All of Monolib

Container data types

The whole library is based on the `Mono` data type, which is simply a `NamedTuple` with three fields

- `data`: this field is reserved for a 1D array representing the values of a signal.
- `sample_rate`: the sample rate of said signal.
- `tags`: any relevant meta data that you wish to attach to said signal.

```
from monolib.containers import mono
from numpy import array

x = mono(array([1, 2, 3]), 8_000, {"type": "B"})
x
```

```
Mono(data=array([1, 2, 3]), sample_rate=8000, tags={'type': 'B'})
```

Since `Mono` is a named tuple, we can deconstruct its components by either unpacking,

```
data, sample_rate, tags = x
print(data, sample_rate, tags)
```

```
[1 2 3] 8000 {'type': 'B'}
```

or by using its named attributes,

```
print(x.data, x.sample_rate, x.tags)
```

```
[1 2 3] 8000 {'type': 'B'}
```

`Mono` containers can be collected as `MonoCollection` types, which are nothing but a `NamedTuple` which includes a tuple of `Mono` containers along with optional `tags` for recording collection-wide meta data.

```
from monolib.containers import collect

x1 = mono(array([1, 2, 3]), 8_000, {"type": "A"})
x2 = mono(array([4, 5, 6]), 8_000, {"type": "B"})
x3 = mono(array([7, 8, 9]), 8_000, {"type": "C"})
```

```
collection = collect(x1, x2, x3, tags={"relevant": "info"})
collection

from monolib._disp import display_collection

display_collection(collection)
```

This abstraction allows us to define composable signal processing transformations which in a functional programming style, which allows us to do some cool and useful stuff.

Chainable transformations

Pretty printing