Exam 3

Functions and Recursion in Lettuce

Key topics include the definition and use of functions, the principles of recursion, and the importance of base and recursive cases in recursive functions.

Functions

Functions are fundamental building blocks in functional programming. They encapsulate reusable code and can be passed as arguments to other functions or returned as values.

Functions in Lettuce

Functions in Lettuce allow for the creation of reusable and composable code blocks:

- **Definition**: Functions are defined using the "fun" keyword, followed by parameters and a body. For example, "fun(x) = x + 1" defines a simple function that increments its input.
- **First-Class Citizens**: Functions can be assigned to variables, passed as arguments, and returned from other functions, enabling higher-order functions.

Recursion

Recursion is a powerful technique where a function calls itself to solve smaller instances of the same problem. It is essential for implementing algorithms that can be naturally divided into similar subproblems.

Recursion in Lettuce

Recursion in Lettuce involves defining functions that call themselves:

- **Base Case**: The condition under which the recursion terminates. For example, in a factorial function, the base case is when the input is 0.
- **Recursive Case**: The part of the function that includes the recursive call, breaking the problem into smaller instances. For instance, "factorial(n) = n * factorial(n-1)".
- **Tail Recursion**: A special form of recursion where the recursive call is the last operation in the function. Tail-recursive functions are optimized by the compiler to prevent stack overflow.

Key Concepts

Key Concepts in Functions and Recursion in Lettuce

This section covers the core principles related to functions and recursion in Lettuce.
Functions:

- **Definition**: Creating reusable code blocks with the "fun" keyword.
- **First-Class Citizens**: Functions can be treated as values, enabling higher-order functions.

Recursion:

- **Base Case**: The terminating condition for recursion.
- **Recursive Case**: The self-referential part of the function that breaks down the problem.
- **Tail Recursion**: A form of recursion optimized by the compiler to prevent stack overflow.