Manipulando áudio com **Pydub**

Live de Python #248

Roteiro

1. Pydub

Introdução as funcionalidades e segmentos de áudio

2. Efeitos

Aplicação de efeitos ao áudio

3. Detecção de silêncio

Detecção, cortes, remoção e split de silêncios

4. Geradores de sinal

Ondas primordiais: Senoides, Dentes de serra, ruído branco e etc...

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Ademar Peixoto, Adilson Herculano, Adriano Ferraz, Alemao, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alfredo Braga, Alisson Souza, Alysson Oliveira, Andre Azevedo, Andre Mesquita, Andre Paula, Aquiles Coutinho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Ayr-ton, Bernardo At, Bernardo Fontes, Bruno Almeida, Bruno Barcellos, Bruno Freitas, Bruno Lopes, Bruno Ramos, Caio Nascimento, Carlos Ramos, Christian Semke, Cristian Firmino, Damianth, Daniel C.s, Daniel Freitas, Daniel Real, Daniel Wojcickoski, Danilo Boas, Danilo Segura, Danilo Silva, David Couto, David Kwast, Davi Souza, Dead Milkman, Delton Porfiro, Denis Bernardo, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino, Edgar, Edilson Ribeiro, Eduardo Silveira, Eduardo Tolmasquim, Emerson Rafael, Erick Andrade, Érico Andrei, Everton Silva, Fabiano, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Correa, Fábio Thomaz, Fabricio Biazzotto, Fabricio Patrocinio, Felipe Augusto, Fernanda Prado, Fernando Celmer, Flávio Meira, Francisco Neto, Francisco Silvério, Gabriel Espindola, Gabriel Mizuno, Gabriel Moreira, Gabriel Paiva, Gabriel Ramos, Gabriel Simonetto, Geandreson Costa, Gilmar José, Giovanna Teodoro, Giuliano Silva, Guilherme Beira, Guilherme Felitti, Guilherme Gall, Guilherme Piccioni, Guilherme Silva, Guionardo Furlan, Gustavo Suto, Haelmo, Haelmo Almeida, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Henrique Andrade, Henrique Domciano, Higor Monteiro, Italo Silva, Janael Pinheiro, Jean Victor, Jefferson Antunes, Joelson Sartori, Jonatas Leon, Jônatas Oliveira, Jônatas Silva, Jorge Silva, José Gomes, Joseíto Júnior, Jose Mazolini, Josir Gomes, Juan Felipe, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Júlio Sanchez, Julio Silva, Kaio Peixoto, Kálita Lima, Leandro Silva, Leandro Vieira, Leonardo Mello, Leonardo Nazareth, Lucas Carderelli, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Pavelski, Lucas Schneider, Lucas Simon, Luciano Filho, Luciano Ratamero, Luciano Teixeira, Luis Eduardo, Luiz Carlos, Luiz Duarte, Luiz Lima, Luiz Paula, Mackilem Laan, Marcelo Araujo, Marcelo Campos, Marcio Moises, Marco Mello, Marcos Gomes, Marina Passos, Mario Henrique, Mateus Lisboa, Mateus Ribeiro, Mateus Silva, Matheus Mendez, Matheus Silva, Matheus Vian, Mírian Batista, Mlevi Lsantos, Murilo Carvalho, Murilo Viana, Natan Cervinski, Nathan Branco, Ocimar Zolin, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Pedro Gomes, Pedro Henrique, Pedro Pereira, Peterson Santos, Pytonyc, Rafael Faccio, Rafael Lopes, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Regis Santos, Renato José, Renato Oliveira, Renê Barbosa, Rene Bastos, Rene Pessoto, Ricardo Silva, Riverfount, Rjribeiro, Robson Maciel, Rodrigo Barretos, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Vaccari, Rodrigo Vieira, Rui Jr, Samanta Cicilia, Samuel Santos, Selmison Miranda, Téo Calvo, Thiago Araujo, Thiago Borges, Thiago Curvelo, Thiago Souza, Tony Dias, Tyrone Damasceno, Valdir, Valdir Tegon, Vcwild, Vinícius Costa, Vinicius Stein, Walter Reis, William Sampaio, Willian Lopes, Wilson Duarte, Yros Aguiar, Zeca Figueiredo

Obrigado você

Pydub

Uma introdução

Pydub

Pydub é uma biblioteca focada em manipulação de arquivo de áudio.

- Criada por James Robert (@jiaaro) em 2011
- Atualmente na versão 0.25 (lançada em março de 2021)
- Licença: **MIT**
- Focada em uma API simples de alto nível

Seu lema é: "**permitir que você faça coisas com áudio de uma forma que não seja estúpida**."

Um tema subjetivo

Sei que para algumas pessoas o assunto "manipular áudio" pode parecer fora da realidade. Então, aqui vão alguns exemplos do que você pode fazer com pydub:

- **Converter arquivos de áudio**: .wav -> .mp3
- **Cortar áudio**: "5 segundos finais de uma música"
- **Efeitos de transição**: fade-in, fade-out, cossfade, ...
- **Juntar arquivos de áudio**:
  - Juntar dois áudios de 1 minuto, tendo 2 minutos de áudio
  - Juntar camadas: Dois áudios tocando ao mesmo tempo
- **Separar canais de áudio**: Stereo em dois canais mono

Mais exemplos do que você pode fazer

- **Detecção de silêncios**: "do segundo 2 ai 4.3 não tem áudio"
- **Manipulação do som**: Aumentar volume, abaixar volume, ...
- **Obter metadados sobre o som**: dBFS, duração, ...
- **Tocar áudio**: pydub.playback
- **Processamento de sinais**: Compressão, EQ, ...
- **Geração de ondas primordiais**: senoidal, Dente de serra, ruído branco...

O que o pydub **não faz!**

- **Aplicar efeitos de plugins** (VST, LV2, AU): pedalboard
- **Análise de frequências e plot**: Librosa
- **IA**: torchaudio
- **Separação de instrumento**: spleeter
- **Transcrição de áudio** (STT): whisper
- **Criação de áudio por texto** (TTS): Coqui
- **Metadados de faixas** (artinas, disco, ...): eyed3

pip install pydub

Instalação

Começando do zero!

O básico

Seguimentos [exemplo_00.py]

O pydub usa um objeto primordial para tudo. Chamado **AudioSegment**. Você pode importar qualquer formato de som com os métodos **from_*formato*** ou então usando o método `**from_file(file='???', format='???')**

```
1  from pydub import AudioSegment
2
3  music_wav = AudioSegment.from_wav('assets/music.wav')
4  music_mp3 = AudioSegment.from_mp3('assets/music.mp3')
5
6  music_mp3 = AudioSegment.from_file('assets/music.mp3', format='mp3')
```

AudioSegment[0] [exemplo_01.py]

Os operadores têm funções simples e fazem com que o código seja expressivo

```python
from pydub import AudioSegment

bass = AudioSegment.from_wav('assets/bass.wav')
pads = AudioSegment.from_wav('assets/pads.wav')

bass + 3  # Aumenta o volume em 3 dB
bass - 3  # Diminui o volume em 3 dB

bass + pads  # Adiciona os pads após o som do baixo
pads + bass  # Adiciona o baixo após o som dos pads

bass * 2  # Dobra a duração do audio, contatena o final no início
```

AudioSegment[1] [exemplo_02.py]

O slice tem um papel fundamental na biblioteca para fatiar o som:

```python
from pydub import AudioSegment

bass = AudioSegment.from_wav('assets/bass.wav')

# Pega os primeiros cinco segundos do audio
bass[1_000 * 5:]
# Pega após os primeiros cinco segundos
bass[:1_000 * 5]
# Pega entre 5 e 10 segundos
bass[1_000 * 5::1_000 * 10]
```

Tocar áudio [exemplo_03.py]

Embora seja uma funcionalidade focada em debug, temos como executar seguimentos de áudio:

```
1   from pydub import AudioSegment
2   from pydub.playback import play
3
4   music = AudioSegment.from_wav('assets/music.wav')
5   play(music)
```

Tocar áudio

Embora seja uma funcionalidade focada em debug, temos como executar seguimentos de áudio:

```
1   from pydub import A
2   from pydub.playback
3
4   music = AudioSeg    nt.from_wav('assets/music.wav')
5   play(music)
```

Qualquer seguimento pode ser executado pelo play

Export [exemplo_04.py]

O pydub, além de obter a entrada de arquivos de áudio, também exporta os seguimentos para áudio

```
1   from pydub import AudioSegment
2
3   bass = AudioSegment.from_wav('assets/bass.wav')
4
5   bass[1_000 * 5::1_000 * 10].export('bass_5_to_10.mp3')
```

Imutabilidade [exemplo_05.py]

A ideia dos seguimentos de áudio é que toda operação gere um novo seguimento

```python
from pydub import AudioSegment

audioseg = AudioSegment.empty()
music = AudioSegment.from_wav('assets/music.wav')

new_seg = audioseg + music

new_seg.export('test.mp3')
```

Metadados [exemplo_06.py]

Para obter algumas informações sobre o arquivo, como seu volume médio, seu tempo de duração, você pode solicitar essas informações para o seguimento

```
1   seg = AudioSegment.from_wav('assets/music.wav')
2
3   # Informações gerais
4   seg.duration_seconds  # Duração
5   seg.raw_data  # O Array
6   seg.channels  # Quantidade de canais
7   seg.frame_rate  # framerate
8
9   # Informações sobre amplitude
10  seg.rms       # Loudness
11  seg.dBFS      # Loudness logarítimo
12  seg.max_dBFS  # Maior amplitude em dB
13  seg.max  # Maior amplitude
```

Metadados [exemplo_06.py]

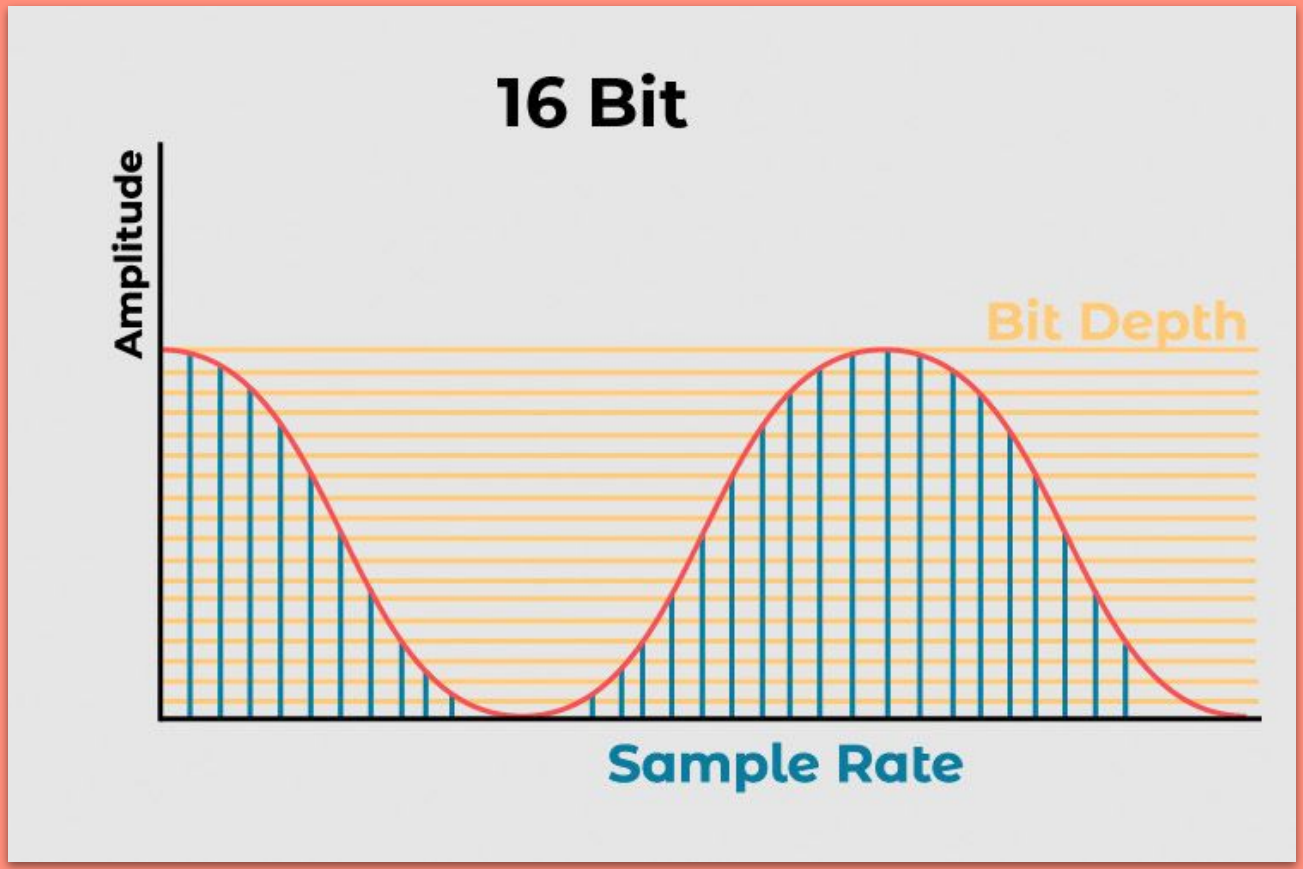
Para obter algumas informações sobre o arquivo, como seu volume médio, seu tempo de duração, você pode solicitar essas informações para o seguimento

```python
seg = AudioSegment.from_wav('assets/music.wav')

# Informações gerais
seg.duration_seconds  # Duração
seg.raw_data  # O Array
seg.channels  # Quantidade de canais
seg.frame_rate  # framerate

# I
seg
seg
seg
seg
```

```python
# Informações sobre frames
seg.frame_count  # Número de frames
seg.frame_width  # Número de bytes em frames
seg.sample_width  # Número de bytes por sample
```

Metadados

Para obter algumas informações sobre o arquivo, como seu volume médio, seu tempo de duração, você pode solicitar essas informações para o seguimento



https://woodandfirestudio.com/en/sample-rate-bit-depth/

Junção de áudio [exemplo_07.py]

Existem diversas formas de unir segmentos:

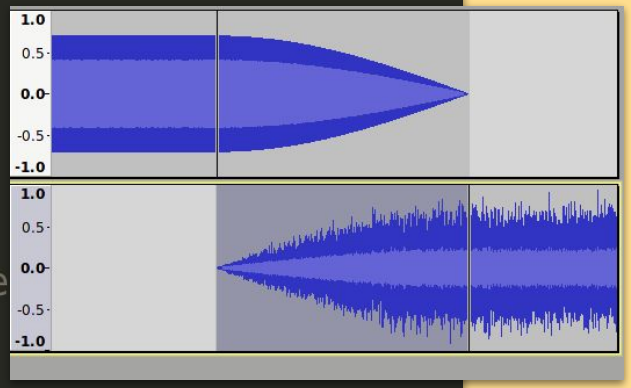
```python
from pydub import AudioSegment

seg_a = AudioSegment.from_wav('assets/bas.wav')
seg_b = AudioSegment.from_wav('assets/drums.wav')

# Adiciona um ao final do outro
seg_c = seg_a + seg_b

# Adiciona um ao final do outro com crossfade
seg_c = seg_a.append(seg_b)  # 0.1 seg
seg_c = seg_a.append(seg_b, crossfade=5000) # 5 seg de fade

# Sobrepõe os dois áudios
seg_c = seg_a.overlay(seg_b)
```

Junção de áudio [exemplo_07.py]

Existem diversas formas de unir segmentos:

```python
from pydub import AudioSegment

seg_a = AudioSegment.from_wav('assets/bas.wav')
seg_b = AudioSegment.from_wav('assets/drums.wav')

# Adiciona um ao final do outro
seg_c = seg_a + seg_b

# Adiciona um ao final do outro com crossfade
seg_c = seg_a.append(seg_b)  # 0.1 seg
seg_c = seg_a.append(seg_b, crossfade=5000) # 5 seg de fade

# Sobrepõe os dois áudios
seg_c = seg_a.overlay(seg_b)
```

Crossfade



```python
from pydub import AudioSegment
from pydub.playback import play

bass = AudioSegment.from_wav('assets/bass.wav')
pads = AudioSegment.from_wav('assets/pads.wav')
drums = AudioSegment.from_wav('assets/drums.wav')
voz = AudioSegment.from_wav('assets/abertura.wav')

play(
  bass
  .overlay(pads)
  .overlay(drums - 3)[:10_000]
  .append(voz[:5_000], crossfade=3_000)
)
```

Efeitos

Aplicação de efeitos de áudio.

O básico

Existem diversos efeitos que podem ser usados em seguimentos de forma básica:

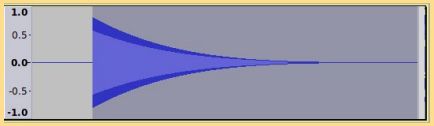
- **fade**(): Aumenta ou abaixa o som progressivamente
- **fade_in**: Aumenta o som progressivamente no início
- **fade_out**: Abaixa o som progressivamente no final
- **reverse**()`: Inverte a frequência do som

```
1  from pydub import AudioSegment
2
3  music_wav = AudioSegment.from_wav('assets/music.wav')
4  music_wav.fade_in().fade_out().reverse()
```

O básico [exemplo_08.py]

Existem diversos efeitos que podem ser usados em seguimentos de forma básica:

- **fade**(): Aumenta ou abaixa o som progressivamente
- **fade_in**: Aumenta o som progressivamente no início
- **fade_out**: Abaixa o som progressivamente no final
- **reverse**()`: Inverte a frequência do som



Fade out

```
1  from pydub import AudioSegment
2
3  music_wav = AudioSegment.from_wav('assets/music.wav')
4  music_wav.fade_in().fade_out().reverse()
```

Efeitos

As funções de efeito são divididas em dois módulos, os oficiais do pydub **pydub.effects** e os efeitos criados com scipy **pydub.scipy_effects**. Para começar vamos ver os efeitos do módulo **effects**:

- **normalize**: Normaliza o sinal para se aproximar de um valor
- **speedup**: Acelera o som em X vezes
- **strip_silence**: Remove os silêncios nas bordas
- **compress_dynamic_range**: Compressão dinâmica
- **low_pass_filter**: Foca em passar somente frequências graves
- **high_pass_filter**: Foca em passar somente frequências agudas
- **pan**: Efeito em stereo
- **apply_gain_stereo:** Aplica o ganho em um só lado

Exemplo do uso de efeitos [exemplo_09.py]

```python
from pydub import AudioSegment
from pydub.playback import play
from pydub.effects import normalize, low_pass_filter

bass = AudioSegment.from_wav('assets/bass.wav')
pads = AudioSegment.from_wav('assets/pads.wav')
drums = AudioSegment.from_wav('assets/drums.wav')
voz = normalize(AudioSegment.from_wav('assets/abertura.wav'))

music = bass.overlay(pads).overlay(drums - 3)

music_bass = low_pass_filter(music, 500)[5_000: 12_000]

play(music_striped.append(voz[:5_000], crossfade=3_000))
```

Efeitos com scipy

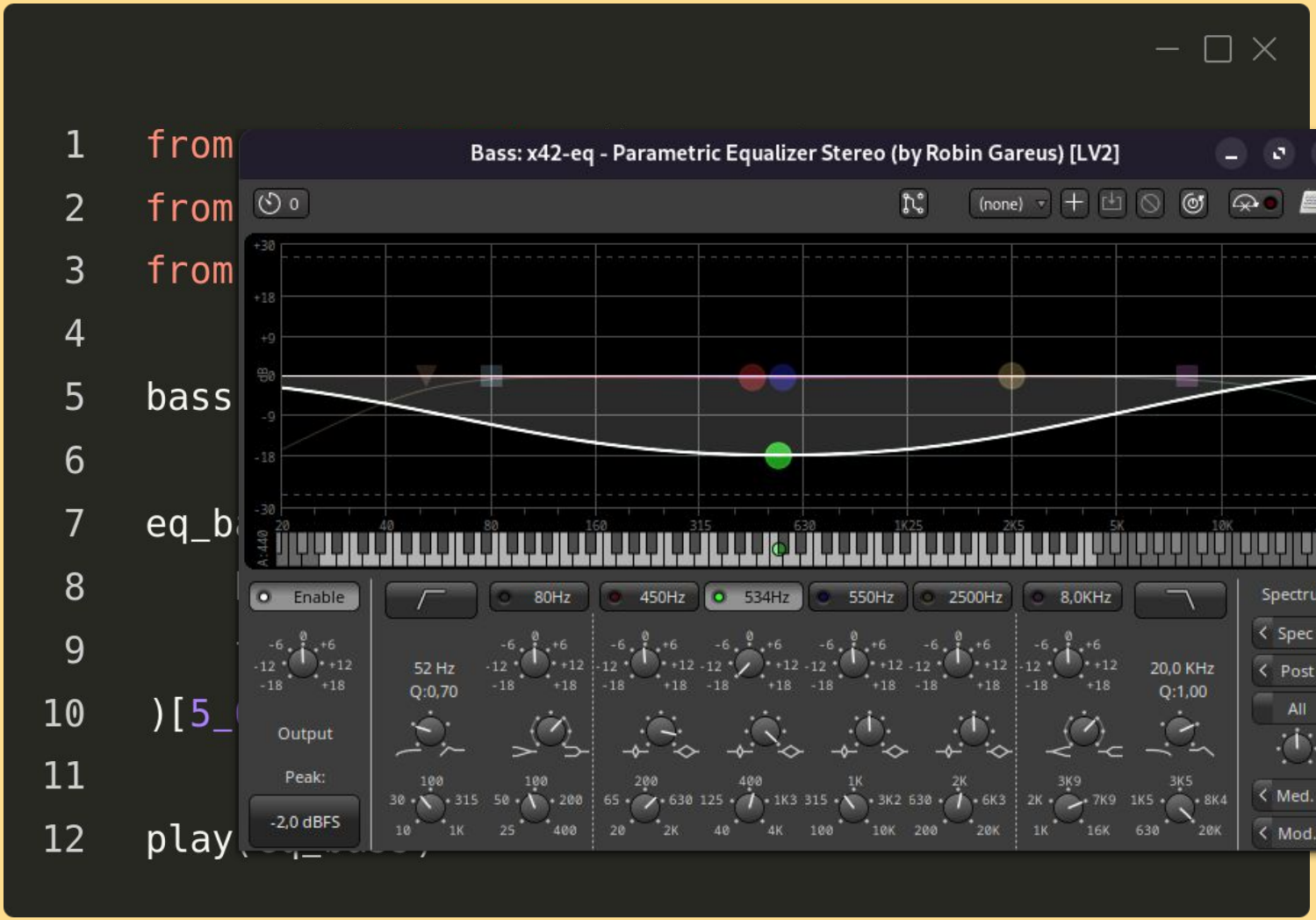
Para usar os efeitos do scipy, você deve instalar o **scipy**:

- **band_pass_filter**: Equalizador de duas bandas
- **eq**: Equalizador por frequência
- **low_pass_filter**: Foca em passar somente frequências graves
- **high_pass_filter**: Foca em passar somente frequências agudas

Exemplo do uso de efeitos [exemplo_10.py]

```python
from pydub import AudioSegment
from pydub.scipy_effects import eq
from pydub.playback import play

bass = AudioSegment.from_wav('assets/bass.wav')

eq_bass = eq(
    bass, focus_freq=500, gain_dB=-18,
    filter_mode='peak', order=3
)[5_000:10_000]

play(eq_bass)
```

Exemplo do uso de efeitos [exemplo_10.py]



```
1   from
2   from
3   from
4
5   bass
6
7   eq_b
8
9
10  )[5_
11
12  play
```

Detecção de silêncio

Cortes, remoção e split de silêncios

Detecção de silêncio

Pydub conta com um módulo especializado em analisar silêncios em arquivos de áudio. Esse módulo conta com algumas funções interessantes:

- **detect_silence**: Detecta os momentos em que existem silêncios
- **detect_nonsilent**: O contrário
- **split_on_silence**: Cria uma lista de Seguimentos do som
- **detect_leading_silence**: Exibe o ms em que o silêncio inicial termina

Essas funções têm quase os mesmos parâmetros:

- **min_silence_len**: Tempo mínimo para ser considerado silêncio
- **silence_thresh**: Sinal mínimo para ser considerado silêncio (basicamente o volume)

```python
from pydub import AudioSegment, silence

audioseg = AudioSegment.from_wav('assets/music.wav')

silence.detect_silence(
    audioseg, min_silence_len=500, silence_thresh=-50
)  # [[0, 4718], [48206, 52330]]

silence.detect_nonsilent(
    audioseg, min_silence_len=500, silence_thresh=-50
)  # [[4718, 48206]]

silences = silence.split_on_silence(
    audioseg, min_silence_len=500, silence_thresh=-50
)  # [<pydub.audio_segment.AudioSegment object at 0x7fcaebbe3cd0>]

silences = silence.detect_leading_silence(
    audioseg, silence_threshold=-50
)  # 4710
```

Casos de uso [exemplo_12.py]

```python
def cut_silences(  # https://github.com/dunossauro/videomaker-helper
  audio_file, output_file, silence_time=400, threshold=-65,
) -> Path:
    audio = AudioSegment.from_file(audio_file)

    silences = silence.split_on_silence(
        audio, min_silence_len=silence_time, silence_thresh=threshold
    )

    combined = AudioSegment.empty()
    for chunk in silences:
        combined += chunk

    combined.export(output_file, format='mp3')

    return Path(output_file)
```

Geradores

Sinais primordiais

Geradores [exemplo_13.py]

O pydub conta com geradores de sinal (ondas) como:

- Senoidal
- Triangular
- Quadrada
- Dente de serra
- Pulse
- Ruído branco

```
1  from pydub.generators import Sawtooth,Triangle, Pulse
2  from pydub.playback import play
3
4  audioseg = Sawtooth(44_000).to_audio_segment()
5  audioseg = audioseg + Triangle(88_000).to_audio_segment()
6
7  play(audioseg)
```

Referências

- Código fonte do pydub: https://github.com/jiaaro/pydub
- Documentação da API do pydub: https://github.com/jiaaro/pydub/blob/master/API.markdown

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3