

Pointfrip Documentation

2022-06-10

"FP forces the programmer, so the constructed functions are always pure."

Design Goals

Infix-combinator mechanism
with object orientation using arrays and index types (object classes)
Vocabulary is built on a system of emergence + specialization
FP as a nestable language - strict form of structured programming
no gotos only calls, tail recursion support
Function level ,Type level
Aligned with APL's right-before-left rule
Separation into pure functions and monad effects
Pattern Matching for Dict
dynamic scope because only instance variables
(out of the box)

Precompiler + FP Interpreter

Type-oriented automaton (higher-order)
Infix combiner mechanism (infix meta combination)

Extensibility of the Language

Extensibility based on the ideas of Guy Steel's "Growing a Language".
the FP project is equipped with an object mechanism, so that one
can define and inherit objects and classes and
can address them in four different ways.

So you

- complex classes,
- fraction classes,
- interval classes,
- matrix classes

etc. can add.

There is also the possibility of meta-programming for the FP language in order to extend the scope of the language (unlike Java) in the direction that you can add your own loops and control structures.

Last changes:

- hex function implemented,
- pointersize function implemented,
- cosmetic work on the source code,
- reference.rtf/.tmdx extended
- Documents translated into English

// (CC0)