Rust JSON Parser Cohort – Syllabus

Program outcome
Build a **high-performance JSON parsing library in Rust** that you can import from Python (via PyO3) – a practical, portfolio-ready project that demonstrates real systems-level skills.

Who it's for
Developers with experience in Python/JS/Ruby who want to add Rust (and performance work) to their toolkit. **No prior Rust experience necessary!**

Tech Stack & Practices

- **Rust (2021 edition)**, `cargo` for builds & tests
- Core concepts: ownership, borrowing, enums, structs, Result/Option
- **PyO3** for Python bindings
- Basic **unit tests** in Rust (`#[test]` + `cargo test`)
- Optional Python tests / benchmarks (pytest)

Learning Path

Setup & Basics → Tokenizer → Value Types → Collections → Python Integration → Optimization & Polish

---

Week-by-Week Breakdown

Note: All weeks take a test-driven-development approach. Tests are provided to validate correct functionality of your Rust code.

Week 1 – Setup & Basic Tokenization

Goal: Get comfortable with Rust basics and produce a working tokenizer that processes basic JSON.

- Project setup with `cargo new`
- Variables, functions, `String` vs `&str`
- Define a `Token` enum
- Implement a simple tokenizer + a few `#[test]`s
- Run tests with `cargo test`

Week 2 – Rust Types & Error Handling

Goal: Model JSON values and handle errors explicitly.

- Enums + `match`
- `Option<T>` and `Result<T, E>` + `?` operator
- Simple `JsonValue` enum for primitives (string, number, bool, null)
- Custom `JsonError` type and basic error reporting
- Tests for valid and invalid inputs

Week 3 – Parser Structure & Borrowing

Goal: Introduce stateful parsing and deepen ownership/borrowing intuition.

- Structs and methods (`impl`)
- References and borrowing (`&`, `&mut`)
- Design a `JsonParser` struct with internal state
- Implement parsing methods + string parsing with escapes
- Tests to exercise parser state and string handling

Week 4 – Collections & Complete Parser

Goal: Support full JSON (arrays + objects, including nesting).

- `Vec<T>` and `HashMap<K, V>`
- Use `Vec` as an explicit stack (iterative parsing instead of recursion)
- Extend `JsonValue` with `Array` and `Object`
- Parse arrays/objects and nested structures
- Test with realistic and more complex JSON examples

Week 5 – Python Integration (PyO3)

Goal: Call your Rust parser from Python.

- FFI basics and the role of PyO3
- Configure `Cargo.toml` for a Python extension module
- Implement Python-callable `parse_json(...)` in Rust (`#[pyfunction]`)
- Convert `JsonValue` ↔ Python types (dict, list, str, etc.)
- Import and exercise the parser from Python

Week 6 – Optimization, Testing & Polish

Goal: Make it feel like a real library: faster, documented, and benchmarked.

- Basic performance measurement and simple benchmarking
- Small optimizations (string handling, allocations)
- Compare speed against Python's built-in `json` module
- Add Rust docs (`///` comments, README examples)
- Clean up the API and code layout

Optional Extra – Performance Deep Dive

For those who finish early and want more:

- Profile the parser and identify bottlenecks
- Apply targeted optimizations to “hot” paths
- Document the before/after results and lessons learned

Outcome: A mini performance report showing how you made Rust actually fast in practice.