A");
}}
class B extends A {
    void funcB(){
            System.out.println("This is class B");
}}
class C extends B { void funcC(){
System.out.println("This is class C");
}}
public class Demo{
public static void main(String args[])
{
 C obj =new C();
obj.funcA();
obj.funcB();
obj.funcC();
}}
```

Output

```
This is class A
This is class B
This is class C
```

Inheritance and Constructors in Java

Constructors in Java are used to initialize the values of the attributes of the object serving the goal to bring Java closer to the real world. We already have a default constructor that is called automatically if no constructor is found in the code.

But if we make any constructor say parameterized constructor in order to initialize some attributes then it must write down the default constructor because it now will be no more automatically called.

EXAMPLE:

```
// Java Program to Illustrate
// Invocation of Constructor
// Calling Without Usage of
// super Keyword
// Class 1
// Super class
class Base {
// Constructor of super class Base()
{
// Print statement
System.out.println("Base Class Constructor Called ");
}
}
// Class 2
// Sub class
class Derived extends Base {
// Constructor of sub class Derived()
{
// Print statement
System.out.println("Derived Class Constructor Called ");
}
}
// Class 3
// Main class class GFG {
// Main driver method
public static void main(String[] args)
{
// Creating an object of sub class
// inside main() method Derived d = new Derived();
```

```
// Note: Here first super class constructor will be
// called there after derived(sub class) constructor
// will be called
}
}
```

Output

Base Class Constructor Called

Derived Class Constructor Called

Output Explanation: Here first superclass constructor will be called thereafter derived(sub-class) constructor will be called because the constructor call is from top to bottom. And yes if there was any class that our Parent class is extending then the body of that class will be executed thereafter landing up to derived classes.

But, if we want to call a parameterized constructor of the base class, then we can call it using super(). The point to note is base class constructor call must be the first line in the derived class constructor.

Implementation: super (_x) is the first line-derived class constructor.

Java

```
// Java Program to Illustrate Invocation
// of Constructor Calling With Usage
// of super Keyword
// Class 1
// Super class
class Base {
int x;
// Constructor of super class
Base(int _x) {x = _x; }
}
```

```
// Class 2
// Sub class
class Derived extends Base {
    int y;
// Constructor of sub class Derived(int _x, int _y)
{
// super keyword refers to super class super(_x);
y = _y;
}
// Method of sub class void Display()
{
// Print statement
System.out.println("x = " + x + ", y = " + y);
}
}
// Class 3
// Main class public class GFG {
// Main driver method
public static void main(String[] args)
{
// Creating object of sub class
// inside main() method
Derived d = new Derived(10, 20);
// Invoking method inside main() method
d.Display();
}
}
```

Output

x = 10, y = 20

Super in Java

Super is a keyword of Java which refers to the immediate parent of a class and is used inside the subclass method definition for calling a method defined in the superclass. A superclass having methods as private cannot be called. Only the methods which are public and protected can be called by the keyword super. It is also used by class constructors to invoke constructors of its parent class.

Syntax:

super. <method-name>(); Usage of Super Class

- Super variables refer to the variable of a variable of the parent class.
- Super () invokes the constructor of immediate parent class.
- Super refers to the method of the parent class

Final in Java

Final is a keyword in Java that is used to restrict the user and can be used in many respects. Final can be used with:

- Class
- Methods
- super()

The super keyword can also be used to access the parent class constructor by adding '()' after it, i.e. super(). Also do remember that 'super()' can call both parametric as well as non-parametric constructors depending upon the situation.

Example:

```
// Java code to demonstrate super()

// Class 1

// Helper class

// Parent class - Superclass

class Person {

        // Constructor of superclass

        Person()

        {

  // Print statement of this class

  System.out.println("Person class Constructor");

        }}
```

```
// Class 2
// Helper class
// Subclass extending the above superclass
class Student extends Person {
    Student()
    {
        // Invoking the parent class constructor
        // with the usage of super() word
        super();
        // Print styatement whenever subclass constructor is
        // called
        System.out.println("Student class Constructor");
     }}
// Class 3
// Main class class GFG {
// Main driver method
public static void main(String[] args)
{
      // Creating object of subclass
// inside main() method
Student s = new Student();
   }}
```

Output:

Person class Constructor

Student class Constructor

Polymorphism in Java

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.

Types of Java polymorphism

In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism
- Runtime Polymorphism

METHOD OVERRIDING IN JAVA

- Understanding the problem without method overriding
- Method overloading vs. method overriding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Java Abstraction

Abstract Classes and Methods

Data abstraction is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

- Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).
- An abstract class can have both abstract and regular methods:
- Delegation is simply passing a duty off to someone/something else.
- Delegation can be an alternative to inheritance.
- Delegation means that you use an object of another class as an instance variable, and forward messages to the instance.
- It is better than inheritance for many cases because it makes you to think about each message you forward, because the instance is of a known class, rather than a new class, and because it doesn't force you to accept all the methods of the super class: you can provide only the methods that really make sense.
- Delegation can be viewed as a relationship between objects where one object forwards certain method calls to another object, called its delegate.
- The primary advantage of delegation is run-time flexibility – the delegate can easily be changed at run-time. But unlike inheritance, delegation is not directly supported by most popular object-oriented languages, and it doesn't facilitate dynamic polymorphism.

```
// Java program to illustrate
// delegation
class RealPrinter {
// the "delegate" void print()
{
System.out.println("The Delegate");
}}
class Printer {
// the "delegator"
RealPrinter p = new RealPrinter();
// create the delegate void print()
{
p.print(); // delegation
}}
public class Tester {
// To the outside world it looks like Printer actually prints.
public static void main(String[] args)
{
Printer printer = new Printer(); printer.print();
}}
```

Output:

The Delegate

Summary:

Java is a versatile, object-oriented, and platform-independent programming language widely used for building applications. Known for its "Write Once, Run Anywhere" philosophy, Java is supported by the Java Virtual Machine (JVM), allowing applications to run on any platform.

Check your understanding

Question-1

Which of the following is not a Java features?

a) Dynamic
b) Architecture Neutral
c) Use of pointers
d) Object-oriented

Question-2

Java program processing always starts with main() method

a) TRUE
b) FALSE

Question-3

What is the main purpose of access modifiers in Java?

a) To provide default values
b) To control visibility and access
c) To define the class structure
d) To initialize objects

Question-4

Java supports both Primitive & Non-Primitive (User Defined) datatypes. Which one of the following is not a primitive datatype?

a) byte
b) short
c) long
d) class

Question-5

What is the main purpose of access modifiers in Java?

a) To provide default values
b) To control visibility and access
c) To define the class structure
d) To initialize objects

Question-6

In java, which method is automatically called when an object is created

a) start()
b) main()
c) init()
d) constructor()

Question-7

Which of the following is not an OOPS concept in Java?

a) Polymorphism
b) Inheritance
c) Compilation
d) Encapsulation

Question-8

public class CppBuzz {

public static void main(String[] args){

int a = 5+5*2+2*2+(2*3);

System.out.println(a); }}

a) 138
b) 264
c) 41
d) 25

Question-9

Which OOP principle bundles data and methods into a single unit?

a) Inheritance
b) Abstraction
c) Encapsulation
d) Polymorphism

Question-10

What is false about constructor?

a) Constructors cannot be synchronized in Java
b) Java does not provide default copy constructor
c) Constructor can have a return type
d) "this" and "super" can be used in a constructor

Chapter 5: RDBMS(MySQL)

Introduction

A Relational Database Management System (RDBMS) like MySQL is essential for managing, storing, and retrieving structured data efficiently. It forms the backbone of many applications, from small websites to large-scale enterprise systems. MySQL is open-source, reducing costs while providing a robust feature set for diverse applications.

- Reliable performance for both small and large systems.
- Easy to set up and use with rich documentation.
- Strong community and third-party tool support.

Explain MySql

It is freely available open-source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL). In MySQL database, information is stored in Tables. A single MySQL database can contain many tables at once and store thousands of individual records.

SQL (Structured Query Language)

SQL is a language that enables you to create and operate on relational databases, which are sets of related information stored in tables.

Different Data Models

A data model refers to a set of concepts to describe the structure of a database, and certain constraints (restrictions) that the database should obey. The four data model that are used for database management are:

1. Relational data model: In this data model, the data is organized into tables (i.e. rows and columns). These Tables are called relations.
2. Hierarchical data model 3. Network data model 4. Object Oriented data model RELATIONAL MODEL TERMINOLOGY

Relation: A table storing logically related data is called a Relation.

- **Tuple**: A row of a relation is generally referred to as a tuple.
- **Attribute**: A column of a relation is generally referred to as an attribute.
- **Degree**: This refers to the number of attributes in a relation.
- **Cardinality**: This refers to the number of tuples in a relation.
- **Primary Key:** This refers to a set of one or more attributes that can uniquely identify tuples within the relation.
- **Candidate Key** : All attribute combinations inside a relation that can serve as primary key are candidate keys as the share candidates for primary key position.
- **Alternate Key**: A candidate key that is not primary key, is called an alternate key.

- **Foreign Key**: A non-key attribute, whose values are derived from the primary key of some other table, is known as foreign key in its current table.

REFERENTIAL INTEGRITY

A referential integrity is a system of rules that a DBMS uses to ensure that relationships between records in related tables are valid, and that users don't accidentally delete or change related data. This integrity is ensured by foreign key.

CLASSIFICATION OF SQL STATEMENTS

SQL commands can be mainly divided into following categories:

Data Definition Language (DDL) Commands

Commands that allow you to perform task, related to data definition e.g.

- Creating, altering and dropping.
- Granting and revoking privileges and roles.
- Maintenance commands.

Data Manipulation Language (DML) Commands

Commands that allow you to perform data manipulation e.g., retrieval, insertion, deletion and modification of data stored in a database.

Transaction Control Language (TCL) Commands

Commands that allow you to manage and control the transactions e.g.

- Making changes to database, permanent
- Undoing changes to database, permanent
- Creating save points
- Setting properties for current transactions.

MYSQL ELEMENTS

1. Literals
2. Datatypes
3. Nulls
4. Comments

LITERALS

It refer to a fixed data value. This fixed data value may be of character type or numeric type. For example, 'replay', 'Raj', '8', '306' are all character literals.

Numbers not enclosed in quotation marks are numeric literals. E.g. 22, 18, 1997 are all numeric literals.

Numeric literals can either be integer literals i.e., without any decimal or be real literals i.e. with a decimal point

e.g. 17 is an integer literal but 17.0 and 17.5 are real literals.

DATA TYPES

Data types are means to identify the type of data and associated operations for handling it. MySQL data types are divided into three categories:

- Numeric
- Date and time
- String types

Numeric Data Type

int – used for number without decimal.

Decimal(m,d) – used for floating/real numbers. m denotes the total length of number and d is number of decimal digits.

Date and Time Data Type

date – used to store date in YYYY-MM-DD format.

time – used to store time in HH:MM:SS format.

String Data Types

char(m) – used to store a fixed length string. m denotes max. number of characters.

varchar(m) – used to store a variable length string. m denotes max. no. of characters.

Difference Between Char and Varchar Data Type

Char Datatype	Varchar Datatype
It specifies a fixed length character String.	It specifies a variable length character string.
When a column is given datatype as CHAR(**n**), then MySQL ensures that all values stored in that column have this length i.e. **n** bytes. If a value is shorter than this length **n** then blanks are added, but the size of value remains **n** bytes.	When a column is given datatype as VARCHAR(**n**), then the maximum size a value in this column can have is **n** bytes. Each value that is stored in this column store exactly as you specify it i.e. no blanks are added if the length is shorter than maximum length **n**.

NULL VALUE

If a column in a row has no value, then column is said to be null, or to contain a null. You should use a null value when the actual value is not known or when a value would not be meaningful.

DATABASE COMMNADS VIEW EXISTING DATABASE

To view existing database names, the command is: SHOW DATABASES;

CREATING DATABASE IN MYSQL

For creating the database in MySQL, we write the following Command

CREATE DATABASE <databasename>;

e.g. In order to create a database Student, command is :

CREATE DATABASE Student;

ACCESSING DATABASE

For accessing already existing database, we write:

USE <databasename>;

e.g. to access a database named Student , we write command as :

USE Student;

DELETING DATABASE

For deleting any existing database, the command is:

DROP DATABASE <databasename>;

e.g. to delete a database , say student, we write command as ;

DROP DATABASE Student;

VIEWING TABLE IN DATABASE

In order to view tables present in currently accessed database, command is: SHOW TABLES;

CREATING TABLES IN MYSQL

Tables are created with the CREATE TABLE command. When a table is created, its columns are named, data types and sizes are supplied for each column.

Syntax of CREATE TABLE command is:

CREATE TABLE <table-name>

(<column name><data type>, <column name><data type> , ……) ;

ECODE	ENAME	GENDER	GRADE	GROSS

We write the following command: CREATE TABLE employee (ECODE integer,

ENAME varchar(20) , GENDER char(1) ,

GRADE char(2) , GROSS integer);

INSERTING DATA INTO TABLE

The rows are added to relations (table) using INSERT command of SQL.

Syntax of INSERT is: INSERT INTO <tablename> [<column list>]

VALUE (<value1>, <value2>, …);

e.g. to enter a row into EMPLOYEE table (created above), we write command as : INSERT INTO employee

VALUES(1001 , 'Ravi' , 'M' , 'E' , 50000);

OR

INSERT INTO employee (ECODE, ENAME, GENDER, GRADE, GROSS) VALUES(1001 , 'Ravi' , 'M' , 'E' , 50000);

ECODE	ENAME	GENDER	GRADE	GROSS
1001	Ravi	M	E	50000

In order to insert another row in EMPLOYEE table, we write again INSERT command: INSERT INTO employee

VALUES(1002 , 'Akash' , 'M' , 'A' , 35000);

ECODE	ENAME	GENDER	GRADE	GROSS
1001	Ravi	M	E	50000
1002	Akash	M	A	35000

INSERTING NULL VALUES

To insert value NULL in a specific column, we can type NULL without quotes and NULL will be inserted in that column. E.g. in order to insert NULL value in ENAME column of above table, we write INSERT command as :

INSERT INTO EMPLOYEE VALUES (1004, NULL, 'M', 'B2', 38965);

ECODE	ENAME	GENDER	GRADE	GROSS
1001	Ravi	M	E4	50000
1002	Akash	M	A1	35000
1004	NULL	M	B2	38965

SIMPLE QUERY USING SELECT COMMAND

The SELECT command is used to pull information from a table. Syntax of SELECT command is : SELECT <column name>, <column name>

FROM <tablename> WHERE <condition name>;

ECODE	ENAME	GENDER	GRADE	GROSS
1001	RAVI	M	E4	50000
1002	AKASH	M	A1	35000
1004	NEELA	F	B2	38965
1005	SUNNY	M	A2	30000
1006	RUBY	F	A1	45000

SELECTING ALL DATA

In order to retrieve everything (all columns) from a table, SELECT command is used as :

SELECT * FROM <tablename>;

e.g. In order to retrieve everything from Employee table, we write SELECT command as:

EMPLOYEE

SELECT * FROM Employee;

SELECTING PARTICULAR COLUMNS EMPLOYEE

- A particular column from a table can be selected by specifying column-names with SELECT command.

E.g. in above table, if we want to select ECODE and ENAME column, then command is :

SELECT ECODE, ENAME FROM EMPLOYEE;

E.g.2 in order to select only ENAME, GRADE and GROSS column, the command is:

SELECT ENAME, GRADE, GROSS FROM EMPLOYEE;

SELECTING PARTICULAR ROWS

We can select particular rows from a table by specifying a condition through WHERE clause along with SELECT statement. E.g. In employee table if we want to select rows where Gender is female, then command is:

SELECT * FROM EMPLOYEE

WHERE GENDER = 'F';

E.g.2. in order to select rows where salary is greater than 48000, then command is:

SELECT * FROM EMPLOYEE

WHERE GROSS > 48000; ELIMINATING REDUNDANT DATA

The DISTINCT keyword eliminates duplicate rows from the results of a SELECT statement.

For example,

SELECT GENDER FROM EMPLOYEE;

GENDER
M
M
F
M
F
F

SELECT DISTINCT(GENDER) FROM EMPLOYEE ;

DISTINCT (GENDER)
M
F

SEARCHING FOR NULL

The NULL value in a column can be searched for in a table using IS NULL in the WHERE clause. E.g. to list employee details whose salary, contain NULL, we use the command:

SELECT * FROM EMPLOYEE WHERE GROSS IS NULL;

e.g. STUDENT

Roll_No	**Name**	**Marks**
1	ARUN	NULL
2	RAVI	56
3	SANJAY	NULL

SORTING RESULTS

Whenever the SELECT query is executed, the resulting rows appear in a pre-decided order. The ORDER BY clause allow sorting of query result. The sorting can be done either in ascending or descending order, the default is ascending.

The ORDER BY clause is used as:

SELECT <column name>, <column name>…. FROM <tablename>

WHERE <condition> ORDER BY <column name>;

e.g.To display the details of employees in EMPLOYEE table in alphabetical order, we use command:

SELECT * FROM EMPLOYEE ORDER BY ENAME;

Output will be:

ECODE	ENAME	GENDER	GRADE	GROSS
1002	Akash	M	A1	35000
1004	Neela	F	B2	38965
1009	Neema	F	A2	52000
1001	Ravi	M	E4	50000
1006	Ruby	F	A1	45000
1005	Sunny	M	A2	30000

MODIFYING DATA IN TABLES : You can modify data in tables using UPDATE command of SQL. The UPDATE command specifies the rows to be changed using the WHERE clause, and the new data using the SET keyword. Syntax of update command is:

UPDATE <tablename>

SET <columnname>=value, <columnname>=value WHERE <condition>;

e.g. to change the salary of employee of those in EMPLOYEE table having employee code 1009 to 55000.

UPDATE EMPLOYEE SET GROSS = 55000 WHERE ECODE = 1009;

UPDATING MORE THAN ONE COLUMNS

to update the salary to 58000 and grade to B2 for those

employee whose employee code is 1001.

UPDATE EMPLOYEE

SET GROSS = 58000, GRADE='B2'

WHERE ECODE = 1009;

OTHER EXAMPLES

Increase the salary of each employee by 1000 in the EMPLOYEE table.

UPDATE EMPLOYEE

SET GROSS = GROSS +100;

Double the salary of employees having grade as 'A1' or 'A2' .

UPDATE EMPLOYEE

SET GROSS = GROSS * 2;

WHERE GRADE='A1' OR GRADE='A2';

Change the grade to 'A2' for those employees whose employee code is 1004 and name is Neela.

UPDATE EMPLOYEE

SET GRADE='A2'

WHERE ECODE=1004 AND GRADE='NEELA';

DELETING DATA FROM TABLES

To delete some data from tables, DELETE command is used. The DELETE command removes rows from a table. The syntax of DELETE command is:

DELETE FROM <tablename>

WHERE <condition>;

For example, to remove the details of those employee from EMPLOYEE table whose grade is A1?

DELETE FROM EMPLOYEE

WHERE GRADE ='A1';

TO DELETE ALL THE CONTENTS FROM A TABLE DELETE FROM EMPLOYEE;

So, if we do not specify any condition with WHERE clause, then all the rows of the table will be deleted. Thus, above line will delete all rows from employee table.

DROPPING TABLES

The DROP TABLE command lets you drop a table from the database. The syntax of DROP TABLE command is:

DROP TABLE <tablename>;

e.g. to drop a table employee, we need to write :

DROP TABLE employee;

Once this command is given, the table name is no longer recognized and no more commands can be given on that table. After this command is executed, all the data in the table along with table structure will be deleted.

MODIFYING COLUMNS

Column name and data type of column can be changed as per following syntax:

ALTER TABLE <table name>

CHANGE <old column name><new column name><new datatype>; If Only data type of column need to be changed, then

ALTER TABLE <table name>

MODIFY <column name><new datatype>;

e.g. 1.In table EMPLOYEE, change the column GROSS to SALARY.

ALTER TABLE EMPLOYEE

CHANGE GROSS SALARY INTEGER;

e.g.2. In table EMPLOYEE , change the column ENAME to EM_NAME and data type from VARCHAR(20) to VARCHAR(30).

ALTER TABLE EMPLOYEE

CHANGE ENAME EM_NAME VARCHAR(30);

e.g.3. In table EMPLOYEE , change the datatype of GRADE column from CHAR(2) to VARCHAR(2).

ALTER TABLE EMPLOYEE MODIFY GRADE VARCHAR(2); DELETING COLUMNS

To delete a column from a table, the ALTER TABLE command takes the following form:

ALTER TABLE <table name>

DROP <column name>;

e.g. to delete column GRADE from table EMPLOYEE, we will write :

ALTER TABLE EMPLOYEE

DROP GRADE;

ADDING/REMOVING CONSTRAINTS TO A TABLE

ALTER TABLE statement can be used to add constraints to your existing table by using it in following manner:

TO ADD PRIMARY KEY CONSTRAINT

ALTER TABLE < table name >

ADD PRIMARY KEY (Column name);

e.g. to add PRIMARY KEY constraint on column ECODE of table EMPLOYEE ,

the command is :

ALTER TABLE EMPLOYEE

ADD PRIMARY KEY (ECODE);

TO ADD FOREIGN KEY CONSTRAINT

ALTER TABLE <table name>

ADD FOREIGN KEY (Column name) REFERENCES Parent Table

(Primary key of Parent Table);

REMOVING CONSTRAINTS

- To remove primary key constraint from a table, we use ALTER TABLE command as:
  ALTER TABLE <table name>

DROP PRIMARY KEY;

- To remove foreign key constraint from a table, we use ALTER TABLE command as:
  ALTER TABLE <table name>

DROP FOREIGN KEY;

ENABLING/DISABLING CONSTRAINTS

Only foreign key can be disabled/enabled in MySQL.

To disable foreign keys:

SET FOREIGN_KEY_CHECKS = 0;

To enable foreign keys:

SET FOREIGN_KEY_CHECKS = 1; INTEGRITY CONSTRAINTS/CONSTRAINTS

A constraint is a condition or check applicable on a field (column) or set of fields (columns). Common types of constraints include:

1)	**NOT NULL**	**Ensures that a column cannot have NULL value**
2)	**DEFAULT**	**Provides a default value for a column when none is specified**
3)	**UNIQUE**	**Ensures that all values in a column are different**
4)	**CHECK**	**Makes sure that all values in a column satisfy certain criteria**
5)	**PRIMARY KEY**	**Used to uniquely identify a row in the table**
6)	**FOREIGN KEY**	**Used to ensure referential integrity of the data**

Functions in MySQL

1) AVG ( )

This function computes the average of given data. e.g. SELECT AVG(SAL) FROM EMPL ;

2) COUNT( )

This function counts the number of rows in a given column.

If you specify the COLUMN name in parenthesis of function, then this function returns rows where COLUMN is not null.

If you specify the asterisk (*), this function returns all rows, including duplicates and nulls.

e.g. SELECT COUNT(*) FROM EMPL ;

e.g.2 SELECT COUNT(JOB) FROM EMPL ;

3) MAX( )

This function returns the maximum value from a given column or expression.

e.g. SELECT MAX(SAL) FROM EMPL ;

4) MIN( )

This function returns the minimum value from a given column or expression.

EMPNO	ENAME	JOB	SAL	DEPTNO
8369	SMITH	CLERK	2985	10
8499	ANYA	SALESMAN	9870	20
8566	AMIR	SALESMAN	8760	30
8698	BINA	MANAGER	5643	20

e.g. SELECT MIN(SAL) FROM EMPL ;

5) SUM( )

This function returns the sum of values in given column or expression.

e.g. SELECT SUM(SAL) FROM EMPL ;

GROUPING RESULT – GROUP BY

The GROUP BY clause combines all those records (row) that have identical values in a particular field (column) or a group of fields (columns).

GROUPING can be done by a column name, or with aggregate functions in which case the aggregate produces a value for each group.

DATABASE TRANSACTIONS

A Transaction is a logical unit of work that must succeed or fail in its entirety. This statement means that a transaction may involve many sub steps, which should either all be carried out successfully or all be ignored if some failure occurs. A Transaction is an atomic operation which may not be divided into smaller operations.

Example of a Transaction

Begin transaction

Get balance from account X Calculate new balance as X – 1000 Store new balance into database file

Get balance from account Y Calculate new balance as Y + 1000 Store new balance into database file End transaction

TRANSACTION PROPERTIES (ACID PROPERTIES)

ATOMICITY(All or None Concept)–This property ensures that either all operations of the transaction are carried outor none are.

CONSISTENCY–This property implies that if the database was in a consistent state before the start of transaction execution, then upon termination of transaction, the database will also be in a consistent state.

ISOLATION–This property implies that each transaction is unaware of other transactions executing concurrently inthe system.

DURABILITY–This property of a transaction ensures that after the successful completion of a transaction, the changes made by it to the database persist, even if there are system failures.

TRANSACTION CONTROL COMMANDS (TCL)

The TCL of MySQL consists of following commands:

BEGIN or START TRANSACTION – marks the beginning of a transaction.

COMMIT – Ends the current transaction by saving database changes and starts a new transaction.

ROLLBACK – Ends the current transaction by discarding database changes and starts a new transaction.

SAVEPOINT – Define breakpoints for the transaction to allow partial rollbacks.

SET AUTOCOMMIT – Enables or disables the default auto commit mode.

Summary:

Learning MySQL is highly beneficial because it equips you with the skills to manage and interact with databases, which are a crucial part of most modern applications.

Check your understanding

Question 1

Which of the following function returns the current date and time in MySql ?

a) CURDATE()
b) Now()
c) CURTIME()
d) DATE()

Question 2

Which of the following function returns the current date in MySql ?

a) CURDATE()
b) Now()
c) CURTIME()
d) DATE()

Question 3

Which of the following function returns the current time in MySql ?

a) CURDATE()
b) Now()
c) CURTIME()
d) DATE()

Question 4

"CREATE TABLE…" Command is used to create which type of table in Mysql ?

a) Permanent Tables
b) Virtual Tables
c) Temporary Tables
d) Both 2 and 3

Question 5

In which language MYSQL is written?

a) PYTHON
b) C/C++
c) JAVA
d) COBOL

Question 6

To see the list of options provided by MYSQL which of the following command is used?

a) HELP
b) –HELP
c) ="-- HELP"
d) None of these

Question 7

To know your MYSQL version which of the following command you should use?

a) VERSION;
b) SELECT VERSION;
c) SELECT VERSION();
d) SELECT VERSON();

Question 8

Can you change the column name using alter command?

a) YES
b) NO

Question 9

What does the SHOW TABLES command do?

a) It displays all the tables of all the databases in the machine.
b) It displays all the tables of a particular database.
c) It only displays the current table.
d) None of these

Question 10

What is the function of DESCRIBE statement?

a) This statement helps us to get the details of the entire row.
b) This statement helps us to get the definition of a particular table at a time.
c) This statement helps us to get the definition of all the tables.
d) None of these

Chapter 6: HTML5

Learning Objective:

HTML stands for HyperText Markup Language. This is the standard language for building web pages and is used to create the general structure of a website/web page. HTML tells the web browser how to display the web site's content when the user loads the website.

HTML (HyperText Markup Language)

HTML is the standard markup language used to create web pages. It's the backbone of a website, providing the structure and content that the web browser renders to the user. **HTML5** is the fifth and latest version of the HTML (HyperText Markup Language), used to structure and present content on the web. It introduces several new features that make web development more powerful, efficient, and user-friendly.

KEY FEATURES OF HTML5

New Semantic Elements:

<header>, <footer>, <article>, <section>, <nav>, and others help structure the content meaningfully, making it more accessible and improving SEO.

Multimedia Support:

Audio: <audio> allows embedding sound content directly on web pages.

Video: <video> allows embedding video files with support for controls, autoplay, and more.

Form Improvements:

New input types: <input type="email">, <input type="date">, <input type="number">, etc.

Improved form validation using HTML attributes like required, pattern, and min/max.

Canvas for Drawing:

<canvas> is used to draw graphics and animations on the fly via JavaScript, commonly used for games and visualizations.

Local Storage & Session Storage:

localStorage and sessionStorage allow storing data on the client's browser, which can be used without needing a database or server.

Geolocation:

HTML5 supports a geolocation API to get the geographic location of the user.

Web Workers:

Allows background processing (multithreading), improving performance for tasks like processing large data sets.

WebSockets:

Supports real-time communication between the client and server over a persistent connection.

Offline Capabilities:

HTML5 provides support for offline applications through features like the Application Cache and Service Workers.

APIs (Application Programming Interfaces):

HTML5 introduces several APIs, like the Geolocation API, Web Storage API, and Device Orientation API, to extend functionality.

BASIC HTML COMPONENTS

- Elements: Tags that define content (e.g., headings, paragraphs, images).
- Attributes: Modify element behavior (e.g., href, src, alt).
- Tags: Surround content to define elements.
- Document Structure: HTML, head, body, and other essential elements.

HTML VERSIONS

- HTML 1.0 (1993): Initial version.
- HTML 2.0 (1995): Added tables, forms, and images.
- HTML 3.2 (1997): Introduced stylesheets and scripts.
- HTML 4.01 (1999): Improved accessibility and internationalization.
- HTML5 (2014): Added multimedia, canvas, and web storage.

HTML EDITORS

Text Editors

- Notepad++ (Free)
- Atom (Free)
- Visual Studio Code (Free)

Key Points:

- Root element: The <html> tag is the top-most element in the HTML document hierarchy.
- Container: It wraps around all other HTML elements.
- Language declaration: The lang attribute should be included inside the <html> tag to declare the language of the web page.
- Best practice: Always include the lang attribute to improve accessibility and search engine optimization (SEO).

Example:

➔ <!DOCTYPE html>
➔ <html lang="en">
➔ <head>
➔ ...
➔ </head>
➔ <body>
➔ ...
➔ </body>
➔ </html>

List of HTML tags and attributes:

HTML TAGS:

- Structural Tags
- <html> - Root element
- <head> - Header section
- <body> - Body section
- <title> - Page title

Headings Tag

1. <h1> - Main heading
2. <h2> - Subheading
3. <h3> - Sub-subheading
4. <h4> - Sub-sub-subheading
5. <h5> - Sub-sub-sub-subheading
6. <h6> - Sub-sub-sub-sub-subheading

Text Formatting

- ➔ <p> - Paragraph
- ➔ - Inline text container
- ➔ - Bold text
- ➔ <strike> - Strikethrough text
- ➔ <u> - Underlined text

Lists

- ➢ - Unordered list
- ➢ - Ordered list
- ➢ - List item
- ➢ <dl> - Definition list
- ➢ <dt> - Definition term
- ➢ <dd> - Definition description

Links

<a> - Anchor (link)

<link> - Link to external stylesheet

Images

- ➢ <img> - Image
- ➢ <figure> - Figure
- ➢ <figcaption> - Figure caption

Tables

- ➢ <table> - Table
- ➢ <tr> - Table row
- ➢ <td> - Table data
- ➢ <th> - Table header
- ➢ <caption> - Table caption

Forms

- <form> - Form
- <input> - Input field
- <textarea> - Text area
- <select> - Dropdown select
- <option> - Option
- <button> - Button

Semantic Tags

- <header> - Header section
- <nav> - Navigation section
- <main> - Main content section
- <section> - Content section
- <article> - Article
- <footer> - Footer section

HTML ATTRIBUTES:

Global Attributes

- id - Unique identifier
- class - CSS class
- style - Inline CSS
- title - Element title
- lang - Language attribute

Link Attributes

- href - Link URL
- target - Link target
- rel - Link relationship

Image Attributes

- src - Image URL
- alt - Image description
- width - Image width
- height - Image height

Table Attributes

- border - Table border
- cellpadding - Cell padding
- cellspacing - Cell spacing

EXAMPLE 1: BASIC HTML STRUCTURE

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This is my first web page.</p>
</body>
</html>
```

EXAMPLE 2: LINK AND IMAGE

```
<!DOCTYPE html>
<html>
<body>
  <a href="(link unavailable)">Visit Google</a>
  <img src="image.jpg" alt="My Image" width="500" height="300">
</body>
</html>
```

EXAMPLE 3: UNORDERED LIST

```
<!DOCTYPE html>
<html>
<body>
  <h1>My Favorite Fruits</h1>
  <ul>
    <li>Apple</li>
    <li>Banana</li>
    <li>Cherry</li>
  </ul>   </body>            </html>
```

EXAMPLE 4: TABLE

```
<!DOCTYPE html>
<html>
<body>
  <h1>Student Information</h1>
  <table border="1">
    <tr>
      <th>Name</th>
      <th>Age</th>
    </tr>
    <tr>
      <td>John Doe</td>
      <td>20</td>
    </tr>
    <tr>
      <td>Jane Doe</td>
      <td>22</td>
    </tr>
  </table>
</body>
</html>
```

EXAMPLE 5: FORM

```
<!DOCTYPE html>
<html>
<body>
  <h1>Contact Form</h1>
  <form action="/submit" method="post">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name"><br><br>
    <label for="email">Email:</label>
```

```
    <input type="email" id="email" name="email"><br><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

EXAMPLE 6: SEMANTIC HTML

```
<!DOCTYPE html>
<html>
<body>
  <header>
    <h1>My Website</h1>
  </header>
  <nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>
  <main>
    <section>
      <h2>Welcome</h2>
      <p>This is my website.</p>
    </section>
  </main>
  <footer>
    <p>&copy; 2024 My Website</p>
  </footer>
</body>
</html>
```

LIST OF HTML COLOR CODES:

Color Names

- Black: black / #000000
- White: white / #FFFFFF
- Red: red / #FF0000
- Green: green / #008000
- Blue: blue / #0000FF
- Yellow: yellow / #FFFF00

HEXADECIMAL COLOR CODES

- #FF0000 - Red
- #008000 - Green
- #0000FF - Blue
- #FFFF00 - Yellow
- #800080 - Purple
- #FFC0CB - Pink

RGB COLOR CODES

- rgb(255, 0, 0) - Red
- rgb(0, 128, 0) - Green
- rgb(0, 0, 255) - Blue
- rgb(255, 255, 0) - Yellow
- rgb(128, 0, 128) - Purple
- rgb(255, 192, 203) - Pink

HTML Color Attributes

- bgcolor - Background color
- text - Text color
- link - Link color
- vlink - Visited link color
- alink - Active link color

HTML Color Code Examples

- <body bgcolor="#FFFFFF">
- <p style="color: blue;">
- <a href="#" style="color: red;">
-

The <iframe> tag :

It is used to embed another HTML document within the current HTML document, creating an inline frame.

Attributes:

- src: Specifies the URL of the document to embed.
- width and height: Define the size of the iframe.
- frameborder: Specifies the border style (0 = no border, 1 = default border).
- scrolling: Controls scrollbar visibility (yes/no/auto).
- srcdoc: Specifies the HTML content to display within the iframe.

Video Embedding:

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

Audio Embedding

```
<audio controls>
  <source src="audio.mp3" type="audio/mp3">
  Your browser does not support the audio element.
</audio>
```

Example:

```
<iframe src="(link unavailable)" width="500" height="300" frameborder="0" scrolling="yes"></iframe>
```

HTML forms are used to collect user input, which is then sent to a server for processing.

Form Basics

- form tag: Defines the form.
- action attribute: Specifies the server-side script to handle the form data.
- method attribute: Specifies the HTTP method (GET, POST, PUT, DELETE).
- enctype attribute: Specifies the data encoding type (e.g., application/x-www-form-urlencoded).

Form Elements

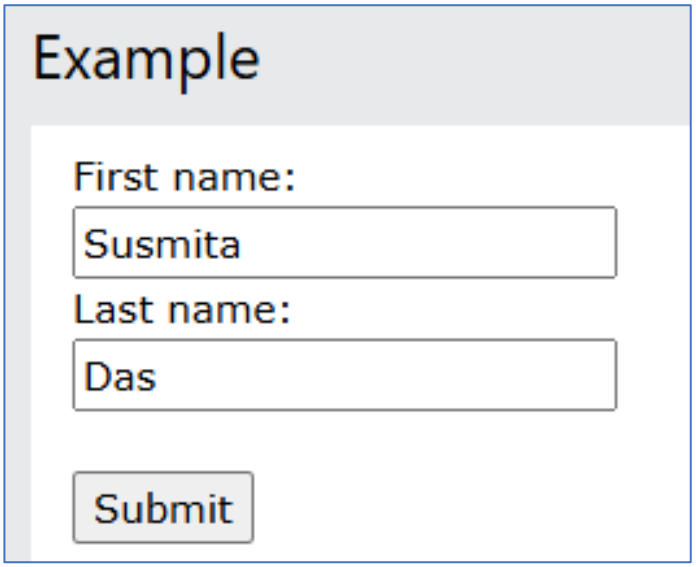
- Input Tag: Text Fields, Checkboxes, Radio Buttons, Submit Buttons.
- Textarea Tag: Multi-Line Text Input.
- Select Tag: Dropdown Menus.
- Option Tag: Options Within A Select Menu.
- Label Tag: Labels For Form Elements.
- Fieldset Tag: Groups Related Form Elements.
- Legend Tag: Describes The Fieldset.

Form Attributes

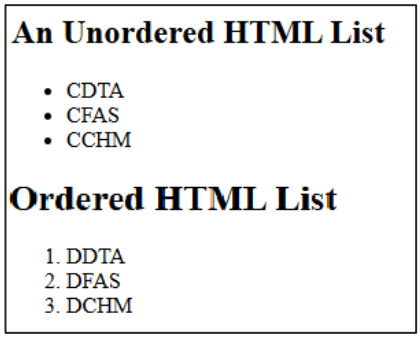
- name attribute: Identifies the form element.
- value attribute: Specifies the initial value.
- placeholder attribute: Provides a hint to the user.
- required attribute: Ensures the field is filled.
- disabled attribute: Disables the field.

Form Submission

- submit button: Submits the form.
- reset button: Resets the form fields.
- JavaScript: Can be used to validate and submit forms.



Ordered List and Unordered List




```
<html>  <body>
<h2>An Unordered HTML List</h2>
<ul>
  <li>CDTA</li>
  <li>CFAS</li>
  <li>CCHM</li>
</ul>
<h2>An Ordered HTML List</h2>
<ol>
  <li>DDTA</li>
  <li>DFAS</li>
  <li>DCHM</li>
</ol>
</body>
</html>
```

HTML <IMG> TAG

The <img> tag is used to embed an image in an HTML page.

Images are not technically inserted into a web page; images are linked to web pages. The <img> tag creates a holding space for the referenced image.

The <img> tag has two required attributes:

- src - Specifies the path to the image
- alt - Specifies an alternate text for the image, if the image for some reason cannot be displayed
- <img src="/Pictures/Pic1.jpg" alt="Stickman" width="24" height="39">

HTML is used to provide structure to a webpage and make it accessible to users of the internet through text, visual formatting and search factors. It is commonly known as the most basic building block of the web, working alongside CSS and JavaScript to create the websites we see while browsing.

HTML table tag

The HTML table tag is used to define a table in an HTML document.

BASIC TABLE STRUCTURE

```
<table>
  <tr>
    <th>Header 1</th>        <th>Header 2</th>
  </tr>
  <tr>
    <td>Cell 1</td>          <td>Cell 2</td>
  </tr>   </table>
```

Table Tags

- table: Defines the table.
- tr: Defines a table row.
- th: Defines a table header cell.
- td: Defines a table data cell.
- caption: Defines the table caption.
- col: Defines a table column.
- colgroup: Defines a group of columns.
- thead: Defines the table header section.
- tbody: Defines the table body section.
- tfoot: Defines the table footer section.

Table Attributes

- border: Specifies the border width.
- cellpadding: Specifies the cell padding.
- cellspacing: Specifies the cell spacing.
- width: Specifies the table width.
- height: Specifies the table height.
- align: Specifies the table alignment.
- bgcolor: Specifies the table background color.

TABLE EXAMPLES

Simple Table

```
<table>  <tr>      <th>Name</th>          <th>Age</th>
          </tr>
 <tr>      <td>John</td>          <td>25</td>   </tr>
 <tr>      <td>Jane</td>          <td>30</td>   </tr>
</table>
```

TABLE WITH CAPTION

```
<table>  <caption>Student Information</caption>
 <tr>      <th>Name</th>          <th>Age</th>
 </tr>
 <tr>      <td>SUMAN</td>        <td>25</td>  </tr>
<tr>       <td>SAMIR</td>        <td>30</td>
 </tr>  </table>
```

Table with Colspan and Rowspan

```
<table>  <tr>
   <th colspan="2">Name</th>     <th>Age</th>
 </tr>
<tr>  <td>PRIYA</td>     <td>DAS</td>          <td>25</td>   </tr>
 <tr>      <td rowspan="2">Who ?</td>   <td>ANIL</td>   <td>30</td>  </tr>
 <tr>      <td>SUSMITA</td>      <td>35</td>  </tr>     </table>
```

Summary:

HTML5 is the latest version of HyperText Markup Language, designed to improve the structure, functionality, and interactivity of web content. It focuses on making web pages more dynamic, mobile-friendly, and accessible while supporting multimedia and real-time features without needing third-party plugins.

Check your understanding

Question:1

What does HTML stand for?

a. Hot Typing Markup Language
b. Home Typing Modern Language
c. Hyper Text Markup Language
d. Home Testing Mixed Language

Question:2

Which one of the following headers has the correct HTML syntax?

a. <h1>Welcome</h1>
b. {{h1}}Welcome{{:h1}}
c. {h1:Welcome}

Question:3

Who invented HTML?

a. Tim Berners-Lee
b. Brendan Eich
c. Guido van Rossum

Question:4

The correct sequence of HTML tags for starting a webpage is -

a. Head, Title, HTML, body
b. HTML, Body, Title, Head
c. HTML, Head, Title, Body
d. HTML, Head, Title, Body

Question:5

Which of the following element is responsible for making the text bold in HTML?

a. <pre>
b. <a>
c. <b>
d.

Question:6

Which of the following tag is used to insert a line-break in HTML?

a.
b. <a>
c. <pre>
d. <b>

Question:7

How to create an unordered list (a list with the list items in bullets) in HTML?

a.
b.
c.
d. <i>

Question:8

Which character is used to represent the closing of a tag in HTML?

a. \
b. !
c. /
d. .

Question:9

How to insert an image in HTML?

a.
b.
c.
d.

Question:10

<input> is -

a. a format tag.
b. an empty tag.
c. All of the above
d. None of the above

THE END

Great! Practice Regularly.