Functional Programming Paradigm

Vishalxviii

Introduction

Functional programming is a programming paradigm in which we try to bind everything in pure mathematical functions style. It is a declarative type of programming style. Its main focus is on "what to solve" in contrast to an imperative style where the main focus is "how to solve". It uses expressions instead of statements. An expression is evaluated to produce a value whereas a statement is executed to assign variables. Those functions have some special features discussed below.

Functional Programming is based on Lambda Calculus:

Lambda calculus is a framework developed by Alonzo Church to study computations with functions. It can be called as the smallest programming language in the world. It gives the definition of what is computable. Anything that can be computed by lambda calculus is computable. It is equivalent to Turing machine in its ability to compute. It provides a theoretical framework for describing functions and their evaluation. It forms the basis of almost all current functional programming languages.

Fact: Alan Turing was a student of Alonzo Church who created Turing machine which laid the foundation of imperative programming style.

Programming Languages that support functional programming: Haskell, JavaScript, Python, Scala, Erlang, Lisp, ML, Clojure, OCaml, Common Lisp, Racket.

Concepts of functional programming:

- Pure functions
- Recursion
- **Referential transparency**
- **Functions are First-Class and can be Higher-Order**
- **Variables are Immutable**

Pure functions: These functions have two main properties. First, they always produce the same output for same arguments irrespective of anything else.

Secondly, they have no side-effects i.e. they do not modify any arguments or local/global variables or input/output streams.

Later property is called immutability. The pure function's only result is the value it returns. They are deterministic.

Programs done using functional programming are easy to debug because pure functions have no side effects or hidden

I/O. Pure functions also make it easier to write parallel/concurrent applications. When the code is written in this style, a smart compiler can do many things – it can parallelize the instructions, wait to evaluate results when needing them, and memorize the results since the results never change as long as the input doesn't change.
example of the pure function:

```
sum(x, y)           // sum is function taking x and y as arguments
    return x + y    // sum is returning sum of x and y without changing them
```

Recursion: There are no "for" or "while" loop in functional languages. Iteration in functional languages is implemented through recursion. Recursive functions repeatedly call themselves, until it reaches the base case.
example of the recursive function:

```
fib(n)
    if (n <= 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
```

Referential transparency: In functional programs variables once defined do not change their value throughout the program. Functional programs do not have assignment statements. If we have to store some value, we define new variables instead. This eliminates any chances of side effects because any variable can be replaced with its actual value at any point of execution. State of any variable is constant at any instant.

Example:

```
x = x + 1 // this changes the value assigned to the variable x.
          // So the expression is not referentially transparent.
```

Functions are First-Class and can be Higher-Order: First-class functions are treated as first-class variable. The first class variables can be passed to functions as parameter, can be returned from functions or stored in data structures. Higher order functions are the functions that take other functions as arguments and they can also return functions.

Example:

```
show_output(f)            // function show_output is declared taking argument f
                          // which are another function
    f();                  // calling passed function

print_gfg()             // declaring another function
    print("hello gfg");

show_output(print_gfg)  // passing function in another function
```

Variables are Immutable: In functional programming, we can't modify a variable after it's been initialized. We can create new variables – but we can't modify existing variables, and this really helps to maintain state throughout the runtime of a program. Once we create a variable and set its value, we can have full confidence knowing that the value of that variable will never change.

Advantages and Disadvantages of Functional programming

Advantages:

1. Pure functions are easier to understand because they don't change any states and depend only on the input given to them. Whatever output they produce is the return value they give. Their function signature gives all the information about them i.e. their return type and their arguments.

2. The ability of functional programming languages to treat functions as values and pass them to functions as parameters make the code more readable and easily understandable.
3. Testing and debugging is easier. Since pure functions take only arguments and produce output, they don't produce any changes don't take input or produce some hidden output. They use immutable values, so it becomes easier to check some problems in programs written uses pure functions.
4. It is used to implement concurrency/parallelism because pure functions don't change variables or any other data outside of it.
5. It adopts lazy evaluation which avoids repeated evaluation because the value is evaluated and stored only when it is needed.

Disadvantages:

1. Sometimes writing pure functions can reduce the readability of code.
2. Writing programs in recursive style instead of using loops can be bit intimidating.
3. Writing pure functions are easy but combining them with the rest of the application and I/O operations is a difficult task.
4. Immutable values and recursion can lead to decrease in performance.

Applications:

- It is used in mathematical computations.
- It is needed where concurrency or parallelism is required.

Fact: Whatsapp needs only 50 engineers for its 900M users *because Erlang is used to implement its concurrency needs. Facebook uses Haskell in its anti-spam system.*

Last Updated : 28 Jun, 2022

Similar Reads

1. Functional Hazard
2. A basic Python Programming Challenge
3. Draw a moving car using computer graphics programming in C
4. Draw a Chess Board using Graphics Programming in C
5. Object Oriented Programming (OOPs) in MATLAB
6. Introduction to CUDA Programming
7. What is Fourth Generation Programming Language?
8. Basics of Computer Programming For Beginners
9. Learning the art of Competitive Programming
10. GATE and Programming Multiple Choice Questions with Solutions

Related Tutorials

1. Computer Science and Programming For Kids