CS795 Functional Web Development - Project 2

Marie-Louise Steenkamp 20722796
Hendrik van Heerden 20916892
Kieren Sinclair 19989059
Christoff Van Zyl 20072015

Department of Computer Sciences
StellenboschUniversity

September 29, 2020

Overview of contents

- What is functional programming?
- Why we use functional programming?
- History of functional programming.
- Functional programming concepts used in project.
- Project requirements.
- Functional programming concepts used in the project

What is functional programming?

- Functional programming languages are designed on the concept of mathematical functions that use conditional expressions and recursion to perform computation.
- Functional programming supports higher-order functions and lazy evaluation features.

What is functional programming (cont)?

- Functional programming languages don't support flow Controls like loop statements and conditional statements like If-Else and Switch Statements. They directly use the functions and functional calls.
- Like OOP, functional programming languages support popular concepts such as Abstraction, Encapsulation, Inheritance, and Polymorphism.

Why we use functional programming.

- Bugs-Free Code - Functional programming does not support state, so there are no side-effect results and we can write error-free codes.
- Efficient Parallel Programming Functional programming languages have no mutable state, so there are no state-change issues.
- Efficiency - Functional programs consist of independent units that can run concurrently.

Why we use functional programming (cont).

- Supports Nested Functions - Functional programming supports Nested Functions.
- Lazy Evaluation - Functional programming supports Lazy Functional Constructs like Lazy Lists, Lazy Maps, etc.

History of functional programming.

- In 1930 Alonzo Church devised a mathematical model of functions called lambda calculus, which captures the essence of computation. It involves function abstraction (like defining functions in Haskell) and application (like calling functions in Haskell).
- Early development of programming languages in the 1950s involved one of the first high-level programming languages called LISP (which stands for List Processing). LISP adopted a functional style. It allowed user functions to be defined, and passed around as values.

History of functional programming (cont).

- During the 1980s, lots of researchers were inventing and extending various functional programming languages. Example languages include ML, Hope and Miranda.
- However research was fragmented across the various languages, and many of them did not have 'open-source' frameworks. So a group of academics formed a committee to design and implement a new language, which would be used as a vehicle for research as well as for teaching functional programming.

History of functional programming (cont).

- After several years of work and arguments, the committee published the first Haskell Language Report in 1990.
- This was a major milestone: at last there was a common functional language around which the research community could unite.
- Functional programming started receiving the recognition it deserved.

Project requirements.

- The assignment given was to significantly modify a full stack web application with a database.
- The web-framework(s) familiar to us should be replaced by a functional language and framework for web development.
- We should take note of various functional programming concepts used.

Functional programming concepts used in project.

The following concepts were used to modify the chosen web application web application:

- Type safety
- Recursion
- Function Composition
- Pure Functions
- High-order Functions
- Currying

Type safety

- What is type safety?
  - Type safety is the extent to which a programming language discourages or prevents type errors.
  - Haskell is a pretty type safe language, it essentially means that values at some type cannot wantonly transform into another type.
  - Type safety allows error-free, well-written programs to be completely safe and reliable.

Recursion

- What is recursion?
  - Recursion is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem.

Recursion cont.

- Example of recursion
  - If you look at the code below, the function maximum' calls itself recursively, this means that the problem of finding the maximum value is spilt up into smaller units and solved each unit at a time.

```
maximum' :: (Ord a) => [a] -> a
maximum' [] = error "maximum of empty list"
maximum' [x] = x
maximum' (x:xs)
    | x > maxTail = x
    | otherwise = maxTail
    where maxTail = maximum' xs
```

Function Composition

- What is function composition?
  - It's the application of a function to the output of another function.

Function Composition cont.

- Example of function composition
  - Function composition is the act of pipelining the result of one function, to the input of another, creating an entirely new function. The example below shows how the functions reverse and sort are used as input to the function desort.

```
-- result of sort is pipelined to reverse
desort = (reverse . sort)

-- the result is a descending sort
countdown = desort [2,8,7,10,1,9,5,3,4,6]
```

Pure Functions

- What is a pure function?
  - A pure function is a function that is deterministic and has no side effects. A function is deterministic if, given the same input, it returns the same output. A side effect is a modification of state outside the local environment of a function.

High-order Functions

- What is a high-order function?
  - High-order functions are just functions that either: take one or more functions as arguments, or returns a function as its output.
  - Functional programming creates the idea that functions are just like any other value.

High-order Functions cont.

- Example of a high-order function
  - The example below shows how even is a first-order function and map and filter are higher-order functions, which can take even as an input.

```
Prelude> even 1
False
Prelude> even 2
True
Prelude> map even [1,2,3,4,5]
[False,True,False,True,False]
Prelude> filter even [1,2,3,4,5]
[2,4]
```

Currying

- What is currying?
  - Currying is a process in functional programming in which we can transform a function with multiple arguments into a sequence of nesting functions. It returns a new function that expects the next argument inline.

Currying cont.

- Example of currying
  - If you look at the code below, the function mul3, whcih has multiple arguments is now turned into a sequence of nesting functions in mul'. Calling mul' will automatically cal mul3 and supply the first two inline arguments.

```
mul3 x y z = x * y * z

mul' = mul3 2 3
```

Thank you!

That concludes our presentation, thank you for listening!