# The Wavelet object

## Wavelet families and builtin Wavelets names

**`Wavelet`** objects are really a handy carriers of a bunch of DWT-specific data like *quadrature mirror filters* and some general properties associated with them.

At first let's go through the methods of creating a **`Wavelet`** object. The easiest and the most convenient way is to use builtin named Wavelets.

These wavelets are organized into groups called wavelet families. The most commonly used families are:

```
>>> import pywt
>>> pywt.families()
['haar', 'db', 'sym', 'coif', 'bior', 'rbio', 'dmey', 'gaus', 'mexh', 'morl', 'cgau', 'shan', 'fbsp', 'cmor']
```

The **`wavelist()`** function with family name passed as an argument is used to obtain the list of wavelet names in each family.

```
>>> for family in pywt.families():
...     print("%s family: " % family + ', '.join(pywt.wavelist(family)))
haar family: haar
db family: db1, db2, db3, db4, db5, db6, db7, db8, db9, db10, db11, db12, db13, db14, db15, db16, db17, db18, db
sym family: sym2, sym3, sym4, sym5, sym6, sym7, sym8, sym9, sym10, sym11, sym12, sym13, sym14, sym15, sym16, sym
coif family: coif1, coif2, coif3, coif4, coif5, coif6, coif7, coif8, coif9, coif10, coif11, coif12, coif13, coif
bior family: bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8, bior3.1, bior3.3, bior3.5, bior3.7,
rbio family: rbio1.1, rbio1.3, rbio1.5, rbio2.2, rbio2.4, rbio2.6, rbio2.8, rbio3.1, rbio3.3, rbio3.5, rbio3.7,
dmey family: dmey
gaus family: gaus1, gaus2, gaus3, gaus4, gaus5, gaus6, gaus7, gaus8
mexh family: mexh
morl family: morl
cgau family: cgau1, cgau2, cgau3, cgau4, cgau5, cgau6, cgau7, cgau8
shan family: shan
fbsp family: fbsp
cmor family: cmor
```

To get the full list of builtin wavelets' names just use the **`wavelist()`** with no argument.

## Creating Wavelet objects

Now when we know all the names let's finally create a **`Wavelet`** object:

```
>>> w = pywt.Wavelet('db3')
```

So.. that's it.

## Wavelet properties

But what can we do with **`Wavelet`** objects? Well, they carry some interesting information.

First, let's try printing a **`Wavelet`** object. This shows a brief information about its name, its family name and some properties like orthogonality and symmetry.

```
>>> print(w)
Wavelet db3
  Family name:    Daubechies
  Short name:     db
  Filters length: 6
  Orthogonal:     True
  Biorthogonal:   True
  Symmetry:       asymmetric
  DWT:            True
  CWT:            False
```

But the most important information are the wavelet filters coefficients, which are used in Discrete Wavelet Transform. These coefficients can be obtained via the **`dec_lo`**, **`Wavelet.dec_hi`**, **`rec_lo`** and **`rec_hi`** attributes, which corresponds to lowpass and highpass decomposition filters and lowpass and highpass reconstruction filters respectively:

```
>>> def print_array(arr):
...     print("[%s]" % ", ".join(["%.14f" % x for x in arr]))
```

Another way to get the filters data is to use the **`filter_bank`** attribute, which returns all four filters in a tuple:

```
>>> w.filter_bank == (w.dec_lo, w.dec_hi, w.rec_lo, w.rec_hi)
True
```

Other Wavelet's properties are:

Wavelet **`name`**, **`short_family_name`** and **`family_name`**:

```
>>> print(w.name)
db3
>>> print(w.short_family_name)
db
>>> print(w.family_name)
Daubechies
```

- Decomposition (**`dec_len`**) and reconstruction (**`rec_len`**) filter lengths:

  ```
  >>> int(w.dec_len) # int() is for normalizing longs and ints for doctest
  6
  >>> int(w.rec_len)
  6
  ```

- Orthogonality (**`orthogonal`**) and biorthogonality (**`biorthogonal`**):

  ```
  >>> w.orthogonal
  True
  >>> w.biorthogonal
  True
  ```

- Symmetry (**`symmetry`**):

```
>>> print(w.symmetry)
asymmetric
```

- Number of vanishing moments for the scaling function `phi` (**`vanishing_moments_phi`**) and the wavelet function `psi` (**`vanishing_moments_psi`**) associated with the filters:

```
>>> w.vanishing_moments_phi
0
>>> w.vanishing_moments_psi
3
```

Now when we know a bit about the builtin Wavelets, let's see how to create custom Wavelets objects. These can be done in two ways:

1. Passing the filter bank object that implements the `filter_bank` attribute. The attribute must return four filters coefficients.

```
>>> class MyHaarFilterBank(object):
...     @property
...     def filter_bank(self):
...         from math import sqrt
...         return ([sqrt(2)/2, sqrt(2)/2], [-sqrt(2)/2, sqrt(2)/2],
...                 [sqrt(2)/2, sqrt(2)/2], [sqrt(2)/2, -sqrt(2)/2])
```

```
>>> my_wavelet = pywt.Wavelet('My Haar Wavelet', filter_bank=MyHaarFilterBank())
```

2. Passing the filters coefficients directly as the `filter_bank` parameter.

```
>>> from math import sqrt
>>> my_filter_bank = ([sqrt(2)/2, sqrt(2)/2], [-sqrt(2)/2, sqrt(2)/2],
...                   [sqrt(2)/2, sqrt(2)/2], [sqrt(2)/2, -sqrt(2)/2])
>>> my_wavelet = pywt.Wavelet('My Haar Wavelet', filter_bank=my_filter_bank)
```

Note that such custom wavelets **will not** have all the properties set to correct values:

```
>>> print(my_wavelet)
Wavelet My Haar Wavelet
  Family name:
  Short name:
  Filters length: 2
  Orthogonal:     False
  Biorthogonal:   False
  Symmetry:       unknown
  DWT:            True
  CWT:            False
```

You can however set a couple of them on your own:

```
>>> my_wavelet.orthogonal = True
>>> my_wavelet.biorthogonal = True
```

```
>>> print(my_wavelet)
Wavelet My Haar Wavelet
  Family name:
  Short name:
  Filters length: 2
  Orthogonal:     True
  Biorthogonal:   True
  Symmetry:       unknown
  DWT:            True
  CWT:            False
```

## And now… the `wavefun`!

We all know that the fun with wavelets is in wavelet functions. Now what would be this package without a tool to compute wavelet and scaling functions approximations?

This is the purpose of the **`wavefun()`** method, which is used to approximate scaling function (`phi`) and wavelet function (`psi`) at the given level of refinement, based on the filters coefficients.

The number of returned values varies depending on the wavelet's orthogonality property. For orthogonal wavelets the result is tuple with scaling function, wavelet function and xgrid coordinates.

```
>>> w = pywt.Wavelet('sym3')
>>> w.orthogonal
True
>>> (phi, psi, x) = w.wavefun(level=5)
```

For biorthogonal (non-orthogonal) wavelets different scaling and wavelet functions are used for decomposition and reconstruction, and thus five elements are returned: decomposition scaling and wavelet functions approximations, reconstruction scaling and wavelet functions approximations, and the xgrid.

```
>>> w = pywt.Wavelet('bior1.3')
>>> w.orthogonal
False
>>> (phi_d, psi_d, phi_r, psi_r, x) = w.wavefun(level=5)
```

**See also:** You can find live examples of **`wavefun()`** usage and images of all the built-in wavelets on the Wavelet Properties Browser page. However, **this website is no longer actively maintained** and does not include every wavelet present in PyWavelets. The precision of the wavelet coefficients at that site is also lower than those included in PyWavelets.