Roblox Lua (Luau) Onboarding and Standards

A practical guide for programmers building Roblox games

This guide defines what every developer must learn and the non-negotiable rules for writing secure, maintainable, and performant Roblox code. The goal is simple: ship features quickly without creating tech debt or exploit risk.

1. Must Learn (Core Topics)

- Luau fundamentals: types, tables, functions, control flow, modules
- OOP patterns (light): objects via tables and metatables; composition over inheritance
- Roblox Studio basics: Explorer, Properties, hierarchy, Services, Play Solo vs Team Test
- Client vs Server: replication rules, what runs where, why server authority matters
- Remotes: RemoteEvents and RemoteFunctions; validation and anti-exploit patterns
- Attributes: replicated state, change signals, when Attributes are a good fit
- UI: ScreenGui, Frames, Buttons, layouts (UIListLayout, UIPadding), UI constraints
- Tweening: TweenService for UI motion, transitions, and feedback
- Saving: DataStore basics plus Profile-style saving concepts (session lock, retries, backoff)
- Performance: RunService usage, avoiding heavy per-frame work, replication cost
- Cleanup: preventing memory leaks from connections and instances; lifecycle management

2. Recommended 7-10 Day Learning Roadmap

Move fast by focusing on deliverables. Each day ends with a small working feature you can demonstrate in Studio.

Day	Focus	Deliverable
1	Luau basics + Studio navigation	A small test place with parts spawning and a simple UI button
2	Modules + simple OOP patterns	A ModuleScript service (e.g., CoinService) with a clean API
3	Client vs Server + Remotes	Client input triggers server validated reward; UI updates
4	UI layouts + TweenService	Shop UI with animated open/close and upgrade button
5	Saving basics	Coins + upgrades load/save without spamming DataStore
6	Security + validation	Server-side checks for every remote; exploit-resistant purchases
7	Performance + cleanup	No runaway loops; connections cleaned on player leave
8-10	Mini project	A tiny simulator loop: earn -> upgrade -> save -> rebirth (optional)

3. Roblox Studio Essentials (Quick Reference)

Know where code and objects live. Many production bugs come from incorrect placement and replication assumptions.

- ServerScriptService: server-only scripts and services
- ServerStorage: server-only assets (not replicated)
- ReplicatedStorage: shared modules and assets used by both client and server
- StarterPlayerScripts: client logic that runs for the player (LocalScript)
- StarterCharacterScripts: scripts inserted into a character on spawn (client or server depending on script type)
- StarterGui: UI templates (ScreenGuis) replicated to each player
- Workspace: the 3D world; replicated objects live here
- CollectionService: tag instances for scalable systems (avoid hard-coded paths)
- RunService: Heartbeat/Stepped/RenderStepped; use carefully for performance
- TweenService: animation and UI feedback via tweens
- DataStoreService: persistent saving (with throttling and failure cases)
- MarketplaceService: developer products and receipts validation

4. Client vs Server (Non-Negotiable Rules)

- Server owns truth: money, inventory, rewards, progression, purchases, leaderboards.
- Client does UI + input: visuals, camera, effects, menus, and sending requests to the server.
- Never trust the client: all remote requests must be validated on the server.
- Validate everything: distance checks, cooldowns, required items, cost, and rate limits.
- Design for replication: assume anything in Workspace can be observed by clients.

Remote validation pattern (example)

```
-- Client (LocalScript)
RemoteEvent:FireServer("BuyUpgrade", { upgradeId = "Speed1" })

-- Server (Script)
RemoteEvent.OnServerEvent:Connect(function(player, action, payload)
    if action == "BuyUpgrade" then
        -- Validate: payload shape, upgrade exists, player has coins, cooldown, etc.
        -- Apply: deduct coins + grant upgrade on the server
        -- Notify: send updated state back to client (Attributes or RemoteEvent)
    end
end)
```

5. Saving and DataStores (Best Practices)

- DataStores can fail. Always use pcall and handle errors.
- Do not save every time coins change. Save on player leave and on a timer (e.g., every 60-180 seconds).
- Use retries with backoff for transient failures (avoid tight retry loops).

- Store compact data (numbers, short strings). Avoid saving large tables every time if you can.
- Keep an in-memory state and only write to DataStore on save moments.
- Understand session locking concepts: only one server should own a profile at once (Profile-style systems help).
- Plan for data versioning: add a version number and migrate old saves safely.

Simple safe save wrapper (example)

```
local function retry(times, fn)
    local lastErr
    for i = 1, times do
        local ok, result = pcall(fn)
        if ok then return true, result end
        lastErr = result
        task.wait(0.5 * i) -- backoff
    end
    return false, lastErr
end
```

6. UI Basics (What to Know)

- Use layout objects: UIListLayout, UIGridLayout, UIPadding to keep UI consistent.
- Use constraints: UIScale, UIAspectRatioConstraint for device and resolution scaling.
- Keep UI logic in LocalScripts; request server actions through remotes.
- Animate feedback with TweenService (hover, press, open/close).

7. Performance and Cleanup

- Avoid heavy per-frame work. If you need a loop, prefer task.wait with a sensible interval.
- Disconnect events and destroy objects you create. Leaks usually come from forgotten connections.
- Avoid creating thousands of instances rapidly; batch spawns and reuse where possible.
- Minimize frequent replication of many instances; replicated spam equals network + client cost.
- Use tags and batching for large systems (CollectionService).

8. Debugging and Testing

Use the tools. Debugging skills are required, not optional.

- Developer Console: view logs, warnings, errors, and client/server separation. (Doc: https://create.roblox.com/docs/studio/developer-console)
- Script Profiler: record scripts to find what is consuming CPU time. (Doc: https://create.roblox.com/docs/studio/optimization/scriptprofiler)
- MicroProfiler: deep performance capture for spikes and frame timing. (Doc: https://create.roblox.com/docs/performance-optimization/microprofiler)
- Performance dashboard: live client/server metrics in production. (Doc: https://create.roblox.com/docs/production/analytics/performance)
- Test with multiple clients (Team Test) and simulate spam clicking to catch remote abuse early.

9. Security Checklist (Avoid Exploits)

- Never accept 'new coin amount' from the client. Client should only request actions, not results.
- Validate remote payloads: required fields exist, types are correct, and values are in allowed ranges.
- Whitelist actions (strings) on the server. Reject unknown actions.
- Rate limit and add cooldowns per action (per player).
- Validate proximity for interactions (e.g., player near the object, line-of-sight if needed).
- Protect double-collect: mark items as collected on the server; ignore repeats.
- Use server-side timestamps and server-side state for any reward timing.
- For purchases: process developer product receipts on the server via MarketplaceService.ProcessReceipt.

10. Suggested Project Structure (Simple and Scalable)

This structure keeps responsibilities clear and prevents spaghetti scripts. Adjust naming to match your studio standards.

```
ReplicatedStorage
  Shared
    Remotes
    Config
    Util

ServerScriptService
  Server
    Services
    Systems

StarterPlayer
  StarterPlayerScripts
    Client
      Controllers
      UI

StarterGui
  UI
    Screens

Workspace
  Map
  Interactables (tagged via CollectionService)
```

11. Coding Standards (Required)

- Modular architecture: systems live in ModuleScripts with clear responsibilities (InventoryService, ShopService, etc.).
- Single responsibility: one module should do one job well.
- Naming: consistent, descriptive names (no random abbreviations).
- No hard-coded paths: prefer tags, configuration tables, or dependency injection.

- Logging: warnings for unexpected states; never silently fail in production code.
- Cleanup: every connection created must be disconnected (or use Trove/Janitor style cleanup).

12. Starter Assignment (Recommended)

- Task A: coin collect system with server authority and UI display.
- Task B: upgrade shop (server-validated purchase) that increases coins per collect.
- Task C: saving coins + upgrade level (safe pcall + periodic save).

13. Learning Resources

Official documentation (start here)

- Roblox Documentation: https://create.roblox.com/docs
- Roblox Engine API Reference: https://create.roblox.com/docs/reference/engine
- RemoteEvents and client-server communication: https://create.roblox.com/docs/scripting/events/remote
- Data stores guide: https://create.roblox.com/docs/cloud-services/data-stores
- Luau language reference: https://luau.org
- MarketplaceService class: https://create.roblox.com/docs/reference/engine/classes/MarketplaceService
- Developer products (receipt processing): https://create.roblox.com/docs/production/monetization/developer-products
- MicroProfiler (performance): https://create.roblox.com/docs/performance-optimization/microprofiler
- Improve performance (best practices): https://create.roblox.com/docs/performance-optimization/improve
- HttpService (web requests): https://create.roblox.com/docs/cloud-services/http-service
- Secrets store (store API keys safely): https://create.roblox.com/docs/cloud-services/secrets
- Roblox DevForum: https://devforum.roblox.com

Community and open-source (optional but useful)

- ProfileService (popular Profile-style saving): https://github.com/MadStudioRoblox/ProfileService
- Trove (cleanup utility): https://github.com/Sleitnick/RbxUtil/tree/main/modules/trove
- Janitor (cleanup utility): https://github.com/howmanysmall/Janitor
- Knit (lightweight framework): https://github.com/Sleitnick/Knit
- Promise library (async patterns): https://github.com/evaera/roblox-lua-promise
- Utility modules (Signal, Trove, etc.): https://github.com/Sleitnick/RbxUtil

YouTube channels (practical learning)

- TheDevKing - beginner to intermediate Roblox scripting
- AlvinBlox - solid fundamentals and explanations
- BrawlDev - systems-based Roblox scripting
- GnomeCode - architecture and advanced scripting patterns

Expectation: before writing production code, you should be able to build a mini loop (earn -> upgrade -> save) with server authority, validated remotes, clean module structure, and proper cleanup.