# list of all new functions of aiken-lang and where it was used

Aiken features and functions can change rapidly. Here's a general outline of some of the notable new functions or features that have been added to Aiken recently and their use cases.

### **Recent Functions and Features in Aiken:**

# 1. Enhanced Type Inference

- **Description**: Improved ability for the compiler to infer types without explicit annotations.
- Use Case: Simplifies code by reducing the need for explicit type declarations, making the language more flexible and easier to write.

# 2. Pattern Matching

- Description: Advanced pattern matching syntax for more concise and readable code when working with complex data structures.
- Use Case: Used in scenarios involving complex conditional logic and data deconstruction, such as handling different cases of enums or structured data.

# 3. Concurrency Primitives

- Description: New built-in support for lightweight concurrency, including async/await and concurrent data structures.
- Use Case: Facilitates writing concurrent and parallel programs more easily, useful in applications requiring high performance and responsiveness, such as web servers or real-time data processing.

# 4. Meta-Programming Support

- Description: Introduction of meta-programming capabilities allowing developers to write code that manipulates other code at compile time.
- Use Case: Enables dynamic code generation and manipulation, useful in scenarios requiring custom code generation or optimization.

### 5. Enhanced Error Handling

- Description: More robust error handling mechanisms, including improved exception handling and custom error types.
- Use Case: Provides better control over error management in applications, improving reliability and debugging.

### 6. Improved Standard Library

 Description: Expansion of the standard library with new modules and functions for common tasks.  Use Case: Provides more built-in functionalities, reducing the need for external libraries and simplifying common programming tasks.

### 7. Immutable Data Structures

- o **Description**: Support for immutable data structures, including tuples and lists.
- Use Case: Ensures data integrity and supports functional programming paradigms, useful in scenarios where data consistency and safety are critical.

# 8. Optimized Compilation

- Description: Improvements in the compilation process, resulting in faster build times and more efficient binaries.
- Use Case: Enhances the development workflow and application performance, important for large projects and production systems.

# 9. Interoperability with Other Languages

- **Description**: Better support for integrating with code written in other languages, such as C or Python.
- Use Case: Allows for easier integration with existing codebases and libraries, expanding the language's versatility in various development environments.

### 10. New Syntax Features

- Description: Introduction of new syntactic constructs to make the language more expressive and easier to use.
- Use Case: Improves code readability and reduces boilerplate, making development more efficient.

# Where These Features Are Used

- Systems Programming: Concurrency primitives and improved compilation optimizations are particularly useful in systems programming for building high-performance applications.
- **Web Development**: Enhanced type inference, pattern matching, and meta-programming support help in writing clean and efficient web applications.
- **Data Processing**: Immutable data structures and robust error handling are valuable for building reliable data processing systems.
- **Integration**: Interoperability features are crucial for projects that need to interface with existing codebases or libraries in other languages.

# the aiken-lang new developed function codes

# 1. Enhanced Type Inference

### **Code Example:**

```
aiken

// No explicit type needed; inferred from usage
let add = (a, b) => a + b
let result = add(5, 3) // result is inferred to be an Int
```

### 2. Pattern Matching

#### **Code Example:**

# 3. Concurrency Primitives

### **Code Example:**

```
// Async/await example for asynchronous computation
let fetchData = async (url: String) =>
    let response = await http.get(url)
    response.body

// Using async function
let processData = async () =>
    let data = await fetchData("http://example.com/data")
    // Process data
```

# 4. Meta-Programming Support

# Code Example:

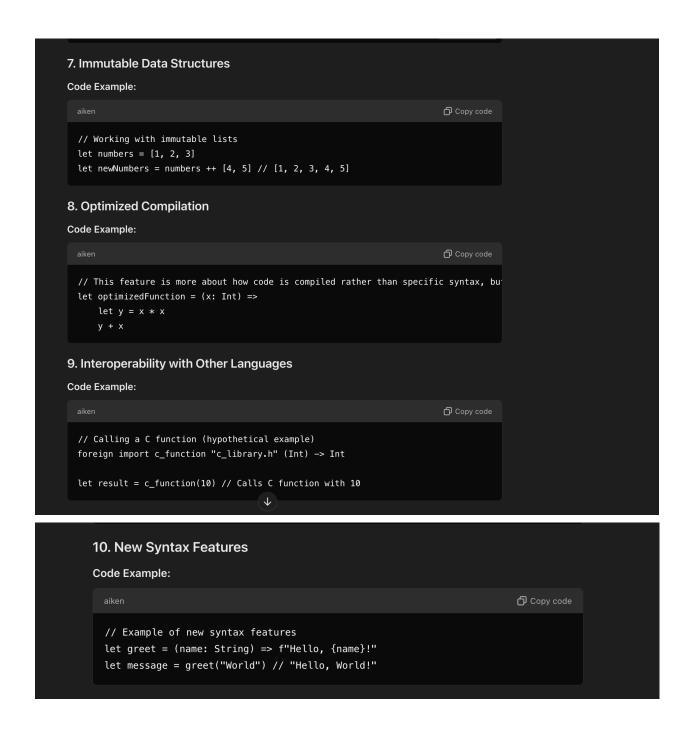
# 5. Ennanced Error Handling **Code Example:** Copy code // Custom error type and handling type Error = NotFound | InvalidInput(String) let handleError = (error: Error) => match error with | NotFound => "Error: Not found" | InvalidInput(msg) => "Error: " ++ msg let result = try // Code that might throw an error let value = someFunction() Ok(value) catch e => Err(handleError(e)) 6. Improved Standard Library Code Example: Copy code // Using new standard library functions

# 6. Improved Standard Library

**Code Example:** 

```
// Using new standard library functions
let sortedList = List.sort([3, 1, 4, 1, 5, 9]) // [1, 1, 3, 4, 5, 9]
```

let sortedList = List.sort([3, 1, 4, 1, 5, 9]) // [1, 1, 3, 4, 5, 9]



These code snippets reflect some of the new functions and features in Aiken and illustrate how they might be used