Functional Programming and Haskell

Functional Programming
What is Functional Programming?

Opinions differ, and it is difficult to give a precise definition. Generally speaking, however, functional programming can be viewed as a style of programming in which the basic method of computation is the application of functions to arguments. In turn, a functional programming language is one that supports and encourages the functional style.

Graham Hutton
What is Functional Programming?

Opinions differ, and it is difficult to give a precise definition. Generally speaking, however, functional programming can be viewed as a style of programming in which the basic method of computation is the application of functions to arguments. In turn, a functional programming language is one that supports and encourages the functional style.

Graham Hutton

What are your current impressions of functional programming?
What features (buzzwords) have you heard about Haskell?
Sum of squares of integers from 1 to $n$
Sum of squares of integers from 1 to $n$

C/C++/Java:

```c
int sumSqs (int n) {
    int res = 0;
    for (int i = 1; i <= n; i++) {
        res += i*i;
    }
    return res;
}
```
Sum of squares of integers from 1 to \( n \)

C/C++/Java:

```c
int sumSqs(int n) {
    int res = 0;
    for (int i = 1; i <= n; i++) {
        res += i*i;
    }
    return res;
}
```

The computation method is variable assignment and loops.
Sum of squares of integers from 1 to $n$

Haskell:

\[
\text{sumSqs } n = \begin{cases} 
0 & \text{if } n \leq 0 \\
(\text{sumSqs } (n - 1)) + (n \times n) & \text{else}
\end{cases}
\]
Sum of squares of integers from 1 to $n$

Haskell:

```
sumSqs n = if n <= 0
  then 0
  else (sumSqs (n - 1)) + (n * n)
```

The computation method is function application and recursion.
Sum of squares of integers from 1 to $n$

Haskell:

```
sumSqs n = if n <= 0
  then 0
  else (sumSqs (n - 1)) + (n * n)

sumSqs n = sum (map sqr [1 .. n])
where sqr i = i * i
```
Sum of squares of integers from 1 to $n$

Haskell:

\[
\text{sumSqs } n = \begin{cases} 
  0 & \text{if } n \leq 0 \\
  \text{sumSqs } (n - 1) + (n \times n) & \text{else}
\end{cases}
\]

\[
\text{sumSqs } n = \text{sum} \ (\text{map} \ \text{sqr} \ [1 .. n])
\]

where \(\text{sqr } i = i \times i\)

\[
\text{sumSqs } n = \text{sum} \ [i \times i \mid i \leftarrow [1 .. n]]
\]
Sum of squares of integers from 1 to $n$

Haskell:

```
sumSqs n = if n <= 0
    then 0
    else (sumSqs (n - 1)) + (n * n)

sumSqs n = sum (map sqr [1 .. n])
    where sqr i = i * i

sumSqs n = sum [i * i | i <- [1 .. n]]

sumSqs n = foldl (\ i s -> s + i * i) 0 [1 .. n]
```

The computation method is function application and recursion.
Historical Origins

Imperative and functional paradigms grew out of research undertaken by Alonzo Church, Haskell Cury, Alan Turing, Stephen Kleene, etc. ~1930s.

- Different formulations of the notion of an algorithm (or effective procedure) based on symbolic manipulation (Church), combinators (Curry), automata (Turing), and recursive function definitions (Kleene).

Church-Turing Thesis

- Every intuitively appealing model of computing is equally powerful
Historical Origins

Turing machine (Turing’s model)
▶ Automaton with an unbounded “tape”
▶ Computes in an imperative way
   ▶ Changing the values in cells on the tape

\( \lambda \)-calculus (Church’s model)
▶ Parameterized expressions
▶ Computes in a functional way
   ▶ Substitute actual arguments for formal parameters
History

- 1930s: Alonzo Church develops the \( \lambda \)-calculus
- 1950s: John McCarthy develops Lisp (first functional language), with some influences from the \( \lambda \)-calculus but retaining variable assignments.
- 1960s: Peter Landin develops ISWIM (first pure functional language), based strongly on the \( \lambda \)-calculus with no assignments.
- 1970s: John Backus develops FP (*Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs*), emphasising higher-order functions and reasoning about programs.
History

- 1970-80s: David Turner develops Miranda (©, ®, ™), emphasizing lazy evaluation.
- 1987: Committee forms to develop Haskell, a standard lazy functional language.
- 1990s: Phil Wadler and other develop type classes and monads, two of the main innovations of Haskell.
- 1999: Haskell 98 Report
- 2003: Haskell 98 Report (Revised)
- 2010: Haskell 2010 Report
- 2010–present: Haskell Platform: standard distribution, library support, new language features, development tools, use in industry, influence on other languages, etc.
Other Functional Languages

- Clean
- Elm
- Standard ML, O’Caml, F#, Reason
- Lisp, Scheme, Racket, Clojure
- Erlang, Elixir

- Scala (Object Oriented + FP)
- Curry, Mercury (Logic + FP)

- “Lambdas” in Java/C++
Distinguishing Features of Functional Languages

- 1st class and higher-order functions
- Algebraic datatypes and pattern matching
- Sophisticated strong and static type systems
  - Type inference
  - Polymorphism (a.k.a., generics)
- Garbage collection
- Precise language definitions