Functional Programming

Haskell

Landmarks in FP

- λ-Calculus -- Alonzo Church, 1936-1941
- LISP -- John McCarthy, 1958
  - S-Expressions: (+ 3 9 17 5)
- ML -- Robin Milner, 1973
- Miranda -- David Turner, 1985
- Haskell -- Simon Peyton Jones and many others, 1990

Other Popular Functional Languages

- Scala -- Java related OOP/FP hybrid
- F# -- Microsoft's .NET OOP/FP hybrid
- Erlang -- Ericsson's fault tolerant, parallel, pure FP
- Clojure -- Java based Lisp dialect
- Scheme/Common Lisp -- Modern Lisp
- OCaml -- Multi-paradigm

Functional

- Simple model of programming: "one value, the result, is computed on the basis of others, the inputs."
- Everything is a function: inputs → output
- Pure Functions
  - Same input, same output
  - No side effects
  - Referentially transparent
- Functions with superpowers

For Pure Functional: NO

statements

assignment statements

variables as we're used to

side-effects

flow of control (for, while, do loops ...)

complex scope rules

direct connection with the CPU and architecture

objects

const

static

+=

++

Why?

Power and elegance

Better modularity and abstraction

Shorter and easier to understand

Eliminates sources of errors

  "Easier" to prove correctness

Order of execution is irrelevant

  Better for concurrent execution: parallel cpu's

Generally very type-safe

Factorial

The factorial function is formally defined by the product

$$n! = \prod_{k=1}^{n} k$$

or recursively defined by

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0. \end{cases}$$

Functional

```
factorial' :: Integer -> Integer
factorial' n = product [1..n]
```

or

```
factorial :: Integer -> Integer
factorial n
    | n == 0 = 1
    | n > 0  = n * factorial (n-1)
```

Imperative

```
public static long factorial( int n )
{
    long fact = 1L;
    for( int i = 1; i <= n; ++i )
    {
        fact *= i;
    }
    return fact;
}
```

Convert list of Integers to list of Strings, then collapse to a single String

Haskell

```
convert :: [Integer] -> [String]
convert xs = map show xs
```

```
(concat . convert) [0..10]
```

```
concat(convert(list))
```

Java

```
public static List<String> convert( List<Integer> list )
{
    List<String> newList = new LinkedList<String>();
    for( Integer i: list )
    {
        newList.add( i.toString() );
    }
    return newList;
}

public static String concat( List<String> list )
{
    StringBuilder sb = new StringBuilder();
    for( String s: list )
    {
        sb.append( s );
    }
    return sb.toString();
}
```

Haskell

```haskell
xs = [2,3,5,7,11]

total = sum (map (3*) xs)

main = print total
```

Java

```java
int[] xs = {2,3,5,7,11};

int sum = 0;
for ( int i = 0; i < xs.length; i++)
{
    sum = sum + 3 * xs[i];
}

System.out.println(total);
```

Suffers from “indexitis”

https://www.fpcomplete.com/school/starting-with-haskell/introduction-to-haskell/1-haskell-basics

It’s OK to laugh.

Why is it that this isn’t THE way to write code?

Why is it that this isn’t THE way to solve problems?