# Index



# Module **utilities**

▶ EXPAND SOURCE CODE

## Functions

```
def print_data(channel, data, flush=True)
```

Prints a data message if printing data is active.

**Args:**

`channel` : Number of the channel where the data was processed from.

`string` : Message to print.

`flush` : Whether to flush when using built-in print function.

▶ EXPAND SOURCE CODE

```
def print_data_alt_color(channel, data, flush=True)
```

Prints a data message with alternate color if printing data is active.

**Args:**

`channel` : Number of the channel where the data was processed from.

`string` : Message to print.

`flush` : Whether to flush when using built-in print function.

▶ EXPAND SOURCE CODE

```
def print_dbg(string, flush=True)
```

Prints a generic debug message if printing and debug printing is active.

**Args:**

`string` : Message to print.

`flush` : Whether to flush when using built-in print function.

▶ EXPAND SOURCE CODE

```
def print_error(string, flush=True)
```

Prints an error message if printing is active.

**Args:**

`string` : Message to print.

`flush` : Whether to flush when using built-in print function.

▶ EXPAND SOURCE CODE

```
def print_info(string, flush=True)
```

Prints an info message if printing is active.

**Args:**

`string` : Message to print.

`flush` : Whether to flush when using built-in print function.

▶ EXPAND SOURCE CODE

```
def print_success(string, flush=True)
```

Prints a success message if printing is active.

**Args:**

`string` : Message to print.

`flush` : Whether to flush when using built-in print function.

▶ EXPAND SOURCE CODE

```
def print_warning(string, flush=True)
```

Prints a warning message if printing is active.

**Args:**

`string` : Message to print.

`flush` : Whether to flush when using built-in print function.

▶ EXPAND SOURCE CODE

## Classes

```
class BColors
```

Colors used to print and debug.

▶ EXPAND SOURCE CODE

### Class variables

```
var BOLD
```

```
var ENDC
```

```
var FAIL
```

```
var HEADER
```

```
var OKBLUE
```

```
var OKCYAN
```

```
var OKGREEN
```

```
var UNDERLINE
```

```
var WARNING
```

```
class Instruments (value, names=None, *, module=None, qualname=None,
                   type=None, start=1)
```

Defines instrument types for audio processing.

▶ EXPAND SOURCE CODE

### Ancestors

enum.Enum

### Class variables

```
var DEFAULT
```

```
var DRUMS
```

```
var GUITAR
```

```
var PIANO
```

```
var STRINGS
```

```
var VOICE
```

### Static methods

```
def from_index(index: int)
```

Returns the enum value of a given index.

**Args:**

`index` : Number representing the instrument.

**Returns:**

Instrument enum of the given index.

**Raises:**

`SetupException` : If the index does not correspond to a known instrument.

▶ EXPAND SOURCE CODE

```
def from_string(s: str) -> Instruments
```

Returns the enum value of a given string.

**Args:**

`s` : String representing the instrument.

**Returns:**

Instrument enum of the given string.

**Raises:**

`SetupException` : If the string does not correspond to a known instrument.

▶ EXPAND SOURCE CODE

### Methods

```
def get_fundamental_frequency_range(self) -> list
```

Returns the fundamental frequency range for a given instrument.

**Returns:** A :py:type:`list` containing the lower and upper frequency range limits for the instrument.

▶ EXPAND SOURCE CODE

```
def get_string(self) -> str
```

Returns the name of the instrument as a string.

**Returns:**

The name of the instrument as a string.

▶ EXPAND SOURCE CODE

```
class Normalizations (value, names=None, *, module=None, qualname=None,
                      type=None, start=1)
```

Defines Normalization types for audio processing.

▶ EXPAND SOURCE CODE

### Ancestors

enum.Enum

### Class variables

```
var MIN_MAX
```

```
var NONE
```

```
var PEAK
```

```
var RMS
```

```
var Z_SCORE
```