Digital Signal Processing

Lecture 3 – Sounds and Signals

상명대학교
컴퓨터과학과
강상욱 교수

Periodic signals I

- Periodic signals : signals that repeat themselves after some period of time.
  - For example, if you strike a bell, it vibrates and generates sound.
- Cycle : Any complete round or series of occurrences that repeats or is repeated.
- Period : The duration of each cycle
- Frequency : the number of cycles per second, which is the inverse of the period.
  - Unit : Hertz (Hz)
  - Figure 1.1 : 439Hz
  - 440Hz : the standard tuning pitch for orchestral music

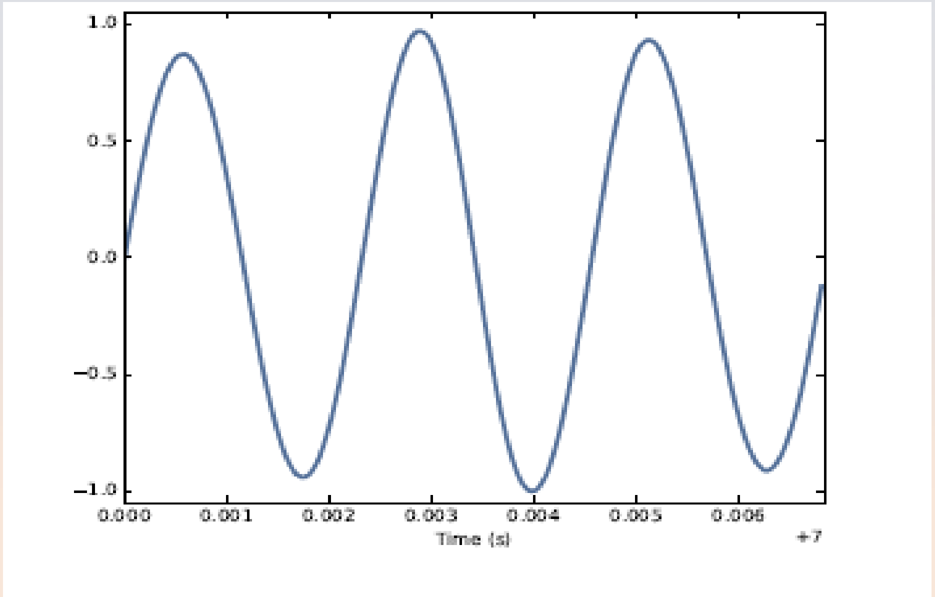


Figure 1.1: Segment from a recording of a bell.

Periodic signals 2

- Most musical instruments produce periodic signals, but the shape of these signals is not sinusoidal.
  - Figure 1.2 shows a segment from a recording of a violin.
- The shape of a periodic signal is called the waveform.
- The shape of the waveform determines the musical timbre (음색), which is our perception of the quality of the sound.
- People usually perceive complex waveforms as rich, warm and more interesting than sinusoids.

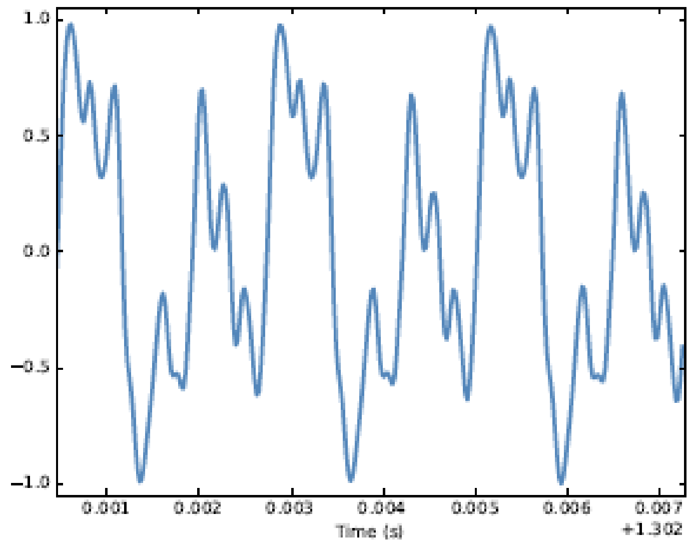


Figure 1.2: Segment from a recording of a violin.

Spectral decomposition I

- The idea : Any signal can be expressed as the sum of sinusoids with different frequencies.
- DFT (Discrete Fourier Transform)
  - Takes a signal and produces its spectrum.
  - Spectrum : the set of sinusoids that add up to produce the signal.
  - FFT (Fast Fourier Transform) : An efficient way to compute the DFT.

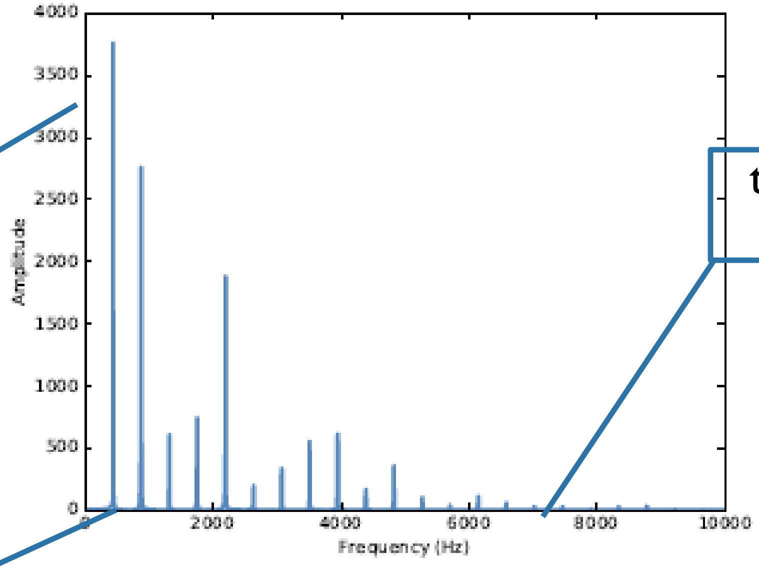


Figure 1.3: Spectrum of a segment from the violin recording.

The strength or amplitude of each freq. component

the range of freq. that make up the signal.

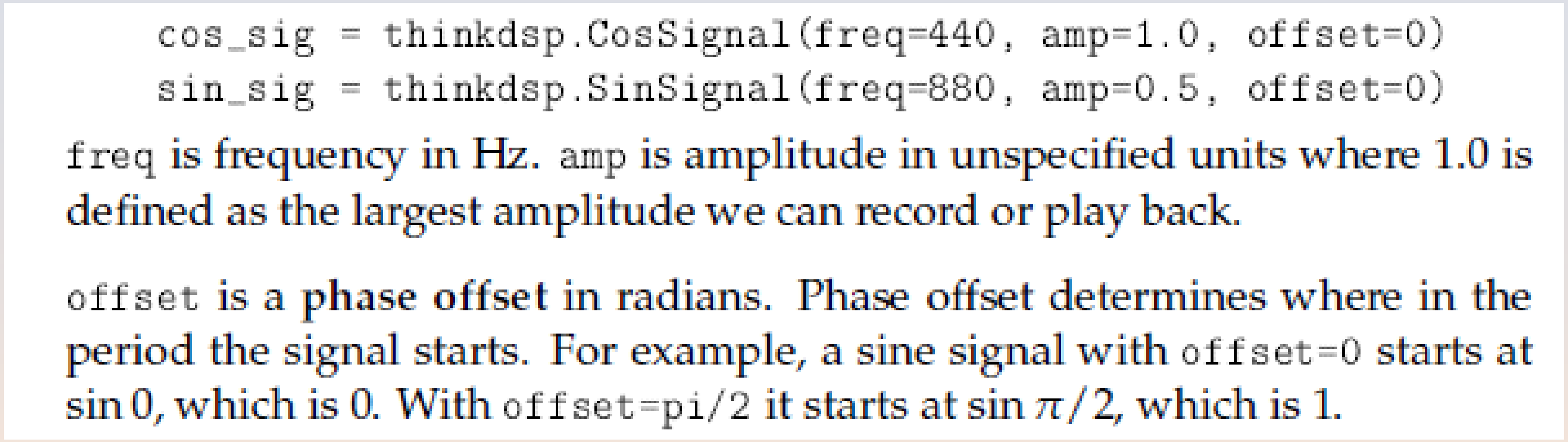
The fundamental freq. is near 440Hz

Spectral decomposition 2

- Fundamental frequency : the lowest frequency component
- Dominant frequency : the frequency component with the largest amplitude.
  - Normally the perceived pitch of a sound is determined by the fundamental freq., even if it is not dominant.
- Harmonics : integer multiples of the fundamental.
  - The other spikes in the spectrum are at 880, 1320, 1760 and 2200 Hz.
  - They are musically harmonious with the fundamental.
  - 440Hz (A4), 880 Hz (A5), 1320 Hz (E6), 1760 Hz (A6), 2200 Hz (C#7) → A major chord

Signals – thinkdsp.py

- Thinkdsp.py : a Python module that contains classes and functions for working with signals and spectrums.
  - Signal class : represents several signal types including Sinusoid.
  - Sinusoid represents both sine and cosine signals.
  - To create sine and cosine signals.



```
    cos_sig = thinkdsp.CosSignal(freq=440, amp=1.0, offset=0)
    sin_sig = thinkdsp.SinSignal(freq=880, amp=0.5, offset=0)
```

`freq` is frequency in Hz. `amp` is amplitude in unspecified units where 1.0 is defined as the largest amplitude we can record or play back.

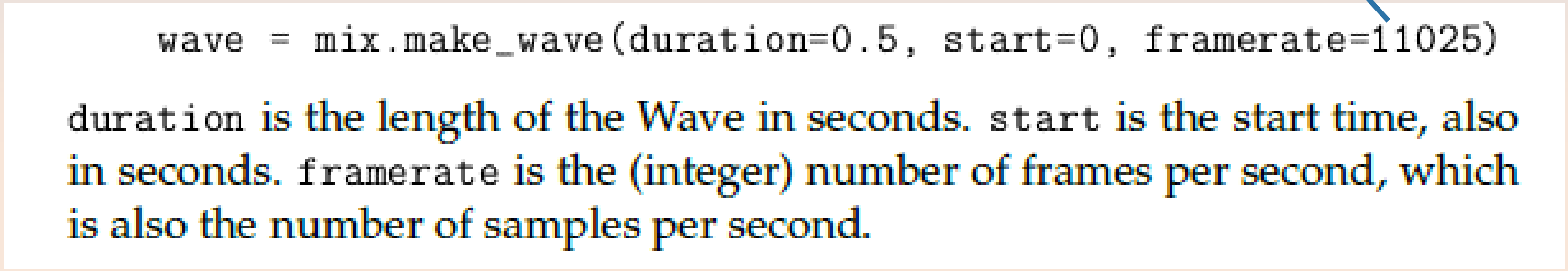
`offset` is a phase offset in radians. Phase offset determines where in the period the signal starts. For example, a sine signal with `offset=0` starts at $\sin 0$, which is 0. With `offset=pi/2` it starts at $\sin \pi/2$, which is 1.

  - Signals have an __add__ method

Signals - Terms

- A signal : a Python representation of a mathematical function.
  - It is defined for all values of t, from negative infinity to infinity.
- You can't do much with a signal until you evaluate it.
  - Evaluation : a sequence o points in time ts and corresponding values of the signal ys.
  - ts and ys can be represented using NumPy arrays and encapsulate them in an object called a "wave".
- A wave : a signal evaluated at a sequence of points in time.
  - Frame : each point in time.
  - Sample : the measurement itself.
  - make_wave : returns a new wave object.

Commonly used in WAV and mp3 audio file formats

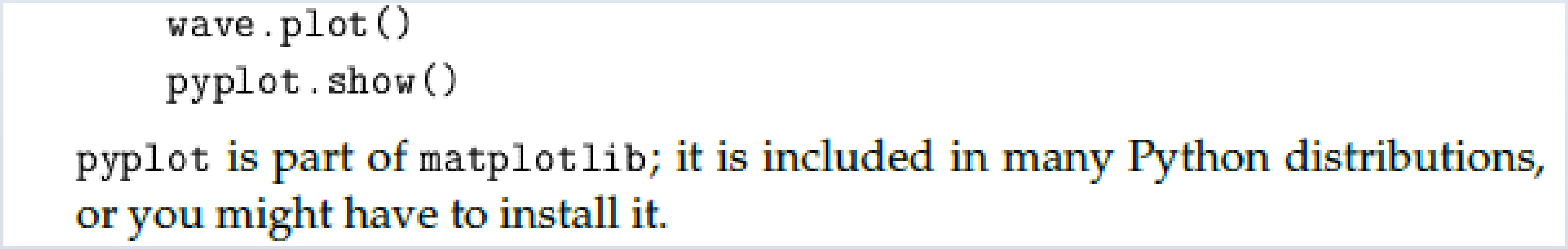


```
    wave = mix.make_wave(duration=0.5, start=0, framerate=11025)
```

`duration` is the length of the Wave in seconds. `start` is the start time, also in seconds. `framerate` is the (integer) number of frames per second, which is also the number of samples per second.

Signals - Plot

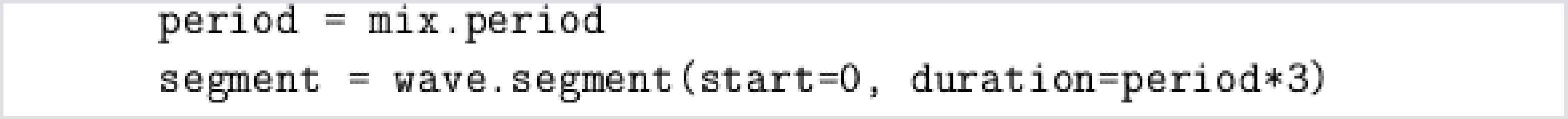
- You can plot the wave like this.



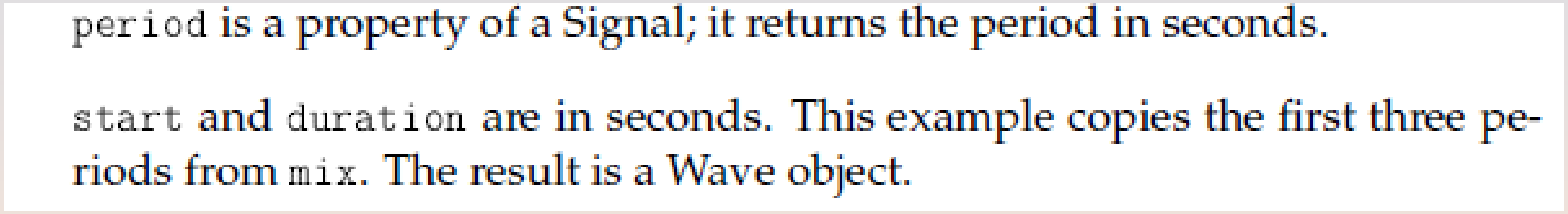
```
wave.plot()
pyplot.show()
```

pyplot is part of matplotlib; it is included in many Python distributions, or you might have to install it.

- To zoom in on a small number of periods



```
period = mix.period
segment = wave.segment(start=0, duration=period*3)
```



period is a property of a Signal; it returns the period in seconds.

start and duration are in seconds. This example copies the first three periods from mix. The result is a Wave object.

  - See Figure 1.4.

Reading and writing waves

- thinkdsp provides read_wave, which reads a WAV file and returns a wave
  - read :
    ```
    violin_wave = thinkdsp.read_wave('input.wav')
    ```
  - write :
    ```
    wave.write(filename='output.wav')
    ```
  - play :
    ```
    thinkdsp.play_wave(filename='output.wav', player='aplay')
    ```
- In Linux : use "aplay".

Spectrums 1

- Wave provides make_spectrum, which returns a spectrum.

```
spectrum = wave.make_spectrum()
```

  - Spectrum provides plot

```
spectrum.plot()
thinkplot.show()
```

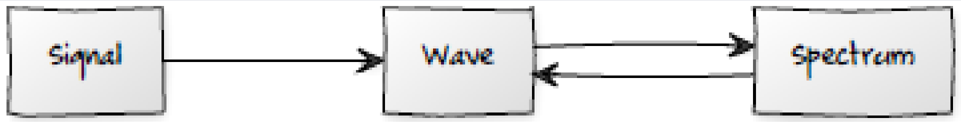


Figure 1.5: Relationships among the classes in `thinkdsp`.

- Spectrum provides three methods that modify the spectrum.
  - low_pass : components above a given cutoff freq. are attenuated by a factor.
  - high_pass : it attenuates components below the cutoff.
  - band_pass : attenuates components in the band of frequencies between two cutoffs.

Spectrums 2

- Generate a high freq. removed wave.
  - Attenuate all freqs. Above 600 Hz by 99%

```
spectrum.low_pass(cutoff=600, factor=0.01)
```

  - To hear the low pass filtered sound.

```
wave = spectrum.make_wave()
wave.play('temp.wav')
```

Play method writes the wave to a file and then plays it.

Wave objects

- thinkdsp.py is a kind of wrapper of NumPy and SciPy.
- A wave object contains three attributes
  - ys : a NumPy array that contains the values in the signal.
  - ts : an array of times where the signal is evaluated.
  - Framerate : the number of samples per unit of time. (usually seconds, days…)
- To modify a wave, you can access the ts and ys directly.
  - Make it louder
  - Make a delay

```
wave.ys *= 2
wave.ts += 1
```

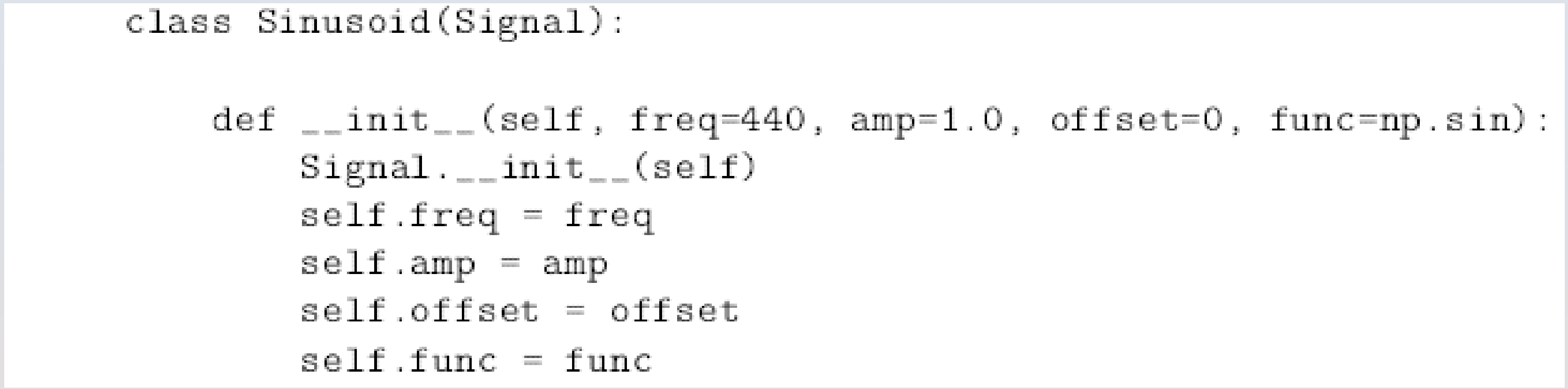
- To modify a wave, use methods provided
  - Make it louder
  - Make a delay

```
wave.scale(2)
wave.shift(1)
```

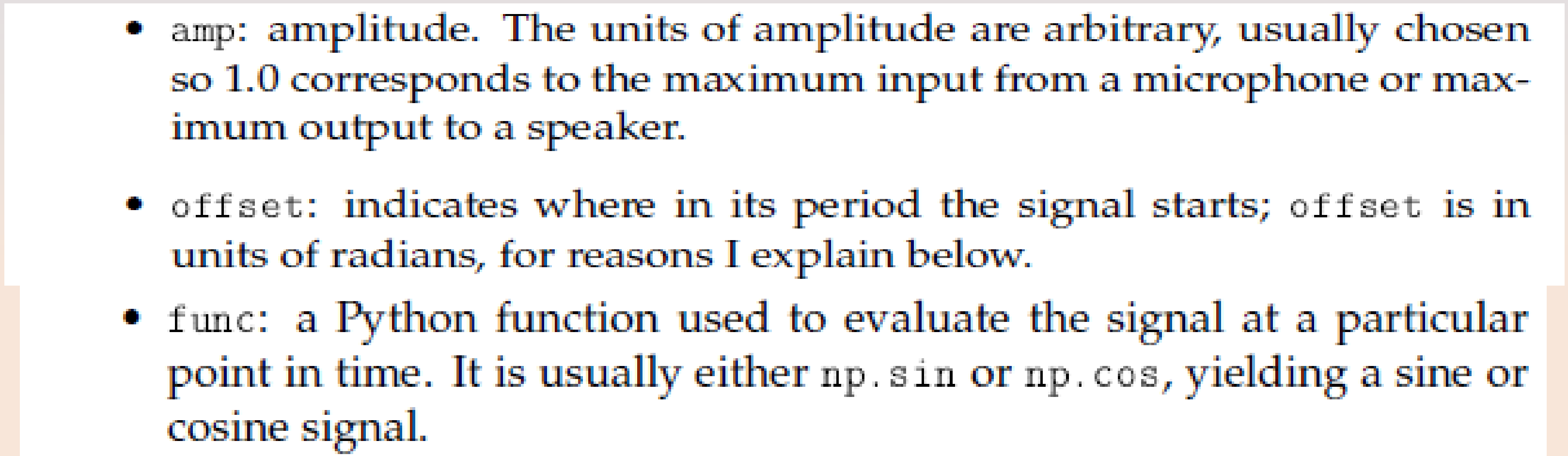
- Read more at http://greenteapress.com/thinkdsp.html.

Signal objects 1

- Signal is a parent class.
  - Provides functions common to all kinds of signals like make_wave.
- Child classes inherit all the methods from



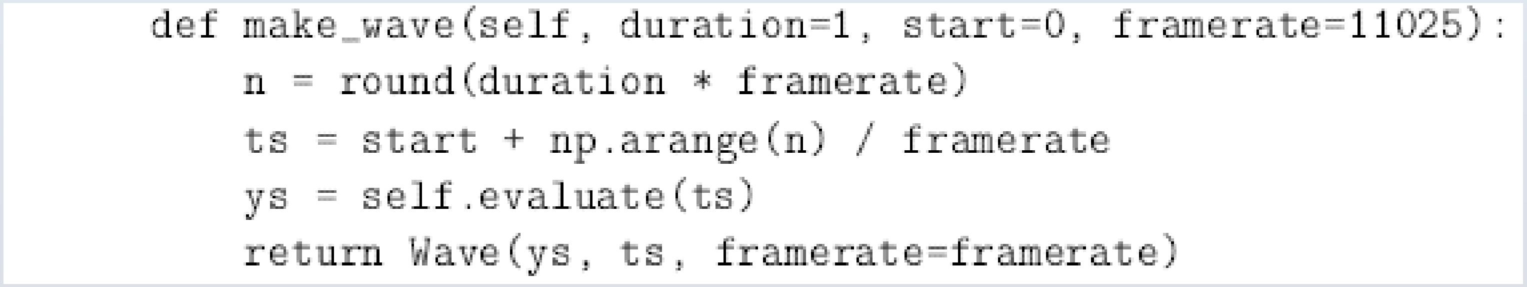
```
class Sinusoid(Signal):

    def __init__(self, freq=440, amp=1.0, offset=0, func=np.sin):
        Signal.__init__(self)
        self.freq = freq
        self.amp = amp
        self.offset = offset
        self.func = func
```



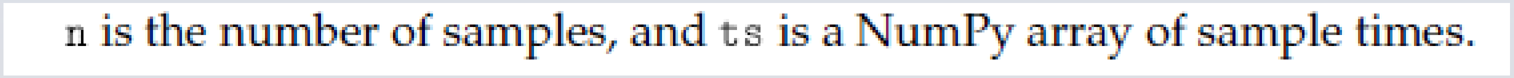
- `amp`: amplitude. The units of amplitude are arbitrary, usually chosen so 1.0 corresponds to the maximum input from a microphone or maximum output to a speaker.
- `offset`: indicates where in its period the signal starts; `offset` is in units of radians, for reasons I explain below.
- `func`: a Python function used to evaluate the signal at a particular point in time. It is usually either `np.sin` or `np.cos`, yielding a sine or cosine signal.

Signal objects 2

- Signal provides make_wave,



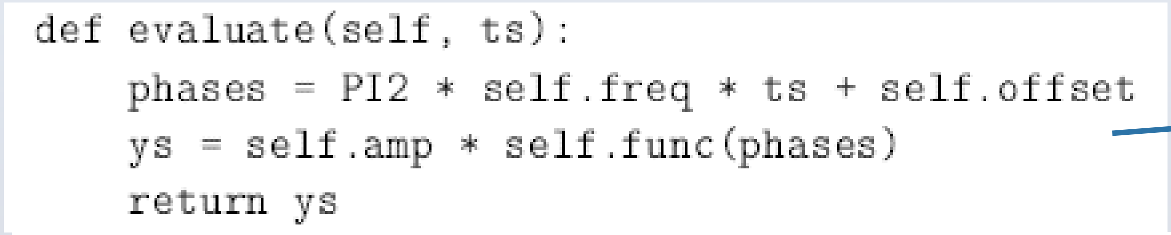
```
def make_wave(self, duration=1, start=0, framerate=11025):
    n = round(duration * framerate)
    ts = start + np.arange(n) / framerate
    ys = self.evaluate(ts)
    return Wave(ys, ts, framerate=framerate)
```



n is the number of samples, and ts is a NumPy array of sample times.

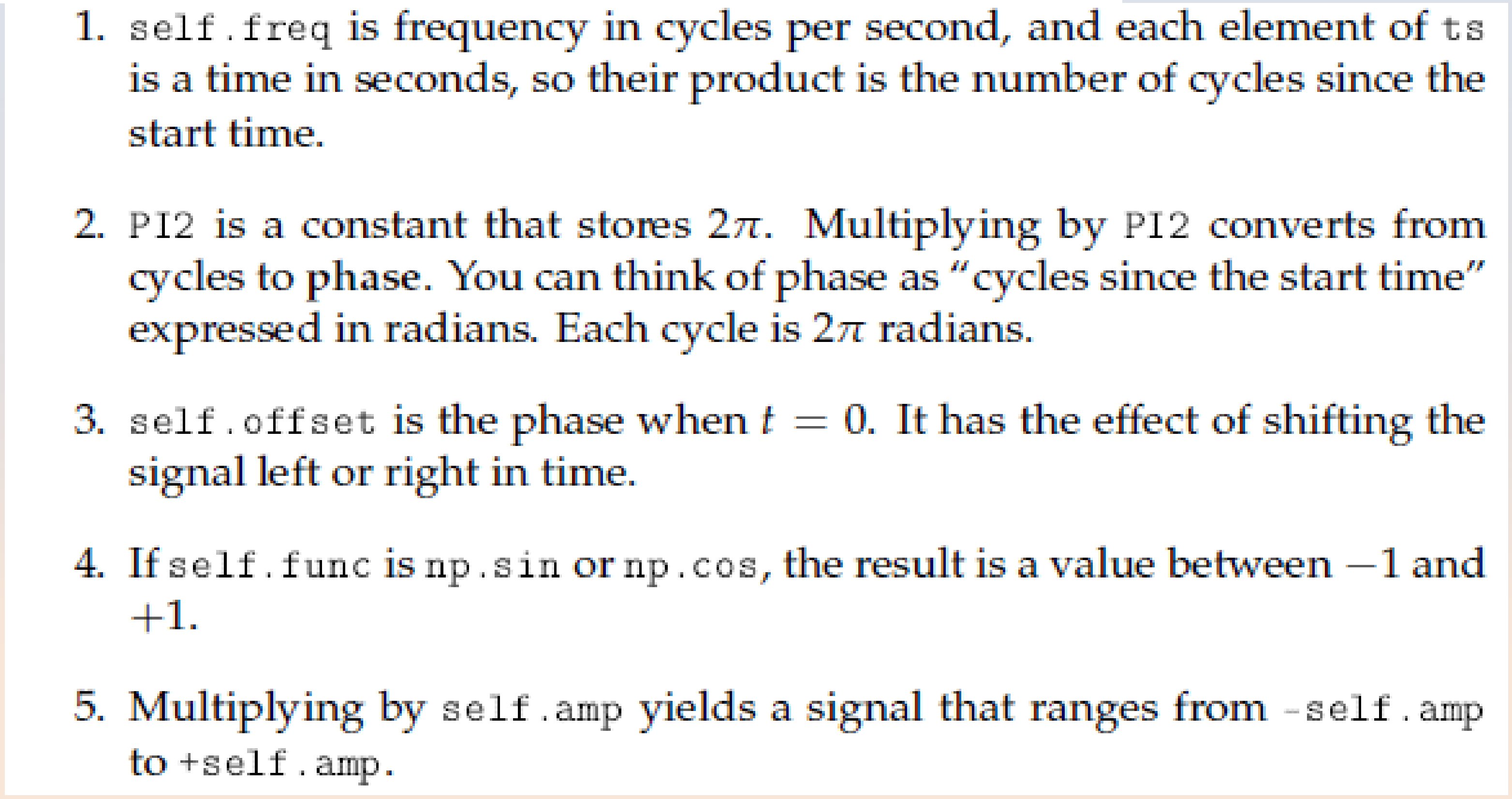
Signal objects 3

- To compute the ys, make_wave invokes evaluate, which is provided by Sinusoid



```
def evaluate(self, ts):
    phases = PI2 * self.freq * ts + self.offset
    ys = self.amp * self.func(phases)
    return ys
```

$$y = A cos(2\pi f t + \theta)$$



1. `self.freq` is frequency in cycles per second, and each element of `ts` is a time in seconds, so their product is the number of cycles since the start time.

2. `PI2` is a constant that stores 2π. Multiplying by `PI2` converts from cycles to phase. You can think of phase as "cycles since the start time" expressed in radians. Each cycle is 2π radians.

3. `self.offset` is the phase when $t = 0$. It has the effect of shifting the signal left or right in time.

4. If `self.func` is `np.sin` or `np.cos`, the result is a value between -1 and $+1$.

5. Multiplying by `self.amp` yields a signal that ranges from `-self.amp` to `+self.amp`.