# Lab 2: Sound Analytics with Raspberry Pi 4/3 using Microphone

### 1. Setup & Environment

**Q1: Which command updates and upgrades the system packages on the Raspberry Pi before installing audio libraries?**
**Answer:**

- sudo apt update
- sudo apt upgrade

---

**Q2: Why is it recommended to create a virtual environment named "audio" for this lab?**
**Answer:**
To isolate the lab's Python dependencies (e.g., PyAudio, sounddevice, librosa) from the system-wide Python packages, preventing version conflicts and keeping the system environment clean.

---

**Q3: How do you create and activate a virtual environment called "audio" on Raspberry Pi?**
**Answer:**

1. sudo apt install python3-venv (if needed)
2. python3 -m venv audio
3. source audio/bin/activate

---

### 2. Connecting & Testing the Microphone

**Q4: Which command would you use to record a 10-second audio file named test.wav using the terminal?**
**Answer:**
arecord --duration=10 test.wav

---

**Q5: How do you play back the recorded file test.wav on Raspberry Pi?**
**Answer:**
aplay test.wav

---

**Q6: Why is it important not to delete the test.wav file after testing the microphone?**
**Answer:**
Because it can be used later for feature extraction or demonstration of filtering, spectral analysis, or other sound processing tasks.

---

**3. Basic Sound Processing with Python**

**Q7: Which libraries can be used to capture audio from a microphone in Python on a Raspberry Pi?**
**Answer:**

- **PyAudio** (pip install pyaudio)
- **sounddevice** (pip install sounddevice)

---

**Q8: What is the Fourier Transform, and why is it important in audio analysis?**
**Answer:**
The Fourier Transform decomposes a time-domain signal into its frequency components. It's essential in audio analysis to understand which frequencies (or pitches) are present in a sound and how strong they are.

---

**Q9: Which Python libraries are commonly used for numerical computations and plotting audio data?**
**Answer:**

- **NumPy** or **SciPy** for numerical operations
- **Matplotlib** for plotting and visualizing waveforms or spectra

---

**4. Visualization of Audio Signals**

**Q10: What does the top plot (time series) represent in an audio waveform visualization?**
**Answer:**
It shows **amplitude vs. time**, indicating how the sound pressure (or signal level) changes over time.

---

**Q11: What does the bottom plot (audio spectrum) typically represent?**
**Answer:**
It shows **frequency vs. amplitude** (or power), revealing which frequencies are present and their relative strengths in the sound at a given moment.

---

**Q12: Why might it be "easier to interpret audio by its spectrum" rather than just by the raw waveform?**
**Answer:**
Because the spectrum highlights the **frequency content** of the sound, making it easier to identify pitches, harmonics, or noise components that aren't immediately obvious in the time-domain waveform.

---

**5. Filtering & Noise Removal**

**Q13: What is the purpose of applying a bandpass filter to an audio signal?**
**Answer:**
A bandpass filter **passes** only the frequencies within a specified range (band) and **attenuates** frequencies outside that range. This can help isolate certain sounds or reduce unwanted noise.

---

**Q14: Which Python libraries or functions might you use to implement a bandpass filter in this lab?**
**Answer:**

- **SciPy** (e.g., scipy.signal.butter, scipy.signal.filtfilt)
- The provided sample code for PyAudio or sounddevice, which may include a filtering section using SciPy's signal-processing tools.

---

**6. Feature Extraction (Spectrogram, Chromogram, Mel-Spectrogram, MFCC)**

**Q15: Which library is commonly used for advanced audio feature extraction (e.g., spectrogram, MFCC) in Python?**
**Answer:**
**Librosa** (installed with pip install librosa).

---

**Q16: What is a Spectrogram, and how is it generated?**
**Answer:**

A Spectrogram is a **time-frequency** representation of a signal, typically showing time on the x-axis, frequency on the y-axis, and amplitude (or power) as color intensity. It's generated by computing the Short-Time Fourier Transform (STFT) over successive time windows.

---

**Q17: Explain the difference between a standard Spectrogram and a Mel-Spectrogram.**
**Answer:**

- **Standard Spectrogram:** Plots frequency linearly (Hz) against time.
- **Mel-Spectrogram:** Converts frequencies to the **Mel scale**, which is more aligned with human auditory perception, and often uses a **logarithmic** or **dB** scale for amplitude.

---

**Q18: What are Mel Frequency Cepstral Coefficients (MFCCs), and why are they useful?**
**Answer:**
MFCCs are a representation of the short-term power spectrum of a sound, mapped onto the Mel scale to mimic human hearing. They are widely used in **speech recognition** and **audio classification** because they effectively capture the **timbre** or "color" of the sound.

---

**Q19: What is a Chromagram or Chroma Feature?**
**Answer:**
A Chromagram represents the intensity of each **pitch class** (e.g., the 12 semitones in Western music) over time. It's useful for analyzing musical content where notes can be mapped into these pitch classes.

---

**7. Advanced Sound Analytics (Speech Recognition)**

**Q20: Which commands install CMUSphinx and Google Speech Recognition dependencies on Raspberry Pi?**
**Answer:**

- sudo apt-get install flac
- pip install pocketsphinx
- pip install SpeechRecognition

---

**Q21: What is the difference between an offline speech recognition model (e.g., CMUSphinx) and a cloud-based API (e.g., Google Speech Recognition)?**
**Answer:**

- **Offline model (CMUSphinx):** Runs locally on the device without internet; generally **faster** for short queries but less accurate.
- **Cloud-based model (Google):** Requires internet; typically **more accurate** but introduces network latency and relies on external servers.

---

**Q22: How can you handle ambient noise when recording audio for speech recognition on the Raspberry Pi?**
**Answer:**
By capturing a short initial segment of ambient noise and using that to calibrate the input (e.g., adjust_for_ambient_noise in SpeechRecognition library), thereby reducing false triggers from background sound.

---

**8. Practical Applications & Further Exploration**

**Q23: Name two potential applications of an audio listening system built on a Raspberry Pi.**
**Answer:**

1. **Voice-enabled services** (e.g., a personal assistant).
2. **Healthcare** (e.g., digital stethoscope, lung sound analysis).
3. **Audio chatbot** or IVR system.
   *(Any two of these or other valid examples.)*

---

**Q24: Give an example of a 'wake word' system and why it might be useful.**
**Answer:**
Examples include **"Ok Google," "Alexa," or "Hey Siri."** These keywords trigger a device to start listening for commands, reducing continuous processing and preventing accidental activations.

---

**Q25: List one way you might improve the accuracy of the speech recognition code on the Raspberry Pi.**
**Answer:**

- Use a **higher-quality microphone**.
- Implement **noise reduction** or a more robust filtering process.
- Increase **training data** or adopt a more advanced model.
- **Calibrate** ambient noise before capturing speech.

---

**9. Additional Technical & Conceptual Questions**

**Q26: Why might you choose sounddevice over pyaudio (or vice versa)?**
**Answer:**

- **sounddevice** can be more straightforward for certain streaming tasks, is well-documented, and integrates nicely with NumPy.
- **PyAudio** is very common, widely supported, and often used in cross-platform audio projects.
  Choice often depends on user preference and specific library compatibility.

---

**Q27: How would you reduce the computational load if your Raspberry Pi struggles to process audio in real time?**
**Answer:**

- Lower the **sample rate** or **bit depth**.
- Use a **shorter FFT size** for spectrograms.
- Reduce **plotting frequency** or use simpler visualization.
- Move advanced computations (e.g., ML tasks) to a remote server if needed.

---

**Q28: In what scenario might you want to visualize a chromogram rather than a standard spectrogram?**
**Answer:**
When analyzing **musical** content, chord progressions, or pitch classes. A chromagram emphasizes pitch classes (like A, B, C, etc.) rather than purely linear frequency.

## Lab 2 Potential Questions for Code

**11. What does the parameter BUFFER = 1024 * 16 specify in the sample code?**

**Answer:**
It sets the number of audio samples captured per frame or chunk during each iteration. A higher buffer size results in larger chunks of audio being processed at a time, which may reduce CPU load but increases latency.

---

**12. What is the purpose of FORMAT = pyaudio.paInt16 in the PyAudio code?**

**Answer:**
It sets the audio format to 16-bit signed integers. This defines how each audio sample is represented in memory and matches the expected format from many microphones and audio systems.

---

**13. Why is CHANNELS = 1 used in both scripts?**

**Answer:**
It specifies that the recording should be in mono (single audio channel). This is sufficient for most basic audio processing tasks and uses less memory and processing power than stereo (2 channels).

---

**14. What is the role of RATE = 44100 in both sample codes?**

**Answer:**
It defines the sample rate, which is the number of audio samples captured per second. A sample rate of 44100 Hz is CD-quality and commonly used in digital audio.

---

**15. In both sample codes, why are matplotlib figures created with two subplots?**

**Answer:**
The first subplot visualizes the **raw waveform** captured from the microphone, and the second subplot displays the **filtered waveform** after applying a bandpass filter. This allows a real-time visual comparison of the original and processed signals.

---

**16. What is the purpose of using plt.show(block=False)?**

**Answer:**
This displays the plot window **without blocking** the execution of the rest of the code. It allows the plot to remain open and be updated in real time while the recording loop continues running.

---

**17. What does the design_filter() function do in both examples?**

**Answer:**
It creates a **bandpass filter** using the SciPy butter() function. The filter allows only frequencies between lowfreq and highfreq to pass and blocks others. It is returned in second-order sections (SOS) format for stability and performance.

---

**18. What is the meaning of the parameters 19400 and 19600 in the filter design?**

**Answer:**
These values define the **lower and upper cutoff frequencies** of the bandpass filter. Only frequencies between 19.4 kHz and 19.6 kHz will be passed, making the filter useful for isolating high-frequency tones or signals in that narrow range.

---

**19. Why is sosfilt(sos, data_int) used?**

**Answer:**
The sosfilt() function applies the bandpass filter (designed using design_filter) to the audio data. It processes the audio signal and outputs a filtered version, which is then used for visualization or further analysis.

---

**20. What does struct.unpack(str(BUFFER) + 'h', data) do in the PyAudio code?**

**Answer:**
It converts the **binary audio stream** (captured by PyAudio) into a NumPy-readable array of 16-bit signed integers (int16). This conversion is necessary because PyAudio reads raw bytes from the microphone, which need to be decoded into numerical values for processing.

---

**21. In the SoundDevice code, what does np.squeeze(data) do?**

**Answer:**
It removes any single-dimensional entries from the array shape. Since sd.rec() returns a

2D array with shape (BUFFER, 1) when using one channel, squeeze() reduces it to a 1D array of shape (BUFFER,) for easier processing and plotting.

---

**22. Why is blocking=True used in the SoundDevice code?**

**Answer:**
Setting blocking=True ensures that the sd.rec() function waits until the entire audio frame is recorded before proceeding. This guarantees synchronization between data capture and processing.

---

**23. What is the purpose of recording execution time with time.time()?**

**Answer:**
It measures how long each frame of audio takes to process (e.g., for filtering). This allows you to calculate the **average execution time per frame**, which helps evaluate whether your system can perform processing in **real time**.

---

**24. What does the command fig.canvas.draw() do?**

**Answer:**
It **redraws** the updated plot with the new audio data. This is necessary to visualize the real-time changes in waveform and filtered output during live audio recording.

---

**25. Why is real-time filtering useful in audio analytics?**

**Answer:**
Real-time filtering allows the user to **isolate frequencies of interest**, remove noise, and immediately observe how the signal changes. This is critical in applications such as voice recognition, noise cancellation, and detecting specific sound events.

---

**26. What does the second subplot titled "FILTERED" typically show?**

**Answer:**
It displays the output of the **bandpass filter**, showing only the audio signal components within the specified frequency range. This helps the user see how the filter removed unwanted frequencies from the original signal.

---

**27. If the filtered signal still contains noise, what are some ways to improve the filtering?**

**Answer:**

- Adjust the **cutoff frequencies** in the design_filter() function.
- Increase the **filter order** to make the filter steeper.
- Apply **multiple filters** in series or combine with noise-reduction techniques like spectral gating.