Contents

1. Introduction

Programming paradigm is an approach to solve problem using programming language. Programming paradigm is making use of available tools and techniques of a programming language to solve a problem using some approach. It is basically a way in which programs in programming language can be organized. Programming paradigm provides set of principles, features, methods and practices for designing computer programs. There are many programming paradigms as of now, certain paradigms are best suited for certain types of projects it usually requires a lot for experience and research to determine best paradigm for the project usually programmer can work with different types of paradigms depending upon the project.

2. Different Types of Programming Paradigm

Some commonly used programming paradigm are:

1. Imperative Programming:

   Imperative programming focuses on providing set of instruction or statements that changes the program's state. In Imperative programming, set of instructions are given to a computer system to be executed in given order to solve a problem. Imperative programming tells a computer how to do it. Programmers are actually concerned with how get a problem solved step by step. [Example: C,C++,Java]

   Example:

```
#include <stdio.h>
int main() {
    int sum = 0;
    for (int i = 1; i <= 10; i++) {
        sum += i;
    }
    printf("The sum of the first 10 positive integers is: %d\n", sum);
    return 0;
}
```

The above C Program shows imperative style of programming because we are actually specifying how to perform the sum of first 10 positive integers. We have specified the steps, initialized the loop from 1 to 10 incrementing by 1 and update the sum with each iteration.

2. Declarative Programming:

   Declarative programming focus on logic and constraints of the problem without specifying the actual steps to achieve the solution. Imperative programming tells a computer what to do instead of how to do. Programmers just declare the desired outcome and the language handles the implementation. [Example: SQL, Prolog]

   Python can achieve a declarative approach by using built-in function and expression to get the desired outcome. A sample python to print the sum of first 10 positive integers is:

```
sum_of_integers = sum(range(1, 11))


print("The sum of the first 10 positive integers is:", sum_of_integers)
```

   Here, code archives the desired outcome using built-in functions and expression, without explicitly specifying the loop as in the imperative approach.

3. Procedural Programming:

   A procedural programming is an approach of dividing a problem into smaller, procedure or functions where each function performs a specific task. Function is executed one after another, where one functions calls another function to get the problem solved. In POP problem is broken down into smaller, manageable sub problems where each sub problems are solved using a function or a procedure.[Example: C]

   Example:

```
#include <stdio.h>
int add(int a, int b);
int subtract(int a, int b);
int multiply(int a, int b);
int main() {
    int num1 = 10;
    int num2 = 5;
    int sum = add(num1, num2);
    printf("Sum: %d\n", sum);
    int difference = subtract(num1, num2);
    printf("Difference: %d\n", difference);
    int product = multiply(num1, num2);
    printf("Product: %d\n", product);
    return 0;
}
int add(int a, int b) {
    return a + b;
}
int subtract(int a, int b) {
    return a - b;
}
int multiply(int a, int b) {
    return a * b;
}
```

To solve a calculation problem the program is divided into three function add(),subtract() and multiply() where each function performs its respective task and returns the result.

4. Functional Programming:

   In functional programming functions are treated as first class citizens, meaning they can be assigned to variables, passed as arguments to other functions, and returned as values from other functions. Another important concept of functional programming is pure functions, where the function always produces the same output for the same input and have no side effects. Data is immutable, meaning once it is created, it cannot be modified. Instead of modifying existing data, functional programming emphasizes creating new data structures through transformations. [Example Haskell, Lisp]

```
numbers = [1, 2, 3, 4, 5]

def is_even(x):
    return x % 2 == 0

even_numbers = list(filter(is_even, numbers))

print("Even numbers:", even_numbers)
```

   We directly apply functional programming concepts to manipulate the list ‘numbers’, demonstrating the power and simplicity of functional programming in Python. Here filter() has is_even() as an argument.

5. Object-Oriented Programming (OOP): Object-oriented programming paradigm is based on concept of real-world entities called objects, which contains data and methods. Programmer can code in a way, that mimics real-world entities. We can think Rectangle as object that length and breath for data and calculate_area() and calculate_perimeter() as methods. OOP makes use of Classes that are blueprint and objects are created as instance of classes. OOP is most popular programming paradigm. OOP is often associated with a bottom-up approach due to its focus on building complex systems from smaller, more manageable components called objects. [Example: C++, Python, Java]

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width

    def calculate_perimeter(self):
        return 2 * (self.length + self.width)

length = 5
width = 3

rectangle = Rectangle(length, width)

area = rectangle.calculate_area()
perimeter = rectangle.calculate_perimeter()

print("Area:", area)
print("Perimeter:", perimeter)
```

Here, ‘Rectangle’ is a class that has length and breath for storing data and calculate_area() and calculate_perimeter() as functions. ‘rectangle’ is an instance of the class Rectangle which is known as object.

3. Advantages, Disadvantages of Different Paradigms

Paradigm	**Pros and Cons**
Imperative	• Imperative programming lets you control how your program runs, making it easy to write algorithms. • It helps optimize performance, which is important for programs that need to run fast. • But it can get complicated and tangled up, causing bugs and making it hard to fix problems. • As your program gets bigger, it becomes harder to maintain and make it work smoothly because of how it handles data changing over time.
Declarative	• Declarative programming focuses on what you want to achieve rather than how to achieve it, making code easier to understand. • It promotes a more abstract and concise coding style, leading to simpler and more readable code. • However, it may lack the fine-grained control over performance optimization that imperative programming offers. • As a result, while easier to comprehend, declarative code might not always be as efficient or suitable for performance-critical applications.
Procedural Programming	• Procedural programming (POP) emphasizes breaking down tasks into procedures or functions, making it straightforward to follow program flow. • It is well-suited for smaller projects or tasks where the focus is on step-by-step execution. • However, as programs grow larger, managing the flow of data and control can become challenging. • Maintenance and scalability may suffer due to the lack of encapsulation and modularity compared to other paradigms.

Paradigm	Pros and Cons
Functional Programming	• Functional programming focuses on using pure functions and immutable data, leading to code that is easier to reason about and test. • It encourages a more declarative style, promoting higher-order functions and function composition. • However, it can be challenging for developers familier to imperative programming paradigms. • While functional programming offers benefits like improved concurrency and easier parallelism, it may not always be the best fit for all types of applications.
Object Oriented Programming	• Object-oriented programming (OOP) organizes code into objects, promoting encapsulation, inheritance, and polymorphism. • It facilitates code reuse, modularity, and easier maintenance through class hierarchies and relationships. • However, OOP can introduce complexity, especially for beginners, and may lead to tightly coupled code. • Additionally, designing effective class hierarchies and managing state can be challenging in larger projects.