**Samples**

* In digital audio, **samples** are discrete data points that represent the amplitude of the sound wave at a specific moment in time. The number of samples per second is determined by the **sample rate** (e.g., 8000 Hz, meaning 8000 samples per second).

**In the project:**

* + You define the **sample rate** as 8000 Hz, meaning you have 8000 samples per second of audio.
  + For a 10-minute audio file, the total number of samples would be: Total Samples=10×60×8000=4,800,000 samples\text{Total Samples} = 10 \times 60 \times 8000 = 4,800,000 \, \text{samples}Total Samples=10×60×8000=4,800,000samples

**2. Frames**

* A **frame** is a short segment of audio data made up of a fixed number of samples. Frames are used to divide audio into smaller pieces for easier processing. Rather than processing an entire audio file at once, which can be computationally expensive, the audio is processed frame by frame.

**In the project:**

* + You are splitting the audio into **frames** with a **frame length** of 8064 samples, meaning each frame contains **8064** samples of the audio signal.
  + The code works on individual frames to process smaller portions of audio (like blending noise, computing spectrograms, etc.).

**3. Frame Length**

* The **frame length** is the number of samples contained within each frame. This determines how much time (in seconds) each frame represents, based on the **sample rate**.

**In the project:**

* + Your **frame length** is set to 8064 samples.
  + Since the **sample rate** is 8000 Hz (8000 samples per second), each frame represents: **Frame Duration=80648000≈1.008 seconds**\{Frame Duration} = \frac{8064}{8000} \approx 1.008 \, \text{seconds}Frame Duration=80008064​≈1.008seconds
  + So, each frame represents a little over **1 second** of audio.

**4. How They Are Linked**

* **Audio samples** are the raw data representing the sound wave. In the project, you're working with noisy and clean voice audio files, each of which is made up of thousands or millions of samples.
* These **samples** are divided into **frames**, which are chunks of audio containing 8064 samples each.
* The **frame length** determines how many samples are in each frame, and therefore how much time each frame covers.

**5. Usage of Frames in Your Project**

In your code, frames are used for several purposes:

* **Creating Training Data**: In **data creation mode**, the audio (both noise and clean voice) is split into frames to blend the noisy audio. Frames are processed in smaller chunks rather than as one large file.
* **Blending Noise**: When creating noisy versions of the clean audio, each **frame** of the clean voice is blended with noise (also in frames), creating a mixed noisy audio that can be used for training the model.
* **Saving and Processing**: The frames are reshaped back into their original form and saved as audio files (e.g., noisy\_voice\_long.wav, voice\_long.wav, noise\_long.wav) or spectrograms.

### ****Understanding Digital Audio Representation****:

Digital audio files like .wav or .mp3 contain sampled audio data, where the sound wave's **amplitude** (loudness) is represented by a series of discrete values (samples). These amplitude values are stored as **integers** in the file, depending on the **bit depth**:

* **16-bit audio**: The most common format in .wav files, stores amplitude values as integers between -32768 and 32767.
* **24-bit or 32-bit audio**: These formats store larger integer ranges for higher resolution (e.g., -8,388,608 to 8,388,607 for 24-bit audio).

These integer values represent the instantaneous loudness of the audio signal at discrete points in time.

### 2. ****Amplitude Normalization****:

When the audio file is loaded into Python using a library like librosa, these integer values are converted into floating-point values. The reason you see values between -1 and 1 in the resulting NumPy array is that the audio data is **normalized**. Normalization scales the integer amplitude values to a floating-point range to make further processing easier and consistent, regardless of the original bit depth.

#### The formula for normalization:

For **16-bit audio**, the normalization is done as follows:

Amplitudenormalized=Amplituderaw215−1\text{Amplitude}\_{\text{normalized}} = \frac{\text{Amplitude}\_{\text{raw}}}{2^{15} - 1}Amplitudenormalized​=215−1Amplituderaw​​

* **Amplitude\_raw**: The original integer amplitude value from the audio file.
* **2^{15} - 1 = 32767**: This is the maximum possible amplitude value for 16-bit audio. The division maps the original value to the range [-1, 1].

For **24-bit audio**:

Amplitudenormalized=Amplituderaw223−1\text{Amplitude}\_{\text{normalized}} = \frac{\text{Amplitude}\_{\text{raw}}}{2^{23} - 1}Amplitudenormalized​=223−1Amplituderaw​​

This scales the amplitude values to the same floating-point range between -1 and 1.

#### Example:

If a 16-bit audio sample has the raw value of 16383 (which is half of the maximum possible 16-bit value 32767), its normalized value would be:

1638332767≈0.5\frac{16383}{32767} \approx 0.53276716383​≈0.5

Similarly, a raw value of -32767 (the minimum possible value) would become:

−3276732767=−1\frac{-32767}{32767} = -132767−32767​=−1

Refer This Link for better understanding  
https://chatgpt.com/share/66f2bb44-1d28-8008-8d20-dfbcb01cf5c4