# Add security to functional programming

* Avoiding Global State
* Focus on writing pure functions, which means functions that have no side effects and always return the same output for the same input. Pure functions make it easier to understand and predict the behavior of your code
* Avoiding Eval and Unsafe Operations
* Secure Input/Output:

Solid Principles

* Single Responsibility Principle (SRP)
* Open/Closed Principle (OCP)
* Liskov Substitution Principle (LSP)
* Interface Segregation Principle (ISP)
* Dependency Inversion Principle (DIP)

scalability in functional programming

* Embrace immutability and write pure functions wherever possible
* Choose functional data structures that have efficient time complexities for common operations like insertion, deletion, and lookup.
* Use asynchronous programming to handle I/O-bound operations efficiently.
* Adopt message passing between components of your system instead of shared mutable state.
* Adopt scalable architecture patterns, such as microservices or actor-based systems, which fit well with functional programming principles.
* Use caching and memorization techniques to store the results of expensive computations and avoid recomputing them when the same inputs occur again.