1.

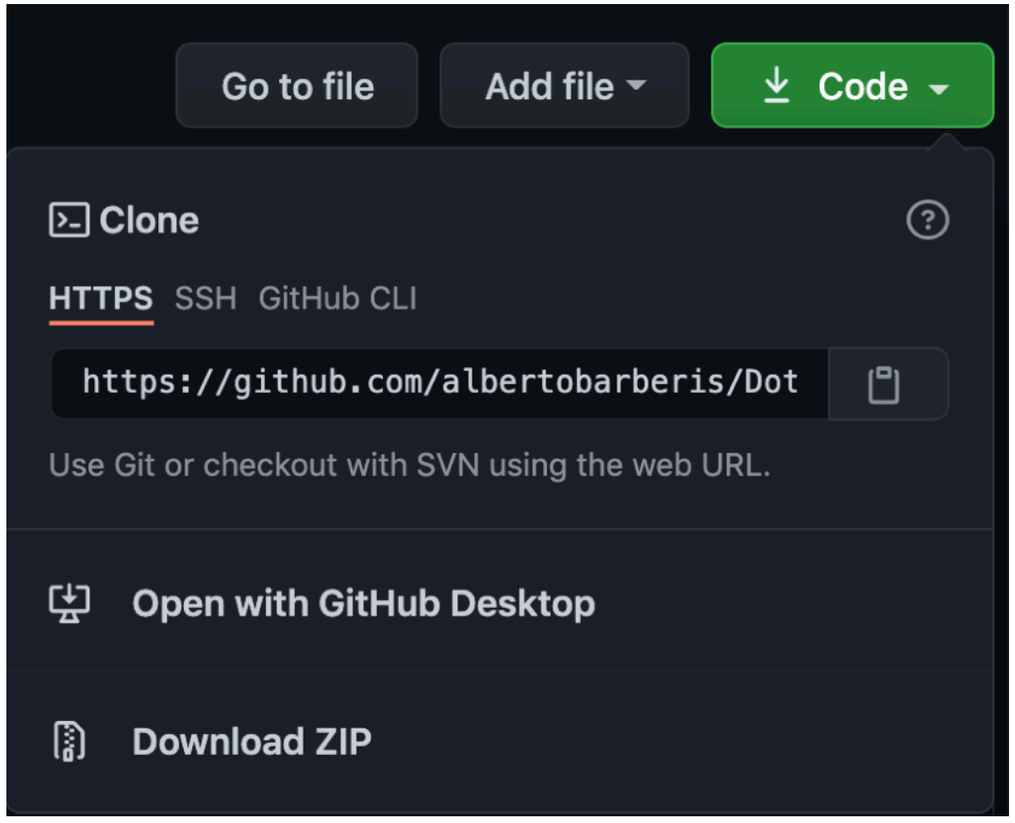
AN INTRODUCTION TO THE THE WEB AUDIO API

Alberto Barberis

MATERIALS

You find all the **materials** (pdf slides, and code) for the workshop at the following link of a **Github repository**.

https://github.com/albertobarberis/WebAudio_Valencia



Click on the green **Code** button and then on **Download ZIP** to download all the materials from the repository.

GOALS

1. understand what is the **web audio** and what are the basic concepts of the **web audio Application Programming Interface** (the **web audio API**);

2. analyse together **some examples** of **web audio applications**;

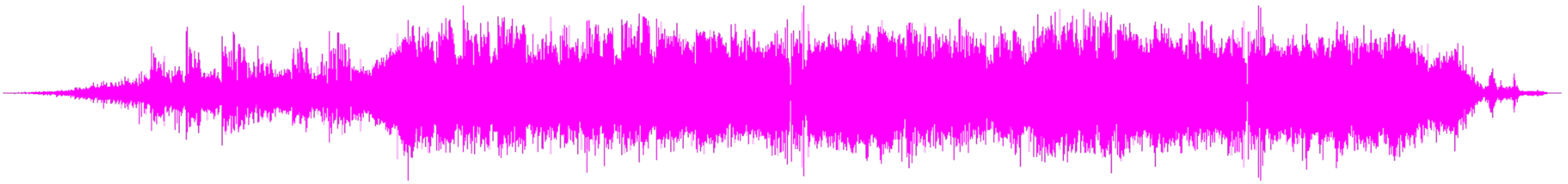
3. summarise what are the **possibilities** offered by the web audio for sound design, music composition and creativity;

4. understand **what you need to know** to deal with the web audio (HTML, css, javascript; music theory and sound design; etc.);

5. **develop** a simple **web audio app**: the ***Dot Drone Generator***.

1. WHAT IS THE WEB AUDIO

1.1 WHAT IS WEB AUDIO



With the expression **web audio** we refer to the technologies that allow to **process/synthesise audio signals** and **compose music** in modern **web applications** (running on a browser like Google Chrome, Firefox, etc).

The web audio allows to **create**, **compose** and **manipulate music in real-time** in **different devices** (smartphone, computer, tablet, smart TV, etc.) which use a browser that supports the **web audio API**.

1.1 WHAT IS WEB AUDIO

The **web audio API** is a rather new technology (2011), made available by the **World Wide Web Consortium (W3C)**, the main **international standards organization** for the World Wide Web.

https://www.w3.org/

Before the **web audio API**, audio could be embedded in browsers, but more complex audio effects were only available through the use of plugins.

Dealing with **web audio applications** design, **web audio music**, and **web audio composition** means primarily dealing with **WEB PROGRAMMING LANGUAGES**.

1.2 PROGRAMMING LANGUAGES

HTML

```html
<!DOCTYPE html>

<html lang="en">

  <head>
    <title>web audio HS2019</title>
    <meta charset="utf-8">
  </head>

  <body>
    <div>
      this is my first web app
    </div>
  </body>

</html>
```

CSS

```css
body {
    margin: 0px;
    font-family: 'Courier New', Courier, monospace;
}
h1 {
    margin: 0px;
    padding: 10px;
    background-color: rgb(88, 88, 255);
    color: white;
    -webkit-user-select: none;

}
```

JAVA

```java
import com.cycling74.max.*;
public class SimpleBiquad extends MSPPerformer
{
    private float freq = 1000.0f;
    private float  c, a0, a1, a2, b1, b2, tin1, tin2,tout1, tout2;
    private float pi = 3.1415926f;
    private float r =  1.4142135f;
    private double sr; // sample rate
    private static final String[] INLET_ASSIST = new String[]{
        "input (sig)"
    };
    private static final String[] OUTLET_ASSIST = new String[]{
        "output (sig)"
    };
    public SimpleBiquad()
    {
        declareInlets(new int[]{SIGNAL, DataTypes.ALL});
        declareOutlets(new int[]{SIGNAL});
        setInletAssist(0, "signal input");
        setInletAssist(1, "Cut-Off Frequency");
        setOutletAssist(OUTLET_ASSIST);
        createInfoOutlet(false);    // suppress info outlet
    }
    public void inlet(float f)
    {

            freq = f;                // this is the cutOff frequency

    }
    public void dspsetup(MSPSignal[] ins, MSPSignal[] outs)
    {

        sr = ins[0].sr; //this is the sample rate

    }
```

JAVASCRIPT

```javascript
function mouseRealeased() {
    initializeIndex();
}
function keyReleased(){
    initializeIndex();
}

function frequer

    var scalar1=
    i.osc.freq(j

    var scalar2=(random(300)+50)*random([-1,1]);
    j.osc.freq(i.freqModulator.mult(scalar2));
}
```

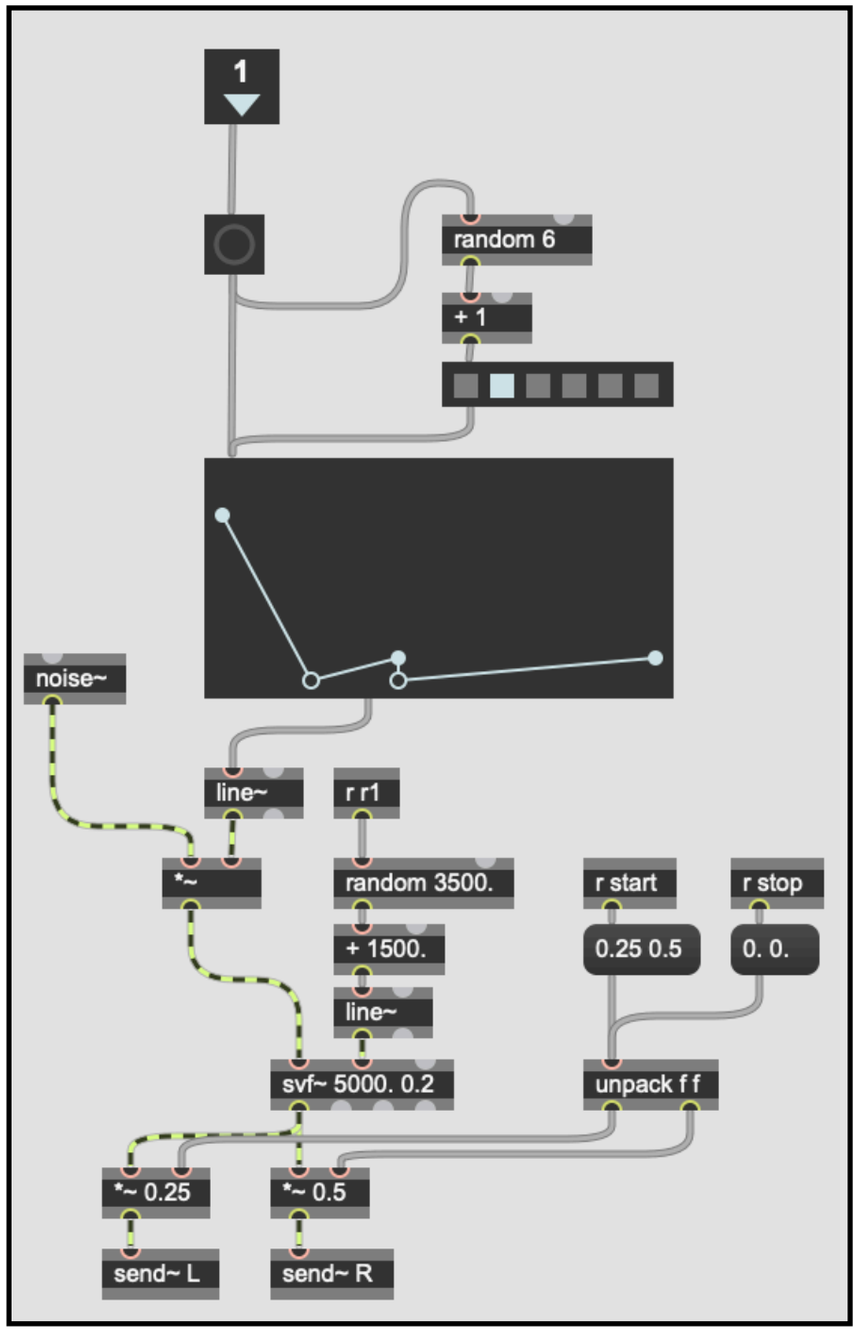
PHP

```php
<?php
$x = 5; // global scope

function myTest() {
  // using x inside this function will generate an error
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

C/C++

```c
/**
    @file
    simplemax
    jeremy ber

    @ingroup
*/

#include "ex
#include "ext_obex.h"                    // required for new s

/// ////////////////////// object struct
typedef struct _simplemax
{
    t_object ob;            // the object itself (must be first)
} t_simplemax;

/// ////////////////////// function prototypes
//// standard set
void *simplemax_new(t_symbol *s, long argc, t_atom *argv);
void simplemax_free(t_simplemax *x);
void simplemax_assist(t_simplemax *x, void *b, long m, char *s);

/// ////////////////////// global class pointer variable
void *simplemax_class;
```

1.2 PROGRAMMING LANGUAGES



MAX/MSP

ARDUINO

```
#include "pitches.h"

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};

void setup() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second divided by the
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(8, melody[thisNote], noteDuration);

    // to distinguish the
    // the note's duratio
    int pauseBetweenNotes
    delay(pauseBetweenNot
    // stop the tone play
    noTone(8);
  }
}

void loop() {
  // no need to repeat th
}
```

some examples of programming languages made for artists / designer

PROCESSING

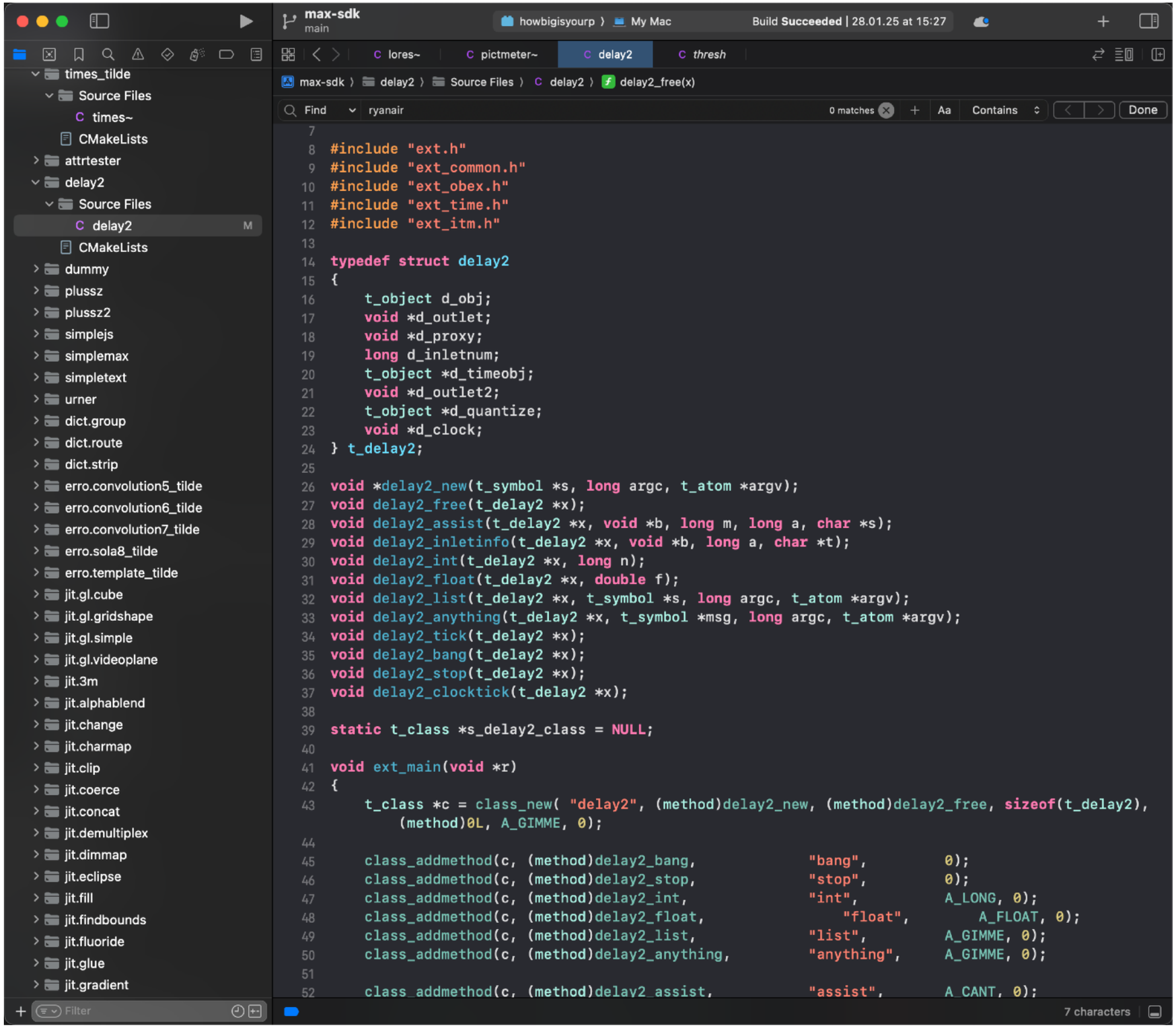
NOC_1_8_motion101_acceleration | Mover

```
class Mover {

  PVector location;
  PVector velocity;
  PVector acceleration;
  float topspeed;

  Mover() {
    location = new PVector(width/2, height/2);
    velocity = new PVector(0, 0);
    acceleration = new PVector(-0.001, 0.01);
    topspeed = 10;
  }
```

1.2 PROGRAMMING LANGUAGES

SDK for create low level Max/MSP Objects

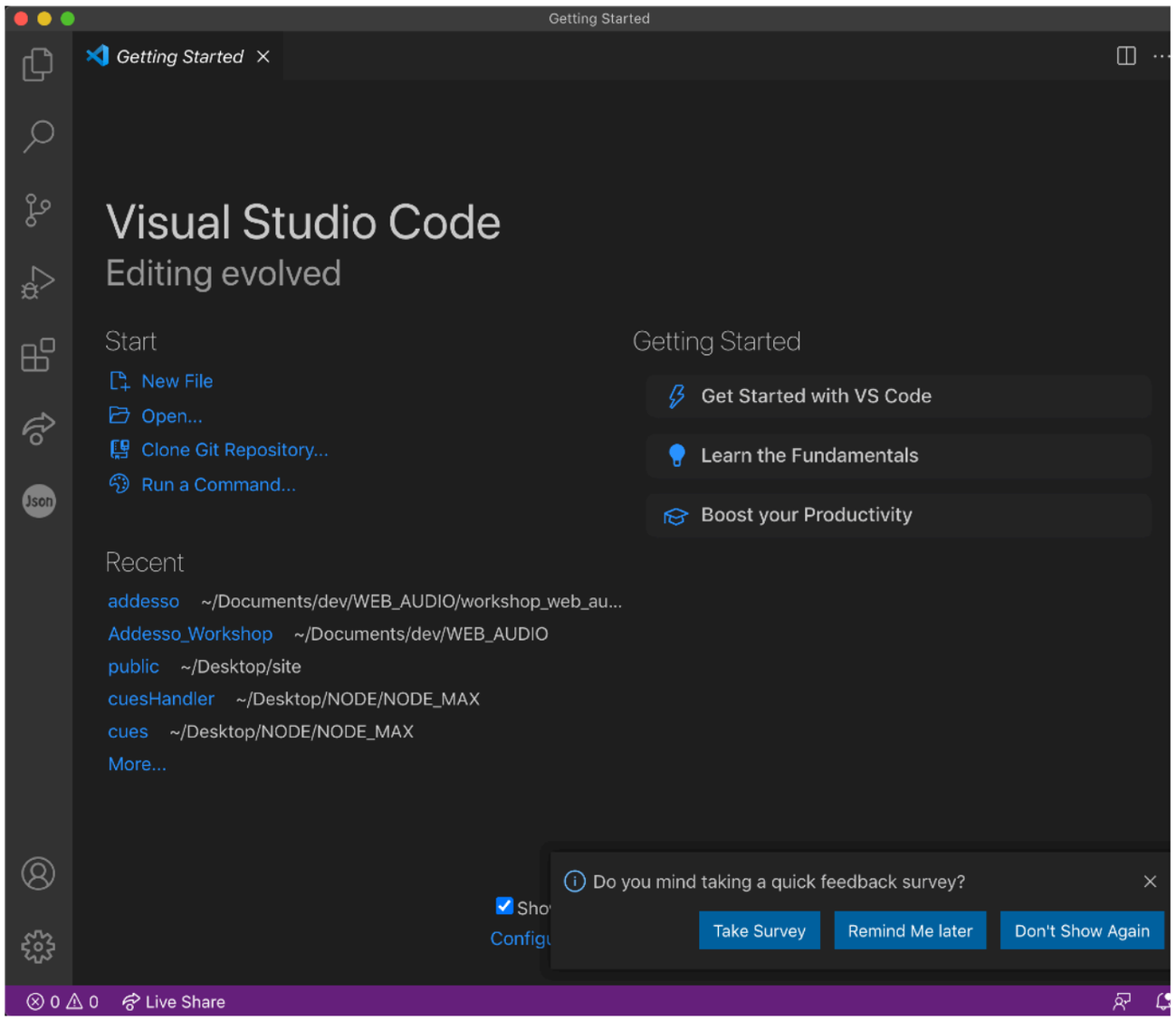
C/C++ Max/MSP SDK




```
#include "ext.h"
#include "ext_common.h"
#include "ext_obex.h"
#include "ext_time.h"
#include "ext_itm.h"

typedef struct delay2
{
    t_object d_obj;
    void *d_outlet;
    void *d_proxy;
    long d_inletnum;
    t_object *d_timeobj;
    void *d_outlet2;
    t_object *d_quantize;
    void *d_clock;
} t_delay2;

void *delay2_new(t_symbol *s, long argc, t_atom *argv);
void delay2_free(t_delay2 *x);
void delay2_assist(t_delay2 *x, void *b, long m, long a, char *s);
void delay2_inletinfo(t_delay2 *x, void *b, long a, char *t);
void delay2_int(t_delay2 *x, long n);
void delay2_float(t_delay2 *x, double f);
void delay2_list(t_delay2 *x, t_symbol *s, long argc, t_atom *argv);
void delay2_anything(t_delay2 *x, t_symbol *msg, long argc, t_atom *argv);
void delay2_tick(t_delay2 *x);
void delay2_bang(t_delay2 *x);
void delay2_stop(t_delay2 *x);
void delay2_clocktick(t_delay2 *x);

static t_class *s_delay2_class = NULL;

void ext_main(void *r)
{
    t_class *c = class_new( "delay2", (method)delay2_new, (method)delay2_free, sizeof(t_delay2),
        (method)0L, A_GIMME, 0);

    class_addmethod(c, (method)delay2_bang,          "bang",       0);
    class_addmethod(c, (method)delay2_stop,          "stop",       0);
    class_addmethod(c, (method)delay2_int,           "int",        A_LONG, 0);
    class_addmethod(c, (method)delay2_float,             "float",      A_FLOAT, 0);
    class_addmethod(c, (method)delay2_list,          "list",       A_GIMME, 0);
    class_addmethod(c, (method)delay2_anything,      "anything",   A_GIMME, 0);

    class_addmethod(c, (method)delay2_assist,        "assist",     A_CANT, 0);
```

1.3 TEXT EDITOR

https://visualstudio.microsoft.com/



If you don't have a text editor you can download and install **Visual Studio Code** (it's free).

For the workshop is better to use **Google Chrome** (better for web audio API compatibility).

1.4 WEB PROGRAMMING

The term **web programming** refers to all programming activities, languages, practices, and technologies that enable the creation of **web applications**.

A **web application** is a **dynamic website** running on a browser where is required a **high user interaction** (opposed to static web pages).

1.4 WEB PROGRAMMING

A **web browser** is a software for accessing the **World Wide Web**, an information system where the resources are identified by an **URL** - Uniform Resource Locator, accessible over the internet.

Internet is the global system of interconnected computer networks that uses the protocol TCP/IP to communicate. It is a physical infrastructure that allows devices to connect and exchange information.

The **Internet** is the underlying network that enables communication between devices, while the **WWW** is a layer on top of the Internet that provides a user-friendly way to access and share information using web pages, links, and browsers.

1.4 WEB PROGRAMMING

Web development is divided into several areas:

- **Web design**;
- **GUI** (graphic user interface) **development**;
- **Client-side** programming and scripting;
- **Server-side** development;
- **Database** management;
- **Network security**;
- …

1.4 WEB PROGRAMMING

Coding in **Web development** includes different languages:

- **markup languages** (HTML, CSS): define the structure, the organization, and the look and feel of a site.

- **client-side scripting languages** (javascript): transform the website from a static page to an interactive application; client-side code runs in the browser.

- **server-side scripting languages** (PHP / node.js): server-side code lives on a server, and serves as go-between architecture, transferring data to the browser, etc..

1.4 WEB PROGRAMMING

a programmer who deals with all aspects of the web development is called a

full stack developer

Do we need all of this for **making music on the browser** ?

actually no …

1.4 WEB PROGRAMMING

We need:

- some lines of **client-side code** (html + css + javascript);
- a **web browser** (like Google Chrome).

We don't need to manage a database, to organise a large amount of data, or to store user data, to implement security protocols, etc.. (but of course we could ...).

Working locally on our machine, **we won't even need a server** where we can save (and request) our client-side code because the browser can load, directly from our file system, the *entry point* of the client-side code: **an HTML file with some CSS and Javascript**.

1.5 API

In the context of computer programming, an **Application Programming Interface** (API) represents a **set of procedures** for carrying out given tasks.

An API lists a **bunch of operations** that developers can use, along with a description of what they do (the **API documentation**).

The developer doesn't necessarily need to know how they work internally, they just **need to know how to use them**.

The term can also indicate a **software library**.

1.6 WEB AUDIO API

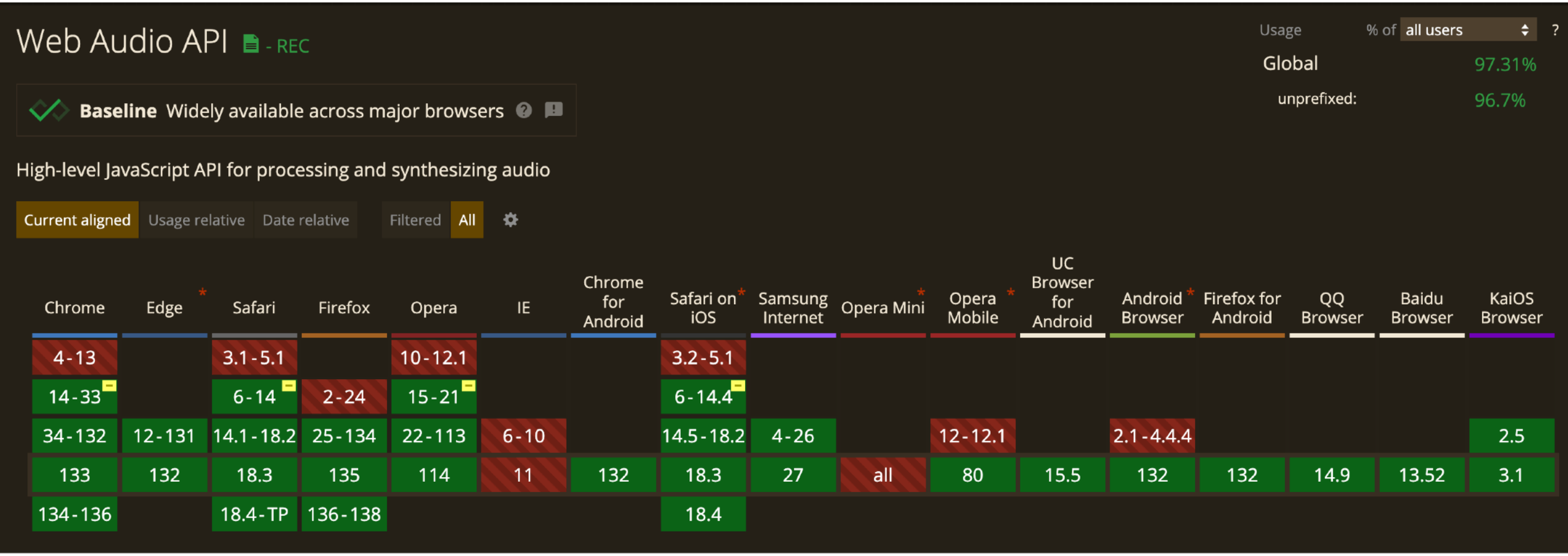
The **web audio API** is a **high level javascript API for processing and synthesizing audio in web applications**. It provides a powerful system for:

- controlling audio on the Web;
- create audio sources;
- add effects to audio;
- create audio visualization;
- compose with algorithmic procedures;
- apply spatial effects;
- …

OFFICIAL DOCUMENTATION:
https://www.w3.org/TR/webaudio/

MOZILLA DOCUMENTATION:
https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

1.6 WEB AUDIO API



https://caniuse.com/?search=web%20audio%20api

The **web audio API** is supported in most of the modern browsers (desktop or mobile).

1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

Why should we be interested in **creating music on the web as composers** ??

1. play **audio tracks** in the browser with a control over different parameters, and **processing** them (DSP);
2. performing **sound analysis** (time and frequency domain);
3. develop **sound visualisation** applications (waveform and spectrum); integrating web audio API with web graphics libraries (webGL, three.js);

1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

4. creating **web applications for music education** (music theory, ear-training, etc.);
5. creating **generative compositions** living in the browser (exploiting the algorithmic logic offered by the object oriented programming language javascript and the whole ecosystem of its libraries and extensions);
6. create **interactive compositions**, in which the users play an important role in the creative process;
7. develop **synths and web musical instruments**;

1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

8. real-time **data analysis and data sonification**, taking advantage of the many APIs to collect and receive data;
9. create **not-fixed media music**: compositions that can be accessed anywhere and from different devices;
10. use the web audio along with **web MIDI** to create integrated browser/desktop/MIDI interface environments;
11. exploit the web audio to create **web IDEs** that interpret other audio languages (e.g. Csound: https://ide.csound.com/);

1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

12. develop **web audio effects for real-time** DSP; with the AudioWorklet it is possible to create high-performance sample-rate DSP algorithms (such as *genish.js*);
13. create **environments for live-coding** and **online music performances** that take advantage of the web technologies and web support;
14. exploit the possibilities of **integration with the entire web ecosystem** (extensions and libraries) to do almost anything!
15. be a **critical digital thinker** ...

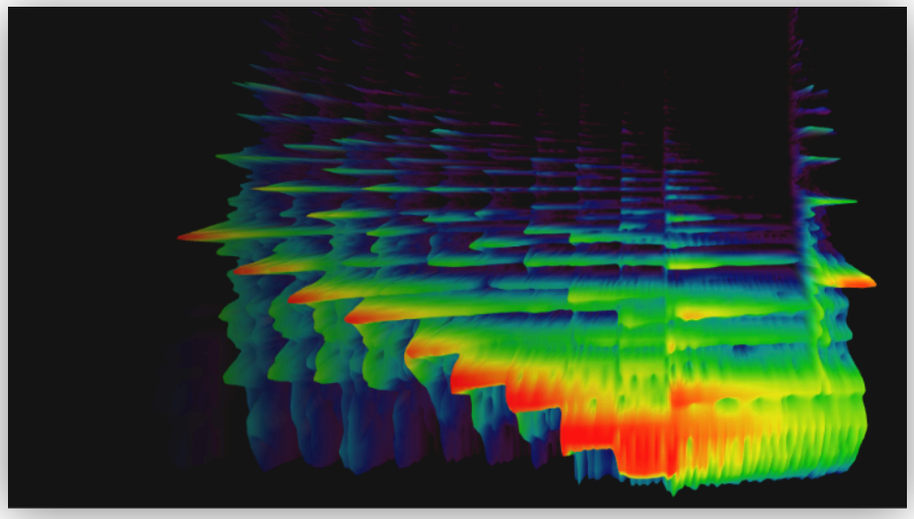
1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

example 1 | VOICE - SPINNER

https://musiclab.chromeexperiments.com/Voice-Spinner/

example 2 | AUDIO ANALYZER

https://musiclab.chromeexperiments.com/Spectrogram/



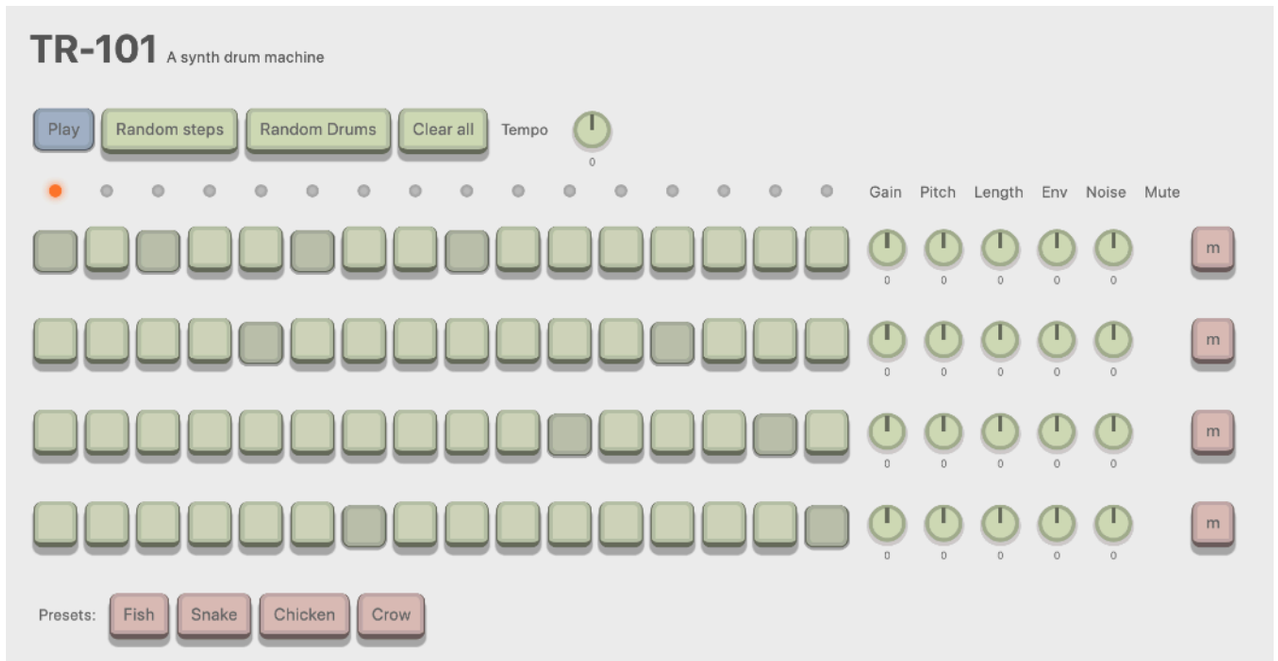
1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

example 3 | MUSIC EDUCATION

https://tamburo11.github.io/SOLO/

example 4 | DRUM MACHINE

https://tr101.vercel.app/



1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

example 5 | DATA SONIFICATION

https://sonicvirus.si.usi.ch/

example 6 | WEB AUDIO SYNTHESIZER

https://albertobarberis.github.io/addesso_synth/

1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

example 7 | WEB PEDAL BOARD

https://pedalboard.netlify.app/

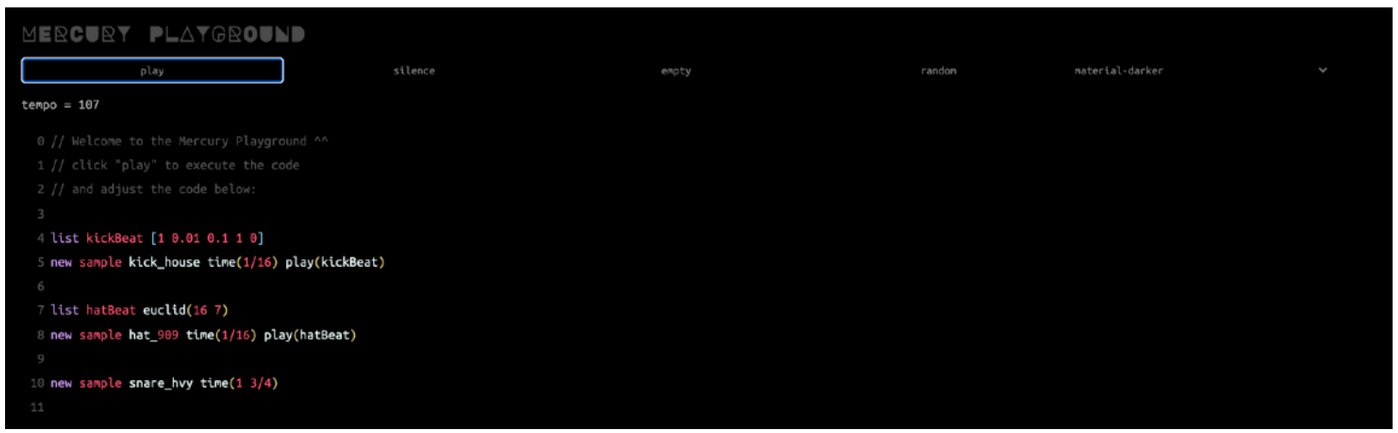
1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

example 8 | FILTER DESIGN

https://borismus.github.io/filter-playground/?equation=%28z%5E2%2B0.5475285649299622z-0.45247143507003784%29%2F%28z%5E2-0.6741842031478882z%2B0.3505924277806516%29

example 9 | LIVE CODING ENVIRONMENT

https://mercury-sketch.glitch.me/



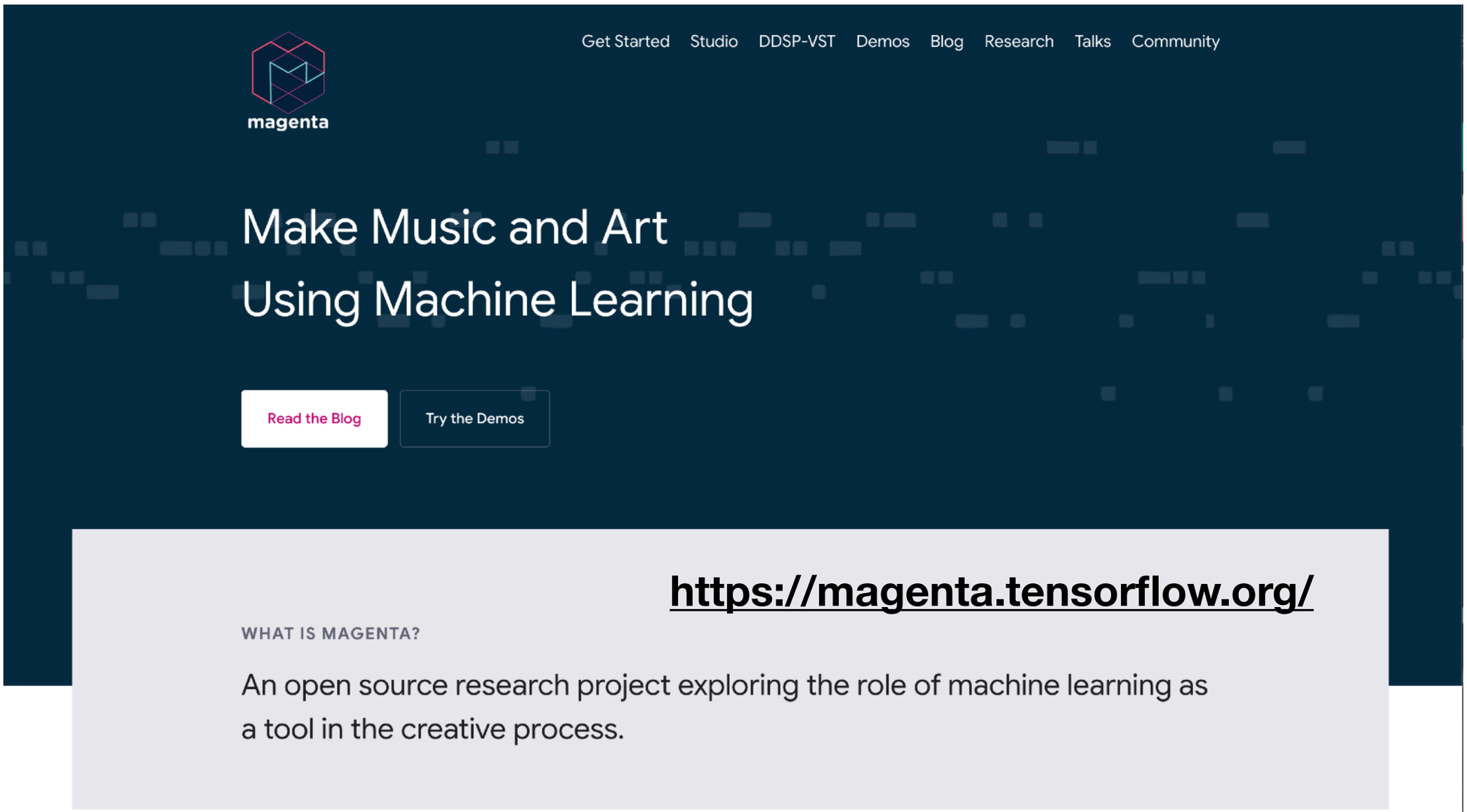
1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

example 10 | GENERATIVE MUSIC

https://tamburo11.github.io/thetricliniumdesireWeb/

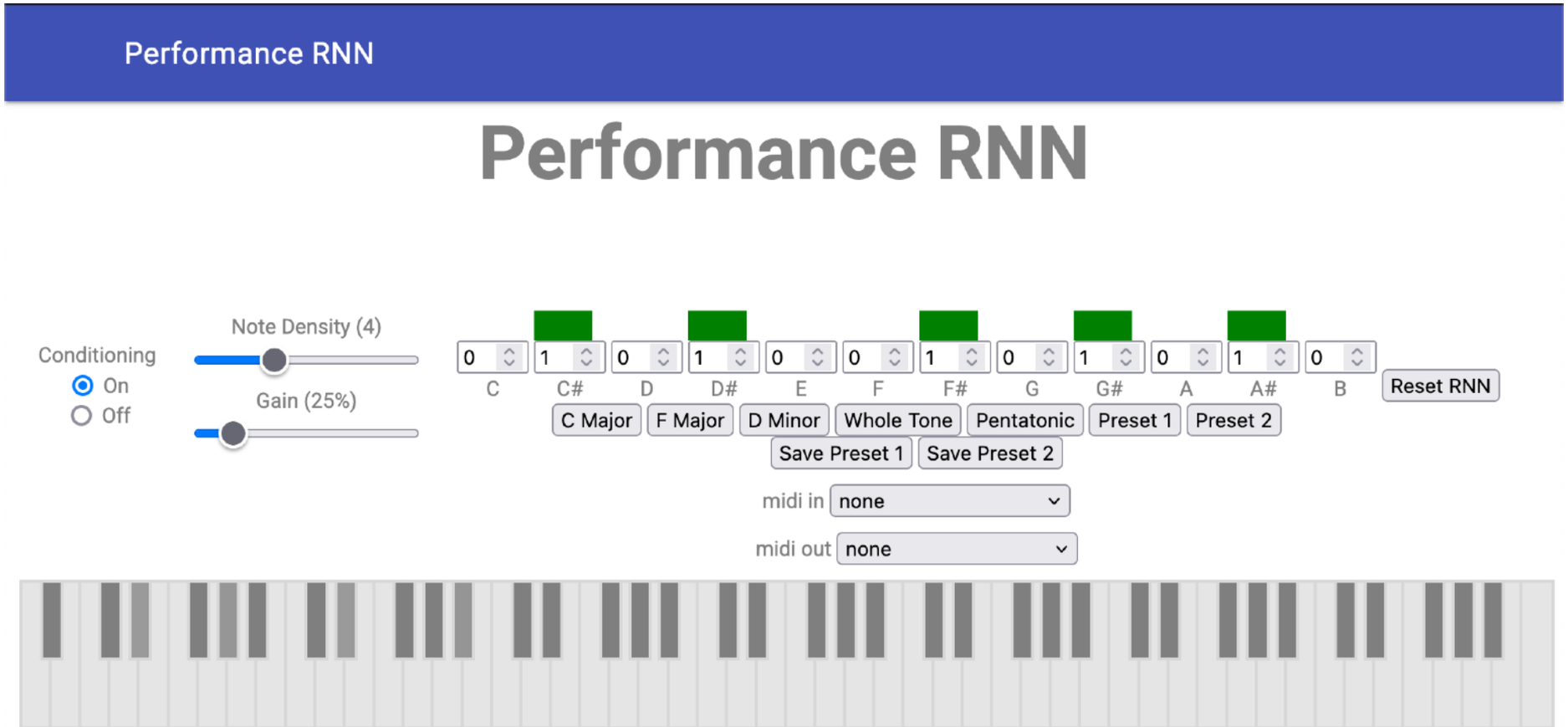
1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

example 11 | MACHINE LEARNING on the WEB



1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

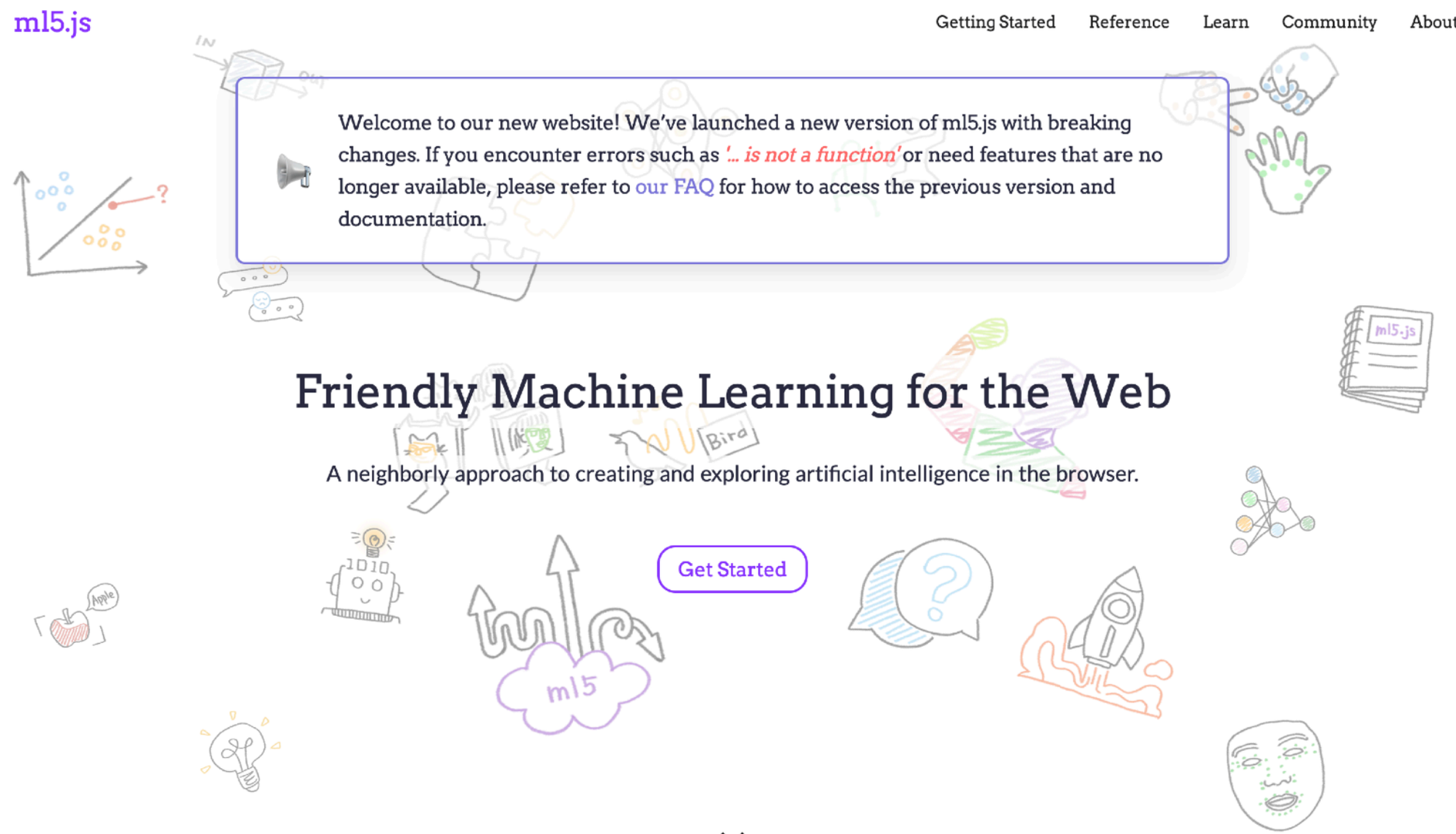
example 12 | MACHINE LEARNING on the WEB - Recurrent NN



https://magenta.tensorflow.org/performance-rnn

1.7 (NON-EXHAUSTIVE) LIST OF WEB AUDIO USES

example 13 | MACHINE LEARNING on the WEB - ml5.js



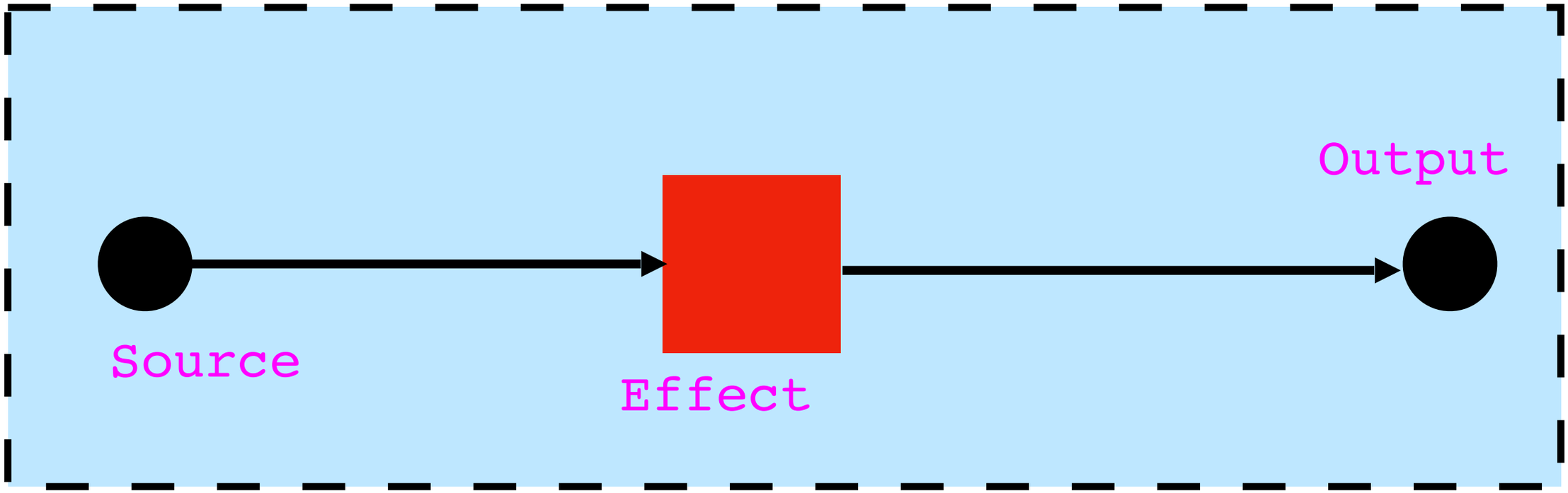
https://ml5js.org/

2. WEB AUDIO

BASIC CONCEPTS

2.1 MAIN FEATURES

1. it uses a **modular routing logic**: with an architecture of type: source -> effects -> output. Very similar to that of visual programming software (such as Max/MSP) or modular analog synthesis environments.



2.1 MAIN FEATURES

2. it works at **high bit-rate** (32-bit floats) for audio processing;

3. it has an accurate **event scheduler** (with low latency), that allows to deal with events in time;

4. it offers the possibility to **automate parameters**, for the creation of envelopes, crossfades, LFOs, etc. ;

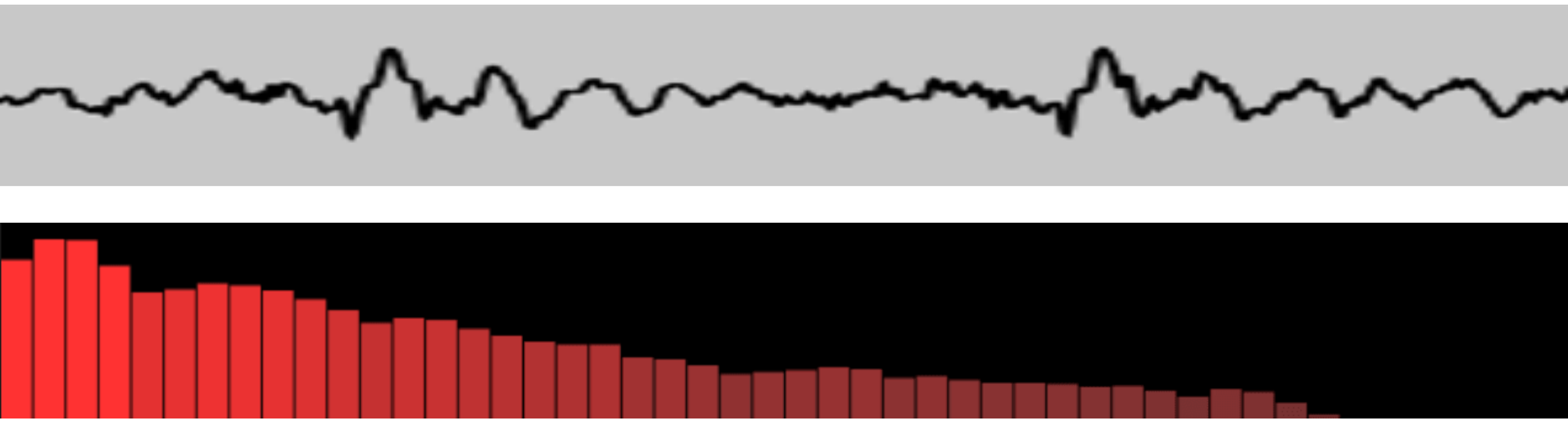
5. it allows **real-time DSP** (Digital Signal Processing);

6. it offers **various spatialization algorithms**;

2.1 MAIN FEATURES

7. it has a **convolution module**, for creating effects such as: convolution reverbs, filters, cross synthesis, etc.;

8. it offers support for **sound visualization** in both time (*waveform*) and frequency domain (*spectrum*);



2.1 MAIN FEATURES

9. it offers **standard effects modules** such as: filters, compressor, delay, reverb. etc.;

10. it has a **waveshaping module** for creating distortion and other non-linear effects;

11. it has a set of **standard waveform oscillators**;

12. offers a **buffer module** for creating a memory array in which to store sample values.

2.2 MODULAR DESIGN

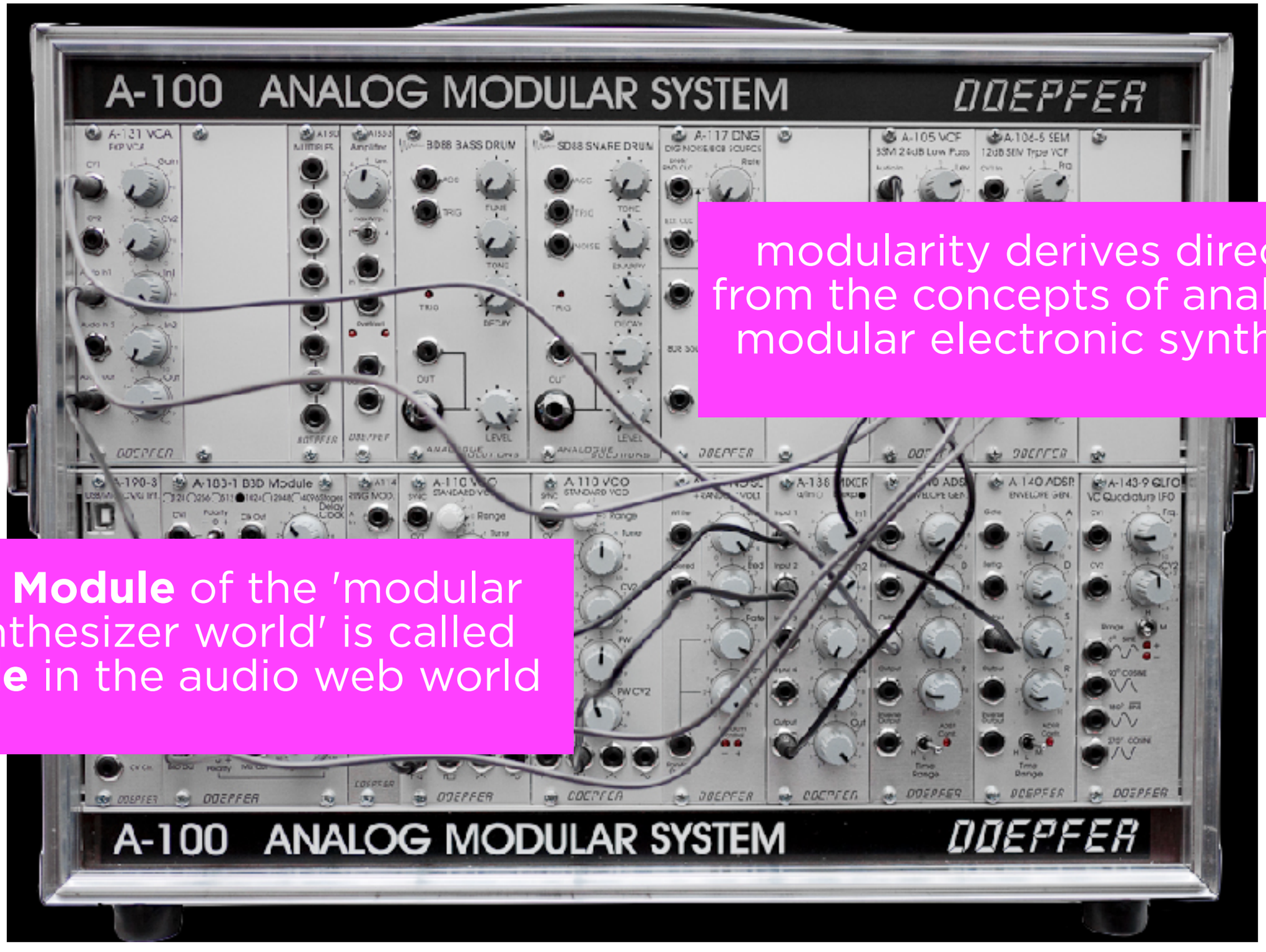
The web audio works with **modular logic**, where **NODES** are connected together to create a stream of signals.

A **NODE** is a **general operator** that performs a certain functionality (such as: generate a sound, modify a sound, etc.).

Nodes are linked together into **chains** to form an **audio routing GRAPH** into an **`audioContext`**, that provides us with the audio rendering context.

The web audio involves handling **audio operations** using **Nodes** inside an **audio context**.

2.2 MODULAR DESIGN



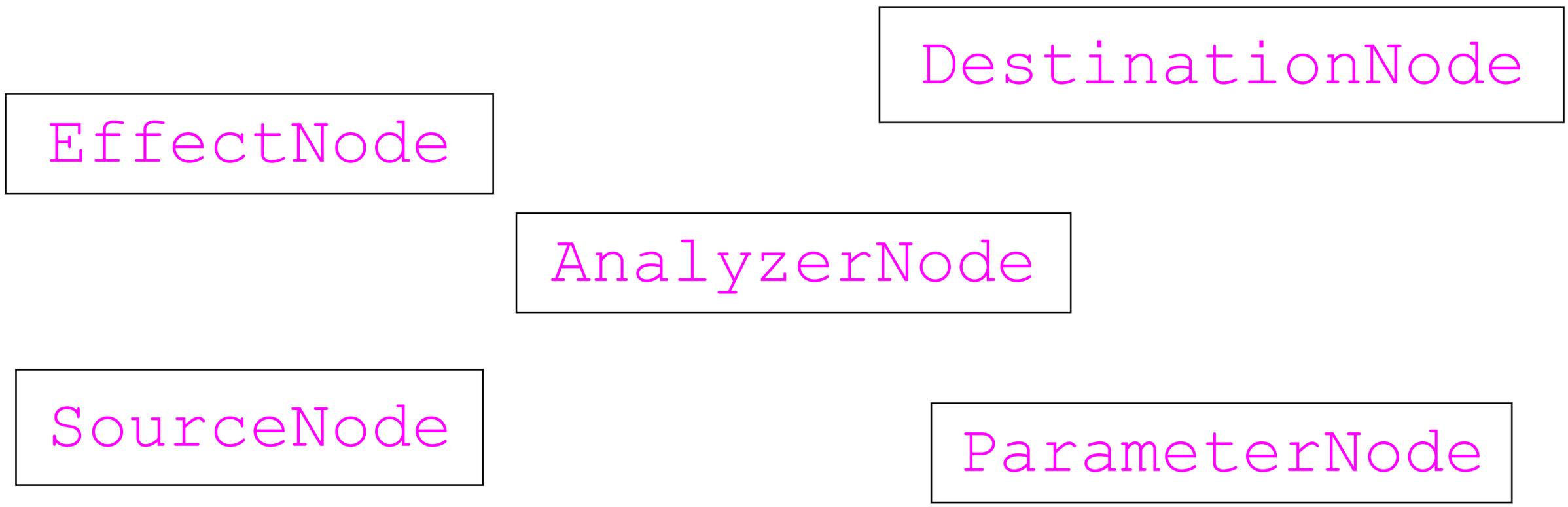
modularity derives directly from the concepts of analogue modular electronic synthesis

the **Module** of the 'modular synthesizer world' is called **Node** in the audio web world

2.2 MODULAR DESIGN

A **Node** is an operator capable of performing anaudio task.

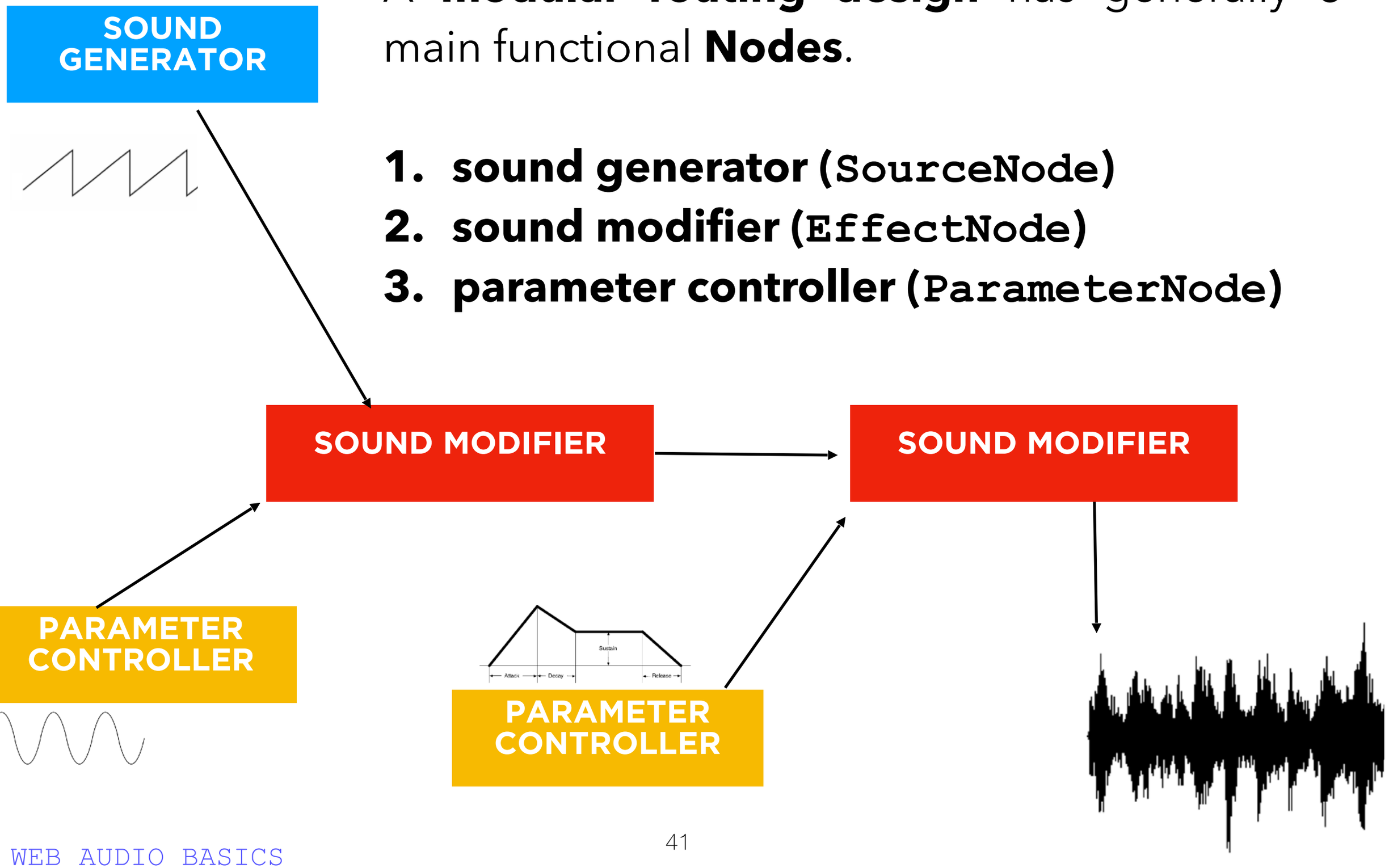
There are different types of **Nodes**, for example: *source nodes*, *effect nodes*, *parameter nodes*, *destination nodes*, *analyzer nodes*, etc..



2.2 MODULAR DESIGN

A **modular routing design** has generally 3 main functional **Nodes**.

1. **sound generator** (`SourceNode`)
2. **sound modifier** (`EffectNode`)
3. **parameter controller** (`ParameterNode`)

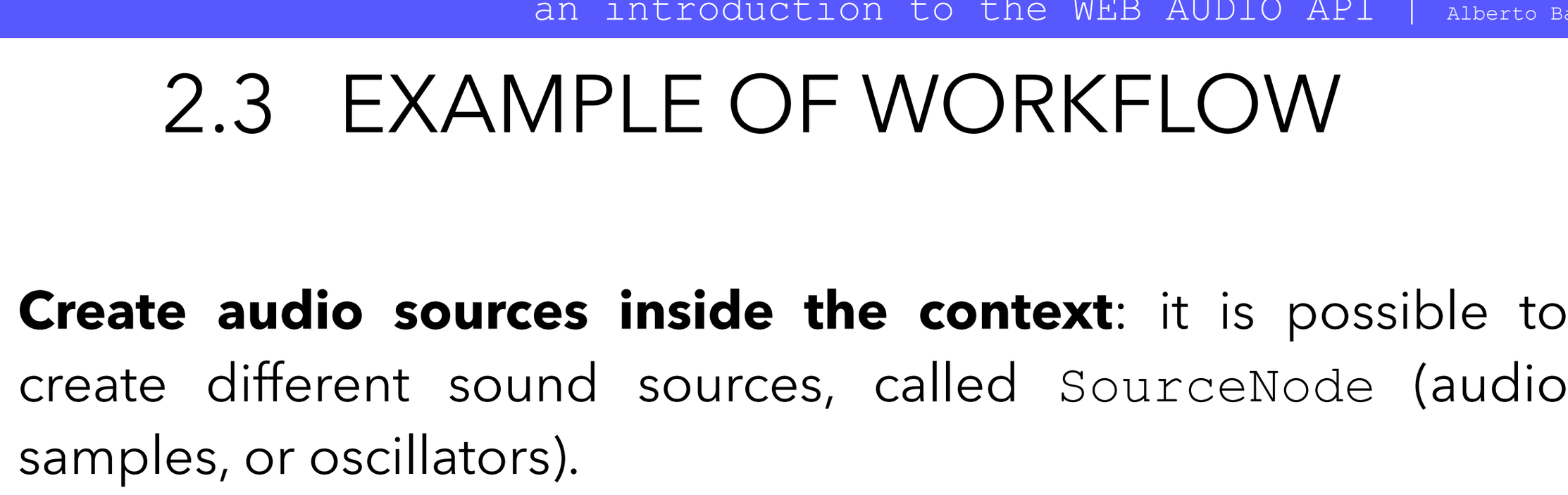


2.3 EXAMPLE OF WORKFLOW

1. **Create an audio context**: an `AudioContext` is a kind of container for AudioNode objects, which allow different kind of audio sources and processing.

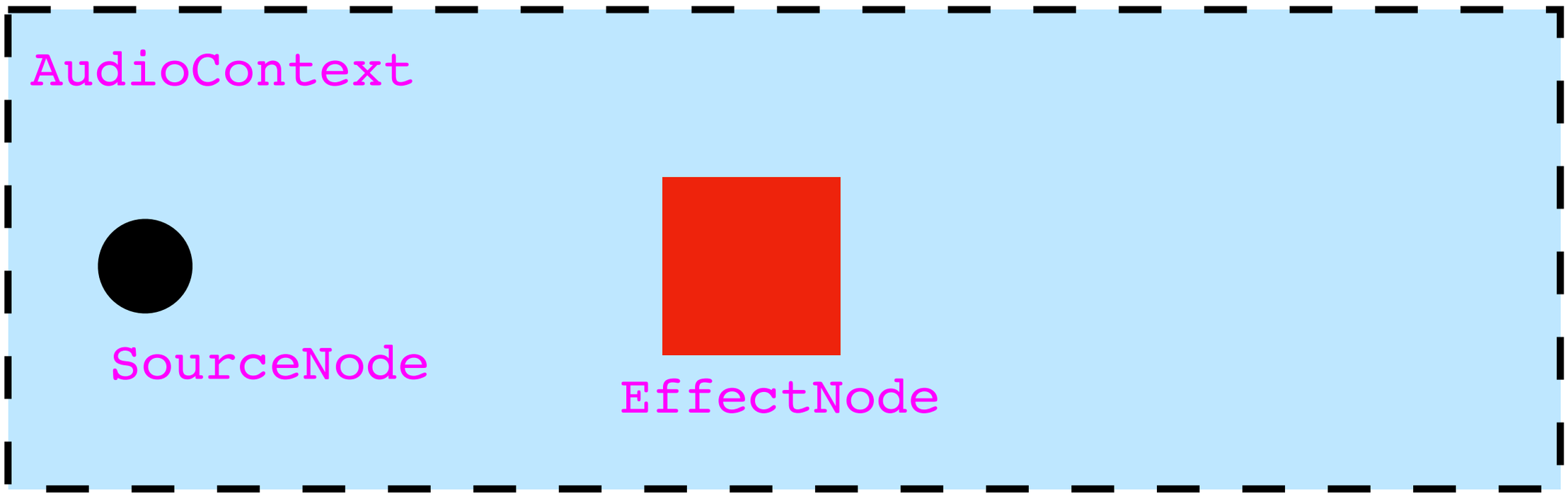
2.3 EXAMPLE OF WORKFLOW

2. **Create audio sources inside the context**: it is possible to create different sound sources, called `SourceNode` (audio samples, or oscillators).



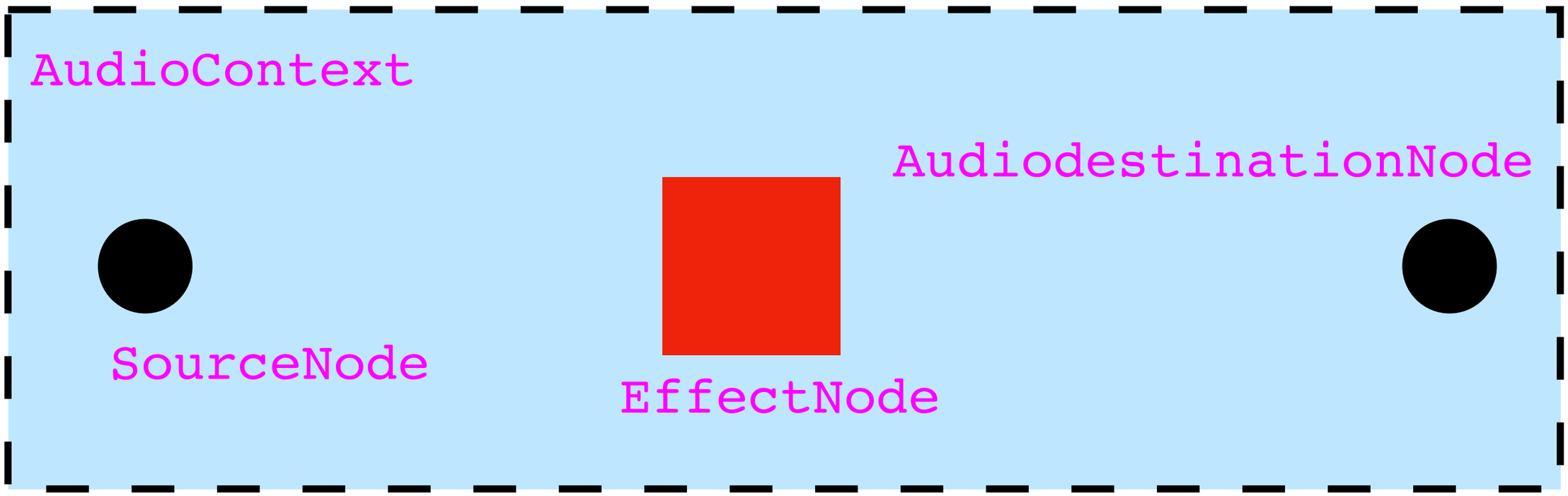
2.3 EXAMPLE OF WORKFLOW

3. **Create effects nodes**: it is possible to apply different audio effects (`EffectNode`) to the audio signal source (filter, delay, compressor, stereo panning, convolver, etc.).



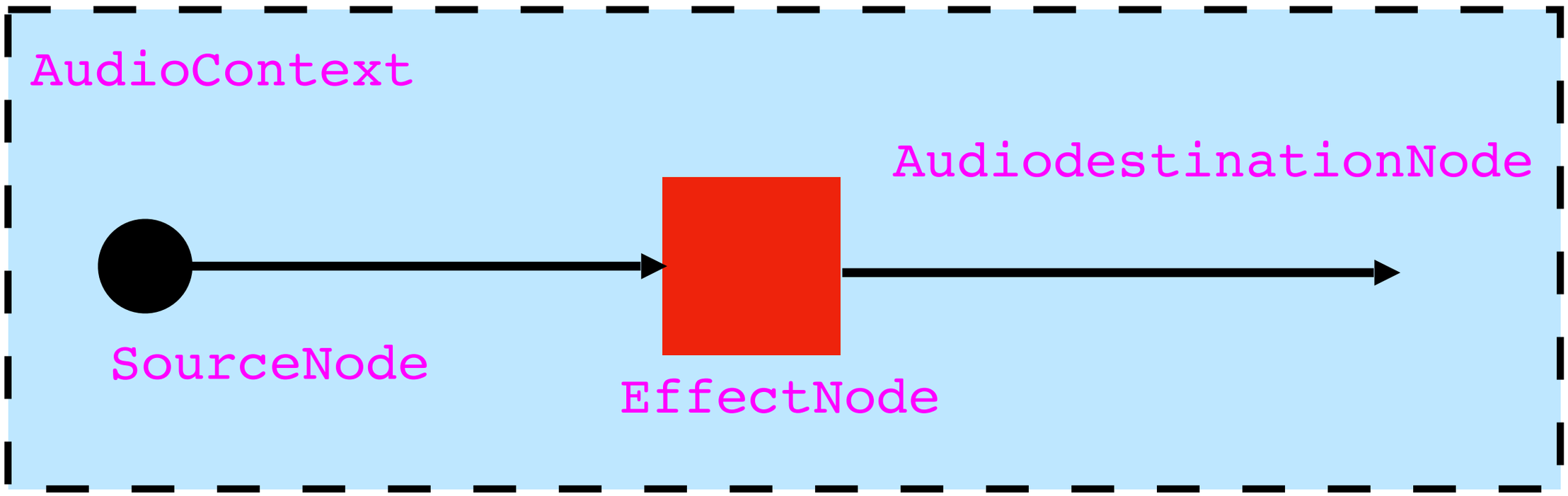
2.3 EXAMPLE OF WORKFLOW

4. **Choose a final destination for the audio chain**: the `AudioDestinationNode` routes the sound inputs to a final audio destination, usually some kind of speaker system.



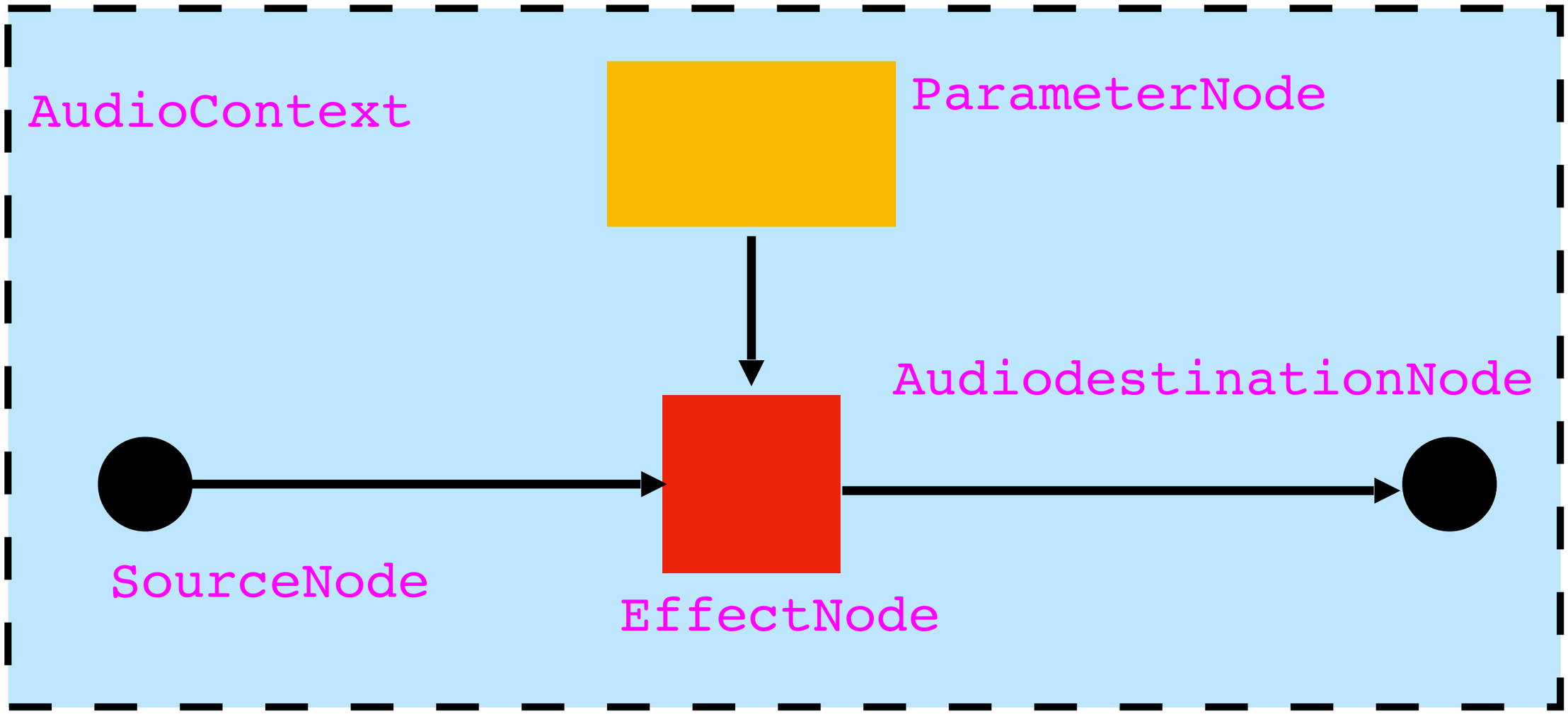
2.3 EXAMPLE OF WORKFLOW

5. **Connect** the **sources** to the **effects** and the **effects** to the **destination** (creating a chain).



2.3 EXAMPLE OF WORKFLOW

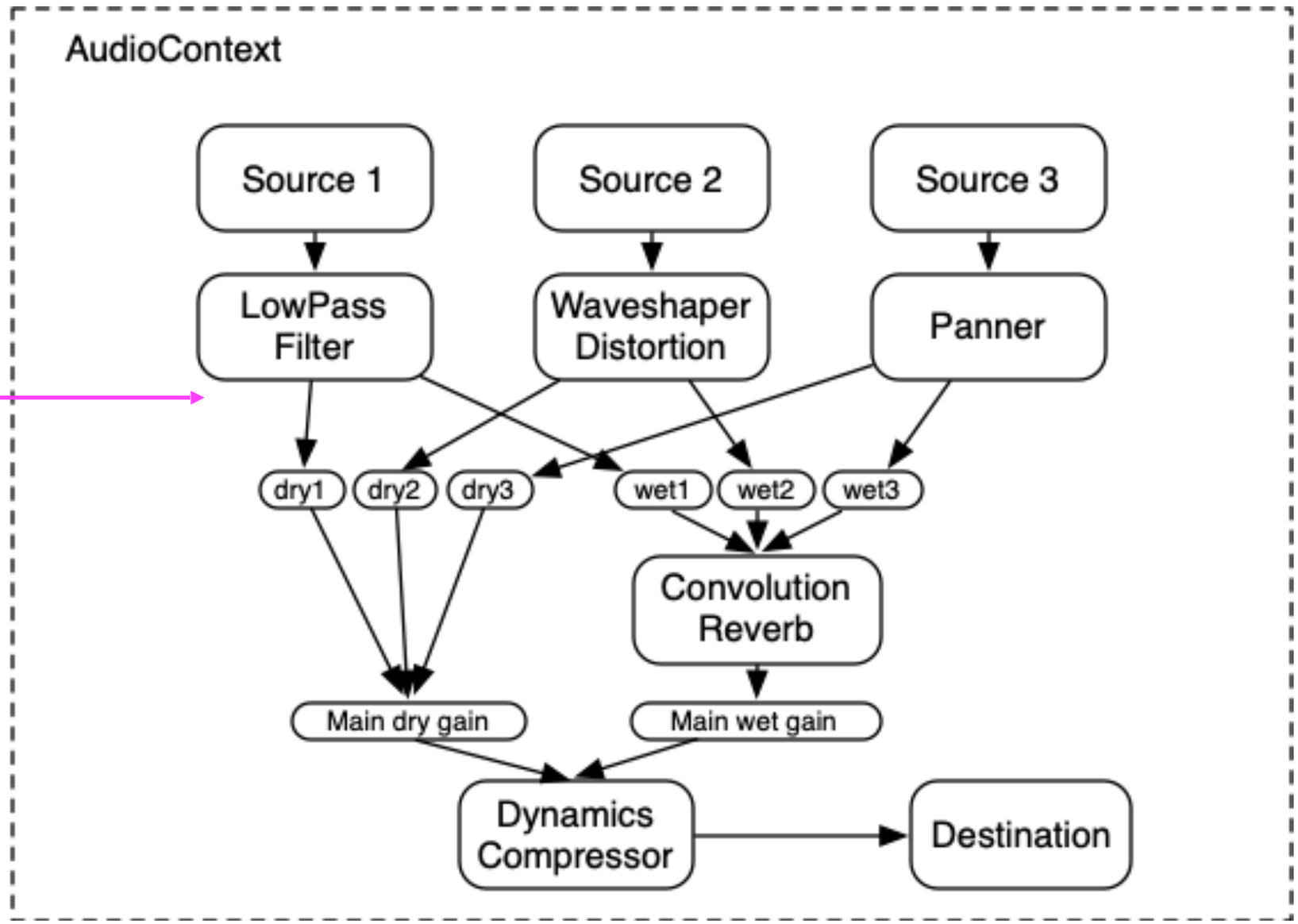
6. **Add** a `parameterNode` and connect it to the `EffectNode`, to control some parameters.



2.3 EXAMPLE OF WORKFLOW

A **graph** is a collection of nodes. Any configuration of nodes (graphs) can be created in the `audioContext`.

example of the flow chart of a possible web audio graph



3. HTML

3.1 HTML

HTML is the standard **markup language for creating Web pages**.

The HTML language manages the **structure** of a web page.

- HTML stands for **Hyper Text Markup Language**;
- HTML describes the **structure** of a Web page;
- HTML consists of a series of **ELEMENTS**;
- HTML **ELEMENTS** tell the browser **how to display the CONTENT**;
- HTML **ELEMENTS** are represented by **TAGS**;
- HTML **TAGS** label pieces of content such as "heading", "paragraph", and so on;
- Browsers use HTML **TAGS** to render the **CONTENT** of the page;

3.2 HTML: ELEMENTS

For each **type of CONTENT** of a web page there is a specific **ELEMENT** (eg: the element of the *page itself*, the element of the *headings*, the element for the *images*, the one for the *texts*, the one for the *hyperlinks*, for the *buttons*, etc.).

ELEMENTS can (and often must) be **nested**

Each **ELEMENTS** has a ***tagName*** that is **unique**.

Syntax for creating an element:

The element **starts** with the syntax: ***<tagName>***
The element **ends** with the syntax: ***</tagName>***

There are some **ELEMENTS** that only have the syntax of the opening tag.

this is the starting point of a tag of type `html`, this tells the browser that the html file starts here

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

this is the tag `head`, which contains meta-informations (like the title of the page itself)

the `title` is an other tag

the body contains the elements which will be shown to the user on the window

this is an element of type `heading 1`, that means that the text into it will be displayed with a big font size (important, for a title)

this is an element that represents a normal `paragraph` of text (with normal font size)

3.3 HTML: ATTRIBUTES

All HTML **ELEMENTS** can have **ATTRIBUTES**.

ATTRIBUTES provide **additional information** for **ELEMENTS**.

ATTRIBUTES are always specified in the **start tag**.

An ATTRIBUTE can be inserted into a tag ELEMENT using a syntax in name/value pairs:

```
<tagName attributeName = "attributeValue" >

</tagName>
```

Let's open the file *index.html with Visual Studio Code*

The tag of type `script` allows us to load a javascript file to our HTML

Here we load one javascript library (p5.js) and the file webAudioWorkshop.js

`b` is the tag for the font style bold

This is an `a` element (anchor element) which creates a hyperlink to web pages

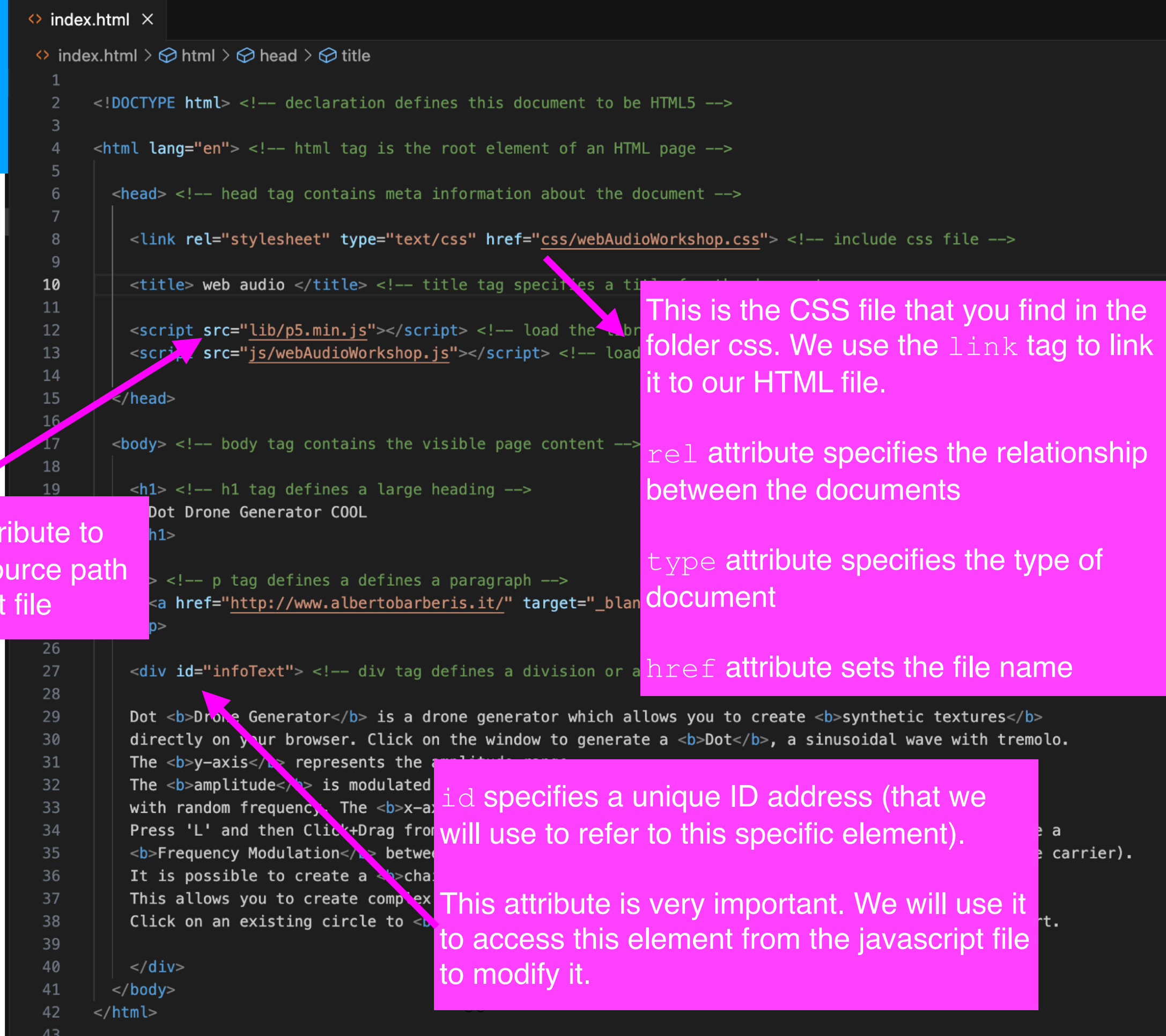
This is a `div` element (division), a general section of the HTML file where we insert a text

index.html

index.html > html > head > title

```
1
2   <!DOCTYPE html> <!-- declaration defines this document to be HTML5 -->
3
4   <html lang="en"> <!-- html tag is the root element of an HTML page -->
5
6     <head> <!-- head tag contains meta information about the document -->
7
8       <link rel="stylesheet" type="text/css" href="css/webAudioWorkshop.css"> <!-- include css file -->
9
10      <title> web audio </title> <!-- title tag specifies a title for the document -->
11
12      <script src="lib/p5.min.js"></script> <!-- load the libraries we need -->
13      <script src="js/webAudioWorkshop.js"></script> <!-- load the javascript file -->
14
15    </head>
16
17    <body> <!-- body tag contains the visible page content -->
18
19      <h1> <!-- h1 tag defines a large heading -->
20        Dot Drone Generator COOL
21      </h1>
22
23      <p> <!-- p tag defines a defines a paragraph -->
24        <a href="http://www.albertobarberis.it/" target="_blank" > Alberto Barberis </a>
25      </p>
26
27      <div id="infoText"> <!-- div tag defines a division or a section in an HTML document; it is a general element -->
28
29      Dot <b>Drone Generator</b> is a drone generator which allows you to create <b>synthetic textures</b>
30      directly on your browser. Click on the window to generate
31      The <b>y-axis</b> represents the amplitude range
32      The <b>amplitude</b> is modulated by a <b>triangular LFO
33      with random frequency. The <b>x-axis</b> represents the f
34      Press 'L' and then Click+Drag from an existing Dot to an
35      <b>Frequency Modulation</b> between them. The first Dot b
36      It is possible to create a <b>chain of modulation</b>: each carrier can become a modulator.
37      This allows you to create complex spectra, to the point of creating very noisy sounds!
38      Click on an existing circle to <b>delete</b> it or to delete the modulation chain of which it is part.
39
40      </div>
41    </body>
42  </html>
```

Let's open the file *index.html with Visual Studio Code*



`src` is the attribute to specify the source path of a javascript file

This is the CSS file that you find in the folder css. We use the `link` tag to link it to our HTML file.

`rel` attribute specifies the relationship between the documents

`type` attribute specifies the type of document

`href` attribute sets the file name

`id` specifies a unique ID address (that we will use to refer to this specific element).

This attribute is very important. We will use it to access this element from the javascript file to modify it.

Dot Drone Generator

Alberto Barberis

Dot **Drone Generator** is a drone generator which allows you to create **synthetic textures** directly on your browser. Click on the window to generate a **Dot**, a sinusoidal wave with tremolo. The **y-axis** represents the amplitude range. The **amplitude** is modulated by a **triangular LFO** (Low Frequency Oscillator), with random frequency. The **x-axis** represents the frequency range. Press 'L' and then Click+Drag from an existing Dot to an other one, to link two sinusoids and create a **Frequency Modulation** between them. The first Dot becomes the modulator of the second one (the carrier). It is possible to create a **chain of modulation**: each carrier can become a modulator. This allows you to create complex spectra, to the point of creating very noisy sounds! Click on an existing circle to **delete** it or to delete the modulation chain of which it is part.

This is the web page that you should see on your browser when you double click on the *index.html* file

If you click on the web page nothing happens yet because the **javascript** file that will manage the user interaction and the sound synthesis methods does not exist (yet)!

4. CSS

4.1 CSS: WHAT IS IT?

CSS (Cascading Style Sheets) is a language that describes the **style of an HTML document** (describes how HTML elements should be displayed).

With the CSS language you can modify the **presentation of the ELEMENTS** of an HTML code (including layout, colors, and fonts).

CSS is designed to enable the **logical separation** of **presentation/style** (CSS) and **content/structure** (HTML).

HTML (contents and structure)

CSS (style and presentation)

https://www.w3schools.com/css/default.asp

4.2 CSS: STYLE RULES AND SPECIFICITY

CSS works applying some **style RULES** to an HTML **ELEMENT**.

It is possible to apply **different RULES to the same ELEMENT**.

The name **cascading** comes from the specified **priority scheme** to determine which style rule applies.

Some rules have a **higher priority** (specificity) than others and **this priority depends on the manner in which the rules are defined**.

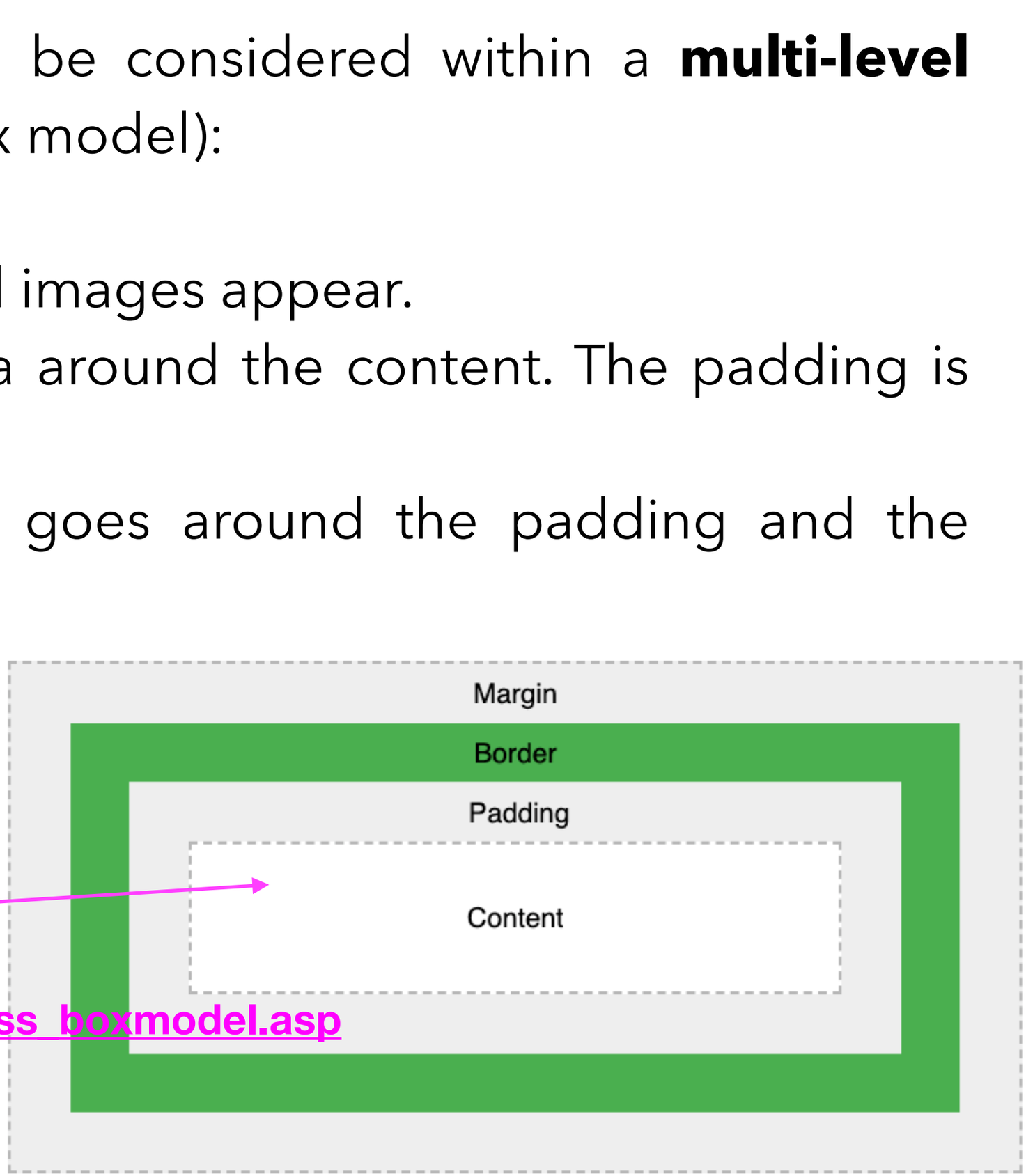
https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity

4.3 CSS: THE BOX MODEL

All HTML **ELEMENTS** can be considered within a **multi-level box** composed by (the box model):

- **Content** : where text and images appear.
- **Padding** : clears an area around the content. The padding is *transparent*;
- **Border** : a border that goes around the padding and the content (can be *visible*);
- **Margin** : clears an area around the border. The margin is *transparent*.

this is the Content
(for example a text)



https://www.w3schools.com/css/css_boxmodel.asp

4.4 CSS: THE SELECTORS AND RULES

The CSS language is made of different **SELECTORS** and **RULES**. The **SELECTORS** selects certain html code elements on which to apply certain style **RULES**.

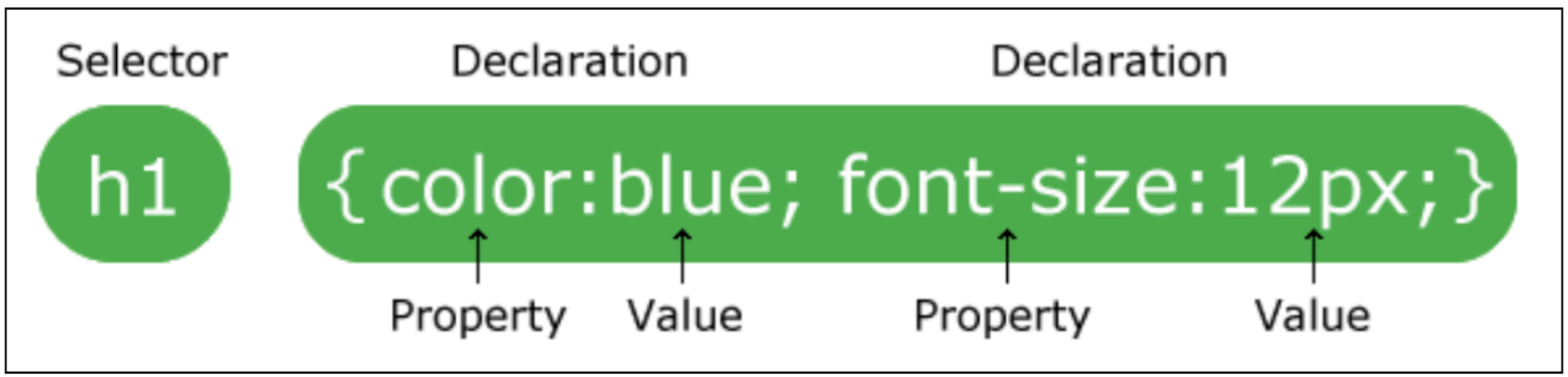
There are different **types of selectors** and **each selector refers to a specific HTML ELEMENT** or to a **GROUP OF ELEMENTS** with some shared characteristics.

```
1   body { /* selector body */
2       margin: 0px;
3       font-family: 'Courier New';
4       user-select: none; /* user can not se
5   }
```

This are the RULES in the form `property: value;` to be applied to the selected elements

This is a SELECTOR: it specifies which elements the rules refer to

4.4 CSS: THE SELECTORS AND RULES



A CSS rule-set consists of:

1. a **SELECTOR**, that points to the element you want to style.
2. a RULE, a **declaration block** in curly braces, that contains one or more declarations separated by *semicolons*. Each declaration includes a **CSS property NAME** and a **VALUE**, separated by a *colon*.

Selector	Example	Example description
.class	.intro	Selects all elements with class="intro"
.class1.class2	.name1.name2	Selects all elements with both *name1* and *name2* set within its class attribute
.class1 .class2	.name1 .name2	Selects all elements with *name2* that is a descendant of an element with *name1*
#id	#firstname	Selects the element with id="firstname"
*	*	Selects all elements
element	p	Selects all <p> elements
element.class	p.intro	Selects all <p> elements with class="intro"

https://www.w3schools.com/cssref/css_selectors.asp

There are different types of selectors and there is also a hierarchy. This allows the definition of different styles in a hierarchical manner (e.g. all elements in the body have a certain font style, but only those within the title tag are also bold).

```css
body { /* selector body */
    margin: 0px;
    font-family: 'Courier New';
    user-select: none; /* user can not select the text */
}
h1 { /* selector h1 */
    margin: 0px;
    padding: 10px;
    background-color: rgb(255, 79, 79);
    color: white;
}
p { /* selector paragraph */
    margin: 0px;
    padding: 10px;
    color: rgb(0, 0, 0);
    border-bottom: solid;
    border-width: 1px;
}
#infoText { /* selector element with ID infoText */
    position: fixed;
    margin-top: 50px;
    margin-left: 100px;
    margin-right: 100px;
    text-align: justify;
    line-height: 130%;
}
a { /* selector element a */
    text-decoration: none;
    color: black;
}
a:hover { /* selector element a when mouse is over */
    color: rgb(255, 79, 79);
}
```

These selectors refer to all the elements of a certain type (eg body, paragraph)

The selector that starts with `#` is a selector that refers to an element with a specific id.

The `div` element that we use to present the text in the HTML file has a special attribute `id="infoText"`.

```html
<div id="infoText">
```

This refers to the elements a but when the mouse is over the text

5. MODULATION SYNTHESIS

5.1 SYNTHESIS TECHNIQUE: MODULATION

The **Dot Drone Generator** is a web audio instrument that uses two **modulation sound synthesis techniques** called:

1. **AMPLITUDE MODULATION** (**AM**) for **a tremolo effect**.
2. **FREQUENCY MODULATION** (**FM**).

We speak of **modulation synthesis** when at least one of the parameters of an oscillator (amplitude/frequency/phase) **varies continuously in time in relation to another signal** (usually a periodic signal).

$$x(t) = A \cdot sin(2\pi ft + \varphi)$$

AM FM PM

$$x(t) = A(t) \cdot sin(2\pi f(t)t + \varphi(t))$$

5.2 SYNTHESIS TECHNIQUE: AM

To obtain a **tremolo** we implement an AM. A sinusoid (**carrier**) will be amplitude modulate by an other oscillator (a triangular wave **modulator**) that oscillates at Low Frequency (LFO).

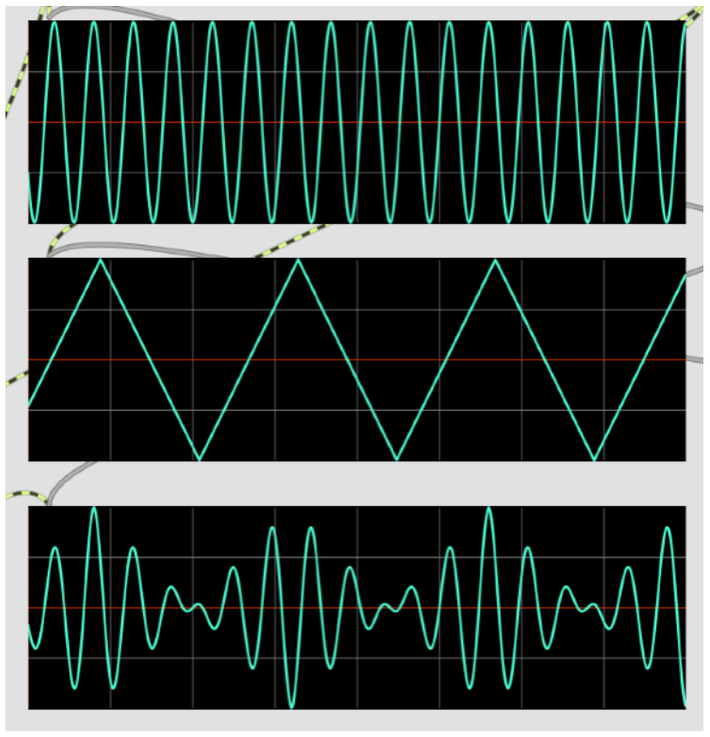
If the frequency of the modulator is in the **sub audio range** (< 15 Hz) the timbre of the carrier does not change, but its volume (amplitude) change accordingly to the shape of the modulator.

$$x(t) = A_{tri}(t) \cdot sin(2\pi ft + \varphi)$$

carrier

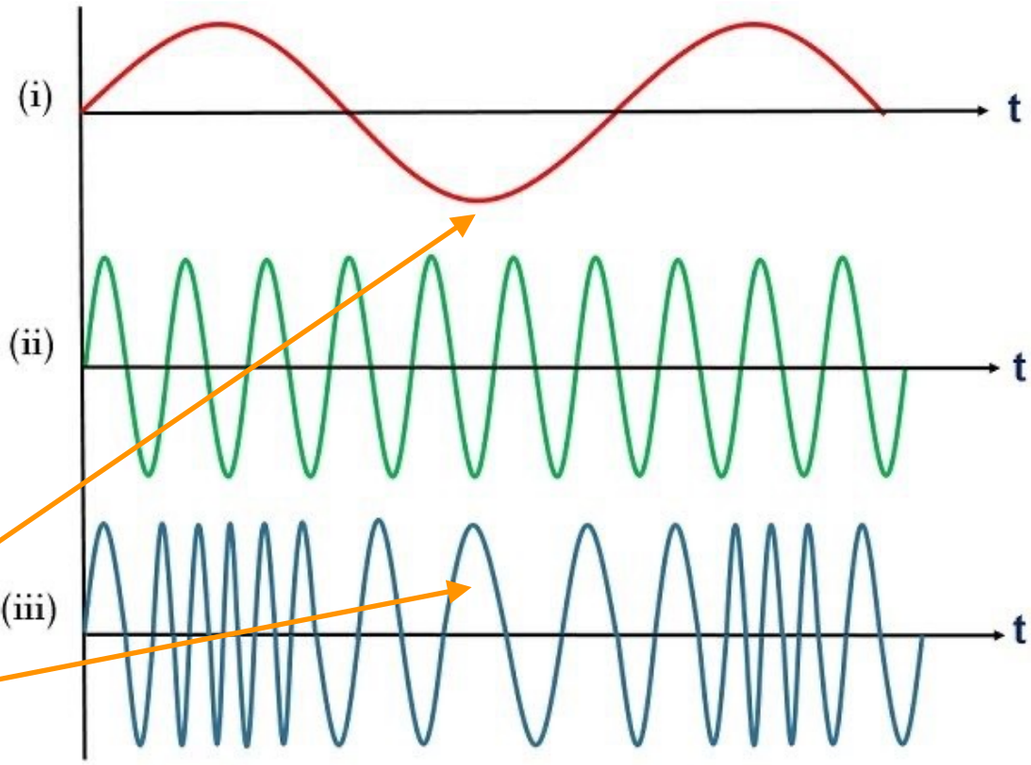
modulator

modulated wave



5.3 SYNTHESIS TECHNIQUE: FM

In the **frequency modulation (FM)**, the frequency of an oscillator is not fixed but it changes accordingly to an other periodic function of time.



(i) Modulating signal
(ii) Carrier waveform
(iii) Frequency modulated signal

modulator

carrier

modulated wave

here the modulator is at its lowest point, so the frequency of the modulated wave is the lowest

5.3 SYNTHESIS TECHNIQUE: FM

In the **linear FM** the frequency of the original signal (**carrier**) is no longer constant but it varies over time periodically, according to the formula:

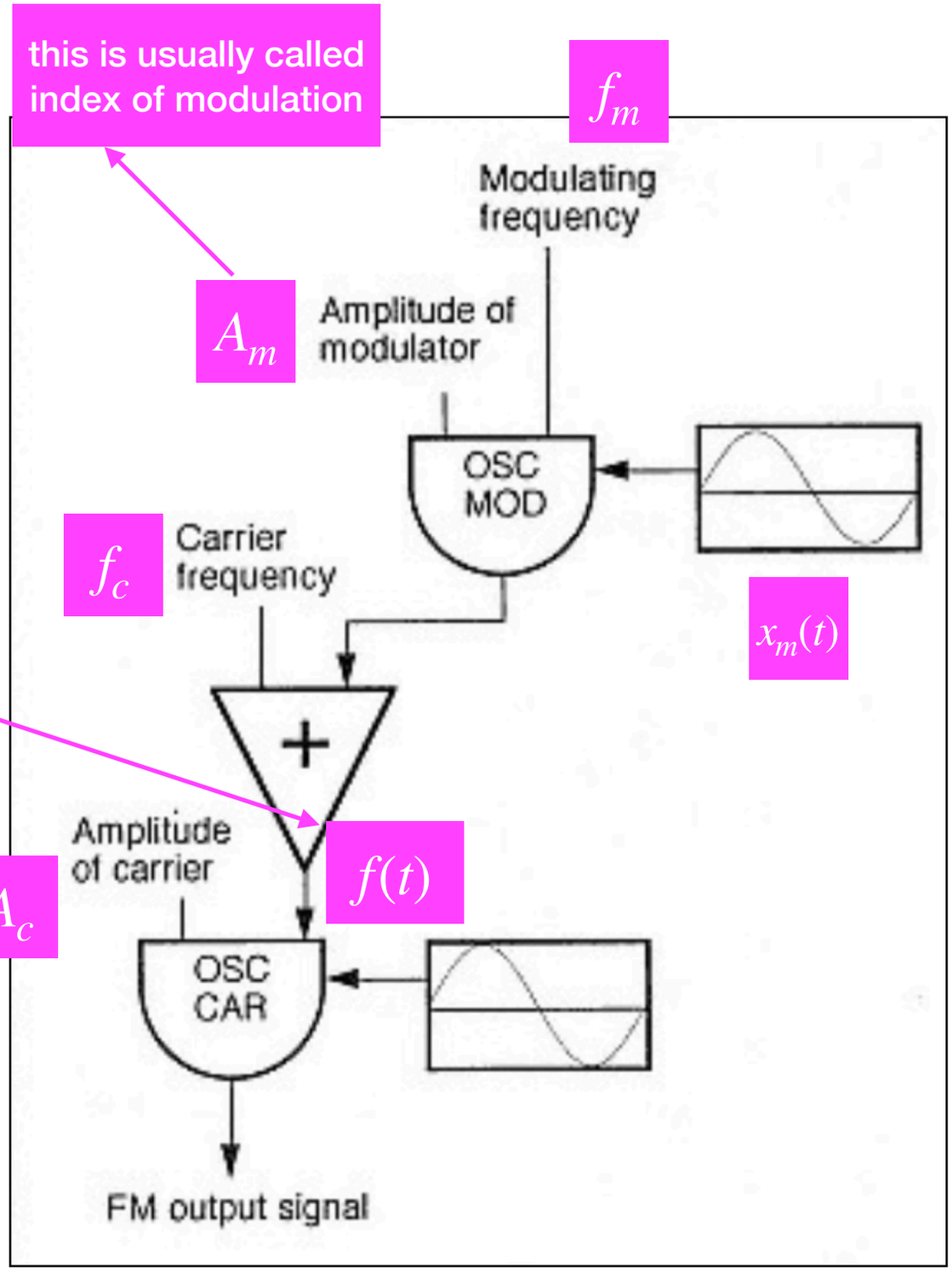
$$f(t) = f_c + A_m \cdot x_m(t)$$

Where:

- f_c = carrier frequency

- A_m = amplitude of the modulator (called **INDEX OF MODULATION**)

- $x_m(t)$ = sinusoidal modulating signal



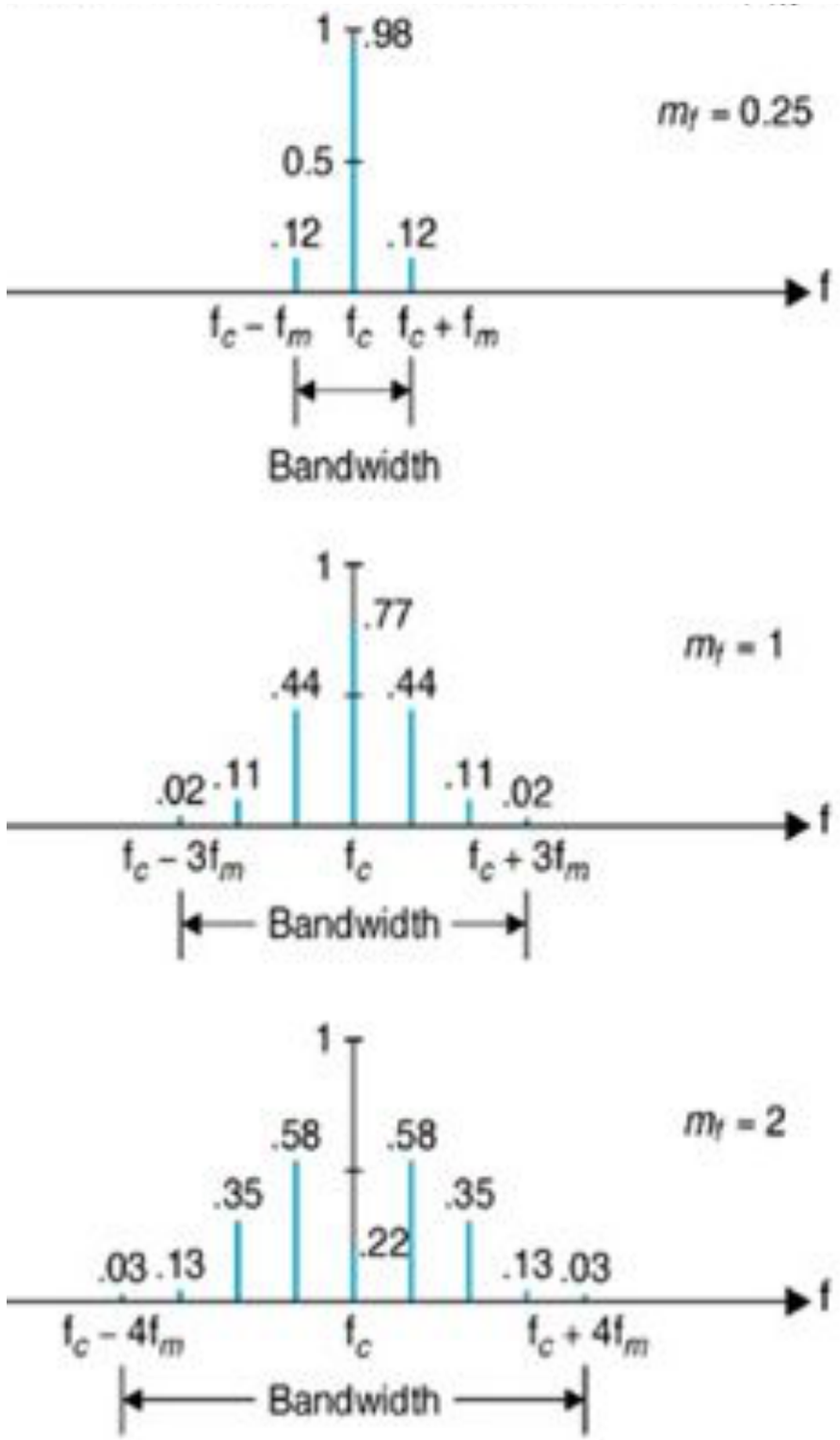
5.3 SYNTHESIS TECHNIQUE: FM

The FM synthesis is often used to generate **complex spectra**, starting from simple signals (sinusoids).

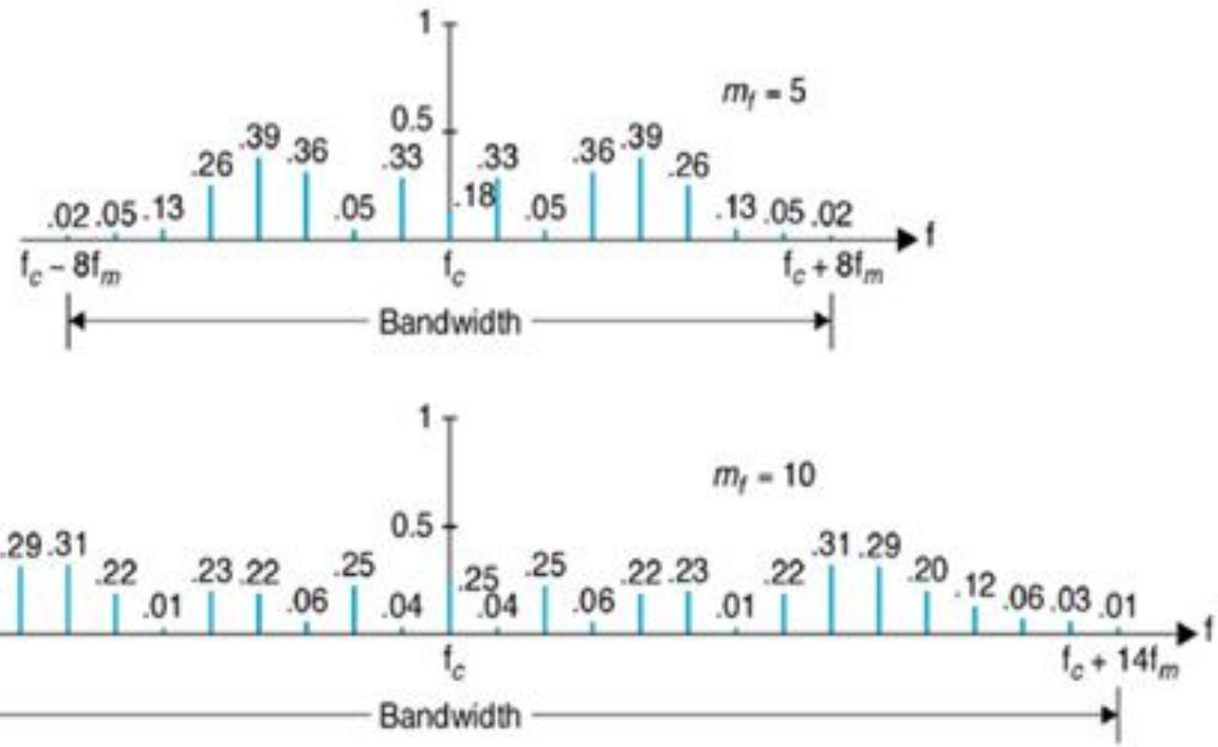
As the **amplitude of the modulator signal** A_m (INDEX OF MODULATION) increases, the following occurs:

- the **carrier frequency decreases** in amplitude;
- sinusoidal **sidebands appear** at defined frequencies: $f_c \pm nf_m$ where n is the integer index of the order of the sidebands;
- the **spectral energy** is "stolen" from the carrier frequency and is re-distributed in some sidebands;
- after a certain threshold of A_m, the **carrier reappears** (this is because the amplitude of the components of the spectrum follows the *Bessel functions*).

5.3 SYNTHESIS TECHNIQUE: FM



example of spectral evolution at different INDEX OF MODULATION



the Chowning FM classical instrument

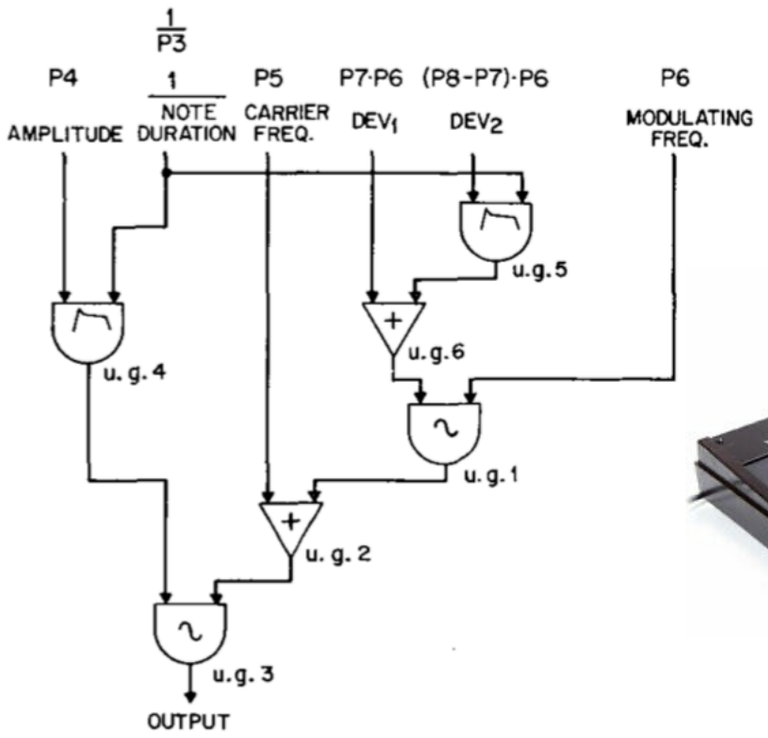
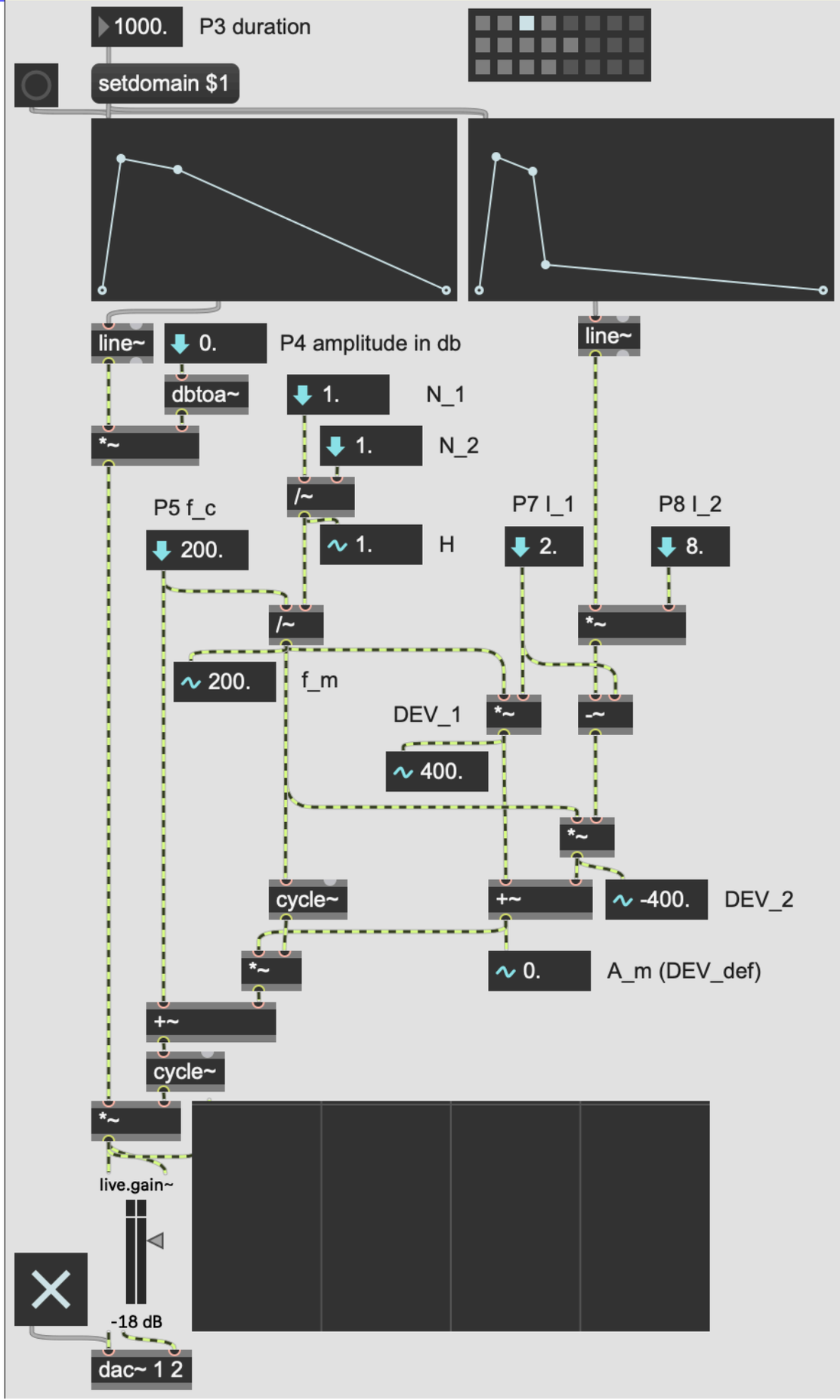


Fig. 10. FM circuit to produce dynamic spectra. Two function generators are added, u.g. 4 and u.g. 5, to produce an amplitude envelope and a modulation index envelope which causes the bandwidth to vary.

parameters for this instrument will have the following function:

P_1 = Begin time of instrument
P_2 = Instrument number
P_3 = Duration of the "note"
P_4 = Amplitude of the output wave
P_5 = Carrier frequency
P_6 = Modulating frequency
P_7 = Modulation index 1, I_1
P_8 = Modulation index 2, I_2.



6. Javascript

6.1 JAVASCRIPT: LANGUAGE

Javascript (JS) transforms **web pages** from **static to dynamic**, thanks to **scripts** (portions of code) **inserted in the HTML** through the <script> tag.

https://www.w3schools.com/js/default.asp

JS code is **executed by the browser**, through a javascript engine (a software that executes JS code). If an HTML document contains a **script**, the instructions contained in it are executed when it is loaded.

The JS instructions can be used to associate functions to events that respond to the interaction of a user with the interface.

JS programming follows an **event model** programming method.

6.2 JAVASCRIPT: FEATURES

- JS is a **procedural scripting language**: it consists of a series of instructions to be executed when the program requires it.

- JS is an **interpreted language**: it is executed by a program called interpreter (the browser). The code is not compiled into machine code beforehand. Instead, the it is translated into machine code line by line by an interpreter at runtime.

- JS is a **dynamic typing language**: it **does not require the definition of data types for variables and functions**.

- JS uses the paradigm of **object oriented programming** (OOP); although it is not a "true" OOP language, it is more and more object oriented.

https://www.w3schools.com/js/default.asp

6.3 JAVASCRIPT: VARIABLE AND CONSTANT

Keywords to create VARIABLES and CONSTANTS.

- **`let`** is the keyword to define a **variable**, that is a sort of container for data of any type (number, string, Object, Array, etc.);
    - the data stored inside the variable can change over time;
    - a variable must be **declared** (with a specific unique name) and eventually **initialized** (with an initial value)

- **`const`** is the keyword to define a **constant**, that is a container for data that can not change;
    - the data stored in a constant can not be substituted;
    - a variable must be **declared** (with a specific unique name) and **initialized** (with a value that will remain constant).

Let's open the file *webAudioWorkshop.js* with Visual Studio Code

```
/** ////////////////////////////////////////////////////////////////////////
  * GLOBAL CONSTANT AND VARIABLES
    * definition
    * initialization
* ////////////////////////////////////////////////////////////////////////*/

const MIN_FREQUENCY_HZ = 5; // min frequency in the x axis
const MAX_FREQUENCY_HZ = 3000; // max frequency in the x axis
const MIN_AMPLITUDE = 0.01; // min amplitude in the y axis
const MAX_AMPLITUDE = 0.5; // max amplitude in the y axis (never higher that
const CANVAS_OFFSET = 100; // offset for the canvas
const TITLE_OFFSET = 75; // sum of text + padding of title and paragraph
const MASTER_GAIN = 0.01; // gain adaptation
const MODULATION_INDEX = 1000; // index of modulation for the FM
const MIN_AM_FREQ = 0.05; // min freq for the AM oscillator
const MAX_AM_FREQ = 0.5; // mas freq for the AM oscillator (never higher that 0.5)
const ATTACK_TIME = 1.6; // attack time for sounds
const RELEASE_TIME = 0.3; // release time for sounds
const MIN_DIAMETER = 10; // release time for sounds
const MAX_DIAMETER = 100; // release time for sounds

let audioContext; // variable that will store the audio context
let masterGain; // a Gain Node for the MASTER volume
let arrayOfDots = []; // array of Dots (oscillators)
let modulator = null; // variable that will contain the modulat
let carrier = null; // variable that will contain the carrier o
```

Here we declare and initialize some constants (usually with a capital letters name)

This is only a variable declaration; there is not initialization. This means that is content is `undefined`

This is the declaration of an empty Array

Here we declare a variable and at the same time we initialize it with the keyword `null`

6.4 JAVASCRIPT: DATA TYPES

The JS possible **data types** are:

1. **PRIMITIVE TYPES**
   - **undefined**: the content of a variable with no data assigned;
   - **boolean**: true or false (1 or 0);
   - **number**: both for integers and floats (always 64 bits);
   - **string**: sequence of characters '..' or ".." or `..`;

2. **OBJECTS**
   - **Objects** of **javascript Classes**, like Array;
   - **function** (special types of Objects);
   - **Objects** defined by "**our" Classes** (defining a class is like defining a type of data).

6.4 JAVASCRIPT: DATA TYPES

```
let arrayOfDots = [];
```

An **Array** is a **data structure**, consisting of a collection of elements with an **INDEX number** and a **VALUE**;

```
let array_name = [ item1, item2, ... ];
```

You access an Array element by referring to the **INDEX number**.

`array_name[0]` contains the item1
`array_name[1]` contains the item2

6.5 JAVASCRIPT: FUNCTIONS

A `function` is a **sequence of instructions**, grouped in a block of code within curly brackets { }, created to **perform a certain task**.

Every function has a **NAME** and may require certain **PARAMETERS**.

A function can be **called (invoked) in any part of the code**, when we need to perform its task.

The value **returned** by the function (set with the keyword `return`) is the **response value** when the function is called (invoked).

If a function is a property of an Object it is called a **METHOD**.

6.5 JAVASCRIPT: FUNCTION

Example of a **function definition**:

```
function functionName(param1, param2){
    let in1 = param1;
    let in2 = param2;
    let output = in1 + in2;
    return output;
}
```

Example of **use of this function**:

```
let sum = functionName(1, 2);
```

This means that in the variable sum we are storing the number 3, the return value of the invoked function.

6.6 JAVASCRIPT: OBJECTS

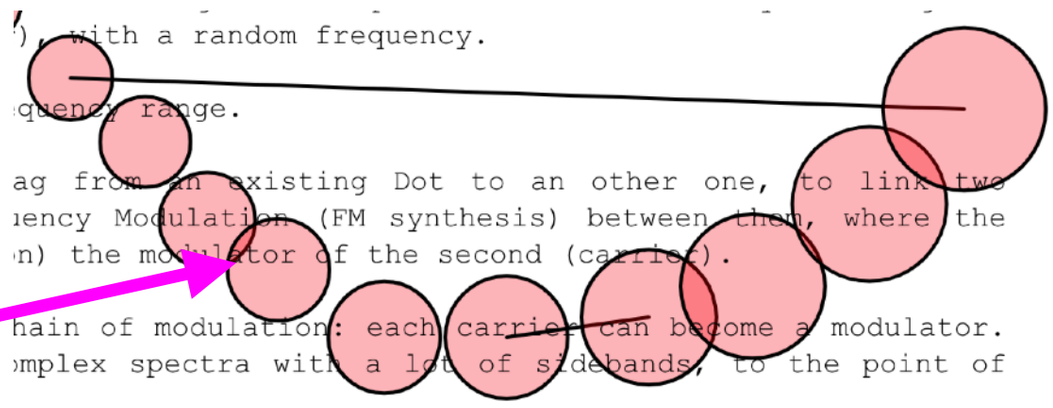
Object-oriented languages are based on **Objects**.

An Object is an **entity** with **properties** and **methods** that **models a real-world entity**.

*For example a car is an **object** with some **properties** (like color and weight) and some **methods** (like "start" or "stop" the engine).*

Here we need a "**sound entity**" called Dot, with some properties. In order to model this entities we create **Dot Objects**.

Each circle is a Dot Object, with some properties, like a specific frequency, an amplitude, a position (x-y) in the 2D space



6.6 JAVASCRIPT: OBJECTS

An **Object** groups a set of ***variables*** and ***functions*** to create a **model of a real-world** entity.

- the **variables of an Object** are called **PROPERTIES** (they tell us about the object);
- the **functions of an Object** are called **METHODS** (they are tasks/ actions associated with the Object).

Usually we want to create **several similar objects**, entities that have the same “structure" (properties and methods, called **fields** of an Object).

To create this **structure** we use the concept of **Class**.

6.6 JAVASCRIPT: OBJECTS

To access the **fields** (properties and methods) of an Object we use the **dot notation**.

The **dot " . "** is called **member operator**: the **property** or **method** to the **right of the dot** is a **member** of the **object** to the **left of the dot.**

We can use the member operator to **add**, **retrieve**, **modify** properties and methods of an Object.

For example, to fetch the property freq of an Object with name dot1 we write: `let frequency = dot1.freq;`

6.7 JAVASCRIPT: CLASS

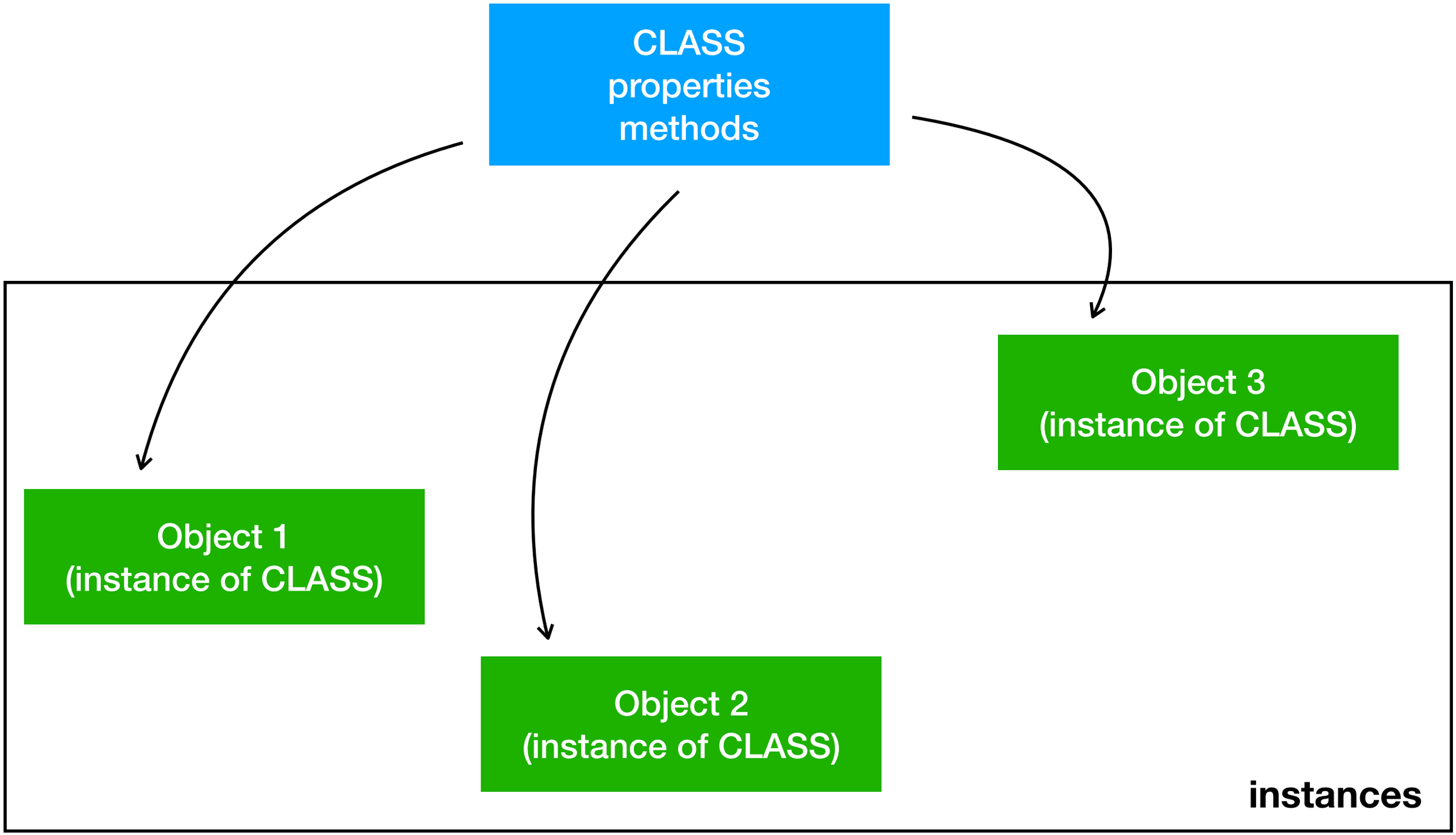
A **Class** defines the **structure** of the **Objects** of that class. A Class includes the definitions of **properties** and **methods** shared by the Objects of that Class.

The **Objects** of a certain **Class** are called **instances of that Class**. Each **instance** has the same structure but different current data.

Eg. two oscillators have a **property** `freq`, one `440.0`, the other `220.0`

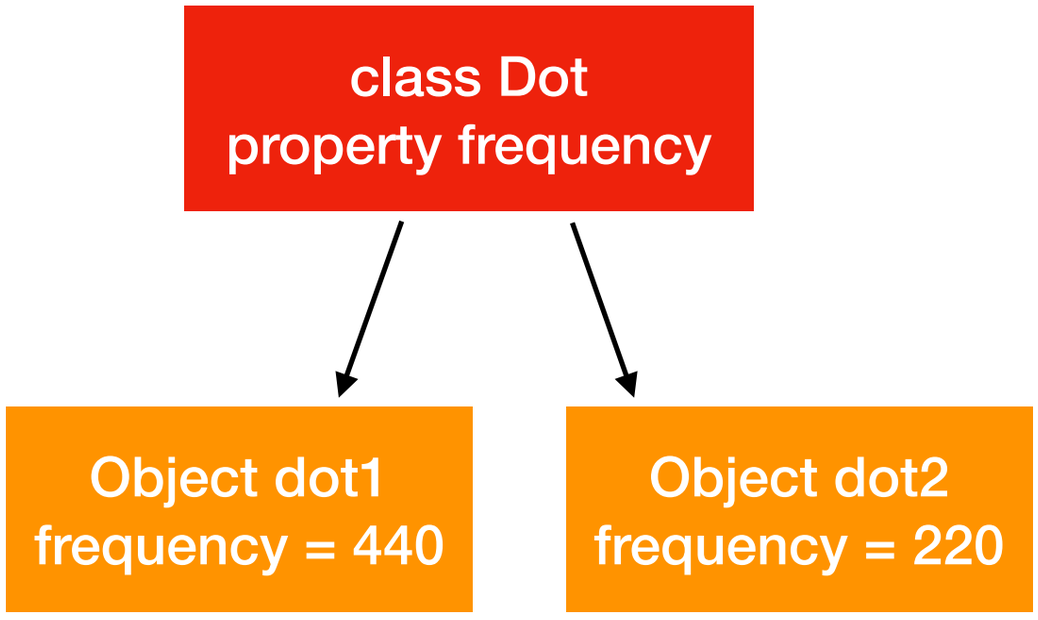
A **Class** is defined using the keyword `class`, followed by the name of the class in **capital letter**. A **Class** must have one `constructor()` method.

6.7 JAVASCRIPT: CLASS



6.8 JAVASCRIPT: `function startAudio()`

A Class is a sort of “custom data type”.



The **`constructor()`** method is a special method:

- it is **executed automatically** when an object of that Class is created using the keyword **`new`**;

- it is used to **initialize the Class properties** using parameters; e.g.: `constructor(par1, par2);`

6.8 let's code a bit

Let's open the file *webAudioWorkshop.js* *with Visual Studio Code*

```
/** ////////////////////////////////////////////
  * SUPPORT FUNCTIONS
    * definition
* ////////////////////////////////////////////////////////////////////// */

function startAudio(){ // audio to start when the first dot is created
    audioContext = new AudioContext(); // create an audioContext
    masterGain = audioContext.createGain(); // create a gain Node for the master gain
    masterGain.gain.value = MASTER_GAIN; // set the level of the master gain (gain adaptation)
    masterGain.connect(audioContext.destination); // connect the master gain to the destination
}
```

We implement this portion of code, defining a function that will start the audio rendering

We create an Object of Type `AudioContext`. The `new` keyword creates an Object of the Class `AudioContext`, defined in the web audio API.

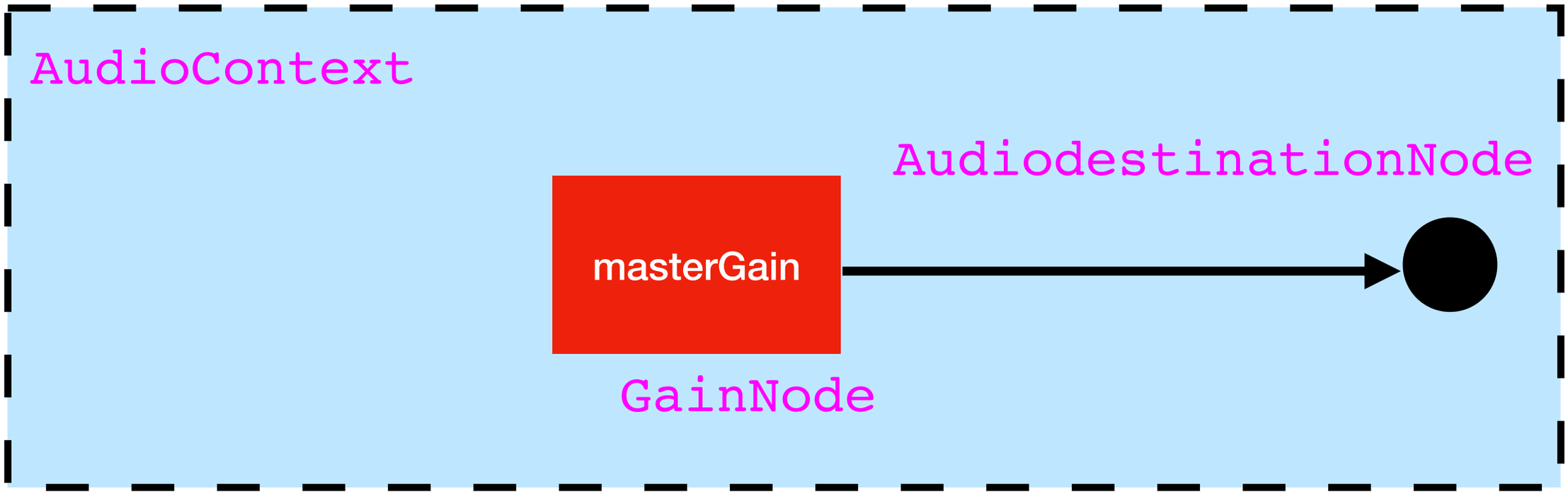
We create a masterGain Object with this syntax: `audioContext.createGain()`, that means that we are calling the method `createGain()`, that returns a Gain Object. `createGain()` is a method of the Class `AudioContext()` that returns a Gain Node.

Then we set the property `value` of the `gain` property of our masterGain assigning to it the MASTER_GAIN constant value already defined.

The `connect()` method is the "virtual connection" between Web Audio Nodes. Here we connect the Master Gain to the `destination`, a property of the `AudioContext` Class.

6.8 let's code a bit

This is the representation of what we created in the previous code, with the first connection we did



Let's open the file *webAudioWorkshop.js* with Visual Studio Code

Definition of the class Dot

the constructor has two parameters, the x and the y position of the center of the Dot in the canvas of the web page

```
/** ////////////////////////////////////////////////////////////
  * CLASS DOT
    * a Dot is an oscillator
* ////////////////////////////////////////////////////////////

class Dot{
    constructor(paramX, paramY){

        this.x = paramX; // x position of the Dot (associated to frequency in Hz)
        this.y = paramY; // y position of the Dot (associated to amplitude)

        /** array of the possible connections to a
         * connections[0] is the possible Dot obje
         * connections[1] is the possible Dot obje
         */
        this.connections = [null, null];

        // define amplitude and frequency of the associated sinusoid
        this.freq = map(this.x, 0, windowWidth, MIN_FREQUENCY_HZ, MAX_FREQUENCY_HZ);
        this.amp = map(this.y, windowHeight - CANVAS_OFFSET, 0, MIN_AMPLITUDE, MAX_AMPLITUDE);

        // set the dimension of the circle associated to this Dot
        this.dimension = map(this.freq, MAX_FREQUENCY_HZ, MIN_FREQUENCY_HZ, MAX_DIAMETER, MIN_DIAMETER);
```

`this` is a keyword that refers to the Object that will be instantiated by the constructor of this class

y is the name of a property; and we assign to it the data stored in the variable paramY passed through the constructor when the Object is created

This is an `Array` of two values (that we set to `null` at the beginning); this will be used to store the modulator (an Object of Class Dot, in the position at index 0) and the carrier (an Object of Class Dot, in the position of index 1) for the FM

the map method is a method of the library p5.js. It allows us to remap a value from one range to another

Let's open the file *webAudioWorkshop.js* with Visual Studio Code

Here we create the main oscillator associated to the Dot. We create a SourceNode of the Web Audio API using the method `createOscillator();`

then we set the **type**, the **frequency** and we start it calling the method `start()`. the keyword `currentTime` contains the current time of the scheduler (a sort of clocker).

`audiocontext.currentTime` tells to the `start()` method when we want the oscillator to start. `currentTime` is the time the scheduler of the web audio API is in that moment.

```
class Dot{
    constructor(paramX, paramY){

        this.x = paramX; // x posi
        this.y = paramY; // y posi

        /** array of the possible co
         * connections[0] is the pos
         * connections[1] is the pos
         */
        this.connections = [null, nu

        // define amplitude and frequency of the associated sinusoid
        this.freq = map(this.x, 0, windowWidth, MIN_FREQUENCY_HZ, MAX_FREQUENCY_HZ);
        this.amp = map(this.y, windowHeight - CANVAS_OFFSET, 0, MIN_AMPLITUDE, MAX_AMPLITUDE);

        // set the dimension of the circle associated to this Dot
        this.dimension = map(this.freq, MAX_FREQUENCY_HZ, MIN_FREQUE

        // the main oscillator (sinusoid)
        this.mainOsc = audioContext.createOscillator(); // create th
        this.mainOsc.type = "sine"; // definte the type
        this.mainOsc.frequency.value = this.freq; // set the frequen
        this.mainOsc.start(audioContext.currentTime); // start the o

        // the amplitude oscillator (triangle) for AM
        this.AMosc = audioContext.createOscillator(); // create the oscillator Node
        this.AMosc.type = "triangle"; // definte the type
        this.AMosc.frequency.value = random(MAX_AM_FREQ) + MIN_AM_FREQ; // set the frequency
        this.AMosc.start(audioContext.currentTime); // start the oscillator
```

Here we create the triangular oscillator that we will use for the AM (tremolo). We define a random frequency using the method `random()` of the p5.js library.

```
// the main oscillator (sinusoid)
this.mainOsc = audioContext.createOscillator(); // create the oscillator Node
this.mainOsc.type = "sine"; // definte the type
this.mainOsc.frequency.value = this.freq; // set the frequency
this.mainOsc.start(audioContext.currentTime); // start the oscillator

// the amplitude oscillator (triangle) for AM
this.AMosc = audioContext.createOscillator(); // create the oscillator Node
this.AMosc.type = "triangle"; // definte the type
this.AMosc.frequency.value = random(MAX_AM_FREQ) + MIN_AM_FREQ; // set the frequency
this.AMosc.start(audioContext.currentTime); // start the oscillator

// create the gain Nodes
this.mainOscGain = audioContext.createGain(); // gain Node for the main Oscillator
this.mainOscGain.gain.value = 0; // set the initial gain to 0

this.AMoscGain = audioContext.createGain(); // gain Node for the AM Oscillator
this.AMoscGain.gain.value = 0; // set the initial gain to 0; it will be from 0 to 2*this.amp (-1 1)*this.amp

this.FMoscGain = audioContext.createGain(); // gain Node for the FM Oscillator (if it is used as modulator) : this is the modulation INDEX
this.FMoscGain.gain.value = 0; // set the initial gain to 0
```

Each `SourceNode` needs a `gainNode` to be used.

Here we create some `gainNode`;

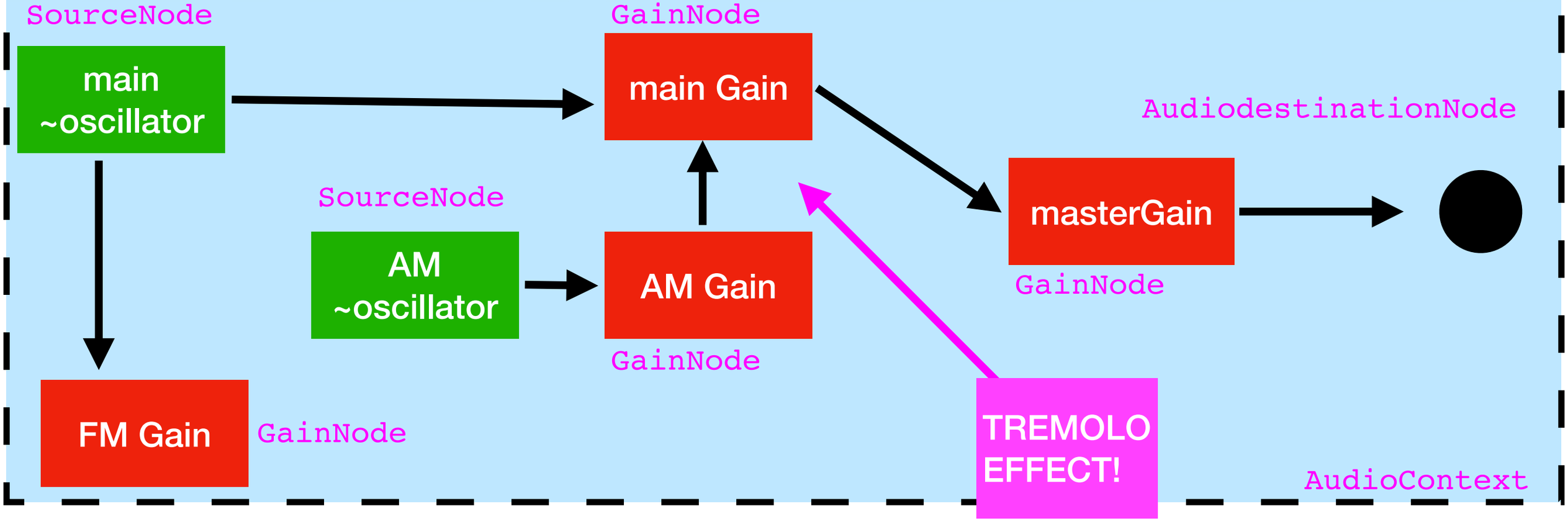
- **the gain for the main oscillator;**
- **the gain for the AM (tremolo) oscillator;**
- **the gain for the FM: the INDEX of MODULATION that we use if this Dot is used as a modulator.**

After this, we set the property `value` of the property `gain` of the Gain Node at 0.

```
        // the connections
        this.mainOsc.connect(this.mainOscGain); // connect the main oscillator to its gain
        this.mainOscGain.connect(masterGain); // connect the main oscillator to the master gain
        this.AMosc.connect(this.AMoscGain); // connect the AM oscillator to its gain
        this.AMoscGain.connect(this.mainOscGain.gain); // connect the AM oscillator gain to the main oscillator gain (amplitude modulation)
        this.mainOsc.connect(this.FMoscGain); // connect the main Osc to the FM oscillator gain (in the case this will be used as modulator)

        // set the main oscillator gain and the AM oscillator gain to to the amplitude in a certain attack time
    }
}
```

Here we connect everything

- the main oscillator is connected to the main Gain
- the main Gain is connected to the master Gain
- the AM oscillator is connected to the AM Gain
- the AM Gain is connected to the property `gain` of the main Gain (this is the AM - tremolo)
- the main oscillator is connected also to the FM osc Gain (the index of modulation)



```
        // the connections
        this.mainOsc.connect(this.mainOscGain); // connect the main oscillator to its gain
        this.mainOscGain.connect(masterGain); // connect the main oscillator to the master gain
        this.AMosc.connect(this.AMoscGain); // connect the AM oscillator to its gain
        this.AMoscGain.connect(this.mainOscGain.gain); // connect the AM oscillator gain to the main oscillator gain (amplitude modulation)
        this.mainOsc.connect(this.FMoscGain); // connect the main Osc to the FM oscillator gain (in the case this will be used as modulator)

        // set the main oscillator gain and the AM oscillator gain to to the amplitude in a certain attack time
        this.mainOscGain.gain.setTargetAtTime(this.amp, audioContext.currentTime, ATTACK_TIME);
        this.AMoscGain.gain.setTargetAtTime(this.amp, audioContext.currentTime, ATTACK_TIME);
    }
}
```

Here we use the method `setTargetAtTime` to create an exponential ramp that goes from 0 (`gain.value` is zero) to the desired amplitude (`this.amp`). We use this to set both `mainOscGain` and `AMoscGain`.

The larger the ATTACK_TIME, the slower the transition.

Here we set both the `mainOscGain` and the `AMoscGain` to `this.amp`.

Example: let's say that `this.amp = 0.2`. This means that both `mainOscGain` and `AMoscGain` have an amplitude of 0.2.

When we modulate `mainOscGain` with `AMoscGain` we change in a periodic triangular way the final amplitude of the mainOscGain from 0 (0.2-0.2) to 0.4 (0.2+0.2).

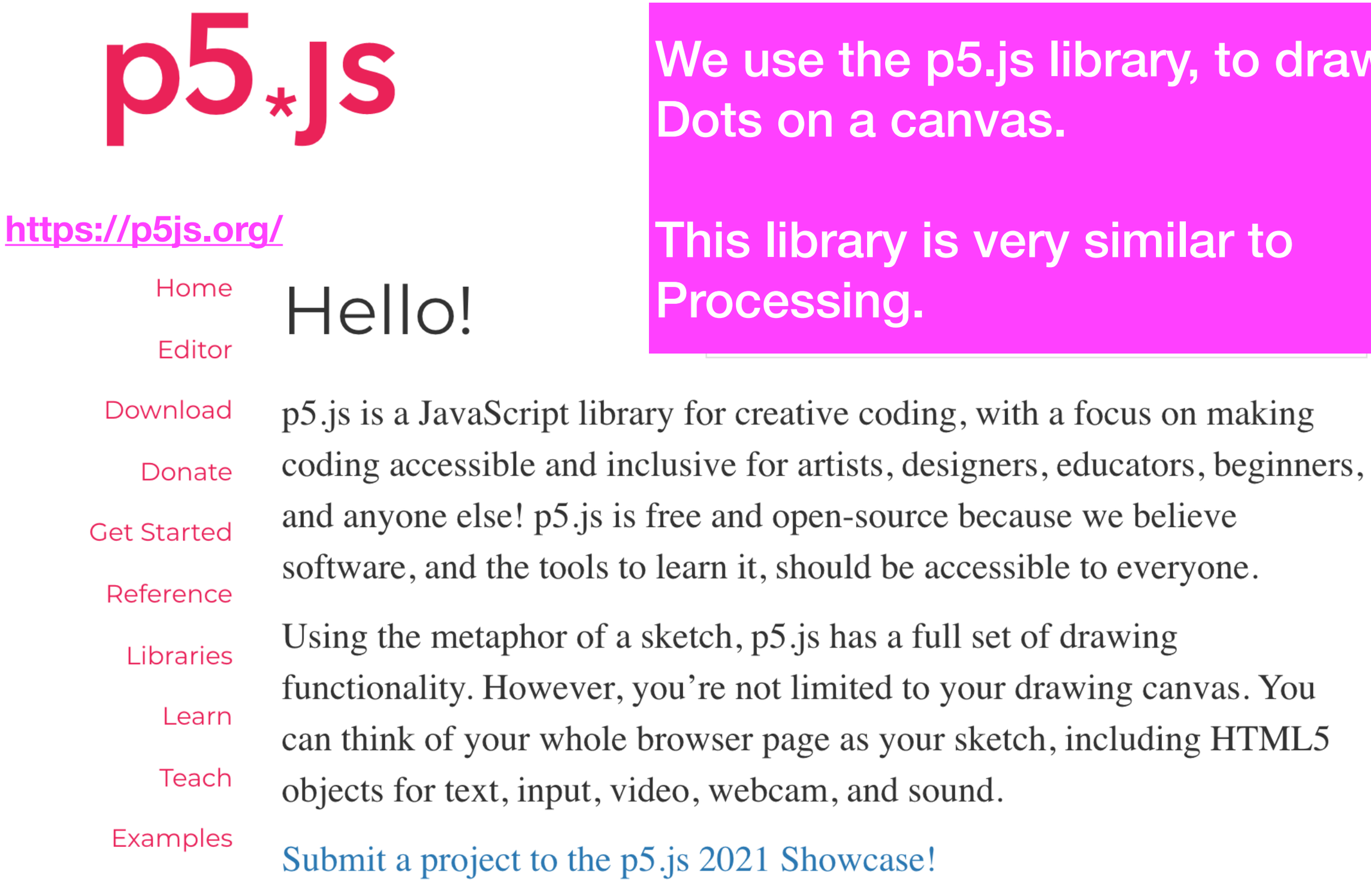
6.9 p5.js

p5.js

https://p5js.org/

We use the p5.js library, to draw Dots on a canvas.

This library is very similar to Processing.



```
/** ////////////////////////////////////////////////////////////////////
* P5.JS FUNCTIONS
  * definition
* //////////////////////////////////////////////////////////////////// */

function setup() { // create the canvas in the windows with an offset
  createCanvas(windowWidth, windowHeight - CANVAS_OFFSET);
}

function windowResized() { // resize the canvas if the user changes the window size
  resizeCanvas(windowWidth, windowHeight - CANVAS_OFFSET);
}
```

Here we implement 2 functions of the p5.js library.

- The `setup` function is called at the beginning when the code is loaded. In this function we create a canvas, that is a white space where we can draw shapes, textes, lines, etc..
- The `windowResized` function is called when the user resize the window. In this case we want to resize the canvas accordingly.

`windowWidth`: system variable that stores the width of the inner window

`windowHeight`: system variable that stores the height of the inner window

6.9 CONDITIONAL STATEMENTS

JS supports the **conditional statement `if .... else`**

```
if(condition){
   // block of code to execute if the condition is true
} else if (otherCondition){
   // block of code to execute if the other condition is true
} else {
   // block of code to execute if both conditions are false
}
```

A condition can be a **boolean** or the result of a **boolean expression**;

```
let flag = true; // true
let expression = 1 > 0; // true
```

6.10 BOOLEAN

A **Boolean expression** is an expression used in programming that produces a Boolean value when evaluated.

A Boolean value is `true` (1) or `false` (0).

The most common **Boolean operator** are:

- NOT (!)
- OR (||)
- AND (&&)

The Boolean algebra follows the **Truth Table**.

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Here we implement the function `mousePressed()` that is invoked any time the user presses the mouse (`mouseX` and `mouseY` contain the mouse position on the canvas). These are the tasks that we implement:

- if the `audioContext` is still undefined we want to call our function `startAudio()` to create the `audioContext`;
- if the user press the mouse out of the canvas (this means that the y is negative) or is pressing 'L', we want to exit from the function (we do this with the keyword `return`);
- we create a new Object of the class `Dot` (using the mouse position as the center of the Dot)

```
function mousePressed() { // function called if the mouse is pressed
    if(!audioContext){ startAudio(); } // if the audio context still does not exists create it
    if(mouseY < 0 || keyIsDown(76)) return; // exit if the mouse is out of the canvas OR user is pressing "L"

    let newDot = new Dot(mouseX,mouseY); // create the Dot calling the constructor of the class
    arrayOfDots.push(newDot); // put the Dot in the array of Dots
}
```

`push()` is a method of the Class `Array` that we use to insert the new Dot just created in the Array of Dots.

`new` invokes the constructor of the Dot class that we have already defined. Here we create an Object of our Class!

`return;` allows us to exit from this portion of code (from the function mousePressed) under certain conditions.

6.11 `for` LOOP

Cycles provide a way to perform operations repeatedly, over a number of times. JS supports the **standard loops**:

The most important is perhaps the **for loop**.

```
for (statement 1; statement 2; statement 3){
    // block to be executed
}
```

- `statement 1`: it is executed once before the code execution
- `statement 2`: it defines the conditions to execute the block of code
- `statement 3`: it is executed every time after the code has been executed

```
function draw(){ // this is the draw function of the p5.js calle dto draw on the canvas
  clear(); // clear the canvas
  fill('rgba(255,0,255,0.3)'); // the color of the Dot: pink with alpha
  strokeWeight(2); // the weight of the border of the elements

  for(let i=0; i<arrayOfDots.length; i++){ // draw each Dot in the array
      let currentDot = arrayOfDots[i];

      ellipse(currentDot.x, currentDot.y, currentDot.dimension, currentDot.dimension); // draw a circle
  }
}
```

Here we implement the function `draw()` of p5.js.

It is called directly after `setup()`.

The `draw()` function continuously executes the lines of code contained inside its block until the program is stopped. The number of times `draw()` executes in each second may be controlled with the `frameRate()` function.

The default frame rate is based on the frame rate of the display (here also called "refresh rate"), which is set to 60 frames per second on most computers.

A frame rate of 24 frames per second (usual for movies) or above will be enough for smooth animations.

```
function draw(){ // this is the draw function of the p5.js calle dto draw on the canvas
  clear(); // clear the canvas
  fill('rgba(255,0,255,0.3)'); // the color of the Dot: pink with alpha
  strokeWeight(2); // the weight of the border of the elements

  for(let i=0; i<arrayOfDots.length; i++){ // draw each Dot in the array
      let currentDot = arrayOfDots[i];

      ellipse(currentDot.x, currentDot.y, currentDot.dimension, currentDot.dimension); // draw a circle
  }
}
```

`length` is a property of Arrays. It returns the number of items stored into an array. In this case we process every object in `ArrayOfDots`.

The function `clear()` deletes the content of the canvas.
The function `fill()` sets the color of each element (this i a red with alpha 0.3).
The function `strokeWeight()` sets the dimension of the borders of the elements.

Then we create a for loop that is executed at each frame (so let's say 60 times for second).

The code is executed in the for loop as many times as there are elements in the `arrayOfDots`. We use the index `i` of the loop to fetch the current Object using the syntax of the Array `arrayOfDots[i]`.

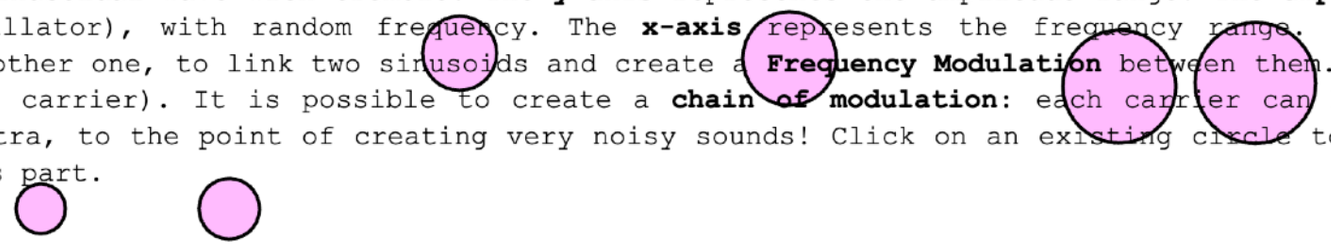
At each cycle we draw a circle using the function `ellipse()` where the two axes have the same dimension.

LET'S DRONE!!!!

Dot Drone Generator

highSCORE Festival 2023 | Alberto Barberis

Dot **Drone Generator** is a drone generator which allows you to create **synthetic textures** directly on your browser. Click on the window to generate a **Dot**, a sinusoidal wave with tremolo. The **y-axis** represents the amplitude range. The **amplitude** is modulated by a **triangular LFO** (Low Frequency Oscillator), with random frequency. The **x-axis** represents the frequency range. Press 'L' and then Click+Drag from an existing Dot to an other one, to link two sinusoids and create a **Frequency Modulation** between them. The first Dot becomes the modulator of the second one (the carrier). It is possible to create a **chain of modulation**: each carrier can become a modulator. This allows you to create complex spectra, to the point of creating very noisy sounds! Click on an existing circle to **delete** it or to delete the modulation chain of which it is part.



6.12 JAVASCRIPT: FM MODULATION

In our list of global variables we created the couple `modulator` and `carrier` that we set to `null`. The goal is to store in these variables always a different pair of modulator/carrier, every time the user click+Drag+L on a couple of Dots.

These variables will contain the modulator (the Dot Object that is selected first when the user type click+Drag+L) and the carrier (the second Dot Object over which the mouse is passed while click+Drag+L).

We create a new function `deleteModulatorCarrier()` that we use to empty the modulator and the carrier variables after that a couple (modulator + carrier) has been found.

```
/** ////////////////////////////////////////////////////////////
  * SUPPORT FUNCTIONS
    * definition
* //////////////////////////////////////////////////////////// */

function startAudio(){ // audio to start when the first dot is created
    audioContext = new AudioContext(); // create an audiocontext
    masterGain = audioContext.createGain(); // create a gian Node for the master gain
    masterGain.gain.value = MASTER_GAIN; // set the level of the master gain (gain adaptation)
    masterGain.connect(audioContext.destination); // connect the master gain to the destination
}

function deleteModulatorCarrier(){
    modulator = null; // remove the object from the variable modulator
    carrier = null; // remove the object from the variable carrier
}
```

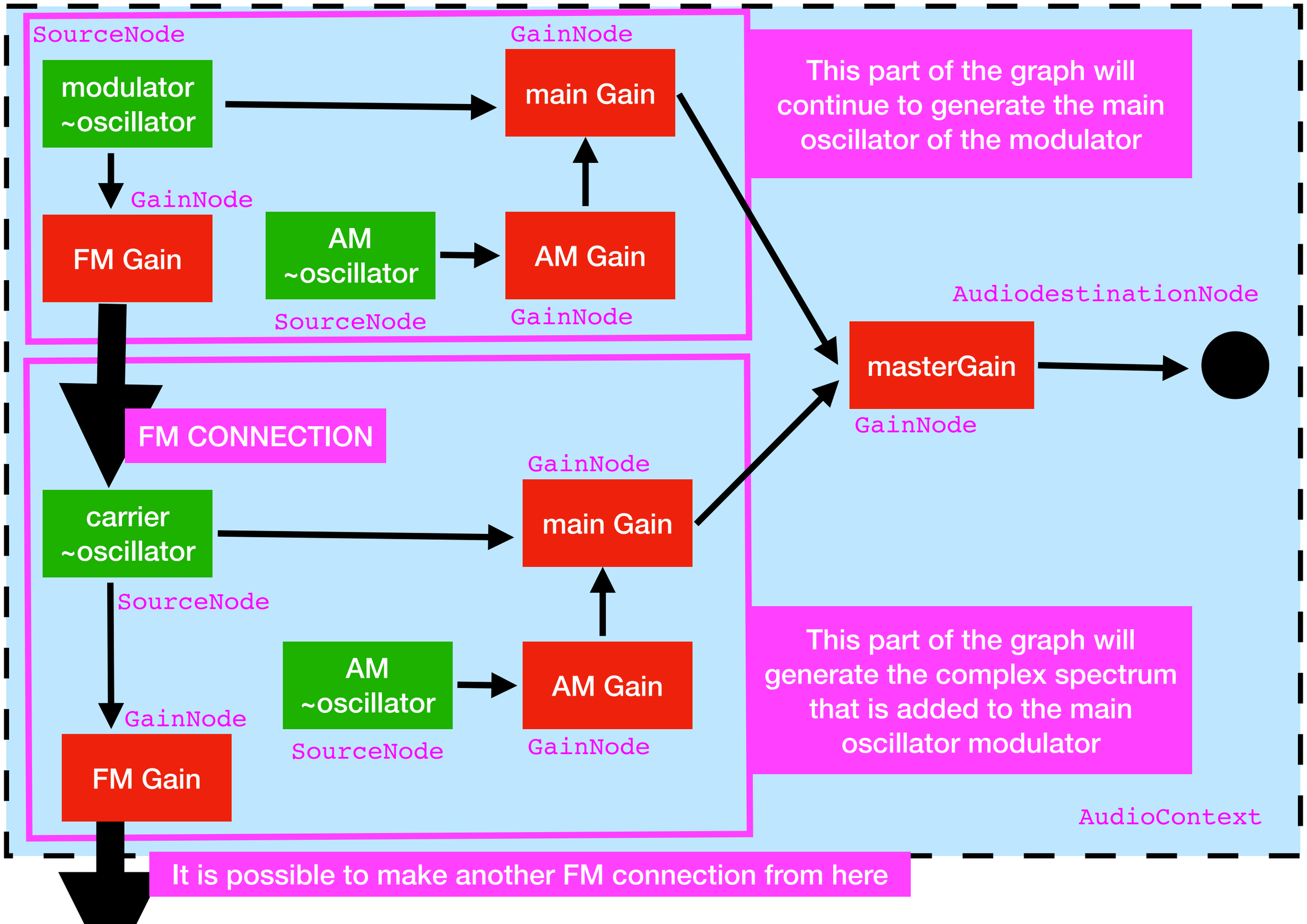
6.12 JAVASCRIPT: FM MODULATION

We create the function `applyFM()` that we use for the FM synthesis between the modulator and the carrier (the two parameters of the function).

To implement the FM we have to connect the **FM oscillator Gain** to the **carrier main oscillator frequency property.**

Then we start the FM oscillator Gain (initially set to zero) setting it to the MODULATION_INDEX constant (the value that controls the amount of sidebands and their amplitudes).

```
/** ////////////////////////////////////////////////////////////////
  * SUPPORT FUNCTIONS
    * definition
* //////////////////////////////////////////////////////////////// */

function startAudio(){ // audio to start when the first dot is created
  audioContext = new AudioContext(); // create an audiocontext
  masterGain = audioContext.createGain(); // create a gian Node for the master gain
  masterGain.gain.value = MASTER_GAIN; // set the level of the master gain (gain adaptation)
  masterGain.connect(audioContext.destination); // connect the master gain to the destination
}

function deleteModulatorCarrier(){
  modulator = null; // remove the object from the variable modulator
  carrier = null; // remove the object from the variable carrier
}

function applyFM(modulator, carrier){ // function that activate the FM synthesis
  modulator.FMoscGain.connect(carrier.mainOsc.frequency); // connect the FMoscGain to the frequency of the carrier main Oscillator (this is the FM)
  modulator.FMoscGain.gain.setTargetAtTime(MODULATION_INDEX, audioContext.currentTime , ATTACK_TIME); // set the FMoscGain to the modulation index value
}
```


SourceNode
modulator ~oscillator
GainNode
main Gain
This part of the graph will continue to generate the main oscillator of the modulator
GainNode
FM Gain
AM ~oscillator
AM Gain
SourceNode
GainNode
AudiodestinationNode
masterGain
GainNode
FM CONNECTION
GainNode
main Gain
carrier ~oscillator
SourceNode
AM ~oscillator
AM Gain
This part of the graph will generate the complex spectrum that is added to the main oscillator modulator
GainNode
FM Gain
SourceNode
GainNode
AudioContext
It is possible to make another FM connection from here

6.12 JAVASCRIPT: FM MODULATION

Here we implement another function of the p5.js library.

The `mouseDragged()` function is called every time the user click + drag the mouse.

In the first line of this function we use an other function of the p5.js the `keyIsDown()`.

`keyIsDown()` returns `true` if the user type a certain letter in the keyboard; the letter is passed as keyCode parameter (we use the “L” for “Link” that has 76 as keyCode).

If the keyboard letter is not the “L” we exit from the function. The “!” means NOT.

```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
    if(!keyIsDown(76)) return; // exit from the function if is not pressed the key 'L'
```

https://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes

Here we create a `for` **loop** that for each Dot in the **Array of Dots** calculate the distance between the current x and y positions of the mouse from the x and y positions of the Dot center.

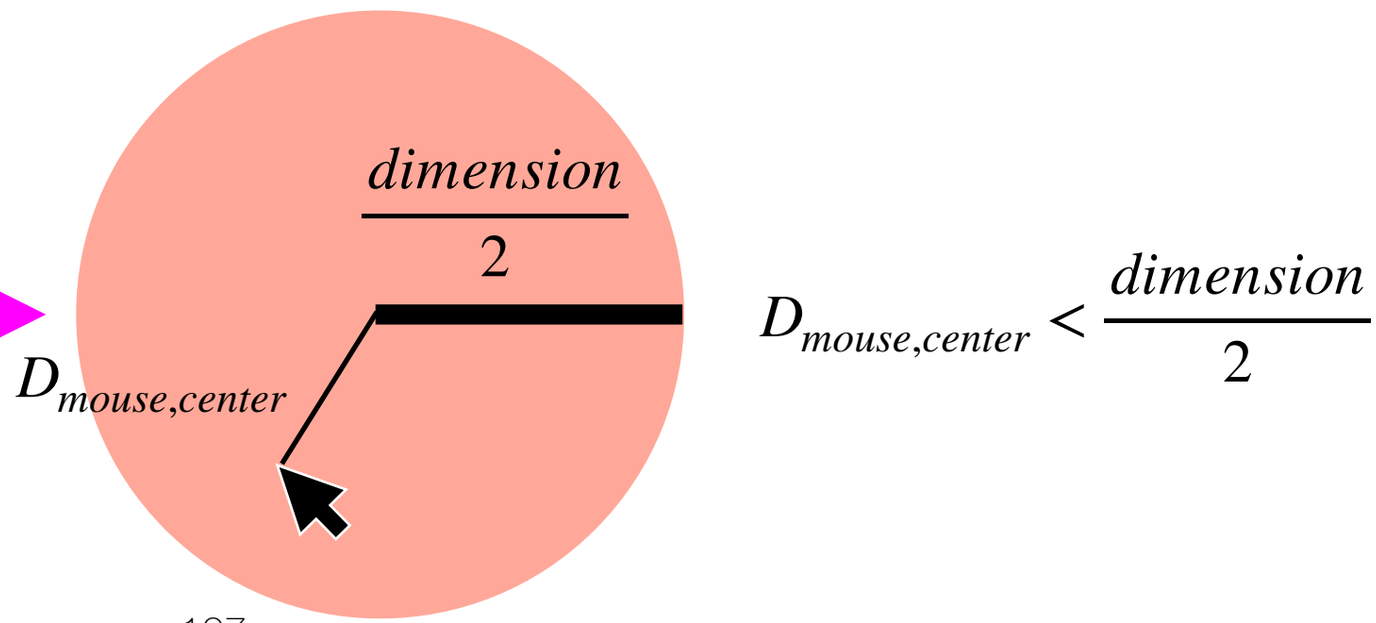
If the distance is less than the dimension/2 of the Dot (i.e. the radius of the circle) we can say that the mouse position is inside a certain Dot.

```
function mouseDragged() { // function called if the mouse is dragge
  if(!keyIsDown(76)) return; // exit from the function if is not pr

  for(let i=0; i<arrayOfDots.length; i++){
      let currentDot = arrayOfDots[i];
      let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the Dot center and the mouse
```

Here we fetch the current Dot from the Array of Dots using the syntax of the Array `arrayOfDots[i]`

Here we calculate the distance between the current Dot center and the current mouse position and we save it in a local variable. `dist()` is a function of the p5.js library

We can say that the mouse is INSIDE the Dot because the distance between the mouse and the center is less than the radius (dimension / 2)



```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
  if(!keyIsDown(76)) return; // exit from the function if is not pressed the key 'L'

  for(let i=0; i<arrayOfDots.length; i++){
      let currentDot = arrayOfDots[i];
      let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the current Dot center and the mouse
position
      /**
       * if there is not a modulator and not a carrier
       * and the mouse is on a Dot
       * and that Dot does not already have a carrier;
       *      set the Dot as a modulator
       */
      if(!modulator && !carrier && d<currentDot.dimension/2 && !currentDot.connections[1] ) {
          modulator = currentDot;
          break; // exit from the for loop
      }
```

`break` is a way to exit from a for loop. In this case if we have found a modulator we can exit.

Here we check the following conditions with one boolean expression:

- the modulator does not exist (`null` is `false` in javascript)
- the carrier does not exist
- the distance is less than the radius
- and the current Dot does not already have a carrier (we fetch the second Object in the Array connections)

If these conditions are verified then: **we have found a modulator!** And we store it in the right variable. We exit also from the current loop cycle, because if the Dot is a modulator it can not be a carrier and it does not make sense to proceed.

```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
  if(!keyIsD
  for(let i=
      let cu
      let d =
position
      /**
       * if
       * and
       * and
       *
       */
      if(!mo
          mo
          br
      }
      /**
       * else if there is a modulator but not a carrier
       * and the mouse is on a Dot
       * and that Dot does not already have a modulator
       * and the current index of the loop is not the one of the modulator
       *      set the Dot as a carrier
       *      assign the carrier to the modulator
       *      assign the modulator to the carrier
       *      and apply the FM
       */
      else if(modulator && !carrier && d<currentDot.dimension/2 && !currentDot.connections[0] && arrayOfDots.indexOf(modulator)!=i){
          carrier = currentDot;
          modulator.connections[1] = carrier;
          carrier.connections[0] = modulator;
          applyFM(modulator, carrier);
          break; // exit from the for loop
      }
  }
}
```

Here we check the following conditions with one boolean expression:

- the modulator exists
- the carrier does not exist
- the distance is less than the radius
- and the current Dot does not already have a modulator (we fetch the first Object in the Array connections)
- and the index of the already found modulator in the arrayOfDots is not the index of the current iteration (otherwise we would find that the modulator is also the carrier)

```
function mouseDragged() { // function called if the mouse is dragged (for the FM synthesis)
  if(!keyIsDown(76)) return; // exit from the function if is not pressed the key 'L'

  for(let i=0; i<arrayOfDots.length; i++){
      let currentDot = arrayOfDots[i];
      let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the current Dot center and the mouse
position
      /**
       * if there is not a m
       * and the mouse is on
       * and that Dot does n
       *      set the Dot as
       */
      if(!modulator && !carr
          modulator = curren
          break; // exit fro
      }
      /**
       * else if there is a modulator but not a carrier
       * and the mouse is on a Dot
       * and that Dot does not already have a modulator
       * and the current index of the loop is not the one of the modulator
       *      set the Dot as a carrier
       *      assign the carrier to the modulator
       *      assign the modulator to the carrier
       *      and apply the FM
       */
      else if(modulator && !carrier && d<currentDot.dimension/2 && !currentDot.connections[0] && arrayOfDots.indexOf(modulator)!=i){
          carrier = currentDot;
          modulator.connections[1] = carrier;
          carrier.connections[0] = modulator;
          applyFM(modulator, carrier);
          break; // exit from the for loop
      }
  }
}
```

If the conditions are all verified, it means that **we have found a carrier!**

We store the Dot Object in the variable.

We say to the modulator what Dot Object is the carrier and viceversa.

Then we apply the FM calling the function we already defined and passing it, as parameters, the **modulator** and the **carrier** that we have found.

```
// when the mouse is released or a key is released delete the stored modulator and carrier
function mouseRealeased() {
    deleteModulatorCarrier();
}
function keyReleased(){
    deleteModulatorCarrier();
}
```

Now we implement other two functions of p5.js. Both of these functions call the function `deleteModulatorCarrier()` that we already implemented.

In this way every time the user releases the mouse, `mouseReleased()`, or releases the key “L” `keyReleased()`, the process of finding modulator and carrier will stop, emptying the variables and making them available for a new search process.

```
function draw(){ // this is the draw function of the p5.js called to draw on the canvas
    clear(); // clear the canvas
    fill('rgba(255,0,255,0.3)'); // the color of the Dot: red with alpha
    strokeWeight(2); // the weight of the border of the elements

    for(let i=0; i<arrayOfDots.length; i++){ // draw each Dot in the array
        let currentDot = arrayOfDots[i];

        ellipse(currentDot.x, currentDot.y, currentDot.dimension, currentDot.dimension); // draw a circle

        let carrier = currentDot.connections[1]; // the carrier of the current Dot
        if(carrier){ // if the carrier exists draw a line between the modulator (current Dot) and the carrier
            line(currentDot.x, currentDot.y, carrier.x, carrier.y);// draw the line
        }
    }
}
```

Now we go back to the `draw()` function.

Here we check if the current Dot in the loop cycle has a carrier, that means that the Object inside `connections[1]` is not `null`.

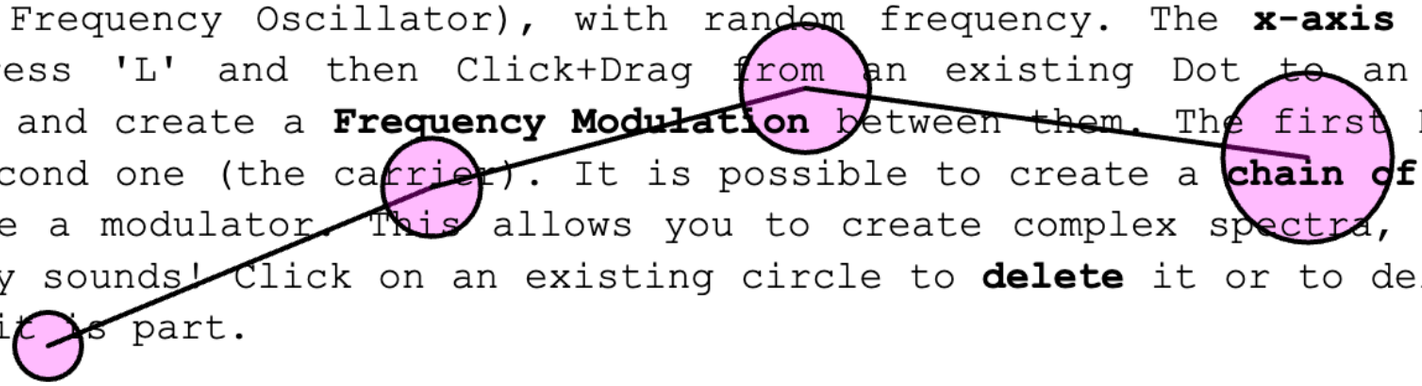
If so, we want to draw a line from the current Dot center to the carrier center. We do this with the p5.js function `line()`.

Now we can start to create some FM-paths in the 2D space of Dots.

Dot Drone Generator

highSCORE Festival 2023 | Alberto Barberis

Dot **Drone Generator** is a drone generator which allows you to create **synthetic textures** directly on your browser. Click on the window to generate a **Dot**, a sinusoidal wave with tremolo. The **y-axis** represents the amplitude range. The **amplitude** is modulated by a **triangular LFO** (Low Frequency Oscillator), with random frequency. The **x-axis** represents the frequency range. Press 'L' and then Click+Drag from an existing Dot to an other one, to link two sinusoids and create a **Frequency Modulation** between them. The first Dot becomes the modulator of the second one (the carrier). It is possible to create a **chain of modulation**: each carrier can become a modulator. This allows you to create complex spectra, to the point of creating very noisy sounds! Click on an existing circle to **delete** it or to delete the modulation chain of which it is part.



Here we implement a function that delete a Dot and all the Dots possibly connected with it (FM chain)

```
function deleteDot(currentDot){ // this is a recursive function that delete a dot from the array
    let indexToDelete = arrayOfDots.indexOf(currentDot); // fetch the index of the Object
    if(indexToDelete == -1) return;

    currentDot.mainOscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the main osc Gain
    currentDot.AMoscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the AM osc Gain
```

This is a RECURSIVE FUNCTION, that means a function that can call itself.

The method `indexOf()` returns the index of the Current Dot. We need this later to delete the element from the Array.

Then we set to 0 the main oscillator Gain and the AM oscillator Gain. This action stops the sound of the Dot.

```
function deleteDot(currentDot){ // this is a recursive function that delete a dot from the array
    let indexToDelete = arrayOfDots.indexOf(currentDot); // fetch the index of the Object
    if(indexToDelete == -1) return;

    currentDot.mainOscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the main osc Gain
    currentDot.AMoscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the AM osc Gain

    let modulator = currentDot.connections[0]; // store in a variable the modulator associated to the Object
    let carrier = currentDot.connections[1]; // store in a variable the carrier associated to the Object

    arrayOfDots.splice(indexToDelete, 1); // remove the Object from the array
```

We fetch the modulator and the carrier that are (possibly) connected to the current Dot.

The method `splice()` of the class `Array` can be used to delete an element from an `Array`.

The first parameter is the index of the Array from where delete and the second parameter specifies how many elements we want to delete. In our case we just want to delete one element, the current Dot itself.

```
splice(start, deleteCount)
```

```
function deleteDot(currentDot){ // this is a recursive function that delete a dot from the array
    let indexToDelete = arrayOfDots.indexOf(currentDot); // fetch the index of the Object
    if(indexToDelete == -1) return;

    currentDot.mainOscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the main osc Gain
    currentDot.AMoscGain.gain.setTargetAtTime(0, audioContext.currentTime, RELEASE_TIME); // set to 0 the AM osc Gain

    let modulator = currentDot.connections[0]; // store in a variable the modulator associated to the Object
    let carrier = currentDot.connections[1]; // store in a variable the carrier associated to the Object

    arrayOfDots.splice(indexToDelete, 1); // remove the Object from the array

  if(modulator){ // if the modulator exists
    deleteDot(modulator); // delete it with a recursion
  }

  if(carrier){ // if the carrier exists
    deleteDot(carrier); // delete it with a recursion
  }
}
```

We want to delete also all the Dots connected to the current Dot. We do the following using the RECURSION of the same deleteDot function:

- if the current Dot has a modulator, delete it;
- if the current Dot has a carrier, delete it;

```
function mousePressed() { // function called if the mouse is pressed
    if(!audioContext){ startAudio(); } // if the audio context still does not exists create it
    if(mouseY < 0 || keyIsDown(76)) return; // exit if the mouse is out of the canvas OR the user is pressing "L"

    for(let i=0; i < arrayOfDots.length; i++){ // check if the mouse is pressed in an existing Dot; if so delete it

        let currentDot = arrayOfDots[i];
        let d = dist(currentDot.x, currentDot.y, mouseX, mouseY); // calculate the distance between the Dot and the mouse

        if(d < currentDot.dimension/2){ // if pressing on an existin
            deleteDot(currentDot); // delete the Dot
            return;
        }
    }

    let newDot = new Dot(mouseX,mouseY); // create the Dot calling t
    arrayOfDots.push(newDot); // put the Dot in the array of Dots
}
```

Every time the user press the mouse, we check with a for loop if the mouse is inside a Dot that already exists. We do this calculating the distance between the Dot center and the mouse position.

We implement the code of the `mousePressed()` function.

We want to delete an existing Dot when it is pressed, and also to delete all the Dots connected to it, if there are any.

If the distance is less than the radius we delete the current Dot, calling our function `deleteDot()`.

If we delete a Dot we exit from the function with the keyword `return`. This means that the following lines are not executed, and so we do not create a new Dot.

Dot Drone Error Generator FM

Alberto Barberis

The Dot Drone Error Generator FM allows you to create a synthetic texture, possibly loosing its control.

~~CLICK~~
on the window to generate a Dot, a sinusoidal wave with tremolo. The y-axis represents the amplitude range.
The amplitude is modulated by a triangular LFO, with random frequency. The x-axis represents the frequency range.

Press
~~L~~
and then Click+Drag from an existing Dot to an other one. FM occurs.
The first Dot becomes the modulator of the second (the carrier).

It is possible to create a chain of modulations and a
~~FEEDBACK~~
chain too!

Press
~~M~~
and then Click+Drag from an existing Dot to move it.

Press
~~R~~
into a Dot to randomize its movement. Closer to the border faster the movement.

~~CLICK~~
on an existing circle to
~~DELETE~~
it or to delete the modulation chain.

If you randomize the movements, Dots can disappear forever.
This is an
~~ERROR~~
.

~~PLAY~~
WITH IT.