Haskell

Kelly Ochikubo

About Haskell

- Purely functional
  - Expressions only
  - No side-effects
- Statically typed, Type Inference
  - sum :: Int -> Int -> Int -- optional
    sum x y = x + y
    - sum 3 4
    - sum 2 “string” -- type error
- Lazy evaluation
  - Elegant and powerful but the trade off is overhead

List Comprehensions

- $Z_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - In Haskell: [0..9]
- $Z_5 \times Z_3 = \{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2),(3,0),(3,1),(3,2),(4,0),(4,1),(4,2)\}$
  - In Haskell: [(x,y) | x <- [0..4], y <- [0..2]]

Higher-Order Functions (HOF)

- Functions are first-class objects
- Functions in Haskell can take a function as a parameter and return functions as return values
  - map :: (a -> b) -> [a] -> [b]
    - Applies the function (a->b) to each element in the list [a] and returns a new list [b]

Project Code - HOF and Recursion

```
findSubgroupsZGroup :: [Int] -> [[Int]]
findSubgroupsZGroup [] = [[]]
findSubgroupsZGroup zgroup = map (findSub n) zgroup where
    n = length zgroup

findSub :: Int -> Int -> [Int]
findSub n 0 = [0]
findSub n gen = [0] ++ [value] ++ next value where
    value = modulus n gen 0
    next val = if (modulus n gen val) == 0
                then []
                else [modulus n gen val] ++ next (modulus n gen val)
```

Group Theory

Z_n and Z_a x Z_b

Project Demonstration

Challenges

- Purely functional
  - Previously only worked with imperative languages
- Code from scratch
  - Creating and implementing algorithms
- Haskell libraries
  - Pro: really powerful
  - Con: can be difficult to use without a good understanding of the language
- Recursion
  - No loops for iteration