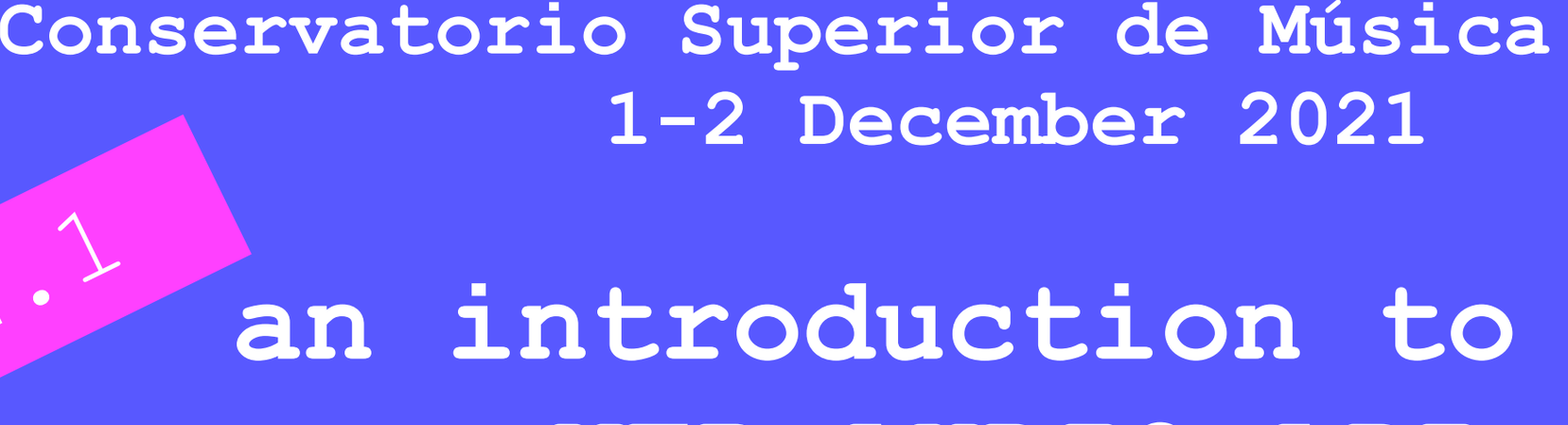
Conservatorio Superior de Música de Murcia
1-2 December 2021

2.1

an introduction to the WEB AUDIO API

Dot Drone Generator

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

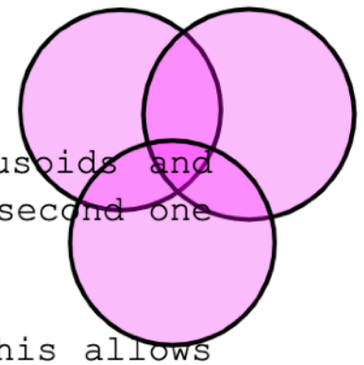


Click on the window to generate a **Dot**, a sinusoidal wave with tremolo.

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).

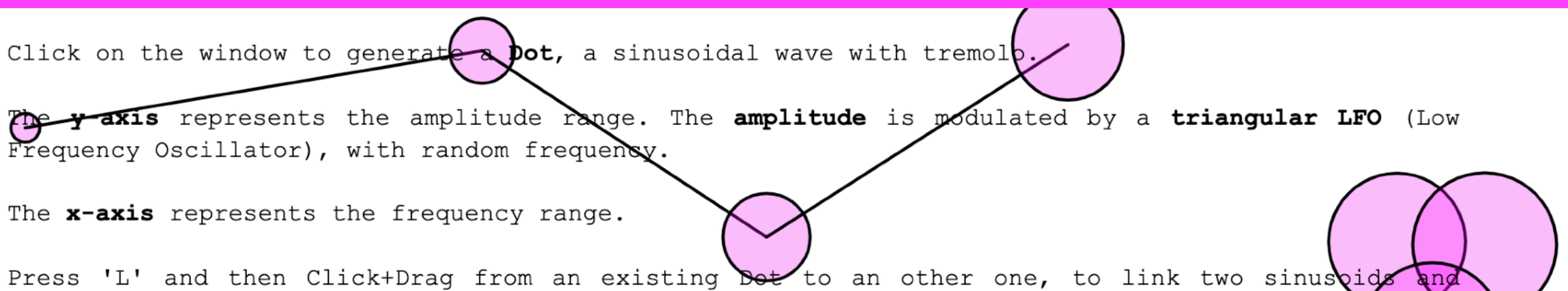


It is possible to create a **chain of modulation**: each carrier can become a modulator. This allows you to create complex spectra, to the point of creating very noisy sounds!

Click on an existing circle to **delete** it or to delete the modulation chain of which it is part.

Alberto Barberis

6. JAVASCRIPT DEALING WITH INTERACTION _ 1



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.

6.1 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.

The JS 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 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 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 stored 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 can not be substituted;

6.3 VARIABLES and CONSTANT

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

const MIN_FREQUENCY_HZ = 5; // min frequency in
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
```

open the file *murcia2021.js* where we find some constant and variables that we use in the future code

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 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 `'..'` or `".."` or `` `..` ``;

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

6.4 DATA TYPE (es: Array)

```
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]` 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 within curly brackets `{ }`, created to **perform a certain task**.

Every `function` has a **name** and may require certain **parameters**.

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

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 a **method**.

6.5 FUNCTION

Example of a **function definition**:

```
function functionName(param1, param2){
    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, the return value of the invoked function.

6.5 FUNCTION

```
/** ////////////////////////////////////////////
  * 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
}
```

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.

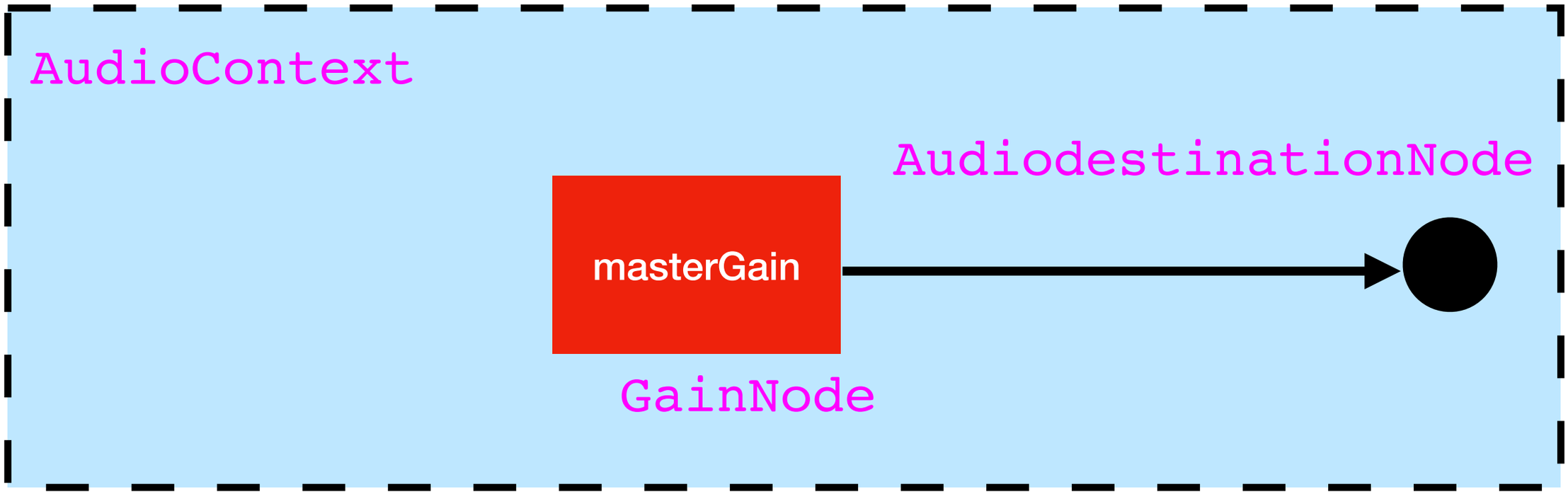
We create a masterGain Object with this syntax: `audioContext.createGain()`, that means that we are calling the method `createGain()`, that returns a Gain Object. `createGain()` is a method of the Class `AudioContext()` that returns a Gain Node.

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.

6.5 FUNCTION

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



6.6 OBJECTS

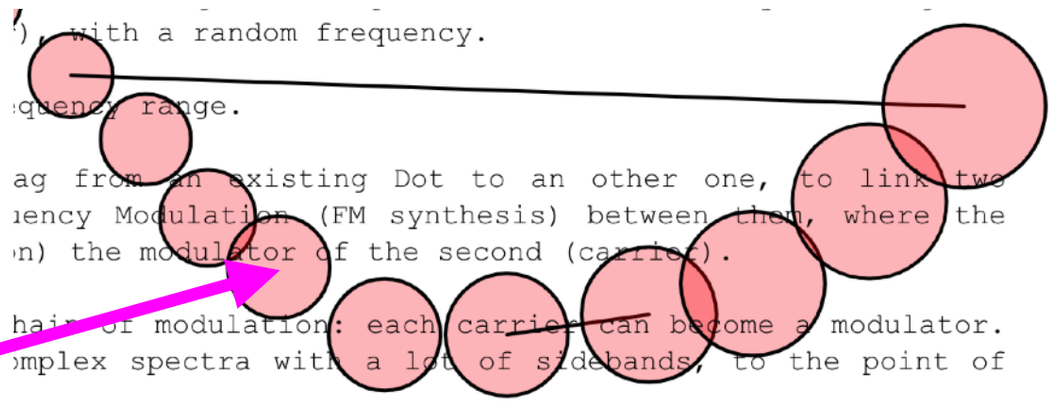
Object-oriented languages are based on **Objects**.

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).*

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

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 OBJECTS

An **Object** groups a set of ***variables*** and ***functions*** (methods) 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 OBJECTS

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

" . " 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 CLASS

A **Class** defines the **structure** of some **Objects**.

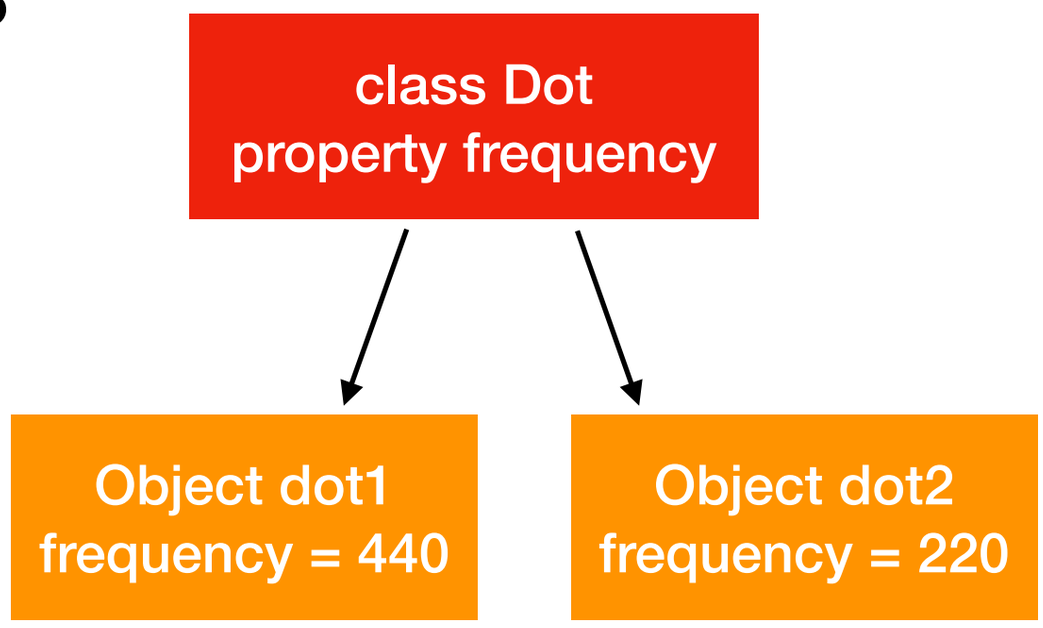
The Class includes **properties** and **methods** that will be shared by the Objects of that class.

The Objects of a certain Class are called **instances of a Class**. Each instance has the same structure but different current data. (Eg. two oscillators have a **property** `freq`, 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 **Objects** (instances) of the Class, that will have different current 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

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

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
         * 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);
```

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

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

y is the name of a property; and we assign to it the data stored in the variable y 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 (in the position at index 0) and the carrier (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

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

        this.x = x; // x position
        this.y = y; // y position

        /** 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
        this.freq = map(this.x, 0, windowWid
        this.amp = map(this.y, windowHeight

        // 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);

        // 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 oscilla
        this.AMosc.type = "triangle"; // definte the type
        this.AMosc.frequency.value = random(MAX_AM_FREQ) + MIN_AM_FREQ; //
        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.

`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(x,y){

        this.x = x; // x position of the Dot (associated to frequency in
        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) connect
         * connections[1] is the possible Dot object (carrier) connected
         */
        this.connections = [null, null];

        // define amplitude and frequency of the associated sinusoid
        this.freq = map(this.x, 0, windowWidth, MIN_FREQUENCY_HZ, MAX_F
        this.amp = map(this.y, windowHeight - CANVAS_OFFSET, 0, MIN_AMPL

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

        // the main oscillator (sinusoid)
        this.mainOsc = audioContext.createOscillator(); // create the os
        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.

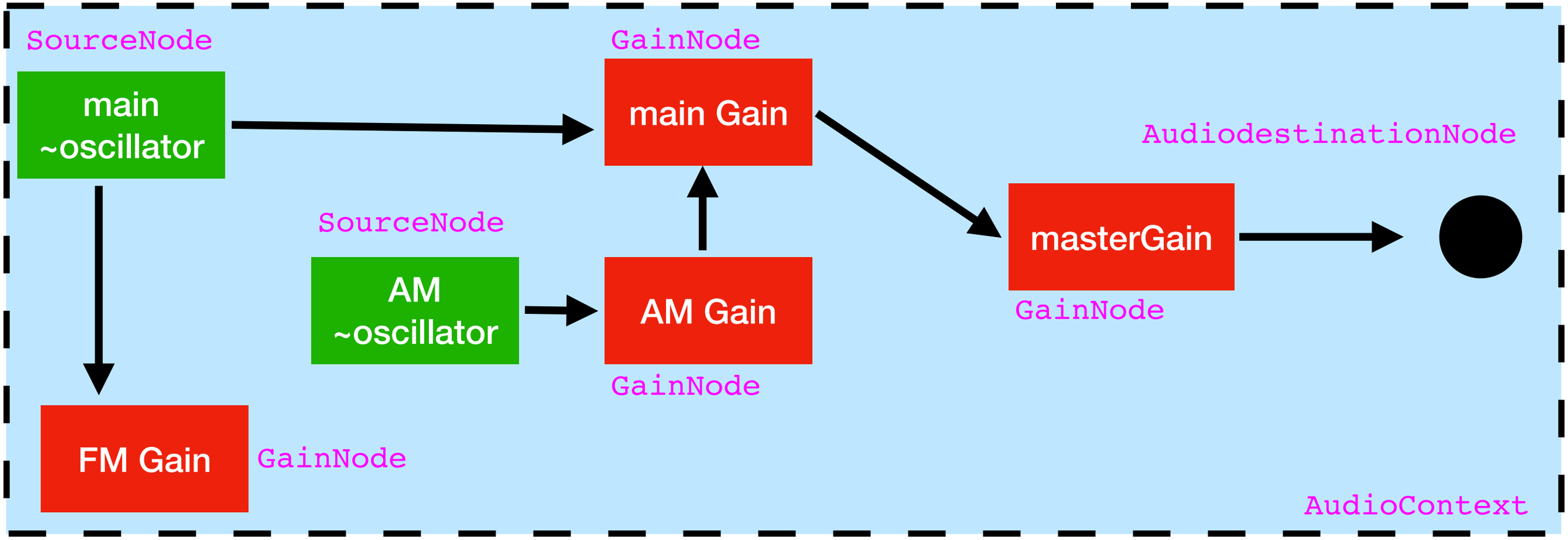
After the creation of the Nodes we set the property `value` of the property `gain` of the Gain Node at 0.

```
        // 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 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)



```
        // 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 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.

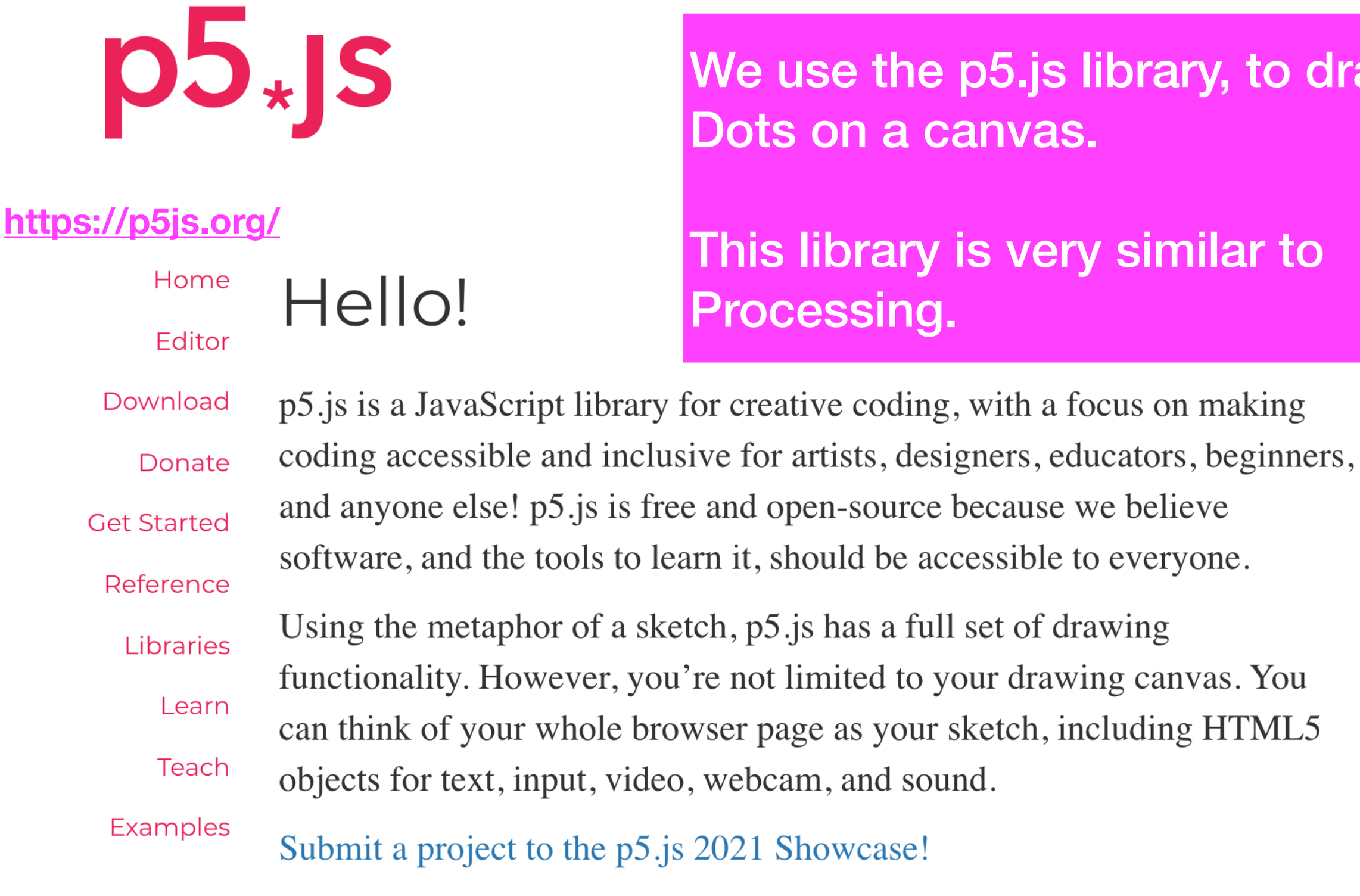
6.8 p5.js

p5.js

https://p5js.org/

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

This library is very similar to Processing.



```
/** ////////////////////////////////////////////////////////////////////
* 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 statement

JS supports all the **classic conditional statements**: **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) or is pressing 'L', we want to exit from the function (we do this with the keyword `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.

```
function mousePressed() { // functtion 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.

6.10 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 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. 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 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.