Putting the Fun Back in Functional Programming

We show by constructing a counterexample that a functional programming language does not have to be complex and cumbersome.

Functional programming (FP) means writing programs that use only applications of pure, side effect-free functions. Programs written this way are essentially mathematics, can be understood as such and manipulated as such. Also, the absence of side effects opens up new implementation possibilities including distributed and parallel systems.

Unfortunately to many people FP means programming in Haskell. Haskell is a powerful functional language with an efficient implementation. But it carries serious baggage, like reliance on monads for even simple input/output. Monads are a fairly sophisticated categorical construct; programmers should not have to know category theory to print out e.g. simple sums.

Fortunately the problems with Haskell are not inherent in FP. To show this we construct (and implement) a simple functional language PyFL which, we claim, is without Haskell's serious problems. I/O in PyFL (Python-based Functional Language) is simple, monad-free and, in many cases, arguably side-effect free. PyFL dispenses with Haskell's elaborate type declarations though not, paradoxically, with type checking. And PyFL has a number of extensions not found in Haskell, including (again, paradoxically) a simple form of while loop.

We sketch the design/implementation of PyFL and give a quick tour of a complete PyFL program to play unbeatable Tic-Tac-Toe (Noughts and Crosses).