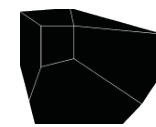


# Arduino Synth

Sintetizadores DIY



casa da música

**DIGITÓPIA**

# Digitópia

A Digitópia é uma plataforma de música digital sediada na Casa da Música, no Porto, que incentiva a audição, a performance e a criação musical. Baseando-se em ferramentas digitais, embora não exclusivamente, enfatiza a criação musical colaborativa, o design de software, a educação musical e a inclusão social, promovendo a emergência de comunidades multiculturais de performers, compositores, curiosos e amantes de música. Com uma presença física composta por 12 computadores e diversos controladores musicais, recebe alunos de várias escolas, promove seminários, é visitada por curiosos, turistas, crianças e seniores, proporcionando experiências musicais auditivas e criativas, enriquecedoras para todos aqueles que as vivenciam.

[www.casadamusica.com/digitopia](http://www.casadamusica.com/digitopia)

[www.facebook.com/DigitopiaCasaDaMusica](https://www.facebook.com/DigitopiaCasaDaMusica)

[www.facebook.com/groups/digitopiacasadamusica](https://www.facebook.com/groups/digitopiacasadamusica)

[www.github.com/Digitopia](https://www.github.com/Digitopia)



# Digitópia Collective

Singular no panorama nacional, o Digitópia Collective é constituído pelos formadores do Serviço Educativo associados à Digitópia, a plataforma artística da Casa da Música reservada à criação musical em suporte tecnológico.

No seu trabalho, o ensemble aplica processos e modelos tão diversos quanto o design de instrumentos digitais, a concepção de hardware próprio, o circuit-bending, a exploração das relações entre imagem e som, a prática de VJ's e DJ's, a digital media ou os sistemas digitais interactivos. Da confluência de linguagens, trazidas por cada elemento do grupo, surge um repertório de música electrónica e digital com um declarado carácter performativo.

# Tiago Ângelo

## Académico

Conservatório em Trombone, CMC

Música Electrónica e Produção Musical na ESART - IPCB

Mestrado Multimédia, Perfil de Sound Design e Música Interactiva na FEUP - UP

## Profissional

Freelancer

Digitópia - Casa da Música

Composição e media interactivos para artes performativas (DEMO, Radar360)

## Artístico

Composição e performance Electrónica

Instalações interactivas

Construção de instrumentos musicais e desenvolvimento de software musical

Tiago Ângelo  
tiagoangelo.tk



# Plano (sábado)

## Manhã

(Breve) Introdução ao Arduino

Introdução à biblioteca Mozzi

## Tarde

*Hands On* - construção de processadores de sinal

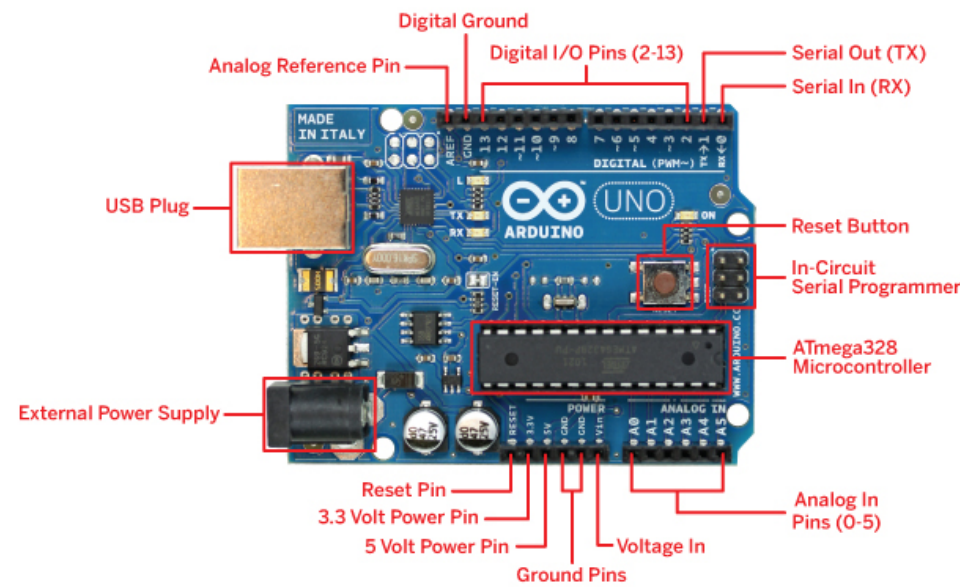


<http://arduino.cc/>



- Plataforma de prototipagem electrónica *open source*;
- Desenvolvida por Massimo Banzi, Tom Igoe, David Cuartielles, David Mellis e Gianluca Martino que começou por ser desenvolvida em 2005 em Ivrea, Itália;
- Documentário: <https://vimeo.com/18539129>

# Hardware



retirado de <https://myp-tech.wikispaces.com/+The+Arduinio+Project>

TX significa *transmit*

RX significa *receive*

Para que serve o pin AREF ? (ver link: <http://blog.arduino.cc/2010/12/13/tutorial-arduino-and-the-aref-pin/>)

# Software (IDE)



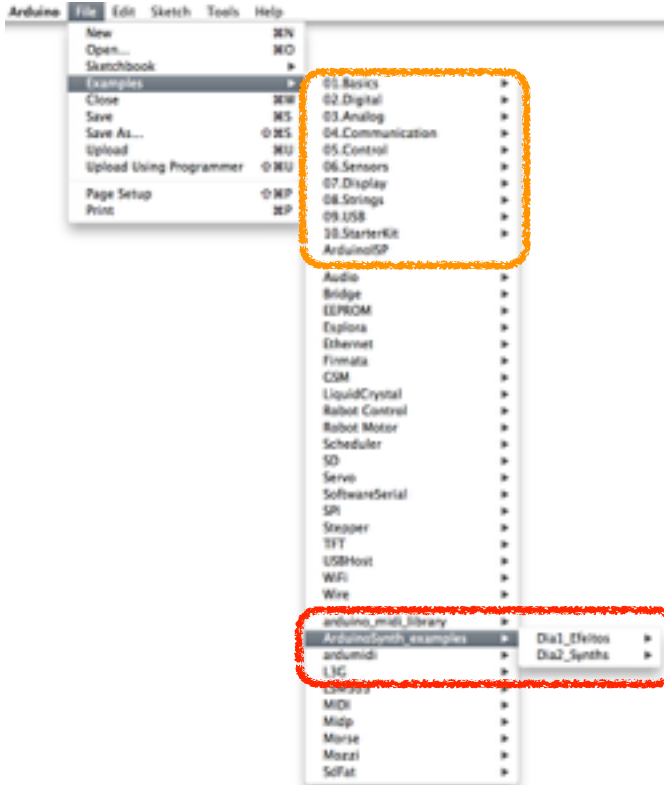
- IDE - *Integrated Development Environment*
- permite editar e carregar um programa para o microcontrolador
- semelhante ao Processing
- baseado em C++



# Instalação

- **OSX** : <http://arduino.cc/en/Guide/Howto>
- **Windows** : <http://arduino.cc/en/guide/windows>
- **Drivers** :
  - **Guia de instalação** : <http://www.ftdichip.com/Support/Documents/InstallGuides.htm>
  - **Link para download** : <http://www.ftdichip.com/Drivers/VCP.htm>

# Arduino - Iniciação



- Exemplos Arduino

Logo: casa da música DIGITÓPIA

Os exemplos referidos em cada página deste documento serão assinalados por baixo do título. Exemplo:

[~/ArduinoSynth\\_examples/Dial\\_Efeitos/\\_01\\_LED\\_LigarDesligar.ino](#)

# Arduino - Iniciação



- Config, Verify & Upload

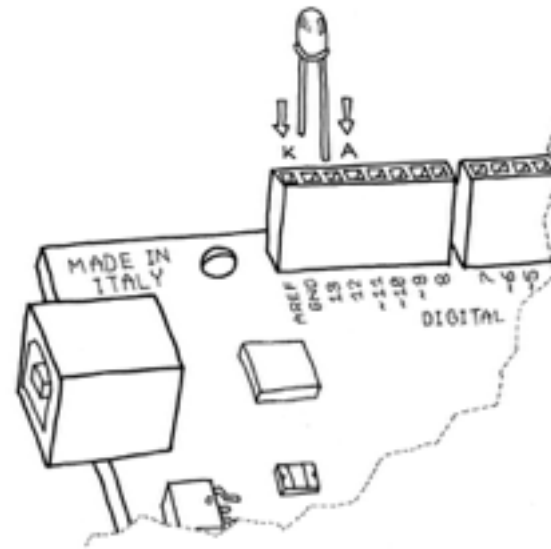
Antes de carregar código para a placa Arduino é necessário configurar o IDE de modo a corresponder à placa utilizada. Para tal deverá escolher a sua placa, o processador que esta possui e a identificação da porta de comunicação.

Para verificar se o código contém erros deverá carregar no botão 'v' (*verify*) ou alternativamente navegar através do menu da aplicação em 'Sketch —> Verify/Compile'. A informação relativa à verificação/compilação será então disposta na *console*.

Se o código não tiver erros poderá então carregá-lo através do botão '—>' (*upload*) ou alternativamente através do menu da aplicação em 'File —> Upload'. (NOTA: não é necessário verificar o código antes de carregar uma vez que o código é verificado antes, não sendo carregado caso contenha erros)

# LED - acender/apagar

~/ArduinoSynth\_examples/Dia I \_Efeitos/\_01\_LED\_LigarDesligar.ino



- Saídas digitais

Para este exemplo iremos usar as saídas digitais do Arduino para controlar um LED. Para tal é necessário ligar o LED conforme indicado na figura, com a perna mais pequena para o *ground (GND)* e a perna maior para o *pin 13*.

# LED - acender/apagar

~/ArduinoSynth\_examples/Dia I \_Efeitos/\_01\_LED\_LigarDesligar.ino



```
// 01_LED_LigarDesligar | Arduino 1.6.1
// Este sketch faz piscar um LED

void setup() { //esta função é executada apenas uma vez quando o Arduino é reiniciado
  pinMode(13, OUTPUT); // define o pin 13 como um pin de saída
}

void loop() { // esta função corre em loop repetidamente
  digitalWrite(13, HIGH); // liga o LED
  delay(1000); // espera 1 segundo (1000 milissegundos)
  digitalWrite(13, LOW); // desliga o LED
  delay(1000); // espera um segundo
}

// este sketch é uma tradução para português do exemplo "Blink.ino"
```

→ Função *setup()*

→ Função *loop()*

- Código Arduino (setup, loop, e outras funções)

A estrutura do código Arduino é definido por duas funções base: *setup()* e *loop()*.

A função *setup()* corre apenas uma vez quando a placa é ligada, enquenato que a função *loop()* é executada repetidamente.

As funções são representadas por dois parêntesis a seguir ao seu nome. Exemplos de outras funções:

*pinMode()* : define o comportamento, i.e. se actua como entrada (*INPUT*) ou saída (*OUTPUT*), de um determinado pin;

*digitalWrite()* : envia um determinado valor para um determinado pin;

*delay()* : interrompe a execução do código apenas durante um determinado período de tempo.

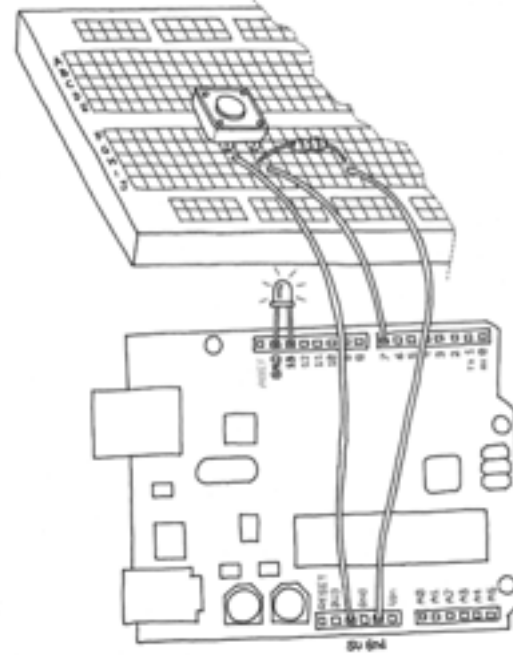
Este exemplo irá manter o LED aceso durante 1 segundo e depois irá apagá-lo durante 1 segundo, repetindo este processo continuamente.

Poderá consultar a referência completa da linguagem Arduino em <http://www.arduino.cc/en/Reference/HomePage>

# Entradas Digitais

~/ArduinoSynth\_examples/Dia I \_Efeitos/\_02\_LED\_ControloDigital.ino

retirado de "Getting Started with Arduino"



- Entradas digitais

Neste exemplo iremos usar um botão para acender e apagar um LED. Para tal é necessário ligar um LED conforme indicado no exercício anterior. Um botão (ou um interruptor ou *switch*) é um mecanismo bastante simples que permite manter ou cortar uma corrente eléctrica num determinado circuito, contendo apenas 2 conexões. Se apenas ligarmos o botão aos +5V do Arduino e a outra conexão ao pin digital, aconteceria que quando o botão não estivesse premido (sem corrente eléctrica) a leitura do pin iria alternar entre ON e OFF (com e sem corrente) uma vez que teríamos um circuito aberto. Para fechar o circuito e obter os valores "correctos" do botão é usada uma resistência ligada ao *ground* (*GND*), esta técnica é denominada de *pull-down*. É possível inverter os valores obtidos ao pressionar o botão se em vez de ligarmos a resistência ao *ground*, conforme demonstrado na figura, a ligarmos aos +5V, denominando-se deste modo *pull-up*.

Note que os pins digitais podem ser usados como entradas ou saídas, dependendo da configuração efectuada no código através da função *pinMode()*.

# Entradas Digitais

~/ArduinoSynth\_examples/Dia I \_Efeitos/\_02\_LED\_ControloDigital.ino



```

02_LED_ControloDigital | Arduino 1.6.1

void setup() { //esta função é executada apenas uma vez quando o Arduino é reiniciado

  pinMode(13, OUTPUT); // define o pin 13 como um pin de saída
  pinMode(2, INPUT); // define o pin 13 como pin de entrada
}

// the loop function runs over and over again forever
void loop() { // esta função corre em loop repetidamente

  if (digitalRead(2) == HIGH){

    digitalWrite(13, HIGH); // liga o LED

  }

  else {

    digitalWrite(13, LOW); // desliga o LED

  }

}

// este exemplo é uma tradução de http://www.arduino.cc/en/Tutorial/Button

```

- Lógica (funções *if* e *else*)

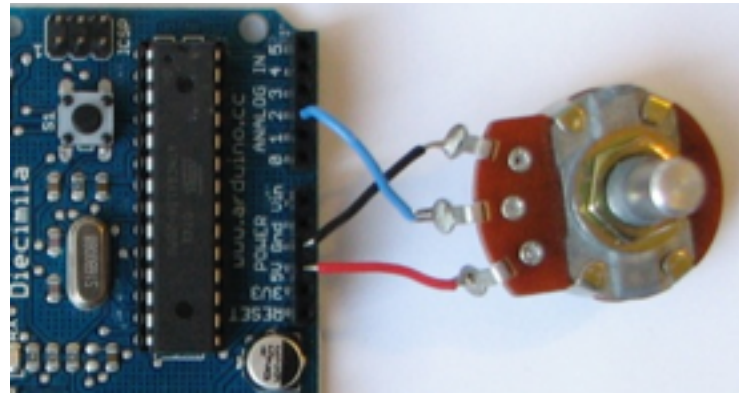


Existe um determinado número de funções cuja função é a de controlar o funcionamento do código efectuando, por exemplo, funções lógicas. Neste caso estamos a usar a função *if()* para determinar se o botão está ou não pressionado. Basicamente esta função permite-nos executar outras funções caso uma determinada condição se prove verdadeira ou falsa. Com esta função podemos usar os seguintes operadores para determinar se uma condição é então falsa ou verdadeira: '==' (igual), '!=' (diferente), '>' (maior), '<' (menor), '>=' (maior ou igual) e '<=' (menor ou igual). Existem outras funções de controlo como: *for()*, *switch()* ou *while()*. Para melhor compreender estas funções veja os exemplos do Arduino, através do menu da aplicação em 'File —> Examples —> 0.5Control'

Note que os valores HIGH e LOW correspondem a 5 e 0 Volts respectivamente, sendo interpretados pelo Arduino através de 8-bits ou seja 255 e 0.

# Entradas Analógicas

~/ArduinoSynth\_examples/Dia I \_Efeitos/\_03\_LED\_ControloAnalogico.ino



retirado de <http://www.arduino.cc/en/tutorial/potentiometer>

- Entradas analógicas

Agora que já vimos como utilizar entradas e saídas digitais vamos então passar às entradas analógicas do Arduino. Note que estas apenas funcionam como entradas ao contrário dos pins digitais.

NOTA: ao contrário das entradas digitais, as analógicas são representadas com 10-bits e não com 8-bits, o que significa que poderão ler valores de 0 a 1023

Neste exercício iremos controlar o tempo que um LED permanece ligado e depois desligado, repetidamente, controlando assim a rapidez com que o LED pisca. Para tal é necessário ligar o pin central de um potenciômetro ao pin analógico ZERO do Arduino (e não ao 2 como está na figura), ligando depois os dois restantes pins ao *GND* e *5V*. Note que se trocar as ligações do *GND* e *5V* irá inverter os valores recebidos no pin analógico do Arduino.



# Entradas Analógicas

~/ArduinoSynth\_examples/Dia I \_Efeitos/\_03\_LED\_ControloAnalogico.ino

```
/*  
declara uma variavel do tipo "int" ( um numero inteiro )  
e atribui-lhe o valor "0"  
*/  
int espera = 0;
```

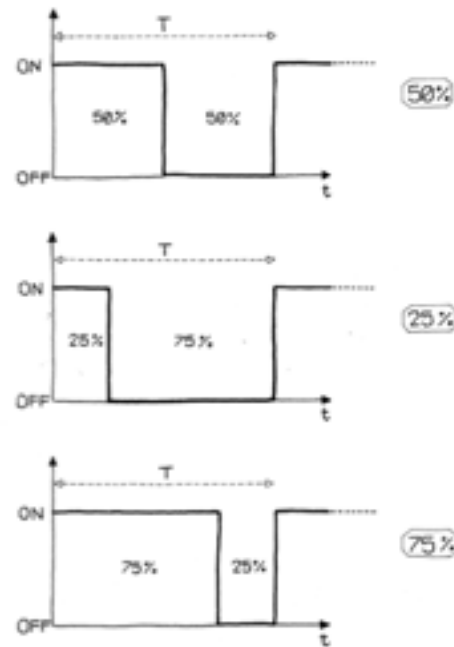
- variáveis e declaração



Uma variável, tal como o nome indica, são segmentos de memória no Arduino com um determinado nome e que podem ser guardadas e manipuladas ao longo do código. Quando declaramos uma variável é necessário atribuir-lhe um nome e um identificador do tipo de variável, para que o Arduino saiba o espaço em memória que irá ocupar. No Arduino é possível criar variáveis do seguinte tipo: **boolean** - variável de apenas dois valores: verdadeiro (*true*) ou falso (*false*) || **byte** - variável de 8-bits (= 1byte) com valores entre 0 e 255 || **int** - variável de 16-bits (= 2 bytes) com valores entre -32768 e 32767 || **float** - variável de 32-bits (= 4 bytes) que permite representar valores decimais. Note que esta variável ocupa bastante espaço na escassa memória do Arduino e as funções que as usam também consomem bastante memória, por isso sempre que possível evite estas variáveis. || **char** - variável de 1byte, com valores entre -128 e 127, usada para representar caracteres. Note no entanto que os caracteres são armazenados como números || **array** - um conjunto de variáveis que pode ser acessado através de um índice (ou localização). Pode servir, entre muitos outros exemplos, para armazenar uma frase ao juntar um conjunto de caracteres (*char*) (ver <https://www.arduino.cc/en/Reference/Array>) || Existem ainda outros tipos de variáveis que poderá consultar em *Data Types* na página <https://www.arduino.cc/en/Reference/HomePage>

# Saídas digitais e PWM

~/ArduinoSynth\_examples/Dia I \_Efeitos/\_04\_LED\_saidaPWM.ino



retirado de "Getting Started with Arduino"



- Pulse Width Modulation
- Saídas digitais PWM

PWM significa *Pulse Width Modulation* (modulação por largura de pulso) e representa uma técnica capaz de obter resultados semelhantes a um sinal analógico através de meios digitais, criando uma onda quadrada - um sinal que alterna entre ligado e desligado (*on/off*). Este padrão *on/off* pode simular voltagens "contínuas" entre 0 Volts e 5 Volts ao alterar a porção de tempo em que o sinal está ligado versus a porção de tempo em que o sinal está desligado (ver imagem). Sendo a duração de tempo em que o sinal está ligado denominado de *pulse width* (largura de pulso). Para obter valores analógicos (entre 0 e 5V) pode-se modular a largura de pulso. Neste exemplo iremos usar esta técnica, implícita na função *analogWrite*, para alterar o brilho de um LED em vez de apenas o manter aceso ou apagado conforme demonstrado nos exemplos anteriores.

Note que nem todos os pins digitais do Arduino permitem utilizar esta técnica, sendo que normalmente têm uma indicação (*pwm* ou ~) para informar que estes pins suportam PWM.

# Saídas digitais e PWM

~/ArduinoSynth\_examples/Dia I \_Efeitos/\_04\_LED\_saidaPWM.ino

```
// tempo para acender o LED
for (i = 0; i <= 255; i++){ // a função "for()" permite criar iterações

    // "analogWrite()" permite-nos enviar valores intermedios entre "LOW" e "HIGH"
    analogWrite(LED, i);
    delay(10);

}

// tempo para apagar o LED
for (i = 255; i >= 0; i --){
    analogWrite(LED, i);
    delay(10);
}
```

parenthesis

declare variable (optional)

initialize

test

increment or decrement

```
for (int x = 0; x < 100; x++) {
    println(x); // prints 0 to 99
}
```

retirado de <https://www.arduino.cc/en/Reference/For>

- controlo e iteração



É possível controlar a forma como código é executado através do uso de funções específicas (ver *Control Structures* em <https://www.arduino.cc/en/Reference/HomePage>).

Uma das funções usadas neste exemplo é a função *for* (para) que basicamente nos permite executar um determinado número de iterações até que uma determinada condição seja atingida.

No seguinte exemplo (ver imagem no topo) são usadas duas funções *for* para criar primeiro uma rampa crescente de valores e depois uma rampa decrescente de valores, fazendo com que o LED ligado neste exemplo vá alternando entre aumentar o brilho e diminuir o brilho de forma contínua.

A função *for* necessita de três entradas: inicialização, condição e execução (ver imagem no fundo).

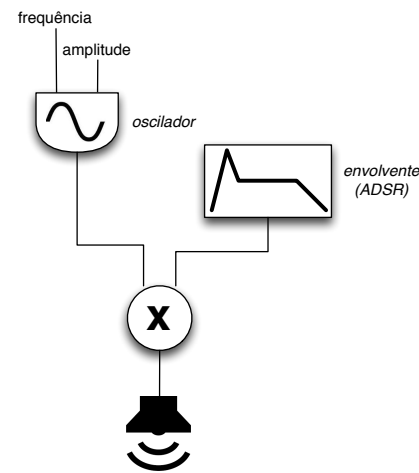
# Leitura recomendada

- Getting Started with Arduino (Massimo Banzi) :  
<http://it-ebooks.info/book/1338/>
- Getting Started in Electronics (Forrest Mims) :  
<https://docs.google.com/file/d/0B5jcnBPSPWQyaTUIOW5NbVJQNw8/edit>
- ( Fórum Arduino : <http://forum.arduino.cc/> )



# Mozzi

<http://sensorium.github.io/Mozzi/>



- Desenvolvida por Tim Barrass
- Biblioteca de síntese sonora para Arduino
- O Arduino passa a ter osciladores, filtros, envolventes, delays, etc...



Esta biblioteca foi desenvolvida por Tim Barrass e para além de suportar a maior parte das placas Arduino, suporta também as placas Teensy.

A Mozzi providencia-nos um conjunto de classes e funções que nos permitem desenvolver modelos de síntese com maior facilidade, permitindo uma taxa de amostragem (*sampling rate*) de ~16KHz ou ~32KHz a uma resolução (*bit-depth*) de 8 ou 14-bits.

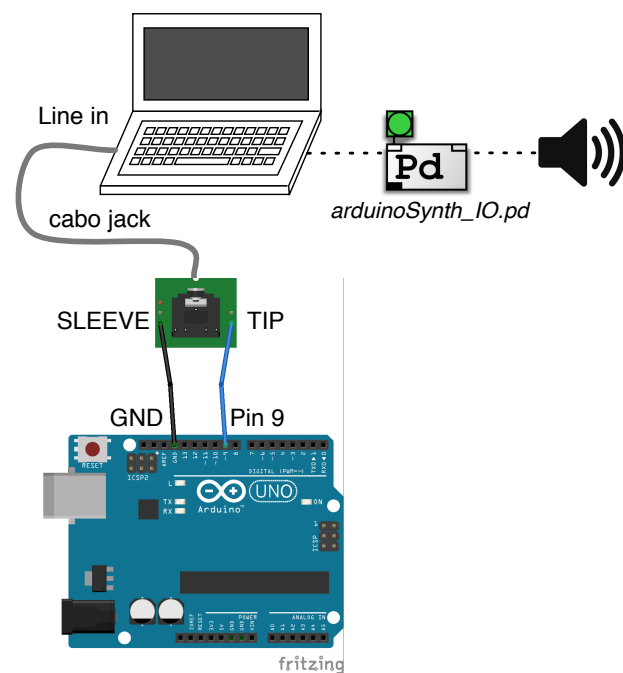
Os seguintes conteúdos deste workshop estão divididos em duas partes: a primeira dedicada à criação de processadores de efeitos (entrada áudio—>processamento—>saída áudio) e segunda dedicada ao desenvolvimento de sintetizadores (entrada MIDI —>síntese—>saída áudio)

Para instalar bibliotecas para Arduino veja o seguinte link: <https://www.arduino.cc/en/guide/libraries>

Para saber mais sobre o tema Áudio Digital pode descarregar os conteúdos do workshop na digitópia em : [https://github.com/Digitopia/Workshops-Crashcourses/releases/tag/ws\\_11](https://github.com/Digitopia/Workshops-Crashcourses/releases/tag/ws_11)



# Iniciação e conexões



- Como instalar a biblioteca ?
- Carregar exemplo
- Como estabelecer uma ligação áudio ?

ligar ao computador apenas para ouvir  
...poderíamos ligar directamente a umas colunas ou auriculares

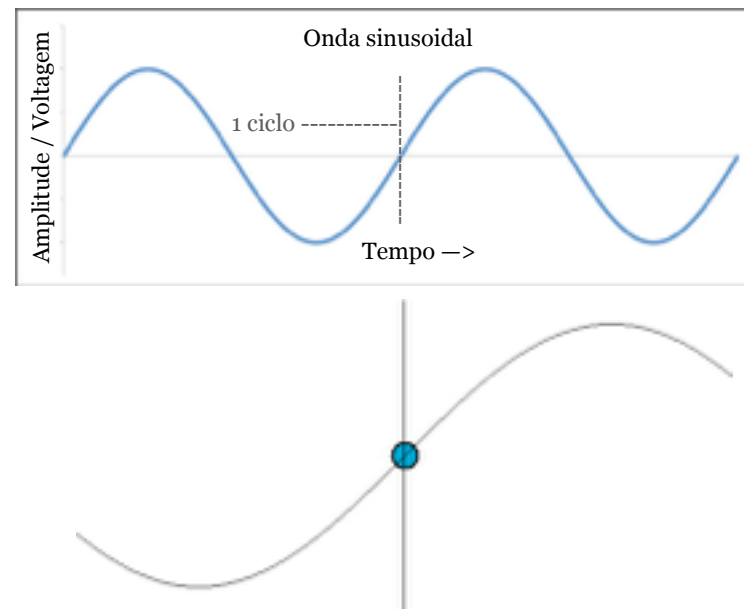


# Arquitectura

```
Skeleton §  
  
#include <MozziGuts.h> // at the top of your sketch  
#define CONTROL_RATE 64 // or some other power of 2  
  
void setup() {  
  startMozzi(CONTROL_RATE);  
}  
  
void updateControl() {  
  // your control code  
}  
  
int updateAudio() {  
  // your audio code which returns an int between -244 and 243  
  // actually, a char is fine  
  return 0;  
}  
  
void loop() {  
  audioHook(); // fills the audio buffer  
}
```

- include Mozzi
- startMozzi + Control Rate
- updateControl
- updateAudio
- audioHook

# Sinusoide



- O que são e como utilizar classes ?
- consultar doc. :

<http://sensorium.github.io/Mozzi/doc/html/index.html>

\_05\_SOM\_SimplesOscilador.ino

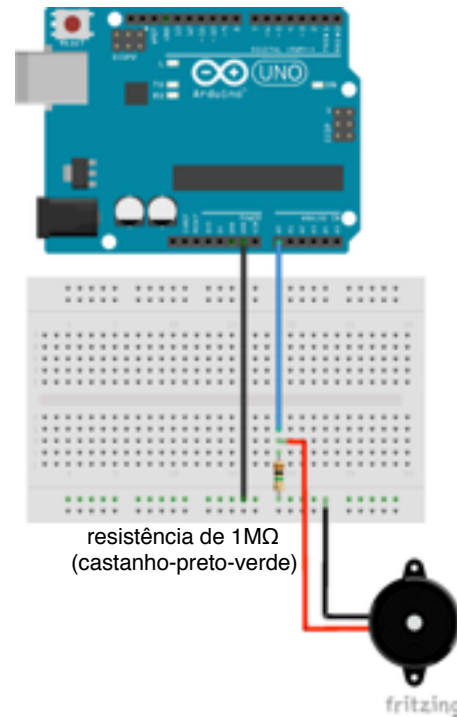
Osciladores e tabelas serão explicados com maior detalhe no dia 2 (domingo)

#define vs. constant





# Piezo in



- conexões

- editar *mozzi\_config.h*

```
72 #define USE_AUDIO_INPUT true
73 //#define USE_AUDIO_INPUT false
```

- função *getAudioInput()*

ATENÇÃO : ATmega168 não têm memória suficiente!!!

\_06\_SOM\_PiezoInput.ino



# Optimização

```
26 // #define AUDIO_MODE STANDARD
27 #define AUDIO_MODE STANDARD_PLUS
28 // #define AUDIO_MODE HIFI
```

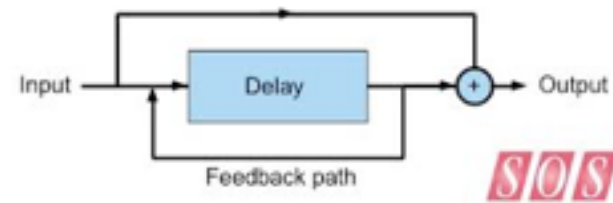
```
61 #define AUDIO_RATE 16384
62 // #define AUDIO_RATE 32768
```

- *Output Modes (STANDARD, STANDARD\_PLUS, HIFI)*
- *Output circuits (reference only)*

STANDARD = 8bit  
HIFI = 14bit

se quiserem podem montar os circuitos de saída durante a tarde

# Feedback Delay

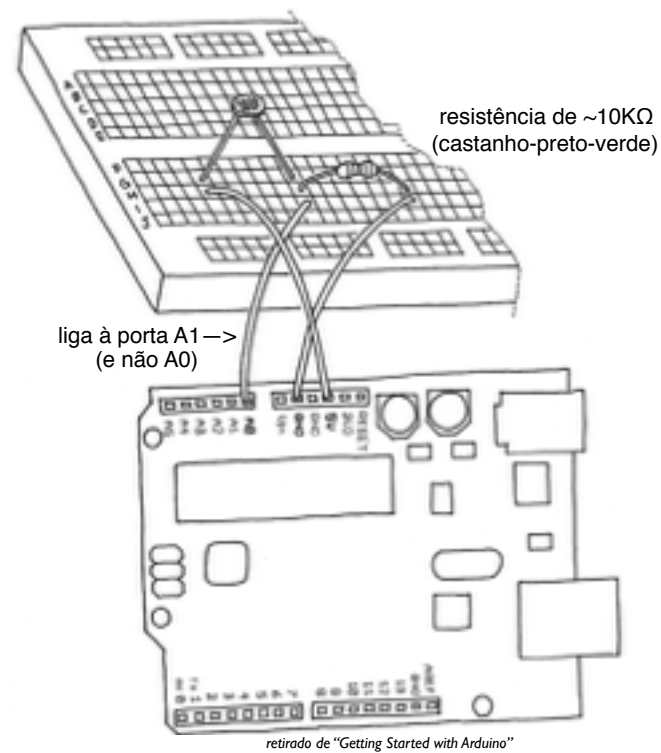


- AudioFeedbackDelay
- Parâmetros : duração do delay, ganho do feedback

*SOS*



# Controlo



- *Serial Monitor* : *Serial.begin()* e *Serial.println()*
- *Lowpass filter*
- Parâmetros : frequência de corte, ressonância/ resposta
- Outras funções : *IntMap*



# Outros Efeitos

- Waveshaping // distorção
- Flanger (feedback delay + modulação no tempo de duração)
- bit distortion (usando bitshif >> e <<)
- AM // tremolo
- Reverb
- .....

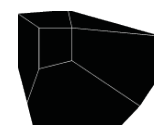
# Extras

- Output modes (standard, standard plus e HiFi) e circuitos de saída
- Entrada áudio adequada (<http://www.instructables.com/id/Arduino-Audio-Input/?ALLSTEPS>)
- Modulações (osciladores, random, etc...)
- Hints & Tips (variáveis, funções)
- Algoritmos: [www.musicdsp.org](http://www.musicdsp.org)

# Arduino Synth

Sintetizadores DIY

[tiago.a.s.angelo@gmail.com](mailto:tiago.a.s.angelo@gmail.com)



casa da música

**DIGITÓPIA**

# Plano (domingo)

## Manhã

Overview - modelos de síntese

Experimentação

## Tarde

Controlo MIDI

*Hands On* - construção de um sintetizador



# Síntese

- contexto histórico / exemplos
- modelos de síntese
  - abstractos/modulação
  - gravações processadas
  - *espectrais*
  - *físicos*
- ref.: *Theory and Technique of*  
e

REF.: *Theory and Technique of Electronic Music* (Miller Puckette)



—> link para o ws do zé

# Tabelas e osciladores

- Síntese *wavetable*
- Formas de onda:  
Documentos/Arduino/  
libraries/Mozzi/tables/
- As tabelas têm uma  
resolução de amplitude de  
8-bits, variando entre -128  
e 127

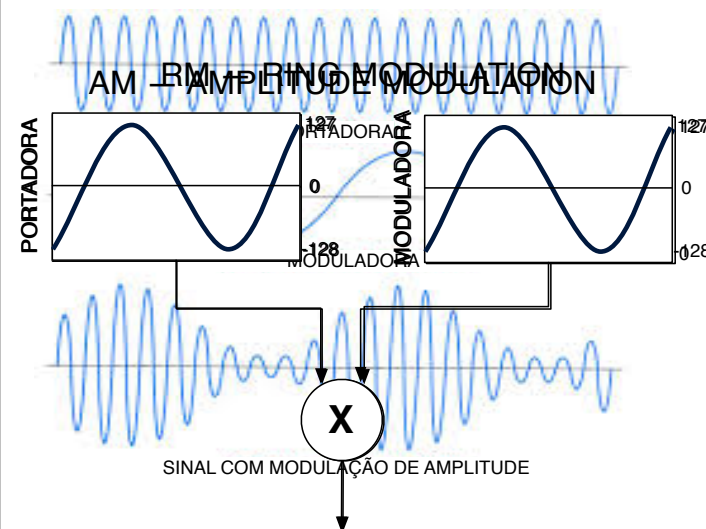
```
/*  
  Basicamente um oscilador le uma tabela  
  de valores que representam uma forma de onda  
*/  
  
// Incluimos a classe 'Oscil' para usar osciladores  
#include <Oscil.h>  
// Incluimos a tabela que queremos usar  
#include <tables/sin2048_int8.h>  
  
//Declaramos o oscilador  
Oscil <SIN2048_NUM_CELLS, AUDIO_RATE> oMeuOscilador(SIN2048_DATA);  
//< Tamanho da tabela, Quantas x faz update> nomeDoOscilador(Dados para o oscilador)
```



explicar onde está a referência do Arduino : menu Help>Find in reference

explicar onde está a referência do Mozzi : ~/Documents/Arduino/libraries/Mozzi/extras/doc/index.html

# Modulação de amplitude (AM/RM)



- Usa um sinal modulador para alterar a amplitude de um sinal portador
- Tremolo (freq. moduladora < 16Hz)
- AM - moduladora unipolar
- RM - moduladora bipolar

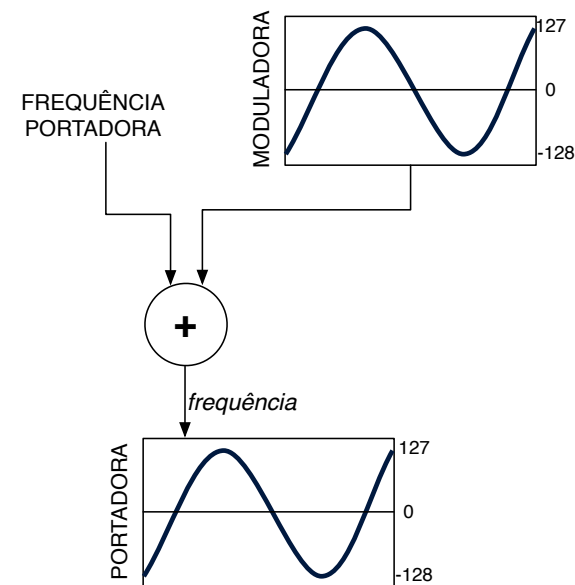
DICA:

Modulation Depth = amplitude da moduladora

# Síntese Aditiva

- segundo Fourier é possível uma onda complexa pode ser reconstruída através da adição de várias ondas sinusoidais
- este tipo de síntese consiste basicamente na adição de vários osciladores
- o timbre é então determinado pela amplitude e frequência de cada oscilador, sendo que frequências próximas poderão criar batimentos, ou seja, as frequências interagem entre si criando depressões e aumentos de amplitude
- ver também exemplo Mozzi “Detuned\_Beats\_Wash.ino”

# Modulação de Frequência (FM)



- Descoberta por John Chowning em 1967
- O seu ponto forte era a possibilidade de criar tons complexos (semelhante à síntese aditiva) com apenas dois osciladores

# Sampling



- Os samples são em tudo semelhantes às tabelas, mas por norma são mais longos (>NUM\_CELLS)
- Estão em Documentos/Arduino/libraries/Mozzi/samples/
- Tal como os osciladores também é possível alterar a frequência!

# MIDI

- o que é ?
- tipos de mensagens (pitch, vel, cc, pg ch, pitch bend...)
- formato das mensagens



—> link para o ws do zé

MIDI – Musical Instrument Digital Interface, protocolo serial (8/14-bits)

# MIDI no Arduino (usb)

1. Carregar o sketch \_05\_MIDI\_SimpleSineNoteOn.ino

2. Abrir o “Hairless midiserial”



**ATENÇÃO:** não é possível carregar código para o Arduino enquanto o Hairless midiserial estiver ligado pois a porta de comunicação fica ocupada



# Control Change (cc)

- como receber MIDI (e enviar)
- *como criar uma entrada MIDI ?*
- enviar MIDI por USB

# Mozzi, MIDI e Funções desactivadas

- Funções (mtof, ftom...)
- ATENÇÃO!!! O Mozzi desactiva as seguintes funções do Arduino:
  - delay(), delayMicroseconds(), millis() e micros()
  - em substituição tem: EventDelay(), Metronome() e mozziMircros()

# + Síntese

- additive synthesis  
(Detuned\_Beats\_Wash.ino)
- PWM phasing  
(PWM\_Phasing.ino)
- Phase distortion  
(PDresonant.ino)
- scrubbing
- granular ?

REF: “Computer Music Tutorial”,

# Extras

- sequências
- standalones (alimentação,...)
- Optimização do código (FixedMath e variáveis Mozzi...)
- Debug
- como criar tabelas (soundtables) ?
- *cv control in/out* ?
- como ler diagramas ?
- circuitos electrónicos pos/pre Arduino synth (filtros, spring reverb, ?)



<http://diyaudioprojects.com/Schematics/>  
<http://www.diyaudiocircuits.com/schematics/>