



# Parselmouth Documentation

*Release 0.4.3*

**Yannick Jadoul**

**Mar 19, 2024**



# GETTING STARTED

<b>1 Installation</b>	<b>3</b>
1.1 Basics . . . . .	3
1.2 Python distributions . . . . .	3
1.3 PsychoPy . . . . .	4
1.4 Troubleshooting . . . . .	5
<b>2 Examples</b>	<b>7</b>
2.1 Plotting . . . . .	7
2.2 Batch processing of files . . . . .	12
2.3 Pitch manipulation and Praat commands . . . . .	15
2.4 PsychoPy experiments . . . . .	17
2.5 Web service . . . . .	21
2.6 Projects using Parselmouth . . . . .	25
<b>3 API Reference</b>	<b>27</b>
<b>4 Citing Parselmouth</b>	<b>53</b>
<b>5 Indices and tables</b>	<b>55</b>
<b>Python Module Index</b>	<b>57</b>
<b>Index</b>	<b>59</b>





**Parselmouth** is a Python library for the [Praat](#) software.

Though other attempts have been made at porting functionality from Praat to Python, Parselmouth is unique in its aim to provide a complete and Pythonic interface to the internal Praat code. While other projects either wrap Praat's scripting language or reimplementing parts of Praat's functionality in Python, Parselmouth directly accesses Praat's C/C++ code (which means the algorithms and their output are exactly the same as in Praat) and provides efficient access to the program's data, but *also* provides an interface that looks no different from any other Python library.

Please note that Parselmouth is currently in premature state and in active development. While the amount of functionality that is currently present is not huge, more will be added over the next few months. As such, *feedback* and possibly *contributions* are highly appreciated.

Drop by our [Gitter chat room](#) or post a message to our [Google discussion group](#) if you have any question, remarks, or requests!

**Warning:** Parselmouth 0.4.0 is the *last version* supporting Python 2. Python 2 has reached End Of Life on January 1, 2020, and is officially not supported anymore: see <https://python3statement.org/>. Please [move to Python 3](#), to be able to keep using new Parselmouth functionality.



## INSTALLATION

### 1.1 Basics

Parselmouth can be installed like any other Python library, using (a recent version of) the Python package manager `pip`, on Linux, macOS, and Windows:

```
pip install praat-parselmouth
```

To update your installed version to the latest release, add `-U` (or `--upgrade`) to the command:

```
pip install -U praat-parselmouth
```

**Warning:** While the Python module itself is called `parselmouth`, the Parselmouth package on the Python Package Index has the name `praat-parselmouth`.

---

**Note:** To figure out if you can or should update, the version number of your current Parselmouth installation can be found in the `parselmouth.VERSION` variables. The version of Praat on which this version of Parselmouth is based and the release date of that Praat version are available as `PRAAT_VERSION` and `PRAAT_VERSION_DATE`, respectively.

---

### 1.2 Python distributions

#### Anaconda

If you use the Anaconda distribution of Python, you can use the same `pip` command in a terminal of the appropriate Anaconda environment, either activated through the [Anaconda Navigator](#) or `conda` tool.

#### Homebrew & MacPorts

We currently do not have Homebrew or MacPorts packages to install Parselmouth. Normally, Parselmouth can just be installed with the accompanying `pip` of these distributions.

#### PyPy

Binary wheels for recent versions of PyPy are available on the Python Package Index (PyPI), and can be installed with `pip`.

#### Other

For other distributions of Python, we are expecting that our package is compatible with the Python versions that are out there and that `pip` can handle the installation. If you are using yet another Python distribution, we are definitely interested in hearing about it, so that we can add it to this list!

## 1.3 PsychoPy

As a Python library, Parselmouth can be used in a PsychoPy experiment. There are two different ways in which PsychoPy can be installed: it can just be manually installed as a standard Python library, in which case Parselmouth can just be installed next to it with pip. For Windows and Mac OS X, however, *standalone* versions of PsychoPy exist, and the software does currently not allow for external libraries to be installed with pip.

To install Parselmouth in a standalone version of PsychoPy, the following script can be opened and run from within the PsychoPy Coder interface: `psychopy_installation.py`

---

**Note:** If running the script results in an error mentioning `TLSV1_ALERT_PROTOCOL_VERSION`, the version of PsychoPy/Python is too old and you will need to follow the manual instructions underneath.

---

Alternatively, you can follow these steps to manually install Parselmouth into a standalone version of PsychoPy:

0. Find out which version of Python PsychoPy is running.
  - To do so, you can run `import sys; print(sys.version_info)` in the *Shell* tab of the PsychoPy Coder interface. Remember the first two numbers of the version (major and minor; e.g., 3.6).
  - On *Windows*, also run `import platform; print(platform.architecture()[0])` and remember whether the Python executable's architecture is 32bit or 64bit.
1. Go to <https://pypi.org/project/praat-parselmouth/>.
2. Download the file `praat_parselmouth-x.y.z-cpVV-cpVVm-AA.whl` (*for Windows*) or `praat_parselmouth-x.y.z-cpVV-cpVVm-macosx_10_6_intel.whl` (*for Mac OS X*) - where:
  - *x.y.z* will be the version of Parselmouth you want to install
  - *VV* are the first two numbers of the Python version
  - For *Windows*, *AA* is `win32` if you have a 32bit architecture, and `win_amd64` for 64bit

Be sure to find the right file in the list, containing both the correct Python version, and `win32/win_amd64` (*Windows*) or `macosx` (*Mac OS X*) in its name!

3. Rename the downloaded file by replacing the `.whl` extension by `.zip`.
4. Extract this zip archive somewhere on your computer, in your directory of choice. Remember the name and location of the extracted folder that contains the file `parselmouth.pyd` (*Windows*) or `parselmouth.so` (*Mac OS X*).
5. Open PsychoPy, open the *Preferences* window, go to the *General* tab.
6. In the *General* tab of the PsychoPy *Preferences*, in the *paths* field, add the folder where you just extracted the Parselmouth library to the list, enclosing the path in quotemarks. (On *Windows*, also replace all \ characters by /.)
  - For example, if the list was empty ([]), you could make it look like `['C:/Users/Yannick/parselmouth-psychopy/']` or `['/Users/yannick/parselmouth-psychopy/']`.
  - On *Windows*, to find the right location to enter in the PsychoPy settings, right click `parselmouth.pyd`, choose *Properties*, and look at the *Location* field.
  - On *Mac OS X*, to find the right location to enter in the PsychoPy settings, right click `parselmouth.so`, choose *Get info*, and look at the *where* field.
  - On *Mac OS X*, dragging the folder into a terminal window will also give you the full path with slashes.
7. Click *Ok* to save the PsychoPy settings, close the *Preferences* window, and restart PsychoPy.

8. *Optional:* if you want to check if Parselmouth was installed correctly, open the PsychoPy Coder interface, open the *Shell* tab, and type `import parselmouth`.

- If this results in an error message, please let us know, and we'll try to help you fix what went wrong!
- If this does not give you an error, congratulations, you can now use Parselmouth in your PsychoPy Builder!

---

**Note:** These instructions were tested with the standalone versions 3.1.3 and 1.85.2 of PsychoPy. Things might have changed since then, so if running the script or following the manual steps results in an error, please do not hesitate to get in touch.

---

## 1.4 Troubleshooting

It is possible that you run into more problems when trying to install or use Parselmouth. Supporting all of the different Python versions out there is not an easy job, as there are plenty of different platforms and setups.

If you run into problems and these common solutions are not solving them, please drop by the [Gitter chat room](#), write a message in the [Google discussion group](#), create a [GitHub issue](#), or write [me](#) a quick email. We would be very happy to solve these problems, so that future users can avoid them!

### 1.4.1 Multiple Python versions

In case you have multiple installations of Python and don't know which `pip` belongs to which Python version (*looking at you, OS X*):

```
python -m pip install praat-parselmouth
```

Finding out the exact location of the `python` executable (to call the previous command) for a certain Python installation can be done by typing the following lines in your Python interpreter:

```
>>> import sys
>>> print(sys.executable)
```

If executing this in your Python shell would for example `print /usr/bin/python`, then you would run `/usr/bin/python -m pip install praat-parselmouth` in a terminal to install Parselmouth. (-U can again be added to update an already installation to the latest version.)

Combining these two approaches, you can install Parselmouth from within Python itself without knowing where that version of Python is installed:

```
>>> import sys, subprocess
>>> subprocess.call([sys.executable, '-m', 'pip', 'install', 'praat-parselmouth'])
```

Extra arguments to `pip` can be added by inserting them as strings into the list of arguments passed to `subprocess.call` (e.g., to update an existing installation of Parselmouth: [..., 'install', '-U', 'praat-parselmouth']).

## 1.4.2 Pip version

If the standard way to install Parselmouth results in an error or takes a long time, try updating pip to the latest version (as pip needs to be a reasonably recent version to install the binary, precompiled wheels) by running

```
pip install -U pip
```

If you do not have pip installed, you follow these instructions to install pip: <https://pip.pypa.io/en/stable/installing/>

## 1.4.3 ImportError: DLL load failed on Windows

Sometimes on Windows, the installation works, but importing Parselmouth fails with an error message saying `ImportError: DLL load failed: The specified module could not be found..`. This error is cause by some missing system files, but can luckily be solved quite easily by installing the “Microsoft Visual C++ Redistributable for Visual Studio 2017”.

The “Microsoft Visual C++ Redistributable for Visual Studio 2019” installer can be downloaded from [Microsoft’s website](#), listed under the “Other Tools and Frameworks” section. These are the direct download links to the relevant files:

- For a 64-bit Python installation: [https://aka.ms/vs/16/release/VC\\_redist.x64.exe](https://aka.ms/vs/16/release/VC_redist.x64.exe)
- For a 32-bit Python installation: [https://aka.ms/vs/16/release/VC\\_redist.x86.exe](https://aka.ms/vs/16/release/VC_redist.x86.exe)

To check which Python version you are using, you can look at the first line of output when starting a Python shell. The version information should contain `[MSC v.xxxx 64 bit (AMD64)]` in a 64-bit installation, or `[MSC v.xxxx 32 bit (Intel)]` in a 32-bit installation.

## EXAMPLES

Parselmouth can be used in various contexts to combine Praat functionality with standard Python features or other Python libraries. The following examples give an idea of the range of possibilities:

### 2.1 Plotting

Using Parselmouth, it is possible to use the existing Python plotting libraries – such as `Matplotlib` and `seaborn` – to make custom visualizations of the speech data and analysis results obtained by running Praat’s algorithms.

The following examples visualize an audio recording of someone saying “*The north wind and the sun [...]*”: `the_north_wind_and_the_sun.wav`, extracted from a [Wikipedia Commons audio file](#).

We start out by importing `parselmouth`, some common Python plotting libraries `matplotlib` and `seaborn`, and the `numpy` numeric library.

```
[1]: import parselmouth

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

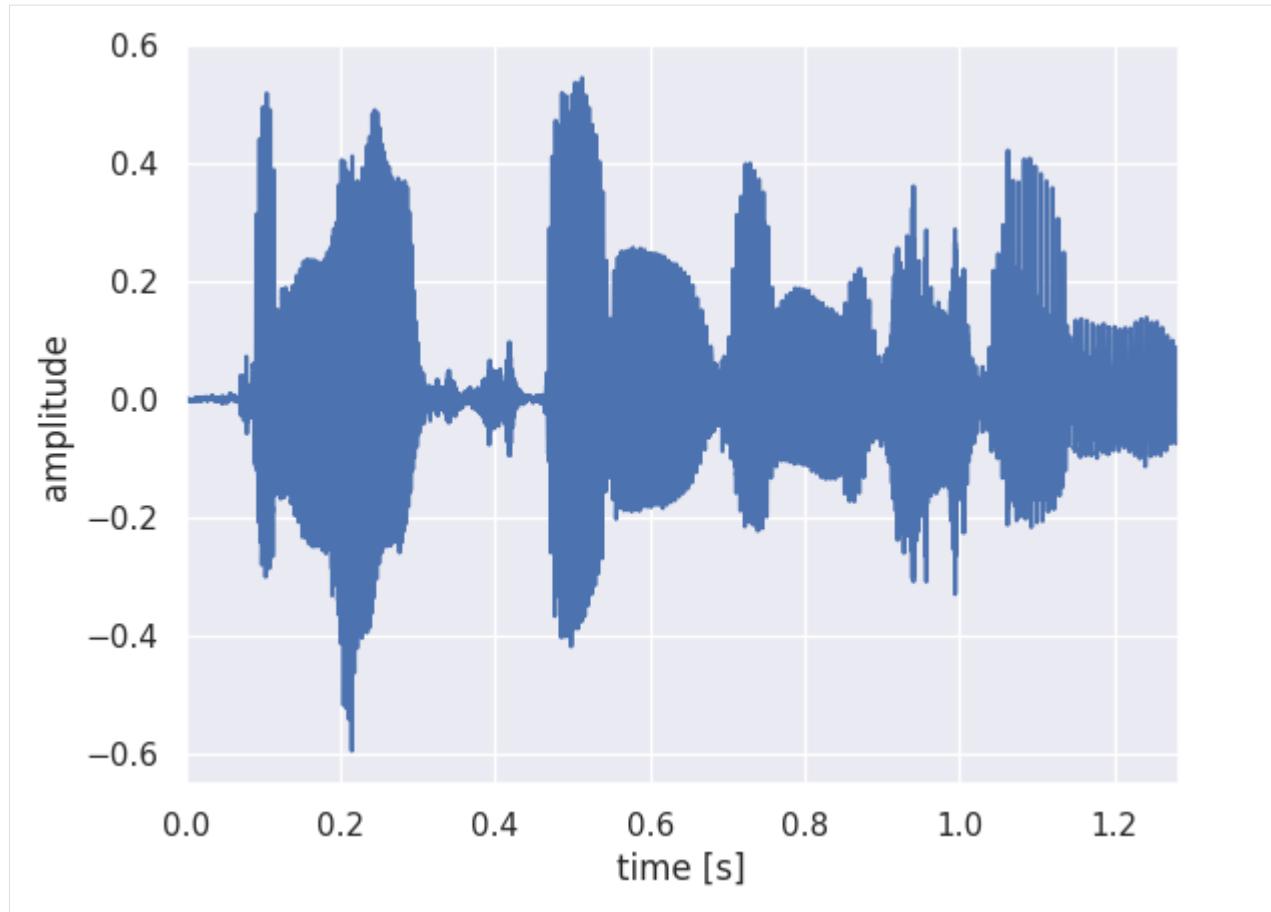
```
[2]: sns.set() # Use seaborn's default style to make attractive graphs
plt.rcParams['figure.dpi'] = 100 # Show nicely large images in this notebook
```

Once we have the necessary libraries for this example, we open and read in the audio file and plot the raw waveform.

```
[3]: snd = parselmouth.Sound("audio/the_north_wind_and_the_sun.wav")
```

`snd` is now a Parselmouth `Sound` object, and we can access its values and other properties to plot them with the common `matplotlib` Python library:

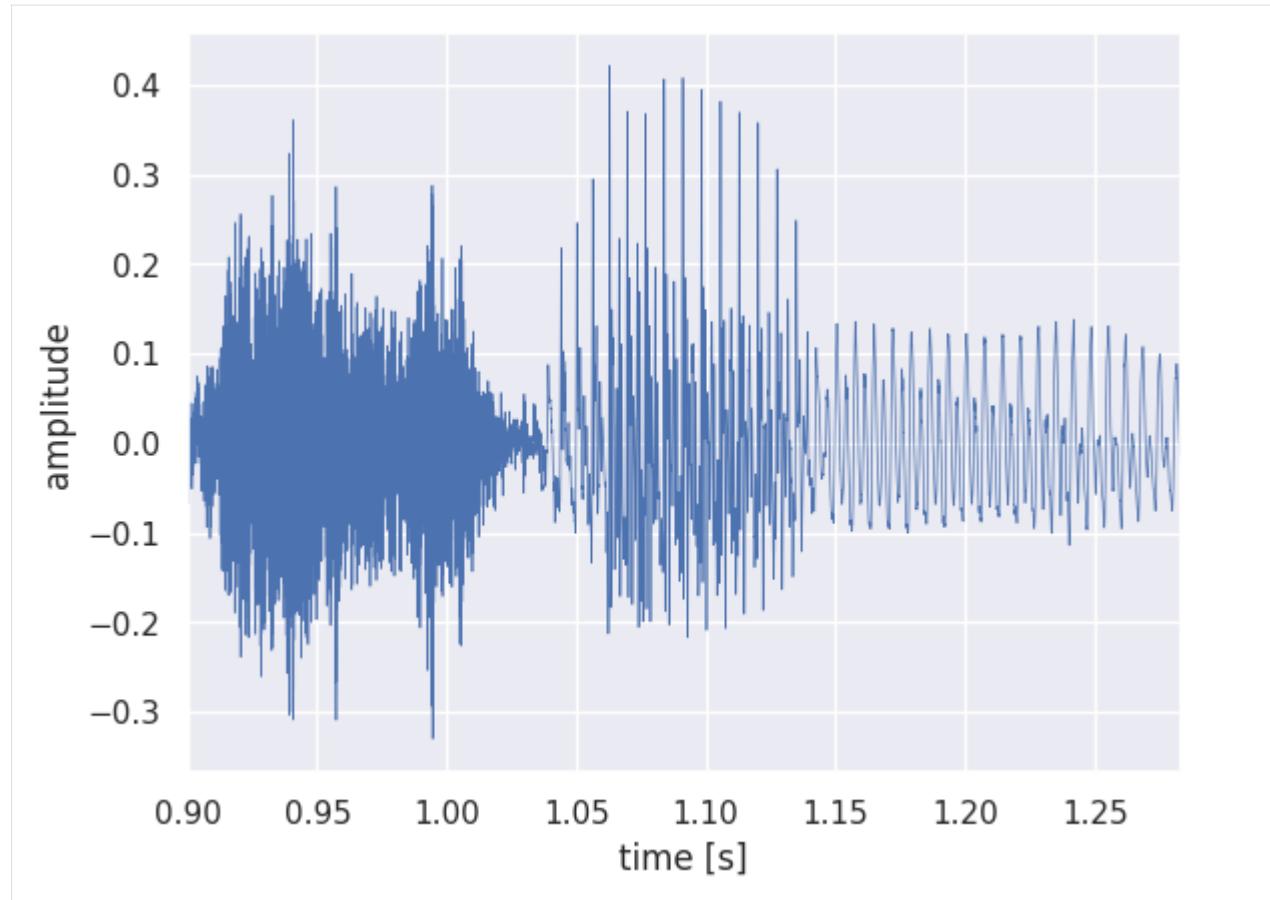
```
[4]: plt.figure()
plt.plot(snd.xs(), snd.values.T)
plt.xlim([snd.xmin, snd xmax])
plt.xlabel("time [s]")
plt.ylabel("amplitude")
plt.show() # or plt.savefig("sound.png"), or plt.savefig("sound.pdf")
```



It is also possible to extract part of the speech fragment and plot it separately. For example, let's extract the word “*sun*” and plot its waveform with a finer line.

```
[5]: snd_part = snd.extract_part(from_time=0.9, preserve_times=True)
```

```
[6]: plt.figure()
plt.plot(snd_part.xs(), snd_part.values.T, linewidth=0.5)
plt.xlim([snd_part.xmin, snd_part xmax])
plt.xlabel("time [s]")
plt.ylabel("amplitude")
plt.show()
```



Next, we can write a couple of ordinary Python functions to plot a Parselmouth Spectrogram and Intensity.

```
[7]: def draw_spectrogram(spectrogram, dynamic_range=70):
    X, Y = spectrogram.x_grid(), spectrogram.y_grid()
    sg_db = 10 * np.log10(spectrogram.values)
    plt.pcolormesh(X, Y, sg_db, vmin=sg_db.max() - dynamic_range, cmap='afmhot')
    plt.ylim([spectrogram.ymin, spectrogram.ymax])
    plt.xlabel("time [s]")
    plt.ylabel("frequency [Hz]")

def draw_intensity(intensity):
    plt.plot(intensity.xs(), intensity.values.T, linewidth=3, color='w')
    plt.plot(intensity.xs(), intensity.values.T, linewidth=1)
    plt.grid(False)
    plt.ylim(0)
    plt.ylabel("intensity [dB]")
```

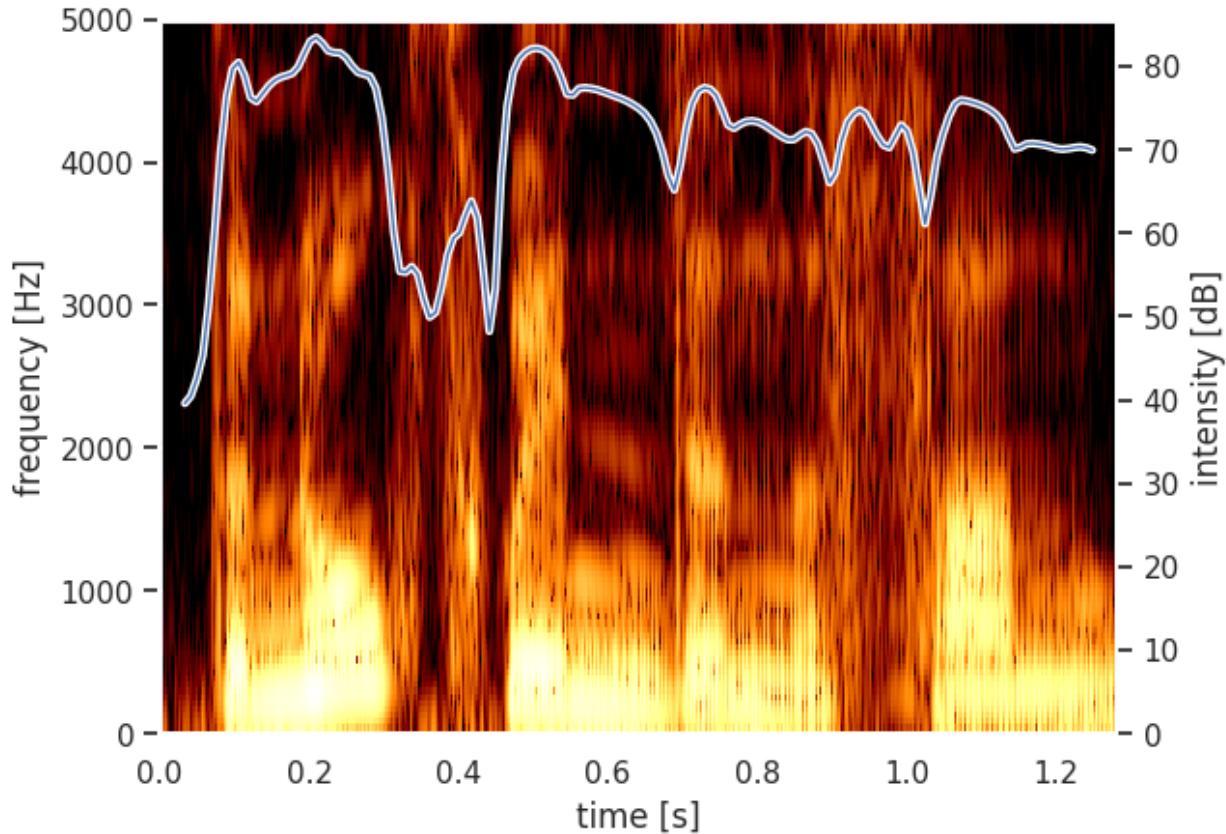
After defining how to plot these, we use Praat (through Parselmouth) to calculate the spectrogram and intensity to actually plot the intensity curve overlaid on the spectrogram.

```
[8]: intensity = snd.to_intensity()
spectrogram = snd.to_spectrogram()
plt.figure()
draw_spectrogram(spectrogram)
```

(continues on next page)

(continued from previous page)

```
plt.twinx()
draw_intensity(intensity)
plt.xlim([snd.xmin, snd xmax])
plt.show()
```



The Parselmouth functions and methods have the same arguments as the Praat commands, so we can for example also change the window size of the spectrogram analysis to get a narrow-band spectrogram. Next to that, let's now have Praat calculate the pitch of the fragment, so we can plot it instead of the intensity.

```
[9]: def draw_pitch(pitch):
    # Extract selected pitch contour, and
    # replace unvoiced samples by NaN to not plot
    pitch_values = pitch.selected_array['frequency']
    pitch_values[pitch_values==0] = np.nan
    plt.plot(pitch.xs(), pitch_values, 'o', markersize=5, color='w')
    plt.plot(pitch.xs(), pitch_values, 'o', markersize=2)
    plt.grid(False)
    plt.ylim(0, pitch.ceiling)
    plt.ylabel("fundamental frequency [Hz]")
```

```
[10]: pitch = snd.to_pitch()
```

```
[11]: # If desired, pre-emphasize the sound fragment before calculating the spectrogram
pre_emphasized_snd = snd.copy()
```

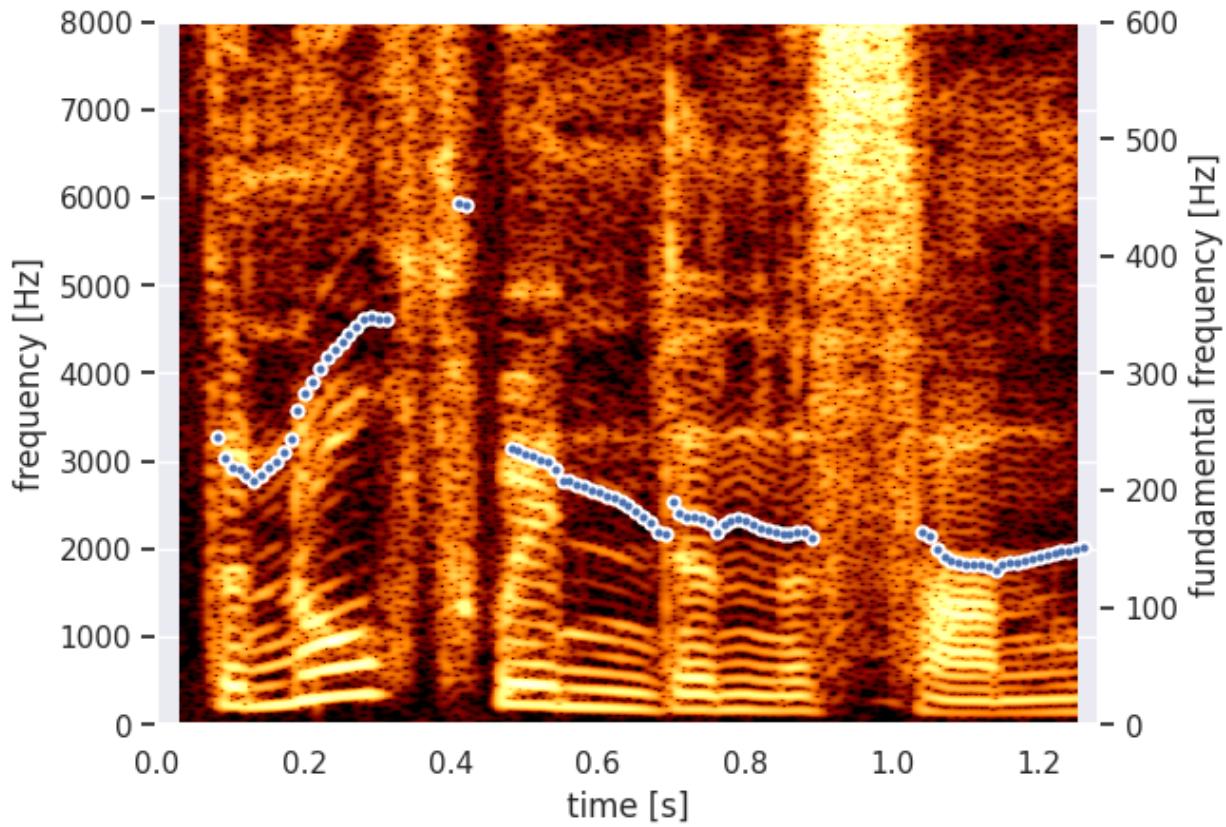
(continues on next page)

(continued from previous page)

```
pre_emphasized_snd.pre_emphasize()
spectrogram = pre_emphasized_snd.to_spectrogram(window_length=0.03, maximum_
frequency=8000)
```

[12]:

```
plt.figure()
draw_spectrogram(spectrogram)
plt.twinx()
draw_pitch(pitch)
plt.xlim([snd.xmin, snd xmax])
plt.show()
```



Using the `FacetGrid` functionality from `seaborn`, we can even plot multiple a structured grid of multiple custom spectrograms. For example, we will read a CSV file (using the `pandas` library) that contains the digit that was spoken, the ID of the speaker and the file name of the audio fragment: `digit_list.csv`, `1_b.wav`, `2_b.wav`, `3_b.wav`, `4_b.wav`, `5_b.wav`, `1_y.wav`, `2_y.wav`, `3_y.wav`, `4_y.wav`, `5_y.wav`

[13]:

```
import pandas as pd

def facet_util(data, **kwargs):
    digit, speaker_id = data[['digit', 'speaker_id']].iloc[0]
    sound = parselmouth.Sound("audio/{}_{}.wav".format(digit, speaker_id))
    draw_spectrogram(sound.to_spectrogram())
    plt.twinx()
    draw_pitch(sound.to_pitch())
    # If not the rightmost column, then clear the right side axis
```

(continues on next page)

(continued from previous page)

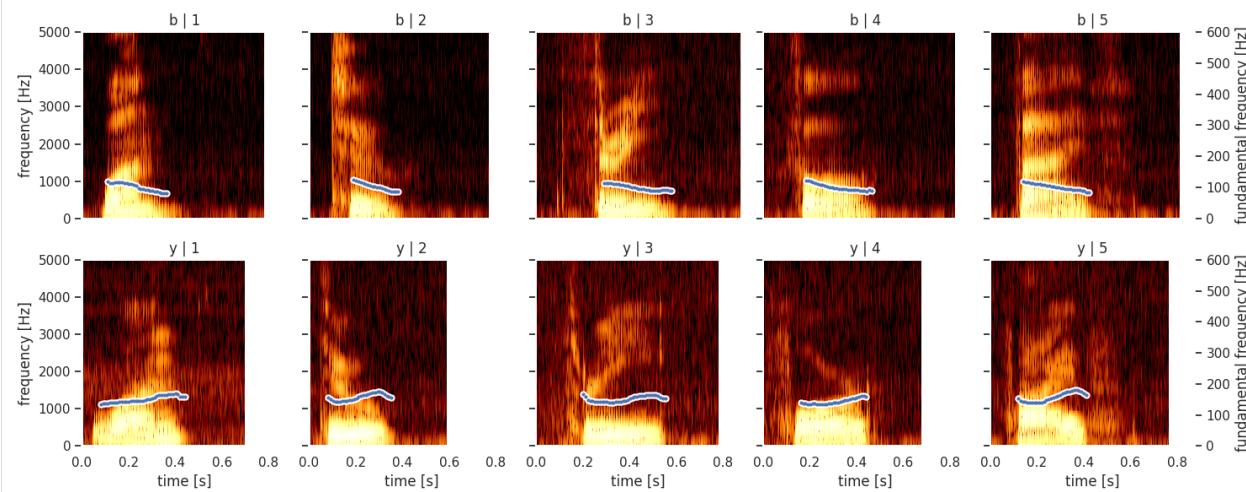
```

if digit != 5:
    plt.ylabel("")
    plt.yticks([])

results = pd.read_csv("other/digit_list.csv")

grid = sns.FacetGrid(results, row='speaker_id', col='digit')
grid.map_dataframe(facet_util)
grid.set_titles(col_template="{col_name}", row_template="{row_name}")
grid.set_axis_labels("time [s]", "frequency [Hz]")
grid.set(facecolor='white', xlim=(0, None))
plt.show()

```



## 2.2 Batch processing of files

Using the Python standard libraries (i.e., the `glob` and `os` modules), we can also quickly code up batch operations e.g. over all files with a certain extension in a directory. For example, we can make a list of all `.wav` files in the `audio` directory, use Praat to pre-emphasize these `Sound` objects, and then write the pre-emphasized sound to a `WAV` and `AIFF` format file.

```
[1]: # Find all .wav files in a directory, pre-emphasize and save as new .wav and .aiff file
import parselmouth

import glob
import os.path

for wave_file in glob.glob("audio/*.wav"):
    print("Processing {}".format(wave_file))
    s = parselmouth.Sound(wave_file)
    s.pre_emphasize()
    s.save(os.path.splitext(wave_file)[0] + "_pre.wav", 'WAV') # or parselmouth.
    SoundFileFormat.WAV instead of 'WAV'
    s.save(os.path.splitext(wave_file)[0] + "_pre.aiff", 'AIFF')
```

```
Processing audio/3_b.wav...
Processing audio/2_y.wav...
Processing audio/4_y.wav...
Processing audio/2_b.wav...
Processing audio/5_b.wav...
Processing audio/1_b.wav...
Processing audio/the_north_wind_and_the_sun.wav...
Processing audio/3_y.wav...
Processing audio/bat.wav...
Processing audio/1_y.wav...
Processing audio/4_b.wav...
Processing audio/5_y.wav...
Processing audio/bet.wav...
```

After running this, the original home directory now contains all of the original .wav files pre-emphasised and written again as .wav and .aiff files. The reading, pre-emphasis, and writing are all done by Praat, while looping over all .wav files is done by standard Python code.

[2]: # List the current contents of the audio/ folder

```
!ls audio/
1_b.wav      2_y_pre.aiff  4_b_pre.wav  bat.wav
1_b_pre.aiff 2_y_pre.wav   4_y.wav       bat_pre.aiff
1_b_pre.wav   3_b.wav      4_y_pre.aiff  bat_pre.wav
1_y.wav       3_b_pre.aiff  4_y_pre.wav   bet.wav
1_y_pre.aiff 3_b_pre.wav   5_b.wav      bet_pre.aiff
1_y_pre.wav   3_y.wav      5_b_pre.aiff  bet_pre.wav
2_b.wav       3_y_pre.aiff  5_b_pre.wav   the_north_wind_and_the_sun.wav
2_b_pre.aiff 3_y_pre.wav   5_y.wav      the_north_wind_and_the_sun_pre.aiff
2_b_pre.wav   4_b.wav      5_y_pre.aiff  the_north_wind_and_the_sun_pre.wav
2_y.wav       4_b_pre.aiff  5_y_pre.wav
```

[3]: # Remove the generated audio files again, to clean up the output from this example

```
!rm audio/*_pre.wav
!rm audio/*_pre.aiff
```

Similarly, we can use the `pandas` library to read a CSV file with data collected in an experiment, and loop over that data to e.g. extract the mean harmonics-to-noise ratio. The `results.csv` has the following structure:

condition	...	pp_id
0	...	1877
1	...	801
1	...	2456
0	...	3126

The following code would read such a table, loop over it, use Praat through Parselmouth to calculate the analysis of each row, and then write an augmented CSV file to disk. To illustrate we use an example set of sound fragments: `results.csv`, `1_b.wav`, `2_b.wav`, `3_b.wav`, `4_b.wav`, `5_b.wav`, `1_y.wav`, `2_y.wav`, `3_y.wav`, `4_y.wav`, `5_y.wav`

In our example, the original CSV file, `results.csv` contains the following table:

```
[4]: import pandas as pd

print(pd.read_csv("other/results.csv"))

    condition pp_id
0            3     y
1            5     y
2            4     b
3            2     y
4            5     b
5            2     b
6            3     b
7            1     y
8            1     b
9            4     y
```

```
[5]: def analyse_sound(row):
    condition, pp_id = row['condition'], row['pp_id']
    filepath = "audio/{}_{}.wav".format(condition, pp_id)
    sound = parselmouth.Sound(filepath)
    harmonicity = sound.to_harmonicity()
    return harmonicity.values[harmonicity.values != -200].mean()

# Read in the experimental results file
dataframe = pd.read_csv("other/results.csv")

# Apply parselmouth wrapper function row-wise
dataframe['harmonics_to_noise'] = dataframe.apply(analyse_sound, axis='columns')

# Write out the updated dataframe
dataframe.to_csv("processed_results.csv", index=False)
```

We can now have a look at the results by reading in the processed\_results.csv file again:

```
[6]: print(pd.read_csv("processed_results.csv"))

    condition pp_id  harmonics_to_noise
0            3     y          22.615414
1            5     y          16.403205
2            4     b          17.839167
3            2     y          21.054674
4            5     b          16.092489
5            2     b          12.378289
6            3     b          15.718858
7            1     y          16.704779
8            1     b          12.874451
9            4     y          18.431586
```

```
[7]: # Clean up, remove the CSV file generated by this example
!rm processed_results.csv
```

## 2.3 Pitch manipulation and Praat commands

Another common use of Praat functionality is to manipulate certain features of an existing audio fragment. For example, in the context of a perception experiment one might want to change the pitch contour of an existing audio stimulus while keeping the rest of the acoustic features the same. Parselmouth can then be used to access the Praat algorithms that accomplish this, from Python.

Since this Praat Manipulation functionality has currently not been ported to Parselmouth's Python interface, we will need to use Parselmouth interface to access *raw* Praat commands.

In this example, we will increase the pitch contour of an audio recording of the word “*four*”, `4_b.wav`, by one octave. To do so, let's start by importing Parselmouth and opening the audio file:

```
[1]: import parselmouth

sound = parselmouth.Sound("audio/4_b.wav")
```

We can also listen to this audio fragment:

```
[2]: from IPython.display import Audio
Audio(data=sound.values, rate=sound.sampling_frequency)

[2]: <IPython.lib.display.Audio object>
```

However, now we want to use the Praat Manipulation functionality, but unfortunately, Parselmouth does not yet contain a Manipulation class and the necessary functionality is not directly accessible through the `Sound` object `sound`. To directly access the Praat commands conveniently from Python, we can make use of the `parselmouth.praat.call` function.

```
[3]: from parselmouth.praat import call

manipulation = call(sound, "To Manipulation", 0.01, 75, 600)
```

```
[4]: type(manipulation)
[4]: parselmouth.Data
```

Note how we first pass in the object(s) that would be selected in Praat's object list. The next argument to this function is the name of the command as it would be used in a script or can be seen in the Praat user interface. Finally, the arguments to this command's parameters are passed to the function (in this case, Praat's default values for “*Time step (s)*”, “*Minimum pitch (Hz)*”, and “*Maximum pitch (Hz)*”). This call to `parselmouth.praat.call` will then return the result of the command as a Python type or Parselmouth object. In this case, a Praat Manipulation object would be created, so our function returns a `parselmouth.Data` object, as a `parselmouth.Manipulation` class does not exist in Parselmouth. However, we can still query the class name the underlying Praat object has:

```
[5]: manipulation.class_name
[5]: 'Manipulation'
```

Next, we can continue using Praat functionality to further use this `manipulation` object similar to how one would achieve this in Praat. Here, note how we can mix normal Python (e.g. integers and lists), together with the normal use of Parselmouth as Python library (e.g., `sound.xmin`) as well as with the `parselmouth.praat.call` function.

```
[6]: pitch_tier = call(manipulation, "Extract pitch tier")

call(pitch_tier, "Multiply frequencies", sound.xmin, sound xmax, 2)
```

(continues on next page)

(continued from previous page)

```
call([pitch_tier, manipulation], "Replace pitch tier")
sound_octave_up = call(manipulation, "Get resynthesis (overlap-add)")
```

[7]: `type(sound_octave_up)`

[7]: `parselmouth.Sound`

The last invocation of `call` resulted in a Praat Sound object being created and returned. Because Parselmouth knows that this type corresponds to a `parselmouth.Sound` Python object, the Python type of this object is not a `parselmouth.Data`. Rather, this object is now equivalent to the one we created at the start of this example. As such, we can use this new object normally, calling methods and accessing its contents. Let's listen and see if we succeeded in increasing the pitch by one octave:

[8]: `Audio(data=sound_octave_up.values, rate=sound_octave_up.sampling_frequency)`

[8]: `<IPython.lib.display.Audio object>`

And similarly, we could also for example save the sound to a new file.

[9]: `sound_octave_up.save("4_b_octave_up.wav", "WAV")`

[10]: `Audio(filename="4_b_octave_up.wav")`

[10]: `<IPython.lib.display.Audio object>`

[11]: `# Clean up the created audio file again`  
`!rm 4_b_octave_up.wav`

We can of course also turn this combination of commands into a custom function, to be reused in later code:

```
[12]: def change_pitch(sound, factor):
    manipulation = call(sound, "To Manipulation", 0.01, 75, 600)

    pitch_tier = call(manipulation, "Extract pitch tier")

    call(pitch_tier, "Multiply frequencies", sound.xmin, sound xmax, factor)

    call([pitch_tier, manipulation], "Replace pitch tier")
    return call(manipulation, "Get resynthesis (overlap-add)")
```

Using Jupyter widgets, one can then change the audio file or the pitch change factor, and interactively hear how this sounds.

*To try this for yourself, open an online, interactive version of this notebook on Binder! (see link at the top of this notebook)*

```
[13]: import ipywidgets
import glob

def interactive_change_pitch(audio_file, factor):
    sound = parselmouth.Sound(audio_file)
    sound_changed_pitch = change_pitch(sound, factor)
    return Audio(data=sound_changed_pitch.values, rate=sound_changed_pitch.sampling_
```

(continues on next page)

(continued from previous page)

```

↳frequency)

w = ipywidgets.interact(interactive_change_pitch,
                        audio_file=ipywidgets.Dropdown(options=sorted(glob.glob("audio/*.
↳wav")), value="audio/4_b.wav"),
                        factor=ipywidgets.FloatSlider(min=0.25, max=4, step=0.05,_
↳value=1.5))

interactive(children=(Dropdown(description='audio_file', index=6, options=('audio/1_b.wav
↳', 'audio/1_y.wav', '...

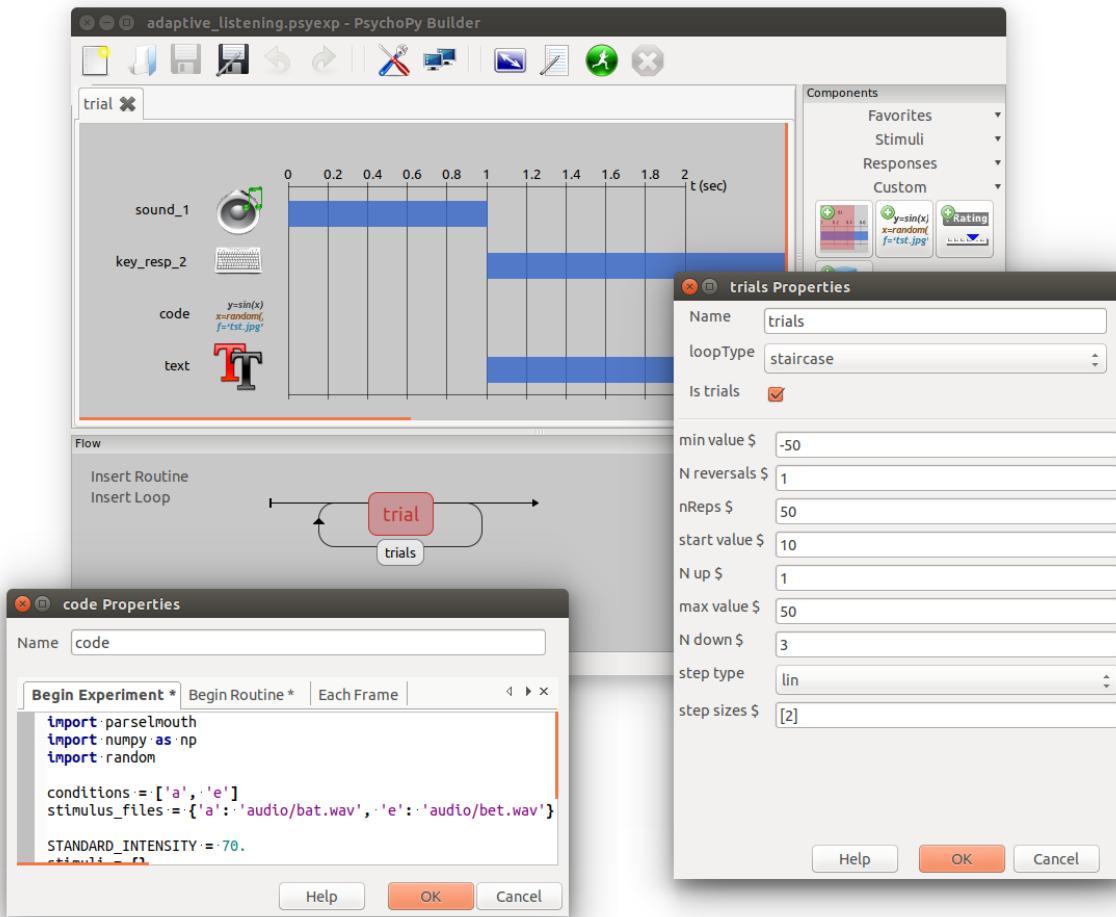
```

## 2.4 PsychoPy experiments

Parselmouth also allows Praat functionality to be included in an interactive *PsychoPy* experiment (refer to the subsection on *installing Parselmouth for PsychoPy* for detailed installation instructions for the PsychoPy graphical interface, the *PsychoPy Builder*). The following example shows how easily Python code that uses Parselmouth can be injected in such an experiment; following an adaptive staircase experimental design, at each trial of the experiment a new stimulus is generated based on the responses of the participant. See e.g. Kaernbach, C. (2001). Adaptive threshold estimation with unforced-choice tasks. *Attention, Perception, & Psychophysics*, 63, 1377–1388., or the PsychoPy tutorial at <https://www.psychopy.org/coder/tutorial2.html>.

In this example, we use an adaptive staircase experiment to determine the minimal amount of noise that makes the participant unable to distinguish between two audio fragments, “bat” and “bet” (`bat.wav`, `bet.wav`). At every iteration of the experiment, we want to generate a version of these audio files with a specific signal-to-noise ratio, of course using Parselmouth to do so. Depending on whether the participant correctly identifies whether the noisy stimulus was “bat” or “bet”, the noise level is then either increased or decreased.

As Parselmouth is just another Python library, using it from the PsychoPy *Coder* interface or from a standard Python script that imports the `psychopy` module is quite straightforward. However, PsychoPy also features a so-called *Builder* interface, which is a graphical interface to set up experiments with minimal or no coding. In this *Builder*, a user can create multiple experimental ‘*routines*’ out of different ‘*components*’ and combine them through ‘*loops*’, that can all be configured graphically:



For our simple example, we create a single routine `trial`, with a Sound, a Keyboard, and a Text component. We also insert a loop around this routine of the type `staircase`, such that PsychoPy will take care of the actual implementation of the loop in adaptive staircase design. The full PsychoPy experiment which can be opened in the *Builder* can be downloaded here: [adaptive\\_listening.psyexp](#)

Finally, to customize the behavior of the `trial` routine and to be able to use Parselmouth inside the PsychoPy experiment, we still add a Code component to the routine. This component will allow us to write Python code that interacts with the rest of the components and with the adaptive staircase loop. The Code component has different tabs, that allow us to insert custom code at different points during the execution of our trial.

First, there is the **Begin Experiment** tab. The code in this tab is executed only once, at the start of the experiment. We use this to set up the Python environment, importing modules and initializing variables, and defining constants:

```
[1]: # ** Begin Experiment **

import parselmouth
import numpy as np
import random

conditions = ['a', 'e']
stimulus_files = {'a': "audio/bat.wav", 'e': "audio/bet.wav"}
```

(continues on next page)

(continued from previous page)

```
STANDARD_INTENSITY = 70.
stimuli = []
for condition in conditions:
    stimulus = parselmouth.Sound(stimulus_files[condition])
    stimulus.scale_intensity(STANDARD_INTENSITY)
    stimuli[condition] = stimulus
```

The code in the **Begin Routine** tab is executed before the routine, so in our example, for every iteration of the surrounding staircase loop. This allows us to actually use Parselmouth to generate the stimulus that should be played to the participant during this iteration of the routine. To do this, we need to access the current value of the adaptive staircase algorithm: PsychoPy stores this in the Python variable `level`. For example, at some point during the experiment, this could be 10 (representing a signal-to-noise ratio of 10 dB):

[2]: `level = 10`

To execute the code we want to put in the **Begin Routine** tab, we need to add a few variables that would be made available by the PsychoPy Builder, normally:

[3]: `# 'filename' variable is also set by PsychoPy and contains base file name of saved log/
→output files
filename = "data/participant_staircase_23032017"

# PsychoPy also create a Trials object, containing e.g. information about the current_
→iteration of the loop
# So let's quickly fake this, in this example, such that the code can be executed without_
→errors
# In PsychoPy this would be a `psychopy.data.TrialHandler` (https://www.psychopy.org/api/
→data.html#psychopy.data.TrialHandler)
class MockTrials:
 def addResponse(self, response):
 print("Registering that this trial was {}successful".format("" if response else
→"un"))
trials = MockTrials()
trials.thisTrialN = 5 # We only need the 'thisTrialN' attribute of the 'trials' variable

# The Sound component can also be accessed by its name, so let's quickly mock that as_
→well
# In PsychoPy this would be a `psychopy.sound.Sound` (https://www.psychopy.org/api/sound.
→html#psychopy.sound.Sound)
class MockSound:
 def setSound(self, file_name):
 print("Setting audio file of Sound component to '{}'".format(file_name))
sound_1 = MockSound()

# And the same for our Keyboard component, `key_resp_2`:
class MockKeyboard:
 pass
key_resp_2 = MockKeyboard()

# Finally, let's also seed the random module to have a consistent output across different_
→runs
random.seed(42)`

```
[4]: # Let's also create the directory where we will store our example output  
!mkdir data
```

Now, we can execute the code that would be in the **Begin Routine** tab:

```
[5]: # ** Begin Routine **  
  
random_condition = random.choice(conditions)  
random_stimulus = stimuli[random_condition]  
  
noise_samples = np.random.normal(size=random_stimulus.n_samples)  
noisy_stimulus = parselmouth.Sound(noise_samples,  
                                    sampling_frequency=random_stimulus.sampling_frequency)  
noisy_stimulus.scale_intensity(STANDARD_INTENSITY - level)  
noisy_stimulus.values += random_stimulus.values  
noisy_stimulus.scale_intensity(STANDARD_INTENSITY)  
  
# use 'filename' to save our custom stimuli  
stimulus_file_name = filename + "_stimulus_" + str(trials.thisTrialN) + ".wav"  
noisy_stimulus.resample(44100).save(stimulus_file_name, 'WAV')  
sound_1.setSound(stimulus_file_name)  
  
Setting audio file of Sound component to 'data/participant_staircase_23032017_stimulus_5.  
wav'
```

Let's listen to the file we have just generated and that we would play to the participant:

```
[6]: from IPython.display import Audio  
Audio(filename="data/participant_staircase_23032017_stimulus_5.wav")  
  
[6]: <IPython.lib.display.Audio object>
```

In this example, we do not really need to have code executed during the trial (i.e., in the **Each Frame** tab). However, at the end of the trial, we need to inform the PsychoPy staircase loop whether the participant was correct or not, because this will affect the further execution the adaptive staircase, and thus value of the `level` variable set by PsychoPy. For this we add a final line in the **End Routine** tab. Let's say the participant guessed “bat” and pressed the a key:

```
[7]: key_resp_2.keys = 'a'
```

The **End Routine** tab then contains the following code to check the participant’s answer against the randomly chosen condition, and to inform the `trials` object of whether the participant was correct:

```
[8]: # ** End Routine **  
  
trials.addResponse(key_resp_2.keys == random_condition)  
Registering that this trial was successful
```

```
[9]: # Clean up the output directory again  
!rm -r data
```

## 2.5 Web service

Since Parselmouth is a normal Python library, it can also easily be used within the context of a web server. There are several Python frameworks that allow to quickly set up a web server or web service. In this examples, we will use [Flask](#) to show how easily one can set up a web service that uses Parselmouth to access Praat functionality such as the pitch track estimation algorithms. This functionality can then be accessed by clients without requiring either Praat, Parselmouth, or even Python to be installed, for example within the context of an online experiment.

All that is needed to set up the most basic web server in Flask is a single file. We adapt the [standard Flask example](#) to accept a sound file, access Parselmouth's [Sound.to\\_pitch](#), and then send back the list of pitch track frequencies. Note that apart from [saving the file that was sent in the HTTP request](#) and encoding the resulting list of frequencies in JSON, the Python code of the `pitch_track` function is the same as one would write in a normal Python script using Parselmouth.

```
[1]: %%writefile server.py

from flask import Flask, request, jsonify
import tempfile

app = Flask(__name__)

@app.route('/pitch_track', methods=['POST'])
def pitch_track():
    import parselmouth

    # Save the file that was sent, and read it into a parselmouth.Sound
    with tempfile.NamedTemporaryFile() as tmp:
        tmp.write(request.files['audio'].read())
        sound = parselmouth.Sound(tmp.name)

    # Calculate the pitch track with Parselmouth
    pitch_track = sound.to_pitch().selected_array['frequency']

    # Convert the NumPy array into a list, then encode as JSON to send back
    return jsonify(list(pitch_track))
```

Writing `server.py`

Normally, we can then run the server typing `FLASK_APP=server.py flask run` on the command line, as explained in the [Flask documentation](#). Please do note that to run this server publicly, in a secure way and as part of a bigger setup, other options are available to deploy! Refer to the [Flask deployment documentation](#).

However, to run the server from this Jupyter notebook and still be able to run the other cells that access the functionality on the client side, the following code will start the server in a separate thread and print the output of the running server.

```
[2]: import os
import subprocess
import sys
import time

# Start a subprocess that runs the Flask server
p = subprocess.Popen([sys.executable, "-m", "flask", "run"], env=dict(**os.environ, FLASK_APP="server.py"), stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

(continues on next page)

(continued from previous page)

```
# Start two subthreads that forward the output from the Flask server to the output of ↵
→ the Jupyter notebook
def forward(i, o):
    while p.poll() is None:
        l = i.readline().decode('utf-8')
        if l:
            o.write("[SERVER] " + l)

import threading
threading.Thread(target=forward, args=(p.stdout, sys.stdout)).start()
threading.Thread(target=forward, args=(p.stderr, sys.stderr)).start()

# Let's give the server a bit of time to make sure it has started
time.sleep(2)

[SERVER] * Serving Flask app 'server.py'
[SERVER] * Debug mode: off
[SERVER] WARNING: This is a development server. Do not use it in a production deployment.
→ Use a production WSGI server instead.
[SERVER] * Running on http://127.0.0.1:5000
[SERVER] Press CTRL+C to quit
```

Now that the server is up and running, we can make a standard HTTP request to this web service. For example, we can send a Wave file with an audio recording of someone saying “*The north wind and the sun [...]*”: `the_north_wind_and_the_sun.wav`, extracted from a Wikipedia Commons audio file.

```
[3]: from IPython.display import Audio
Audio(filename="audio/the_north_wind_and_the_sun.wav")
[3]: <IPython.lib.display.Audio object>
```

To do so, we use the `requests` library in this example, but we could use any library to send a standard HTTP request.

```
[4]: import requests
import json

# Load the file to send
files = {'audio': open("audio/the_north_wind_and_the_sun.wav", 'rb')}
# Send the HTTP request and get the reply
reply = requests.post("http://127.0.0.1:5000/pitch_track", files=files)
# Extract the text from the reply and decode the JSON into a list
pitch_track = json.loads(reply.text)
print(pitch_track)

[0.0, 0.0, 0.0, 0.0, 0.0, 245.46350823786898, 228.46732333120045, 220.229881904913, ↵
→ 217.9494117767135, 212.32120094882643, 208.42371077564596, 213.3210292245136, 219.
→ 22164169979897, 225.08564349338334, 232.58018420251648, 243.6102854675347, 267.
→ 9586673940531, 283.57192373203253, 293.09087794771966, 303.9716558501677, 314.
→ 16812500255537, 320.11744147538917, 326.34395013825196, 333.3632387299925, 340.
→ 0277922275489, 345.8240749033839, 348.57743419008335, 346.9665344057159, 346.
→ 53179321965666, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 445.1355539937184, 442.
→ 99367847432956, 0.0, 0.0, 0.0, 0.0, 236.3912949256524, 233.77304383699934, 231.
→ 61759183978316, 229.252937317608, 226.5388725505901, 223.6713912521482, 217.]
```

(continues on next page)

(continued from previous page)

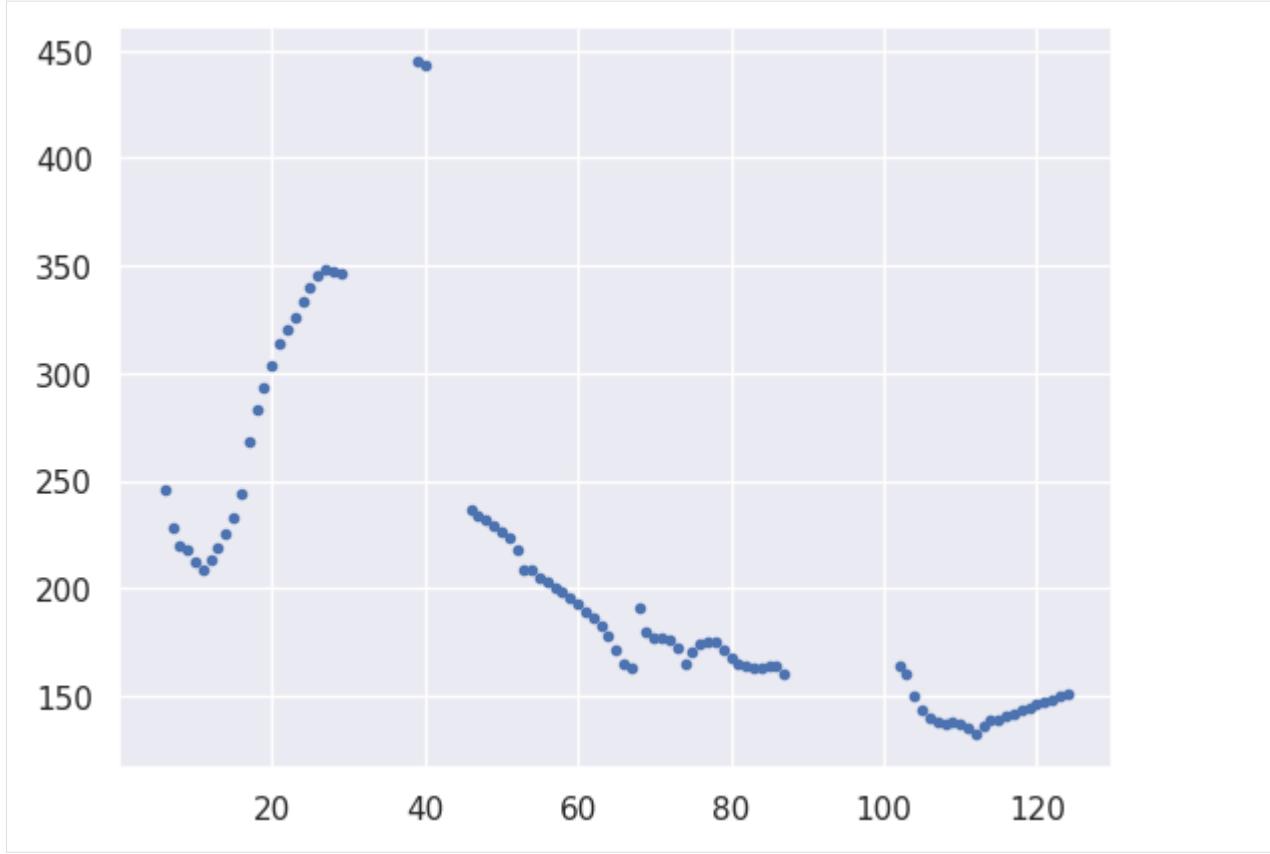
→ 56247158178041, 208.75233223541412, 208.36854272051312, 205.1132684638252, 202.  
→ 99628328370704, 200.74245529822406, 198.379243723561, 195.71387722456126, 192.  
→ 92640662381228, 189.55087006373063, 186.29856999154498, 182.60612897184708, 178.  
→ 0172095327713, 171.7286500573546, 164.43397092360505, 163.15047735066148, 190.  
→ 94898597265222, 180.11404296436555, 177.42215658133307, 176.85852955755865, 175.  
→ 90234348007218, 172.72381274834703, 165.07291074214982, 170.84308758689093, 173.  
→ 84326581969435, 175.39817924857263, 174.73813404735137, 171.30666910901442, 167.  
→ 57344824865035, 165.26925804867895, 164.0488248694515, 163.3665771538607, 162.  
→ 9182321154844, 164.4049979046003, 164.16734205916592, 160.17875848111373, 0.0, 0.0, 0.  
→ 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 163.57343758482958, 160.  
→ 63654708070163, 150.27906547408838, 143.6142724404569, 139.70737167424176, 138.  
→ 15535972924215, 137.401926952887, 137.45520345586323, 136.78723483908712, 135.  
→ 18334597312617, 132.3066180187801, 136.04747210818914, 138.65745092917942, 139.  
→ 1335736781387, 140.238485464634, 141.83711308294014, 143.10991285599226, 144.  
→ 40501561368708, 146.07295382762607, 147.47513524525806, 148.1692013818143, 149.  
→ 54122031709116, 151.0336292203337]

Since we used the standard `json` library from Python to decode the reply from server, `pitch_track` is now a normal list of `floats` and we can for example plot the estimated pitch track:

```
[5]: import matplotlib.pyplot as plt  
      import seaborn as sns
```

```
[6]: sns.set() # Use seaborn's default style to make attractive graphs  
plt.rcParams['figure.dpi'] = 100 # Show nicely large images in this notebook
```

```
[7]: plt.figure()
plt.plot([float('nan') if x == 0.0 else x for x in pitch_track], '.')
plt.show()
```



Refer to the [examples on plotting](#) for more details on using Parselmouth for plotting.

Importantly, Parselmouth is thus only needed by the server; the client only needs to be able to send a request and read the reply. Consequently, we could even use a different programming language on the client's side. For example, one could make build a HTML page with JavaScript to make the request and do something with the reply:

```
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <script type="text/javascript" src="jquery.min.js"></script>
    <script type="text/javascript" src="plotly.min.js"></script>
    <script type="text/javascript">
        var update_plot = function() {
            var audio = document.getElementById("audio").files[0];
            var formData = new FormData();
            formData.append("audio", audio);

            $.getJSON({url: "http://127.0.0.1:5000/pitch_track", method: "POST",
                data: formData, processData: false, contentType: false,
                success: function(data){
                    Plotly.newPlot("plot", [{ x: [...Array(data.length).keys()],
                        y: data.map(function(x) { return x == 0 ?
                            undefined : x; }), type: "lines" }]);}});
        };
    </script>
</head>
```

(continues on next page)

(continued from previous page)

```
<body>
<form onsubmit="update_plot(); return false;">
    <input type="file" name="audio" id="audio" />
    <input type="submit" value="Get pitch track" />
    <div id="plot" style="width:1000px;height:600px;"></div>
</form>
</body>
```

Again, one thing to take into account is the security of running such a web server. However, apart from deploying the flask server in a secure and performant way, we also need one extra thing to circumvent a standard security feature of the browser. Without handling Cross Origin Resource Sharing (CORS) on the server, the JavaScript code on the client side will not be able to access the web service's reply. A Flask extension exists however, [Flask-CORS](#), and we refer to its documentation for further details.

[8]: # Let's shut down the server  
p.kill()

[9]: # Cleaning up the file that was written to disk  
!rm server.py

## 2.6 Projects using Parselmouth

The following projects provide larger, real-life examples and demonstrate the use of Parselmouth:

- The [my-voice-analysis](#) and [myprosody](#) projects by Shahab Sabahi (@Shahabks) provide Python libraries for voice analysis and acoustical statistics, interfacing Python to his previously developed Praat scripts.
- David R. Feinberg (@drfeinberg) has written multiple Python scripts and programs with Parselmouth to analyse properties of speech recordings:
  - [Praat Scripts](#) is a collection of Praat scripts used in research, translated into Python.
  - [Voice Lab Software](#) is a GUI application to measure and manipulate voices.

---

**Note:** If you have a project using Parselmouth that could be useful for others, and want to add it to this list, do let us know on [Gitter](#)!

---



## API REFERENCE

```
parselmouth.VERSION = '0.4.3'
```

This version of Parselmouth.

```
parselmouth.PRAAT_VERSION = '6.1.38'
```

The Praat version on which this version of Parselmouth is based.

```
parselmouth.PRAAT_VERSION_DATE = '2 January 2021'
```

The release date of the Praat version on which this version of Parselmouth is based.

```
exception parselmouth.PraatError
```

Bases: RuntimeError

```
exception parselmouth.PraatFatal
```

Bases: BaseException

```
exception parselmouth.PraatWarning
```

Bases: UserWarning

```
class parselmouth.AmplitudeScaling
```

Bases: pybind11\_object

```
__eq__(self: object, other: object) → bool
```

```
__hash__(self: object) → int
```

```
__index__(self: parselmouth.AmplitudeScaling) → int
```

```
__init__(self: parselmouth.AmplitudeScaling, value: int) → None
```

```
__init__(self: parselmouth.AmplitudeScaling, arg0: str) → None
```

```
__int__(self: parselmouth.AmplitudeScaling) → int
```

```
__ne__(self: object, other: object) → bool
```

```
__repr__(self: object) → str
```

```
__str__()
```

name(self: handle) -> str

```
INTEGRAL = <AmplitudeScaling.INTEGRAL: 1>
```

```
NORMALIZE = <AmplitudeScaling.NORMALIZE: 3>
```

```
PEAK_0_99 = <AmplitudeScaling.PEAK_0_99: 4>
```

```
SUM = <AmplitudeScaling.SUM: 2>

property name

property value

class parselmouth.CC
    Bases: TimeFrameSampled, Sampled

    class Frame
        Bases: pybind11_object

        __getitem__(self: parselmouth.CC.Frame, i: int) → float
        __init__(*args, **kwargs)
        __len__(self: parselmouth.CC.Frame) → int
        __setitem__(self: parselmouth.CC.Frame, i: int, value: float) → None
        to_array(self: parselmouth.CC.Frame) → numpy.ndarray[numpy.float64]
        property c
        property c0
        __getitem__(self: parselmouth.CC, i: int) → parselmouth.CC.Frame
        __getitem__(self: parselmouth.CC, ij: Tuple[int, int]) → float
        __init__(*args, **kwargs)
        __iter__(self: parselmouth.CC) → Iterator
        __setitem__(self: parselmouth.CC, ij: Tuple[int, int], value: float) → None
        get_c0_value_in_frame(self: parselmouth.CC, frame_number: Positive[int]) → float
        get_frame(self: parselmouth.CC, frame_number: Positive[int]) → parselmouth.CC.Frame
        get_number_of_coefficients(self: parselmouth.CC, frame_number: Positive[int]) → int
        get_value_in_frame(self: parselmouth.CC, frame_number: Positive[int], index: Positive[int]) → float
        to_array(self: parselmouth.CC) → numpy.ndarray[numpy.float64]
        to_matrix(self: parselmouth.CC) → parselmouth.Matrix

        property fmax
        property fmin
        property max_n_coefficients

class parselmouth.Data
    Bases: Thing

    class FileFormat
        Bases: pybind11_object
        __eq__(self: object, other: object) → bool
```

```
__hash__(self: object) → int
__index__(self: parselmouth.Data.FileFormat) → int
__init__(self: parselmouth.Data.FileFormat, value: int) → None
__init__(self: parselmouth.Data.FileFormat, arg0: str) → None
__int__(self: parselmouth.Data.FileFormat) → int
__ne__(self: object, other: object) → bool
__repr__(self: object) → str
__str__()
    name(self: handle) -> str
BINARY = <FileFormat.BINARY: 2>
SHORT_TEXT = <FileFormat.SHORT_TEXT: 1>
TEXT = <FileFormat.TEXT: 0>
property name
property value
__copy__(self: parselmouth.Data) → parselmouth.Data
__deepcopy__(self: parselmouth.Data, memo: dict) → parselmouth.Data
__eq__(self: parselmouth.Data, other: parselmouth.Data) → bool
__init__(*args, **kwargs)
__ne__(self: parselmouth.Data, other: parselmouth.Data) → bool
copy(self: parselmouth.Data) → parselmouth.Data
static read(file_path: str) → parselmouth.Data
    Read a file into a parselmouth.Data object.
```

**Parameters**

`file_path` (`str`) – The path of the file on disk to read.

**Returns**

The Praat Data object that was read.

**Return type**

`parselmouth.Data`

**See also:**

*Praat: “Read from file...”*

```
save(self: parselmouth.Data, file_path: str, format: parselmouth.Data.FileFormat = <FileFormat.TEXT: 0>) → None
save_as_binary_file(self: parselmouth.Data, file_path: str) → None
save_as_short_text_file(self: parselmouth.Data, file_path: str) → None
```

```
save_as_text_file(self: parselmouth.Data, file_path: str) → None
    __hash__ = None

class parselmouth.Formant
    Bases: TimeFrameSampled, Sampled
    __init__(*args, **kwargs)
    get_bandwidth_at_time(self: parselmouth.Formant, formant_number: Positive[int], time: float, unit:
        parselmouth.FormantUnit = <FormantUnit.HERTZ: 0>) → float
    get_value_at_time(self: parselmouth.Formant, formant_number: Positive[int], time: float, unit:
        parselmouth.FormantUnit = <FormantUnit.HERTZ: 0>) → float

class parselmouth.FormantUnit
    Bases: pybind11_object
    __eq__(self: object, other: object) → bool
    __hash__(self: object) → int
    __index__(self: parselmouth.FormantUnit) → int
    __init__(self: parselmouth.FormantUnit, value: int) → None
    __init__(self: parselmouth.FormantUnit, arg0: str) → None
    __int__(self: parselmouth.FormantUnit) → int
    __ne__(self: object, other: object) → bool
    __repr__(self: object) → str
    __str__()
        name(self: handle) -> str
    BARK = <FormantUnit.BARK: 1>
    HERTZ = <FormantUnit.HERTZ: 0>
    property name
    property value

class parselmouth.Function
    Bases: Data
    __init__(*args, **kwargs)
    scale_x_by(self: parselmouth.Function, scale: Positive[float]) → None
    scale_x_to(self: parselmouth.Function, new_xmin: float, new_xmax: float) → None
    shift_x_by(self: parselmouth.Function, shift: float) → None
    shift_x_to(self: parselmouth.Function, x: float, new_x: float) → None
    property xmax
```

```

property xmin
property xrange

class parselmouth.Harmonicity
    Bases: TimeFrameSampled, Vector
        __init__(*args, **kwargs)
        get_value(self: parselmouth.Harmonicity, time: float, interpolation: parselmouth.ValueInterpolation = <ValueInterpolation.CUBIC: 2>) → float

class parselmouth.Intensity
    Bases: TimeFrameSampled, Vector
        class AveragingMethod
            Bases: pybind11_object
                __eq__(self: object, other: object) → bool
                __hash__(self: object) → int
                __index__(self: parselmouth.Intensity.AveragingMethod) → int
                __init__(self: parselmouth.Intensity.AveragingMethod, value: int) → None
                __init__(self: parselmouth.Intensity.AveragingMethod, arg0: str) → None
                __int__(self: parselmouth.Intensity.AveragingMethod) → int
                __ne__(self: object, other: object) → bool
                __repr__(self: object) → str
                __str__()
                    name(self: handle) -> str
            DB = <AveragingMethod.DB: 3>
            ENERGY = <AveragingMethod.ENERGY: 1>
            MEDIAN = <AveragingMethod.MEDIAN: 0>
            SONES = <AveragingMethod.SONES: 2>
            property name
            property value
            __init__(*args, **kwargs)
            get_average(self: parselmouth.Intensity, from_time: Optional[float] = None, to_time: Optional[float] = None, averaging_method: parselmouth.Intensity.AveragingMethod = <AveragingMethod.ENERGY: 1>) → float
            get_value(self: parselmouth.Intensity, time: float, interpolation: parselmouth.ValueInterpolation = <ValueInterpolation.CUBIC: 2>) → float

parselmouth.Interpolation
    alias of ValueInterpolation

```

```
class parselmouth.MFCC
    Bases: CC

    __init__(*args, **kwargs)

    convolve(self: parselmouth.MFCC, other: parselmouth.MFCC, scaling: parselmouth.AmplitudeScaling =
        <AmplitudeScaling.PEAK_0_99: 4>, signal_outside_time_domain:
        parselmouth.SignalOutsideTimeDomain = <SignalOutsideTimeDomain.ZERO: 1>) →
        parselmouth.Sound

    cross_correlate(self: parselmouth.MFCC, other: parselmouth.MFCC, scaling:
        parselmouth.AmplitudeScaling = <AmplitudeScaling.PEAK_0_99: 4>,
        signal_outside_time_domain: parselmouth.SignalOutsideTimeDomain =
        <SignalOutsideTimeDomain.ZERO: 1>) → parselmouth.Sound

    extract_features(self: parselmouth.MFCC, window_length: Positive[float] = 0.025, include_energy: bool
        = False) → parselmouth.Matrix

    to_matrix_features(self: parselmouth.MFCC, window_length: Positive[float] = 0.025, include_energy:
        bool = False) → parselmouth.Matrix

    to_sound(self: parselmouth.MFCC) → parselmouth.Sound

class parselmouth.Matrix
    Bases: SampledXY

    __init__(*args, **kwargs)

    as_array(self: parselmouth.Matrix) → numpy.ndarray[numpy.float64]

    at_xy(self: parselmouth.Matrix, x: float, y: float) → float

    formula(self: parselmouth.Matrix, formula: str, from_x: float | None = None, to_x: float | None = None,
        from_y: float | None = None, to_y: float | None = None) → None

    formula(self: parselmouth.Matrix, formula: str, x_range: Tuple[float | None, float | None] = (None, None),
        y_range: Tuple[float | None, float | None] = (None, None)) → None

    get_column_distance(self: parselmouth.Matrix) → float

    get_highest_x(self: parselmouth.Matrix) → float

    get_highest_y(self: parselmouth.Matrix) → float

    get_lowest_x(self: parselmouth.Matrix) → float

    get_lowest_y(self: parselmouth.Matrix) → float

    get_maximum(self: parselmouth.Matrix) → float

    get_minimum(self: parselmouth.Matrix) → float

    get_number_of_columns(self: parselmouth.Matrix) → int

    get_number_of_rows(self: parselmouth.Matrix) → int

    get_row_distance(self: parselmouth.Matrix) → float

    get_sum(self: parselmouth.Matrix) → float
```

---

```

get_value_at_xy(self: parselmouth.Matrix, x: float, y: float) → float
get_value_in_cell(self: parselmouth.Matrix, row_number: Positive[int], column_number: Positive[int])
    → float

get_x_of_column(self: parselmouth.Matrix, column_number: Positive[int]) → float
get_y_of_row(self: parselmouth.Matrix, row_number: Positive[int]) → float
save_as_headerless_spreadsheet_file(self: parselmouth.Matrix, file_path: str) → None
save_as_matrix_text_file(self: parselmouth.Matrix, file_path: str) → None
set_value(self: parselmouth.Matrix, row_number: Positive[int], column_number: Positive[int], new_value:
    float) → None

property n_columns
property n_rows
property values

class parselmouth.Pitch
    Bases: TimeFrameSampled, Sampled
    class Candidate
        Bases: pybind11_object
        __init__(*args, **kwargs)
        property frequency
        property strength

    class Frame
        Bases: pybind11_object
        __getitem__(self: parselmouth.Pitch.Frame, i: int) → parselmouth.Pitch.Candidate
        __init__(*args, **kwargs)
        __len__(self: parselmouth.Pitch.Frame) → int
        as_array(self: parselmouth.Pitch.Frame) → numpy.ndarray
        select(self: parselmouth.Pitch.Frame, candidate: parselmouth.Pitch.Candidate) → None
        select(self: parselmouth.Pitch.Frame, i: int) → None
        unvoice(self: parselmouth.Pitch.Frame) → None
        property candidates
        property intensity
        property selected
        __getitem__(self: parselmouth.Pitch, i: int) → parselmouth.Pitch.Frame
        __getitem__(self: parselmouth.Pitch, ij: Tuple[int, int]) → parselmouth.Pitch.Candidate

```

---

`__init__(*args, **kwargs)`

`__iter__(self: parselmouth.Pitch) → Iterator`

`count_differences(self: parselmouth.Pitch, other: parselmouth.Pitch) → str`

`count_voiced_frames(self: parselmouth.Pitch) → int`

`fifth_down(self: parselmouth.Pitch, from_time: float | None = None, to_time: float | None = None) → None`

`fifth_up(self: parselmouth.Pitch, from_time: float | None = None, to_time: float | None = None) → None`

`formula(self: parselmouth.Pitch, formula: str) → None`

`get_frame(self: parselmouth.Pitch, frame_number: Positive[int]) → parselmouth.Pitch.Frame`

`get_mean_absolute_slope(self: parselmouth.Pitch, unit: parselmouth.PitchUnit = <PitchUnit.HERTZ: 0>) → float`

`get_slope_without_octave_jumps(self: parselmouth.Pitch) → float`

`get_value_at_time(self: parselmouth.Pitch, time: float, unit: parselmouth.PitchUnit = <PitchUnit.HERTZ: 0>, interpolation: parselmouth.ValueInterpolation = <ValueInterpolation.LINEAR: 1>) → float`

`get_value_in_frame(self: parselmouth.Pitch, frame_number: int, unit: parselmouth.PitchUnit = <PitchUnit.HERTZ: 0>) → float`

`interpolate(self: parselmouth.Pitch) → parselmouth.Pitch`

`kill_octave_jumps(self: parselmouth.Pitch) → parselmouth.Pitch`

`octave_down(self: parselmouth.Pitch, from_time: float | None = None, to_time: float | None = None) → None`

`octave_up(self: parselmouth.Pitch, from_time: float | None = None, to_time: float | None = None) → None`

`path_finder(self: parselmouth.Pitch, silence_threshold: float = 0.03, voicing_threshold: float = 0.45, octave_cost: float = 0.01, octave_jump_cost: float = 0.35, voiced_unvoiced_cost: float = 0.14, ceiling: Positive[float] = 600.0, pull_formants: bool = False) → None`

`smooth(self: parselmouth.Pitch, bandwidth: Positive[float] = 10.0) → parselmouth.Pitch`

`step(self: parselmouth.Pitch, step: float, precision: Positive[float] = 0.1, from_time: float | None = None, to_time: float | None = None) → None`

`subtract_linear_fit(self: parselmouth.Pitch, unit: parselmouth.PitchUnit = <PitchUnit.HERTZ: 0>) → parselmouth.Pitch`

`to_array(self: parselmouth.Pitch) → numpy.ndarray[parselmouth.Pitch.Candidate]`

`to_matrix(self: parselmouth.Pitch) → parselmouth.Matrix`

`to_sound_hum(self: parselmouth.Pitch, from_time: float | None = None, to_time: float | None = None) → parselmouth.Sound`

`to_sound_pulses(self: parselmouth.Pitch, from_time: float | None = None, to_time: float | None = None) → parselmouth.Sound`

```

to_sound_sine(self: parselmouth.Pitch, from_time: float | None = None, to_time: float | None = None,
    sampling_frequency: Positive[float] = 44100.0, round_to_nearest_zero_crossing: float =
    True) → parselmouth.Sound

unvoice(self: parselmouth.Pitch, from_time: float | None = None, to_time: float | None = None) → None

property ceiling
property max_n_candidates
property selected
property selected_array

class parselmouth.PitchUnit
    Bases: pybind11_object
    __eq__(self: object, other: object) → bool
    __hash__(self: object) → int
    __index__(self: parselmouth.PitchUnit) → int
    __init__(self: parselmouth.PitchUnit, value: int) → None
    __init__(self: parselmouth.PitchUnit, arg0: str) → None
    __int__(self: parselmouth.PitchUnit) → int
    __ne__(self: object, other: object) → bool
    __repr__(self: object) → str
    __str__()
        name(self: handle) -> str
    ERB = <PitchUnit.ERB: 8>
    HERTZ = <PitchUnit.HERTZ: 0>
    HERTZ_LOGARITHMIC = <PitchUnit.HERTZ_LOGARITHMIC: 1>
    LOG_HERTZ = <PitchUnit.LOG_HERTZ: 3>
    MEL = <PitchUnit.MEL: 2>
    SEMITONES_1 = <PitchUnit.SEMITONES_1: 4>
    SEMITONES_100 = <PitchUnit.SEMITONES_100: 5>
    SEMITONES_200 = <PitchUnit.SEMITONES_200: 6>
    SEMITONES_440 = <PitchUnit.SEMITONES_440: 7>
    property name
    property value

class parselmouth.Sampled
    Bases: Function

```

```
__init__(*args, **kwargs)  
__len__(self: parselmouth.Sampled) → int  
x_bins(self: parselmouth.Sampled) → numpy.ndarray[numpy.float64]  
x_grid(self: parselmouth.Sampled) → numpy.ndarray[numpy.float64]  
xs(self: parselmouth.Sampled) → numpy.ndarray[numpy.float64]  
property dx  
property nx  
property x1  
  
class parselmouth.SampledXY  
Bases: Sampled  
__init__(*args, **kwargs)  
y_bins(self: parselmouth.SampledXY) → numpy.ndarray[numpy.float64]  
y_grid(self: parselmouth.SampledXY) → numpy.ndarray[numpy.float64]  
ys(self: parselmouth.SampledXY) → numpy.ndarray[numpy.float64]  
property dy  
property ny  
property y1  
property ymax  
property ymin  
property yrangle  
  
class parselmouth.SignalOutsideTimeDomain  
Bases: pybind11_object  
__eq__(self: object, other: object) → bool  
__hash__(self: object) → int  
__index__(self: parselmouth.SignalOutsideTimeDomain) → int  
__init__(self: parselmouth.SignalOutsideTimeDomain, value: int) → None  
__init__(self: parselmouth.SignalOutsideTimeDomain, arg0: str) → None  
__int__(self: parselmouth.SignalOutsideTimeDomain) → int  
__ne__(self: object, other: object) → bool  
__repr__(self: object) → str  
__str__()  
    name(self: handle) -> str
```

```

SIMILAR = <SignalOutsideTimeDomain.SIMILAR: 2>
ZERO = <SignalOutsideTimeDomain.ZERO: 1>

property name
property value

class parselmouth.Sound
Bases: TimeFrameSampled, Vector

class ToHarmonicityMethod
Bases: pybind11_object

    __eq__(self: object, other: object) → bool
    __hash__(self: object) → int
    __index__(self: parselmouth.Sound.ToHarmonicityMethod) → int
    __init__(self: parselmouth.Sound.ToHarmonicityMethod, value: int) → None
    __init__(self: parselmouth.Sound.ToHarmonicityMethod, arg0: str) → None
    __int__(self: parselmouth.Sound.ToHarmonicityMethod) → int
    __ne__(self: object, other: object) → bool
    __repr__(self: object) → str
    __str__()
        name(self: handle) -> str

AC = <ToHarmonicityMethod.AC: 1>
CC = <ToHarmonicityMethod.CC: 0>
GNE = <ToHarmonicityMethod.GNE: 2>

property name
property value

class ToPitchMethod
Bases: pybind11_object

    __eq__(self: object, other: object) → bool
    __hash__(self: object) → int
    __index__(self: parselmouth.Sound.ToPitchMethod) → int
    __init__(self: parselmouth.Sound.ToPitchMethod, value: int) → None
    __init__(self: parselmouth.Sound.ToPitchMethod, arg0: str) → None
    __int__(self: parselmouth.Sound.ToPitchMethod) → int
    __ne__(self: object, other: object) → bool
    __repr__(self: object) → str

```

```
__str__(self)
    name(self: handle) -> str

AC = <ToPitchMethod.AC: 0>

CC = <ToPitchMethod.CC: 1>

SHS = <ToPitchMethod.SHS: 3>

SPINET = <ToPitchMethod.SPINET: 2>

property name

property value

__init__(self: parselmouth.Sound, other: parselmouth.Sound) → None
__init__(self: parselmouth.Sound, values: numpy.ndarray[numpy.float64], sampling_frequency: Positive[float] = 44100.0, start_time: float = 0.0) → None
__init__(self: parselmouth.Sound, file_path: str) → None

autocorrelate(self: parselmouth.Sound, scaling: parselmouth.AmplitudeScaling =
    <AmplitudeScaling.PEAK_0_99: 4>, signal_outside_time_domain:
    parselmouth.SignalOutsideTimeDomain = <SignalOutsideTimeDomain.ZERO: 1>) →
    parselmouth.Sound

static combine_to_stereo(sounds: List[parselmouth.Sound]) → parselmouth.Sound
static concatenate(sounds: List[parselmouth.Sound], overlap: NonNegative[float] = 0.0) →
    parselmouth.Sound

convert_to_mono(self: parselmouth.Sound) → parselmouth.Sound
convert_to_stereo(self: parselmouth.Sound) → parselmouth.Sound

convolve(self: parselmouth.Sound, other: parselmouth.Sound, scaling: parselmouth.AmplitudeScaling =
    <AmplitudeScaling.PEAK_0_99: 4>, signal_outside_time_domain:
    parselmouth.SignalOutsideTimeDomain = <SignalOutsideTimeDomain.ZERO: 1>) →
    parselmouth.Sound

cross_correlate(self: parselmouth.Sound, other: parselmouth.Sound, scaling:
    parselmouth.AmplitudeScaling = <AmplitudeScaling.PEAK_0_99: 4>,
    signal_outside_time_domain: parselmouth.SignalOutsideTimeDomain =
    <SignalOutsideTimeDomain.ZERO: 1>) → parselmouth.Sound

de_emphasize(self: parselmouth.Sound, from_frequency: float = 50.0, normalize: bool = True) → None
deepen_band_modulation(self: parselmouth.Sound, enhancement: Positive[float] = 20.0, from_frequency:
    Positive[float] = 300.0, to_frequency: Positive[float] = 8000.0,
    slow_modulation: Positive[float] = 3.0, fast_modulation: Positive[float] = 30.0,
    band_smoothing: Positive[float] = 100.0) → parselmouth.Sound

extract_all_channels(self: parselmouth.Sound) → List[parselmouth.Sound]
extract_channel(self: parselmouth.Sound, channel: int) → parselmouth.Sound
extract_channel(self: parselmouth.Sound, arg0: str) → parselmouth.Sound

extract_left_channel(self: parselmouth.Sound) → parselmouth.Sound
```

```
extract_part(self: parselmouth.Sound, from_time: Optional[float] = None, to_time: Optional[float] = None, window_shape: parselmouth.WindowShape = <WindowShape.RECTANGULAR: 0>, relative_width: Positive[float] = 1.0, preserve_times: bool = False) → parselmouth.Sound  
  
extract_part_for_overlap(self: parselmouth.Sound, from_time: Optional[float] = None, to_time: Optional[float] = None, overlap: Positive[float]) → parselmouth.Sound  
  
extract_right_channel(self: parselmouth.Sound) → parselmouth.Sound  
  
get_energy(self: parselmouth.Sound, from_time: float | None = None, to_time: float | None = None) → float  
  
get_energy_in_air(self: parselmouth.Sound) → float  
  
get_index_from_time(self: parselmouth.Sound, time: float) → float  
  
get_intensity(self: parselmouth.Sound) → float  
  
get_nearest_zero_crossing(self: parselmouth.Sound, time: float, channel: int = 1) → float  
  
get_number_of_channels(self: parselmouth.Sound) → int  
  
get_number_of_samples(self: parselmouth.Sound) → int  
  
get_power(self: parselmouth.Sound, from_time: float | None = None, to_time: float | None = None) → float  
  
get_power_in_air(self: parselmouth.Sound) → float  
  
get_rms(self: parselmouth.Sound, from_time: float | None = None, to_time: float | None = None) → float  
  
get_root_mean_square(self: parselmouth.Sound, from_time: float | None = None, to_time: float | None = None) → float  
  
get_sampling_frequency(self: parselmouth.Sound) → float  
  
get_sampling_period(self: parselmouth.Sound) → float  
  
get_time_from_index(self: parselmouth.Sound, sample: int) → float  
  
lengthen(self: parselmouth.Sound, minimum_pitch: Positive[float] = 75.0, maximum_pitch: Positive[float] = 600.0, factor: Positive[float]) → parselmouth.Sound  
  
multiply_by_window(self: parselmouth.Sound, window_shape: parselmouth.WindowShape) → None  
  
override_sampling_frequency(self: parselmouth.Sound, new_frequency: Positive[float]) → None  
  
pre_emphasize(self: parselmouth.Sound, from_frequency: float = 50.0, normalize: bool = True) → None  
  
resample(self: parselmouth.Sound, new_frequency: float, precision: int = 50) → parselmouth.Sound  
  
reverse(self: parselmouth.Sound, from_time: float | None = None, to_time: float | None = None) → None  
  
save(self: parselmouth.Sound, file_path: str, format: parselmouth.SoundFileFormat) → None  
  
scale_intensity(self: parselmouth.Sound, new_average_intensity: float) → None  
  
set_to_zero(self: parselmouth.Sound, from_time: float | None = None, to_time: float | None = None, round_to_nearest_zero_crossing: bool = True) → None
```

```
to_formant_burg(self: parselmouth.Sound, time_step: Positive[float] | None = None,
                  max_number_of_formants: Positive[float] = 5.0, maximum_formant: float = 5500.0,
                  window_length: Positive[float] = 0.025, pre_emphasis_from: Positive[float] = 50.0) →
    parselmouth.Formant

to_harmonicity(self: parselmouth.Sound, method: parselmouth.Sound.ToHarmonicityMethod =
    <ToHarmonicityMethod.CC: 0>, *args, **kwargs) → object

to_harmonicity_ac(self: parselmouth.Sound, time_step: Positive[float] = 0.01, minimum_pitch:
    Positive[float] = 75.0, silence_threshold: float = 0.1, periods_per_window:
    Positive[float] = 1.0) → parselmouth.Harmonicity

to_harmonicity_cc(self: parselmouth.Sound, time_step: Positive[float] = 0.01, minimum_pitch:
    Positive[float] = 75.0, silence_threshold: float = 0.1, periods_per_window:
    Positive[float] = 1.0) → parselmouth.Harmonicity

to_harmonicity_gne(self: parselmouth.Sound, minimum_frequency: Positive[float] = 500.0,
                     maximum_frequency: Positive[float] = 4500.0, bandwidth: Positive[float] = 1000.0,
                     step: Positive[float] = 80.0) → parselmouth.Matrix

to_intensity(self: parselmouth.Sound, minimum_pitch: Positive[float] = 100.0, time_step: Positive[float] |
    None = None, subtract_mean: bool = True) → parselmouth.Intensity

to_mfcc(self: parselmouth.Sound, number_of_coefficients: Positive[int] = 12, window_length:
    Positive[float] = 0.015, time_step: Positive[float] = 0.005, firstFilterFrequency: Positive[float] =
    100.0, distance_between_filters: Positive[float] = 100.0, maximum_frequency: Positive[float] | None
    = None) → parselmouth.MFCC

to_pitch(self: parselmouth.Sound, time_step: Positive[float] | None = None, pitch_floor: Positive[float] =
    75.0, pitch_ceiling: Positive[float] = 600.0) → parselmouth.Pitch

to_pitch(self: parselmouth.Sound, method: parselmouth.Sound.ToPitchMethod, *args, **kwargs) → object

to_pitch_ac(self: parselmouth.Sound, time_step: Positive[float] | None = None, pitch_floor: Positive[float]
    = 75.0, max_number_of_candidates: Positive[int] = 15, very_accurate: bool = False,
    silence_threshold: float = 0.03, voicing_threshold: float = 0.45, octave_cost: float = 0.01,
    octave_jump_cost: float = 0.35, voiced_unvoiced_cost: float = 0.14, pitch_ceiling:
    Positive[float] = 600.0) → parselmouth.Pitch

to_pitch_cc(self: parselmouth.Sound, time_step: Positive[float] | None = None, pitch_floor: Positive[float]
    = 75.0, max_number_of_candidates: Positive[int] = 15, very_accurate: bool = False,
    silence_threshold: float = 0.03, voicing_threshold: float = 0.45, octave_cost: float = 0.01,
    octave_jump_cost: float = 0.35, voiced_unvoiced_cost: float = 0.14, pitch_ceiling:
    Positive[float] = 600.0) → parselmouth.Pitch

to_pitch_shs(self: parselmouth.Sound, time_step: Positive[float] = 0.01, minimum_pitch: Positive[float] =
    50.0, max_number_of_candidates: Positive[int] = 15, maximum_frequency_component:
    Positive[float] = 1250.0, max_number_of_subharmonics: Positive[int] = 15,
    compression_factor: Positive[float] = 0.84, ceiling: Positive[float] = 600.0,
    number_of_points_per_octave: Positive[int] = 48) → parselmouth.Pitch

to_pitch_spinet(self: parselmouth.Sound, time_step: Positive[float] = 0.005, window_length:
    Positive[float] = 0.04, minimum_filter_frequency: Positive[float] = 70.0,
    maximum_filter_frequency: Positive[float] = 5000.0, number_of_filters: Positive[int] =
    250, ceiling: Positive[float] = 500.0, max_number_of_candidates: Positive[int] = 15) →
    parselmouth.Pitch
```

```

to_spectrogram(self: parselmouth.Sound, window_length: Positive[float] = 0.005, maximum_frequency:
    Positive[float] = 5000.0, time_step: Positive[float] = 0.002, frequency_step: Positive[float]
    = 20.0, window_shape: parselmouth.SpectralAnalysisWindowShape =
    <SpectralAnalysisWindowShape.GAUSSIAN: 5>) → parselmouth.Spectrogram

to_spectrum(self: parselmouth.Sound, fast: bool = True) → parselmouth.Spectrum

property n_channels
property n_samples
property sampling_frequency
property sampling_period

class parselmouth.SoundFileFormat
    Bases: pybind11_object

    _eq_(self: object, other: object) → bool
    _hash_(self: object) → int
    _index_(self: parselmouth.SoundFileFormat) → int
    _init_(self: parselmouth.SoundFileFormat, value: int) → None
    _init_(self: parselmouth.SoundFileFormat, arg0: str) → None
    _int_(self: parselmouth.SoundFileFormat) → int
    _ne_(self: object, other: object) → bool
    _repr_(self: object) → str
    _str_()
        name(self: handle) -> str
    AIFC = <SoundFileFormat.AIFC: 2>
    AIFF = <SoundFileFormat.AIFF: 1>
    FLAC = <SoundFileFormat.FLAC: 5>
    KAY = <SoundFileFormat.KAY: 6>
    NEXT_SUN = <SoundFileFormat.NEXT_SUN: 3>
    NIST = <SoundFileFormat.NIST: 4>
    RAW_16_BE = <SoundFileFormat.RAW_16_BE: 12>
    RAW_16_LE = <SoundFileFormat.RAW_16_LE: 13>
    RAW_24_BE = <SoundFileFormat.RAW_24_BE: 14>
    RAW_24_LE = <SoundFileFormat.RAW_24_LE: 15>
    RAW_32_BE = <SoundFileFormat.RAW_32_BE: 16>
    RAW_32_LE = <SoundFileFormat.RAW_32_LE: 17>

```

```
RAW_8_SIGNED = <SoundFileFormat.RAW_8_SIGNED: 10>
RAW_8_UNSIGNED = <SoundFileFormat.RAW_8_UNSIGNED: 11>
SESAM = <SoundFileFormat.SESAM: 7>
WAV = <SoundFileFormat.WAV: 0>
WAV_24 = <SoundFileFormat.WAV_24: 8>
WAV_32 = <SoundFileFormat.WAV_32: 9>

property name
property value

class parselmouth.SpectralAnalysisWindowShape
Bases: pybind11_object
__eq__(self: object, other: object) → bool
__hash__(self: object) → int
__index__(self: parselmouth.SpectralAnalysisWindowShape) → int
__init__(self: parselmouth.SpectralAnalysisWindowShape, value: int) → None
__init__(self: parselmouth.SpectralAnalysisWindowShape, arg0: str) → None
__int__(self: parselmouth.SpectralAnalysisWindowShape) → int
__ne__(self: object, other: object) → bool
__repr__(self: object) → str
__str__()
    name(self: handle) -> str
BARTLETT = <SpectralAnalysisWindowShape.BARTLETT: 2>
GAUSSIAN = <SpectralAnalysisWindowShape.GAUSSIAN: 5>
HAMMING = <SpectralAnalysisWindowShape.HAMMING: 1>
HANNING = <SpectralAnalysisWindowShape.HANNING: 4>
SQUARE = <SpectralAnalysisWindowShape.SQUARE: 0>
WELCH = <SpectralAnalysisWindowShape.WELCH: 3>

property name
property value

class parselmouth.Spectrogram
Bases: TimeFrameSampled, Matrix
__init__(*args, **kwargs)
get_power_at(self: parselmouth.Spectrogram, time: float, frequency: float) → float
```

---

```

synthesize_sound(self: parselmouth.Spectrogram, sampling_frequency: Positive[float] = 44100.0) →
    parselmouth.Sound

to_sound(self: parselmouth.Spectrogram, sampling_frequency: Positive[float] = 44100.0) →
    parselmouth.Sound

to_spectrum_slice(self: parselmouth.Spectrogram, time: float) → parselmouth.Spectrum

class parselmouth.Spectrum
    Bases: Matrix

    __getitem__(self: parselmouth.Spectrum, index: int) → complex

    __init__(self: parselmouth.Spectrum, values: numpy.ndarray[numpy.float64], maximum_frequency:
        Positive[float]) → None

    __init__(self: parselmouth.Spectrum, values: numpy.ndarray[numpy.complex128], maximum_frequency:
        Positive[float]) → None

    __setitem__(self: parselmouth.Spectrum, index: int, value: complex) → None

    cepstral_smoothing(self: parselmouth.Spectrum, bandwidth: Positive[float] = 500.0) →
        parselmouth.Spectrum

    get_band_density(self: parselmouth.Spectrum, band_floor: float | None = None, band_ceiling: float | None
        = None) → float

    get_band_density(self: parselmouth.Spectrum, band: Tuple[float | None, float | None] = (None, None)) →
        float

    get_band_density_difference(self: parselmouth.Spectrum, low_band_floor: float | None = None,
        low_band_ceiling: float | None = None, high_band_floor: float | None =
        None, high_band_ceiling: float | None = None) → float

    get_band_density_difference(self: parselmouth.Spectrum, low_band: Tuple[float | None, float | None] =
        (None, None), high_band: Tuple[float | None, float | None] = (None, None)) → float

    get_band_energy(self: parselmouth.Spectrum, band_floor: float | None = None, band_ceiling: float | None
        = None) → float

    get_band_energy(self: parselmouth.Spectrum, band: Tuple[float | None, float | None] = (None, None)) →
        float

    get_band_energy_difference(self: parselmouth.Spectrum, low_band_floor: float | None = None,
        low_band_ceiling: float | None = None, high_band_floor: float | None =
        None, high_band_ceiling: float | None = None) → float

    get_band_energy_difference(self: parselmouth.Spectrum, low_band: Tuple[float | None, float | None] =
        (None, None), high_band: Tuple[float | None, float | None] = (None, None)) → float

    get_bin_number_from_frequency(self: parselmouth.Spectrum, frequency: float) → float

    get_bin_width(self: parselmouth.Spectrum) → float

    get_center_of_gravity(self: parselmouth.Spectrum, power: Positive[float] = 2.0) → float

    get_central_moment(self: parselmouth.Spectrum, moment: Positive[float], power: Positive[float] = 2.0)
        → float

```

---

```
get_centre_of_gravity(self: parselmouth.Spectrum, power: Positive[float] = 2.0) → float
get_frequency_from_bin_number(self: parselmouth.Spectrum, band_number: Positive[int]) → float
get_highest_frequency(self: parselmouth.Spectrum) → float
get_imaginary_value_in_bin(self: parselmouth.Spectrum, bin_number: Positive[int]) → float
get_kurtosis(self: parselmouth.Spectrum, power: Positive[float] = 2.0) → float
get_lowest_frequency(self: parselmouth.Spectrum) → float
get_number_of_bins(self: parselmouth.Spectrum) → int
get_real_value_in_bin(self: parselmouth.Spectrum, bin_number: Positive[int]) → float
get_skewness(self: parselmouth.Spectrum, power: Positive[float] = 2.0) → float
get_standard_deviation(self: parselmouth.Spectrum, power: Positive[float] = 2.0) → float
get_value_in_bin(self: parselmouth.Spectrum, bin_number: Positive[int]) → complex
lpc_smoothing(self: parselmouth.Spectrum, num_peaks: Positive[int] = 5, pre_emphasis_from: Positive[float] = 50.0) → parselmouth.Spectrum
set_imaginary_value_in_bin(self: parselmouth.Spectrum, bin_number: Positive[int], value: float) → None
set_real_value_in_bin(self: parselmouth.Spectrum, bin_number: Positive[int], value: float) → None
set_value_in_bin(self: parselmouth.Spectrum, bin_number: Positive[int], value: complex) → None
to_sound(self: parselmouth.Spectrum) → parselmouth.Sound
to_spectrogram(self: parselmouth.Spectrum) → parselmouth.Spectrogram
property bin_width
property df
property fmax
property fmin
property highest_frequency
property lowest_frequency
property n_bins
property nf

class parselmouth.TextGrid
Bases: Function

__init__(self: parselmouth.TextGrid, start_time: float, end_time: float, tier_names: str, point_tier_names: str) → None
__init__(self: parselmouth.TextGrid, start_time: float, end_time: float, tier_names: List[str] = [], point_tier_names: List[str] = []) → None
```

```
__init__(self: parselmouth.TextGrid, tgt_text_grid: tgt.core.TextGrid) → None
static from_tgt(tgt_text_grid: tgt.core.TextGrid) → parselmouth.TextGrid
to_tgt(self: parselmouth.TextGrid, *, include_empty_intervals: bool = False) → tgt.core.TextGrid

class parselmouth.Thing
    Bases: pybind11_object
        __init__(*args, **kwargs)
        __str__(self: parselmouth.Thing) → str
        info(self: parselmouth.Thing) → str
        property class_name
        property full_name
        property name

class parselmouth.TimeFrameSampled
    Bases: TimeFunction, Sampled
        __init__(*args, **kwargs)
        frame_number_to_time(self: parselmouth.Sampled, frame_number: Positive[int]) → float
        get_frame_number_from_time(self: parselmouth.Sampled, time: float) → float
        get_number_of_frames(self: parselmouth.Sampled) → int
        get_time_from_frame_number(self: parselmouth.Sampled, frame_number: Positive[int]) → float
        get_time_step(self: parselmouth.Sampled) → float
        t_bins(self: parselmouth.Sampled) → numpy.ndarray[numpy.float64]
        t_grid(self: parselmouth.Sampled) → numpy.ndarray[numpy.float64]
        time_to_frame_number(self: parselmouth.Sampled, time: float) → float
        ts(self: parselmouth.Sampled) → numpy.ndarray[numpy.float64]
        property dt
        property n_frames
        property nt
        property t1
        property time_step

class parselmouth.TimeFunction
    Bases: Function
        __init__(*args, **kwargs)
        get_end_time(self: parselmouth.Function) → float
```

```
get_start_time(self: parselmouth.Function) → float
get_total_duration(self: parselmouth.Function) → float
scale_times_by(self: parselmouth.Function, scale: Positive[float]) → None
scale_times_to(self: parselmouth.Function, new_start_time: float, new_end_time: float) → None
shift_times_by(self: parselmouth.Function, seconds: float) → None
shift_times_to(self: parselmouth.Function, time: float, new_time: float) → None
shift_times_to(self: parselmouth.Function, time: str, new_time: float) → None

property centre_time
property duration
property end_time
property start_time
property time_range
property tmax
property tmin
property total_duration
property trange

class parselmouth.ValueInterpolation
    Bases: pybind11_object
    __eq__(self: object, other: object) → bool
    __hash__(self: object) → int
    __index__(self: parselmouth.ValueInterpolation) → int
    __init__(self: parselmouth.ValueInterpolation, value: int) → None
    __init__(self: parselmouth.ValueInterpolation, arg0: str) → None
    __int__(self: parselmouth.ValueInterpolation) → int
    __ne__(self: object, other: object) → bool
    __repr__(self: object) → str
    __str__()
        name(self: handle) -> str
CUBIC = <ValueInterpolation.CUBIC: 2>
LINEAR = <ValueInterpolation.LINEAR: 1>
NEAREST = <ValueInterpolation.NEAREST: 0>
SINC70 = <ValueInterpolation.SINC70: 3>
```

```

SINC700 = <ValueInterpolation.SINC700: 4>

property name
property value

class parselmouth.Vector
    Bases: Matrix

        __add__(self: parselmouth.Vector, number: float) → parselmouth.Vector
        __iadd__(self: parselmouth.Vector, number: float) → parselmouth.Vector
        __imul__(self: parselmouth.Vector, factor: float) → parselmouth.Vector
        __init__(*args, **kwargs)
        __isub__(self: parselmouth.Vector, number: float) → parselmouth.Vector
        __itruediv__(self: parselmouth.Vector, factor: float) → parselmouth.Vector
        __mul__(self: parselmouth.Vector, factor: float) → parselmouth.Vector
        __radd__(self: parselmouth.Vector, number: float) → parselmouth.Vector
        __rmul__(self: parselmouth.Vector, factor: float) → parselmouth.Vector
        __sub__(self: parselmouth.Vector, number: float) → parselmouth.Vector
        __truediv__(self: parselmouth.Vector, factor: float) → parselmouth.Vector
        add(self: parselmouth.Vector, number: float) → None
        divide(self: parselmouth.Vector, factor: float) → None
        get_value(self: parselmouth.Vector, x: float, channel: Optional[int] = None, interpolation: parselmouth.ValueInterpolation = <ValueInterpolation.CUBIC: 2>) → float
        multiply(self: parselmouth.Vector, factor: float) → None
        scale(self: parselmouth.Vector, scale: Positive[float]) → None
        scale_peak(self: parselmouth.Vector, new_peak: Positive[float] = 0.99) → None
        subtract(self: parselmouth.Vector, number: float) → None
        subtract_mean(self: parselmouth.Vector) → None

class parselmouth.WindowShape
    Bases: pybind11_object

        __eq__(self: object, other: object) → bool
        __hash__(self: object) → int
        __index__(self: parselmouth.WindowShape) → int
        __init__(self: parselmouth.WindowShape, value: int) → None
        __init__(self: parselmouth.WindowShape, arg0: str) → None

```

```
__int__(self: parselmouth.WindowShape) → int
__ne__(self: object, other: object) → bool
__repr__(self: object) → str
__str__()

name(self: handle) -> str

GAUSSIAN1 = <WindowShape.GAUSSIAN1: 5>
GAUSSIAN2 = <WindowShape.GAUSSIAN2: 6>
GAUSSIAN3 = <WindowShape.GAUSSIAN3: 7>
GAUSSIAN4 = <WindowShape.GAUSSIAN4: 8>
GAUSSIAN5 = <WindowShape.GAUSSIAN5: 9>

HAMMING = <WindowShape.HAMMING: 4>
HANNING = <WindowShape.HANNING: 3>
KAISER1 = <WindowShape.KAISER1: 10>
KAISER2 = <WindowShape.KAISER2: 11>
PARABOLIC = <WindowShape.PARABOLIC: 2>

RECTANGULAR = <WindowShape.RECTANGULAR: 0>
TRIANGULAR = <WindowShape.TRIANGULAR: 1>

property name
property value
```

`parselmouth.read(file_path: str) → parselmouth.Data`

Read a file into a `parselmouth.Data` object.

**Parameters**

`file_path (str)` – The path of the file on disk to read.

**Returns**

The Praat Data object that was read.

**Return type**

`parselmouth.Data`

**See also:**

*Praat: “Read from file...”*

`parselmouth.praat.call(command: str, *args, **kwargs) → object`

`parselmouth.praat.call(object: parselmouth.Data, command: str, *args, **kwargs) → object`

`parselmouth.praat.call(objects: List[parselmouth.Data], command: str, *args, **kwargs) → object`

Call a Praat command.

This function provides a Python interface to call available Praat commands based on the label in the Praat user interface and documentation, similar to the Praat scripting language.

Calling a Praat command through this function roughly corresponds to the following scenario in the Praat user interface or scripting language:

1. Zero, one, or multiple `parselmouth.Data` objects are put into Praat's global object list and are 'selected'.
2. The Python argument values are converted into Praat values; see below.
3. The Praat command is executed on the selected objects with the converted values as arguments.
4. The result of the command is returned. The type of the result depends on the result of the Praat command; see below.
5. Praat's object list is emptied again, such that a future execution of this function is independent from the current call.

The use of `call` is demonstrated in the *Pitch manipulation and Praat commands* example.

#### Parameters

- **object** (`parselmouth.Data`) – A single object to add to the Praat object list, which will be selected when the Praat command is called.
- **objects** (`List[parselmouth.Data]`) – Multiple objects to be added to the Praat object list, which will be selected when the Praat command is called.
- **command** (`str`) – The Praat action to call. This is the same command name as one would use in a Praat script and corresponds to the label on the button in the Praat user interface.
- **\*args** – The list of values to be passed as arguments to the Praat command. Allowed types for these arguments are:
  - `int` or `float`: passed as a Praat numeric value
  - `bool`: converted into "yes"/"no"
  - `str`: passed as Praat string value
  - `numpy.ndarray`: passed as Praat vector or matrix, if the array contains numeric values and is 1D or 2D, respectively.

#### Keyword Arguments

- **extra\_objects** (`List[parselmouth.Data]`) – Extra objects added to the Praat object list that will not be selected when the command is called (default value: `[]`).
- **return\_string** (`bool`) – Return the raw string written in the Praat info window instead of the converted Python object (default value: `False`).

#### Returns

The result of the Praat command. The actual value returned depends on what the Praat command does. The following types can be returned:

- If `return_string=True` was passed, a `str` value is returned, which contains the text that would have been written to the Praat info window.
- A `float`, `int`, `bool`, or `complex` value is returned when the Praat command would write such a value to the Praat info window.
- A `numpy.ndarray` value is returned if the command returns a Praat vector or matrix.
- A `parselmouth.Data` object is returned if the command always creates exactly one object. If the actual type of the Praat object is available in Parselmouth, an object of a subtype of `parselmouth.Data` is returned.
- A list of `parselmouth.Data` objects is returned if the command can create multiple new objects (even if this particular execution of the command only added one object to the Praat object list).

- A `str` is returned when a string or info text would be written to the Praat info window.

**Return type**

`object`

**See also:**

`parselmouth.praat.run`, `parselmouth.praat.run_file`, *Praat: “Scripting”*

`parselmouth.praat.run(script: str, *args, **kwargs) → object`

`parselmouth.praat.run(object: parselmouth.Data, script: str, *args, **kwargs) → object`

`parselmouth.praat.run(objects: List[parselmouth.Data], script: str, *args, **kwargs) → object`

Run a Praat script.

Given a string with the contents of a Praat script, run this script as if it was run inside Praat itself. Similarly to `parselmouth.praat.call`, Parselmouth objects and Python argument values can be passed into the script.

Calling this function roughly corresponds to the following sequence of steps in Praat:

1. Zero, one, or multiple `parselmouth.Data` objects are put into Praat’s global object list and are ‘selected’.
2. The Python argument values are converted into Praat values; see `call`.
3. The Praat script is opened and run with the converted values as arguments; see *Praat: “Scripting 6.1. Arguments to the script”*.
4. The results of the execution of the script are returned; see below.
5. Praat’s object list is emptied again, such that a future execution of this function is independent from the current call.

Note that the script will be run in Praat’s so-called ‘batch’ mode; see *Praat: “Scripting 6.9. Calling from the command line”*. Since the script is run from inside a Python program, the Praat functionality is run without graphical user interface and no windows (such as “View & Edit”) can be opened by the Praat script. However, the functionality in these windows is also available in different ways: for example, opening a `Sound` object in a “View & Edit” window, making a selection, and choosing “Extract selected sound (windowed)...” can also be achieved by directly using the “Extract part...” command of the `Sound` object.

**Parameters**

- **object** (`parselmouth.Data`) – A single object to add to the Praat object list, which will be selected when the Praat script is run.
- **objects** (`List[parselmouth.Data]`) – Multiple objects to be added to the Praat object list, which will be selected when the Praat script is run.
- **script** (`str`) – The content of the Praat script to be run.
- **\*args** – The list of values to be passed as arguments to the Praat script. For more details on the allowed types of these argument, see `call`.

**Keyword Arguments**

- **extra\_objects** (`List[parselmouth.Data]`) – Extra objects added to the Praat object list that will not be selected when the command is called (default value: `[]`).
- **capture\_output** (`bool`) – Intercept and also return the output written to the Praat info window, instead of forwarding it to the Python standard output; see below (default value: `False`).
- **return\_variables** (`bool`) – Also return a `dict` of the Praat variables and their values at the end of the script’s execution; see below (default value: `False`).

## Returns

A list of `parselmouth.Data` objects selected at the end of the script's execution.

Optionally, extra values are returned:

- A `str` containing the intercepted output if `capture_output=True` was passed.
- A `dict` mapping variable names (`str`) to their values (`object`) if `return_variables` is `True`. The values of Praat's variables get converted to Python values:
  - A Praat string variable, with a name ending in `$`, is returned as `str` value.
  - A Praat vector or matrix variable, respectively ending in `#` or `##`, is returned as `numpy.ndarray`.
  - A numeric variable, without variable name suffix, is converted to a Python `float`.

## Return type

`object`

## See also:

`parselmouth.praat.run_file`, `parselmouth.praat.call`, *Praat: “Scripting”*

`parselmouth.praat.run_file(path: str, *args, **kwargs) → object`

`parselmouth.praat.run_file(object: parselmouth.Data, path: str, *args, **kwargs) → object`

`parselmouth.praat.run_file(objects: List[parselmouth.Data], path: str, *args, **kwargs) → object`

Run a Praat script from file.

Given the filename of a Praat script, the script is read and run the same way as a script string passed to `parselmouth.praat.run`. See `run` for details on the manner in which the script gets executed.

One thing to note is that relative filenames in the Praat script (including those in potential ‘include’ statements in the script; see *Praat: “Scripting 5.8. Including other scripts”*) will be resolved relative to the path of the script file, just like in Praat. Also note that Praat accomplishes this by temporarily changing the current working during the execution of the script.

## Parameters

- **object** (`parselmouth.Data`) – A single object to add to the Praat object list, which will be selected when the Praat script is run.
- **objects** (`List[parselmouth.Data]`) – Multiple objects to be added to the Praat object list, which will be selected when the Praat script is run.
- **path** (`str`) – The filename of the Praat script to run.
- **\*args** – The list of values to be passed as arguments to the Praat script. For more details on the allowed types of these argument, see `call`.

## Keyword Arguments

- **keep\_cwd** (`bool`) – Keep the current working directory (see `os.getcwd`) when running the script, rather than changing it to the script's parent directory, as Praat does by default (default value: `False`). Note that even when set to `True`, the filenames in the Praat script's include statements will be resolved relatively to the directory containing the script.
- **\*\*kwargs** – See `parselmouth.praat.run`.

## Returns

See `parselmouth.praat.run`.

## Return type

`object`

**See also:**

`parselmouth.praat.run`, `parselmouth.praat.call`, `Praat: “Scripting”`

---

**CHAPTER  
FOUR**

---

## CITING PARSELMOUTH

A manuscript introducing **Parselmouth** (and supplementary material) has been published in the *Journal of Phonetics*. Scientific work and publications can now cite **Parselmouth** in the following way:

Jadoul, Y., Thompson, B., & de Boer, B. (2018). Introducing **Parselmouth**: A Python interface to Praat. *Journal of Phonetics*, 71, 1-15. <https://doi.org/10.1016/j.wocn.2018.07.001>

```
@article{parselmouth,
  author = "Yannick Jadoul and Bill Thompson and Bart de Boer",
  title = "Introducing {P}arselmouth: A {P}ython interface to {P}raat",
  journal = "Journal of Phonetics",
  volume = "71",
  pages = "1--15",
  year = "2018",
  doi = "https://doi.org/10.1016/j.wocn.2018.07.001"
}
```

Since **Parselmouth** exposes existing Praat functionality and algorithm implementations, we suggest also citing Praat when using **Parselmouth** in scientific research:

Boersma, P., & Weenink, D. (2021). Praat: doing phonetics by computer [Computer program]. Version 6.1.38, retrieved 2 January 2021 from <http://www.praat.org/>

```
@misc{praat,
  author = "Paul Boersma and David Weenink",
  title = "{P}raat: doing phonetics by computer [{C}omputer program]",
  howpublished = "Version 6.1.38, retrieved 2 January 2021 \url{http://www.praat.org/}",
  year = "2021"
}
```



---

**CHAPTER  
FIVE**

---

**INDICES AND TABLES**

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### p

`parselmouth`, 27

`parselmouth.praat`, 48



# INDEX

## Symbols

`__add__()` (*parselmouth.Vector* method), 47  
`__copy__()` (*parselmouth.Data* method), 29  
`__deepcopy__()` (*parselmouth.Data* method), 29  
`__eq__()` (*parselmouth.AmplitudeScaling* method), 27  
`__eq__()` (*parselmouth.Data* method), 29  
`__eq__()` (*parselmouth.Data.FileFormat* method), 28  
`__eq__()` (*parselmouth.FormantUnit* method), 30  
`__eq__()` (*parselmouth.Intensity.AveragingMethod* method), 31  
`__eq__()` (*parselmouth.PitchUnit* method), 35  
`__eq__()` (*parselmouth.SignalOutsideTimeDomain* method), 36  
`__eq__()` (*parselmouth.Sound.ToHarmonicityMethod* method), 37  
`__eq__()` (*parselmouth.Sound.ToPitchMethod* method), 37  
`__eq__()` (*parselmouth.SoundFileFormat* method), 41  
`__eq__()` (*parselmouth.SpectralAnalysisWindowShape* method), 42  
`__eq__()` (*parselmouth.ValueInterpolation* method), 46  
`__eq__()` (*parselmouth.WindowShape* method), 47  
`__getitem__()` (*parselmouth.CC* method), 28  
`__getitem__()` (*parselmouth.CC.Frame* method), 28  
`__getitem__()` (*parselmouth.Pitch* method), 33  
`__getitem__()` (*parselmouth.Pitch.Frame* method), 33  
`__getitem__()` (*parselmouth.Spectrum* method), 43  
`__hash__()` (*parselmouth.Data* attribute), 30  
`__hash__()` (*parselmouth.AmplitudeScaling* method), 27  
`__hash__()` (*parselmouth.Data.FileFormat* method), 28  
`__hash__()` (*parselmouth.FormantUnit* method), 30  
`__hash__()` (*parselmouth.Intensity.AveragingMethod* method), 31  
`__hash__()` (*parselmouth.PitchUnit* method), 35  
`__hash__()` (*parselmouth.SignalOutsideTimeDomain* method), 36  
`__hash__()` (*parselmouth.Sound.ToHarmonicityMethod* method), 37  
`__hash__()` (*parselmouth.Sound.ToPitchMethod* method), 37  
`__hash__()` (*parselmouth.SoundFileFormat* method), 41  
`__hash__()` (*parselmouth.SpectralAnalysisWindowShape* method), 42  
`__hash__()` (*parselmouth.ValueInterpolation* method), 46  
`__hash__()` (*parselmouth.WindowShape* method), 47  
`__iadd__()` (*parselmouth.Vector* method), 47  
`__imul__()` (*parselmouth.Vector* method), 47  
`__index__()` (*parselmouth.AmplitudeScaling* method), 27  
`__index__()` (*parselmouth.Data.FileFormat* method), 29  
`__index__()` (*parselmouth.FormantUnit* method), 30  
`__index__()` (*parselmouth.Intensity.AveragingMethod* method), 31  
`__index__()` (*parselmouth.PitchUnit* method), 35  
`__index__()` (*parselmouth.SignalOutsideTimeDomain* method), 36  
`__index__()` (*parselmouth.Sound.ToHarmonicityMethod* method), 37  
`__index__()` (*parselmouth.Sound.ToPitchMethod* method), 37  
`__index__()` (*parselmouth.SoundFileFormat* method), 41  
`__index__()` (*parselmouth.SpectralAnalysisWindowShape* method), 42  
`__index__()` (*parselmouth.ValueInterpolation* method), 46  
`__index__()` (*parselmouth.WindowShape* method), 47  
`__init__()` (*parselmouth.AmplitudeScaling* method), 27  
`__init__()` (*parselmouth.CC* method), 28  
`__init__()` (*parselmouth.CC.Frame* method), 28  
`__init__()` (*parselmouth.Data* method), 29  
`__init__()` (*parselmouth.Data.FileFormat* method), 29  
`__init__()` (*parselmouth.Formant* method), 30  
`__init__()` (*parselmouth.FormantUnit* method), 30  
`__init__()` (*parselmouth.Function* method), 30  
`__init__()` (*parselmouth.Harmonicity* method), 31  
`__init__()` (*parselmouth.Intensity* method), 31  
`__init__()` (*parselmouth.Intensity.AveragingMethod* method), 31  
`__init__()` (*parselmouth.MFCC* method), 32

`__init__(parselmouth.Matrix method), 32`  
`__init__(parselmouth.Pitch method), 33`  
`__init__(parselmouth.Pitch.Candidate method), 33`  
`__init__(parselmouth.Pitch.Frame method), 33`  
`__init__(parselmouth.PitchUnit method), 35`  
`__init__(parselmouth.Sampled method), 35`  
`__init__(parselmouth.SampledXY method), 36`  
`__init__(parselmouth.SignalOutsideTimeDomain method), 36`  
`__init__(parselmouth.Sound method), 38`  
`__init__(parselmouth.Sound.ToHarmonicityMethod method), 37`  
`__init__(parselmouth.Sound.ToPitchMethod method), 37`  
`__init__(parselmouth.SoundFormat method), 41`  
`__init__(parselmouth.SpectralAnalysisWindowShape method), 42`  
`__init__(parselmouth.Spectrogram method), 42`  
`__init__(parselmouth.Spectrum method), 43`  
`__init__(parselmouth.TextGrid method), 44`  
`__init__(parselmouth.Thing method), 45`  
`__init__(parselmouth.TimeFrameSampled method), 45`  
`__init__(parselmouth.TimeFunction method), 45`  
`__init__(parselmouth.ValueInterpolation method), 46`  
`__init__(parselmouth.Vector method), 47`  
`__init__(parselmouth.WindowShape method), 47`  
`__int__(parselmouth.AmplitudeScaling method), 27`  
`__int__(parselmouth.Data.Format method), 29`  
`__int__(parselmouth.FormantUnit method), 30`  
`__int__(parselmouth.Intensity.AveragingMethod method), 31`  
`__int__(parselmouth.PitchUnit method), 35`  
`__int__(parselmouth.SignalOutsideTimeDomain method), 36`  
`__int__(parselmouth.Sound.ToHarmonicityMethod method), 37`  
`__int__(parselmouth.Sound.ToPitchMethod method), 37`  
`__int__(parselmouth.SoundFormat method), 41`  
`__int__(parselmouth.SpectralAnalysisWindowShape method), 42`  
`__int__(parselmouth.ValueInterpolation method), 46`  
`__int__(parselmouth.WindowShape method), 47`  
`__isub__(parselmouth.Vector method), 47`  
`__iter__(parselmouth.CC method), 28`  
`__iter__(parselmouth.Pitch method), 34`  
`__itruediv__(parselmouth.Vector method), 47`  
`__len__(parselmouth.CC.Frame method), 28`  
`__len__(parselmouth.Pitch.Frame method), 33`  
`__len__(parselmouth.Sampled method), 36`  
`__mul__(parselmouth.Vector method), 47`  
`__ne__(parselmouth.AmplitudeScaling method), 27`  
`__ne__(parselmouth.Data method), 29`  
`__ne__(parselmouth.Data.Format method), 29`  
`__ne__(parselmouth.FormantUnit method), 30`  
`__ne__(parselmouth.Intensity.AveragingMethod method), 31`  
`__ne__(parselmouth.PitchUnit method), 35`  
`__ne__(parselmouth.SignalOutsideTimeDomain method), 36`  
`__ne__(parselmouth.Sound.ToHarmonicityMethod method), 37`  
`__ne__(parselmouth.Sound.ToPitchMethod method), 37`  
`__ne__(parselmouth.SoundFormat method), 41`  
`__ne__(parselmouth.SpectralAnalysisWindowShape method), 42`  
`__ne__(parselmouth.ValueInterpolation method), 46`  
`__ne__(parselmouth.WindowShape method), 48`  
`__radd__(parselmouth.Vector method), 47`  
`__repr__(parselmouth.AmplitudeScaling method), 27`  
`__repr__(parselmouth.Data.Format method), 29`  
`__repr__(parselmouth.FormantUnit method), 30`  
`__repr__(parselmouth.Intensity.AveragingMethod method), 31`  
`__repr__(parselmouth.PitchUnit method), 35`  
`__repr__(parselmouth.SignalOutsideTimeDomain method), 36`  
`__repr__(parselmouth.Sound.ToHarmonicityMethod method), 37`  
`__repr__(parselmouth.Sound.ToPitchMethod method), 37`  
`__repr__(parselmouth.SoundFormat method), 41`  
`__repr__(parselmouth.SpectralAnalysisWindowShape method), 42`  
`__repr__(parselmouth.ValueInterpolation method), 46`  
`__repr__(parselmouth.WindowShape method), 48`  
`__rmul__(parselmouth.Vector method), 47`  
`__setitem__(parselmouth.CC method), 28`  
`__setitem__(parselmouth.CC.Frame method), 28`  
`__setitem__(parselmouth.Spectrum method), 43`  
`__str__(parselmouth.AmplitudeScaling method), 27`  
`__str__(parselmouth.Data.Format method), 29`  
`__str__(parselmouth.FormantUnit method), 30`  
`__str__(parselmouth.Intensity.AveragingMethod method), 31`  
`__str__(parselmouth.PitchUnit method), 35`  
`__str__(parselmouth.SignalOutsideTimeDomain method), 36`  
`__str__(parselmouth.Sound.ToHarmonicityMethod method), 37`  
`__str__(parselmouth.Sound.ToPitchMethod method), 37`  
`__str__(parselmouth.SoundFormat method), 41`

`__str__()` (*parselmouth.SpectralAnalysisWindowShape method*), 42  
`__str__()` (*parselmouth.Thing method*), 45  
`__str__()` (*parselmouth.ValueInterpolation method*), 46  
`__str__()` (*parselmouth.WindowShape method*), 48  
`__sub__()` (*parselmouth.Vector method*), 47  
`__truediv__()` (*parselmouth.Vector method*), 47

## A

`AC` (*parselmouth.Sound.ToHarmonicityMethod attribute*), 37  
`AC` (*parselmouth.Sound.ToPitchMethod attribute*), 38  
`add()` (*parselmouth.Vector method*), 47  
`AIFC` (*parselmouth.SoundFormat attribute*), 41  
`AIFF` (*parselmouth.SoundFormat attribute*), 41  
`AmplitudeScaling` (*class in parselmouth*), 27  
`as_array()` (*parselmouth.Matrix method*), 32  
`as_array()` (*parselmouth.Pitch.Frame method*), 33  
`at_xy()` (*parselmouth.Matrix method*), 32  
`autocorrelate()` (*parselmouth.Sound method*), 38

## B

`BARK` (*parselmouth.FormantUnit attribute*), 30  
`BARTLETT` (*parselmouth.SpectralAnalysisWindowShape attribute*), 42  
`bin_width` (*parselmouth.Spectrum property*), 44  
`BINARY` (*parselmouth.Data.FileFormat attribute*), 29

## C

`c` (*parselmouth.CC.Frame property*), 28  
`c0` (*parselmouth.CC.Frame property*), 28  
`call()` (*in module parselmouth.praat*), 48  
`candidates` (*parselmouth.Pitch.Frame property*), 33  
`CC` (*class in parselmouth*), 28  
`CC` (*parselmouth.Sound.ToHarmonicityMethod attribute*), 37  
`CC` (*parselmouth.Sound.ToPitchMethod attribute*), 38  
`CC.Frame` (*class in parselmouth*), 28  
`ceiling` (*parselmouth.Pitch property*), 35  
`centre_time` (*parselmouth.TimeFunction property*), 46  
`cepstral_smoothing()` (*parselmouth.Spectrum method*), 43  
`class_name` (*parselmouth.Thing property*), 45  
`combine_to_stereo()` (*parselmouth.Sound static method*), 38  
`concatenate()` (*parselmouth.Sound static method*), 38  
`convert_to_mono()` (*parselmouth.Sound method*), 38  
`convert_to_stereo()` (*parselmouth.Sound method*), 38  
`convolve()` (*parselmouth.MFCC method*), 32  
`convolve()` (*parselmouth.Sound method*), 38  
`copy()` (*parselmouth.Data method*), 29  
`count_differences()` (*parselmouth.Pitch method*), 34

`count voiced frames()` (*parselmouth.Pitch method*), 34  
`cross_correlate()` (*parselmouth.MFCC method*), 32  
`cross_correlate()` (*parselmouth.Sound method*), 38  
`CUBIC` (*parselmouth.ValueInterpolation attribute*), 46

## D

`Data` (*class in parselmouth*), 28  
`Data.FileFormat` (*class in parselmouth*), 28  
`DB` (*parselmouth.Intensity.AveragingMethod attribute*), 31  
`de_emphasize()` (*parselmouth.Sound method*), 38  
`deepen_band_modulation()` (*parselmouth.Sound method*), 38  
`df` (*parselmouth.Spectrum property*), 44  
`divide()` (*parselmouth.Vector method*), 47  
`dt` (*parselmouth.TimeFrameSampled property*), 45  
`duration` (*parselmouth.TimeFunction property*), 46  
`dx` (*parselmouth.Sampled property*), 36  
`dy` (*parselmouth.SampledXY property*), 36

## E

`end_time` (*parselmouth.TimeFunction property*), 46  
`ENERGY` (*parselmouth.Intensity.AveragingMethod attribute*), 31  
`ERB` (*parselmouth.PitchUnit attribute*), 35  
`extract_all_channels()` (*parselmouth.Sound method*), 38  
`extract_channel()` (*parselmouth.Sound method*), 38  
`extract_features()` (*parselmouth.MFCC method*), 32  
`extract_left_channel()` (*parselmouth.Sound method*), 38  
`extract_part()` (*parselmouth.Sound method*), 38  
`extract_part_for_overlap()` (*parselmouth.Sound method*), 39  
`extract_right_channel()` (*parselmouth.Sound method*), 39

## F

`fifth_down()` (*parselmouth.Pitch method*), 34  
`fifth_up()` (*parselmouth.Pitch method*), 34  
`FLAC` (*parselmouth.SoundFormat attribute*), 41  
`fmax` (*parselmouth.CC property*), 28  
`fmax` (*parselmouth.Spectrum property*), 44  
`fmin` (*parselmouth.CC property*), 28  
`fmin` (*parselmouth.Spectrum property*), 44  
`Formant` (*class in parselmouth*), 30  
`FormantUnit` (*class in parselmouth*), 30  
`formula()` (*parselmouth.Matrix method*), 32  
`formula()` (*parselmouth.Pitch method*), 34  
`frame_number_to_time()` (*parselmouth.TimeFrameSampled method*), 45  
`frequency` (*parselmouth.Pitch.Candidate property*), 33  
`from_tgt()` (*parselmouth.TextGrid static method*), 45  
`full_name` (*parselmouth.Thing property*), 45

Function (*class in parselmouth*), 30

## G

GAUSSIAN (*parselmouth.SpectralAnalysisWindowShape attribute*), 42  
 GAUSSIAN1 (*parselmouth.WindowShape attribute*), 48  
 GAUSSIAN2 (*parselmouth.WindowShape attribute*), 48  
 GAUSSIAN3 (*parselmouth.WindowShape attribute*), 48  
 GAUSSIAN4 (*parselmouth.WindowShape attribute*), 48  
 GAUSSIAN5 (*parselmouth.WindowShape attribute*), 48  
 get\_average() (*parselmouth.Intensity method*), 31  
 get\_band\_density() (*parselmouth.Spectrum method*), 43  
 get\_band\_density\_difference() (*parselmouth.Spectrum method*), 43  
 get\_band\_energy() (*parselmouth.Spectrum method*), 43  
 get\_band\_energy\_difference() (*parselmouth.Spectrum method*), 43  
 get\_bandwidth\_at\_time() (*parselmouth.Formant method*), 30  
 get\_bin\_number\_from\_frequency() (*parselmouth.Spectrum method*), 43  
 get\_bin\_width() (*parselmouth.Spectrum method*), 43  
 get\_c0\_value\_in\_frame() (*parselmouth.CC method*), 28  
 get\_center\_of\_gravity() (*parselmouth.Spectrum method*), 43  
 get\_central\_moment() (*parselmouth.Spectrum method*), 43  
 get\_centre\_of\_gravity() (*parselmouth.Spectrum method*), 43  
 get\_column\_distance() (*parselmouth.Matrix method*), 32  
 get\_end\_time() (*parselmouth.TimeFunction method*), 45  
 get\_energy() (*parselmouth.Sound method*), 39  
 get\_energy\_in\_air() (*parselmouth.Sound method*), 39  
 get\_frame() (*parselmouth.CC method*), 28  
 get\_frame() (*parselmouth.Pitch method*), 34  
 get\_frame\_number\_from\_time() (*parselmouth.TimeFrameSampled method*), 45  
 get\_frequency\_from\_bin\_number() (*parselmouth.Spectrum method*), 44  
 get\_highest\_frequency() (*parselmouth.Spectrum method*), 44  
 get\_highest\_x() (*parselmouth.Matrix method*), 32  
 get\_highest\_y() (*parselmouth.Matrix method*), 32  
 get\_imaginary\_value\_in\_bin() (*parselmouth.Spectrum method*), 44  
 get\_index\_from\_time() (*parselmouth.Sound method*), 39  
 get\_intensity() (*parselmouth.Sound method*), 39  
 get\_kurtosis() (*parselmouth.Spectrum method*), 44  
 get\_lowest\_frequency() (*parselmouth.Spectrum method*), 44  
 get\_lowest\_x() (*parselmouth.Matrix method*), 32  
 get\_lowest\_y() (*parselmouth.Matrix method*), 32  
 get\_maximum() (*parselmouth.Matrix method*), 32  
 get\_mean\_absolute\_slope() (*parselmouth.Pitch method*), 34  
 get\_minimum() (*parselmouth.Matrix method*), 32  
 get\_nearest\_zero\_crossing() (*parselmouth.Sound method*), 39  
 get\_number\_of\_bins() (*parselmouth.Spectrum method*), 44  
 get\_number\_of\_channels() (*parselmouth.Sound method*), 39  
 get\_number\_of\_coefficients() (*parselmouth.CC method*), 28  
 get\_number\_of\_columns() (*parselmouth.Matrix method*), 32  
 get\_number\_of\_frames() (*parselmouth.TimeFrameSampled method*), 45  
 get\_number\_of\_rows() (*parselmouth.Matrix method*), 32  
 get\_number\_of\_samples() (*parselmouth.Sound method*), 39  
 get\_power() (*parselmouth.Sound method*), 39  
 get\_power\_at() (*parselmouth.Spectrogram method*), 42  
 get\_power\_in\_air() (*parselmouth.Sound method*), 39  
 get\_real\_value\_in\_bin() (*parselmouth.Spectrum method*), 44  
 get\_rms() (*parselmouth.Sound method*), 39  
 get\_root\_mean\_square() (*parselmouth.Sound method*), 39  
 get\_row\_distance() (*parselmouth.Matrix method*), 32  
 get\_sampling\_frequency() (*parselmouth.Sound method*), 39  
 get\_sampling\_period() (*parselmouth.Sound method*), 39  
 get\_skewness() (*parselmouth.Spectrum method*), 44  
 get\_slope\_without\_octave\_jumps() (*parselmouth.Pitch method*), 34  
 get\_standard\_deviation() (*parselmouth.Spectrum method*), 44  
 get\_start\_time() (*parselmouth.TimeFunction method*), 45  
 get\_sum() (*parselmouth.Matrix method*), 32  
 get\_time\_from\_frame\_number() (*parselmouth.TimeFrameSampled method*), 45  
 get\_time\_from\_index() (*parselmouth.Sound method*), 39  
 get\_time\_step() (*parselmouth.TimeFrameSampled method*), 45  
 get\_total\_duration() (*parselmouth.TimeFunction*

*method), 46*  
**get\_value()** (*parselmouth.Harmonicity method*), 31  
**get\_value()** (*parselmouth.Intensity method*), 31  
**get\_value()** (*parselmouth.Vector method*), 47  
**get\_value\_at\_time()** (*parselmouth.Formant method*), 30  
**get\_value\_at\_time()** (*parselmouth.Pitch method*), 34  
**get\_value\_at\_xy()** (*parselmouth.Matrix method*), 32  
**get\_value\_in\_bin()** (*parselmouth.Spectrum method*), 44  
**get\_value\_in\_cell()** (*parselmouth.Matrix method*), 33  
**get\_value\_in\_frame()** (*parselmouth.CC method*), 28  
**get\_value\_in\_frame()** (*parselmouth.Pitch method*), 34  
**get\_x\_of\_column()** (*parselmouth.Matrix method*), 33  
**get\_y\_of\_row()** (*parselmouth.Matrix method*), 33  
**GNE** (*parselmouth.Sound.ToHarmonicityMethod attribute*), 37

## H

**HAMMING** (*parselmouth.SpectralAnalysisWindowShape attribute*), 42  
**HAMMING** (*parselmouth.WindowShape attribute*), 48  
**HANNING** (*parselmouth.SpectralAnalysisWindowShape attribute*), 42  
**HANNING** (*parselmouth.WindowShape attribute*), 48  
**Harmonicity** (*class in parselmouth*), 31  
**HERTZ** (*parselmouth.FormantUnit attribute*), 30  
**HERTZ** (*parselmouth.PitchUnit attribute*), 35  
**HERTZ\_LOGARITHMIC** (*parselmouth.PitchUnit attribute*), 35  
**highest\_frequency** (*parselmouth.Spectrum property*), 44

## I

**info()** (*parselmouth.Thing method*), 45  
**INTEGRAL** (*parselmouth.AmplitudeScaling attribute*), 27  
**Intensity** (*class in parselmouth*), 31  
**intensity** (*parselmouth.Pitch.Frame property*), 33  
**Intensity.AveragingMethod** (*class in parselmouth*), 31  
**interpolate()** (*parselmouth.Pitch method*), 34  
**Interpolation** (*in module parselmouth*), 31

## K

**KAISER1** (*parselmouth.WindowShape attribute*), 48  
**KAISER2** (*parselmouth.WindowShape attribute*), 48  
**KAY** (*parselmouth.SoundFormat attribute*), 41  
**kill\_octave\_jumps()** (*parselmouth.Pitch method*), 34

## L

**lengthen()** (*parselmouth.Sound method*), 39

**LINEAR** (*parselmouth.ValueInterpolation attribute*), 46  
**LOG\_HERTZ** (*parselmouth.PitchUnit attribute*), 35  
**lowest\_frequency** (*parselmouth.Spectrum property*), 44  
**lpc\_smoothing()** (*parselmouth.Spectrum method*), 44

## M

**Matrix** (*class in parselmouth*), 32  
**max\_n\_candidates** (*parselmouth.Pitch property*), 35  
**max\_n\_coefficients** (*parselmouth.CC property*), 28  
**MEDIAN** (*parselmouth.Intensity.AveragingMethod attribute*), 31  
**MEL** (*parselmouth.PitchUnit attribute*), 35  
**MFCC** (*class in parselmouth*), 31  
**module**  
  *parselmouth*, 27  
  *parselmouth.praat*, 48  
**multiply()** (*parselmouth.Vector method*), 47  
**multiply\_by\_window()** (*parselmouth.Sound method*), 39

## N

**n\_bins** (*parselmouth.Spectrum property*), 44  
**n\_channels** (*parselmouth.Sound property*), 41  
**n\_columns** (*parselmouth.Matrix property*), 33  
**n\_frames** (*parselmouth.TimeFrameSampled property*), 45  
**n\_rows** (*parselmouth.Matrix property*), 33  
**n\_samples** (*parselmouth.Sound property*), 41  
**name** (*parselmouth.AmplitudeScaling property*), 28  
**name** (*parselmouth.Data.FileFormat property*), 29  
**name** (*parselmouth.FormantUnit property*), 30  
**name** (*parselmouth.Intensity.AveragingMethod property*), 31  
**name** (*parselmouth.PitchUnit property*), 35  
**name** (*parselmouth.SignalOutsideTimeDomain property*), 37  
**name** (*parselmouth.Sound.ToHarmonicityMethod property*), 37  
**name** (*parselmouth.Sound.ToPitchMethod property*), 38  
**name** (*parselmouth.SoundFormat property*), 42  
**name** (*parselmouth.SpectralAnalysisWindowShape property*), 42  
**name** (*parselmouth.Thing property*), 45  
**name** (*parselmouth.ValueInterpolation property*), 47  
**name** (*parselmouth.WindowShape property*), 48  
**NEAREST** (*parselmouth.ValueInterpolation attribute*), 46  
**NEXT\_SUN** (*parselmouth.SoundFormat attribute*), 41  
**nf** (*parselmouth.Spectrum property*), 44  
**NIST** (*parselmouth.SoundFormat attribute*), 41  
**NORMALIZE** (*parselmouth.AmplitudeScaling attribute*), 27  
**nt** (*parselmouth.TimeFrameSampled property*), 45  
**nx** (*parselmouth.Sampled property*), 36  
**ny** (*parselmouth.SampledXY property*), 36

## O

`octave_down()` (*parselmouth.Pitch method*), 34  
`octave_up()` (*parselmouth.Pitch method*), 34  
`override_sampling_frequency()` (*parselmouth.Sound method*), 39

## P

`PARABOLIC` (*parselmouth.WindowShape attribute*), 48  
`parselmouth`  
  module, 27  
`parselmouth.praat`  
  module, 48  
`path_finder()` (*parselmouth.Pitch method*), 34  
`PEAK_0_99` (*parselmouth.AmplitudeScaling attribute*), 27  
`Pitch` (*class in parselmouth*), 33  
`Pitch.Candidate` (*class in parselmouth*), 33  
`Pitch.Frame` (*class in parselmouth*), 33  
`PitchUnit` (*class in parselmouth*), 35  
`PRAAT_VERSION` (*in module parselmouth*), 27  
`PRAAT_VERSION_DATE` (*in module parselmouth*), 27  
`PraatError`, 27  
`PraatFatal`, 27  
`PraatWarning`, 27  
`pre_emphasize()` (*parselmouth.Sound method*), 39

## R

`RAW_16_BE` (*parselmouth.SoundFormat attribute*), 41  
`RAW_16_LE` (*parselmouth.SoundFormat attribute*), 41  
`RAW_24_BE` (*parselmouth.SoundFormat attribute*), 41  
`RAW_24_LE` (*parselmouth.SoundFormat attribute*), 41  
`RAW_32_BE` (*parselmouth.SoundFormat attribute*), 41  
`RAW_32_LE` (*parselmouth.SoundFormat attribute*), 41  
`RAW_8_SIGNED` (*parselmouth.SoundFormat attribute*), 41  
`RAW_8_UNSIGNED` (*parselmouth.SoundFormat attribute*), 42  
`read()` (*in module parselmouth*), 48  
`read()` (*parselmouth.Data static method*), 29  
`RECTANGULAR` (*parselmouth.WindowShape attribute*), 48  
`resample()` (*parselmouth.Sound method*), 39  
`reverse()` (*parselmouth.Sound method*), 39  
`run()` (*in module parselmouth.praat*), 50  
`run_file()` (*in module parselmouth.praat*), 51

## S

`Sampled` (*class in parselmouth*), 35  
`SampledXY` (*class in parselmouth*), 36  
`sampling_frequency` (*parselmouth.Sound property*), 41  
`sampling_period` (*parselmouth.Sound property*), 41  
`save()` (*parselmouth.Data method*), 29  
`save()` (*parselmouth.Sound method*), 39

`save_as_binary_file()` (*parselmouth.Data method*), 29  
`save_as_headerless_spreadsheet_file()` (*parselmouth.Matrix method*), 33  
`save_as_matrix_text_file()` (*parselmouth.Matrix method*), 33  
`save_as_short_text_file()` (*parselmouth.Data method*), 29  
`save_as_text_file()` (*parselmouth.Data method*), 29  
`scale()` (*parselmouth.Vector method*), 47  
`scale_intensity()` (*parselmouth.Sound method*), 39  
`scale_peak()` (*parselmouth.Vector method*), 47  
`scale_times_by()` (*parselmouth.TimeFunction method*), 46  
`scale_times_to()` (*parselmouth.TimeFunction method*), 46  
`scale_x_by()` (*parselmouth.Function method*), 30  
`scale_x_to()` (*parselmouth.Function method*), 30  
`select()` (*parselmouth.Pitch.Frame method*), 33  
`selected` (*parselmouth.Pitch property*), 35  
`selected` (*parselmouth.Pitch.Frame property*), 33  
`selected_array` (*parselmouth.Pitch property*), 35  
`SEMITONES_1` (*parselmouth.PitchUnit attribute*), 35  
`SEMITONES_100` (*parselmouth.PitchUnit attribute*), 35  
`SEMITONES_200` (*parselmouth.PitchUnit attribute*), 35  
`SEMITONES_440` (*parselmouth.PitchUnit attribute*), 35  
`SESAM` (*parselmouth.SoundFormat attribute*), 42  
`set_imaginary_value_in_bin()` (*parselmouth.Spectrum method*), 44  
`set_real_value_in_bin()` (*parselmouth.Spectrum method*), 44  
`set_to_zero()` (*parselmouth.Sound method*), 39  
`set_value()` (*parselmouth.Matrix method*), 33  
`set_value_in_bin()` (*parselmouth.Spectrum method*), 44  
`shift_times_by()` (*parselmouth.TimeFunction method*), 46  
`shift_times_to()` (*parselmouth.TimeFunction method*), 46  
`shift_x_by()` (*parselmouth.Function method*), 30  
`shift_x_to()` (*parselmouth.Function method*), 30  
`SHORT_TEXT` (*parselmouth.Data.FileFormat attribute*), 29  
`SHS` (*parselmouth.Sound.ToPitchMethod attribute*), 38  
`SignalOutsideTimeDomain` (*class in parselmouth*), 36  
`SIMILAR` (*parselmouth.SignalOutsideTimeDomain attribute*), 36  
`SINC70` (*parselmouth.ValueInterpolation attribute*), 46  
`SINC700` (*parselmouth.ValueInterpolation attribute*), 46  
`smooth()` (*parselmouth.Pitch method*), 34  
`SONES` (*parselmouth.Intensity.AveragingMethod attribute*), 31  
`Sound` (*class in parselmouth*), 37

`Sound.ToHarmonicityMethod` (*class in parselmouth*), 37  
`Sound.ToPitchMethod` (*class in parselmouth*), 37  
`SoundFileFormat` (*class in parselmouth*), 41  
`SpectralAnalysisWindowShape` (*class in parselmouth*), 42  
`Spectrogram` (*class in parselmouth*), 42  
`Spectrum` (*class in parselmouth*), 43  
`SPINET` (*parselmouth.Sound.ToPitchMethod attribute*), 38  
`SQUARE` (*parselmouth.SpectralAnalysisWindowShape attribute*), 42  
`start_time` (*parselmouth.TimeFunction property*), 46  
`step()` (*parselmouth.Pitch method*), 34  
`strength` (*parselmouth.Pitch.Candidate property*), 33  
`subtract()` (*parselmouth.Vector method*), 47  
`subtract_linear_fit()` (*parselmouth.Pitch method*), 34  
`subtract_mean()` (*parselmouth.Vector method*), 47  
`SUM` (*parselmouth.AmplitudeScaling attribute*), 27  
`synthesize_sound()` (*parselmouth.Spectrogram method*), 42

**T**

`t1` (*parselmouth.TimeFrameSampled property*), 45  
`t_bins()` (*parselmouth.TimeFrameSampled method*), 45  
`t_grid()` (*parselmouth.TimeFrameSampled method*), 45  
`TEXT` (*parselmouth.Data.FileFormat attribute*), 29  
`TextGrid` (*class in parselmouth*), 44  
`Thing` (*class in parselmouth*), 45  
`time_range` (*parselmouth.TimeFunction property*), 46  
`time_step` (*parselmouth.TimeFrameSampled property*), 45  
`time_to_frame_number()` (*parselmouth.TimeFrameSampled method*), 45  
`TimeFrameSampled` (*class in parselmouth*), 45  
`TimeFunction` (*class in parselmouth*), 45  
`tmax` (*parselmouth.TimeFunction property*), 46  
`tmin` (*parselmouth.TimeFunction property*), 46  
`to_array()` (*parselmouth.CC method*), 28  
`to_array()` (*parselmouth.CC.Frame method*), 28  
`to_array()` (*parselmouth.Pitch method*), 34  
`to_formant_burg()` (*parselmouth.Sound method*), 39  
`to_harmonicity()` (*parselmouth.Sound method*), 40  
`to_harmonicity_ac()` (*parselmouth.Sound method*), 40  
`to_harmonicity_cc()` (*parselmouth.Sound method*), 40  
`to_harmonicity_gne()` (*parselmouth.Sound method*), 40  
`to_intensity()` (*parselmouth.Sound method*), 40  
`to_matrix()` (*parselmouth.CC method*), 28  
`to_matrix()` (*parselmouth.Pitch method*), 34

`to_matrix_features()` (*parselmouth.MFCC method*), 32  
`to_mfcc()` (*parselmouth.Sound method*), 40  
`to_pitch()` (*parselmouth.Sound method*), 40  
`to_pitch_ac()` (*parselmouth.Sound method*), 40  
`to_pitch_cc()` (*parselmouth.Sound method*), 40  
`to_pitch_shs()` (*parselmouth.Sound method*), 40  
`to_pitch_spinet()` (*parselmouth.Sound method*), 40  
`to_sound()` (*parselmouth.MFCC method*), 32  
`to_sound()` (*parselmouth.Spectrogram method*), 43  
`to_sound()` (*parselmouth.Spectrum method*), 44  
`to_sound_hum()` (*parselmouth.Pitch method*), 34  
`to_sound_pulses()` (*parselmouth.Pitch method*), 34  
`to_sound_sine()` (*parselmouth.Pitch method*), 34  
`to_spectrogram()` (*parselmouth.Sound method*), 40  
`to_spectrogram()` (*parselmouth.Spectrum method*), 44  
`to_spectrum()` (*parselmouth.Sound method*), 41  
`to_spectrum_slice()` (*parselmouth.Spectrogram method*), 43  
`to_tgt()` (*parselmouth.TextGrid method*), 45  
`total_duration` (*parselmouth.TimeFunction property*), 46  
`trange` (*parselmouth.TimeFunction property*), 46  
`TRIANGULAR` (*parselmouth.WindowShape attribute*), 48  
`ts()` (*parselmouth.TimeFrameSampled method*), 45

**U**

`unvoice()` (*parselmouth.Pitch method*), 35  
`unvoice()` (*parselmouth.Pitch.Frame method*), 33

**V**

`value` (*parselmouth.AmplitudeScaling property*), 28  
`value` (*parselmouth.Data.FileFormat property*), 29  
`value` (*parselmouth.FormantUnit property*), 30  
`value` (*parselmouth.Intensity.AveragingMethod property*), 31  
`value` (*parselmouth.PitchUnit property*), 35  
`value` (*parselmouth.SignalOutsideTimeDomain property*), 37  
`value` (*parselmouth.Sound.ToHarmonicityMethod property*), 37  
`value` (*parselmouth.Sound.ToPitchMethod property*), 38  
`value` (*parselmouth.SoundFileFormat property*), 42  
`value` (*parselmouth.SpectralAnalysisWindowShape property*), 42  
`value` (*parselmouth.ValueInterpolation property*), 47  
`value` (*parselmouth.WindowShape property*), 48  
`ValueInterpolation` (*class in parselmouth*), 46  
`values` (*parselmouth.Matrix property*), 33  
`Vector` (*class in parselmouth*), 47  
`VERSION` (*in module parselmouth*), 27

**W**

`WAV` (*parselmouth.SoundFileFormat attribute*), 42

`WAV_24` (*parselmouth.SoundFileFormat attribute*), 42  
`WAV_32` (*parselmouth.SoundFileFormat attribute*), 42  
`WELCH` (*parselmouth.SpectralAnalysisWindowShape attribute*), 42  
`WindowShape` (*class in parselmouth*), 47

## X

`x1` (*parselmouth.Sampled property*), 36  
`x_bins()` (*parselmouth.Sampled method*), 36  
`x_grid()` (*parselmouth.Sampled method*), 36  
`xmax` (*parselmouth.Function property*), 30  
`xmin` (*parselmouth.Function property*), 30  
`xrange` (*parselmouth.Function property*), 31  
`xs()` (*parselmouth.Sampled method*), 36

## Y

`y1` (*parselmouth.SampledXY property*), 36  
`y_bins()` (*parselmouth.SampledXY method*), 36  
`y_grid()` (*parselmouth.SampledXY method*), 36  
`ymax` (*parselmouth.SampledXY property*), 36  
`ymin` (*parselmouth.SampledXY property*), 36  
`yrange` (*parselmouth.SampledXY property*), 36  
`ys()` (*parselmouth.SampledXY method*), 36

## Z

`ZERO` (*parselmouth.SignalOutsideTimeDomain attribute*),  
37