Page 1

Introduction

Haskell is a statically-typed, lazy functional programming language. It is known for its strong type system, which allows for safe and reliable code, and its lazy evaluation, which defers computation until it is needed.

Data Types and Structures

* **Values:** Haskell values are immutable and have a well-defined type.
* **Types:** Haskell provides a rich type system, including basic types (e.g., Int, Bool), user-defined types (e.g., data), algebraic data types (e.g., Either a b, Maybe a), and type synonyms (e.g., type Length = Int).
* **Lists:** Lists are immutable ordered sequences of elements.
* **Tuples:** Tuples are immutable fixed-length collections of elements of different types.

Code Snippets:

```haskell
-- Define a data type
data Person = Person String Int
```

```haskell
-- Pattern matching on data types
```

```
case person of
    Person "Alice" age -> putStrLn $ "Alice is" ++ show age ++ "years old."
    _                  -> putStrLn "Unknown person."
```

Page 2

Functions

* **Function Definition:** Functions are defined using the syntax `f :: a -> b`, where `a` is the input type and `b` is the output type.
* **Pattern Matching:** Patterns can be used to deconstruct and process function arguments.
* **Lambda Functions:** Lambda functions are anonymous functions that are defined using the syntax `\x -> ...`.
* **Recursion:** Haskell functions can be recursive, allowing for elegant solutions to complex problems.

Code Snippets:

```haskell
-- Define a recursive function to calculate the factorial
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n-1)
```

```haskell
-- Use lambda functions to filter a list
filterPositive :: [Int] -> [Int]
filterPositive xs = filter (\x -> x > 0) xs
```

Page 3

Input and Output

* **Input:** Use `getLine` to read a line of input from the standard input.
* **Output:** Use `putStr` or `putStrLn` to write to the standard output.
* **Files:** Use the `readFile` and `writeFile` functions to read from and write to files.

Code Snippets:

```haskell
-- Read a line of input
name <- getLine

-- Write a message to the standard output
putStrLn "Hello, " ++ name ++ "!"
```

```haskell
```

```
-- Read contents of a file
contents <- readFile "input.txt"

-- Write contents to a file
writeFile "output.txt" contents
```

Additional Features

* **Monads:** Monads are a powerful abstraction that encapsulates state and computation, simplifying complex code.
* **Pattern Guards:** Pattern guards can be used to constrain pattern matching, ensuring that certain conditions are met.
* **Type Classes:** Type classes provide a form of polymorphism, allowing functions to operate on a wide range of types.

Resources

* [Haskell Wiki](https://wiki.haskell.org/)
* [Learn You a Haskell for Great Good!](https://www.learnyouahaskell.com/)
* [Haskell Tutorial](https://www.haskell.org/tutorial/)