**Week 3 summary:**

**Topics and Key Learnings from the Lectures**

**Lecture 3-1: Approximation Algorithms**

1. **Approximation Methods for Mathematical Functions:**

• **Sequential Search:** Brute-force approach to find approximate solutions, such as square roots, with trade-offs between accuracy (epsilon) and runtime.

• **Bisection Method:** Efficient technique dividing search intervals in half, reducing iterations logarithmically.

• **Newton-Raphson Method:** A derivative-based approach for fast convergence in solving mathematical equations like square roots.

2. **Implementing Functions in Java:**

• Example implementations for calculating square roots using different algorithms.

• Function overloading for flexibility in handling multiple inputs.

• Role of precision (epsilon) and its effect on results.

3. **Function Modularity and API Design:**

• Functions as black-box abstractions.

• Benefits of modularity: code reuse, reduced complexity, easier maintenance.

• Testing and skeleton design roles in development.

**Lecture 3-2: Handling Characters and Text**

1. **Character Encoding and Representation:**

• **ASCII vs Unicode:**

• ASCII (8-bit) for basic western characters.

• Unicode (16-bit) for broader global character representation.

• Characters as numeric codes in Java (char type).

2. **String Manipulation in Java:**

• String methods such as length, charAt, substring, and indexOf to process text.

• Overloaded methods (indexOf for characters and substrings).

3. **Tokenization:**

• **Tokenizer API Design:**

• Breaking a stream of characters into meaningful tokens.

• API for tokenizing strings with functions like hasMoreChars(), nextChar(), and nextInt().

4. **Practical Examples:**

• Text processing to identify characters, parse tokens, and modify strings.

• Implementation of text transformations, like capitalizing the first letter of words.

5. **System Output:**

• Exploration of System.out.println() and its overloaded variants for diverse data types.

**Overall Themes:**

• **Java Programming:**

• Core concepts such as function implementation, overloading, and modularity.

• Text processing techniques using char and String.

• **Algorithmic Thinking:**

• Approximation techniques balancing accuracy and performance.

• Efficient computation methods like Newton-Raphson.

• **API and Modularity:**

• Designing reusable, abstracted APIs.

• Benefits of dividing code into manageable modules for collaboration and testing.