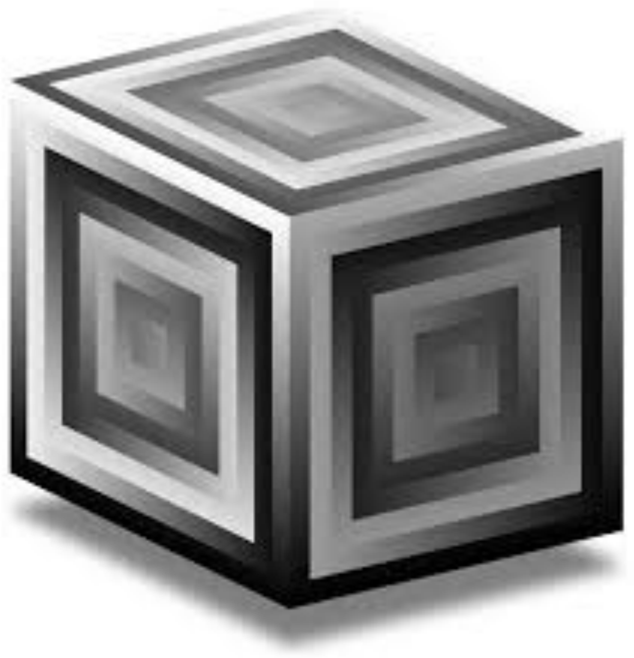
# Types of Waves

## Topics Addressed

- Sawtooth Wave
- Triangle Wave
- Square Wave
- Pulse/Rectangle Wave



## Recap

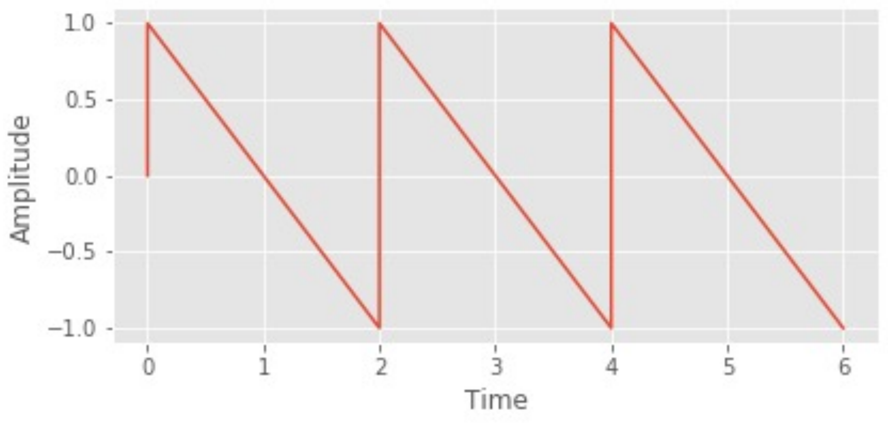
- Sine waves have three important properties: frequency, amplitude and phase
- Complex sounds are sounds that contain two or more sine waves.
- The harmonic series is a specific mathematical combination of sine waves that are perceived by our ears as fusing together into one pitched sound.
    - The lowest note in the harmonic series is the fundamental
    - Sine waves at higher frequencies in the series are called overtones
    - Changes to relative strength of these notes impacts our interpretation of the timbre of the sound
- Intervals (the measure of musical distance) is perceived as the ratio of two frequencies, **not** the absolute difference.

## Fourier Transform

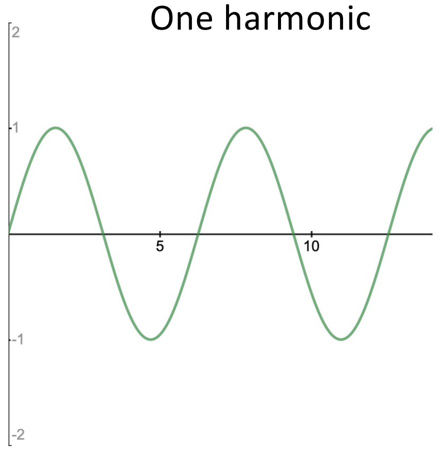
- We have seen in the context of pitch, that many pitches are the result of complex sounds which are simply two or more waves.
- It turns out that ANY periodic signal or sound can be reduced to a combination of weighted sine waves.
  - Attributed to Joseph Fourier who showed how to break down a signal into its constituent parts
- We are going to study several basic waves. All of these can be thought of as combinations of various sine waves.
- We can think of sine waves as the basic building blocks of sound.
- Much more on Fourier and his transform/series later.

## Sawtooth Wave

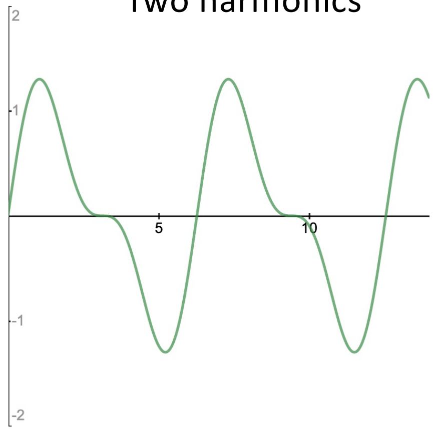
- The sawtooth wave is derived from the frequencies of the harmonic series.
- Harmonics: all harmonics
- Amplitude: 1/(Harmonic Number)
- Optional: Shifting the phase of the even harmonics by 180 degrees will make a sawtooth wave that ramps up instead of ramping down. No difference in how we perceive the sound.



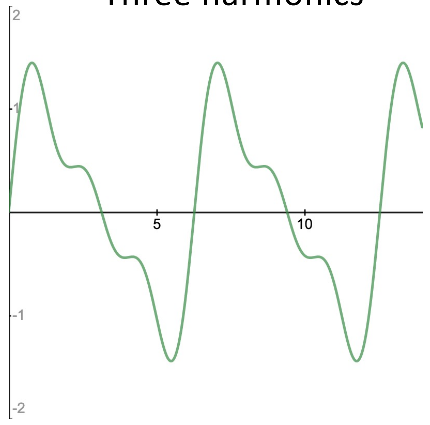
One harmonic



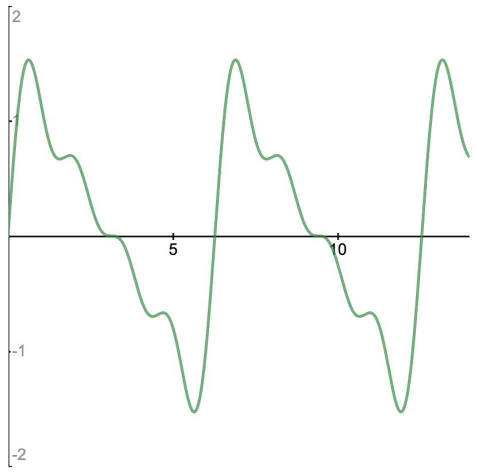
Two harmonics



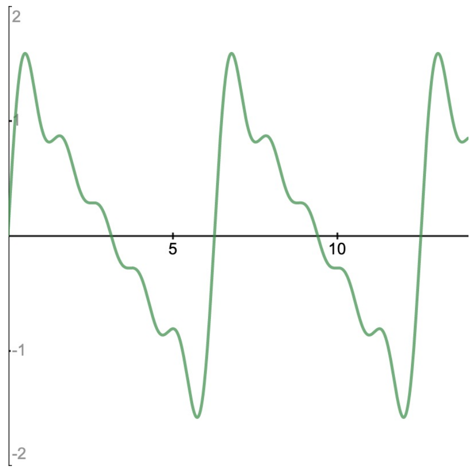
Three harmonics



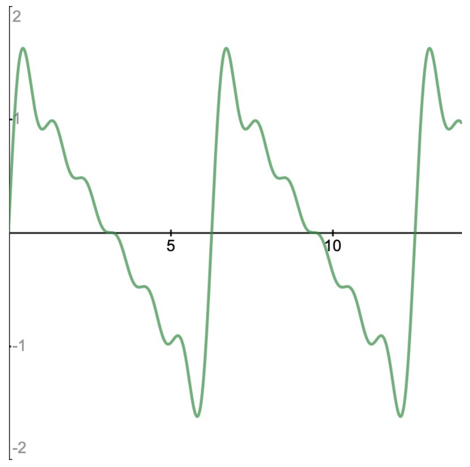
Four harmonics



Five harmonics



Six harmonics



## Additive Synthesis

- Creating a sawtooth wave out of 2 to *n* sine waves is an example of additive synthesis. Additive synthesis is the technique of adding sine waves together to create complex sounds and timbres.
- A sawtooth wave with two harmonics of frequency *f* is equivalent to $\frac{\sin(2\pi f x)}{1} + \frac{\sin(2\pi(2f)x)}{2}$
- A sawtooth wave with three harmonics of frequency *f* is equivalent to $\frac{\sin(2\pi f x)}{1} + \frac{\sin(2\pi(2f)x)}{2} + \frac{\sin(2\pi(3f)x)}{3}$
- In sclang, we write `{SinOsc.ar(440, 0, 1) + SinOsc.ar(880, 0, 0.5} + …}.play`

## Summation Notation of Sawtooth Wave

- If a sawtooth wave is an infinite sum of sine waves, then let's use summation notation to express the wave.
- Consider $g(t)$ to be a sawtooth wave as a function of $t$ (time) in seconds with a fundamental frequency $f$ and fundamental amplitude $A$.
- Then…

$$g(t) = \sum_{n=1}^{\infty} \frac{A}{n} \sin(2\pi f n t)$$

$n$ represents the harmonic number where $n = 1$ is the fundamental

## .dup method

- The `.dup` method is very helpful in sclang and you will use it often. We will need it to make a sawtooth wave out of simple sine waves.

```
2.dup; // When called on a number, it duplicates the number,
       // storing the contents in an array
2.dup(3); // Duplicates the number three times
{|x| x}.dup(20); // On a function, it creates and evaluates n
                 // functions where the ith function
                 // between 0 and n-1 is passed i
440 ! 2;  // ! is a shorthand for dup
```

## Summing Arrays

- In Python, the + operator concatenated two lists together. In sclang, the + operator on arrays adds each element pointwise.

```
[1, 2] + [3, 4] // evalutes to [4, 6]
[1, 2, 3] + [-1, -2, -3] // evalutes to [0, 0, 0]
```

- This is particularly useful because we can sum sine waves together using + operators. Recall that stereo audio signals are returned in an array of two channels (left and right)

## Sum Two Audio Signals

- We can use the same principle to some two audio signals. In the example below, the sum of the two signals produces an audio signal with 400Hz and 600Hz in the left speaker and 800Hz and 1000Hz in the right speaker.
- The sum of the two arrays evaluates to an array representing the left channel audio and right channel audio.

```
var sig1 = [SinOsc.ar(400), SinOsc.ar(800)]
var sig2 = [SinOsc.ar(600), SinOsc.ar(1000)]
sig1 + sig2
```

# Sawtooth Wave in Code

```
~saw = {
  arg freq = 300, fundAmp = 0.2;
  var numHarmonics = 25;
  var sig = [0, 0];

  for(1, numHarmonics, { |n| // harmonic number
    // add an array of two sines wave
    sig = sig + SinOsc.ar(freq * n ! 2, 0, fundAmp/n)
  });

  sig // return value is the array for left/right speaker
};
```

## LFSaw and Saw

- SuperCollider provides two different UGens to create saw waves.
- One is Saw and the other is LFSaw.
  - Saw is a band limited oscillator. More on that when we discussing aliasing.
  - LFSaw is a non-bandlimited oscillator.
  - For now, band limited oscillators have fewer harmonics. But we won’t hear too much of a difference, if any.

```
~saw.play(args: [\freq, 300,
                 \funAmp, 0.1]);
{LFSaw.ar(300, 0, 0.1)!2}.play;
{Saw.ar(300, 0.1)!2}.play;
```

## Aside: Caveats with Functions that use .play

- Avoid using UGens defined outside the context of a function.
- The below example seems like it should work perfectly fine but will fail silently.

```
// Buggy!!
var test = SinOsc.ar(440, mul: 0.1);
{test}.play;
```

## Aside: Caveats with Functions that use .play

- In general, you should be careful when using loops and conditionals inside a function that will later use the .play method
- Here is a seemingly innocuous snippet of code:

```
~buggy = {
    |whichOne| // A boolean as argument
    if (whichOne, {
        SinOsc.ar(440);
    }, {
        Pulse.ar(440);
    });
};
```

## Aside: Caveats with Functions that use .play

- The issue with the previous code is that the server scsynth can only handle mathematical calculations.  Booleans are not allowed.  Sclang describes calculations that the server will perform in the future.
- Because the previous code uses a conditional to choose between two UGens, a boolean is unfortunately necessary.
- This specific error is called "Non-Boolean in test"
- For more info on this particular issue visit here: http://supercollider.sourceforge.net/wiki/index.php/If_statements_in_a_SynthDef
- You **can** use conditionals/loops that evaluate to numerical results (see Triangle Wave example)

# Aside: Caveats with Functions that use .play

Someone has the great idea of innocuously attempting to abstract our previous saw function by allowing a parameter that determines the number of harmonics in the wave.

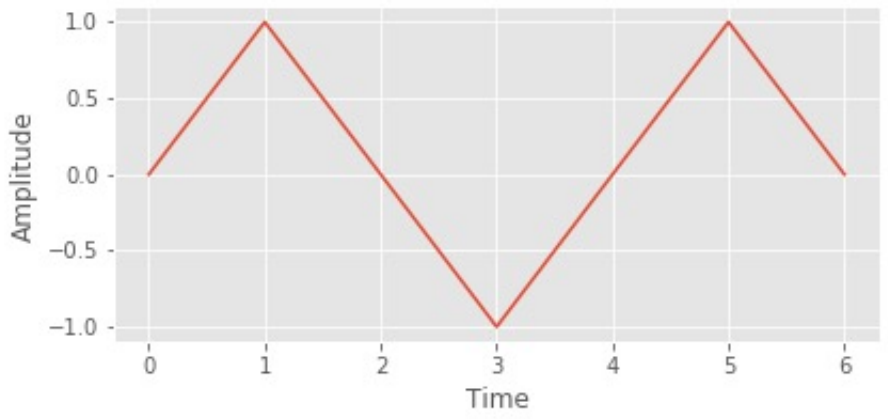
```
(
~buggySaw = {
    arg freq = 300, funAmp = 0.6, numHarms = 30;
    var sig = {
        |i| // One less than the harmonic num which are one indexed (not zero)
        SinOsc.ar(freq * (i + 1), 0, funAmp/(i + 1)) // Freq and amp come from harmonic number
    }.dup(numHarms).sum;
    sig ! 2; // Return the stereo signal.  ! equivalent to dup.
};
)
```

## Aside: Caveats with Functions that use .play

- What went wrong with the previous example?
- The server needs to know exactly how many UGens are needed for a given function.
- Dynamically allocating UGens at runtime is **not** allowed by the server.
- For more information on the topic, please visit https://supercollider.github.io/tutorials/error-primitive-basicnew-failed

## Triangle Wave

- Harmonics: only odd numbered harmonics
- Amplitude: 1/(Harmonic Number)^2
- Phase: Every other harmonic is 180 degrees out of phase



## Triangle Wave

$$g(t) = \sum_{i=1}^{\infty} (-1)^i \frac{A}{n^2} \sin(2\pi f n t)$$

where $n = 2i - 1$

This summation makes use of the fact that $-\sin(x) = \sin(x + \pi)$ to handle alternating odd harmonics 180 degrees out of phase.

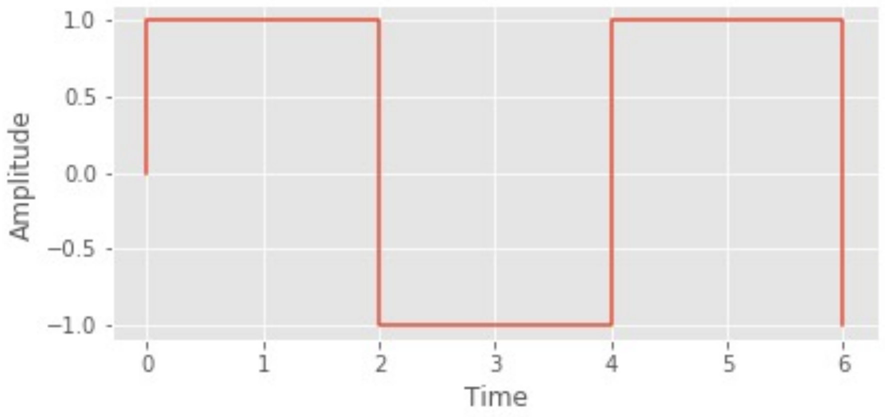
# Triangle Wave in Code

```
~triangle = {
  arg freq = 300, fundAmp = 0.3;
  var numHarmonics = 30;
  var sig = [0, 0];

  for(1, numHarmonics, {
    |i|
    var n = 2 * i - 1; // Create the harmonic number
    var phase = if(i % 2 == 0, {0}, {pi}); // Alternate phase
    sig = sig + SinOsc.ar(freq * n, phase, fundAmp * (1/n.squared));
  });

  sig
}
```

Note: The if statement is okay here because the true and false expressions evaluate to numerical values!

## Square Wave

- Harmonics: Odd Numbered Harmonics
- Amplitudes: 1/Harmonic Number
- Phase: All harmonics in phase



# Exercise: Square Wave in Code

```
~square = {
  arg freq = 300, fundAmp = 0.3;
  var numHarmonics = 30;
  var sig = [0, 0];

  for(1, numHarmonics, {
    |i|
    var n = 2 * i - 1; // Create the harmonic number
    sig = sig + SinOsc.ar(freq * n, 0, fundAmp/n);
  });

  sig
}
```

## Exercise: Write the partials of a square wave

Write the partials of a square wave using summation notation. Recall that a square wave has only odd harmonics, the amplitudes of the harmonics are (1/harmonic number), and all harmonics are in phase.

$$A \sum_{i=1}^{\infty} \frac{\sin(2\pi n f t)}{n}$$

$n = 2i - 1$. The positive integers for $i$ produce the harmonics (i.e., $n$) of 1, 3, 5, 7, etc.

## Exercise: Adjust the Phase of a Square Wave

- Adjust the phase of a square wave of $f = 1$ by 0.5 seconds by shifting the square wave of frequency $f$ to the right.

Original Signal

$$g(t) = A\sum_{i=1}^{\infty}\frac{\sin(2\pi nt)}{n}$$

What would the result of $g(t) + h(t)$ be?

ANSWER: 0!

New Signal

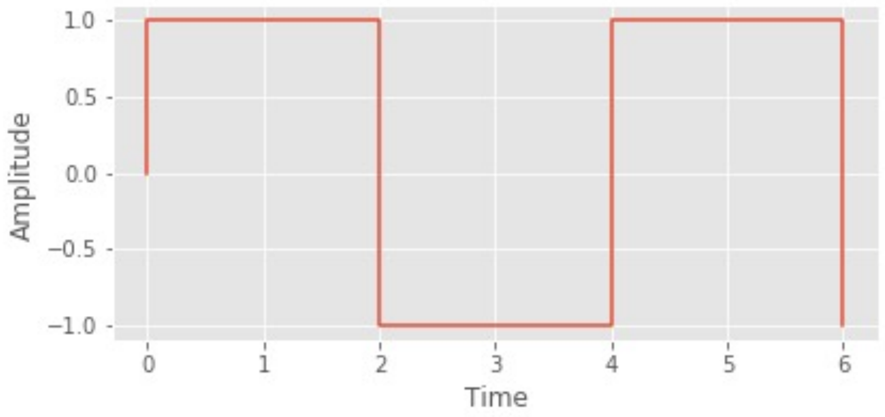
$$h(t) = g(t - 0.5)$$

$$h(t) = A\sum_{i=1}^{\infty}\frac{\sin(2\pi n(t - 0.5))}{n}$$

$$h(t) = A\sum_{i=1}^{\infty}\frac{\sin(2\pi nt - \pi n)}{n}$$

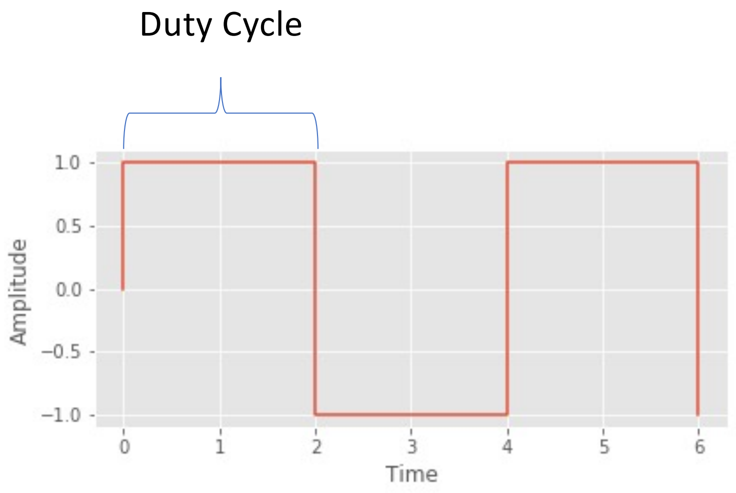
## Pulse/Rectangle Wave

- Pulse waves or rectangle waves are generalizations of the square waves.
- A square wave's period has equal portion at high and low amplitudes. In a rectangle wave, those proportions need not be equal. Therefore, all square waves are rectangle waves but not vice versa.



## Pulse/Rectangle Waves

- We call the duty cycle the proportion of the wave’s period at a high amplitude. The duty cycle ranges from 0 to 1.
- Categorizing the relative strengths of the harmonics can be complicated for a rectangle wave. If your curious, check out the Wikipedia page on pulse waves which details the Fourier transform of the pulse wave into its constituent sine waves: https://en.wikipedia.org/wiki/Pulse_wave



A square wave’s duty cycle is 0.5, meaning that 50% of the wave’s period is at a high amplitude.

## Pulse/Rectangle Waves

The equation for a Pulse wave is more complicated but it still relies upon summing sinusoids (in this case cosine waves). Note that the sine in this equation is simply a scaling factor for amplitude as it is not a function of time.

$$g(t) = dA + \sum_{n=1}^{\infty} \frac{2A}{\pi n} \sin(\pi d n) \cos(2\pi f n t)$$

# Pulse Wave in Code

```
~pulse = {
  arg freq = 300, fundAmp = 0.2, d = 0.5;
  var numHarmonics = 30;
  var sig = [0, 0];

  for(1, numHarmonics, {
    |n| // harmonic number
    var harmonic = (2 * fundAmp)/(pi * n) * (n * pi * d).sin * SinOsc.ar(freq * n, pi/2);
    sig = sig + harmonic;
  });

  (d * fundAmp) + sig // 2/pi multiplies each element in the array
}
```

## Triangle/Rectangle Wave in SuperCollider

- SuperCollider offers one kind of triangle wave, a non-bandlimited UGen called `LFTri`
- SuperCollider offers two kinds of pulse wave generators
  - The class `LFPulse` – a non-bandlimited pulse wave
  - The class `Pulse` – a bandlimited pulse wave
  - Note that in both of these UGens the duty cycle is called width
- Check out the UGen `Klang` which offers a bank of sine waves that can be useful for creating your own waves via additive synthesis.

## Visualizing Waveforms

- SuperCollider provides two ways for visualizing your waveforms.
  - The method `.plot` -> This can be applied to any function that returns a UGen
    - This will plot the waveform and will be helpful for you on assignments to verify that the waveform you are trying to produce is actually the waveform you are using
  - The class `FreqScope` -> Monitors audio output by analyzing the frequency spectrum
    - Not a method. You need to create a new `FreqScope` object which will bring up a GUI display
    - The FreqScope will display all frequencies in the output audio and their relative strength.

## Experiment

- We've introduced the notion of band limited oscillators. Oscillator is simply another term for periodic waveforms.
- The class `LFTri` is a non-band limited oscillator, meaning that it has all the harmonics in our audible frequency spectrum (20Hz to 20000Hz).
- With our `~triangle` oscillator that we made using additive synthesis , let's change the number of harmonics from 20 to 5. So here we will severely bandlimit our triangle wave.
- Now let's compare how they sound. Do we hear an audible difference? Then, let's compare using a `FreqScope`.

```
~triangle = {
  arg freq = 300, fundAmp = 0.3;
  var numHarmonics = 30;
  var sig = [0, 0];

  for(1, numHarmonics, {
    |i|
    var n = 2 * i - 1; // Create the harmonic number
    var phase = if(i % 2 == 0, {0}, {pi}); // Alternate phase
    sig = sig + SinOsc.ar(freq * n, phase, fundAmp * (1/n.squared));
  });

  sig
}

f = FreqScope.new;
~triangle.play(args: [\freq, 400, \fundAmp, 0.4]);
{LFTri.ar(400, 0, 0.6) ! 2}.play;
```

## Analysis

- You can see that the frequency spectrum of our band limited Triangle wave is missing many of the upper harmonics that `LFTri` has.
- How does this affect our perception of the sound?
  - More harmonics tend to have "buzzier" sound to them
  - With only five harmonics our triangle wave can sound more muted and maybe even "duller" in sound (i.e., less bright)
  - No difference in terms of pitch. We still perceive the same fundamental frequency but the change in partials affects how perceive the timbre of the two oscillators

## Examples in Music

- These waveforms are ubiquitous in all popular music.
- Square Wave
  - E.T. by Katy Perry ft. Kanye – the 16th note synth lead
- Sine Wave
  - Big Poppa by The Notorious B.I.G. – the high lead
- Sawtooth Wave
  - Head like a Hole by Nine Inch Nails – bass line synth
- Triangle Wave
  - Flashlight by Parliament – high lead; the bass line is a square wave
- Any song with synthesizers is more than likely using one of these basic waveforms as its initial sound source and then processes them with a variety of effects.

# In Summary

- Any periodic signal can be constructed from a sum of sine waves
  - The process of creating complex sounds from sine waves or other constituent parts is called additive synthesis
- The sine wave, sawtooth wave, pulse wave, and triangle waves are classic waveforms since the advent of electronic music
  - They are used **everywhere** in the music we hear today and were/are the basis of many synthesizers
- Sawtooth waves, pulse waves, and triangle waves come in bandlimited and non-bandlimited forms
  - Non-bandlimited forms can be produce strange artifacts at higher frequencies (see aliasing coming up)
  - Bandlimited forms are "safer" but are not as rich harmonically – they are not true representation of the waveforms