**FUNCTIONAL PROGRAMMING**

Functional programming is something that existed for much longer than object oriented, dating back to the ancient days of Turing machines. It has seen ebbs and flows in popularity over time, based on whatever programming language and its foundational paradigms have taken over as part of the developer’s toolkit. For a long time, object oriented thinking has dominated the community. It’s one of the first things taught in Computer Science degrees and most talked about when it comes to learning programming.

It cannot depend on any hidden state or global variables. The evaluation of the function should not cause any side effects, such as mutation of mutable objects or output to I/O devices or modify any fields outside of the function scope. And it does not modify its input parameters.

## **Functional programming vs Object-oriented programming**

It’s important to compare these programming ways to identify the differences and identify correct usage of functional programming. Primary concern of functional programming is not ‘ how to do it ’ but rather ‘ what to do ’. Major difference could be identified as using the declarative programming model in functional programming rather than it’s competitor’s imperative programming model.

**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 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 the 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.

# **Pure Functions**

A function is called a pure [function](http://en.wikipedia.org/wiki/Pure_function) if it always returns the same result for same argument values and it has no side effects like modifying an argument (or global variable) or outputting something. The only result of calling a pure function is the return value. Examples of pure functions are strlen(), pow(), sqrt() etc. Examples of impure functions are printf(), rand(), time(), etc.

***Example :-***

\_\_attribute\_\_ ((pure)) return-type fun-name(arguments1, …)

{

/\* function body \*/

}