Lab 05: Signal workshop

You'll write a signal-analysis script yourself, one function at a time: it synthesizes a chord out of harmonic-rich tones, plots its waveform and spectrum, filters it with an RC low-pass, and exports the chord before and after filtering so you can hear the filter change its timbre without changing its pitch.

This lab works differently from Labs 01–04: instead of moving sliders, you fill in a scaffold, run it, and listen to what it produces.

Lab package: `labs/lab-05-signal-workshop/`

Before you start

- Finish Labs 01–04 first, because this lab puts the ideas from their snippets together into one script.
- Read all four movements, What is a signal? through More filter shapes.
- Keep Music and signals open for its note-frequency table.
- In a terminal in the noVNC desktop, go to the lab folder:

```
. /foss/designs/common/.designinit
mod ad101
cd labs/lab-05-signal-workshop
```

Time: about 2 hours.

How the scaffold works

Open `src/workshop.py` in an editor. Seven functions are stubbed out with `raise NotImplementedError(...)`, and each one has a docstring that tells you exactly what to write. The part numbers below match the `Part` labels in those docstrings and in the comments in `run()`:

Part	Function	What it does
1	`synthesize_chord`	Builds a C-major triad out of harmonic-rich tones
1	`plot_waveform`	Plots voltage against time
2	`compute_spectrum`	Wraps `common.adsig.spectrum`
2	`plot_spectrum`	Plots magnitude in dB against frequency
3	`apply_lowpass`	Filters a signal by multiplying its spectrum by $H(f)$
4	`export_wav`	Writes a signal to a `.wav` file
4	`play_audio`	Plays a `.wav` file, with a fallback if there's no audio device

The helpers the docstrings mention (`harmonic_sum`, `house_style`, `spectrum`, `to_db`, `rc_lowpass`) are already imported at the top of the file from `labs/common/adsig.py`, the course's signal library.

The script's `run()` function is already written. It calls your functions in that order and builds three figures from them: the waveform, the spectrum, and a before/after comparison that uses `plot_spectrum` twice. You'll run it two ways:

```
python3 src/workshop.py --headless   # save PNGs + WAVs into results/, skip live playback
python3 src/workshop.py              # full run: plot windows + live audio, if available
```

Until every function is filled in, the script stops at the first one that still raises `NotImplementedError`, and the error message says which step that is. That's how you'll know a part works: once its function is filled in, the error moves on to the next one.

If you're stuck on one function, `solutions/workshop_solution.py` has a filled-in reference, but try to get further on your own first. If you do run it, the end of this page shows how to do that without overwriting your own results.

Run the empty scaffold

1. Run it before changing anything:

```
python3 src/workshop.py --headless
```

You should see a traceback ending in `NotImplementedError: TODO: synthesize the chord — see the docstring above`.

Part 1: Build and plot the chord

How do you turn three notes into one harmonic-rich chord, and plot it?

1. Fill in `synthesize_chord`. Its docstring has you build each note with `harmonic_sum(t, fundamental=freq, harmonics=[1, 3, 5, 7, 9])`, add the three notes, and divide by the number of notes so the result stays roughly within ±1. The notes are `CHORD_NOTES` at the top of the file: C4, E4 and G4 at 261.6, 329.6 and 392.0 Hz.

   This is the chord from Lab 02, Part 1, except that each note now has odd harmonics, like the square wave from Lab 02, Part 2, and those high harmonics give the filter in Part 3 something it can audibly change.

2. Fill in `plot_waveform`: time in milliseconds on the x-axis, voltage on the y-axis, with `house_style(ax)` applied so it matches Labs 01–04.

3. Run `python3 src/workshop.py --headless` again.

You should see the traceback move on and end in `NotImplementedError: TODO: compute the spectrum — see the docstring above`. `run()` saves all three figures together after the filter step, so you'll check your waveform plot once Part 3 works.

Part 2: Compute and plot the spectrum

How do you get from the chord's samples to its spectrum in dB?

1. Fill in `compute_spectrum`. It calls `spectrum(v, fs)` from `common.adsig`, which wraps `numpy.fft.rfft` the way your Lab 03 snippets did by hand, and also scales the result so that a pure tone's peak comes out close to its amplitude rather than at a raw FFT magnitude.

2. Fill in `plot_spectrum`: magnitude in dB (use `to_db`) against frequency, with the x-axis limited to something readable for a chord. The docstring suggests `ax.set_xlim(0, 3000)`.

3. Run the script again.

You should see the error move on to `NotImplementedError: TODO: apply the RC low-pass — see the docstring above`.

Part 3: Filter the chord

What does an RC low-pass do to the chord's spectrum?

To filter a signal in the frequency domain, you transform it, multiply by the filter's response $H(f)$, and transform back. For the RC low-pass:

$$H(f) = \frac{1}{\sqrt{1 + (f/f_c)^2}}\, e^{-j \arctan(f/f_c)} \qquad f_c = \frac{1}{2\pi RC}$$

The first factor is the magnitude and the exponential carries the phase, the same two curves as the Bode plot in Lab 04. j is the imaginary unit ($j^2 = -1$; electrical engineers write j instead of i). You don't need to work with complex numbers by hand here, because numpy does the arithmetic once you've built `H`.

1. Fill in `apply_lowpass`, following the four numbered steps in its docstring: get `freqs`, get the magnitude and phase from `rc_lowpass` (the phase comes back in degrees, so convert it with `np.deg2rad`), build the complex `H`, then multiply in the frequency domain and transform back with `np.fft.irfft`. It's the same method as the `rc_lowpass_filter` snippet in Lab 04, Part 3.

2. `FILTER_R` and `FILTER_C` at the top of `src/workshop.py` set the filter to R = 1 kΩ and C = 150 nF. Work out f_c and write it down. Then run `python3 src/workshop.py --headless` again.

   You should see `Filter cutoff fc = 1061 Hz`, then three `wrote …` lines for `results/workshop-waveform.png`, `results/workshop-spectrum.png` and `results/workshop-before-after.png`. The run then stops at `NotImplementedError: TODO: export a WAV file — see the docstring above`.

3. Open the three PNGs and check them:

   - Waveform: a dense, harmonic-rich wiggle, not a clean sine, because each note is built from several harmonics.
   - Spectrum: three clear peaks near 262 Hz (C4), 330 Hz (E4) and 392 Hz (G4), plus smaller peaks at their odd harmonics.
   - Before/after: the three fundamental peaks survive filtering, and the higher-harmonic peaks are visibly shorter after filtering than before.

Part 4: Export and listen

Can you hear what the filter did?

1. Fill in `export_wav`. Its docstring gives the normalization to 16-bit integers and the standard-library `wave` calls.

2. Fill in `play_audio`. Try `sounddevice` first, with the imports and the playback all inside one `try/except Exception`. If it fails, print the file's path and tell the user to open it from their host's file manager. Playback failing must never crash the script.

   Neither `sounddevice` nor `soundfile` (which the docstring uses to read the WAV back) is installed in the course image. Both are listed as optional in this lab's `requirements.txt`; install them with `pip install --user sounddevice soundfile` if you want to try live playback. That install goes into the container, not your `workspace` folder, so you'd have to repeat it if you remove the container. Without them your fallback runs instead, and that's fine, because the noVNC desktop may not have a sound device anyway; see Python in the workbench.

3. Run the headless version:

   ```
   python3 src/workshop.py --headless
   ```

   You should see the same output as Part 3, then `wrote …/results/chord-unfiltered.wav` and `wrote …/results/chord-filtered.wav`, with no traceback.

4. Run the full version:

   ```
   python3 src/workshop.py
   ```

The three plots open first. Close them, and the script plays the unfiltered chord and then the filtered one, or prints your fallback message for each if live playback isn't available.

5. Listen to `chord-unfiltered.wav` and `chord-filtered.wav`. If playback doesn't work over your noVNC session, open them from your host machine: they're in `workspace/modules/ad101/labs/lab-05-signal-workshop/results/`.

   You should hear the same notes in both files. The filtered one sounds duller and more muffled, but its pitch is the same.

   Why does the filter change the chord's timbre but not its pitch?

   Answer: A1, in Answers at the end.

Finishing

You're done when `python3 src/workshop.py --headless`, running your own seven functions, finishes without a traceback and writes three PNGs and two WAVs into `results/`, and your before/after spectrum shows the three fundamentals surviving and the higher harmonics shorter.

Questions

1. Why does the filter change the chord's timbre but not its pitch? (Answered in Part 4, step 5.)

Submit

Nothing to submit.

To compare your work with the reference, run the solution with its own output folder so it doesn't overwrite your files:

```
python3 solutions/workshop_solution.py --headless --out results/solution
```

Next on the Course path: this was the last step of AD101. Carry on with AD102: Linear Circuits & Fabrication.

Answers

A1. Part 4: Export and listen

Pitch is set by the fundamentals, 261.6, 329.6 and 392.0 Hz, all well below f_c = 1061 Hz, so they come through almost unchanged; that's the three peaks surviving in your before/after spectrum from Part 3. Timbre is the recipe of higher harmonics on top of each note, and the low-pass turns those down.