Audio App Learning Schedule

Below is a prioritized roadmap of the programming language and JUCE framework concepts you'll need to take your Ear Fatigue/Auto Dynamicsplugin from idea to working prototype. I've grouped them by "Learn First," "Then," and "Later," so you can focus your study in a logical order.

Learn First • C++ Fundamentals:

1. Syntax and build process (headers, cpp files, compilation)
2. Basic types, functions, control flow
3. Classes & objects, constructors/destructors, RAII – References vs. pointers, smart pointers (std::unique_ptr, std::shared_ptr)
4. std::vector and other STL containers
5. Namespaces, header guards/pragma once

Real TimeAudio Programming Concepts:

1. Audio callback model (processBlock)
2. Block vs. sample processing; sample rate and buffer size
3. Avoiding dynamic allocation in the audio thread
4. Thread safety basics (lock freepatterns, atomic operations)

JUCE Project Setup:

1. Installing JUCE and Projucer
2. Creating a new Audio Plugin project template
3. Understanding the module structure (juce_core, juce_audio_basics, juce_audio_processors, juce_gui_basics)

Then

JUCE AudioProcessor API:

1. prepareToPlay, releaseResources, processBlock overrides
2. Getting input/output buffers and channels

3. Implementing basic gain or bypass processing

Parameter Management:

1. AudioProcessorValueTreeState for parameter storage
2. SliderAttachment, ButtonAttachment to hook GUI to parameters
3. Preset handling (save/load state information)

Basic GUI Components:

1. AudioProcessorEditor skeleton: paint(), resized()
2. juce::Slider, juce::Button, juce::Label, layout with resized()
3. Understanding repaint()/setRepaintsOnParameterChange

Intermediate

1. DSP Algorithms for Metering & Dynamics
2. Peak and RMS level detection
3. Simple envelope follower (attack/decay)
4. LUFS approximation basics (block RMS + integration)
5. Mapping DSP results to “traffic□ light”ranges • JUCE DSP Module
6. juce::dsp::FFT for any spectral components
7. juce::dsp::WindowingFunction for FFT
8. juce::dsp::IIR or FIR filters (if you need spectral pre□ filtering)

Custom Graphics & Visualization:

1. juce::Graphics drawing primitives (lines, fills, gradients)
2. juce::Image and off□ screenbuffers for efficient refresh
3. Timer callbacks (juce::Timer) to trigger GUI updates at ~30–60 Hz

Later

1. Reference□ TrackFile I/O
2. juce::AudioFormatManager
3. AudioFormatReader to load WAV/MP3
4. Buffering and real□ timeanalysis of a reference track

5. Advanced UI/UX
6. LookAndFeel customization for traffic‑light‑color themes
7. Responsive component layouts (FlexBox or custom algorithms)
8. User preferences storage (PropertiesFile)

Plugin Deployment & Formats:

1. VST3, AU, AAX settings in Projucer
2. Code signing, building debug vs. release bundles
3. Performance Optimization
4. SIMD optimizations (juce::dsp::SIMDRegister)
5. Reducing GC‑style overhead: avoid heap allocations in processBlock
6. Profiling with Instruments/Visual Studio Profiler
7. Testing & Continuous Integration
8. Unit tests for DSP code (Catch2 or GoogleTest)
9. Automated build scripts (CMake, Projucer CLI)
10. Smoke testing in multiple DAWs

Can Learn Even Later / Optional:

1. Multi‑threaded UI vs. audio‑thread communication patterns
2. Online auto‑update frameworks or plugin licensing
3. Integration of machine‑learning–based loudness prediction
4. Advanced psychoacoustics libraries or cross‑platform VST3 SDK hacks

By following this sequence you'll first build a rock‑solid foundation in C++ and real‑time audio, then get up and running in JUCE, and only afterwards layer in the DSP, UI polish, file I/O, and deployment skills you need to finish the Ear‑Fatigue/Auto‑Dynamics plugin. Good luck, and happy coding!