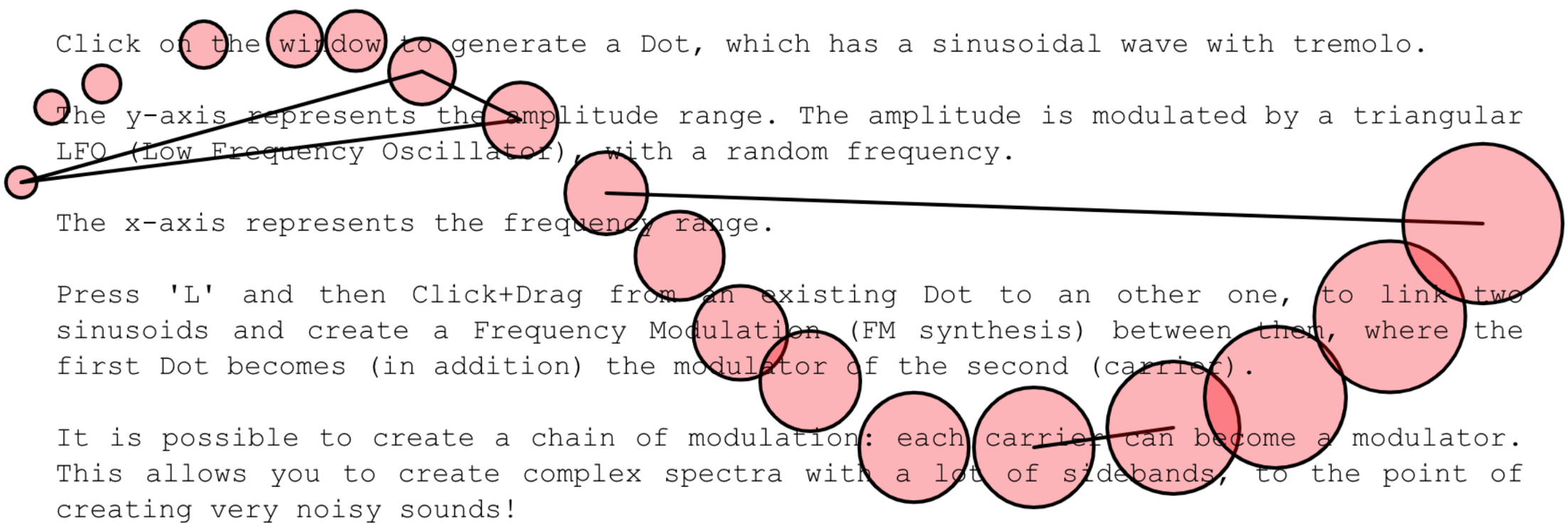
an introduction to the WEB AUDIO API

Day 3

highSCORE

Dot Drone Generator is a drone generator which allows you to create synthetic textures and chords directly on your browser.



Click on the window to generate a Dot, which has a sinusoidal wave with tremolo.

The y-axis represents the amplitude range. The amplitude is modulated by a triangular LFO (Low Frequency Oscillator), with a 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 (FM synthesis) between them, where the first Dot becomes (in addition) the modulator of the second (carrier).

It is possible to create a chain of modulation: each carrier can become a modulator. This allows you to create complex spectra with a lot of sidebands, 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.

Alberto Barberis

2021

6. JAVASCRIPT DEALING WITH INTERACTION - 1

Dot Drone Generator is a drone generator which allows you to create synthetic textures and chords directly on your browser.

Click on the window to generate a Dot, which has a sinusoidal wave with tremolo.

The y-axis represents the amplitude range. The amplitude is modulated by a triangular LFO (Low Frequency Oscillator), with a 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 (FM synthesis) between them, where the first Dot becomes (in addition) the modulator of the second (carrier).

It is possible to create a chain of modulation: each carrier can become a modulator. This allows you to create complex spectra with a lot of sidebands, to the point of creating very noisy sounds!

Dot Drone Generator

highSCORE Festival 2021 | Alberto Barberis

Dot Drone Generator is a drone generator which allows you to create synthetic textures and chords directly on your browser.

Click on the window to generate a Dot, which has a sinusoidal wave with tremolo.

The y-axis represents the amplitude range. The amplitude is modulated by a triangular LFO (Low Frequency Oscillator), with a 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 (FM synthesis) between them, where the first Dot becomes (in addition) the modulator of the second (carrier).

It is possible to create a chain of modulation: each carrier can become a modulator. This allows you to create complex spectra with a lot of sidebands, 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.

6.1 JAVASCRIPT LANGUAGE

Javascript (JS) is the programming language that allows to **transform web pages from static to dynamic**, thanks to **scripts** (portions of code) inserted in the HTML code through the `<script>` tag.

The 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 only once, when it is loaded.

These instructions can be used to associate functions to events that respond to the interaction of a user with the interface (ex: I click on a button, a text appears).

JS programming follows an **event model**.

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

6.2 JAVASCRIPT FEATURES

- JS is an **imperative and procedural scripting language**: it consists of a series of instructions (procedures) to be executed when the program requires it.

- JS is an **interpreted language**: it is executed by a program called interpreter (the browser).

- JS is a **dynamic typing language**: it **does not require the definition of data types to be associated with 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 VARIABLES and CONSTANT

In JS there are two 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 store inside the variable can change over time;

- `const` is the keyword to define a **constant**, that is a container for data that can not change; after the definition and initialization of a constant the data con not be substituted;

Let's CODE!!

6.3 VARIABLES and CONSTANT

Here we open the file *hs2021.js* where we find already some constant and variables that we are going to use in the future 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 0.5)
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 modulator oscillator (initialized to null)
let carrier = null; // variable that will contain the carrier oscillator (initialized to null)
```

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.

6.4 DATA TYPE

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 between ‘ ‘ or “ ” or ``;

2. **OBJECTS**
   - **objects of javascript classes**, like the Array;
   - **`function`** (functions are "types", or rather, to be more precise, they are objects!);
   - **objects defined by our classes**;

6.4 DATA TYPE (es: Array)

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

An array is a special variable, which can hold more than one value at a time. It is **data structure**, consisting of a collection of elements with index and value;

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

You access an array element by referring to the **index number**.

`array_name[0]` is the `item1`.
`array_name[1]` is the `item2`.

6.5 FUNCTION

A `function` is a **sequence of instructions**, grouped in a block of code { ... } (curly brackets), with the purpose of **performing a certain task**. Every `function` has a **name**.

A `function` can be **called in any part of the code**, as many times as we need to perform its task.

A `function` may require certain **parameters** to work;

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

If a function is a property of an Object it is called `method`.

6.5 FUNCTION

Example of a **function definition**:

```
function functionName(parameter1, parameter2){

    let out1 = parameter1;
    let out2 = parameter2;

    return out1 + out2;
}
```

Example of **use of this function**:

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

This means that in the variable `sum` we are storing the number 3.

Let's CODE!!

6.5 FUNCTION

We add this portion to our javascript file, defining a function that will start the audio rendering.

```
26  /** ////////////////////////////////////////////////////////////////
27      * SUPPORT FUNCTIONS
28        * definition
29  * //////////////////////////////////////////////////////////////// */
30
31  function startAudio(){ // audio to start when the first dot is created
32      audioContext = new AudioContext(); // create an audiocontext
33      masterGain = audioContext.createGain(); // create a gian Node for the master gain
34      masterGain.gain.value = MASTER_GAIN; // set the level of the master gain (gain adaptation)
35      masterGain.connect(audioContext.destination); // connect the master gain to the destination
36  }
```

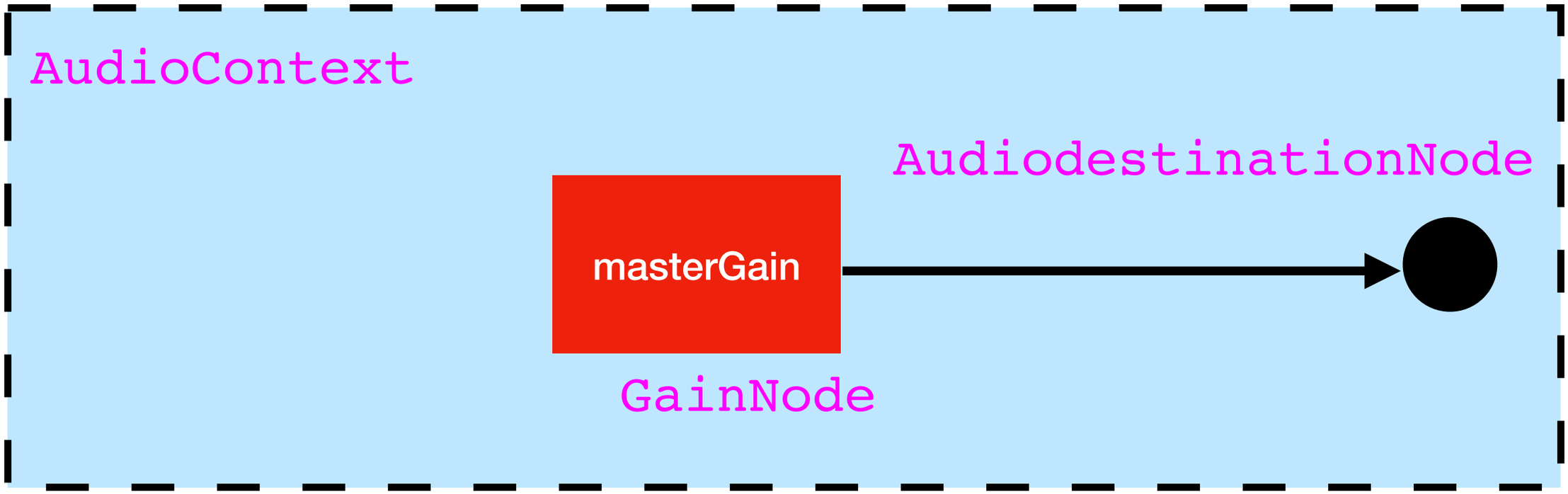
We create an Object of Type AudioContext (using the syntax required by the Web Audio API)

We create a master gain Object with this syntax: `audioContext.createGain()`, that means that we are calling the method `createGain()` to do this, that returns a gain Object.

The `connect()` method is the "virtual connection" between Web Audio Nodes. Here we connect the Master Gain to the destination.

6.5 FUNCTION

This is the representation of the connection that we did before in the code



6.6 OBJECTS

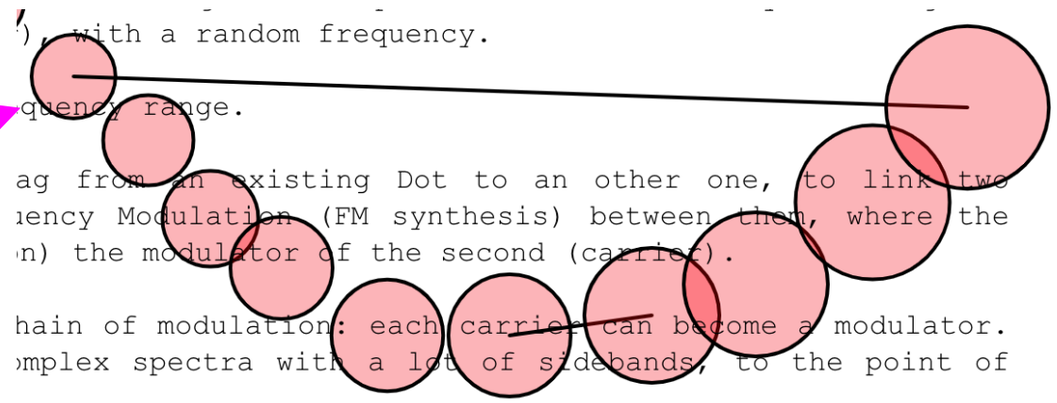
The object-oriented languages are based on **objects**.

An objects 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).*

In our situation we want to create a **sound entity** called Dot. So we will create Dot objects with some common properties.

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



6.6 OBJECTS

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

In an object:

- the **variables** are called **PROPERTIES** (they tell us about the object);
- the **functions** 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).

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

6.6 OBJECTS

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

The **dot .** is called a **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 it to **add**, **retrieve**, **modify** properties and methods.

For example, to fetch the property `freq` of an object with name `dot1` we use the following:

```
dot1.freq
```

6.7 CLASS

A **class** defines the **common structure of some objects**.

It includes **properties** and **methods** that will be shared by the objects of that class.

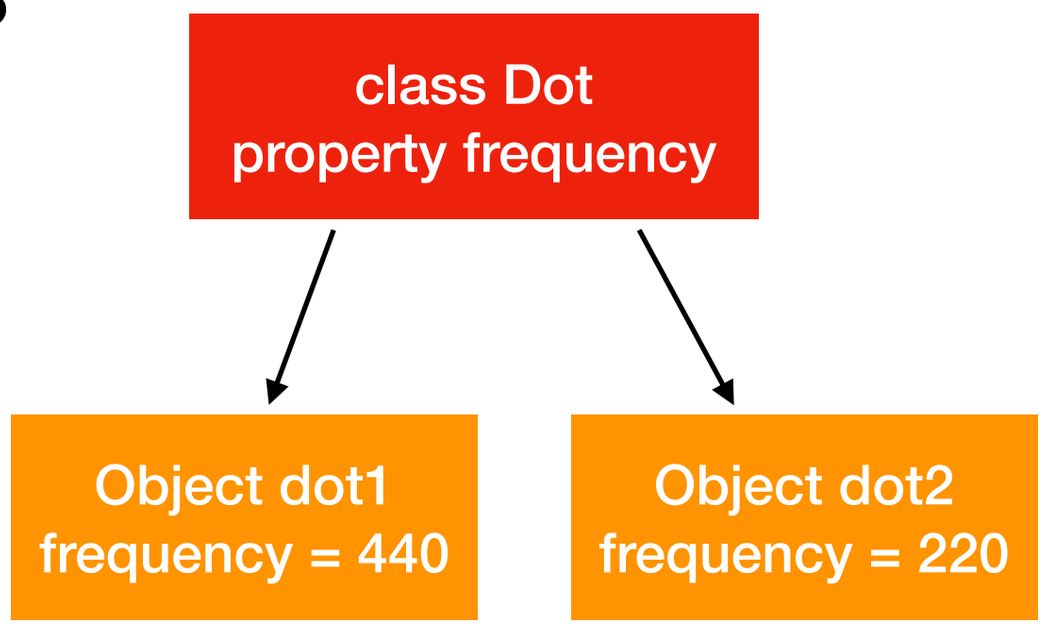
These objects are called **instances of a class**. Each instance has the same structure but different data (eg two oscillators have a *property* frequency, one with *data* 440, one 220).

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 CLASS

From a `class` we can create different instances (Objects) of the same `class`, with different data.



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

- it is executed automatically when an object of that class is created using the keyword **`new`**;
- is used to **initialize the class properties** using parameters; `constructor(par1, par2);`

Definition of the class Dot

the constructor has two parameters, the x and the y position of the center of the Dot

```
class Dot{
    constructor(x,y){

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

        /** array of the possible connections to a Dot
         * connections[0] is the possible Dot object (modulator) connected to this dot
         * connections[1] is the possible Dot object (carrier) connected to this dot
         */
        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);
```

Let's CODE!!

`this` is a keyword that refers to the Object that will be instantiated

the map method is a method of the library p5.js that we are using. I allows us to remap a value from one range to another

Y is the name of a property; and we give to this property the data that will be passed through the constructor

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

We start the definition of the class Dot

Let's CODE!!

```
class Dot{
    constructor(x,y){

        this.x = x; // x position of the Dot
        this.y = y; // y position of the Dot

        /** array of the possible connection
         * connections[0] is the possible Do
         * connections[1] is the possible Do
         */
        this.connections = [null, null];

        // define amplitude and frequency of the associated sinusoid
        this.freq = map(this.x, 0, windowWid
        this.amp = map(this.y, windowHeight

        // set the dimension of the circle a
        this.dimension = map(this.freq, MAX_

        // 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
```

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 timer).

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.

Let's CODE!!

```
class Dot{
    constructor(x,y){

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

        /** array of the possible connect
         * connections[0] is the possible
         * connections[1] is the possible
         */
        this.connections = [null, null];

        // define amplitude and frequency
        this.freq = map(this.x, 0, windo
        this.amp = map(this.y, windowHeig

        // set the dimension of the circl
        this.dimension = map(this.freq, M

        // 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
```

Here we create some gainNode of the WEB Audio API, in fact each SourceNode needs a gainNode to be used.

We create:

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

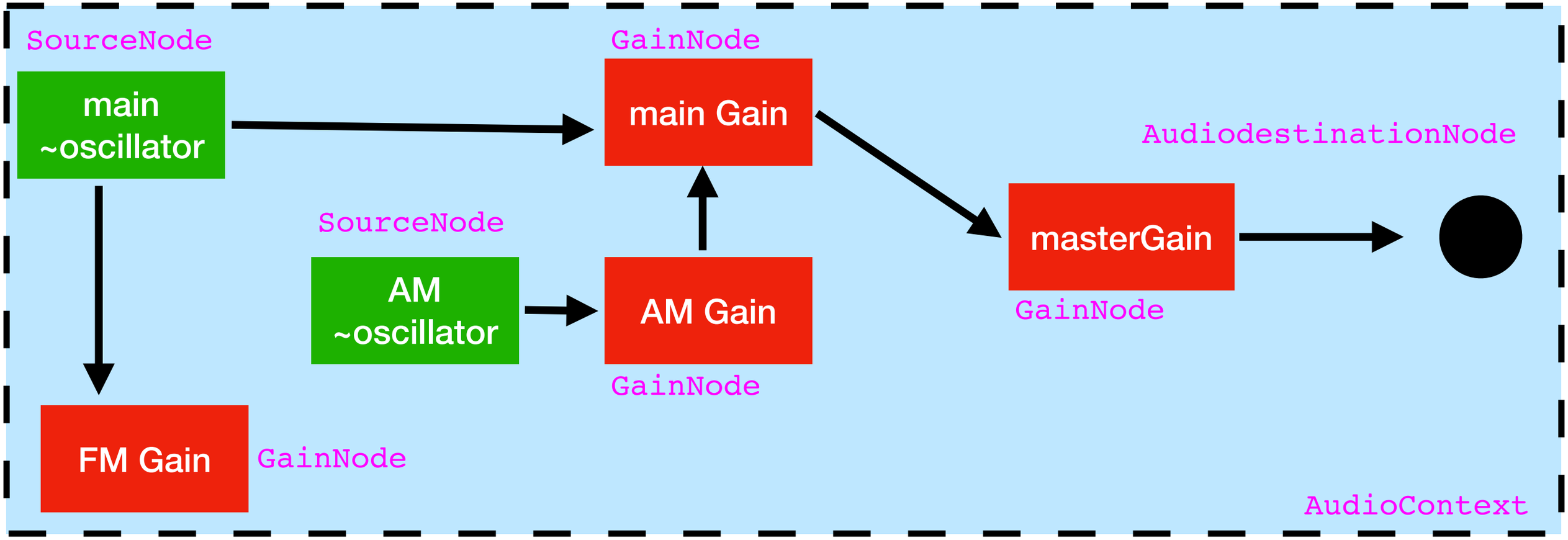
```
this.FMoscGain = audioContext.createGain(); // gain Node for the FM Oscillator (if it is used as modulator) : this is the modulation INDEX
this.FMoscGain.gain.value = 0; // set the initial gain to 0

// 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)
```

Let's CODE!!

Here we make the connections:

- 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 main Gain (this is the AM - tremolo)
- the main oscillator is connected also to the FM orc Gain (the index of modulation)



highSCORE

```
    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

    // 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);
  }
}
```

Let's CODE!!

Here we use the method setTargetAtTime of the Web Audio API to create an exponential ramp that goes from 0 to the desired amplitude.

The larger the ATTACK_TIME is, the slower the transition will be.

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

Example: let's say that `this.amp` is 0.2; this means that `mainOscGain` (that is fixed at 0.2) will be modulated by a triangle waveform that goes from -0.2 to 0.2.

Therefore, the final modulated `mainOscGain` will be a triangle that goes from 0 to 0.4.

6.8 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.

Home
Editor
Download
Donate
Get Started
Reference
Libraries
Learn
Teach
Examples

Hello!

Search p5js.org

p5.js is a JavaScript library for creating creative coding, with a focus on making coding accessible and inclusive for artists, designers, educators, beginners, and anyone else! p5.js is free and open-source because we believe software, and the tools to learn it, should be accessible to everyone.

Using the metaphor of a sketch, p5.js has a full set of drawing functionality. However, you're not limited to your drawing canvas. You can think of your whole browser page as your sketch, including HTML5 objects for text, input, video, webcam, and sound.

Submit a project to the p5.js 2021 Showcase!

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

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

```
98  /** ////////////////////////////////////////////////////////////////////////
99    * P5.JS FUNCTIONS
100     * definition
101   * //////////////////////////////////////////////////////////////////////// */
102
103   function setup() { // create the canvas in the windows with an offset
104       createCanvas(windowWidth, windowHeight - CANVAS_OFFSET);
105   }
106
107   function windowResized() { // resize the canvas if the user changes the window size
108       resizeCanvas(windowWidth, windowHeight - CANVAS_OFFSET);
109   }
```

Let's CODE!!

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

- The `setup` function is called ad 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.

6.9 conditional statement

JS supports all the classic conditional statements:

if .... else
if .... else
switch

```
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
```

Here we implement the function `mousePressed()` that is called any time the user presses the mouse. 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) we want to exit from the function (we do this with `return`);
- we want to create a new Object of our class `Dot` (using the mouse position as the center of the Dot)
- we want to add this Object to the `arrayOfDots` using the method push of the class Array.

```
104     createCanvas(windowWidth, windowHeight - CANVAS_OFFSET);
105 }
106
107 function windowResized() { // resize the canvas i
108     resizeCanvas(windowWidth, windowHeight - CANV
109 }
110
111 function mousePressed() { // functtion called if the mouse is pressed
112     if(!audioContext){ startAudio(); } // if the audio context still does not exists create it
113     if(mouseY < 0) return; // exit if the mouse is out of the canvas
114
115     let newDot = new Dot(mouseX,mouseY); // create the Dot calling the constructor of the class
116     arrayOfDots.push(newDot); // put the Dot in the array of Dots
117 }
```

`new` invokes the constructor of the Dot class that we have already defined.

Let's CODE!!

6.10 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 called to draw on the canvas
    clear(); // clear the canvas
    fill('rgba(255,0,0,0.3)'); // 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's CODE!!

Here we implement the function `draw()` of p5.js. 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 called to draw on the canvas
    clear(); // clear the canvas
    fill('rgba(255,0,0,0.3)'); // 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's CODE!!

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

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 in the loop is executed 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.