**Types of programming languages**

1. **Procedural programming languages:**

A procedural language follows a sequence of statements or commands in order to achieve a desired output. Each series of steps is called a procedure, and a program written in one of these languages will have one or more procedures within it. Example - C and C++, Java, Pascal, BASIC.

1. **Functional programming languages:**

Rather than focusing on the execution of statements, functional languages focus on the output of mathematical functions and evaluations. Each function–a reusable module of code–performs a specific task and returns a result. The result will vary depending on what data you input into the function. Example – Scala, Erlang, Haskell, Elixir, F#

1. **Object-oriented programming languages:**

This type of language treats a program as a group of objects composed of data and program elements, known as attributes and methods. Objects can be reused within a program or in other programs. This makes it a popular language type for complex programs, as code is easier to reuse and scale. Example – Java, Python, PHP, C++, Ruby.

1. **Scripting languages:**

Programmers use scripting languages to automate repetitive tasks, manage dynamic web content, or support processes in larger applications. Example- PHP, Ruby, Python, bash

1. **Logic programming languages:**

Instead of telling a computer what to do, a logic programming language expresses a series of facts and rules to instruct the computer on how to make decisions. Example- Prolog, Absys, Datalog

## Functional Programming

**What is Functional Programming?**

Functional programming (also called FP) is a way of thinking about software construction by creating pure functions. It avoids concepts of shared state, mutable data observed in Object Oriented Programming.

Functional languages emphasis on expressions and declarations rather than execution of statements. Therefore, unlike other procedures which depend on a local or global state, value output in FP depends only on the arguments passed to the function.

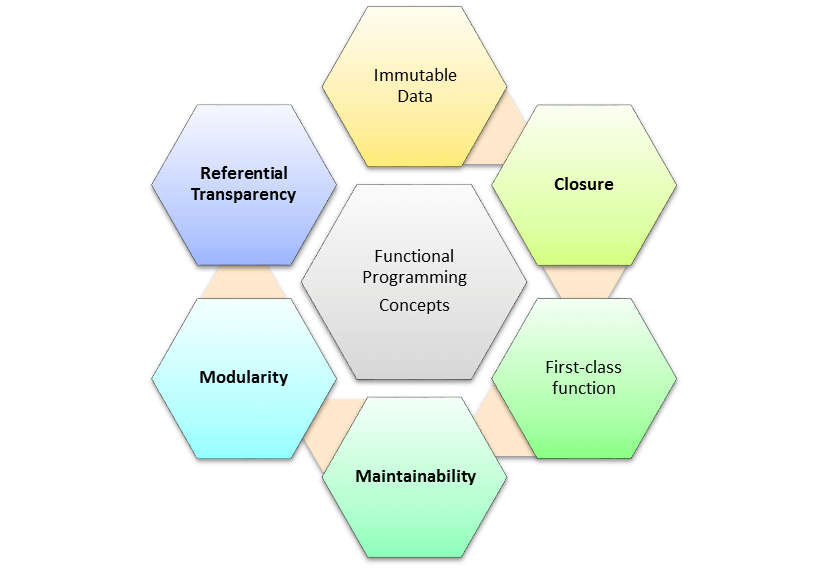
**Characteristics of Functional Programming**

1. Functional programming method focuses on results, not the process
2. Emphasis is on what is to be computed
3. Data is immutable
4. Functional programming Decompose the problem into ‘functions
5. It is built on the concept of mathematical functions which uses conditional expressions and recursion to do perform the calculation
6. It does not support iteration like loop statements and conditional statements like If-Else

**History of Functional programming**

1. The foundation for Functional Programming is Lambda Calculus. It was developed in the 1930s for the functional application, definition, and recursion
2. LISP was the first functional programming language. McCarthy designed it in 1960
3. In the late 70’s researchers at the University of Edinburgh defined the ML(Meta Language)
4. In the early 80’s Hope language adds algebraic data types for recursion and equational reasoning
5. In the year 2004 Innovation of Functional language ‘Scala.’

## Basic Functional Programming Terminology and Concepts



1. First-Class Functions:

Another key benefit of first-class functions is that there are no limits and restrictions on how to use functions. In other words, they behave like any other variable.

First-class functions lay the groundwork for other modifications, like currying, higher-order functions, and closures.

1. Higher-Order Functions:

Higher-order functions take functions as an argument or return the functions. Aside from modularizing programs, you can also use higher-order functions to make functions polymorphic (i.e., allows for the use of a single code multiple times).

1. Immutability

Immutable objects do not change. In FP, you initialize and implement objects and values without changing their values or state. You can create new objects and values if necessary, but you can’t modify an existing object or value’s state.

Functional programming’s immutability aligns with a key principle of mathematics—that objects can’t change their state. This principle is evident in even the most basic mathematical formulas.

Immutability prevents problems from arising and spreading in your code. In a multithreaded application, a thread can act on an immutable object without altering the other threads since no one is modifying the object. In concurrent applications, this prevents errors in your code.

1. Recursion

Recursion takes place when a function calls itself. A recursive function executes code and repeatedly runs the function until it meets an exit condition. You can find this pattern in a standard loop: the loop declares an initial variable and executes the code to be done with the variable. This continues until it meets a stopping or exit condition.

You can use a recursion instead of a loop. However, a loop can never substitute for a recursion.

Here are some key points to using recursive functions in your JavaScript code:

Recursion often gets the job done properly when sorting tree structures (i.e., node relationships).

If you find yourself repeatedly nesting loops within loops, a recursive function can help you extract data and reuse the function for different trees. For a more in-depth explanation, check out this tutorial on using recursive functions in JavaScript.

1. Function Composition

Function composition allows you to combine pure functions to create more complicated ones. The same principle applies in mathematics: the result of one function continues as the argument of the following function, and the result of the last function is the result of the whole.

In coding, we can combine multiple steps into a single code line or into a new function.

1. Referential Transparency

Referential transparency allows a value to replace its expression in a program (or anything with the same value). This happens without changing the result of the program. This logic dictates that methods should always return the same values for given arguments without additional effects.

Referential transparency makes reasoning about programs a more straightforward process. It renders each subprogram independent, which dramatically simplifies refactoring and unit testing.

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.

Pure function

A Pure Function is a function (a block of code) that always returns the same result if the same arguments are passed. It does not depend on any state or data change during a program’s execution. Rather, it only depends on its input arguments. Pure function does not produce any observable side effects such as network requests or data mutation etc.

**Object-Oriented Programming vs Functional Programming**

1. OOP uses the imperative programming model, meaning functions are invariably coded in every step needed to solve a problem. You code each operation with the code itself specifying how to solve the problem. This model requires the programmer to know which functions are necessary to solve a problem instead of relying on models that can solve the problems.
2. FP uses the declarative programming model, meaning it relies on the underlying concepts of a programming language to execute the necessary steps to reach the predetermined outcome.
3. Imperative programs focus on the step-by-step process of solving a problem, whereas declarative programs focus on the result of solving a problem.
4. Another critical difference is mutability: OOP uses mutable data while FP uses immutable data. You can alter (or mutate) mutable objects after creation, whereas you can’t for immutable objects. In FP, you’ll need to make a copy of the object and use that copy to write the rest of your code.
5. Overall, immutable code is easier to update, more efficient to manage, and easier to test and debug. And because variables are constant, the resulting code is easier to understand and reason about. Many programmers and software developers prefer to work with FP models.
6. Ultimately, the right programming paradigm for you will depend on your intended application. OOP works best for standardized and straightforward projects, whereas FP works best for projects that require scalability and flexibility.

## Currying

Currying is the transformation of a function with multiple arguments into a sequence of single-argument functions. That means converting a function like this f(a, b, c, ...) into a function like this f(a)(b)(c)... . Currying is helpful when you have to frequently call a function with a fixed argument.

Normal Function :

function isGreaterThan(a, b) {

return b > a;

}

isGreaterThan(2, 5)

Currying function :

function isGreaterThan(a) {

return function(b) {

return b > a;

}

}

isGreaterThan(2)(5);

const isGreaterThan = a => b => b > a;