1 Introduction

my childhood was great. things really started to slow down in my adolescence, and now I just feel... slow.

© Original Artist

"Look son there's functional architecture and there's functional architecture!"
Introduction

- What is FP about? What is it about FP?
- Key aspects of FP
What is FP about?

Problem/question:
Given some input of type A, produce output of type B

Solution/answer:
Define function $f : A \rightarrow B$
What is FP about?

Solution/answer: Define function \( f : A \rightarrow B \)

How?
(1) Use predefined functions
(2) Divide & conquer

"Really? — My people always say multiply and conquer."
What is it about FP?

Driving Forces

Safety first

Generality

Elegance

Referential Transparency

Strong Typing

Higher-order functions

Polymorphism, Type classes, ...

Lazy Evaluation

Monads
FP = Antithesis to EUP

End user:
Wants to get the job done,
Couldn’t care less about the program itself

Functional programmer:
Wants perfect program,
Couldn’t care less about getting job done

‘Compulsive refactoring disorder’ (CRD)
1 Introduction

• What is FP about? What is it about FP?
• Key aspects of FP
‘How to’ FP

- Identify/define types
- Define functions
- Refactor
Defining Functions

Recursion

```
sum :: [Int] → Int
sum xs = if null xs then 0
         else head xs + sum (tail xs)
```

(1) Case analysis

Pattern Matching

```
sum :: [Int] → Int
sum [] = 0
sum (x:xs) = x + sum xs
```

(2) Data decomposition

Higher-Order Functions

```
sum :: [Int] → Int
sum = foldr (+) 0
```

variables & recursion not needed!
Equations: Explaining Computation

\[
\text{sum} :: [\text{Int}] \to \text{Int} \\
\text{sum} \; [\;] \; = \; 0 \\
\text{sum} \; (x:x:s) \; = \; x + \text{sum} \; x:s
\]

Only pattern matching & substitution
No additional structures (state) required
Equations: Support Refactoring

The only valid measurement of code quality: WTFs/minute

(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics
Equations: Support Refactoring

Introduction

sum :: [Int] → Int
sum [] = 0
sum (x:xs) = x + sum xs

foldr f u [] = u
foldr f u (x:xs) = x \( f \) foldr f u xs

sum = foldr (+) 0

Refactoring definitions

Only pattern matching & substitution
Higher-Order Functions

Functional Programmers do it at a higher order!
Higher-Order Functions

State: first-order glue

Functional Programming

Imperative & OO Programming

Functions are values
Higher-Order Functions

Higher-order function \( \equiv \) control structure

Ability to define higher-order functions: Define your own control structures

Expressiveness for Defining Domain-Specific Languages