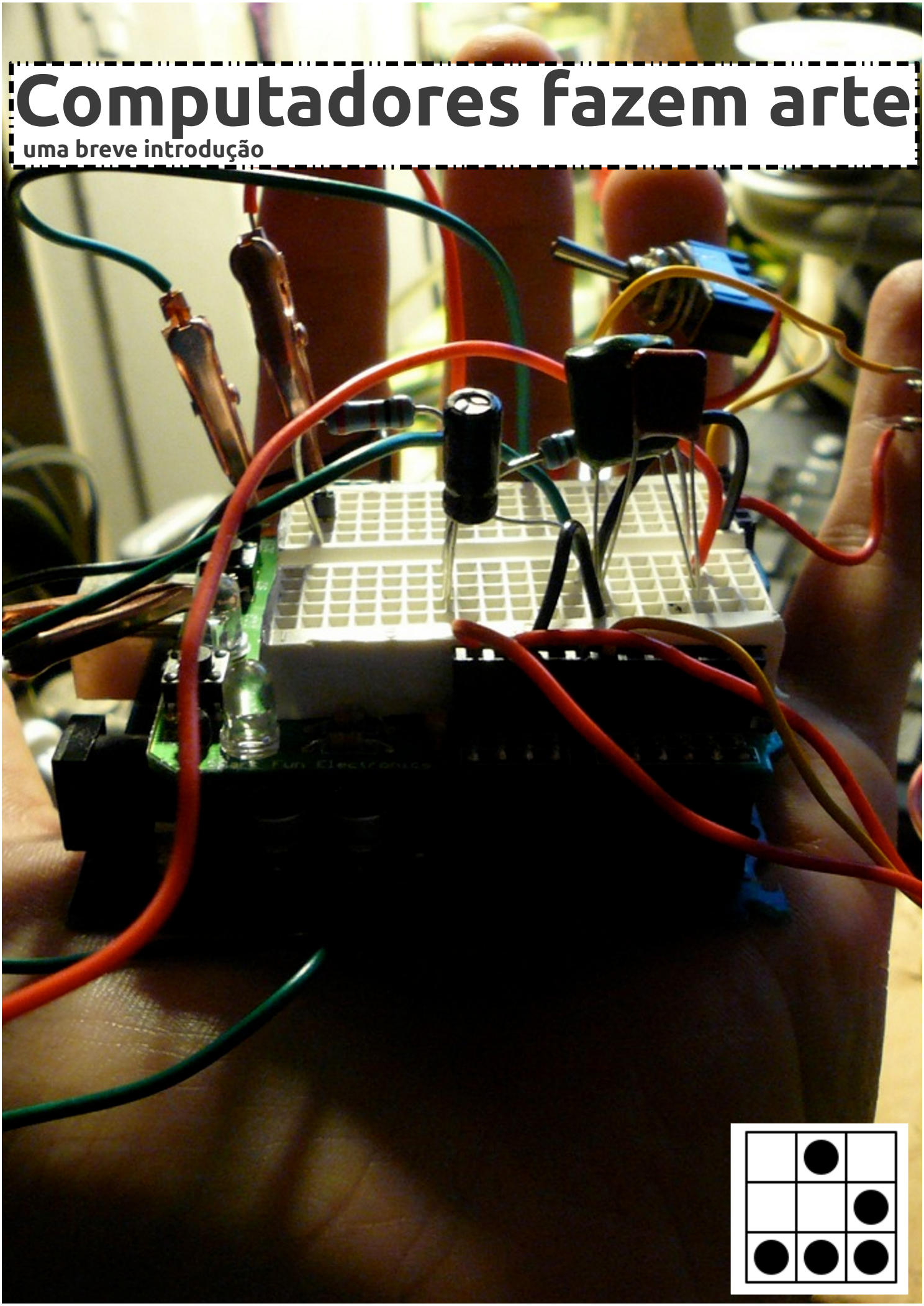
Computadores fazem arte

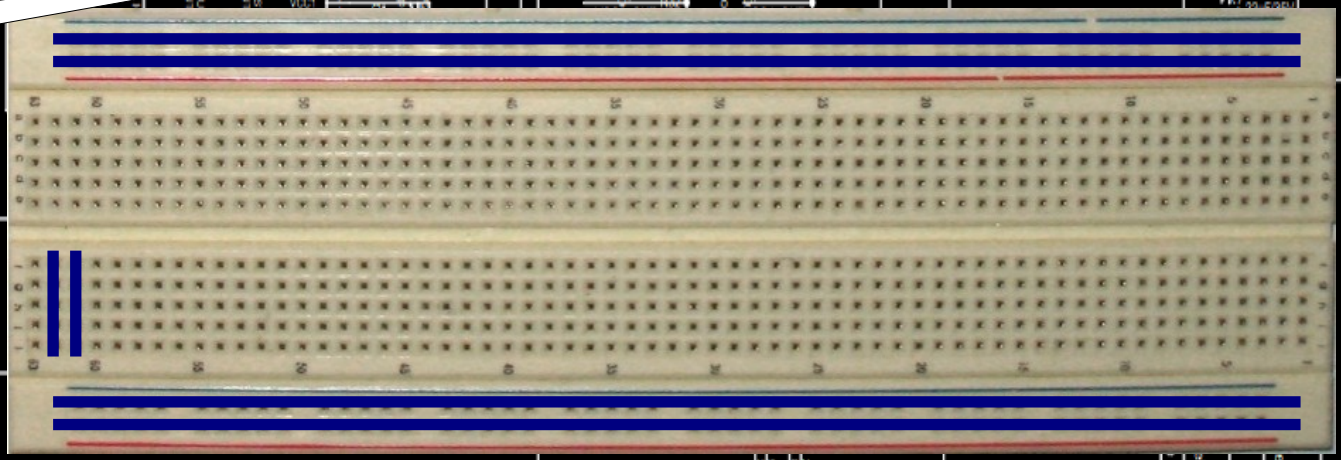
uma breve introdução



{{{ ANTES DE COMEÇAR ... }}}

Prazer, nós seremos suas ferramentas! Nossos nomes são:

E esta é a protoboard, onde iremos experimentar todos os nossos circuitos!

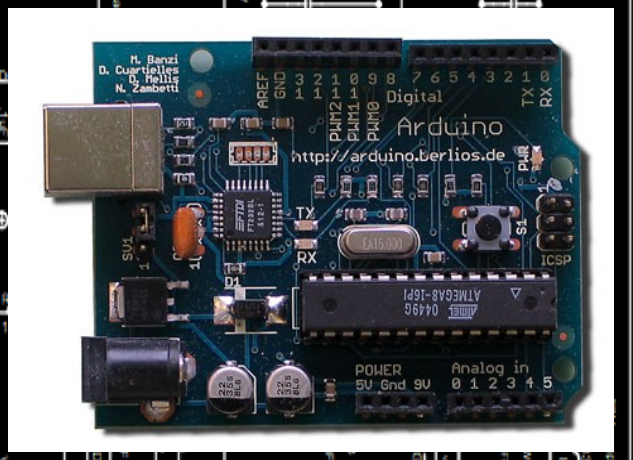


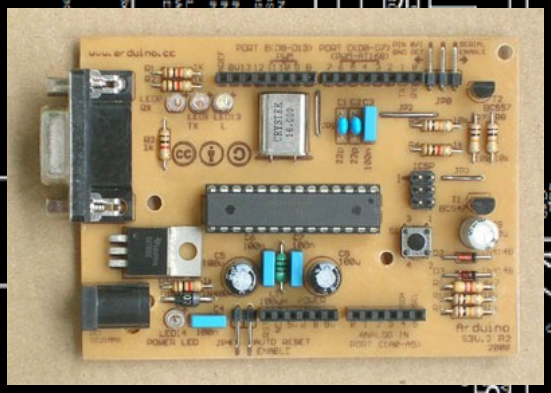
As linhas em azul indicam como a protoboard funciona internamente.
Os pinos de cada linha estão todos conectados.

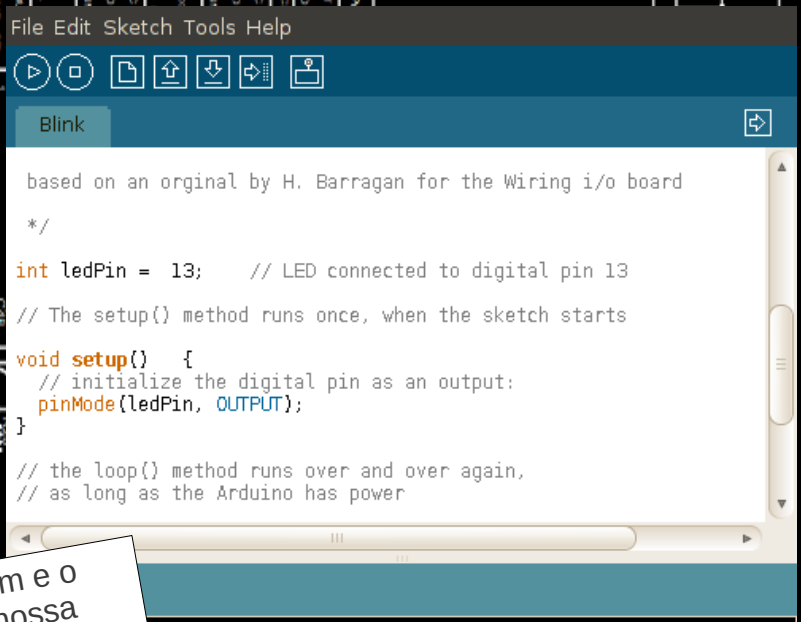
(((Arduino)))

Arduino é uma iniciativa de se criar um ambiente para desenvolvimento de ***hardware livre***. Arduino não é só a plaquinha que faz a comunicação de sensores/atuadores com o computador, mas também é uma linguagem de programação e um ambiente de desenvolvimento integrado.

Com **Arduino** podemos conectar diversos tipos de **sensores** (botões, potenciômetros, LDRs, ...) e capturar sensações do mundo físico para o mundo abstrato do computador. Ou o inverso: usar o computador para, através de **atuadores** (motores, LEDs, ...) modificar o mundo físico onde vivemos.





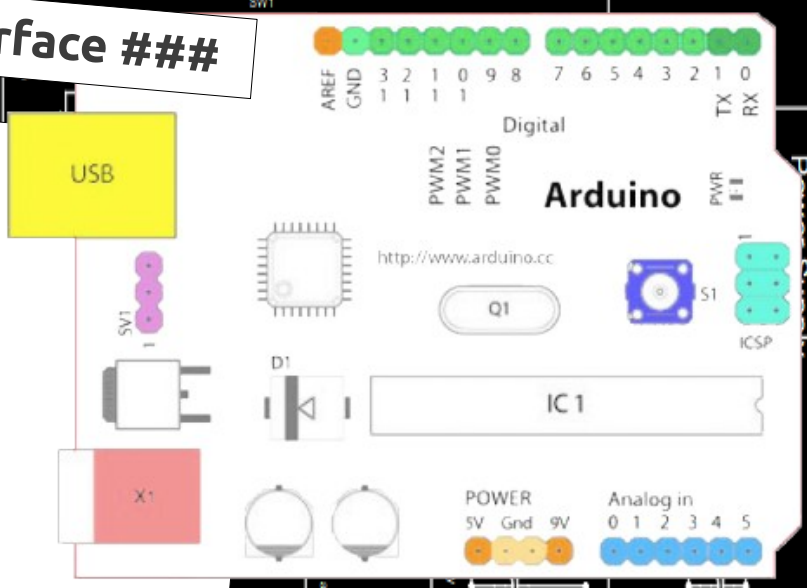


Tanto a placa **Arduino** quando a linguagem e o editor são **livres**. Ou seja, podemos criar nossa própria versão da placa, da linguagem ou do editor livremente! Prova disso é que existem dezenas versões da plaquinha: **FreeDuino, Severino, Lilypad Arduino** ...

Download (Linux/Windows/MacOSX) → **http://arduino.cc**

Conhecendo a Interface

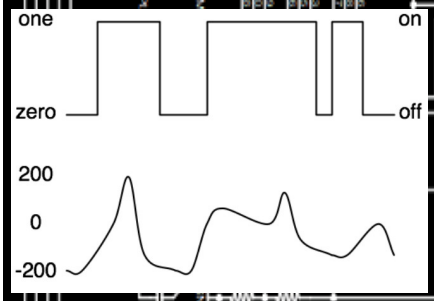
- Pinos digitais 2-13 (verde)
- Pinos digitais (RX, TX) 0,1 (verde)
- Reset (azul)
- Pinos analógicos (azul)
- Alimentação 5v e 9v (laranja)
- Terra (verde e laranja)
- Alimentação Externa 9v – 12v (rosa)
- USB (amarelo)
- Jumper para mudar alimentação do Diecimila (roxo)



%%% Introduzindo Conceitos %%%

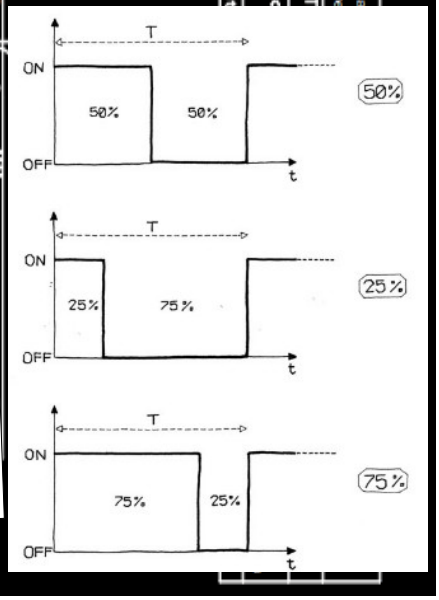
Tudo bem. Quer dizer então que o Arduino possui entradas e saídas digitais e analógicas. Mas o que isso significa?

Quando usamos estes termos, estamos tratando da representação numérica de informação. No mundo que nos rodeia, no geral, a informação é analógica. Peguemos como exemplo a variação de temperatura, hora está 14 graus, hora 20, 35... Ou seja, temos muitos valores. O Arduino trabalha muito bem com informações que variem de 0 até 1024, e são nos pinos analógicos que iremos ligar a maioria dos sensores.



Atualmente, os computadores usam o sistema de representação binária para realizar cálculos, ou seja, eles reconhecem apenas valores 0 (desligado, 0 volts) ou 1 (ligado, 5 volts). Por isso temos os pinos digitais, onde ligamos coisas como botões, que possuem apenas dois estados, ligado ou desligado.

Então se computadores conhecem apenas 0 e 1, como conseguimos jogar para a saída dele valores analógicos? Bem, existe uma técnica chamada PWM (Pulse Width Modulation) que nos ajuda nisso. Oscilando rapidamente uma saída digital, entre 0 e 1, tem-se a impressão de que a quantidade de energia enviada para o circuito é variável. Essa técnica pode ser utilizada para variar a intensidade luminosa de um led, a velocidade de um motor, etc. Por exemplo, deixando 100% do tempo a saída em 1, a energia no circuito será sempre total. Deixando 50% do tempo a saída em 1, teremos a impressão de que a energia no circuito será a metade. No Arduino, as portas 9, 10 e 11 podem ser usadas como PWM.



&&& Conversando com o Arduino &&&

E agora, sabendo dessas coisas, como faço para o Arduno desempenhar as tarefas de que Preciso? Assim como quando precisamos conversar com outros povos, para conversar com o Arduino precisamos aprender a sua linguagem. Por parão, linguagens de algo nível para Computadores são baseadas no inglês, veja algumas construções que o Arduino entende:

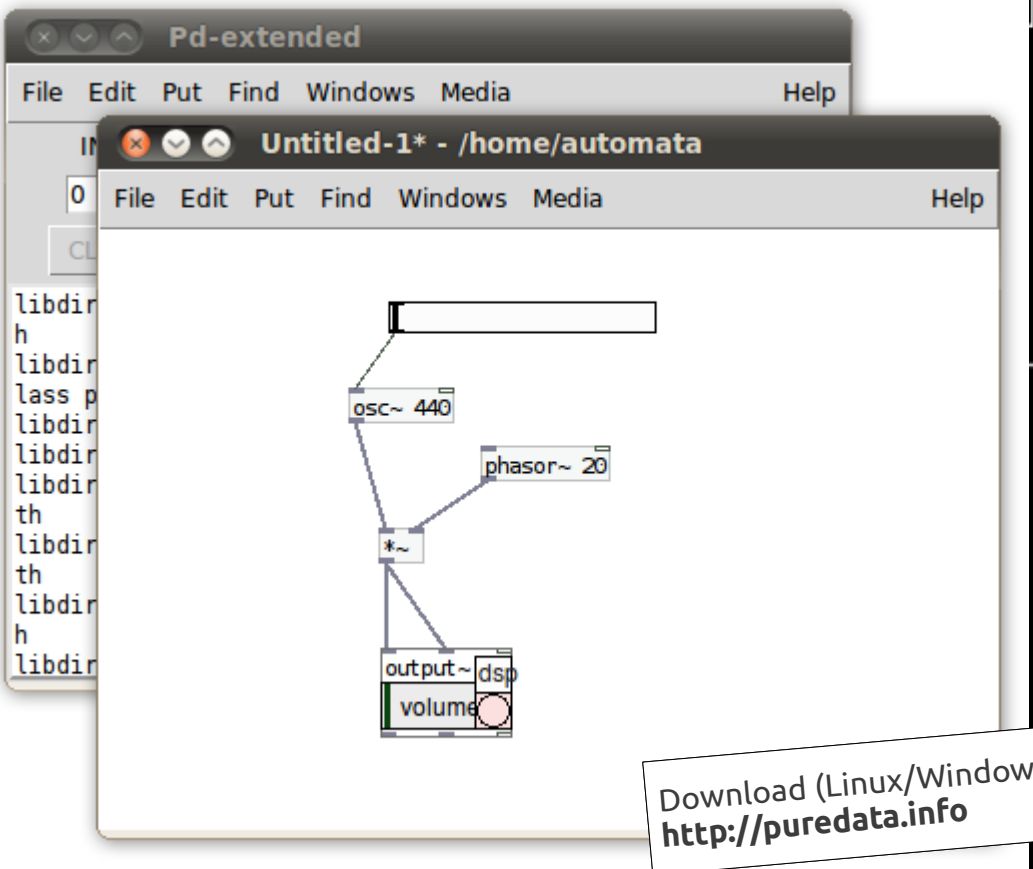
- **setup()**
  Executado somente uma vez quando o microcontrolador é ligado
- **loop()**
  Roda repeditamente o programa dentro desse bloco
- **pinMode(***<pino>, <INPUT/OUTPUT>***)**
  Configura um pino como antrada ou como saída
- **digitalWrite(***<pino>, <HIGH/LOW>***)**
  Configura o estado de uma saída digital como HIGH ou LOW
- **digitalRead(<pino>)**
  Lê o etado de uma entrada digital
- **analogWrite(<pino>, <valor: 0 – 255>)**
  Escreve um valor em uma saída analógica
- **analogRead(<pino>)**
  Lê o estado de uma entrada analógica
- **delay(<n>)**
  Pausa o processamento durante n milésimos de segundo
- **random(<inicio>, <fim>)**
  Retorna um número entre inicio e fim
- **Serial.begin(9600)**
  Inicializa a comunicação serial com um determinado *boundrate*
- **Serial.print(mensagem, <HEX/DEC/BIN/BYTE>)**
  Envia uma mensagem, do tipo especificado para a porta serial
- **Serial.read()**
  Lê da porta serial

[[[Pure Data]]]

Muitos já ouviram falar de **linguagem de programação** quando falamos em computação. Os nomes **Java, C++, Python**, podem até serem conhecidos por alguns. Todas elas são exemplos de linguagens de programação.

Pure Data também é uma linguagem de programação, porém, ao invés dessas linguagens que citamos, que se baseiam na escrita de textos (que chamamos de **código-fonte** ou simplesmente **código**), **Pure Data** não lida com textos, mas com objetos gráficos. É por isso que a chamamos de **Linguagem de programação visual.**

Pure Data foi desenvolvido por **Miller Puckette** com o objetivo de produzir e processar sinais de áudio, porém, atualmente para a produção e processamento de qualquer tipo de sinal. Ou seja, podemos tanto trabalhar com sons quanto com gráficos e animações.

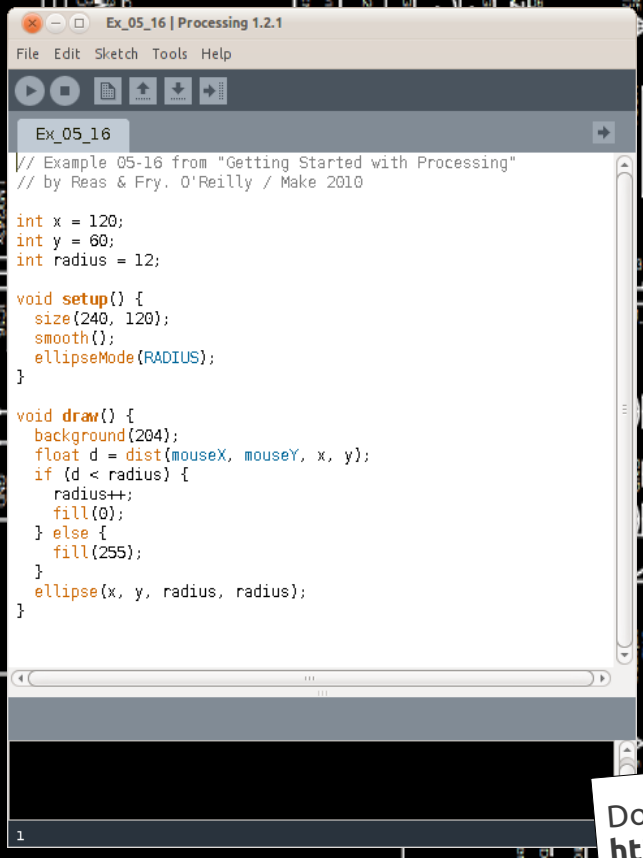


Download (Linux/Windows/MacOSX) → **http://puredata.info**

[[[Processing]]]

Processing é um ambiente e uma linguagem de progamação *open source* para pessoas interessadas Em criar imagens, animações e interações. Inicialmente foi desenvolvida para ensinar os fundamentos da programação de computadores dentro de um contexto visual, mas acabou evoluindo de tal forma a torna-se uma ferramenta completa, possibilitando criar trabalhos de nível profissional. Atualmente, existem milhares de estudantes, artistas, designers, pesquisadores e curiosos utilizando o e **Processing** para aprender, prototipar e produzir.

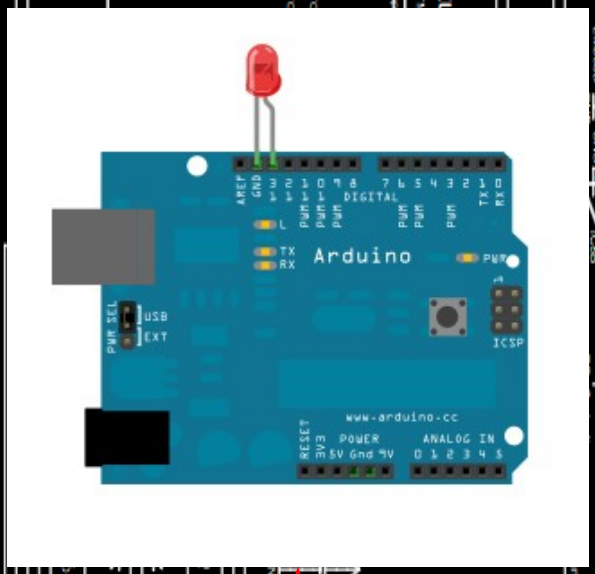
Processing foi iniciado por **Ben Fry** e **Casey Reas**. O desenvolvimento É feito por um pequeno grupo de voluntários. Veja a lista completa → http://www.processing.org/about/people/



Download (Linux/Windows/MacOSX) → **http://processing.org**

__Piscando LEDs__

Na maioria das linguagens de programação, o primeiro programa que você aprende a escrever é aquele que imprime *"Olá mundo"* na tela do computador. Como uma placa **Arduino** não tem tela, em vez disso fazemos um LED piscar :-)



Os LEDs (ou *Light Emitting Diode*, ou Diodo Emissor de Luz) têm polaridade, isto é, eles apenas acendem se você orientar as suas perninhas (terminais) corretamente. A perna mais longa é normalmente o positivo (+) que vai conectado ao **pino 13**. A perna mais curta conecta-se ao **GND (TERRA)**; o corpo do LED normalmente terá um pequeno chanfro achatado indicando essa perna. Se o LED não acender, tente inverter os terminais (você não danificará o LED se ligá-lo invertido por um curto período... **não tenha medo...**).

Monte o circuito
No **Arduino**

Copie no editor
do **Arduino**

Diodo Emissor
de Luz

piscando.pde

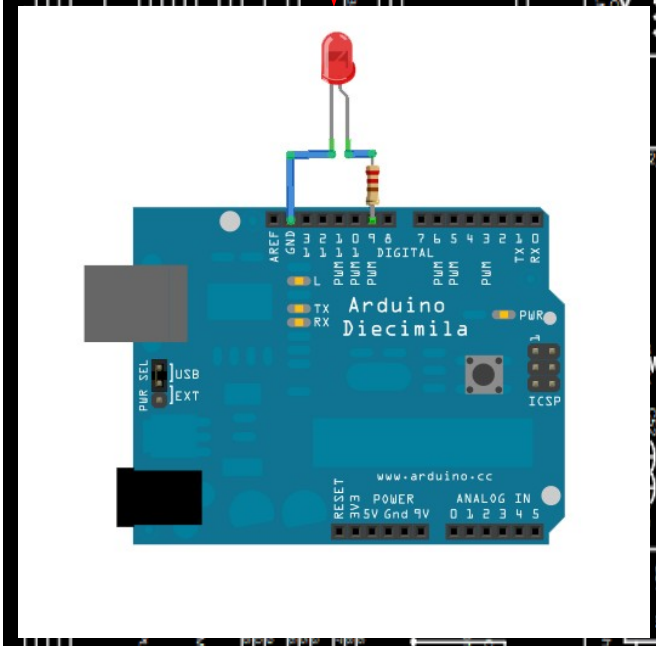
```
int ledPin = 13;

void setup(){
  pinMode(ledPin, OUTPUT);
}

void loop(){
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

_Brilhando LEDs__

Ao invés de fazer o LED somente ligar e desligar, podemos fazê-lo brilhar. Usamos uma saída um pouco diferente do *Arduino*, um pino de saída digital que chamamos de PWM. Replique o circuito anterior, só que agora, ao invés de usar o pino 13, use o **pino 9**, que permite saída PWM e grave o código no **Arduino** para testá-lo.

Monte o circuito No **Arduino**

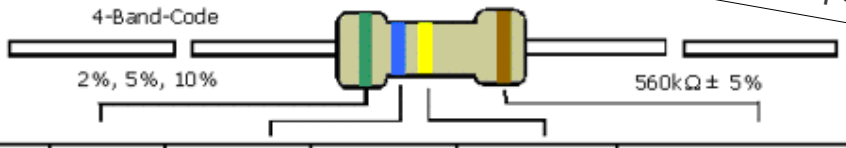
Copie no editor do **Arduino**



brilha.pde

```
int value = 0;
int ledpin = 9;

void setup(){
      pinMode(ledpin, OUTPUT);
}

void loop(){
  for(value = 0; value <= 255; value+=5){
    analogWrite(ledpin, value);
    delay(30);
  }
  for(value = 255; value >=0; value-=5){
    analogWrite(ledpin, value);
    delay(30);
  }
}
```

Os resistores possuem anéis coloridos. Eles servem para indicar o seu valor



COLOR	1st BAND	2nd BAND	3rd BAND	MULTIPLIER	TOLERANCE	
Black	0	0	0	1Ω		
Brown	1	1	1	10Ω	± 1%	(F)
Red	2	2	2	100Ω	± 2%	(G)
Orange	3	3	3	1KΩ		
Yellow	4	4	4	10KΩ		
Green	5	5	5	100KΩ	±0.5%	(D)
Blue	6	6	6	1MΩ	±0.25%	(C)
Violet	7	7	7	10MΩ	±0.10%	(B)
Grey	8	8	8		±0.05%	
White	9	9	9			
Gold				0.1	± 5%	(J)
Silver				0.01	± 10%	(K)

Power Supply

Wave Bubble mainboard
Creative Commons 2.5 Attrib/ShareAlike
TITLE: cigpack-main rc1a
Document Number:
http://www.ladyada.net/make/wavebubble
Date: 3/15/2007 08:13:50p
Sheet: 1/1
REV: RC 1a

__Botões__

O interruptor momentâneo (popular botão) é um componente que conecta dois pontos de um circuito ao pressioná-lo. O exemplo a seguir liga um LED Quando precionamos o botão.

Botão

Quando o botão está livre (não pressionado), não há conexão entre as suas duas pernas, de forma que o pino do **Arduino** está conectado aos **5V** e ao ler o pino, obtemos **HIGH.** Quando o botão é fechado (pressionado), ocorre a conexão entre suas pernas, de forma que o pino do *Arduino* é ligado e obtemos **LOW** (O pino ainda se mantém conectado aos **5 volts**, mas o resistor de *pull-up* faz com que o pino esteja mais próximo do GND).

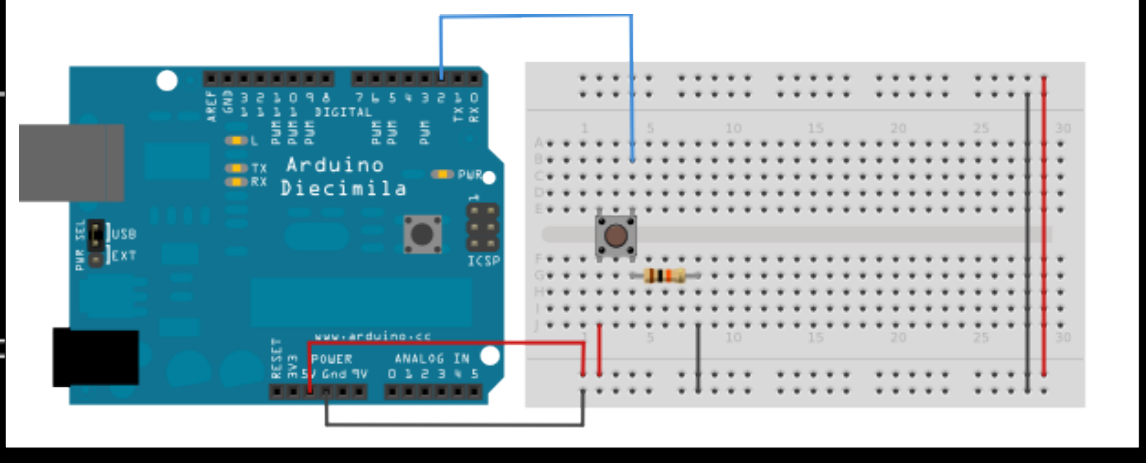
Se o pino digital for desconectado da montagem, o LED poderá piscar de forma irregular. Isto porque dizemos que a entrada está *flutuando* – isto é, estará entre valores de tensão elétrica aleatórios entre HIGH e LOW. É por isso que utiliza-se um resistor de *pull-up* ou *pull-down* no circuito.

Copie no editor do **Arduino**

botao.pde

```
int ledPin = 13;
int inputPin = 2;
int val = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(inputPin, INPUT);
}

void loop(){
  val = digitalRead(inputPin);
  if (val == HIGH) {
    digitalWrite(ledPin, LOW);
  } else {
    digitalWrite(ledPin, HIGH);
  }
}
```

Monte o circuito No **Arduino**



__Potenciômetros__

Um potenciômentro é um simples botão giratório que fornece uma resistência variável e que pode ser *lida* pelo **Arduino** como um valor analógico. No exemplo a seguir, esse valor é usado para controlar a taxa que o LED estará piscando.

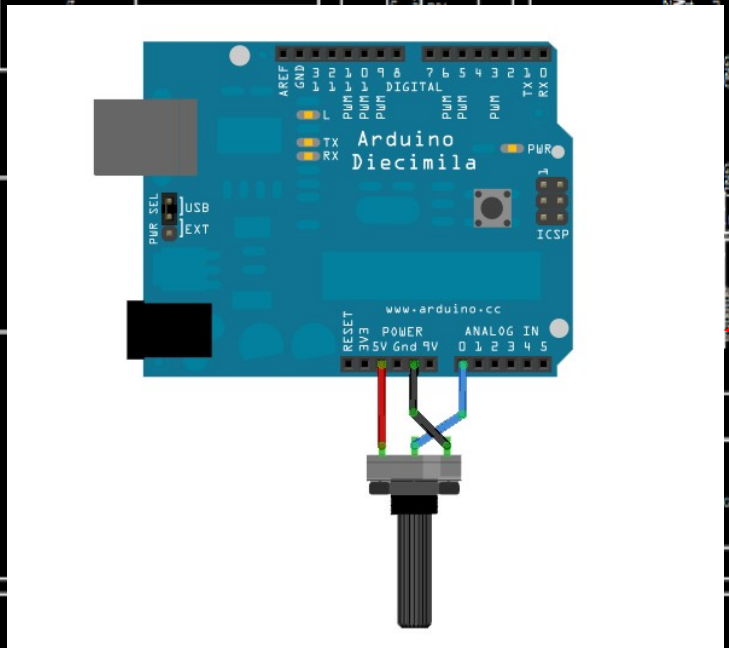
Potenciômetro

Conectamos três fios à placa **Arduino**. O primeiro vai no GND (terra) do Arduino a partir da Perna esquerda do potenciômetro (ele possui geralmente três pernas). O segundo fio será Para **5 volts** do Arduino à perna direita. O terceiro e último fio será colocado na **entrada Analógica 0** à perna central do potenciômetro.

Se girarmos o potenciômetro, alteramos a resistência em cada lado do contato elétrico que vai conectada à sua perna central. Isso provoca a mudança na *proximidade* da perna central aos **5 votls** ou ao **GND (terra)**, o que implica numa mudança no valor analógico de entrada. Quando o potenciômetro for levado até o final da escala, teremos por exemplo zero volts a ser fornecido ao pino de entrada da escala, haverá 5 volts a ser fornecido ao pino do Arduino e, ao lê-lo, teremos 1023. Em qualquer posição intermediária do potenciômetro, teremos um valor entre **0** e **1023**, que será proporcional à tensão elétrica sendo aplicada Ao pino do *Arduino*.

potenciometro.pde

```
int ledPin = 9;
int potPin = 0;
int value = 0;

void setup(){
  pinMode(ledPin, OUTPUT);
  pinMode(potPin, INPUT);
}

void loop(){
  value = analogRead(potPin);
  delay(100);
  analogWrite(ledPin, value/4);
}
```



Copie no editor do **Arduino**

Monte o circuito No **Arduino**

__LDR__

LDR – *Ligh Dependent Diode* faz o inverso que o LED. Ambos são diodos, mas ao Invés de emitr luz como fazem os LEDs, o LDR a recebe! Seu comportamento é Muito parecido com o potênciomentro, conforme a intensidade da luz Aumentar/diminuir também irá aumentar/diminuir a sua resistência, alterando a Tensão do circuito e, por conseguinte, permitindo que tenhamos 1024 valores Diferentes. Nada melhor do que testar seu comportamento! Replique o circuito Seguinte e copie o código no editor do Arduino.

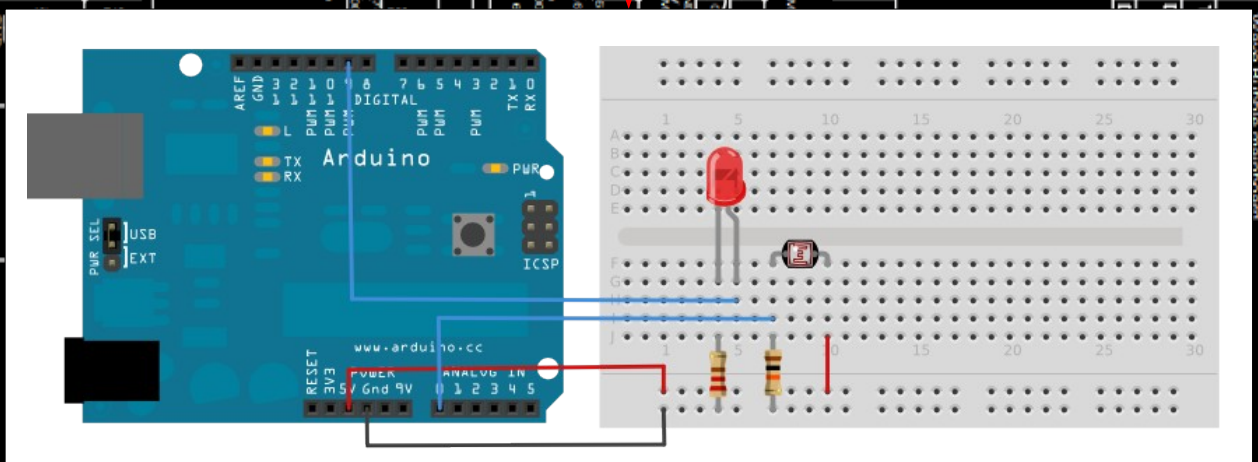
LDR

ldr.pde

```
int ledPin = 9;
int ldrPin = 0;
int value = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(ldrPin, INPUT);
}

void loop(){
  value = analogRead(ldrPin);
  delay(100);
  analogWrite(ledPin, value/4);
}
```

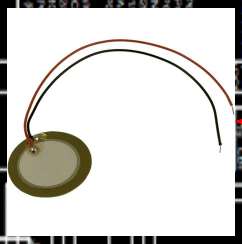
Copie no editor do **Arduino**

Monte o circuito No **Arduino**



__Piezo como entrada__

Potenciômetros variam sua resistência quando giramos seu botão. LDRs variam a Resistência conforme a intensidade de luz. Piezoelétricos são cristais geram uma Tensão conforme a pressão incidida sobre eles. Podemos usá-los para capturar Vibrações no ambiente. São também úteis como saída: fazendo-os vibrar em uma Determinada frequência, podemos produzir sons! Mas vamos primeiramente usá-los Como entrada, para isso, replique o circuito e copie o código!

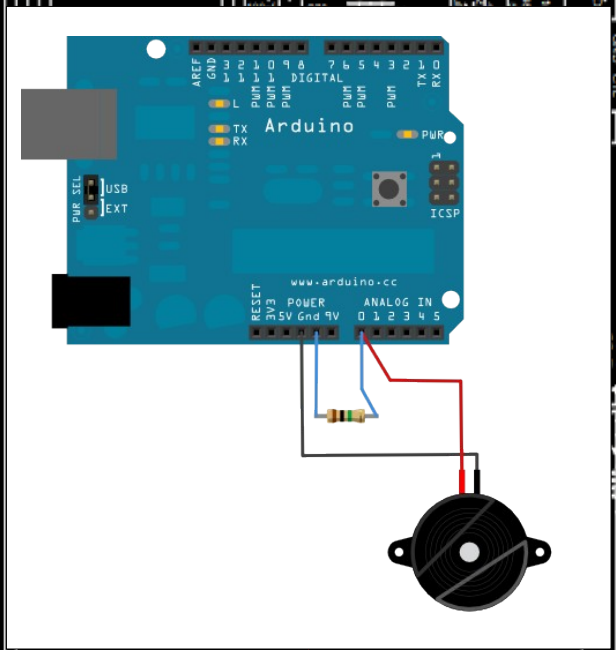


Piezo

Copie no editor do **Arduino**

piezo_entrada.pde

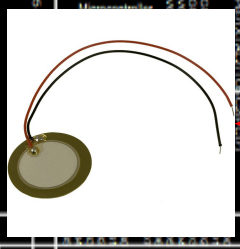
```
int ledPin = 13;
int knockSensor = 0;
byte val = 0;
int statePin = LOW;
int THRESHOLD = 100;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  val = analogRead(knockSensor);
  if(val >= THRESHOLD){
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    delay(10);
  }
}
```



Monte o circuito No **Arduino**

__Piezo como saída__

Agora sim vamos usar o piezoelétrico como saída. Replique o circuito e o código em seu mundo real e abstrato. Gere som. Abstrato? Real?



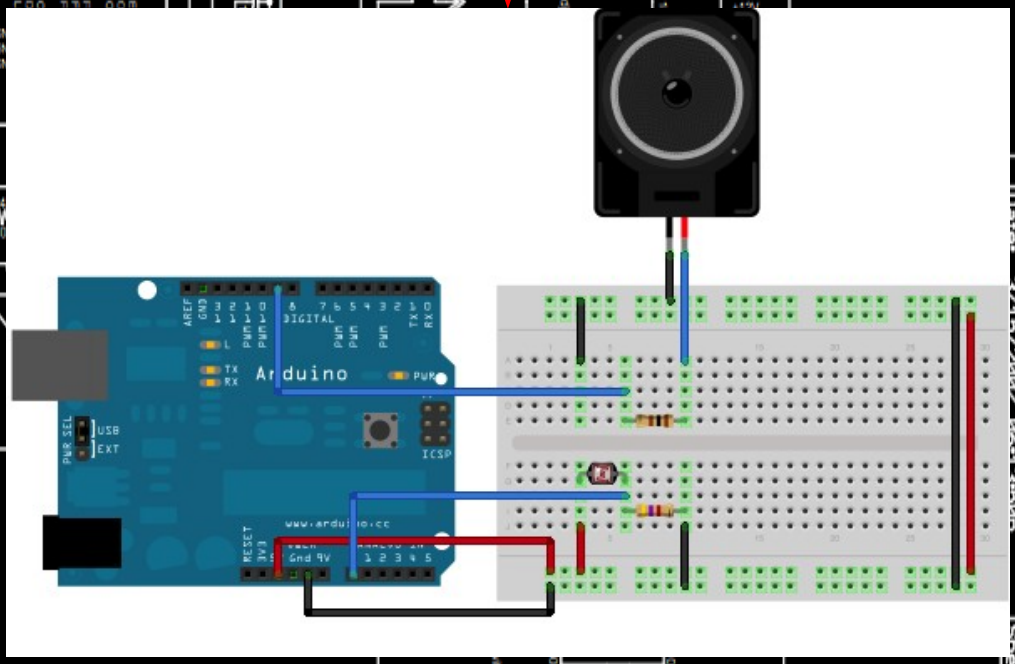
Piezo

piezo_saida.pde

```
int piezo = 9;
int ldr = 0;

void setup(){
}

void loop(){
  int sensorReading = analogRead(ldr);
  int pitch = map(sensorReading, 400, 1000, 100, 1000);
  tone(piezo, pitch, 10);
}
```

Copie no editor do **Arduino**

Monte o circuito No **Arduino**



__ Piscando LEDs + PD __

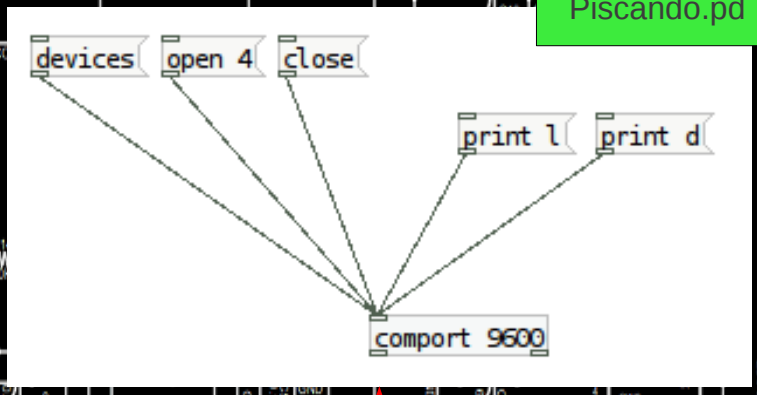
Podemos utilizar o computador para controlar o Arduino. Para isso geralmente usamos Uma linguagem de programação. Aqui usaremos *Pure Data*, um **linguagem de Programação visual.** Para isso temos que fazer o **código** (programa) que executa no Arduino se comunicar com o código que é executado no computador, em *Pure Data*

Vamos aproveitar o mesmo circuito que você construiu para piscar o LED, mas agora modificando o código *Arduino*. Também usaremos um **código e**m *Pure Data*, que chamamos de *patch*. Abre seu *Arduino* e seu *Pure Data* E copie os repectivos códigos.

piscando_pd.pde

```
int ledPin = 13;

void setup(){
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  int msg = Serial.read();
  if((char)msg == 'l'){
    digitalWrite(ledPin, HIGH);
  }
  else if((char)msg == 'd'){
    digitalWrite(ledPin, LOW);
  }
}
```

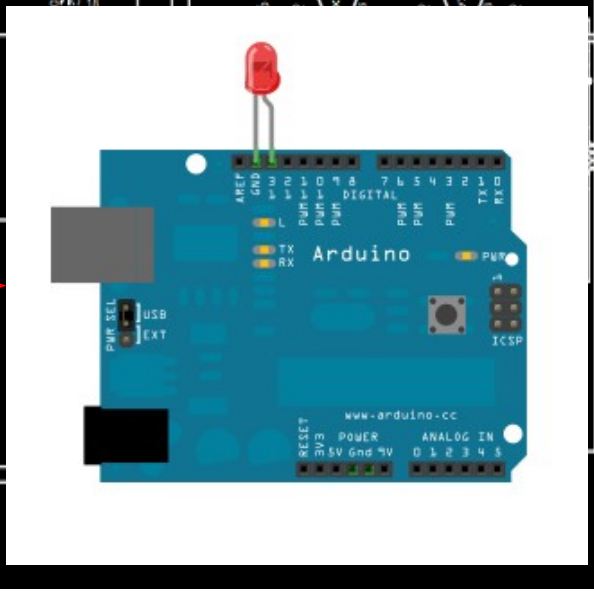
Piscando.pd



Monte o circuito No **Arduino**

Copie no editor do **Arduino**

Copie no editor do **Pure Data**



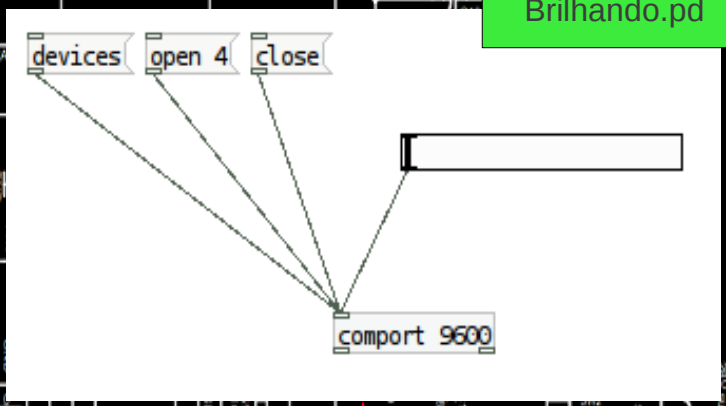
__ Brilhando LEDs + PD __

Assim como fizemos um LED piscar com *Pure Data*, podemos fazê-lo brilhar também. aproveite o mesmo circuito que você criou para fazer um led brilhar (conectando o positivo do LED no **pino 9** do Arduino e o negativo no GND). Modifique apenas o código no Arduino e replique o *patch* abaixo em Pure Data.

brilhando_pd.pde

```
int ledPin = 9;

void setup(){
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  int msg = Serial.read();
  if(msg >= 0){
    analogWrite(ledPin, msg);
  }
}
```

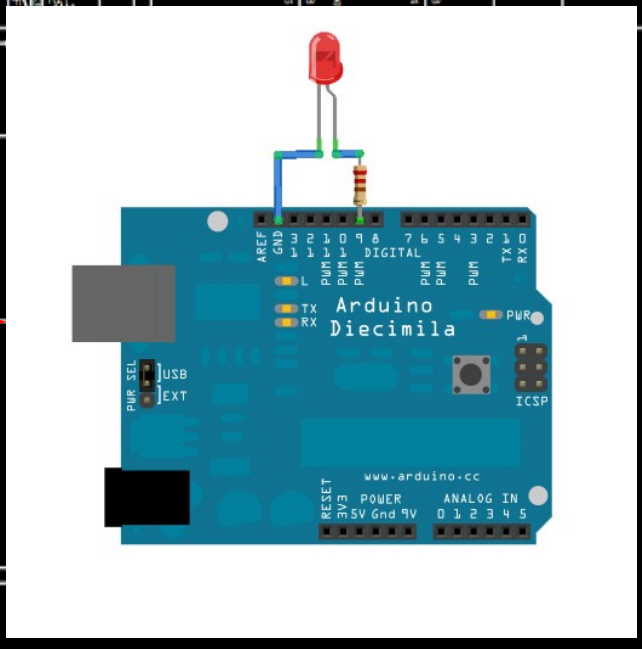
Brilhando.pd



Copie no editor do **Arduino**

Copie no editor do **Pure Data**

Monte o circuito No **Arduino**



__ Botão + PD __

Da mesma maneira que fizemos para os LEDs, podemos não só controlar o Arduino com O computador, mas também fazer o inverso: controlar o computador com o Arduno! Para isso, novamente, cole os códigos abaixo no Arduino e no Pure Data, aproveitando O mesmo circuito que você montou para o botão.

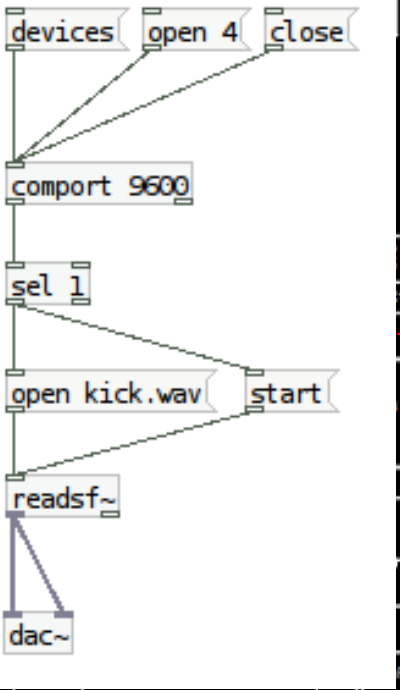
Copie no editor do **Pure Data**

Pure Data fornece inúmeros objetos (as caixinhas...) para lidarmos com todo tipo de sinal (sinal de áudio, sinal de vídeo, ...). O objeto que estamos usando nesse *patch*, por exemplo, faz a leitura de um arquivo *WAV* qualquer. Com um pouco mais de trabalho, podemos criar facilmente uma *drum machine*.

botao_pd.pde

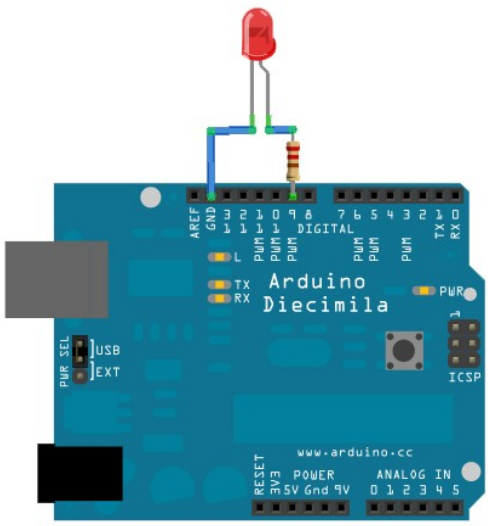
```
int buttonPin = 2;

void setup(){
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}

void loop(){
  int x = digitalRead(buttonPin);
  Serial.write(x);
}
```

botao.pd



Copie no editor do **Arduino**

Monte o circuito No **Arduino**



__ Potenciômetro + PD__

Novamente, vamos incluir *Pure Data* e o computador na história. Monte o circuito para o potenciômetro que você já viu e cole os respectivos **código-fonte** e **patch** no Arduino e Pure Data. O que estamos fazendo é usando o potenciômetro para controlar o raio de uma esfera desenhada na tela do computador. Podemos usar essa valor lido do potenciômetro para modificar qualquer parâmetro de objetos em *Pure Data* (cor do objeto, posição na tela, rotação, tamanho, ...).

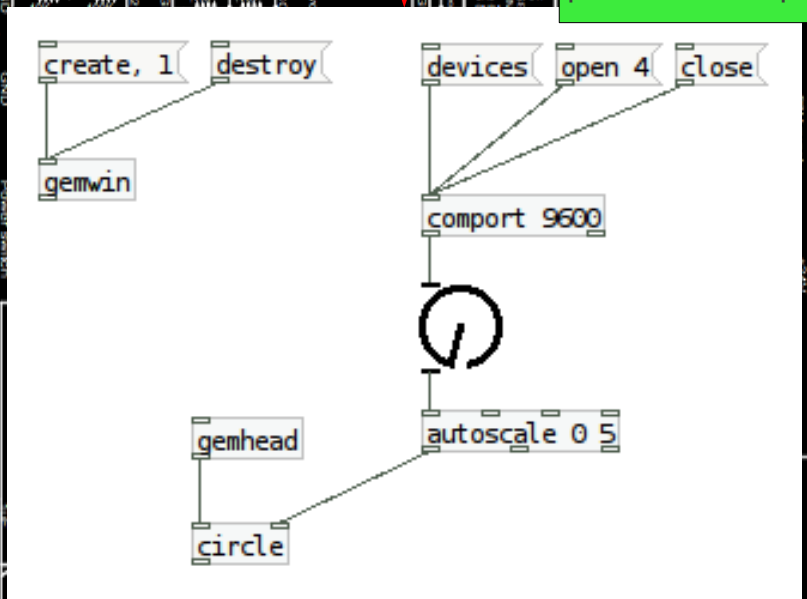
potenciometro_pd.pde

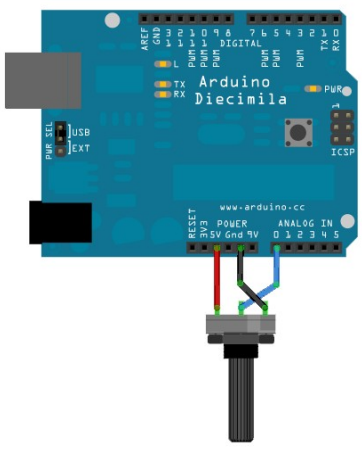
```
int potPin = 0;
Int value = 0;

void setup(){
  pinMode(potPin, OUTPUT);
  Serial.begin(9600);
}

void loop(){
  Value = analogRead(potPin);
  Serial.write(map(value, 0, 1023, 0, 127));
}
```

Copie no editor do **Arduino**

Copie no editor do **Pure Data**

potenciometro.pd





Monte o circuito No **Arduino**

__ LDR + PD __

Vamos misturar novamente Arduino e Pure Data. Monte novamente o circuito para usar o LDR e cole os códigos abaixo no Arduino e Pure Data.

Ao invés de reproduzirmos um arquivo de áudio, agora estamos usando o computador para criar (sintetizar) o som! Usamos o LDR ligado no Arduino para modificar a frequência dessa onda sonora. Porém, podemos utilizar outros objetos de *Pure Data* que lidam com áudio para gerarmos qualquer som que desejarmos.

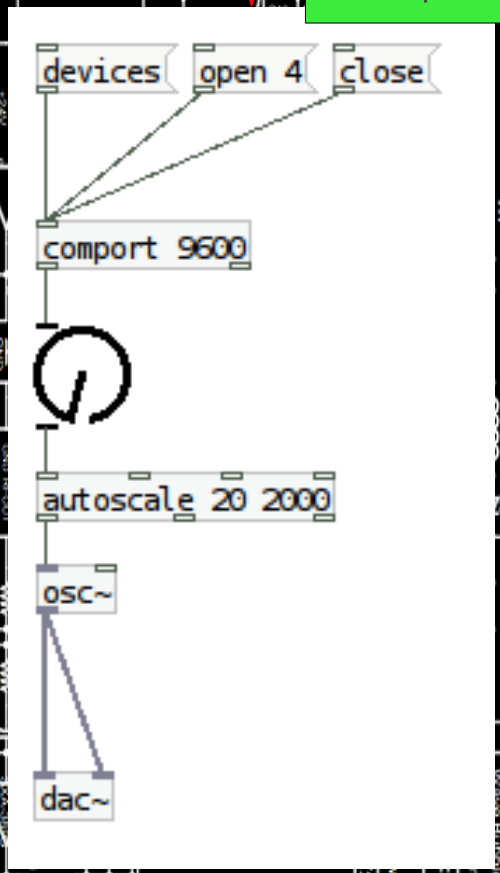
ldr_pd.pde

```
int entrada = 0;
int valor = 0;

void setup(){
  pinMode(entrada, INPUT);
  Serial.begin(9600);
}

void loop(){
  valor = analogRead(entrada);
  Serial.write(valor);
}
```

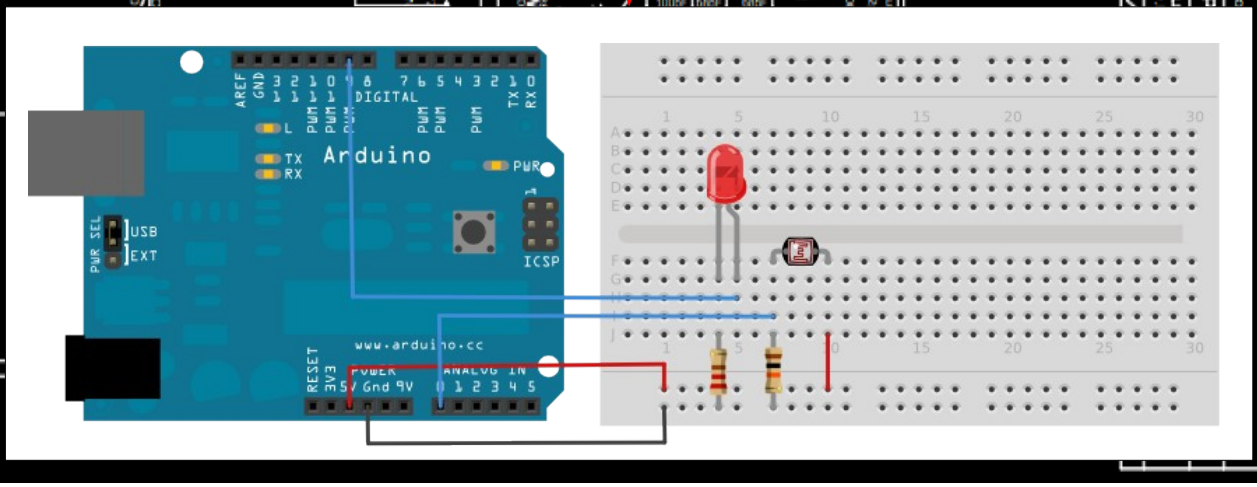
Copie no editor do **Arduino**

Copie no editor do **Pure Data**

ldr.pd



Monte o circuito No **Arduino**



__Piezo + PD__

Juntado *Pure Data* à sopa de circuitos e código ... monte o circuito que usamos para receber os valores do piezoelétrico com o *Arduino*. Replique o *patch* em *Pure Data* e veja o que acontece ...

piezo_pd.pde

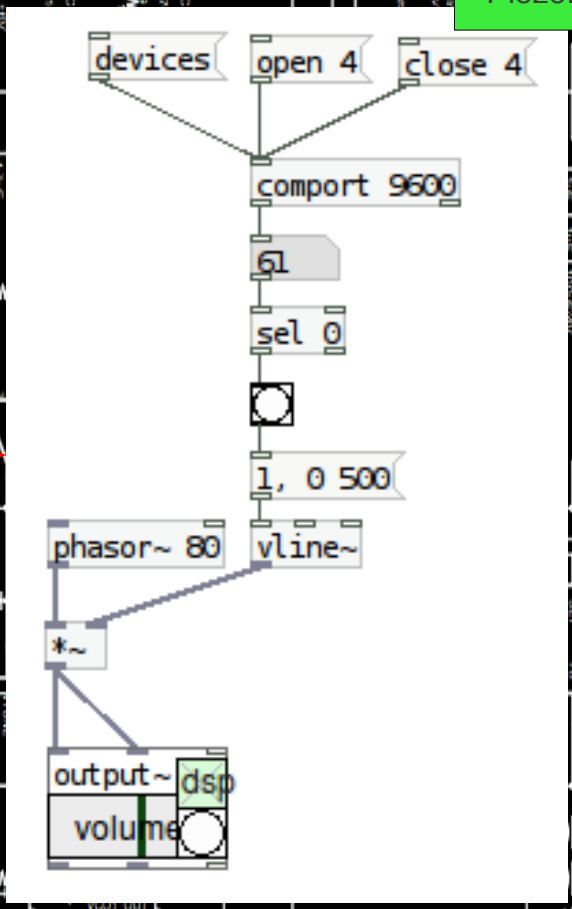
```
int entrada = 0;
int valor = 0;

void setup(){
  pinMode(entrada, INPUT);
  Serial.begin(9600);
}

void loop(){
  Valor = analogRead(entrada);
  Serial.write(valor);
}
```

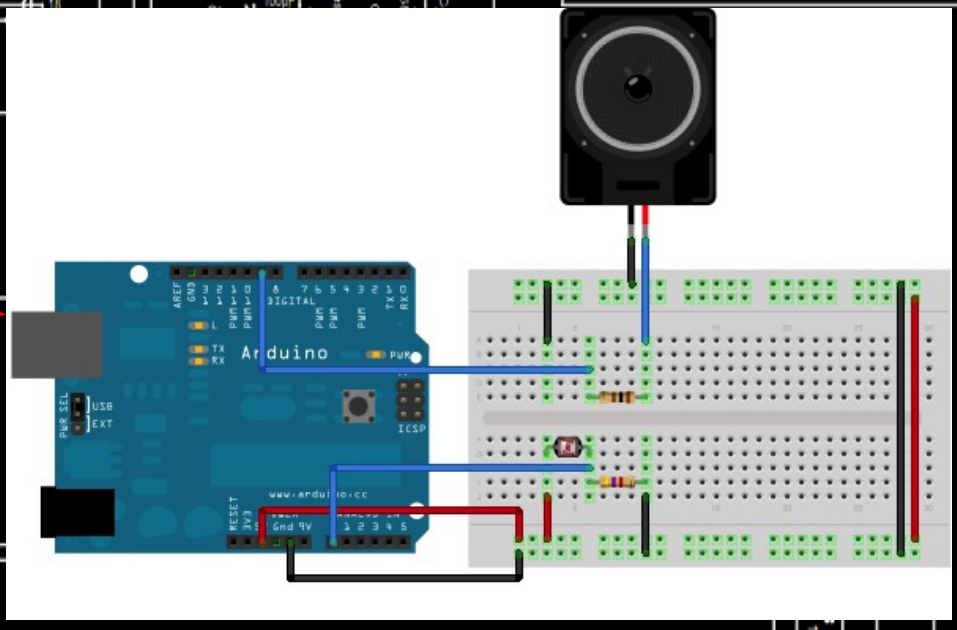
Copie no editor do **Arduino**

Monte o circuito No **Arduino**

Monte o circuito No **Arduino**

Piezo.pd

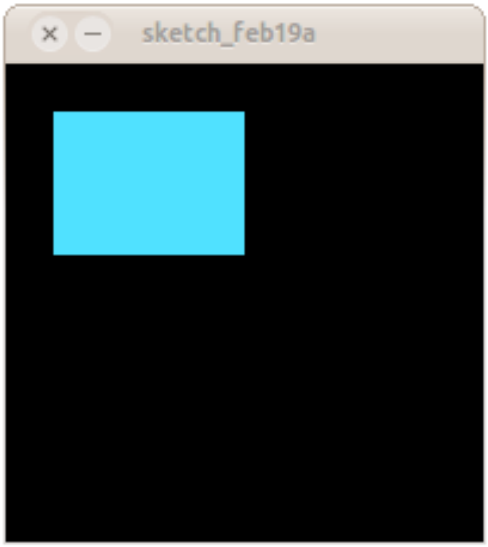




__Processing__

Agora que já sabemos como funciona o Arduino e Pure Data, que tal aprender um pouco sobre Processing e sua ligação com o Arduino? Nada melhor do que colocar a mão na massa, então vamos lá, primeramente aprenderemos a utilizar apenas o Processing e onde procurar informações.

```
size(200,200);
background(0);

noStroke();
fill(80,225,255);
rect(20,20,80,60);
print("Ola Pessoas!");
```

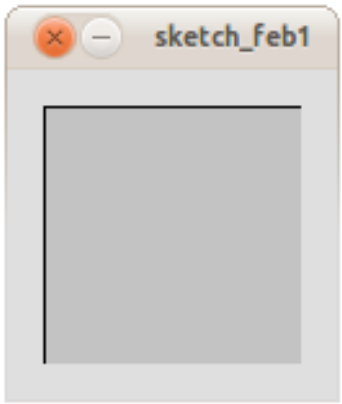


Copie no editor do **Processing**

Mais exemplos em →
http://www.processing.org/learning/

Referencia de funções →
http://www.processing.org/reference/

```
int value = 0;

void setup(){
  size(100, 100);
}

void draw() {
  fill(value);
  rect(0, 0, 100, 100);
}

void mouseMoved() {
  value = value + 5;
  if (value > 255) {
    value = 0;
  }
}
```



__LED + Processing__

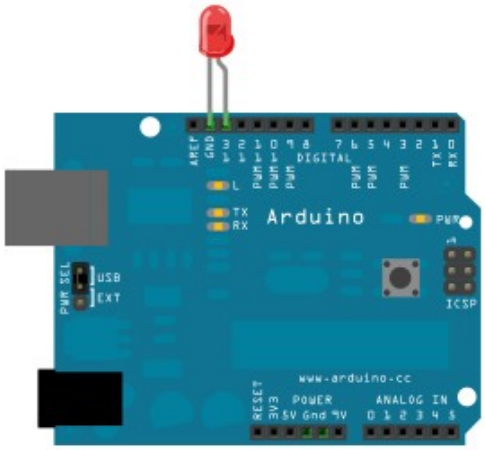
Agora que sabemos como funciona o Processing, vamos exemplificar sua interligação com o Arduino e acender um LED.

Copie no editor do **Processing**

Copie no editor do **Arduino**

```
const int ledPin = 0;
int byteEntrada;

void setup(){
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop(){
  if (Serial.available() > 0){
    byteEntrada = Serial.read();
    if (byteEntrada == 'H'){
      digitalWrite(ledPin, HIGH);
    }
    if (byteEntrada == 'L'){
      digitalWrite(ledPin, LOW);
    }
  }
}
```

```
import processing.serial.*;

float boxX;
float boxY;
int boxSize = 20;
boolean mouseOverBox = false;

Serial port;

void setup(){
  size(200,200);
  boxX = width/2.0;
  boxY = height/2.0;
  rectMode(RADIUS);

  println(Serial.list());

  port = new Serial(this, Serial.list()[0], 9600);
}

void draw(){
  background(0);

  if(mouseX > boxX - boxSize && mouseX < boxX + boxSize &&
    mouseY > boxY - boxSize && mouseY < boxY + boxSize){
    mouseOverBox = true;
    stroke(255);
    fill(153);
    port.write('H');
  }
  else{
    stroke(153);
    fill(153);
    port.write('L');
    mouseOverBox = false;
  }
  rect(boxX, boxY, boxSize, boxSize);
}
```



Monte o circuito No **Arduino**

__Potenciômetro + Processing__

Legal, já conseguimos ligar um LED pelo Processing, que controlar o potenciômetro?
Então vamos lá :-)

Copie no editor do **Arduino**

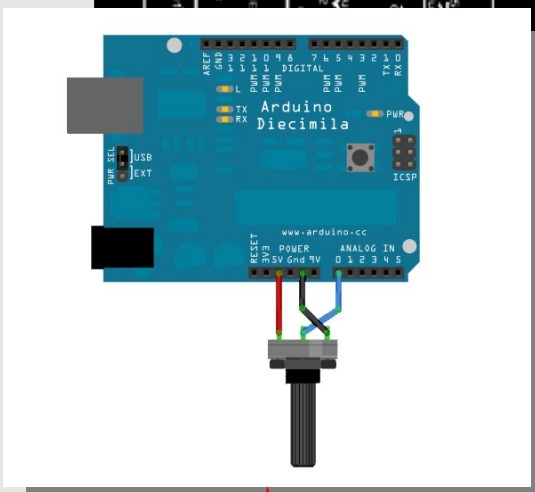
```
void setup(){
  Serial.begin(9600);
}

void loop(){
  Serial.println(analogRead(A0));
  Delay(10);
}
```

Envia o valor do pino 0 da entrada analógica.

Espera um bit para que o conversor analogico-digital possa estabilizar após a última leitura

Copie no editor do **Processing**

```
import processing.serial.*;

Serial port;
int xPos = 1;

void setup(){
  Size(400,300);

  println(Serial.list());

  port = new Serial(this, Serial.list()[0], 9600);
  port.bufferUntil('\n');
  background(0);
}

void draw(){
}

Void serialEvent(Serial port){
  String inString = port.readStringUntil('\n');

  if(inString != null){
    inString = trin(inString);
    float inByte = float(inString);
    inByte = map(inByte, 0, 1023, 0, height);

    stroke(127,34,255);
    line(xPos, height, xPos, height – inByte);

    if (xPos >= width){
      xPos = 0;
      background(0);
    }
    else{
     xPos++;
    }
  }
}
```



Monte o circuito No **Arduino**

>> Exemplos <<

O que torna o ***software/hardware livre*** interessante é que temos disponível na internet uma infinidade de projetos prontos para montarmos e sair usando. **Experimente!** Procure pelos projetos e **replique-os, modifique-os, use-os e abuse-os!**

AUDUINO → Sintetizador de áudio usando Arduino
http://code.google.com/p/tinkerit/wiki/Auduino

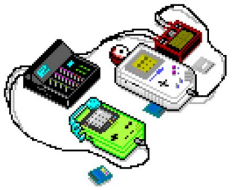
AUDUINO PUNK CONSOLE → Sintetizador e sequenciador de áudio usando Arduino
http://www.beavisaudio.com/projects/digital/ArduinoPunkConsole/

~~~ Referências ~~~

Arduino → http://arduino.cc
Pure Data → http://puredata.info
Coletivo MuSA → http://musa.cc
Artesanato de Volts → http://artesanato.devolts.org
Des).(centro → http://pub.descentro.org
Robótica Livre → http://roboticalivre.org
Metareciclagem → http://rede.metareciclagem.org
Estúdio Livre → http://estudiolivre.org
MSST → http://devolts.org/msst

__Notas__

__Notas__



>> EVENTOS <<

FISL (Junho/Julho) → http://fisl.softwarelivre.org
Latinoware (Outubro 2011) → http://latinoware.org
SoLiSC (Outubro/Novembro 2011) → http://solisc.org.br

Patrocínio

Computadores fazer ARTEEÊÊÊÊÊÊ,
artistas . . .

Feito com: