JS

Beats in the Browser

@rowdyrabouw

梯 郁太郎

Ikutarō Kakehashi


Roland

Beats in the Browser

@rowdyrabouw

Beats in the Browser

@rowdyrabouw

Beats in the Browser

@rowdyrabouw


Roland
TR-909
RHYTHM COMPOSER

Roland introduces the Latest Rhythm Machine

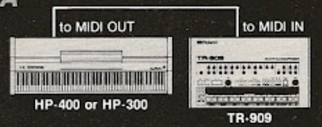
TR-909 RHYTHM COMPOSER

- 96 rhythm patterns can be memorized—48 patterns in each of two banks.
- There are two banks, each having four tracks for storing programmed rhythm chains of up to 896 measures.
- Eleven kinds of sound sources—Bass Drum, Snare Drum, Low Tom, Mid Tom, High Tom, Rim Shot, Hand Clap, Closed and Open Hi-hat, and Crash and Ride Cymbals. Several controls such as Tune or Level are provided for each sound sources. You can accurately tailor their sound. A Shuffle function and Flam effect are also provided to create full variety of drumming.
- There are two loading modes. In the real-time mode, you can create rhythm patterns by tapping the sound source buttons to the rhythm. In the single-step mode, you can program a rhythm pattern by writing each sound source in the each step. The same as with the TR-808 and TR-606 rhythm machines.
- A large LED display indicates the measure number and tempo in numerical number.
- A tape interface is provided for data storage on cassette tapes.
- In addition to the Master Output, each sound source can be outputted separately using Multi-Output jacks.
- Three MIDI jacks are provided—two for output and one for input. They allow the TR-909 to be connected with other MIDI devices to synchronize or to use external sound sources. Or, the TR-909 can be controlled by the device connected to it. This allows you to create complex, sophisticated drumming.
- The TR-909 has a 5-pin DIN Sync jack for synchronized performance. It can be started or stopped by the optional DP-2 foot switch. It also generates the Rim Shot trigger to control an external device.
- The use of an M-64C RAM cartridge (available from Roland) doubles the memory capacity of the TR-909 for the rhythm patterns and rhythm chains.

REAR PANEL

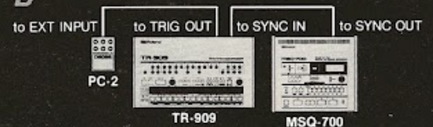
Typical TR-909 uses

A



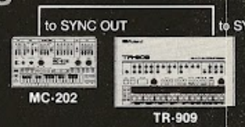
The sound sources of the TR-909 can be played using the keys of the PIANO PLUS 400. The volume changes depending on how forcefully the key is touched.

B



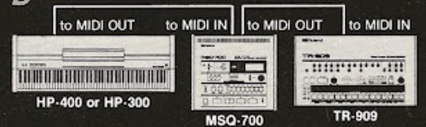
The data of the MSQ-700 synchronizes with the TR-909. It is also possible to control external sound sources, such as the PC-2 percussion synthesizer, using the TR-909's trigger signal.

C



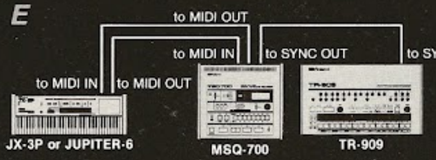
The MC-202 data controls the TR-909.

D



The PIANO PLUS 400 MIDI information controls both the MSQ-700 and TR-909. Turn the Mix Out switch of the MSQ-700 on to control the volume of the TR-909.

E



The MSQ-700 controls both the JX-3P and TR-909.

DIFFERENT DRUMMER

We don't call the TR-909 a drum machine for some very good reasons. True, it's a machine that makes drum sounds, but that's the end of any similarities between run-of-the-mill drum machines and the TR-909 Rhythm Composer. In fact, playing with the TR-909 is more like playing with a real drummer than anything else. Here's why. The Best Sounds. We start with digital recordings of real drums, then through a 3-D waveform analysis, re-create the sounds through a hybird digital/analog process. Not only does this provide the best drum sounds, but also the most flexible. Change the snap of the snare, the decay of the bass, you call it. The sounds you get are the sounds you really want. Even better—in addition to the 11 internal drum sounds, add up to 16 more drum sounds (digital and analog) through external sound modules. That means 27 drum sounds with no major surgery. The Best Programming. Program a roll on most drum machines and you'll see why they're called machines. That's why the TR-909 gives you the choice of Step Programming (highly visual and accurate) PLUS the additional spontaneity of Real-time Programming. The TR-909 also gives the most expressive and easily programmed dynamics. The Most Flexibility. Think of any way to interface, and you'll find it on the TR-909. MIDI, Sync-24, Tape Memory Save/Load, RAM-Pak Program storage, they're all here. So what does this mean? It means that years from now, when other drum machines are sitting in the closet gathering dust, your TR-909 will still be on the job. Hook up the TR-909 through MIDI to a personal computer (like the Apple II or IBM PC). Only Roland has the Hardware and the Software to make it possible. The Most Expression. Compare the results you get from the TR-909 Rhythm Composer with any drum machine. Because why would you want a machine, when you can have a Rhythm Composer?
RolandCorp US, 7200 Dominion Circle, LA, CA 90040.

Roland

Roland Makes It Happen!

Different Drummer

We don't call the TR-909 a drum machine for some very good reasons. True, it's a machine that makes drum sounds, but that's the end of any similarities between run-of-the-mill drum machines and the TR-909 Rhythm Composer. In fact, playing with the TR-909 is more like playing with a real drummer than anything else. Here's why. The Best Sounds. We start with digital recordings of real drums, then through a 3-D waveform analysis, re-create the sounds through a hybird digital/analog process. Not only does this provide the best drum sounds, but also the most flexible. Change the snap of the snare, the decay of the bass, you call it. The sounds you get are the sounds you really want. Even better-in addition to the 11 internal drum sounds, add up to 16 more drum sounds (digital and analog) through external sound modules. That means 27 drum sounds with no major surgery. The Best Programming.

Program a roll on most drum machines and you'll see why they're called machines. That's why the TR-909 gives you the choice of Step Programming (highly visual and accurate) PLUS the additional spontaneity of Real-time Programming. The TR-909 also gives the most expressive and easily programmed dynamics. The Most Flexibility. Think of any way to interface, and you'll find it on the TR-909. MIDI, Sync-24, Tape Memory Save/Load, RAM-Pak Program storage, they're all here. So what does this mean? It means that years from now, when other drum machines are sitting in the closet gathering dust, your TR-909 will still be on the job. Hook up the TR-909 through MIDI to a personal computer (like the Apple II or IBM PC). Only Roland has the Hardware and the Software to make it possible. The Most Expression. Compare the results you get from the TR-909 Rhythm Composer with any drum machine. Because why would you want a machine, when you can have a Rhythm Composer?

Roland Makes It Happen!

Program a roll on most drum machines and you'll see why they're called machines. That's why the TR-909 gives you the choice of Step Programming (highly visual and accurate) PLUS the additional spontaneity of Real-time Programming. The TR-909 also gives the most expressive and easily programmed dynamics. The Most Flexibility. Think of any way to interface, and you'll find it on the TR-909. MIDI, Sync-24, Tape Memory Save/Load, RAM-Pak Program storage, they're all here. So what does this mean? It means that years from now, when other drum machines are sitting in the closet gathering dust, your TR-909 will still be on the job. Hook up the TR-909 through MIDI to a personal computer (like the Apple II or IBM PC). Only Roland has the Hardware and the Software to make it possible. The Most Expression. Compare the results you get from the TR-909 Rhythm Composer with any drum machine. Because why would you want a machine, when you can have a Rhythm Composer?

Roland Makes It Happen!

Beats in the Browser

@rowdyrabouw

Beats in the Browser

@rowdyrabouw


BRITISH

Beats in the Browser

@rowdyrabouw

ROWDY RABOUW

Club DJ in the late 80's and early 90's

Build my first website in 1996

Front-End Focused Senior DevOps Engineer

Google Developer Expert in Web Technologies

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 OscillatorNode generates periodic waveforms for creating sounds in real-time.

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


});
```

FREQUENCY

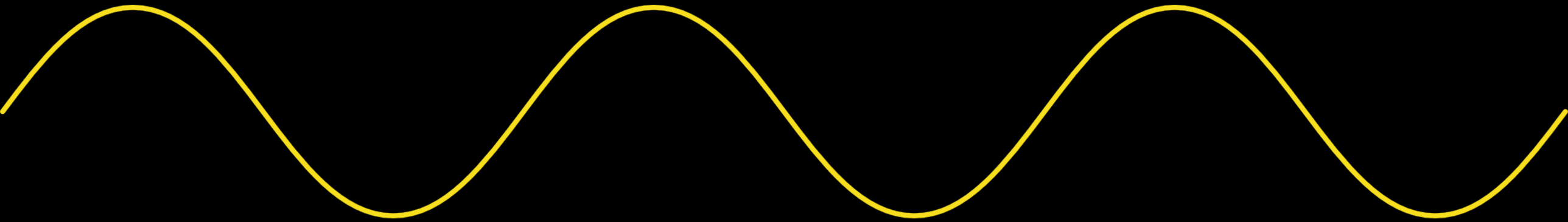
The frequency property of the OscillatorNode determines the pitch of the generated sound. It represents the number of cycles per second, measured in Hertz (Hz).

```
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.

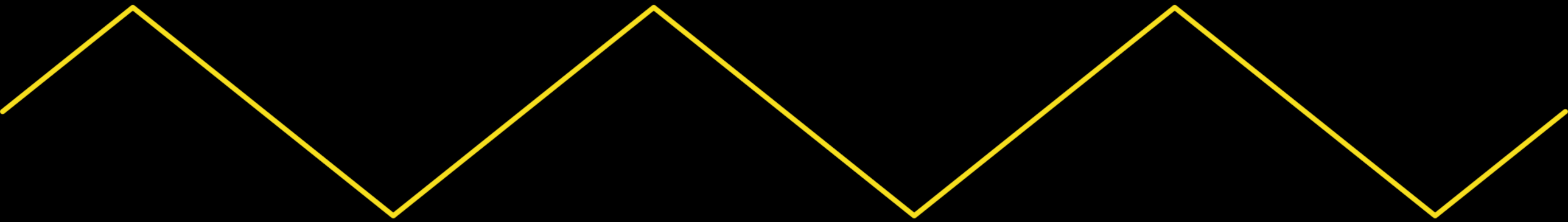
SINE



Beats in the Browser

@rowdyrabouw

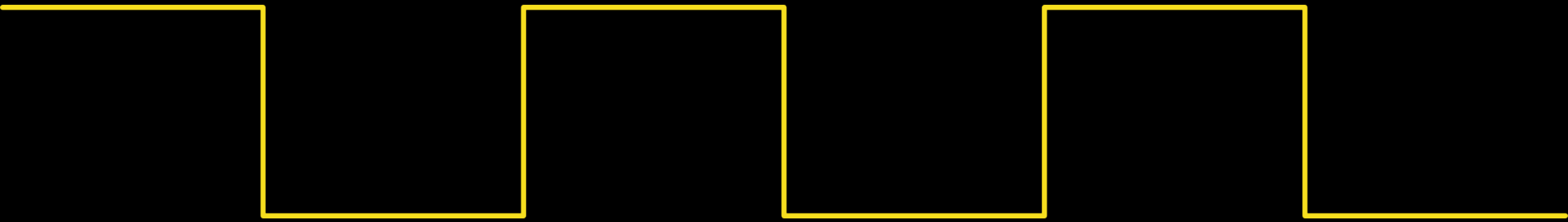
TRIANGLE



Beats in the Browser

@rowdyrabouw

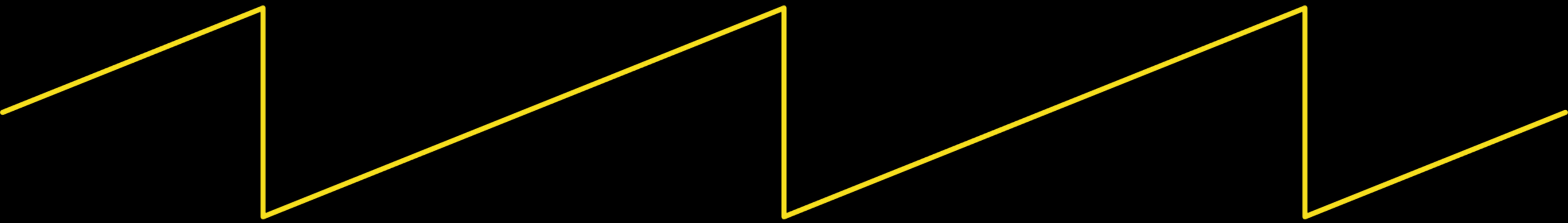
SQUARE



Beats in the Browser

@rowdyrabouw

SAWTOOTH



Beats in the Browser

@rowdyrabouw

```
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 + 2);
```

Beats in the Browser

@rowdyrabouw


Roland
TR-909
RHYTHM COMPOSER

Beats in the Browser

@rowdyrabouw

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="40" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

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

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

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




});
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

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


  });
});
```

```
<button class="pad" data-note="16" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></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>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>

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

```
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) ⟹ {



};
```

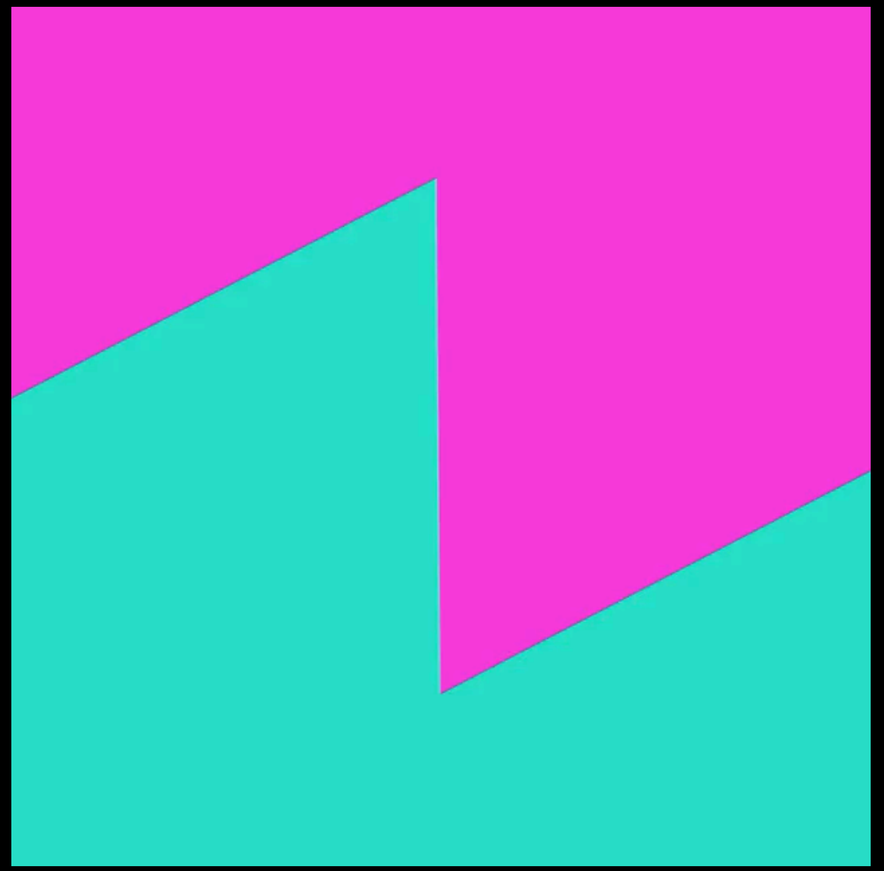
```
const loadFile = async (wav) ⟹ {
  const response = await fetch(wav);
  const arrayBuffer = await response.arrayBuffer();
  const audioBuffer = await ctx.decodeAudioData(arrayBuffer);
  return audioBuffer;
};


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();
};
```

Beats in the Browser

@rowdyrabouw



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('909-kick.wav').toDestination();
```

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

kick.start();
```

```
<button class="pad" data-note="40" data-wav="909-kick"></button>
<button class="pad" data-note="41" data-wav="909-clap"></button>
<button class="pad" data-note="42" data-wav="909-closed-hat"></button>
<button class="pad" data-note="43" data-wav="909-open-hat"></button>
```

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

```
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();
};
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>

const kick = new Tone.Player('909-kick.wav').toDestination();
const clap = new Tone.Player('909-clap.wav').toDestination();
const closed = new Tone.Player('909-closed-hat.wav').toDestination();
const open = new Tone.Player('909-open-hat.wav').toDestination();
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>


const kick = new Tone.Player('909-kick.wav').toDestination();
const clap = new Tone.Player('909-clap.wav').toDestination();
const closed = new Tone.Player('909-closed-hat.wav').toDestination();
const open = new Tone.Player('909-open-hat.wav').toDestination();

const pads = document.querySelectorAll(".pad");
pads.forEach((pad) => {
  pad.addEventListener("click", async () => {
    switch (pad.dataset.note) {
      case "40": kick.start();   break;
      case "41": clap.start();   break;
      case "42": closed.start(); break;
      case "43": open.start();   break;
    }
 });
});
```

@rowdyrabouw

Beats in the Browser

@rowdyrabouw

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

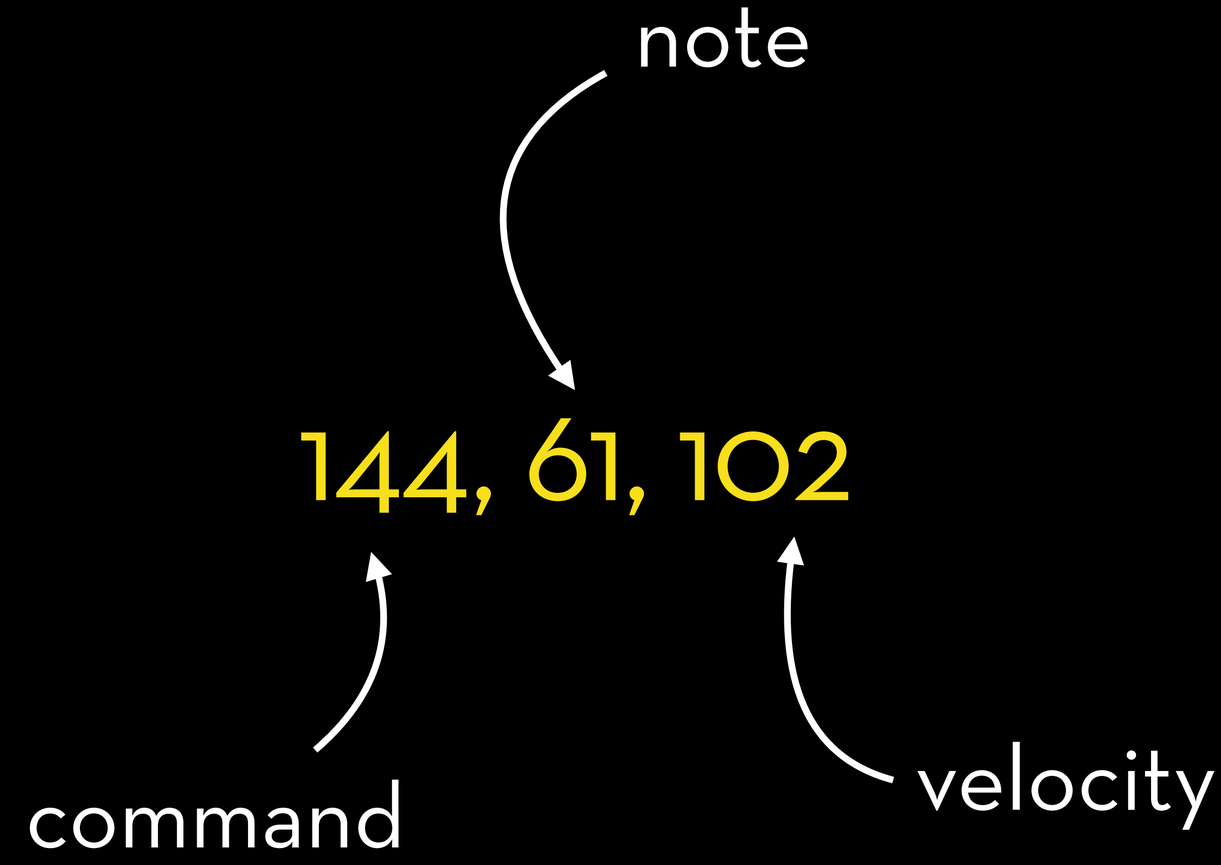
@rowdyrabouw

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

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

const succes = (midiAccess) => {




}
```

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

const succes = (midiAccess) => {

  const inputs = midiAccess.inputs;




}
```

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

const succes = (midiAccess) => {

  const inputs = midiAccess.inputs;

  inputs.forEach((input) => {
   input.addEventListener("midimessage", handleInput);
  });

}
```


note
144, 61, 102
command
velocity

Beats in the Browser

@rowdyrabouw

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>

const handleInput = (input) => {




};
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>

const handleInput = (input) => {

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




};
```

```
<button class="pad" data-note="40"></button>
<button class="pad" data-note="41"></button>
<button class="pad" data-note="42"></button>
<button class="pad" data-note="43"></button>

const handleInput = (input) => {

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

  if(velocity > 0) {
    const pad = document.querySelector(`[data-note="${note}"]`);
    pad.click();
  }

};
```

Beats in the Browser

@rowdyrabouw


Roland
TR-909
RHYTHM COMPOSER

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.


FIRE
GRID
BROWSER
SELECT
PATTERN
CHANNEL
MIXER
USER 1
USER 2
MUTE
SOLO
STEP
ACCENT
NOTE
SNAP
DRUM
TAP
PERFORM
OVERVIEW
SHIFT
ALT
PATTERN/
SONG
METRONOME
WAIT
COUNTDOWN
LOOP REC
AKAI
PROFESSIONAL
FL STUDIO

SEQUENCER

pan low

pan high

pitch

bpm

120 bpm

Init

Midi

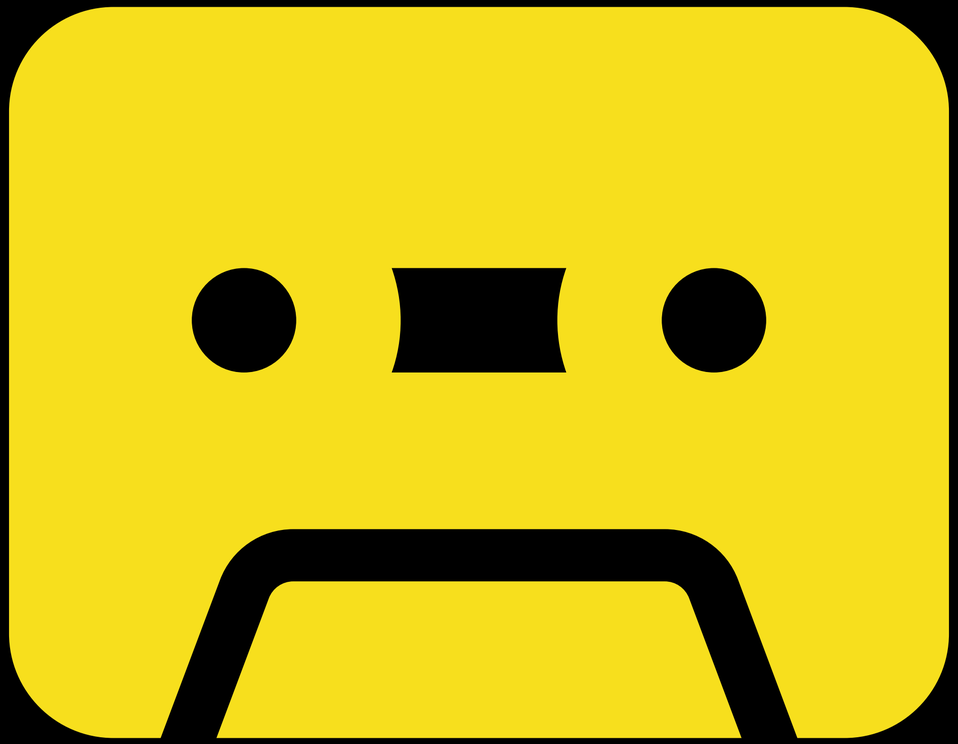
hbpm

```
<div class="kick">
  <label><input type=“checkbox" data-mute="kick" /></label>
  <label><input type="checkbox" data-note="54" /></label>
  <label><input type="checkbox" data-note="55" /></label>
  <label><input type="checkbox" data-note="56" /></label>
  <label><input type="checkbox" data-note="57" /></label>
  <label><input type="checkbox" data-note="58" /></label>
  <label><input type="checkbox" data-note="59" /></label>
  <label><input type="checkbox" data-note="60" /></label>
  <label><input type="checkbox" data-note="61" /></label>
  <label><input type="checkbox" data-note="62" /></label>
  <label><input type="checkbox" data-note="63" /></label>
  <label><input type="checkbox" data-note="64" /></label>
  <label><input type="checkbox" data-note="65" /></label>
  <label><input type="checkbox" data-note="66" /></label>
  <label><input type="checkbox" data-note="67" /></label>
  <label><input type="checkbox" data-note="68" /></label>
  <label><input type="checkbox" data-note="69" /></label>
</div>
```

```
const handleInput = (input) => {

  const note = input[1];
  const velocity = input[2];

  if (velocity > 0) {
    if (note >= 54 && note <= 117) {
      document.querySelector(`[data-note="${note}"]`).click();
    }
  }

}
```

```
const initSequencer = () => {

  Tone.Transport.scheduleRepeat((time) => {
    repeat(time);
  }, 16);

}
```

```
let index = 0;
const steps = 16;

const repeat = (time) => {

  // simplified, only showing kicks

  let step = index % 16;

  const muteKicks = document.querySelector('.kick [data-mute]');
  const kicks =
    document.querySelector(`.kick label:nth-child(${step + 2}) input`);

  if (!muteKicks.checked && kicks.checked) {
    kick.start(time);
  }

  index++;
}
```



```
const recorder = new Tone.Recorder();
```

```
const recorder = new Tone.Recorder();

const record = async () => {
  await recorder.start();
}
```

```
const recorder = new Tone.Recorder();

const record = async () => {
  await recorder.start();
}

const stop = async () => {
  const recording = await recorder.stop();




}
```

```
const recorder = new Tone.Recorder();

const record = async () => {
  await recorder.start();
}

const stop = async () => {
  const recording = await recorder.stop();

  const url = URL.createObjectURL(recording);
  const audio = document.createElement('audio');
  audio.controls = true;
  audio.src = url;


}
```

```
const recorder = new Tone.Recorder();

const record = async () => {
  await recorder.start();
}

const stop = async () => {
  const recording = await recorder.stop();

  const url = URL.createObjectURL(recording);
  const audio = document.createElement('audio');
  audio.controls = true;
  audio.src = url;

  document.querySelector('#recordings').append(audio);
}
```


FIRE
GRID
BROWSER
SELECT
PATTERN
CHANNEL
MIXER
USER 1
USER 2
MUTE
SOLO
STEP
ACCENT
NOTE
SNAP
DRUM
TAP
PERFORM
OVERVIEW
SHIFT
ALT
PATTERN/
SONG
METRONOME
WAIT
COUNTDOWN
LOOP REC
AKAI
PROFESSIONAL
FL STUDIO


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

Beats in the Browser

@rowdyrabouw

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.

```
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});
```

```
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});

// ‘array of objects’
const server = await device.gatt.connect();
```

```
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});

// ‘array of objects’
const server = await device.gatt.connect();

// ‘object’
const service = await server.getPrimaryService(0xffe5);
```

```
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});

// ‘array of objects’
const server = await device.gatt.connect();

// ‘object’
const service = await server.getPrimaryService(0xffe5);

// ‘property’
const characteristic = await service.getCharacteristic(0xffe9);
```

```
const device = await navigator.bluetooth.requestDevice({
  filters: [
    { namePrefix: 'Bluetooth-Device' } // which device?
  ],
  optionalServices: [0xffe5] // what service(s)? → uuid
});

// ‘array of objects’
const server = await device.gatt.connect();

// ‘object’
const service = await server.getPrimaryService(0xffe5);

// ‘property’
const characteristic = await service.getCharacteristic(0xffe9);

await characteristic.startNotifications();
```


POLAR

```
characteristic.addEventListener("characteristicvaluechanged", (event) => {



});
```

```
characteristic.addEventListener("characteristicvaluechanged", (event) => {
  const hbpm = event.target.value.getUint8(1);


});
```

```
characteristic.addEventListener("characteristicvaluechanged", (event) => {
  const hbpm = event.target.value.getUint8(1);
  Tone.Transport.bpm.value = hbpm;
});
```

Beats in the Browser

@rowdyrabouw


Roland
TR-09
RHYTHM COMPOSER
TRACK PLAY
TRACK WRITE
PATTERN PLAY
PATTERN WRITE
EDIT
COMP
STEP
TEMPO
TAP
BACK
FWD
BANK
PATTERN WRITE MODE
AVAILABLE MEAS
CYCLE GUIDE
MENU
LAST MEAS
TRIGGER OUT
ENTER
TOTAL ACCENT
BASS DRUM
SNARE DRUM
LOW TOM
MID TOM
HI TOM
RIM
CLAP
CLOSED
OPEN
CLOSED
CRASH
RIDE
COPY
INS
DEL

TACK SÅ MYCKET !

rowdy.codes/swetugg