5.2 FUNDAMENTALS OF FUNCTIONAL PROGRAMMING LANGUAGES

Functional programming languages are specially designed to handle symbolic computation and list processing applications. Functional programming is based on mathematical functions. Some of the popular functional programming languages include: Lisp, Python, Erlang, Haskell, Clojure, etc.

Functional programming languages are categorized into two groups, i.e. −

Pure Functional Languages − These types of functional languages support only the functional paradigms. For example − Haskell.

Impure Functional Languages − These types of functional languages support the functional paradigms and imperative style programming. For example − LISP.

Functional Programming – Characteristics

The most prominent characteristics of functional programming are as follows −

- Functional programming languages are designed on the concept of mathematical functions that use conditional expressions and recursion to perform computation.
- Functional programming supports higher-order functions and lazy evaluation features.
- Functional programming languages don't support flow Controls like loop statements and conditional statements like If-Else and Switch Statements. They directly use the functions and functional calls.
- Like OOP, functional programming languages support popular concepts such as Abstraction, Encapsulation, Inheritance, and Polymorphism.

Functional Programming Languages:

- The design of the imperative languages is based directly on the von Neumann architecture

- Efficiency is the primary concern, rather than the suitability of the language for software development.

- The design of the functional languages is based on mathematical functions

- A solid theoretical basis that is also closer to the user, but relatively unconcerned with the

architecture of the machines on which programs will run

Mathematical Functions:

Def: A mathematical function is a mapping of members of one set, called the domain set, to another set, called the range set.

A lambda expression specifies the parameter(s) and the mapping of a function in the following form f(x) x * x * x for the function cube (x) = x * x * x functions.

- Lambda expressions are applied to parameter(s) by placing the parameter(s) after the expression

e. g. (f(x) x * x * x)(3) which evaluates to 27.

Functional Forms:

Def: A higher-order function, or functional form, is one that either takes functions as parameters or yields a function as its result, or both.

1.Function Composition:

A functional form that takes two functions as parameters and yields a function whose result is a

function whose value is the first actual parameter function applied to the result of the application of the second Form: h(f)° g which means h (x) f (g (x))

2. Construction:

A functional form that takes a list of functions as parameters and yields a list of the results of applying each of its parameter functions to a given parameter

Form: [f, g]

For f (x) = x * x * x and g (x) = x + 3,

[f, g] (4) yields (64, 7)

3. Apply-to-all:

A functional form that takes a single function as a parameter and yields a list of values obtained by applying the given function to each element of a list of parameters

Form:

For h (x) =x * x * x

f(h, (3, 2, 4)) yields (27, 8, 64)

LISP –

LISP is the first functional programming language, it contains two forms those are

1. Data object types: originally only atoms and lists

2. List form: parenthesized collections of sub lists and/or atoms

e.g., (A B (C D) E)

Fundamentals of Functional Programming Languages:

- The objective of the design of a FPL is to mimic mathematical functions to the greatest extent possible

- The basic process of computation is fundamentally different in a FPL than in an imperative language

- In an imperative language, operations are done and the results are stored in variables for later use

- Management of variables is a constant concern and source of complexity for imperative programming

- In an FPL, variables are not necessary, as is the case in mathematics

- In an FPL, the evaluation of a function always produces the same result given the same parameters

- This is called referential transparency

A Bit of LISP:

- Originally, LISP was a type less language. There were only two data types, atom and list

- LISP lists are stored internally as single-linked lists

- Lambda notation is used to specify functions and function definitions, function applications, and data all have the same form E .g :,

If the list (A B C) is interpreted as data it is a simple list of three atoms, A, B, and C If it is interpreted as a function application, it means that the function named A is applied to the two parameters, B and C

- The first LISP interpreter appeared only as a demonstration of the universality of the computational capabilities of the notation