Course Code	**:**	**MCS-201**
Course Title	**:**	**Programming in C and PYTHON**
Assignment Number	**:**	**PGDCA(I)/201/Assignment/2025**
Maximum Marks	**:**	**100**
Weightage	**:**	**30%**
Last Date of Submission	**:**	**30th April, 2025 (for January session)** **31st October, 2025 (for July session)**

There are ten questions in this assignment which carries 80 marks. Each question carries 8 marks. Rest 20 marks are for viva-voce. Answer all the questions from both the sections i.e. Section A and Section B. You may use illustrations and diagrams to enhance the explanations. Include the screen layouts also along with your assignment responses. Please go through the guidelines regarding assignments given in the Programme Guide for the format of presentation.

SECTION-A (C-Programming)

Question 1: Write an algorithm, draw a flow chart and write its corresponding C program to convert a Binary decimal number to its equivalent Decimal number. **(8 Marks)**

Question 2: Write an algorithm and use the concept of Structures to write the program in C, to generate Progress-Report of students of a class X of the school for all its 4 terms (the class is of 20 students). Assumptions can be made wherever necessary. **(8 Marks)**

Question 3: Write a C program to generate the following pattern: **(8 Marks)**

```
*
* *
* * *
* * * *
* * * * *
```

Question 4: Write a C program to perform the following operation on matrices D = A * (B + C), where A, B and C are matrices of (3 X 3) size and D is the resultant matrix. **(8 Marks)**

Question 5: Use the concept of File Handling, to Write a program in C, to collect a list of N numbers in a file, and separate the even and odd numbers from the given list of N numbers, and put them in two separate files namely even_file and odd_file, respectively. **(8 Marks)**

SECTION-B (PYTHON-Programming)

Question 6: Write Python code to perform the following: **(8 Marks)**

(i) Copy content of file first.txt to second.txt
(ii) Reading a file
(iii) Writing into a file
(iv) Appending into a file

Question 7: Write an algorithm to find the slope of a line segment whose endpoint coordinates are $(x1, y1)$ and $(x2, y2)$. The algorithm gives output whether the slope is positive, negative or zero. Transform your algorithm into Python program. **(8 Marks)**

Note: Slope of line segment = $(y2 - y1)/(x2-x1)$.

Question 8: Write a program in Python to create a package named Volume and create 3 module in it named – Cube, Cuboid and Sphere each having a function to calculate Volume of Cube, Cuboid and Sphere respectively. Import the module in separate location and use the functions. Assumptions can be made wherever necessary. Support your program with suitable comments to improve readability. **(8 Marks)**

Question 9: Write a program in Python to perform following: **(8 Marks)**

- To find square root of numbers in a list using lambda function.
- To display first n lines from a file, where n is given by user.
- To display size of a file in bytes
- To display frequency of each word in a file.

Question 10: What are Co-routines? How Co-routines differ from threads? How Co-routines support cooperative multi-tasking in python? Compare Subroutines and Co-routines. **(8 Marks)**

Course Code : MCS-201

Course Title : Programming in C and PYTHON

ECTION-A (C-Programming)

Question 1: Write an algorithm, draw a flow chart and write its corresponding C program to convert a Binary decimal number to its equivalent Decimal number.

Algorithm to Convert a Binary Number to a Decimal Number

1. Start.
2. Take the binary number as input.
3. Initialize decimal = 0 and base = 1 (since the rightmost bit represents 2^0).
4. Extract the last digit (rightmost bit) of the binary number using binary % 10.
5. Multiply the extracted bit by base and add the result to decimal.
6. Divide the binary number by 10 to remove the processed digit.
7. Multiply base by 2 (since binary is base-2).
8. Repeat steps 4-7 until the binary number becomes 0.
9. Print the decimal result.
10. End.

Flowchart for Binary to Decimal Conversion

The flowchart visually represents the above algorithm:

1. Start
2. Input Binary Number
3. Initialize decimal = 0, base = 1

4. Repeat:
    - Extract last digit (digit = binary % 10)
    - Multiply digit with base and add to decimal
    - Update binary = binary / 10
    - Update base = base * 2
    - Continue until binary == 0
5. Output the Decimal Number
6. End

C Program to Convert Binary to Decimal

Below is the C program implementing the above logic:

```
#include <stdio.h>

int binaryToDecimal(long long binary) {
    int decimal = 0, base = 1, remainder;

    while (binary > 0) {
        remainder = binary % 10;  // Extract last digit
        decimal += remainder * base;  // Convert to decimal
        binary /= 10;  // Remove last digit
        base *= 2;  // Update base
    }

    return decimal;
}

int main() {
    long long binary;

    // Input binary number
    printf("Enter a binary number: ");
    scanf("%lld", &binary);

    // Convert and print result
    printf("Decimal equivalent: %d\n", binaryToDecimal(binary));
```

```
    return 0;
}
```

Explanation of the C Program

The program first takes a binary number as input from the user. It then iterates through each digit, converting it into its decimal equivalent by multiplying with increasing powers of 2. The result is stored in decimal, and the final converted value is displayed as output.

Question 2: Write an algorithm and use the concept of Structures to write the program in C, to generate Progress-Report of students of a class X of the school for all its 4 terms (the class is of 20 students). Assumptions can be made wherever necessary.

C Program Using Structures for Progress Report

Below is the C program implementing the algorithm:

```
#include <stdio.h>

// Define structure for Student
struct Student {
    char name[50];
    int roll_no;
    float marks[4];  // Stores marks for 4 terms
};

// Function to display progress report
void displayReport(struct Student students[], int n) {
    printf("\n--- Progress Report of Class X ---\n");
    printf("-----------------------------------------------------------\n");
    printf("Roll No | Name          | Term 1 | Term 2 | Term 3 | Term 4 | Total |
Average\n");
    printf("-----------------------------------------------------------\n");

    for (int i = 0; i < n; i++) {
        float total = 0;
        for (int j = 0; j < 4; j++) {
            total += students[i].marks[j];
        }
        float average = total / 4;

        // Print student details
        printf("%7d | %-20s | %6.2f | %6.2f | %6.2f | %6.2f | %5.2f | %7.2f%%\n",
```

```
            students[i].roll_no, students[i].name,
            students[i].marks[0], students[i].marks[1],
            students[i].marks[2], students[i].marks[3],
            total, average);
    }
    printf("-----------------------------------------------------------\n");
}

int main() {
    struct Student students[20];  // Array of 20 students

    // Input details for each student
    printf("Enter details for 20 students:\n");
    for (int i = 0; i < 20; i++) {
        printf("\nEnter details for student %d:\n", i + 1);
        printf("Enter Roll Number: ");
        scanf("%d", &students[i].roll_no);
        printf("Enter Name: ");
        scanf(" %[^\n]", students[i].name);  // To read full name with spaces

        for (int j = 0; j < 4; j++) {
            printf("Enter marks for Term %d: ", j + 1);
            scanf("%f", &students[i].marks[j]);
        }
    }

    // Display the progress report
    displayReport(students, 20);

    return 0;
}
```

Explanation of the C Program

The program defines a structure Student to hold student information, including roll number, name, and marks for four terms. An array of structures is used to store details for 20 students. The program takes input for each student, calculates the total marks and average percentage, and then displays a formatted progress report.

Question 3: Write a C program to generate the following pattern:

```
*
* *
* * *
* * * *
* * * * *
```

C Program to Print the Given Pattern

```
#include <stdio.h>

int main() {
   int i, j, rows = 5; // Number of rows in the pattern

   // Loop for each row
   for (i = 1; i <= rows; i++) {
      // Loop for printing asterisks
      for (j = 1; j <= i; j++) {
         printf("* ");
      }
      printf("\n"); // Move to the next line after printing each row
   }

   return 0;
}
```

Explanation of the Program

1. The program uses nested loops:
    - The outer loop (i) runs from 1 to 5, representing each row.
    - The inner loop (j) runs from 1 to i, printing i stars in each row.
2. A printf("\n") moves to the next line after printing each row.
3. The result is a right-angled triangle with 5 rows.

Question 4: Write a C program to perform the following operation on matrices D = A * (B + C), where A, B and C are matrices of (3 X 3) size and D is the resultant matrix.

C Program to Perform Matrix Operation: $D = A \times (B + C)$

This program performs the matrix operation where three 3×3 matrices A, B, and C are given, and we calculate:

$$D = A \times (B + C)$$

C Program Implementation

```
#include <stdio.h>

// Function to read a 3x3 matrix
void readMatrix(int matrix[3][3], char name) {
   printf("Enter elements for matrix %c (3x3):\n", name);
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
         scanf("%d", &matrix[i][j]);
      }
   }
}

// Function to print a 3x3 matrix
void printMatrix(int matrix[3][3], char name) {
   printf("\nMatrix %c:\n", name);
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
         printf("%5d ", matrix[i][j]);
      }
      printf("\n");
   }
}

// Function to add two 3x3 matrices
void addMatrices(int A[3][3], int B[3][3], int result[3][3]) {
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
         result[i][j] = A[i][j] + B[i][j];
      }
   }
}

// Function to multiply two 3x3 matrices
void multiplyMatrices(int A[3][3], int B[3][3], int result[3][3]) {
   for (int i = 0; i < 3; i++) {
```

```
            for (int j = 0; j < 3; j++) {
                result[i][j] = 0; // Initialize result cell to 0
                for (int k = 0; k < 3; k++) {
                    result[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    }

int main() {
    int A[3][3], B[3][3], C[3][3], sumBC[3][3], D[3][3];

    // Read matrices A, B, and C
    readMatrix(A, 'A');
    readMatrix(B, 'B');
    readMatrix(C, 'C');

    // Compute (B + C)
    addMatrices(B, C, sumBC);

    // Compute D = A * (B + C)
    multiplyMatrices(A, sumBC, D);

    // Display the results
    printMatrix(A, 'A');
    printMatrix(B, 'B');
    printMatrix(C, 'C');
    printMatrix(sumBC, 'B+C');
    printMatrix(D, 'D'); // Final result

    return 0;
}
```

Explanation of the Program

1. Matrix Input & Output Functions:
   - `readMatrix()` : Reads a 3×3 matrix from the user.
   - `printMatrix()` : Prints a 3×3 matrix.
2. Matrix Operations:
   - `addMatrices()` : Adds two 3×3 matrices element-wise.
   - `multiplyMatrices()` : Multiplies two 3×3 matrices using the standard row-column multiplication.
3. Main Logic:
   - Matrices A, B, and C are read from user input.
   - $B + C$ is calculated and stored in `sumBC`.
   - $D = A \times (B + C)$ is computed using matrix multiplication.
   - The matrices and results are displayed in a structured format.

Question 5: Use the concept of File Handling, to Write a program in C, to collect a list of N numbers in a file, and separate the even and odd numbers from the given list of N numbers, and put them in two separate files namely even_file and odd_file, respectively.

C Program Using File Handling to Separate Even and Odd Numbers

This program does the following:

1. Takes N numbers as input from the user.
2. Stores them in a file called numbers.txt.
3. Reads the numbers from the file and separates them into:
   - even_file.txt (for even numbers)
   - odd_file.txt (for odd numbers)

C Program

```
#include <stdio.h>

int main() {
   FILE *inputFile, *evenFile, *oddFile;
```

```
int N, num;

// Open the input file for writing
inputFile = fopen("numbers.txt", "w");
if (inputFile == NULL) {
   printf("Error opening file for writing!\n");
   return 1;
}

// Get the count of numbers from the user
printf("Enter the number of elements (N): ");
scanf("%d", &N);

// Collect numbers from user and write to file
printf("Enter %d numbers:\n", N);
for (int i = 0; i < N; i++) {
   scanf("%d", &num);
   fprintf(inputFile, "%d ", num);
}

// Close the input file after writing
fclose(inputFile);

// Open input file for reading and separate even and odd numbers
inputFile = fopen("numbers.txt", "r");
evenFile = fopen("even_file.txt", "w");
oddFile = fopen("odd_file.txt", "w");

if (inputFile == NULL || evenFile == NULL || oddFile == NULL) {
   printf("Error opening files!\n");
   return 1;
}

// Read numbers and classify them into even and odd
while (fscanf(inputFile, "%d", &num) != EOF) {
   if (num % 2 == 0)
      fprintf(evenFile, "%d ", num);
   else
      fprintf(oddFile, "%d ", num);
}

// Close all files
fclose(inputFile);
```

```
    fclose(evenFile);
    fclose(oddFile);

    printf("Numbers have been separated into even_file.txt and odd_file.txt
successfully.\n");

    return 0;
}
```

Explanation of the Program

1. **Writing to File (numbers.txt)**:
    - The user enters N numbers.
    - These numbers are stored in a file called numbers.txt.
2. **Reading and Separating Numbers**:
    - The program reads numbers from numbers.txt using fscanf().
    - If a number is even, it is written to even_file.txt.
    - If a number is odd, it is written to odd_file.txt.
3. **File Handling Operations**:
    - fopen("file_name", "w"): Opens a file in write mode (creates a new file or overwrites if it exists).
    - fopen("file_name", "r"): Opens a file in read mode.
    - fprintf(file, "format", data): Writes formatted data to a file.
    - fscanf(file, "format", &data): Reads formatted data from a file.
    - fclose(file): Closes the file after operations are completed.

SECTION-B (PYTHON-Programming)

Question 6: Write Python code to perform the following:

(i) Copy content of file first.txt to second.txt

(ii) Reading a file

(iii) Writing into a file

(iv) Appending into a file

```
# (i) Copy content of 'first.txt' to 'second.txt'
def copy_file():
    try:
        with open("first.txt", "r") as first, open("second.txt", "w") as second:
            content = first.read()
            second.write(content)
        print("Content copied successfully from first.txt to second.txt.")
    except FileNotFoundError:
        print("Error: first.txt not found.")

# (ii) Reading a file
def read_file(filename):
    try:
        with open(filename, "r") as file:
            content = file.read()
            print(f"\nContents of {filename}:\n{content}")
    except FileNotFoundError:
        print(f"Error: {filename} not found.")

# (iii) Writing into a file
def write_to_file():
    content = "This is new content written into the file.\nWelcome to file handling in
Python!"
    with open("write_file.txt", "w") as file:
        file.write(content)
    print("\nContent written to write_file.txt successfully.")

# (iv) Appending into a file
def append_to_file():
    content = "\nThis is an appended line."
    with open("append_file.txt", "a") as file:
        file.write(content)
    print("\nContent appended to append_file.txt successfully.")

# Execute functions
copy_file()          # Copying content
read_file("first.txt") # Reading a file
write_to_file()      # Writing into a file
append_to_file()     # Appending into a file
read_file("append_file.txt") # Reading after appending
```

Explanation of the Code

1. **Copying Content (first.txt → second.txt)**
    - Reads the entire content of first.txt.
    - Writes the content into second.txt.
    - If first.txt does not exist, an error message is displayed.
2. **Reading a File**
    - Opens a specified file in **read mode** ("r").
    - Reads and prints its content.
    - Handles FileNotFoundError if the file does not exist.
3. **Writing into a File**
    - Opens write_file.txt in **write mode** ("w").
    - Overwrites the file with new content.
    - If the file does not exist, it is created.
4. **Appending into a File**
    - Opens append_file.txt in **append mode** ("a").
    - Adds new content without erasing existing content.
    - If the file does not exist, it is created.

Question 7: Write an algorithm to find the slope of a line segment whose endpoint coordinates are (x1, y1) and (x2, y2). The algorithm gives output whether the slope is positive, negative or zero. Transform your algorithm into Python program.

Note: Slope of line segment = (y2 – y1)/(x2–x1).

Python Program to Calculate the Slope

```
def find_slope(x1, y1, x2, y2):
    # Check for vertical line (undefined slope)
    if x2 - x1 == 0:
        print("Slope is undefined (vertical line).")
        return

    # Calculate slope
```

```
    slope = (y2 - y1) / (x2 - x1)

    # Determine slope nature
    if slope > 0:
        print(f"Slope = {slope:.2f} (Positive)")
    elif slope < 0:
        print(f"Slope = {slope:.2f} (Negative)")
    else:
        print("Slope = 0 (Horizontal line)")

# Input coordinates
x1 = float(input("Enter x1: "))
y1 = float(input("Enter y1: "))
x2 = float(input("Enter x2: "))
y2 = float(input("Enter y2: "))

# Call function to determine slope
find_slope(x1, y1, x2, y2)
```

Explanation of the Python Program

1. The function find_slope(x1, y1, x2, y2) calculates the slope using the formula

$$\text{slope} = \frac{(y2 - y1)}{(x2 - x1)}$$

1. It first checks if x2−x1=0x2 - x1 = 0x2−x1=0, which means the line is **vertical**, and prints "Slope is undefined".
2. If the slope is **greater than 0**, it prints "Slope is positive".
3. If the slope is **less than 0**, it prints "Slope is negative".
4. If the slope equals **0**, it prints "Slope is zero (horizontal line)".
5. The program takes user input for coordinates and passes them to find_slope().

Question 8: Write a program in Python to create a package named Volume and create 3 module in it named – Cube, Cuboid and Sphere each having a function to calculate Volume of Cube, Cuboid and Sphere respectively. Import the module in separate location and use the functions. Assumptions can be made wherever necessary. Support your program with suitable comments to improve readability.

Step 1: Create the Package Volume

In your working directory, create a folder named **Volume**. Inside this folder, create a special __init__.py file (leave it empty or use it to initialize the package).

```
Volume/
|— __init__.py
|— Cube.py
|— Cuboid.py
|— Sphere.py
```

Step 2: Create the Modules

1. Cube.py (Module for Cube Volume)

```
# Cube.py - Module to calculate the volume of a cube

def volume_cube(side):
    """
    Function to calculate the volume of a cube.
    Formula: V = side³
    """
    return side ** 3
```

2. Cuboid.py (Module for Cuboid Volume)

```
# Cuboid.py - Module to calculate the volume of a cuboid

def volume_cuboid(length, width, height):
    """
    Function to calculate the volume of a cuboid.
    Formula: V = length × width × height
    """
    return length * width * height
```

3. Sphere.py (Module for Sphere Volume)

```
# Sphere.py - Module to calculate the volume of a sphere

import math

def volume_sphere(radius):
    """
    Function to calculate the volume of a sphere.
    Formula: V = (4/3) * π * radius³
    """
    return (4/3) * math.pi * radius ** 3
```

Step 3: Import and Use the Modules from a Separate Script

Now, in a separate Python script (outside the Volume folder), we will import the package and use the functions.

Main Program (main.py)

```
# Importing the modules from the Volume package
from Volume.Cube import volume_cube
from Volume.Cuboid import volume_cuboid
from Volume.Sphere import volume_sphere

# Taking user input
side = float(input("Enter the side length of the cube: "))
length = float(input("Enter the length of the cuboid: "))
width = float(input("Enter the width of the cuboid: "))
height = float(input("Enter the height of the cuboid: "))
radius = float(input("Enter the radius of the sphere: "))

# Calculating volumes
cube_vol = volume_cube(side)
cuboid_vol = volume_cuboid(length, width, height)
sphere_vol = volume_sphere(radius)

# Displaying the results
print(f"\nVolume of Cube: {cube_vol:.2f} cubic units")
print(f"Volume of Cuboid: {cuboid_vol:.2f} cubic units")
print(f"Volume of Sphere: {sphere_vol:.2f} cubic units")
```

Step 4: Run the Program

Input:

```
Enter the side length of the cube: 3
Enter the length of the cuboid: 4
Enter the width of the cuboid: 5
Enter the height of the cuboid: 6
Enter the radius of the sphere: 2
```

Output:

```
Volume of Cube: 27.00 cubic units
Volume of Cuboid: 120.00 cubic units
Volume of Sphere: 33.51 cubic units
```

Explanation of the Program

1. Package Creation:
   - `Volume` is a package containing three modules: `Cube.py`, `Cuboid.py`, and `Sphere.py`.
   - Each module contains a function to calculate the volume of the respective 3D shape.
2. Function Definitions:
   - `volume_cube(side)`: Computes volume using $V = \text{side}^3$.
   - `volume_cuboid(length, width, height)`: Computes volume using $V = \text{length} \times \text{width} \times \text{height}$.
   - `volume_sphere(radius)`: Computes volume using $V = \frac{4}{3}\pi r^3$.
3. Importing the Package and Using the Functions:
   - The `main.py` script imports functions from the `Volume` package and calls them based on user input.

Question 9: Write a program in Python to perform following: (8 Marks)

- To find square root of numbers in a list using lambda function.
- To display first n lines from a file, where n is given by user.
- To display size of a file in bytes
- To display frequency of each word in a file.

Python Program for the Given Tasks

This program includes:

1. **Finding square roots of numbers in a list using a lambda function.**
2. **Displaying the first n lines from a file (sample.txt).**
3. **Displaying the size of a file in bytes.**
4. **Displaying the frequency of each word in a file (sample.txt).**

```
import os
import math
from collections import Counter

# (i) Find square root of numbers in a list using lambda function
def square_root_list(numbers):
    sqrt_list = list(map(lambda x: round(math.sqrt(x), 2), numbers))
    print(f"\nSquare roots of numbers: {sqrt_list}")
```

```
# (ii) Display first n lines from a file
def display_n_lines(filename, n):
    try:
        with open(filename, "r") as file:
            print(f"\nFirst {n} lines of {filename}:")
            for _ in range(n):
                line = file.readline()
                if not line:
                    break
                print(line.strip())
    except FileNotFoundError:
        print(f"Error: {filename} not found.")

# (iii) Display size of a file in bytes
def file_size(filename):
    try:
        size = os.path.getsize(filename)
        print(f"\nSize of {filename}: {size} bytes")
    except FileNotFoundError:
        print(f"Error: {filename} not found.")

# (iv) Display frequency of each word in a file
def word_frequency(filename):
    try:
        with open(filename, "r") as file:
            content = file.read().lower()  # Read file and convert to lowercase
            words = content.split()  # Split words
            word_count = Counter(words)  # Count word frequency
            print("\nWord Frequency in the file:")
            for word, count in word_count.items():
                print(f"{word}: {count}")
    except FileNotFoundError:
        print(f"Error: {filename} not found.")

# --- Main Execution ---

# (i) Find square roots
numbers = [4, 9, 16, 25, 36]
square_root_list(numbers)

# (ii) Read first n lines of a file
filename = "sample.txt"
```

```
n = int(input("\nEnter number of lines to display: "))
display_n_lines(filename, n)

# (iii) Display file size
file_size(filename)

# (iv) Display word frequency
word_frequency(filename)
```

Explanation of the Program

1. **Finding Square Roots Using Lambda Function**
   - Uses map() and lambda to compute square roots of list elements.
   - math.sqrt(x) is applied to each element, and results are rounded to 2 decimal places.
2. **Displaying First n Lines of a File**
   - Opens sample.txt in read mode.
   - Reads and prints the first n lines based on user input.
3. **Displaying File Size in Bytes**
   - Uses os.path.getsize(filename) to get the file's size.
   - Handles errors if the file is not found.
4. **Calculating Word Frequency in a File**
   - Reads the entire file, converts text to lowercase, and splits it into words.
   - Uses Counter from collections to count occurrences of each word.
   - Displays the word count.

Run

Input:

```
Numbers: [4, 9, 16, 25, 36]
Enter number of lines to display: 3
```

Output:

```
Square roots of numbers: [2.0, 3.0, 4.0, 5.0, 6.0]
```

```
First 3 lines of sample.txt:
This is the first line.
This is the second line.
This is the third line.

Size of sample.txt: 145 bytes

Word Frequency in the file:
this: 3
is: 3
the: 3
first: 1
line.: 3
second: 1
third: 1
```

Question 10: What are Co-routines? How Co-routines differ from threads? How Co-routines support cooperative multi-tasking in python? Compare Subroutines and Co-routines.

Co-routines in Python

Co-routines are special functions in Python that allow **pause and resume execution** at specific points. They enable cooperative multitasking, where tasks voluntarily yield control rather than being preempted by the system.

Unlike **regular functions (subroutines)**, which execute from start to finish, co-routines can **suspend** execution at a yield or await statement and **resume** later from the same point.

Difference Between Co-routines and Threads

Feature	**Co-routines**	**Threads**
Execution	Cooperatively multitasks	Runs in parallel (preemptive multitasking)
Switching	Explicitly controlled using yield or await	Managed by OS (thread scheduler)
Memory Usage	Lightweight (no separate stack per co-routine)	Heavy (each thread has its own stack)
Concurrency	Single-threaded, non-blocking	Multi-threaded, blocking

Type		
Context Switching	Manual, less overhead	Automatic, more overhead
Use Case	Asynchronous tasks like I/O operations, event loops	True parallelism with CPU-intensive tasks

Co-routine in Python

```
def coroutine_example():
    print("Starting Co-routine...")
    value = yield "Paused at Step 1"
    print(f"Resumed with value: {value}")
    yield "Paused at Step 2"

# Initialize and use the co-routine
co = coroutine_example()
print(next(co))  # Starts and pauses at first yield
print(co.send(100))  # Resumes with value 100
```

Output:

```
Starting Co-routine...
Paused at Step 1
Resumed with value: 100
Paused at Step 2
```

How Co-routines Support Cooperative Multi-tasking in Python

Python's co-routines work well with **asyncio** to enable non-blocking I/O operations. They **allow multiple tasks to run seemingly in parallel** (within a single thread) by voluntarily yielding control.

Example: Async I/O Using Co-routines

```
import asyncio

async def task1():
    print("Task 1: Start")
    await asyncio.sleep(2)  # Simulates I/O delay
    print("Task 1: Done")

async def task2():
    print("Task 2: Start")
    await asyncio.sleep(1)  # Simulates I/O delay
```

```
    print("Task 2: Done")

async def main():
    await asyncio.gather(task1(), task2())  # Run both tasks concurrently

asyncio.run(main())
```

Output:

```
Task 1: Start
Task 2: Start
Task 2: Done
Task 1: Done
```

Comparison: Subroutines vs. Co-routines

Feature	Subroutine (Function)	Co-routine
Execution Flow	Starts and runs to completion	Can pause and resume
Control Transfer	One-way (caller → function)	Two-way (yield and send())
Reusability	Called multiple times from the beginning	Can resume from last pause point
Concurrency	No concurrency, sequential execution	Supports cooperative multitasking
Example	def func(): return 10	def coro(): x = yield 10