Functional Paradigm

Key Concepts

Expressions

Functions

Parametric Polymorphism

Data Abstraction

Lazy Evaluation

Expressions compute old values into new values

Functions are first class values,and abstract. Higher Order.

Parametric Polymorphism operations on type family rather than one type

Lazy evaluation relates to expression evaluation - eager normal lazy

Evaluation Order

Eager Evaluation - evaluate the parameters at point of call

Normal Order Evaluation - evaluate the parameter only when needed

Example

Sqr n = n\*n

Sqr m+1 is called and m is 6

Eager - evaluate m+1 to 7

Then n \* n which is 49

Normal Order - Bind value m+1 to n

M+1 \* m+1

Church - Rosser Property

If an expression can be evaluated at all, it can be evaluated as normal order.

If an expressio can be evaluated in different orders then all of these will evaluate the will give the same result

Lazy Evaluation - Evaluate the actual parameter wen the argument is needed and store the value for whenever it is needed

Haskell uses lazy evaluation. Most others use eager evaluation

Functional Pragmatics

Power gained by higher order functions and lazy evaluation

Large Programs contain many functions and therefore problematica

Haskell - Pure functional language

No side effects

Primitive Types - Bool Char Numerics

Composite Types - Tuples, Algebraic, Lists, Functions

Lists - head and tail Parsed recursively

Haskell Bindings And Scope

Collection of modules

MAIN module

In a module there are constants types and functions

The scope is what is inside of block expression

Pattern Matching

Given value v if there exists bindings for these identifiers such that pattern yields v then pattern matches v

Higher Order Functions

Functions are first class values

Functions that have parameters which are ther functions are higher order functions

Haskell library contains filter map foldr and foldl