Here's a revised version of the article, formatted in markdown, that argues strongly against OOP and asserts that functional programming is the better path, while incorporating your feedback and maintaining the core ideas:

Why Object-Oriented Programming Was a Mistake

Object-oriented programming (OOP) has been sold as a cornerstone of software development, but it's a flawed paradigm we never should have embraced. Functional programming, with its emphasis on immutability and simplicity, offers a clearer, more robust alternative. Comparing the two is like contrasting a convoluted bureaucracy with a streamlined system—OOP is a path we should have avoided.

The False Promise of OOP

OOP was pitched as a way to model the world by bundling data and behavior into objects. But this premise falls apart under scrutiny. Functional programming, which predates many OOP concepts, already provided elegant solutions without the baggage. The idea that OOP and functional programming can coexist misses the point: OOP's core assumptions are fundamentally misguided.

Immutable vs. Mutable OOP: A Flawed Spectrum

OOP is often divided into **mutable** and **immutable** camps, but both are problematic:

- **Mutable OOP**, dominant in languages like C++, Java, C#, and JavaScript, allows objects to change state freely. This leads to chaos—mutable state is an antipattern that breeds complexity and undermines reliability.
- **Immutable OOP**, seen in constructs like the `String` class, tries to mitigate this by preventing state changes. But it's a half-measure, patching a broken system rather than rethinking it.

Mutable OOP is a disaster for reasoning about code. Programs become harder to understand as state changes ripple unpredictably. Concurrency? Forget it—mutable state invites race conditions and brittle synchronization hacks. Programmers who've wrestled with these issues know the pain: mutable OOP is a mistake.

Immutable OOP fares slightly better but doesn't redeem the paradigm. It's a compromise that still clings to OOP's flawed idea of tying functions to datatypes. We don't need this crutch.

Functional Programming: The Path We Should Have Taken

Functional programming, with its focus on immutability and pure functions, avoids OOP's pitfalls entirely. It's not just compatible with good design—it's the foundation of it. Languages like Erlang demonstrate how immutability simplifies concurrency and reasoning, leaving OOP's complexity in the dust.

Consider the `String` class, a staple of OOP:

- Language designers pack it with a fixed set of methods (`substring`, `toUpperCase`, etc.), assuming they've anticipated every need.
- They usually do a decent job, but why should developers be limited to this predefined toolbox?

If you need a custom string operation, OOP forces you to subclass `String` or create a new class. This is unnecessary friction. Functional programming sidesteps this by treating functions as standalone entities, not shackled to a datatype.

The OOP Straitjacket: Tying Functions to Datatypes

OOP's insistence on binding functions to datatypes is its fatal flaw. Functions often rely on specific datatypes, but that's no reason to lock them inside a class. For example:

- Want to write a new string function? In OOP, you're stuck extending a class or bloating an existing one.
- In functional programming, you write a standalone function and move on. No ceremony, no fuss.

Modern languages provide **modules**, **libraries**, and **namespaces** to group related functions without OOP's rigidity. These mechanisms let you organize code logically, linking functions to datatypes without chaining them to a class. OOP's encapsulation feels like a relic when you see how cleanly functional programming handles this.

Conclusion: Abandon OOP

OOP was a wrong turn in the history of programming. Its obsession with mutable state and datatype-bound functions creates complexity, undermines concurrency, and limits flexibility. Functional programming, with its immutable data and loosely coupled functions, is the path we should have followed all along. Modules and namespaces give us all the organization we need without OOP's baggage.

It's time to admit the mistake and move forward. Functional programming isn't just an alternative—it's the future.