STANDARDIZED LADDER OF FUNCTIONAL PROGRAMMING

FUNCTIONAL PROGRAMMING FLAVOR: STATICALLY-TYPED, CATEGORY-THEORETIC — HASKELL, PURESCRIPT, SCALA, ETC

The LambdaConf Ladder of Functional Programming (LOFP) is a standardized grouping of concepts and skills relevant to functional programming in a statically-typed, category-theoretic programming environment. LOFP is used to categorize LambdaConf's workshops, talks, presentations, books, and courseware, so that aspiring functional programmers can better identify material that matches their backgrounds.

FIRE KERAMIK

CONCEPTS

- Immutable Data
- Second-Order Functions
- Constructing & Destructuring
- Function Composition
- First-Class Functions & Lambdas

SKILLS

- Use second-order functions (map, filter, fold) on immutable data structures
- Destructure values to access their components
- Use data types to represent optionality
- Read basic type signatures
- Pass lambdas to second-order functions

FIRE LUBLINE

CONCEPTS

- Algebraic Data Types
- Pattern Matching
- Parametric Polymorphism
- General Recursion
- Type Classes, Instances, & Laws
- Lower-Order Abstractions (Equal, Semigroup, Monoid, etc)
- Referential Transparency & Totality
- Higher-Order Functions
- Partial-Application, Currying, & Point-Free Style

SKILLS

- Solve problems without nulls, exceptions, or type casts
- Process & transform recursive data structures using recursion
- Able to use functional programming "in the small"
- Write basic monadic code for a concrete monad
- Create type class instances for custom data types
- Model a business domain with ADTs
- Write functions that take and return functions
- Reliably identify & isolate pure code from impure code
- Avoid introducing unnecessary lambdas & named parameters

ICE SKRIG

CONCEPTS

- Generalized Algebraic Data Types
- Higher-Kinded Types
- Rank-N Types
- Folds & Unfolds
- Higher-Order Abstractions (Category, Functor, Monad)
- Basic Optics
- Efficient Persistent Data Structures
- Existential Types
- Embedded DSLs using Combinators

SKILLS

- Able to use functional programming "in the large"
- Test code using generators and properties
- Write imperative code in a purely functional way through monads
- Use popular purely functional libraries to solve business problems
- Separate decision from effects
- Write a simple custom lawful monad
- Write production medium-sized projects
- Use lenses & prisms to manipulate data
- Simplify types by hiding irrelevant data with existentials

FIRE BRUSHE

CONCEPTS

- Codata
- (Co)Recursion Schemes
- Advanced Optics
- Dual Abstractions (Comonad)
- Monad Transformers
- Free Monads & Extensible Effects
- Functional Architecture
- Advanced Functors (Exponential, Profunctors, Contravariant)
- Embedded DSLs using GADTs, Finally Tagless
- Advanced Monads (Continuation, Logic)
- Type Families, Functional Dependencies

SKILLS

- Design a minimally-powerful monad transformer stack
- Write concurrent and streaming programs
- Use purely functional mocking in tests
- Use type classes to modularly model different effects
- Recognize type patterns & abstract over them
- Use functional libraries in novel ways
- Use optics to manipulate state
- Write custom lawful monad transformers
- Use free monads / extensible effects to separate concerns
- Encode invariants at the type level
- Effectively use FDs / type families to create safer code

ICE THERMAGON

CONCEPTS

- High-Performance
- Kind Polymorphism
- Generic Programming
- Type-Level Programming
- Dependent-Types, Singleton Types
- Category Theory
- Graph Reduction
- Higher-Order Abstract Syntax
- Compiler Design for Functional Languages
- Profunctor Optics

SKILLS

- Design a generic, lawful library with broad appeal
- Prove properties manually using equational reasoning
- Design & implement a new functional programming language
- Create novel abstractions with laws
- Write distributed systems with certain guarantees
- Use proof systems to formally prove properties of code
- Create libraries that do not permit invalid states
- Use dependent-typing to prove more properties at compile-time
- Understand deep relationships between different concepts
- Profile, debug, & optimize purely functional code with minimal sacrifices

FANTASYLAND