JS

Beats in the Browser

@rowdyrabouw

ローランド株式会社


Roland

Beats in the Browser

@rowdyrabouw


Roland
TR-909
RHYTHM COMPOSER

ROWDY RABOUW

I apologize, but as of my knowledge cutoff in September 2021, I couldn't find any notable or well-known public figures or individuals named "Rowdy Rabouw."

ROWDY RABOUW

Front-End Focused Senior DevOps Engineer

Build my first website in 1996

Used to be a Club DJ in the late 80's and early 90's

WEB AUDIO API

The Web Audio API is a JavaScript API that provides a set of audio-related functionalities for creating and manipulating audio content in web applications.

AUDIOCONTEXT

The AudioContext represents an audio-processing graph containing audio sources, nodes, and audio destinations.

```
const ctx = new AudioContext();
```

OSCILLATORNODE

The AudioContext represents an audio-processing graph containing audio sources, nodes, and audio destinations.

```
const ctx = new AudioContext();
const osc = new OscillatorNode(ctx, {


});
```

FREQUENCY

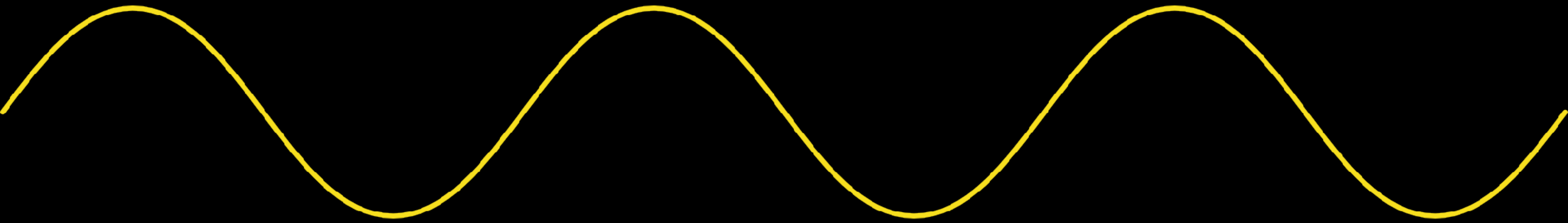
The frequency property of the OscillatorNode determines the pitch of the generated sound. It represents the number of cycles per second, measured in Hertz (Hz).

```
const ctx = new AudioContext();
const osc = new OscillatorNode(ctx, {
  frequency: 440,

});
```

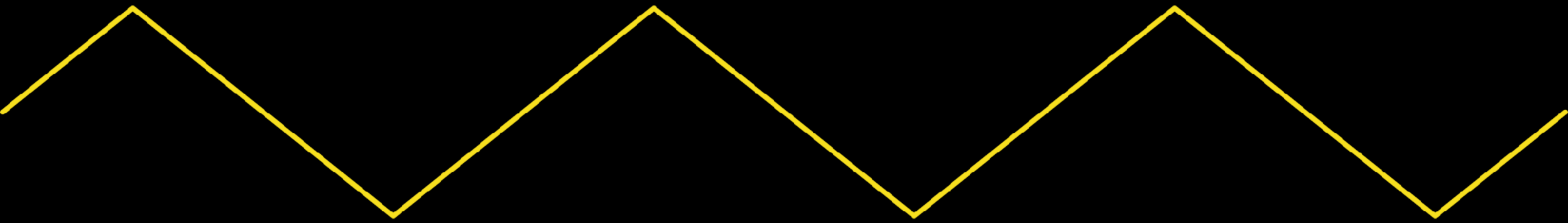
WAVEFORM

The OscillatorNode can generate different types of waveforms.
Each waveform has a distinct timbre and harmonic content.

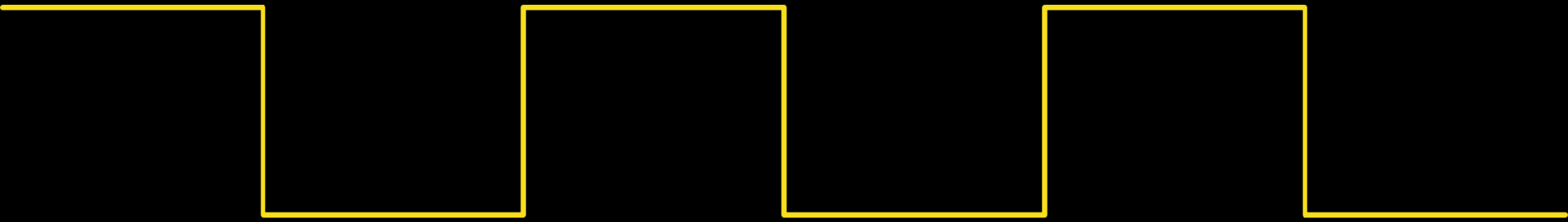
SINE



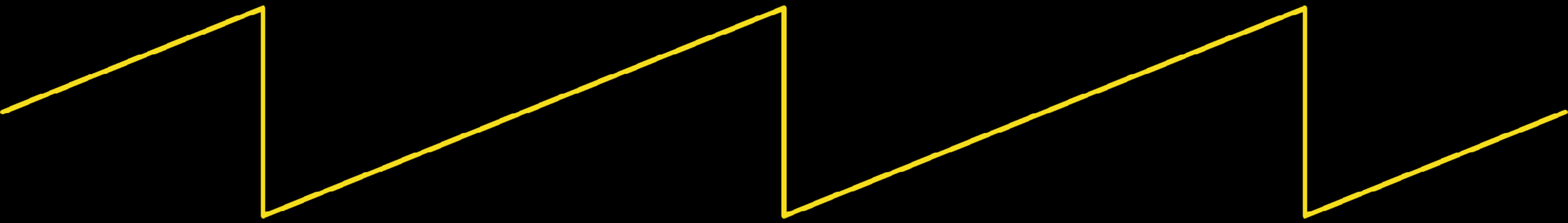
TRIANGLE



SQUARE



SAWTOOTH



```
const ctx = new AudioContext();
const osc = new OscillatorNode(ctx, {
  frequency: 440,
  type: "sine",
});
```

```
const ctx = new AudioContext();
const osc = new OscillatorNode(ctx, {
  frequency: 440,
  type: "sine",
});

osc.connect(ctx.destination);
osc.start(ctx.currentTime);
osc.stop(ctx.currentTime + 0.3);
```

DEMO GENERATING SOUNDS

WEB MIDI API

The Web MIDI API is a JavaScript API that allows web applications to communicate and interact with MIDI (Musical Instrument Digital Interface) devices connected to a user's computer or device.

Beats in the Browser

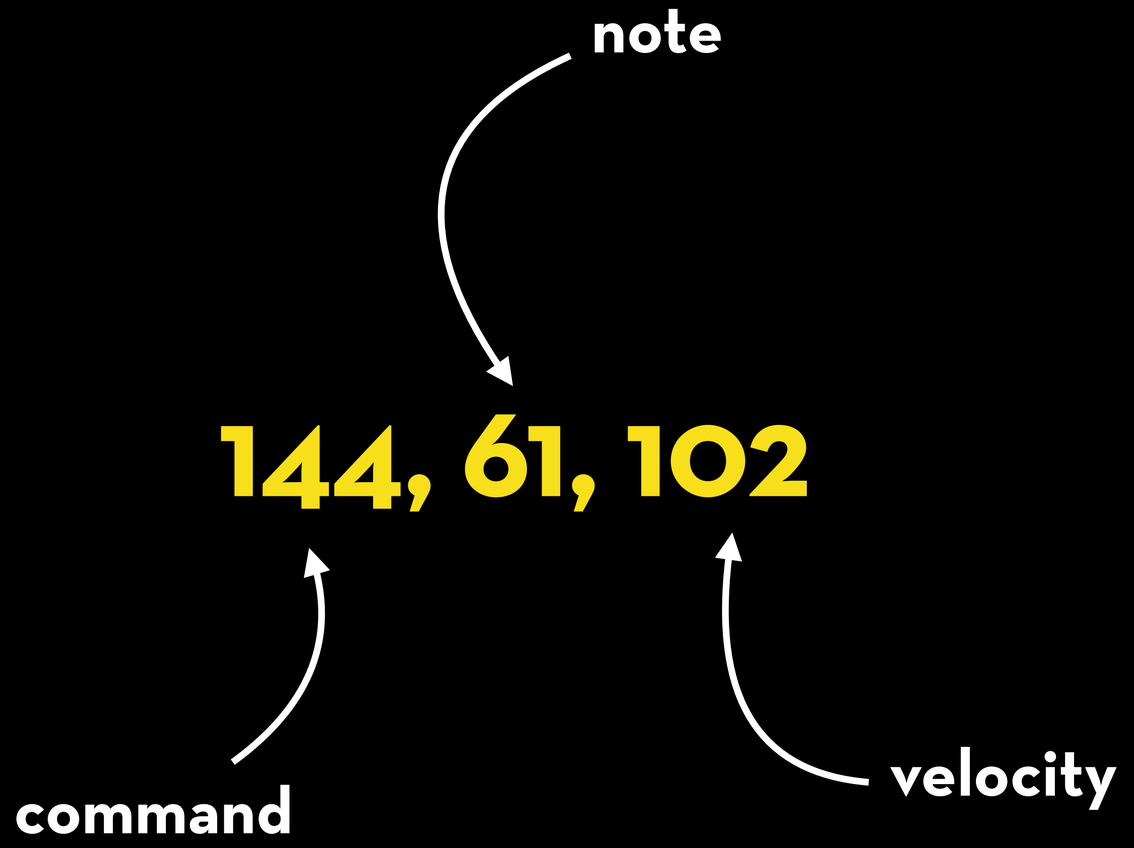
@rowdyrabouw

```
if (navigator.requestMIDIAccess) {
  navigator.requestMIDIAccess().then(succes, failure);
}
```

```
const succes = (midiAccess) ⇒ {

  midiAccess.addEventListener("statechange", (event) ⇒ {
    console.info(event.port.manufacturer); // AKAI
    console.info(event.port.name);         // MPK mini 3
  });









}
```

```
const succes = (midiAccess) ⇒ {

  midiAccess.addEventListener("statechange", (event) ⇒ {
    console.info(event.port.manufacturer); // AKAI
    console.info(event.port.name);         // MPK mini 3
  });

  const inputs = midiAccess.inputs;



}
```

```
const succes = (midiAccess) ⇒ {

  midiAccess.addEventListener("statechange", (event) ⇒ {
    console.info(event.port.manufacturer); // AKAI
    console.info(event.port.name);         // MPK mini 3
  });

  const inputs = midiAccess.inputs;

  inputs.forEach((input) ⇒ {
   input.addEventListener("midimessage", handleInput);
  });
}
```


note
144, 61, 102
command
velocity

DEMO
MIDI - COMMANDS

```
const handleInput = (input) ⇒ {

 const command  = input.data[0];
 const note     = input.data[1];
 const velocity = input.data[2];

 switch (command) {
    case 144: //noteOn
      noteOn(note, velocity);
      break;
    case 128: // noteOff
      noteOff(note);
      break;
 }
}
```

```
const noteOn = (note, velocity) => {

}
```

```
const noteOn = (note, velocity) => {
  const oscillator = audioContext.createOscillator();

}
```

```
const noteOn = (note, velocity) => {
  const oscillator = audioContext.createOscillator();

  const oscillatorGain = audioContext.createGain();
  oscillator.connect(oscillatorGain);
  oscillator.gain = oscillatorGain;
```

```
}
```

```
const noteOn = (note, velocity) => {
  const oscillator = audioContext.createOscillator();

  const oscillatorGain = audioContext.createGain();
  oscillator.connect(oscillatorGain);
  oscillator.gain = oscillatorGain;

  const velocityGain = audioContext.createGain();
  oscillatorGain.connect(velocityGain);
  velocityGain.connect(audioContext.destination);
```

```
}
```

```
const noteOn = (note, velocity) => {
  const oscillator = audioContext.createOscillator();

  const oscillatorGain = audioContext.createGain();
  oscillator.connect(oscillatorGain);
  oscillator.gain = oscillatorGain;

  const velocityGain = audioContext.createGain();
  oscillatorGain.connect(velocityGain);
  velocityGain.connect(audioContext.destination);

  const velocityGainAmount = (1 / 127) * velocity;
  velocityGain.gain.value = velocityGainAmount;




}
```

```
const noteOn = (note, velocity) => {
  const oscillator = audioContext.createOscillator();

  const oscillatorGain = audioContext.createGain();
  oscillator.connect(oscillatorGain);
  oscillator.gain = oscillatorGain;

  const velocityGain = audio
  oscillatorGain
  veloci
```

```
let frequency = 440;
const midiToFrequency = (number) => {
  return (frequency / 32) * (2 ** ((number - 9) / 12));
}
```

```
                                    ocity;
                                  ityGainAmount;
  .type = 'sine';
  oscillator.frequency.value = midiToFrequency(note);
```

```
}
```

```
const noteOn = (note, velocity) => {
  const oscillator = audioContext.createOscillator();

  const oscillatorGain = audioContext.createGain();
  oscillator.connect(oscillatorGain);
  oscillator.gain = oscillatorGain;

  const velocityGain = audioContext.cre
  oscillatorGain.connect(vel
  velocityGain.co

  const oscillators = [];

  .frequency.value = midiToFrequency(note);

  oscillators[note.toString()] = oscillator;

}
```

```
const noteOn = (note, velocity) => {
  const oscillator = audioContext.createOscillator();

  const oscillatorGain = audioContext.createGain();
  oscillator.connect(oscillatorGain);
  oscillator.gain = oscillatorGain;

  const velocityGain = audioContext.createGain();
  oscillatorGain.connect(velocityGain);
  velocityGain.connect(audioContext.destination);

  const velocityGainAmount = (1 / 127) * velocity;
  velocityGain.gain.value = velocityGainAmount;

  oscillator.type = 'sine';
  oscillator.frequency.value = midiToFrequency(note);

  oscillators[note.toString()] = oscillator;

  oscillator.start();
}
```

```
const noteOff = (note) => {

}
```

```
const noteOff = (note) => {
  const oscillator = oscillators[note.toString()];
  const oscillatorGain = oscillator.gain;




}
```

```
const noteOff = (note) => {
  const oscillator = oscillators[note.toString()];
  const oscillatorGain = oscillator.gain;

  oscillatorGain.gain.setValueAtTime(oscillatorGain.gain.value,
                                     audioContext.currentTime);
  oscillatorGain.gain.exponentialRampToValueAtTime(0.0001,
                                   audioContext.currentTime + 0.03);









}
```

```
const noteOff = (note) => {
  const oscillator = oscillators[note.toString()];
  const oscillatorGain = oscillator.gain;

  oscillatorGain.gain.setValueAtTime(oscillatorGain.gain.value,
                                     audioContext.currentTime);
  oscillatorGain.gain.exponentialRampToValueAtTime(0.0001,
                                   audioContext.currentTime + 0.03);

  setTimeout(() => {
    oscillator.stop();
    oscillator.disconnect();
  }, 50);


}
```

```
const noteOff = (note) => {
  const oscillator = oscillators[note.toString()];
  const oscillatorGain = oscillator.gain;

  oscillatorGain.gain.setValueAtTime(oscillatorGain.gain.value,
                                     audioContext.currentTime);
  oscillatorGain.gain.exponentialRampToValueAtTime(0.0001,
                                   audioContext.currentTime + 0.03);

  setTimeout(() => {
    oscillator.stop();
    oscillator.disconnect();
  }, 50);

  delete oscillators[note.toString()];
}
```

DEMO
MIDI - SYNTHESISING

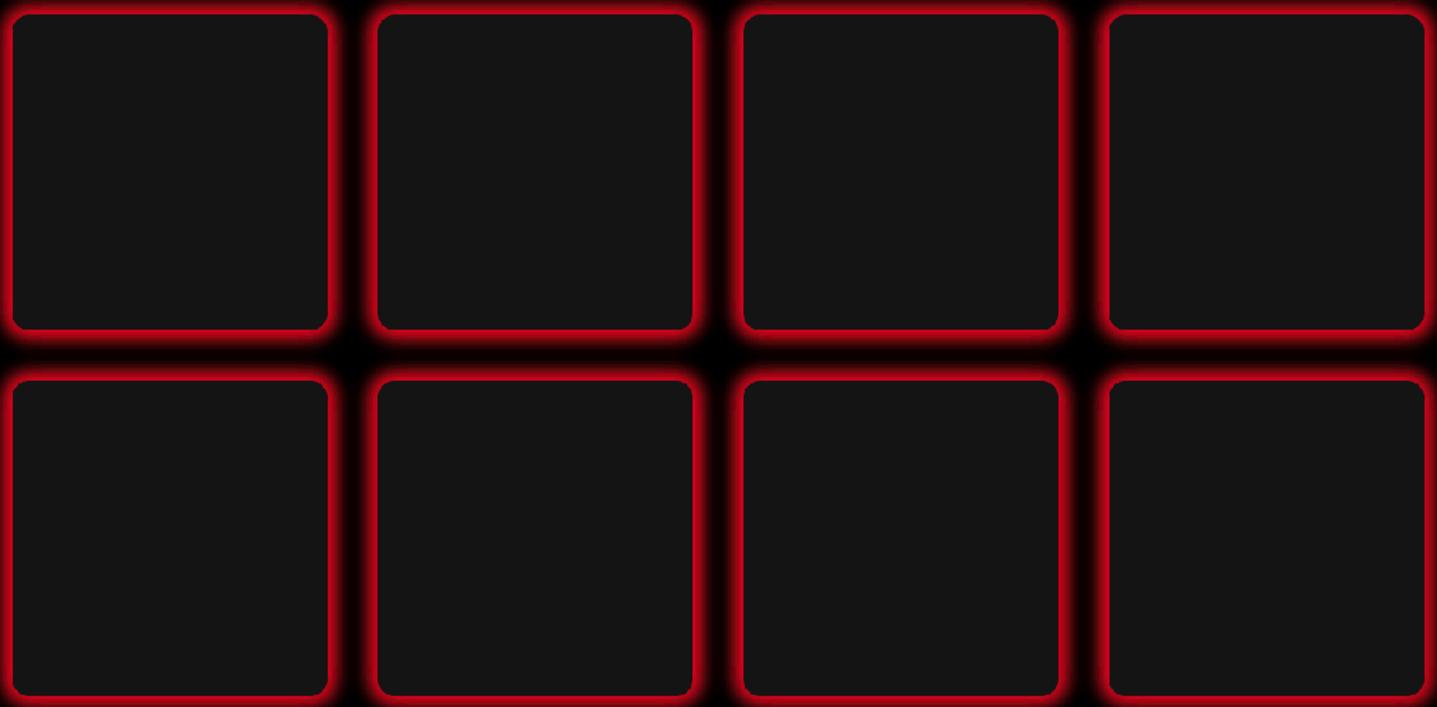
Beats in the Browser

@rowdyrabouw

DEMO SYNTHESISING DRUMS

WAV (WAVEFORM AUDIO FILE FORMAT)

A WAV file is used for storing uncompressed audio data,
resulting in high-quality audio reproduction.
It was developed by Microsoft and IBM in 1991.



```
<button class="pad" data-note="16" data-wav="909-kick"></button>
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>

const pads = document.querySelectorAll(".pad");
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>


const pads = document.querySelectorAll(".pad");
pads.forEach((pad) ⇒ {



});
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>


const pads = document.querySelectorAll(".pad");
pads.forEach((pad) ⇒ {
  pad.addEventListener("click",       () ⇒ {


  });
});
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>


const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {
  pad.addEventListener("click", async () => {
    const sample = await loadFile(`${pad.dataset.wav}.wav`);

  });
});
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>


const pads = document.querySelectorAll(".pad");
pads.forEach((pad) ⇒ {
  pad.addEventListener("click", async () ⇒ {
    const sample = await loadFile(`${pad.dataset.wav}.wav`);
    playWav(sample);
  });
});
```

Beats in the Browser

@rowdyrabouw

```
<button class="pad" data-note="16" data-wav="909-kick"></button>


const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {
  pad.addEventListener("click", async () => {
    const sample = await loadFile(`${pad.dataset.wav}.wav`);
    playWav(sample);
  });
});


const handleInput = (input) => {
  const note = input.data[1];
  const pad = document.querySelector(`[data-note="${note}"]`);
  pad.click();
};
```

```
const loadFile =          (wav) => {


};

const playWav = (audioBuffer) => {


};
```

```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);



};

const playWav = (audioBuffer) ⟹ {



};
```

```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();


};


const playWav = (audioBuffer) ⟹ {



};
```

```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);

};


const playWav = (audioBuffer) ⟹ {



};
```

```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⟹ {



};
```

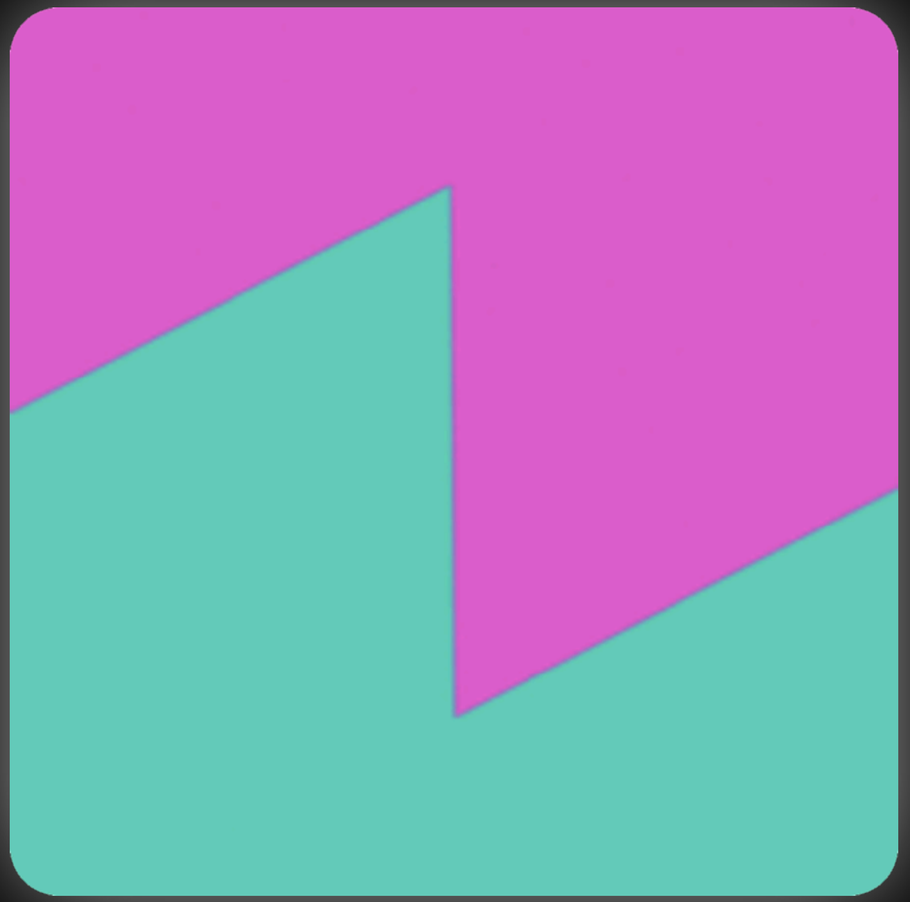
```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⟹ {



};
```

```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⟹ {
  const wav = new AudioBufferSourceNode(ctx, { buffer: audioBuffer });


};
```

```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⟹ {
  const wav = new AudioBufferSourceNode(ctx, { buffer: audioBuffer });
  wav.connect(ctx.destination);

};
```

```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


const playWav = (audioBuffer) ⟹ {
  const wav = new AudioBufferSourceNode(ctx, { buffer: audioBuffer });
  wav.connect(ctx.destination);
  wav.start();
};
```

DEMO
MIDI - SAMPLING



TONE.JS

Tone.js is an open-source JavaScript library that provides a framework for creating interactive and dynamic music and sound in web applications.

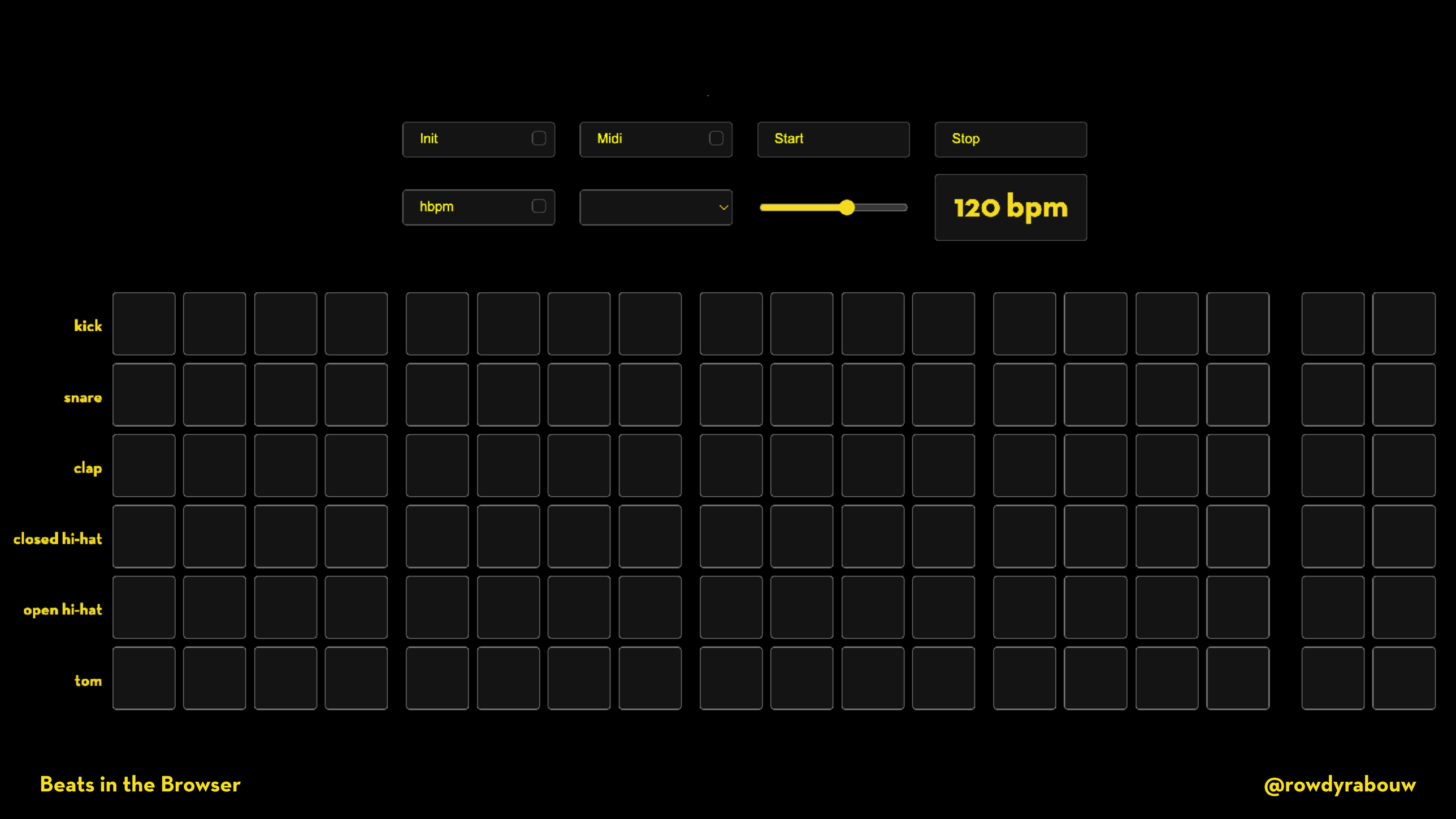
```
const kick = new Tone.Player('wav/909-kick.wav').toDestination();
```

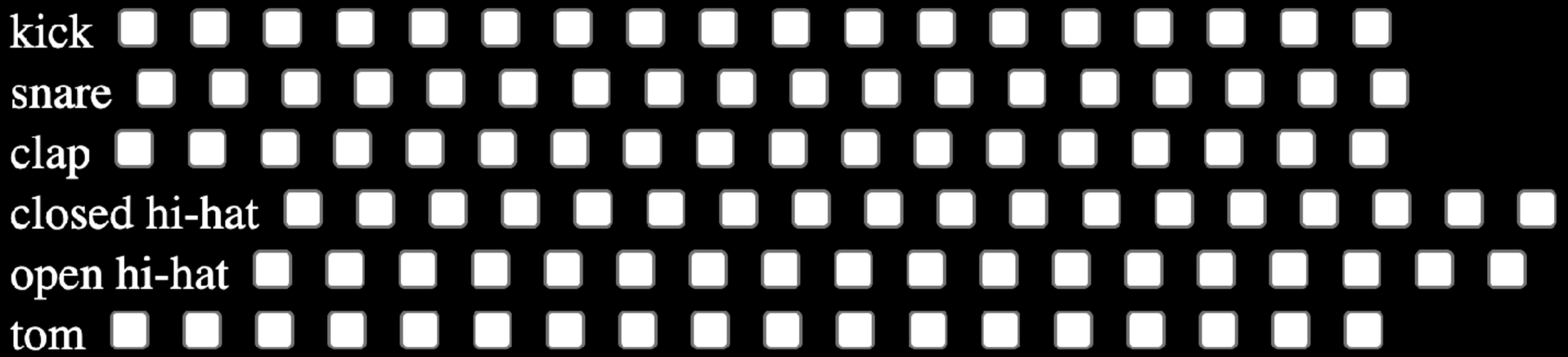
```
const kick = new Tone.Player('wav/909-kick.wav').toDestination();

kick.start(time);
```

SEQUENCER

A sequencer is a device or software application used in music production to create, edit, and arrange musical sequences or patterns. It is a fundamental tool in electronic music production.


Roland
TR-909
RHYTHM COMPOSER


Init
Midi
Start
Stop
hbpm
120 bpm
kick
snare
clap
closed hi-hat
open hi-hat
tom
Beats in the Browser
@rowdyrabouw


kick
snare
clap
closed hi-hat
open hi-hat
tom


AKAI
professional
LPD8
LAPTOP PAD CONTROLLER
PAD 1
PAD 2
PAD 3
PAD 4
PAD 5
PAD 6
PAD 7
PAD 8
PROG 1
PROG 2
PROG 3
PROG 4
PROGRAM
PAD
CC
PROG CHNG
K1
K2
K3
K4
K5
K6
K7
K8
MIN
MAX

DEMO
SEQUENCER

WEB BLUETOOTH API

The Web Bluetooth API allows web applications to interact with Bluetooth devices. It provides a way for websites to discover nearby devices, establish connections with them, and exchange data.

DEMO
SEQUENCER WITH WEB BLUETOOTH API

BONUS DEMO
ANALYZING AND VISUALIZING

DANKE SCHÖN!

https://rowdy.codes/enterjs