Functional Programming

To iterate is human, to recurse divine. -Anonymous

Other Programming Paradigms and a Misperception

- Imperative Programming (IP for short): we tell the computer what to do and what steps to take. Iteration is common and the focus is on steps the computer must take and on assigning values to variables.

- OOP: a style of IP where each data object maintains internally its state (variables that have values assigned). “Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).” -Wikipedia

- For years I had thought that the declarative style of SQL was FP. I found I was wrong, and although SQL does have aspects of FP it is not FP.

What is *Functional Programming*?

- No single definition of FP, the concept is to apply values to functions rather than using iterations and assigning values to variables

- “Functional programming is a declarative style of developing computer programs using mathematical expressions that are evaluated, instead of using statements that can change the state. Functional programs utilize expressions that are only dependent upon the inputs provided ensuring that the outputs and effects of the function are not dependent upon global variables or local states. Functions in functional programming are treated like any other value, so they can be modified and used as input or output parameters. Functional programming languages are well-defined and well-typed lending to analysis that is comparatively simple compared to nonfunctional programming languages [2166].”

  Computer Security Art and Science, 2nd Edition by Matt Bishop

- The remainder of this presentation will expand and develop the idea
- What is a “function”? Something that takes input(s) and produces an output. A functions is self-contained, that is, it depends on nothing outside of itself.

History

- Originates in 1930's with math of Alonzo Church
    - https://en.wikipedia.org/wiki/Alonzo_Church
    - Lamda Calculus
    - Nobody really sure where name "Lamda" came from
    - Lamda is lower case Greek letter λ , upside down Y
    - For us is just a shorthand for a function
        - a lamda means a function, often not named
- FP has been around since the beginning and was implemented with Lisp the big 4 of the 1950s: Fortran, Algol, Cobol, Lisp. Followed by APL 1960s
- 1970s John Backus Turing Award lecture spurs further research
    - "Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer…" -John Backus 1977 Turing Award Lecture paper
- Today we have Haskell, Clojure and many other purely functional languages, Scala spans both IP and FP, all big languages support some features of FP.

Salient Features of FP

- **First-class and higher order functions**
  - **A higher order function is a function that either takes another function as an argument or returns a function.**
  - Functions can be parameters to other functions
  - Map, Reduce and other Higher Order Functions (ryanguill.com) (JavaScript)
- **Pure functions** (level of re-use in FP is at the function level, in OOP it's the class)
  - Functions have no side effects (and no I/O, I/O is a side effect)
  - Functions always do same thing for a given input
- **Recursion** (instead of loops)
  - Rather than iterations/steps and assignment of values to variables
- **Immutable values** (aka referential transparency) instead of variables
  - Set it and forget it - once a value is created and bound to a name it never changes
- **Lazy evaluation**
  - Evaluation when needed not up front
  - Enables defining infinite data structures for example set of natural numbers

- *OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts.*
  —Michael Feathers quoted in *Functional Thinking* by Neal Ford

Salient Features of FP Continued

- FP makes heavy use of limited set of common data structures and many operations

- Common simple data structures
  - (Linked) Lists
  - Arrays
  - Dictionary aka "hash map" associative arrays
  - Work equally as easy for array, list, dictionary

And the application of highly optimized operations (functions). These are some of the most common and heavily used:

- map (lower case "m")
  - Not to be confused with a "hash map" aka associative array for example a ColdFusion Structure (java.lang.HashMap) or C# Dictionary object which has default and various alternate hash map types to choose from
  - Means to apply a value and operations on each element
- reduce
  - To sum up or boil down a list of elements to a value
- filter
  - Select (projection) of subset of values which meet a selection predicate
- find
- sort
- All feature built-in iterators

Why FP?

- "Functional programming has its roots in academia, evolving from the lambda calculus, a formal system of computation based only on functions. Functional programming has historically been less popular than imperative programming…" https://en.wikipedia.org/wiki/Functional_programming

- Why?
  - Physics and Engineering
    - Moore's Law, power and heat dissipation and other engineering problems. It is getting harder and harder to clock CPUs faster and overcome physical limitations in each new generation of faster CPUs
  - Engineers struggle to make the CPU go faster, but they can give us more of them
    - Multi core processors
    - Cloud architecture
  - Author Neal Ford points out the evolution of programming languages and faster processor speeds enables a higher-level thought process and less work and mistakes for the programmer. The computer can tolerate some internal extra work. For example Java and C# have implicit garbage collection and C and C++ do not, but today they perform as well in most situations.
  - FP better supports concurrency – avoids use of mutable variables
  - FP supports reasoning about a program – that is the use of logic to verify correctness of a program

- Hence the rise of dedicated FP languages in popularity and the addition of FP features to widely used programming languages
- Less code, and once understood simple syntax, equals less mistakes and better re-use
- It's here to stay – the FP features added to CFML, C#, Java are part of the languages and not 3rd party libraries.
- It's not either / or – use best tool for the job

Example 1 Code in *functional manner*

- Show a loop/iteration over an objects values
- Contrast with object.sum() function
  - Note the simplicity - once the syntax is understood
  - Member functions instead of calling static methods
  - https://cfdocs.org/member

Example 2

- Show a sum of values using concurrent threads – if the problem is obvious bear with me here for a minute. Imagine there's a lot more code here. The larger problem is that the overwriting of a shared and mutable variable can be hard to spot.
- Can fix by separating the sums into temp vars, but still doesn't solve the problem of being able to spot unintended interactions.
- Show FP equivalent avoids use of local vars – sort of pushes the details down to underlying compiler and runtime system.

Example 3

- A simple operation on a list.
- Take a list of string values and output them in sorted order from shortest to longest.
- FP highly optimized operations on common data types – less code = less room for mistakes
  - Takes some time to get used to the syntax
  - Once understood you’ll look for opportunities and avoid writing out custom for loops

Example 4

- Real world example
- Over/Under loaded academic program for given AY-T
- Cadets now given flag by AARS – allowed to O/U
- Some cadets have flag but are not actually O/U
    - Mistake? In what way?
- Task is to create an exceptions report that shows cadets w/o flag and who are not already included in the report output.
- O/U proc is ancient and uses cursors and part of task here is not to create more work by modifying the proc (imagine the proc is used in other places and changing its outputs is not an option)
- Apply some FP know-how and create the solution

Further Reading

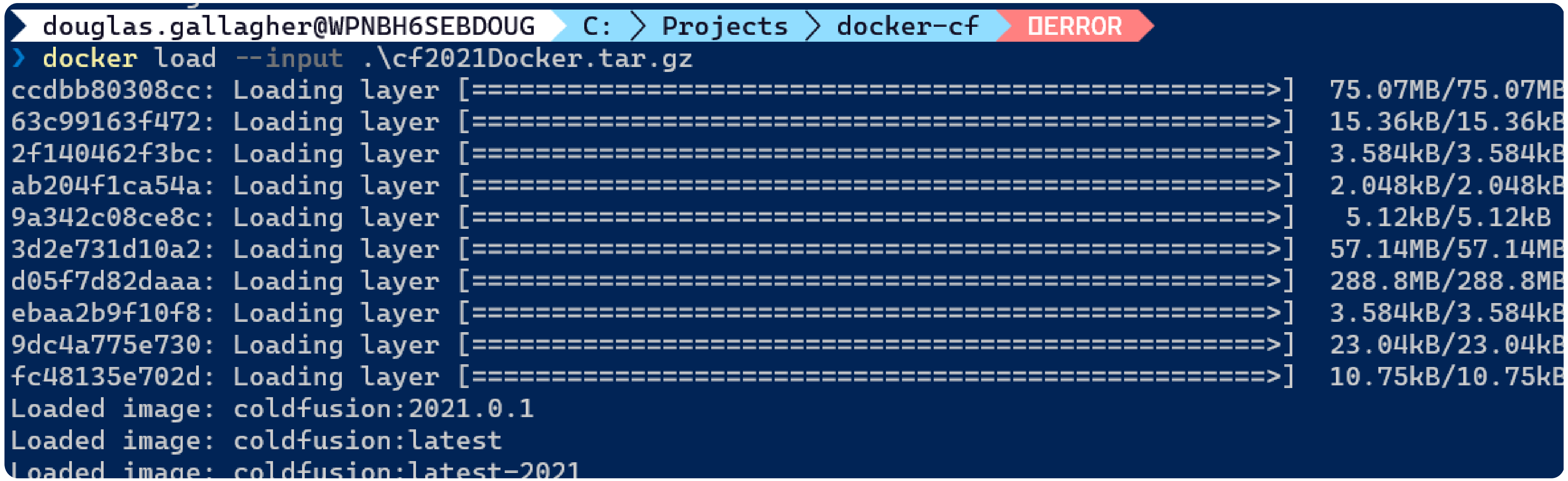
- Code samples and slides (PDF) are here: https://github.com/douggal/functional-prg-cf

- "The problem with a completely new programming paradigm isn't learning a new language. After all, everyone reading this has learned numerous computer languages—language syntax is merely details. The tricky part is learning to *think* in a different way."

  -Neal Ford *Functional Thinking* Ch 1.

- *Functional Thinking* O'Reilly video with presenter Neal Ford
- *Functional Thinking* by Neal Ford
- *Programming in Scala* by Odersky, Spoon, Venners
  - Scala is a JVM language, and it's used at West Point as a teaching language
  - Multi-paradigm - supports IP but built for, bridges to, and encourages FP
- *Thinking Recursively* by Eric Roberts
- 1977 Turing Award Lecture John Backus https://amturing.acm.org/award_winners/backus_0703524.cfm

- [Wikipedia](), [CFDocs.org](), many others
- Computerphile YouTube channel https://www.youtube.com/results?search_query=computerphile+functional+programming



```
 douglas.gallagher@WPNBH6SEBDOUG  C:  Projects  docker-cf  ERROR
❯ docker load --input .\cf2021Docker.tar.gz
ccdbb80308cc: Loading layer [==================================================>]  75.07MB/75.07MB
63c99163f472: Loading layer [==================================================>]  15.36kB/15.36kB
2f140462f3bc: Loading layer [==================================================>]  3.584kB/3.584kB
ab204f1ca54a: Loading layer [==================================================>]  2.048kB/2.048kB
9a342c08ce8c: Loading layer [==================================================>]   5.12kB/5.12kB
3d2e731d10a2: Loading layer [==================================================>]  57.14MB/57.14MB
d05f7d82daaa: Loading layer [==================================================>]  288.8MB/288.8MB
ebaa2b9f10f8: Loading layer [==================================================>]  3.584kB/3.584kB
9dc4a775e730: Loading layer [==================================================>]  23.04kB/23.04kB
fc48135e702d: Loading layer [==================================================>]  10.75kB/10.75kB
Loaded image: coldfusion:2021.0.1
Loaded image: coldfusion:latest
Loaded image: coldfusion:latest-2021
```

ColdFusion Docker Image

- Download link: https://www.adobe.com/support/coldfusion/downloads.html
- UG: https://helpx.adobe.com/coldfusion/user-guide.html/coldfusion/using/docker-images-coldfusion.ug.html
- `docker container run -dt -p 8500:8500 -v ${PWD}/:/app -e acceptEULA=YES -e password=ColdFusion123 -e importCFSettings=config.json -e installModules=all coldfusion:latest`