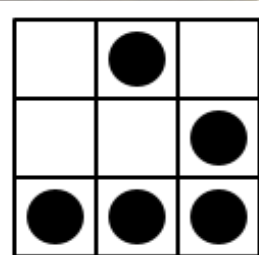
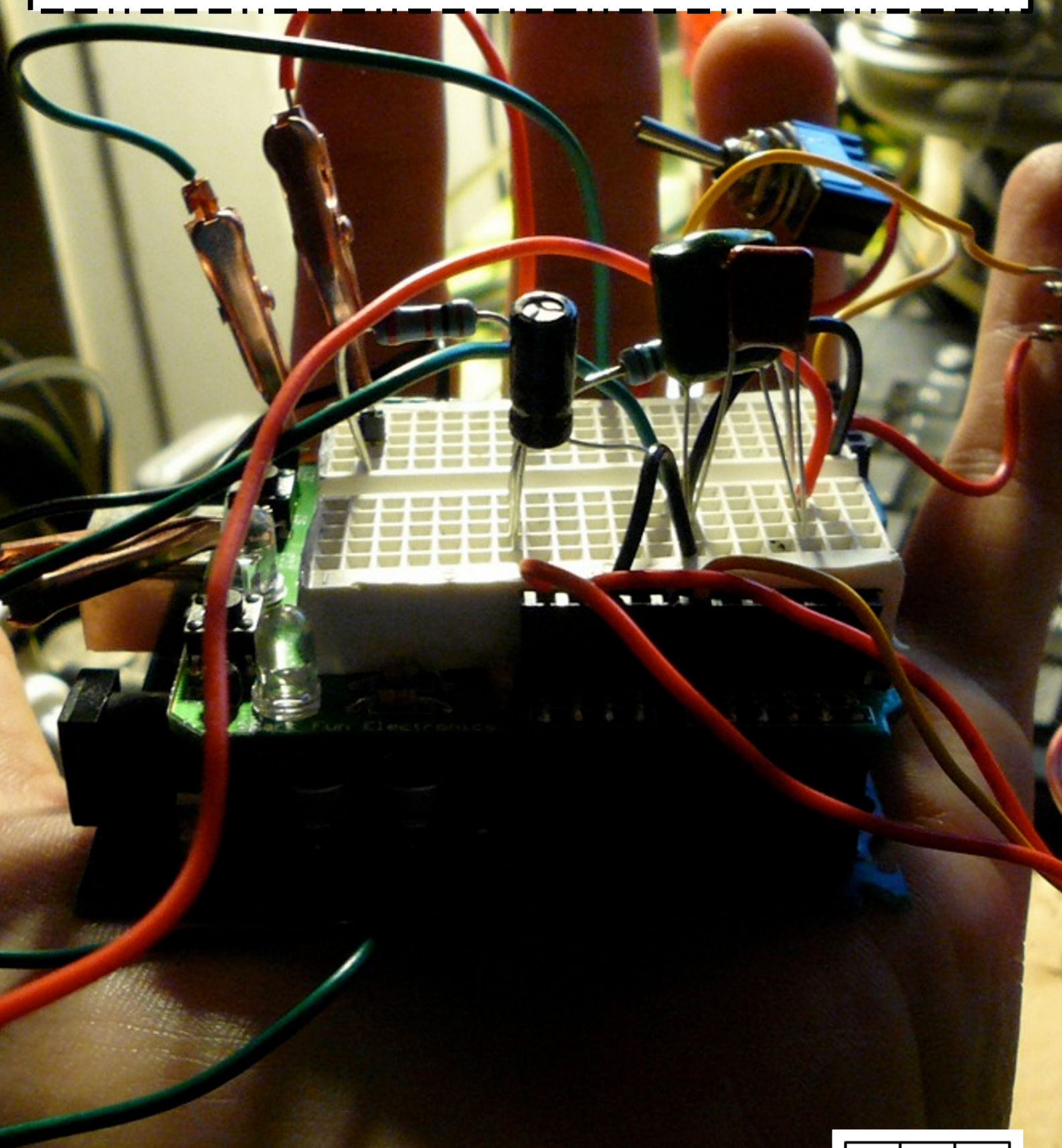


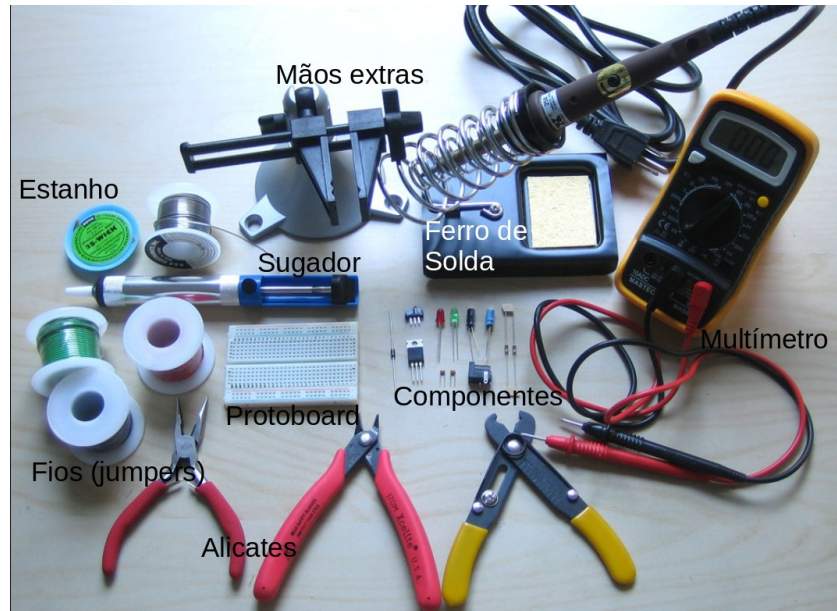
Computadores fazem arte

uma breve introdução

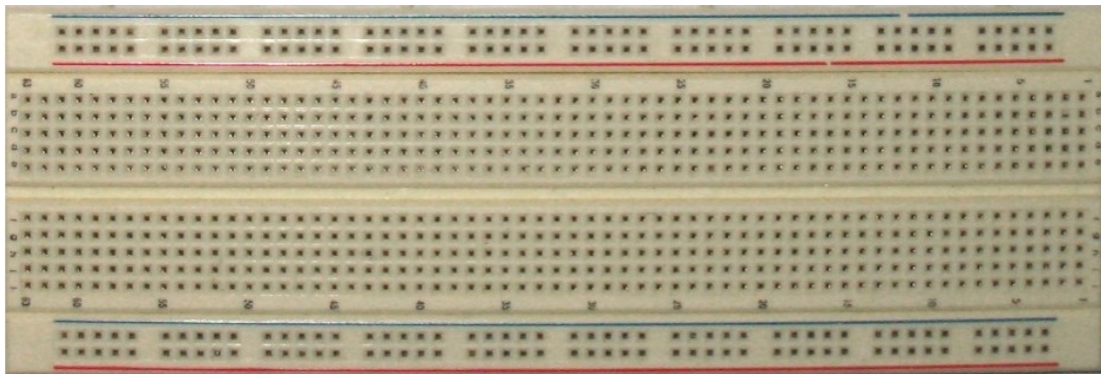


{{{ ANTES DE COMEÇAR ... }}}}

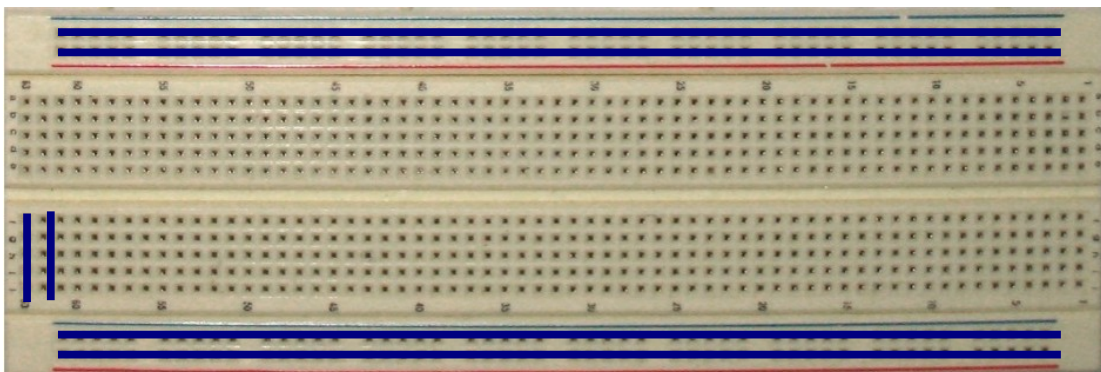
Prazer, nós seremos as suas ferramentas! Nossos nomes são:



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.

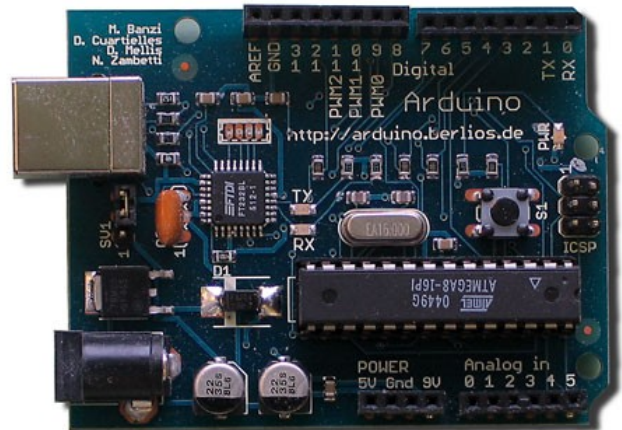


(((ARDUINO)))

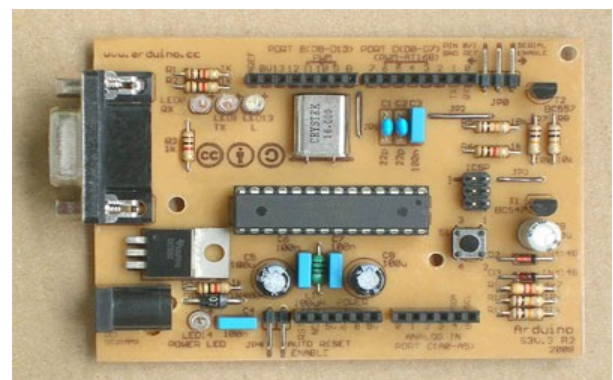


Arduino é uma iniciativa de se criar um ambiente para desenvolvimento de *hardware livre*. **Arduino** não é só a plaquinha que faz a comunicação de sensores/atuadores com o computador, mas também é uma **linguagem de programação** e um **ambiente de desenvolvimento integrado**.

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 de versões da plaquinha: **FreeDuino**, **Severino**, **Lilypad Arduino**, ...



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

```
File Edit Sketch Tools Help
[Icons]
Blink [Share]

based on an original by H. Barragan for the Wiring i/o board
*/

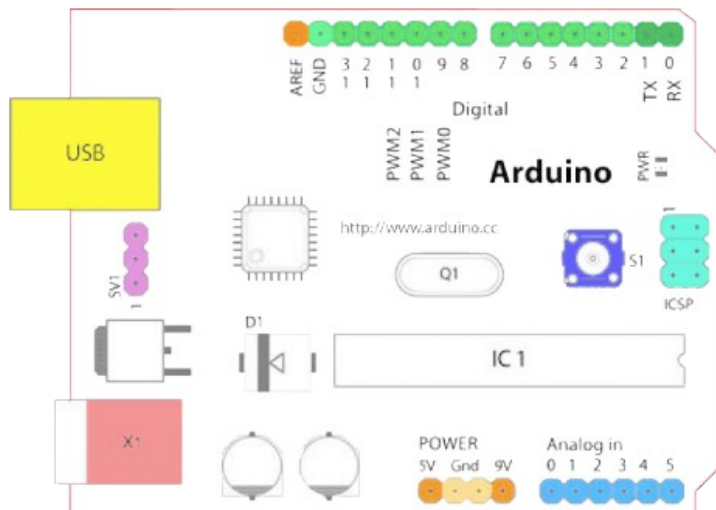
int ledPin = 13;    // LED connected to digital pin 13
// The setup() method runs once, when the sketch starts
void setup() {
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
}

// the loop() method runs over and over again,
// as long as the Arduino has power

1
```

CONHECENDO A INTERFACE

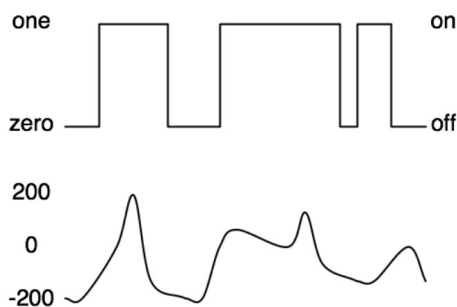
- Pinos digitais 2-13 (verde)
- Pinos digitais (RX,TX) 0,1 (verde)
- Reset (azul)
- Pinos analógicos (azul)
- Alimentação 5v e 9v (laranja)
- Terra (verde e laranja)
- Alimentação externa 9v – 12v (rosa)
- USB (amarelo)
- Jumper para mudar a alimentação do Diecimila (roxo)



%%% INTRODUZINDO CONCEITOS %%%

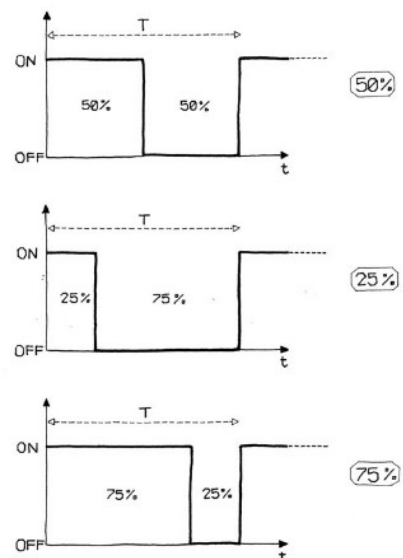
Tudo bem. Quer dizer então que o Arduino possui entradas e saídas digitais e analógicas. Mas, o que isso significa?

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 Arduino 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 padrão, linguagens de alto 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 repetidamente o programa dentro desse bloco

.pinMode(<pin>, <INPUT/OUTPUT>)

Configura um pino como entrada ou como saída

.digitalWrite(<pin>, <HIGH/LOW>)

Configura o estado de uma saída digital como HIGH ou LOW

.digitalRead(<pin>)

Lê o estado de uma entrada digital

.analogWrite(<pin>, <valor: 0-255>)

Escreve um valor em uma saída analógica

.analogRead(<pin>)

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 baudrate

.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 **linguagens 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.

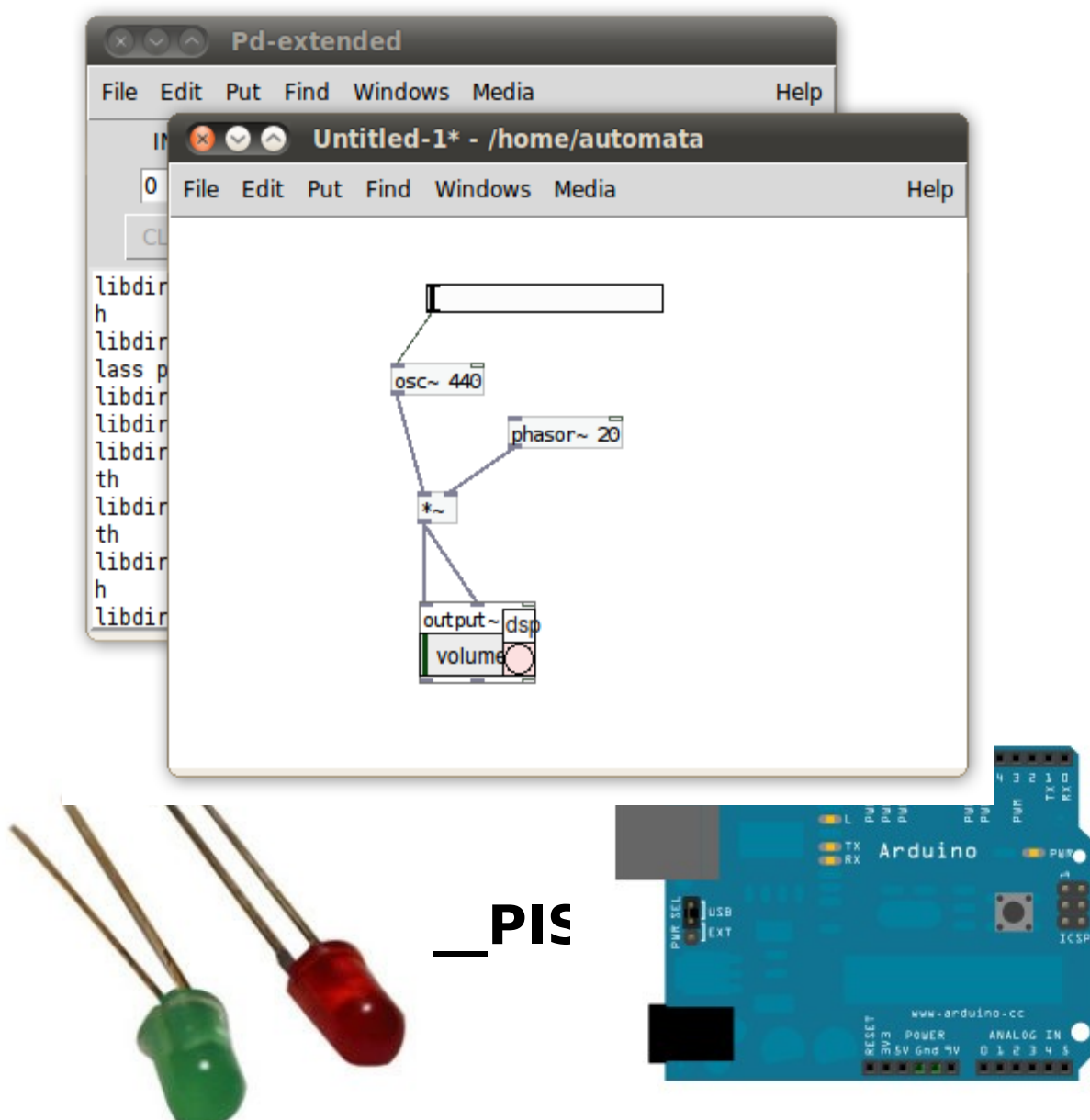


Pure Data também é uma linguagem de programação, porém, ao invés dessas linguagens que citamos, que se baseiam na escrita de textos (que chamamos de **código-fonte** ou simplesmente **código**), **Pure Data** não lida com textos, mas com objetos gráficos. É por isso que a chamamos de **linguagem de programação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

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

Misturando **Pure Data** com **Arduino**, tornamos o computador uma poderosa ferramenta para amplificar nossas capacidades humanas de expressão.

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



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 :-)

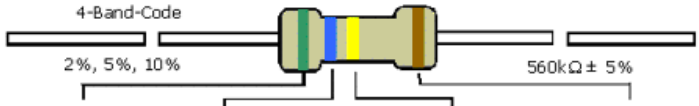
```
int ledPin = 13;

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


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

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...).

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)



_PISC

Podemos utilizar o computador para controlar o *Arduino*. Para isso geralmente usamos uma linguagem de programação. Aqui usaremos *Pure Data*, uma **linguagem de programação visual**.

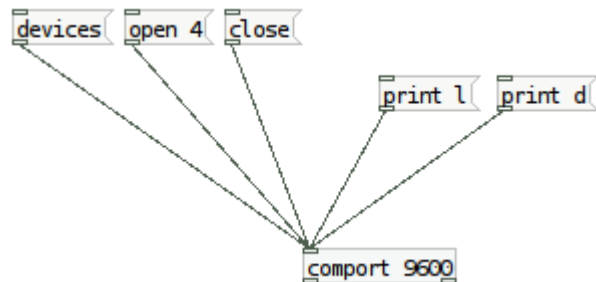
Para isso temos que fazer o **código** (programa) que executa no *Arduino* se comunicar com o código que é executado no computador, em *Pure Data*.

Vamos aproveitar o mesmo circuito que você construiu para piscar o LED, mas agora modificando o código *Arduino*. Também usaremos um **código** em *Pure Data*, que chamamos de *patch*. Abra seu *Arduino* e seu *Pure Data* e cole os respectivos códigos.

```
int ledPin = 13;

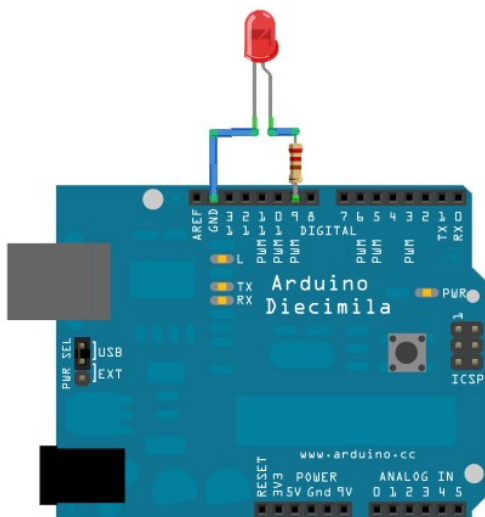
void setup(){
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

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



__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.



```
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);
  }
}
```

__JS + PD__

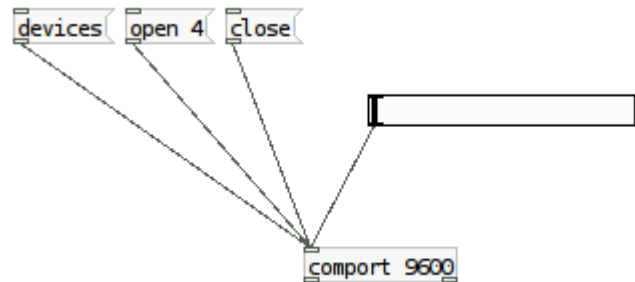
Em *Pure Data*, podemos fazê-lo brilhar também. Aproveite o mesmo led brilhar (conectando o positivo do LED no **pino 9**

do Arduino e o negativo no **GND**). Modifique apenas o código no Arduino e replique o *patch* abaixo em *Pure Data*.

```
int ledPin = 9;

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

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



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 pressionamos o botão.

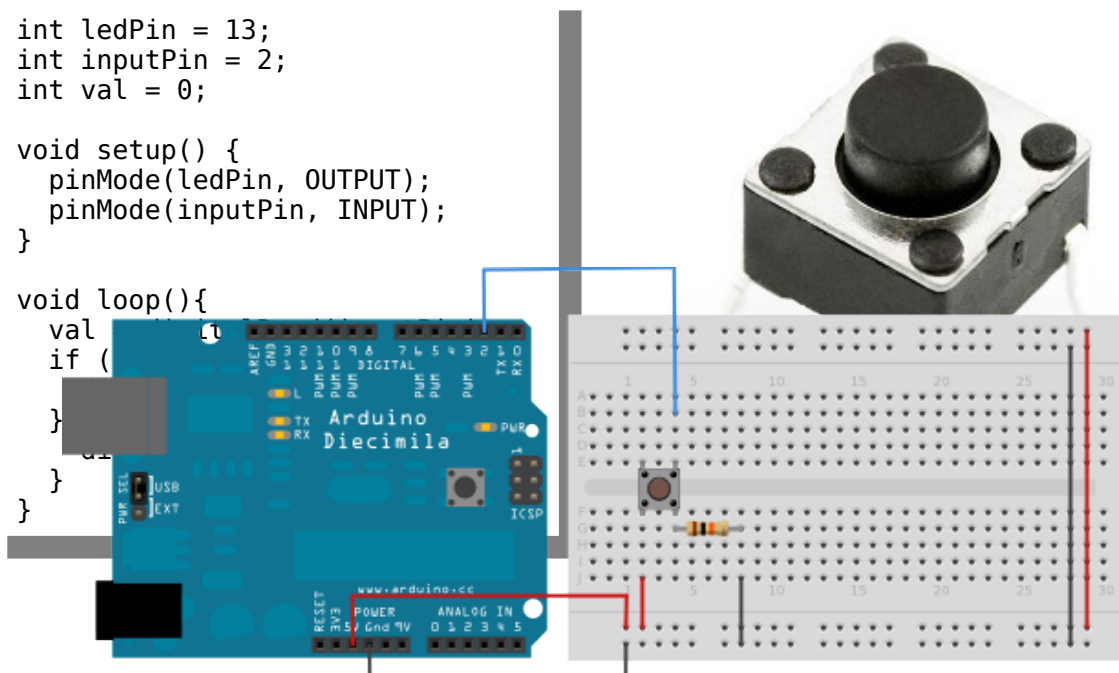
Quando o botão está livre (não pressionado), não há conexão entre as suas duas pernas, de forma que o pino do *Arduino* está conectado aos **5V** e ao ler o pino, obtemos **HIGH**. Quando o botão é fechado (pressionado), ocorre a conexão entre suas pernas, de forma que o pino do *Arduino* é ligado ao **GND** 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.

```
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, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```



BOTÕES + PD

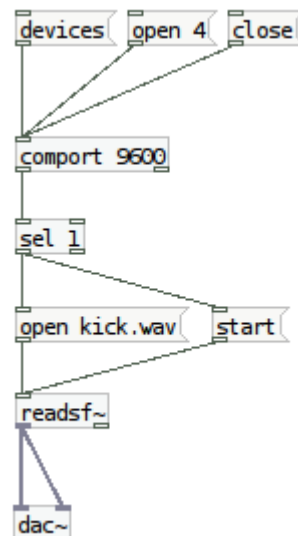
Da mesma maneira que fizemos para os LEDs, podemos não só controlar o *Arduino* com o computador, mas também fazer o inverso: controlar o computador com o *Arduino*! Para isso, novamente, cole os códigos abaixo no *Arduino* e no *Pure Data*, aproveitando o mesmo circuito que você montou para o botão.

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*.

```
int pinoBotao = 2;

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

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



POTENCIÔMETROS



Um *potenciômetro* 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.

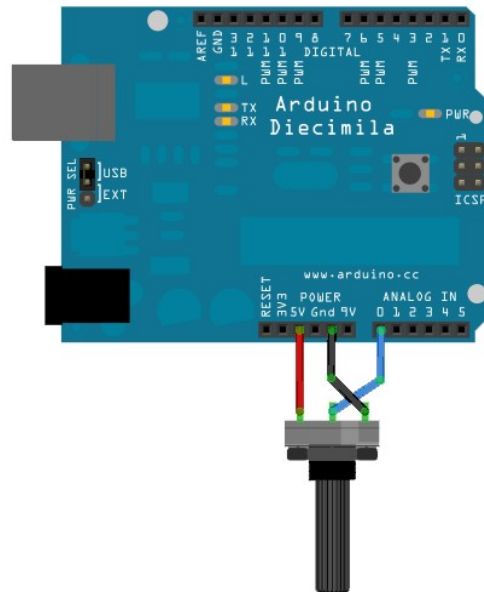
Conectamos três fios à placa *Arduino*. O primeiro vai do **terra** do *Arduino* a partir da perna esquerda do potenciômetro (ele possui geralmente três pernas). O segundo fio vai dos **5 volts** do *Arduino* à perna direita. O terceiro e último fio vai da **entrada analógica 2** à perna central do potenciômetro.

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 volts** ou ao **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 do *Arduino* e, assim, ao lê-lo obteremos zero. Quando giramos o potenciômetro até o outro extremo 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*.

```
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);
}
```



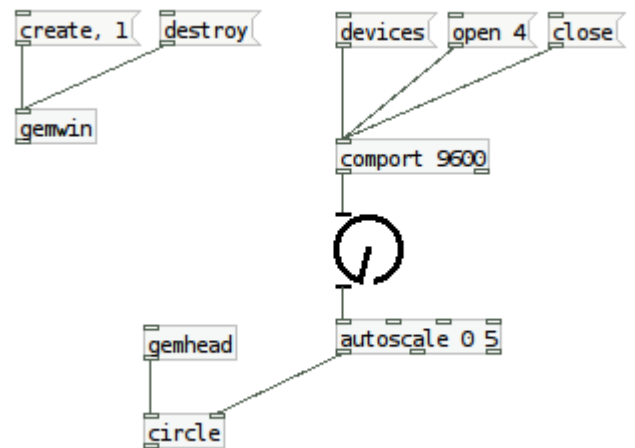
POTENCIÔMETROS + PD

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 esse valor lido do potenciômetro para modificar qualquer parâmetro de objetos em *Pure Data* (cor do objeto, posição na tela, rotação, tamanho, ...).

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

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

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



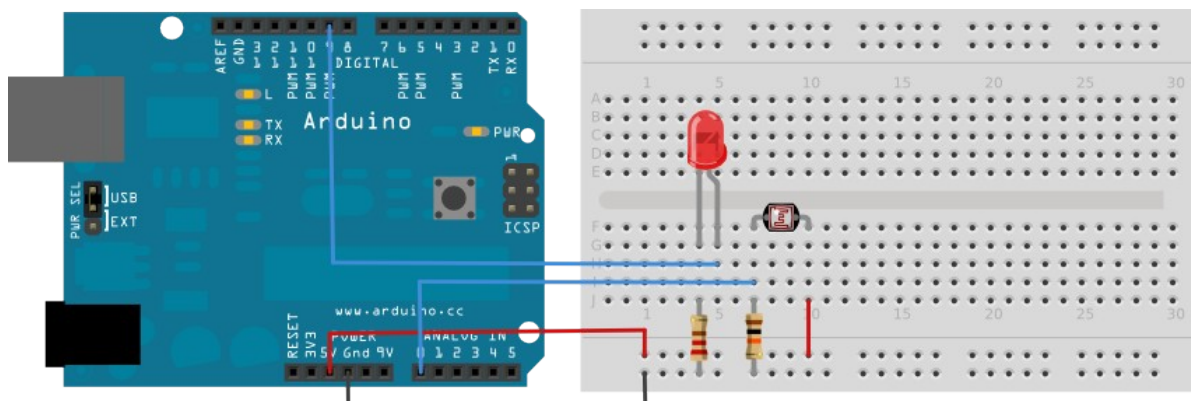
__LDR__

Um LDR, ou *Light Dependent Diode* faz o inverso que o LED. Ambos são diodos, mas ao invés de emitir luz como fazem os LEDs, o LDR a recebe! Seu comportamento é muito parecido com o potenciômetro, 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 cole o **código** no editor do *Arduino*.

```
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);
}
```



__LDR + PD__

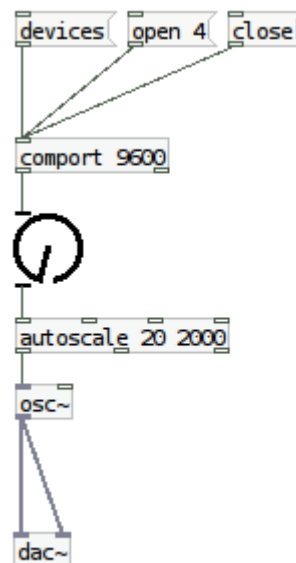
Vamos misturar novamente *Arduino* e *Pure Data*. Monte novamente o circuito para usar o LDR e cole os códigos abaixo no *Arduino* e *Pure Data*.

Ao invés de reproduzirmos um arquivo de áudio, agora estamos usando o computador para criar (sintetizar) o som! Usamos o LDR ligando 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.

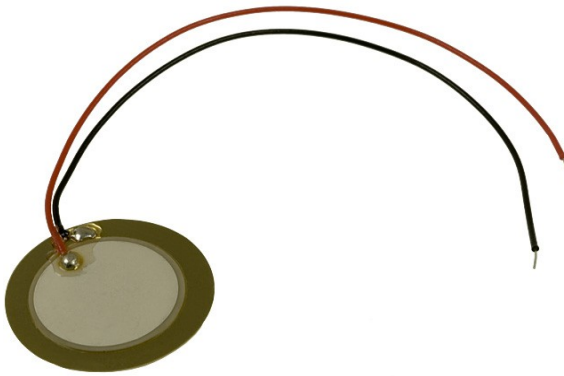
```
int entrada = 0;
int valor = 0;

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

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



__PIEZO como entrada__

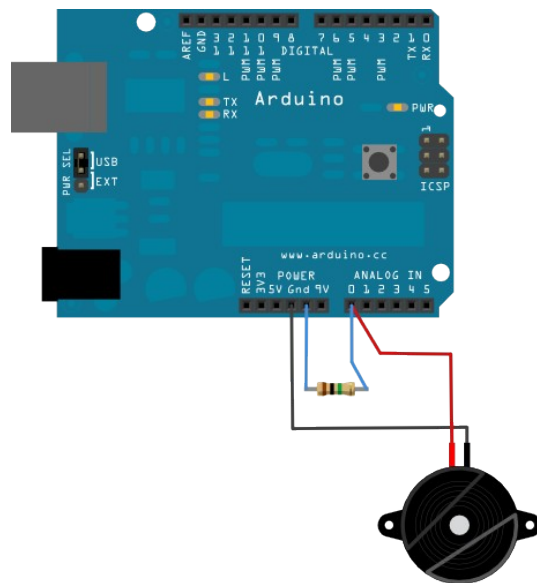


Potenciômetros variam sua resistência quando giramos seu botão. LDRs variam a resistência conforme a intensidade de luz. Piezoelétricos são cristais geram uma tensão conforme a pressão incidida sobre eles. Podemos usá-los para capturar vibrações no ambiente. São também úteis como saída: fazendo-os vibrar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

```
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);
  }
}
```



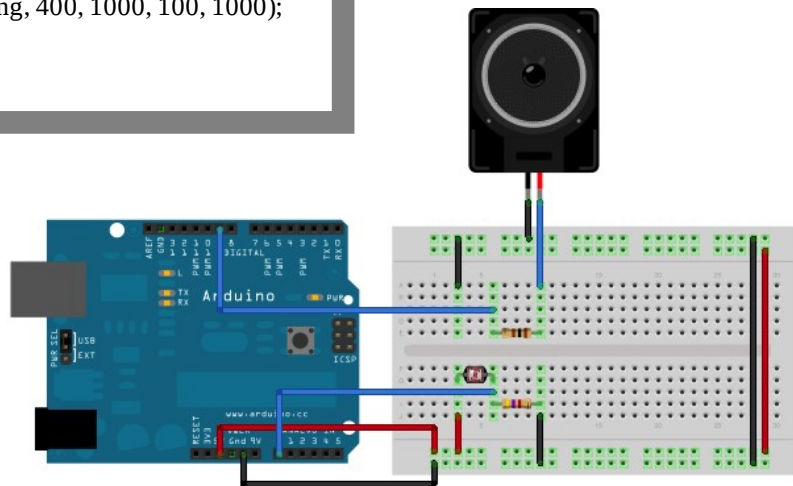
__PIEZO como saída__

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

```
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);
}
```



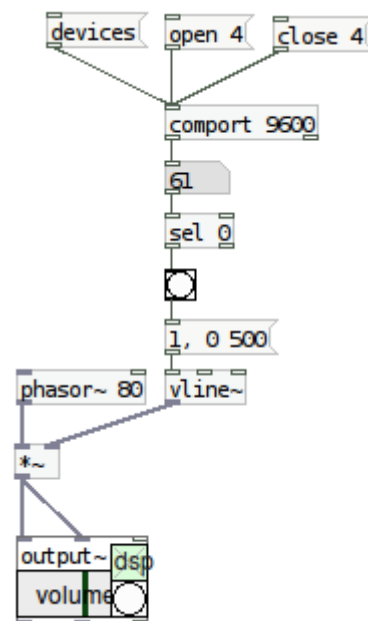
__PIEZO + PD__

Juntando *Pure Data* à sopa de circuitos e códigos... 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...

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

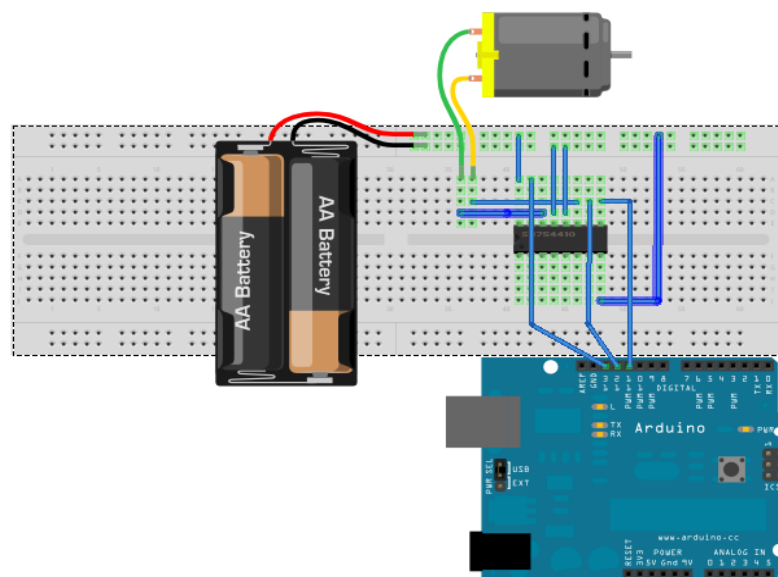
void setup() {
  pinMode(entrada, INPUT);
  Serial.begin(9600);
}

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



__MOTORES__

Motores são muito interessantes quando desejamos modificar o mundo físico à nossa volta. Quando lidamos com eles, temos de nos preocupar em como ligá-los e desligá-los na ordem correta, qual a intensidade que eles irão girar, em qual sentido, ... ou seja, são muitos detalhes para nos preocuparmos! Para nos ajudar, podemos usar *chips* especializados nessa tarefa. Existem vários *chips* com essa finalidade. Vamos usar o **CI L293D** nesse caso. Replique o circuito e código em *Arduino* para compreender melhor.



```
int motor1Pin1 = 13; // pin 15 on L293D
int motor1Pin2 = 12; // pin 10 on L293D
int enablePin = 11; // pin 9 on L293D
```

```
void setup() {
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enablePin, OUTPUT);
  digitalWrite(enablePin, HIGH);
}
```

```
void loop() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
}
```

MOTORES + PD

Aproveitando o circuito que você criou para ligar um motor ao *Arduino*, modifique somente o

código e crie o *patch* em *Pure Data*.



>>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, abuse-os!

AUDUINO

sintetizador de áudio usando Arduino

<http://code.google.com/p/tinkerit/wiki/Auduino>

ARDUINO 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>

>> EVENTOS <<

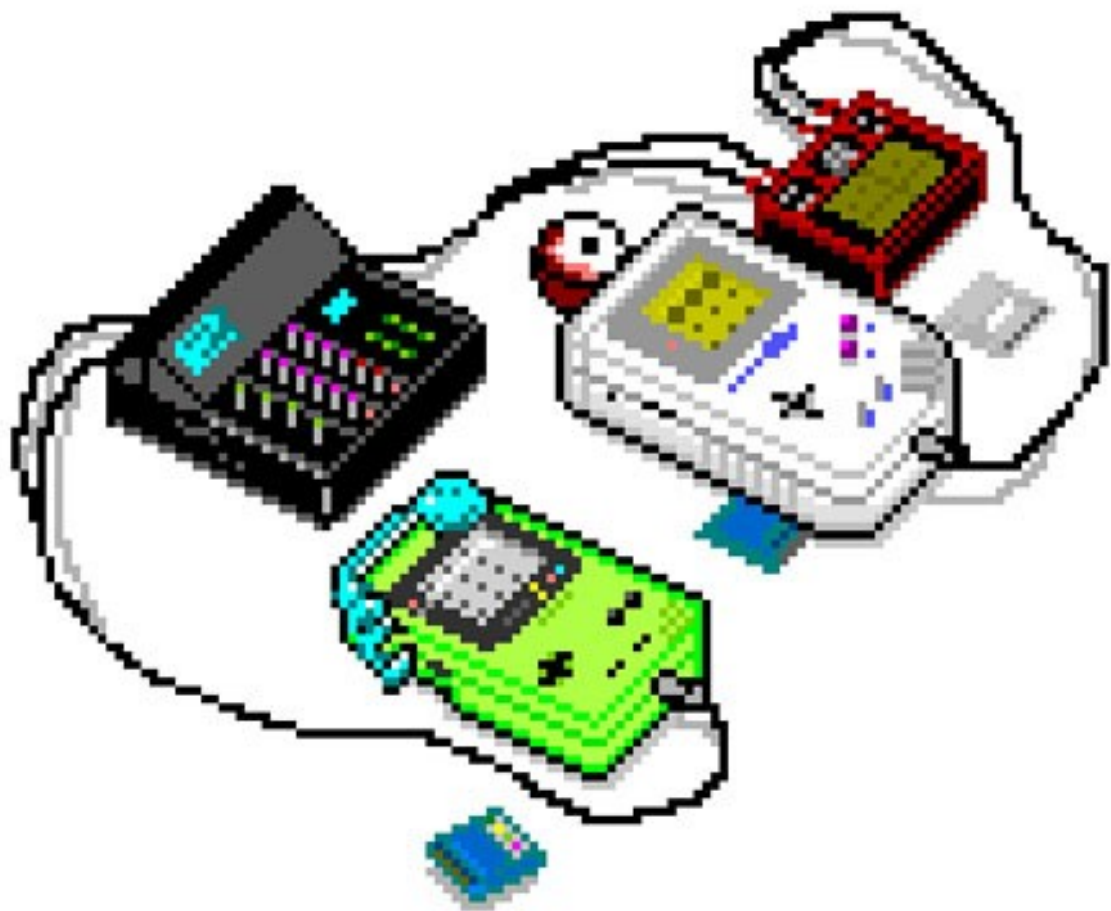
Sub>Valadares (JUNHO 2010) → <http://submidialogias.descentro.org/>

FISL (JULHO 2010) → <http://fisl.software.org>
LatinoWare (OUTUBRO 2010) → <http://latinoware.org>
SoLiSC (NOVEMBRO 2010) → <http://solisc.org.br>



Chico já dizia: Computadores fazem arte, artistas fazem dinheiro.
UseM:





Atari 2600

