Procedural Programming:

Definition:
Procedural programming is a programming paradigm that follows a step-by-step approach where a program is divided into procedures or routines. These procedures contain a series of computational steps that are executed in order.

Example (in C):
```c
#include <stdio.h>

// Function to calculate the sum of two numbers
int add(int a, int b) {
    return a + b;
}

// Function to calculate the difference of two numbers
int subtract(int a, int b) {
    return a - b;
}

// Main program
int main() {
    int result_sum = add(5, 3);
    int result_diff = subtract(5, 3);

    printf("Sum: %d\n", result_sum);
    printf("Difference: %d\n", result_diff);

    return 0;
}
```

In this example, `add` and `subtract` are procedures, and the program follows a procedural structure.

Object-Oriented Programming (OOP):

Definition:
Object-oriented programming is a programming paradigm that uses objects, which are instances of classes, to organize and structure code. Objects encapsulate data and behavior, and interactions occur through method calls.

Example (in Java):
```java
// Class representing a simple calculator
class Calculator {
    // Fields (data)
    private int result;

    // Constructor
    public Calculator() {
```

```
        this.result = 0;
    }

    // Method to add two numbers
    public void add(int a, int b) {
        result = a + b;
    }

    // Method to subtract two numbers
    public void subtract(int a, int b) {
        result = a - b;
    }

    // Method to get the result
    public int getResult() {
        return result;
    }
}

// Main program
public class Main {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        calculator.add(5, 3);
        System.out.println("Result: " + calculator.getResult());
    }
}
```

In this example, `Calculator` is a class, and `add`, `subtract`, and `getResult` are methods. The program follows an object-oriented structure.

Functional Programming:

Definition:
Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions. In functional programming, functions are first-class citizens, meaning they can be passed around as arguments and returned as values.

Example (in Haskell):
```haskell
-- Function to calculate the sum of two numbers
add :: Int -> Int -> Int
add a b = a + b

-- Function to calculate the difference of two numbers
subtract :: Int -> Int -> Int
subtract a b = a - b

-- Main program
main :: IO ()
main = do
    let resultSum = add 5 3
```

```
    resultDiff = subtract 5 3

  putStrLn $ "Sum: " ++ show resultSum
  putStrLn $ "Difference: " ++ show resultDiff
```

In this example, `add` and `subtract` are pure functions. The program follows a functional structure, and there is an absence of mutable state.

Each paradigm has its strengths and is suitable for different types of problems. Procedural programming is often straightforward for tasks with a clear sequence of steps. Object-oriented programming is effective for modeling real-world entities and managing complexity through encapsulation. Functional programming is valuable for tasks involving transformations and computations on data, emphasizing immutability and pure functions. Many modern programming languages support a combination of these paradigms.