

Márcio Gomes
Assessoria de Inclusão Digital
SMED - Porto Alegre - RS

Conceitos, referências e programações básicas com Arduino



Agosto 2014

O conteúdo deste material é publicado sob a licença Creative Commons. Você tem a liberdade de compartilhar, copiar, distribuir, transmitir e criar obras derivadas ainda que para fins comerciais, contanto que o crédito seja atribuído ao autor e que essas obras derivadas sejam licenciadas sob os mesmos termos.



Material elaborado para uso exclusivamente educativos.

Dados Internacionais de Catalogação na Publicação (CIP)

G633c Gomes, Márcio Luciano Santos Silva.

Conceitos, referências e programações básicas com Arduino /
Márcio Gomes. – Porto Alegre : SMED, 2014.

83 p. : il.

E-book disponível em:

<<http://websmed.portoalegre.rs.gov.br/escolas/robotica/ebook.htm>>

1. Educação. 2. Robótica. 3. Arduino. I. Título.

CDU 37:681.3

Catalogação elaborada pela Biblioteca da Secretaria Municipal de Educação de Porto Alegre/SMED

SUMÁRIO

| | |
|--|----|
| CONCEITOS BÁSICOS..... | 5 |
| O que é ARDUINO?..... | 5 |
| Porque usar ARDUINO? | 5 |
| O hardware do ARDUINO... | 6 |
| Como é programado o ARDUINO? | 7 |
| Como trabalhar com a protoboard? | 9 |
| O que são níveis lógicos?..... | 11 |
| PROJETOS INICIAIS..... | 13 |
| Programar um LED pisca-pisca..... | 13 |
| Utilizando variáveis | 21 |
| Trabalhando com um botão | 25 |
| Emitindo som com arduino..... | 31 |
| Lendo a intensidade luminosa..... | 34 |
| PROJETOS MAIS COMPLEXOS | 39 |
| Protegendo o arduino | 39 |
| Acionando um relé | 42 |
| Usando um potenciômetro | 49 |
| Movimentando um Motor | 58 |
| Movimentando um Motor DC | 60 |
| Movimentando um Servo Motor..... | 70 |
| Sensor ultrasônico HC-SR04 | 76 |

APRESENTAÇÃO

A robótica pedagógica é um importante viés no universo de ensino-aprendizagem empreendido na rede municipal de ensino de Porto Alegre. O trabalho desenvolvido ao longo de anos demonstra quão significativo é a aprendizagem mediada pela robótica, quão eficaz e abrangente se manifesta como ferramenta concreta na relação com o conteúdo educacional, quão importante é na abrangência e amplitude de habilidades e competências desenvolvidas, quão motivadora ela é para todos os envolvidos, quer discentes, quer docentes, ou quaisquer dos membros da comunidade escolar.

Desenvolver um robô em nossas oficinas é algo instigante para todos, pois envolve a criatividade na manipulação de diversos componentes, impulsiona o pensar sobre possibilidades de montagens, incita o desenvolvimento de estratégias para atingir objetos, encoraja a elaborar simples e complexas programações, induz ao diálogo com as mais diferentes áreas do conhecimento, apresenta um resultado palpável e concreto a partir das próprias possibilidades dos envolvidos.

A medida em que os alunos e alunas percebem-se capazes de projetar, montar e programar ações com seus robôs, também fortalecem sua autopercepção de maneira positiva consoante ao desenvolvimento de suas próprias habilidades.

A robótica provoca a necessidade da apropriação e incorporação de tecnologias que não fizeram parte da formação da imensa maioria dos profissionais que atuam na educação. Nosso desafio enquanto docentes é de aprender, junto com os alunos, algumas técnicas e formas de melhor trabalharmos na construção, montagem e programação dos robôs, vencendo o receio paralisante que geralmente as tecnologias impactam aos não iniciados.

O presente ebook quer auxiliar nesta tarefa, ao ajudar nas questões técnicas relacionadas a programação do Arduino com a sua IDE original. Este ebook será sucedido com outras publicações que potencializem o caráter pedagógico do que está apresentado nesta edição.

CONCEITOS BÁSICOS

O QUE É ARDUINO?

fonte de pesquisa:

<http://arduino.cc/>

<http://opentapajos.org/?p=741>

http://www.labdegaragem.com.br/wiki/index.php?title=Sobre_Arduino

A palavra "Arduino" é nome próprio italiano que tem origem germânica. É composto pelas palavras *hard/hart* (forte - brave, hardy, strong) e *win* (amigo em saxão antigo) formando *Hardwin* (Grande Amigo), que foi latinizado para *Ardovinus*, e depois para o italiano *Arduino*.

O Arduino é uma plataforma de prototipagem eletrônica, criado na Itália, que teve como objetivo ser um ambiente de desenvolvimento o mais flexível e acessível para as pessoas que não tinham experiência com programação. É destinado a artistas, designers, entusiastas e qualquer pessoa interessada em criar objetos ou ambientes interativos.



POR QUE USAR ARDUINO?

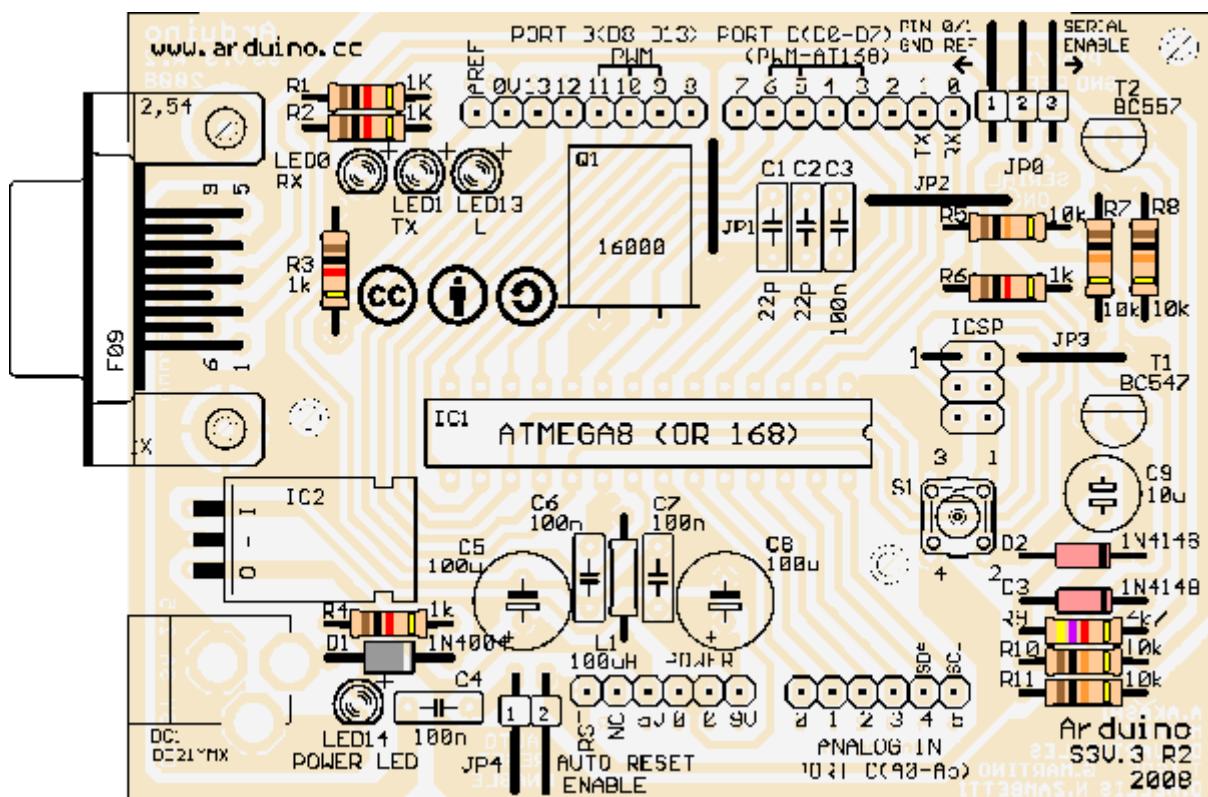
Recomenda-se o uso do Arduino porque permite o desenvolvimento de inúmeros projetos como sistemas interativos, robôs, automação residencial, protótipos industriais ou amadores, com baixo custo e com nível de aprendizado acessível.

Pelo fato de todo material ser disponibilizado em código aberto (open-source) não é preciso

gastar com licenças nem ao adquirir o hardware, a placa Arduino, tampouco para usar a interface de programação, o software. O software é disponibilizado para várias plataformas: windows, Linux e Mac OS.

O Arduino é completamente um software livre. Se quiser construir seu próprio software ou modificar um, você é livre para isso. Além disso, o Web site oficial do Arduino contém um wiki extensivo no qual amostras de código e exemplos são compartilhados livremente. Os desenvolvedores do Arduino tentam manter sua linguagem fácil de usar para iniciantes, mas flexível o bastante para usuários avançados.

É possível construir manualmente sua própria placa Arduino. Há um esquema detalhado no próprio site do projeto: <http://arduino.cc/en/uploads/Main/ArduinoSeverinoManual2.pdf>.

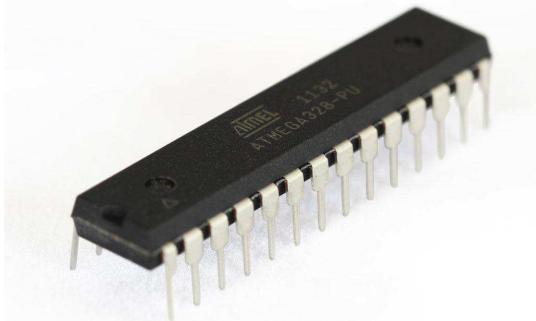


O Hardware do ARDUINO...

A base do Arduino é o microcontrolador. Um microcontrolador é um CI (circuito eletrônico miniaturizado, também conhecido como chip) que incorpora várias funcionalidades. Alguns vezes os microcontroladores são chamados de “computador de um único chip”. São utilizados em diversas aplicações de sistemas embarcados, tais como: carros, eletrodomésticos, aviões, automação residencial, etc.

E é esse chip que possui todo hardware para obter dados externos, processar esses dados e devolver para o mundo externo. Os desenvolvedores do Arduino optaram em usar a linha de micro controladores da empresa ATMEL. A linha utilizada é a ATMega. Existem placas

Arduino oficiais com diversos modelos desta linha, mas os mais comuns são as placas com os chips ATMega8, ATMega162 e ATMega328p.



Há só um TIPO de ARDUINO?

Há vários tipos diferentes de Arduino desenvolvidos pelos seus idealizadores, além de muitas outras versões clones ou derivadas desses.

Dependendo do tipo de projetos há formatos e configurações de hardware específicas. O Arduino Uno é um dos modelos mais utilizados, mas Arduino Mega, por exemplo, possui muito mais portas de entrada, possibilitando a criação de dispositivos maiores e mais complexos.

Já o Arduino LilyPad foi desenvolvida para vestimentas e tecidos inteligentes. Ele pode ser costurado diretamente sobre tecido e de modo similar ser conectado com fontes de alimentação, sensores e atuadores com linha condutiva.

Há uma versão gaúcha do Arduino, o Mateduino (<http://matehackers.org/doku.php?id=mateduino>) desenvolvido pela MateHackers.



como é Programado o ARDUINO?

O Arduino tem seu próprio ambiente de programação, **Arduino Integrated Development Environment (IDE)**, que pode ser baixado gratuitamente no site: <http://arduino.cc/en/Main/Software>. Esse ambiente de desenvolvimento do Arduino é um compilador gcc (C e C++) que usa uma interface gráfica construída em Java. A linguagem de programação utilizada no arduino é derivada dos projetos Processing (<http://www.processing.org/>) e Wiring (<http://wiring.org.co/>).

Além do ambiente de programação para o Arduino, existem outros softwares que podem facilitar o entendimento e documentação dessa tecnologia:

- **Fritzing** [<http://fritzing.org/>] é um ambiente de desenvolvimento de software dentro do projeto Arduino. Possibilita que os usuários possam documentar seus protótipos e, principalmente, que possam ilustrar a implementação de um projeto real de uma maneira fácil e intuitiva de ser entendida por outros usuários.
- **Minibloq** [<http://blog.minibloq.org/>] é um ambiente de desenvolvimento gráfico para Arduino onde a programação é feita conectando blocos. O principal objetivo é auxiliar o ensino de programação e, em especial, o ensino de robótica em nível de ensino médio.
- Scratch for Arduino - **S4A** [<http://s4a.cat/>] é outro ambiente de desenvolvimento gráfico para Arduino, uma modificação do programa Scratch.
- **Ardublock** [www.ardublock.com] - é uma aplicação desenvolvida em java que roda dentro da IDE do Arduino. Ela é semelhante ao S4A, sendo outro ambiente de desenvolvimento gráfico.

A função do Arduino IDE é, alem de escrever o código do programa, compilar esse programa e enviá-lo para a placa do Arduino.



Imagen da abertura do programa do Arduino.

COMO PROGRAMAR COM ARDUINO IDE?

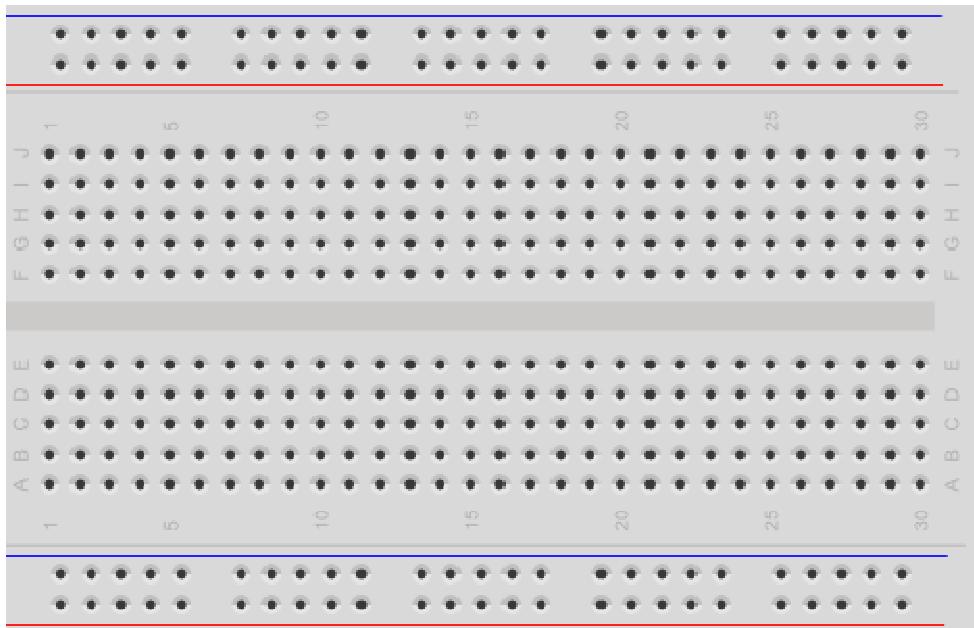
Depois de ter baixado e instalado o programa IDE é necessário configurar o drive, caso você esteja utilizando o windows. Há documentação na internet explicando passo a passo a instalação como neste link: <http://renatoaloi.blogspot.com.br/2011/10/installando-arduino-guia-completo.html>.

A programação do Arduino é fácil de ser compreendida, mesmo para pessoas que nunca tiveram contato com linguagem de programação anteriormente. A seguir listaremos uma série de experimentos simples que usaremos para aprender a programar no Arduino.

COMO TRABALHAR COM A PROTOBOARD?

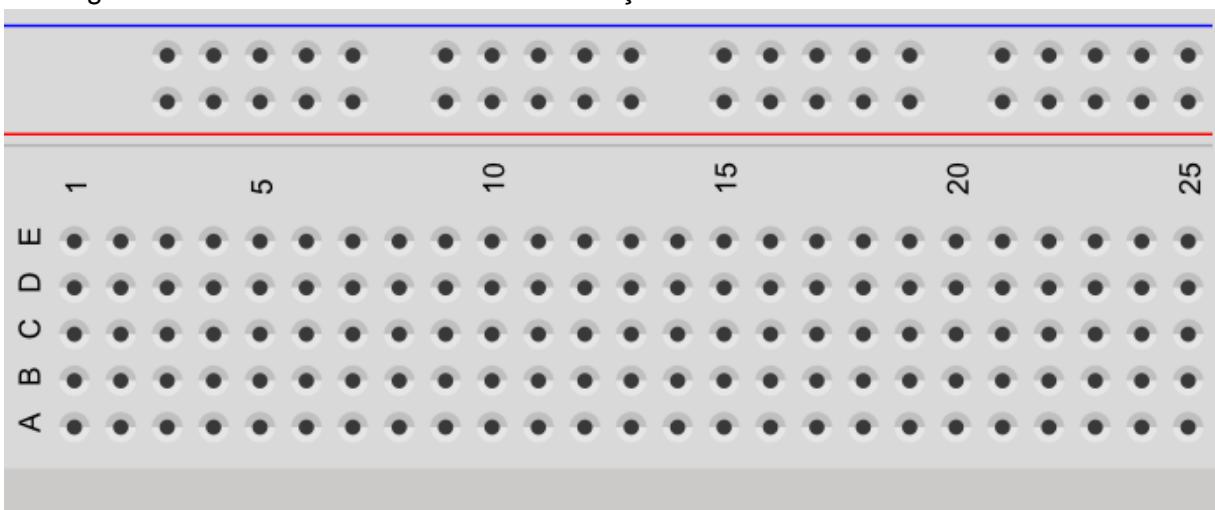
Baseados <http://www.eletronicadidatica.com.br/equipamentos/protoboard/protoboard.htm>

Antes de iniciarmos os experimentos vamos entender como funciona a protoboard. Uma protoboard, também conhecida como matriz de contatos, é utilizada para fazer montagens provisórias, teste de projetos, além de inúmeras outras aplicações.



A protoboard é montada por uma base de plástico com várias entradas destinadas à inserção dos terminais dos componentes permitindo que seja possível montar um circuito elétrico sem a necessidade de solda.

Internamente existem ligações determinadas que interconectam os orifícios, permitindo a montagem de circuitos eletrônicos sem a utilização de solda.

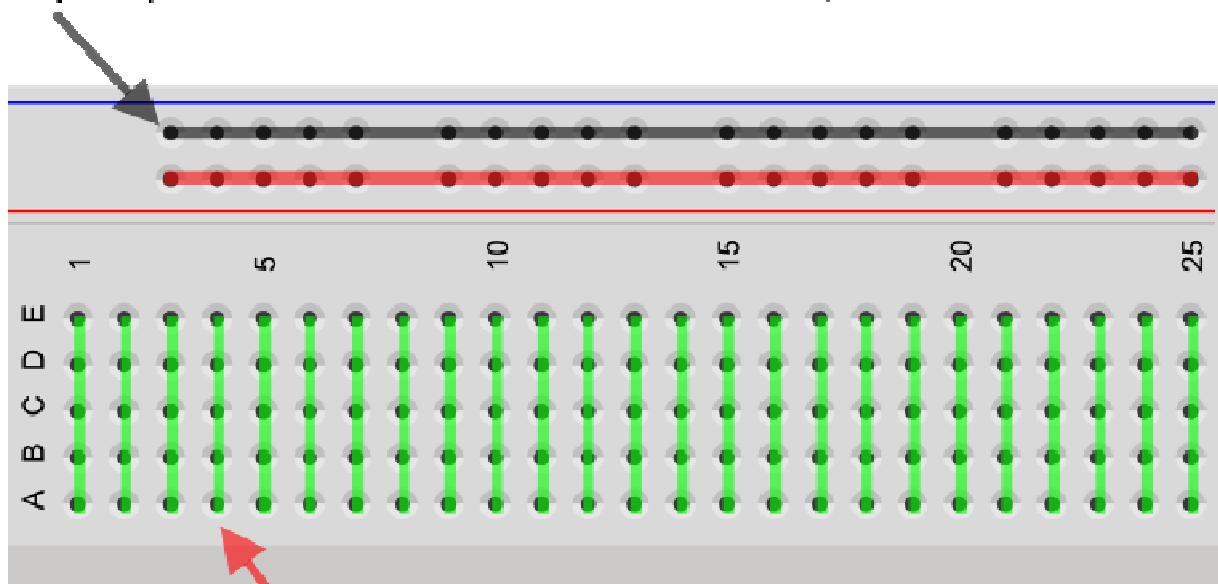


Acima temos uma figura com um “recorte” da protoboard. Na figura bem acima temos uma lista azul e outra vermelha. Na primeira fileira, cada entrada, orifício, está conectado ao outro na horizontal e apenas na horizontal. Na segunda fileira, também. Ali é conectado a alimentação da protobord, sendo o azul o negativo e o vermelho, o positivo.

Na mesma figura observamos os números que representam as colunas e as letras que representam as fileiras horizontais. Nesta parte da protoboard, os orifícios estão interconectados por colunas.

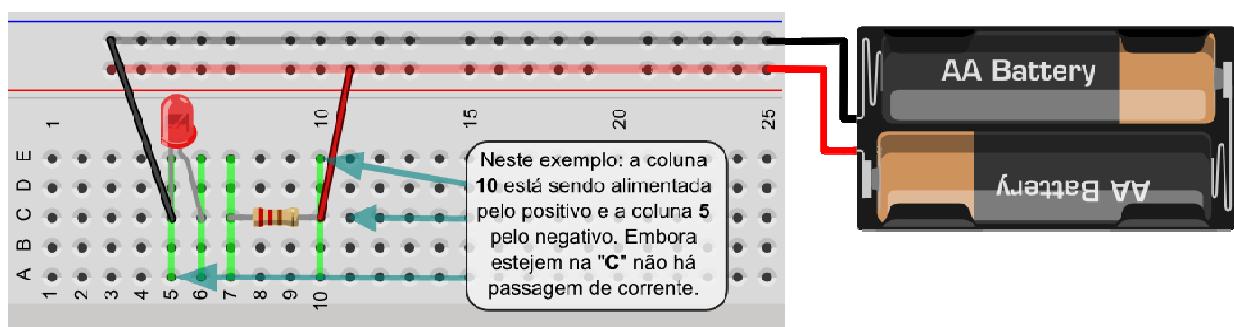
A imagem abaixo ilustra melhor:

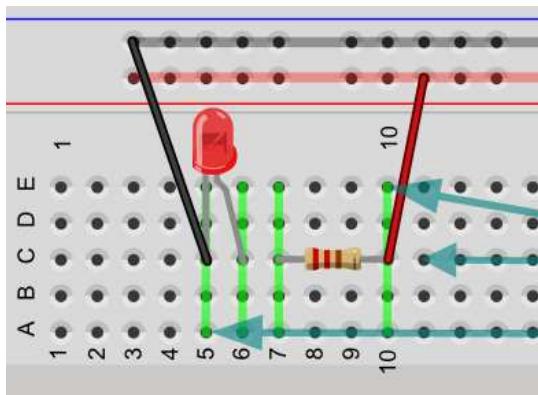
Aqui os pontos estão interconectados neste sentido, em linha.



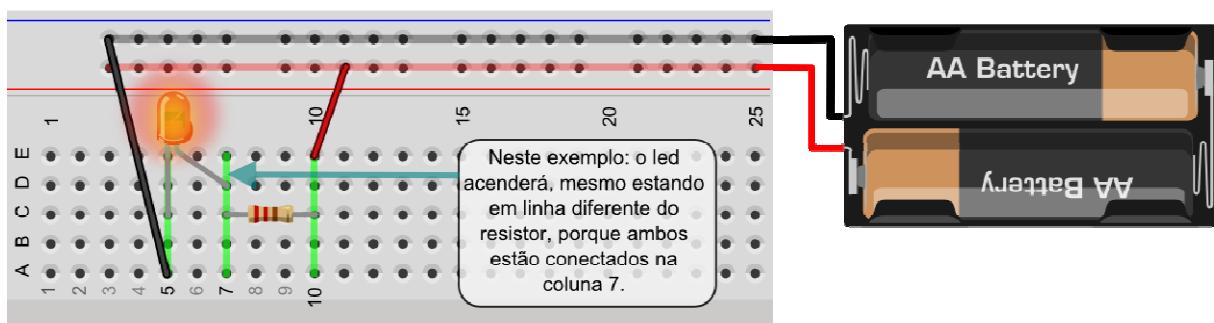
Aqui os pontos estão interconectados em colunas.

Se ligarmos os terminais de vários componentes na mesma coluna eles entrarão em curto. Os terminais de um componente devem ser colocados em colunas diferentes. Vamos analisar a figura abaixo:





O led está posicionado com um terminal na linha **C** dentro da coluna **5**, e com o outro terminal na coluna **6** da mesma linha. O led não acenderá embora todos os componentes estejam na mesma linha. Isso porque, entre a coluna **6** e **7** não há passagem de corrente.



O que são níveis lógicos?

Baseado <http://www.electronicadigital.com/site/curso-eletronica-digital/8-licao2.html?start=1>

Um outro conhecimento importante é o conceito de nível lógico. É um conceito simples, mas importante para realmente entendermos o que estamos programando.

Como já vimos anteriormente, o arduino trabalha com pinos analógicos e digitais, através deles faz a leitura ou escrever valores baseados na corrente elétrica. Por padrão os **pinos digitais** trabalham com duas condições possíveis: com corrente elétrica ou sem corrente elétrica significativa. Quando escrevemos em um pino digital colocando ele em nível alto, ele passa a fornecer uma corrente de 5v. Quando escrevemos nele colocando em nível baixo e não passa mais corrente.

Então a presença de uma tensão será indicada como nível **1** ou **HIGH (nível alto)** enquanto que a ausência de uma tensão será indicada por nível **0** ou **LOW (nível baixo)**. Esses são os níveis lógicos.

Importante dizer que os níveis lógicos estão presentes em outros equipamentos eletrônicos. O 0 ou LOW será sempre uma tensão nula, ou ausência de sinal num ponto do circuito, mas o nível lógico 1 ou HIGH pode variar de acordo com o circuito analisado. Nos computadores desktop, a tensão usada para a alimentação de todos os circuitos lógicos, por exemplo, é de 5 V. Assim, o nível 1 ou HIGH de seus circuitos será sempre uma tensão de 5 V. Nos notebooks é usada uma tensão de alimentação menor, da ordem de 3,2 V, portanto, nestes

circuitos um nível 1 ou HIGH sempre corresponderá a uma tensão desse valor.

Nível lógico

| | |
|--------------------|--------------------|
| Estado 0 | Estado 1 |
| 0 V | 5 V |
| Falso | Verdadeiro |
| Desligado | Ligado |
| Nível baixo ou LOW | Nível alto ou HIGH |

PROJETOS INICIAIS

Programar um LED pisca-pisca

fonte de pesquisa:

<http://ferpinheiro.wordpress.com/2011/05/18/arduino-controlando-leds-pelo-teclado/>
<http://www.ladyada.net/learn/arduino/lesson3.html>

Componentes necessários:



1 placa Arduino



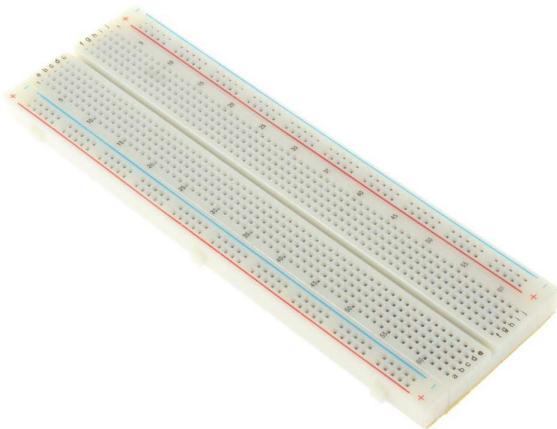
1 cabo USB



1 Led



1 resistor 220r (vermelho, vermelho, marrom)



1 protoboard

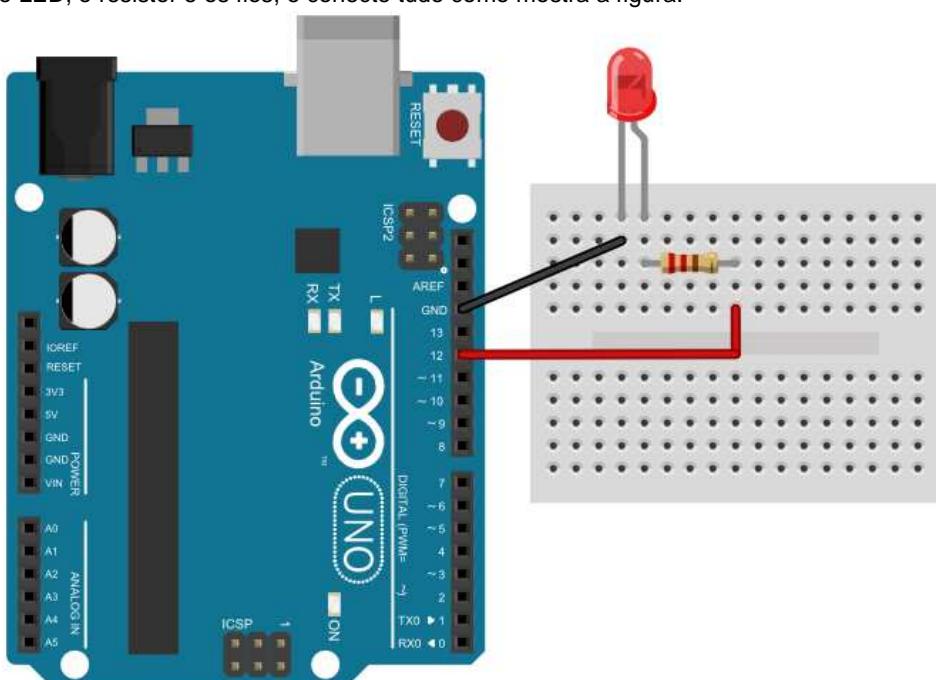


Fios diversos - jumpers

Conectando os componentes:

⚠ Sempre conecte componentes com a placa Arduino desligada ⚠

Primeiro, certifique-se de que seu Arduino esteja **desligado**, desconectando-o do cabo USB. Agora, pegue sua protoboard, o LED, o resistor e os fios, e conecte tudo como mostra a figura.

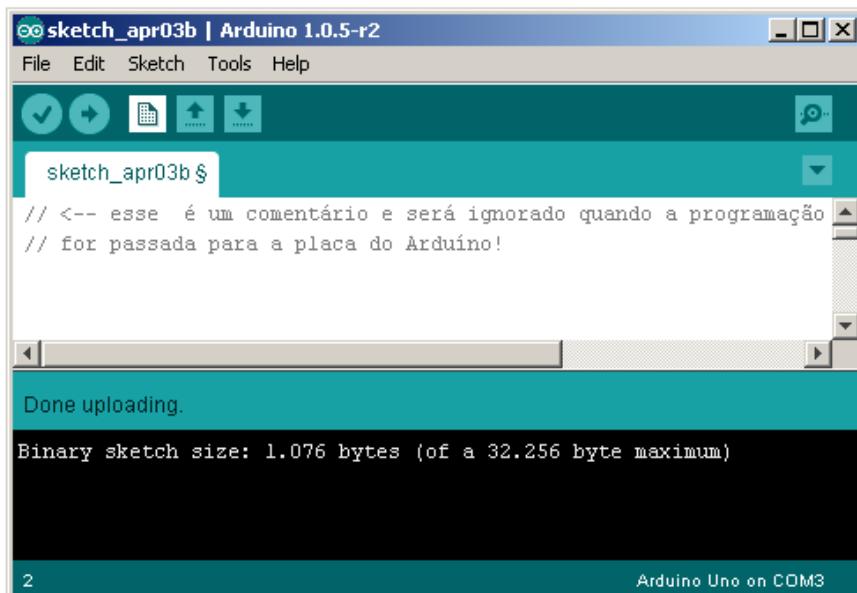


Lembrando que o terminal positivo do Led deve ser encaixado junto ao terminal do resistor, do contrário não acenderá. O terminal mais longo é o positivo, o anodo do led.

Digitando o código na IDE:

Ao decorrer dos experimentos vamos explicando como trabalhar com a IDE do Arduino. Neste primeiro exemplo apresentaremos o funcionamento básico da programação.

Vamos iniciar nossa programação escrevendo um comentário. Deixar comentário no código nos ajuda na organização do mesmo, principalmente para projetos muito extensos.



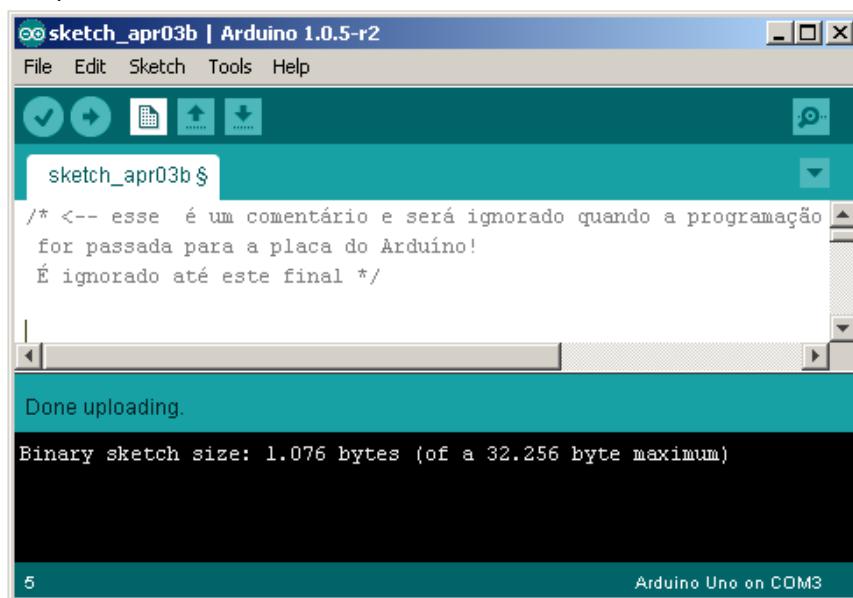
The screenshot shows the Arduino IDE interface. The title bar reads "sketch_apr03b | Arduino 1.0.5-r2". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for save, upload, and other functions. The code editor window contains the following text:

```
// <-- esse é um comentário e será ignorado quando a programação  
// for passada para a placa do Arduino!
```

The status bar at the bottom right shows "Arduino Uno on COM3".

||| Começamos digitando duas barras “//”, sem as aspas. Tudo o que escrevermos em uma linha que iniciar com “//” é interpretado como comentário e vai ser ignorado pelo compilador. No exemplo acima foi escrito um comentário na 1^º linha de programação e outro na 2^º linha.

Mas poderia ser assim:



The screenshot shows the Arduino IDE interface with the same title bar and menu as the previous image. The code editor window contains the following text:

```
/* <-- esse é um comentário e será ignorado quando a programação  
for passada para a placa do Arduino!  
É ignorado até este final */
```

The status bar at the bottom right shows "Arduino Uno on COM3".

/*

/* Tudo o que ficar entre o asterisco e a barra será considerado comentário pelo compilador. */

Agora vamos digitar o seguinte código:

```
void setup( )
{
    pinMode(12, OUTPUT);
}

void loop( )
{
    digitalWrite(12, HIGH);
    delay(1000);
    digitalWrite(12, LOW);
    delay(1000);
}
```

Depois clique em Verificar  verify. Se o código estiver correto não irá apontar nenhum erro. Depois, com o Arduino conectado ao computador, clique em  upload. A programação será enviada para o Arduino e o Led deve começar a piscar.

Comprendendo o código na IDE - explicação:

Há duas funções que são obrigatórias na IDE do Arduino. São elas a **setup** e a **loop**. Independente de sua programação essas funções precisarão serem digitadas.

A função **setup**, é de configuração. Quando passamos a programação para o Arduino, essa parte será executada apenas uma **única vez** diferente da função **loop** que, como o próprio nome diz, será executada infinitamente.

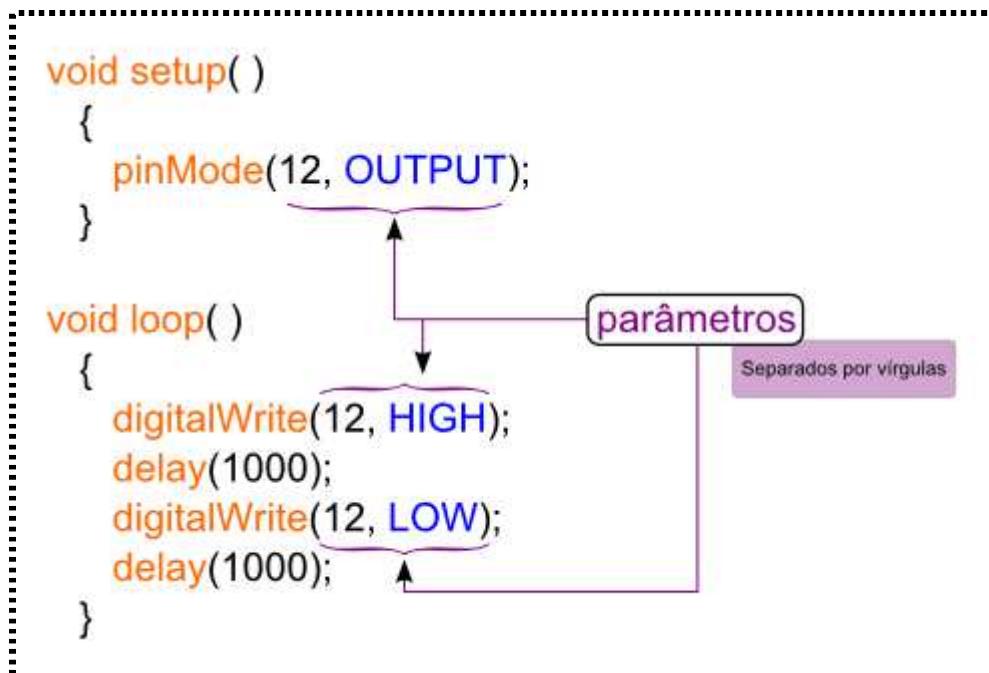
Mas o que é uma função? Função é uma procedimento, um código, que é criado para algum fim e que será utilizado mais de uma vez. Então, para não precisarmos digitar um código repetidamente, criamos uma função que armazena esse bloco de código. O arduino já vem com funções, digamos, pré-programadas. Esse é o caso da **loop** e da **setup**, mas podemos criar as nossas próprias funções. Vamos ver isso mais adiante.

Tipo Função (parâmetros)
{
Corpo
}

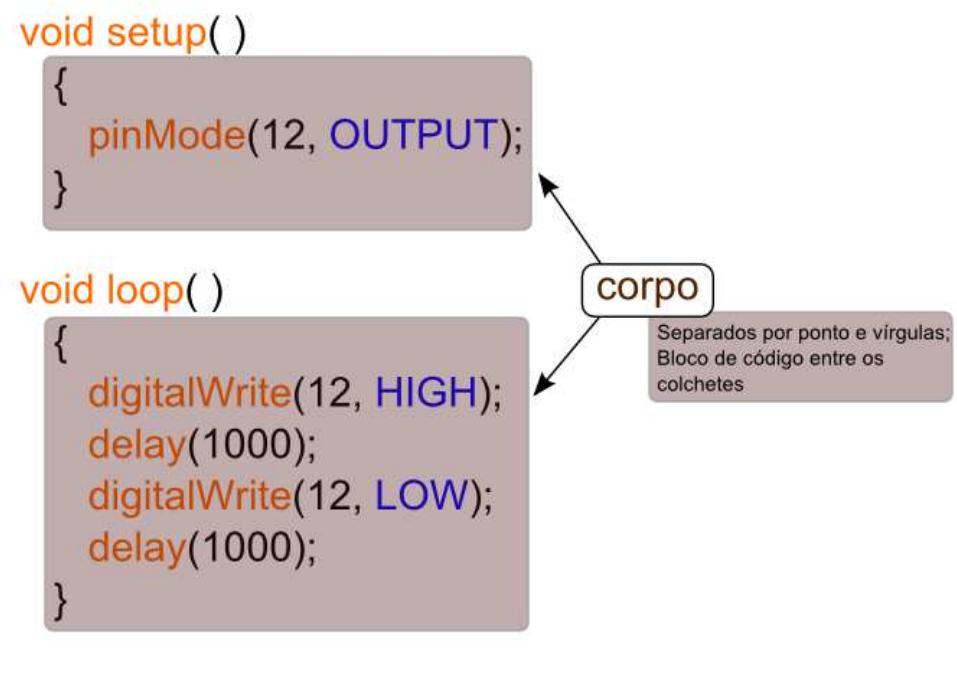
Utilizamos na função a estrutura mostrada na imagem acima. Os Parâmetros: (Ficam entre parenteses) **São separados por vírgulas**. O Corpo, os blocos de código: (Ficam entre chaves) **São separados por ponto e vírgula**.

Vale ressaltar que o código no arduino é sensível a caixa, isso significa que considera diferentes letras maiúsculas e minúscula. Na prática o Arduino interpreta isso **VOID**, diferente disso **void**, ou disso **Void**, ou disso **vOid**... embora sejam a mesma palavra!

Identificando os **parâmetros** no código estudado:



Identificando o **corpo** no código estudado:



Sempre começamos o código pela função **setup**. Esta função é do tipo **void**. Há várias outras tipologias, mas a **void** significa que não vai haver um retorno para esta função, ela apenas será executada. Mais tarde isso será melhor explicado. No momento é importante saber que deve ser colocado o tipo **void** antes do setup e do loop. Existem várias outras funções que fazem parte da biblioteca do arduino. Elas nos auxiliam a “poupar” o trabalho de programar tudo do zero. A referência de todas as funções da biblioteca arduino podem ser vistas no site oficial do projeto em: <http://arduino.cc/en/Reference/HomePage>.

```

pinMode(12, OUTPUT);

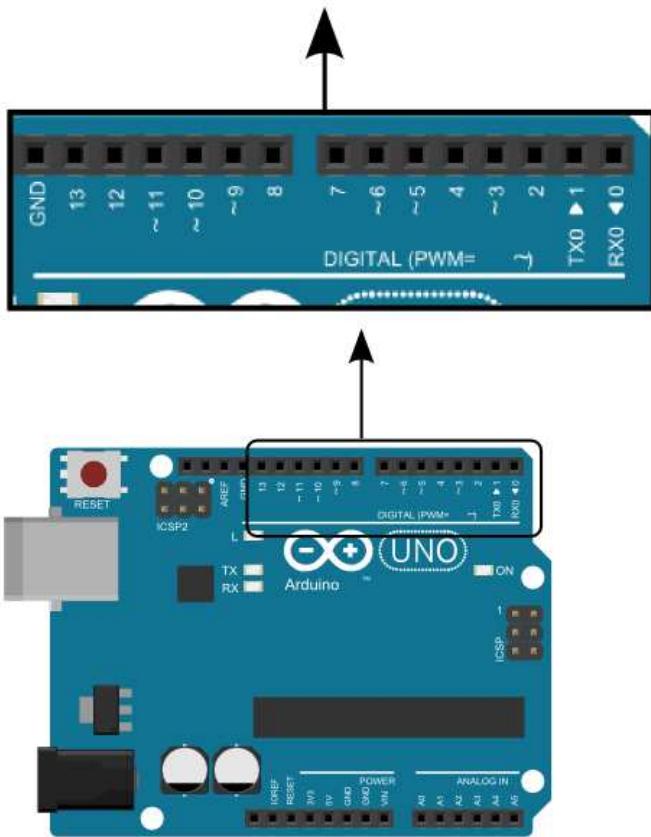
void setup()
{
    pinMode(12, OUTPUT);
}

void loop()
{
    digitalWrite(12, HIGH);
    delay(1000);
    digitalWrite(12, LOW);
    delay(1000);
}

```

No exemplo da nossa programação, temos dentro do corpo da função **setup** uma outra função: **pinMode**. Declaramos a pinMode para dizer que vamos utilizar o pino **12** da placa arduino. E vamos utilizá-la como saída (por isso **OUTPUT**), ou seja, nós vamos **fornecer** corrente elétrica já que queremos acender um Led.

pinos digitais



Mas poderíamos, em um outro projeto, querer saber se o Led estivesse acesso, ou se um botão/interruptor estivesse pressionado. Neste caso gostaríamos de fazer a **leitura** da corrente elétrica. Neste outro caso, iríamos declarar a pinMode, com entrada (ou **INPUT**).

As portas digitais tem 2 estados, ligada ou desligada. No arduino usamos **HIGH** ou **1** para assinalar que a porta deverá entrar no estado ligado. Usamos **LOW** ou **0** (zero) para assinalar que o estado da porta será desligado.

LOW,

```
void setup()
{
    pinMode(12, HIGH);
}

void loop()
{
    digitalWrite(12, HIGH);
    delay(1000);
    digitalWrite(12, LOW);
    delay(1000);
}
```

A função **digitalWrite** exerce, nesta programação, o atributo de ligar ou desligar o pino digital. Esta primeira linha da programação está liberando a corrente elétrica para o pino **12**.

```
void setup()
{
    pinMode(12, OUTPUT);
}

void loop()
{
    digitalWrite(12, HIGH);
    delay(1000);
    digitalWrite(12, LOW);
    delay(1000);
}
```

A função **delay** define o tempo, que neste caso é de 1 segundo (ou 1000 milisegundos). Depois é chamada a função **digitalWrite** que está retendo a corrente elétrica no pino 12. E novamente a função **delay** que define mais 1 segundo de espera.

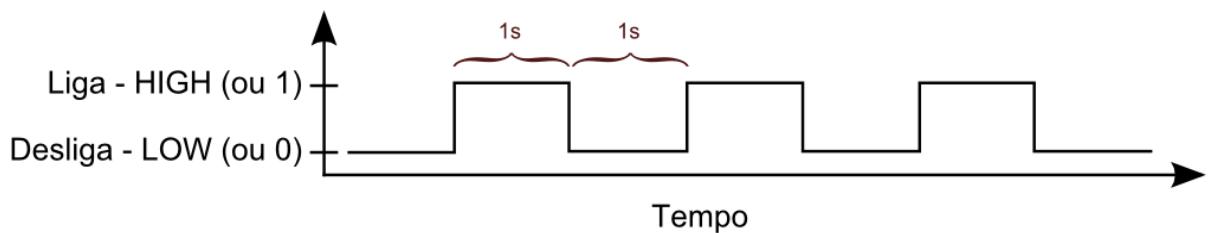
Na prática, o led que está no pino **12** acende durante 1 segundo e depois permanece apagado por outro segundo. Como este bloco de programação está dentro do corpo da função **loop**, será repetido indefinidamente dando o efeito de pisca-pisca.

O que está acontecendo eletronicamente?

Na programação estamos dizendo para o arduino que ele deve ativar o pino 12 através da função **pinMode**. Os demais pinos continuaram inativos. E que ele será ativado em modo de escrita, ou seja, para fornecer uma corrente elétrica (**OUTPUT**).

Na função **loop** estamos dizendo que a forma de escrita deve ser digital (**digitalWrite**). Existem outros tipos de leitura e escritas no arduino (análogo, pwm, interrupções) mas não vamos tratar desses recursos neste momento!

Estamos dizendo que o Arduino deve escrever **HIGH** no pino 12 deixando em nível alto (HIGH) durante 1000 milisegundos (1 segundo). O que significa que este pino deixara passar 5 volts durante 1 segundo. E depois estamos informando que o Arduino escrever **LOW** no pino 12 deixando-o em nível baixo (LOW) durante outros 1000 milisegundos.



Utilizando variáveis

fonte de pesquisa:

<http://renatoaloi.blogspot.com.br/2011/11/variaveis-do-arduino.html>
<http://multilogica-shop.com/Referencia>

Componentes necessários:

Utilizaremos os mesmos componentes do projeto anterior.

Aprimorando o código - usando Variáveis e Constantes - Escopo:

O uso da variável é um recurso importante na programação. A variável dentro do código funciona como se fosse uma “caixa” onde se guarda uma informação para ser usada posteriormente. A informação armazenada dependerá do tipo de variável que criaremos.

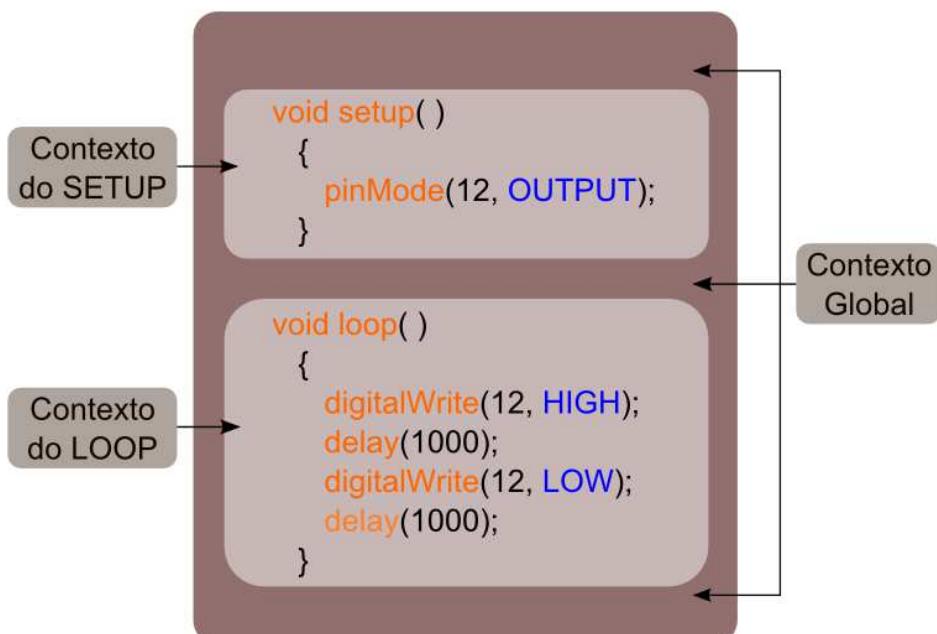
Exemplos de declaração de uma variável:

```
int pinodoled; // aqui iniciamos um variável sem um valor  
int pinodoled = 12; // aqui já iniciamos uma variável determinando um valor
```

Usaremos uma variável para armazenar o pino na qual o led deverá estar conectado, então onde tem escrito o pino 12, deverá ser escrito o nome da variável criada.

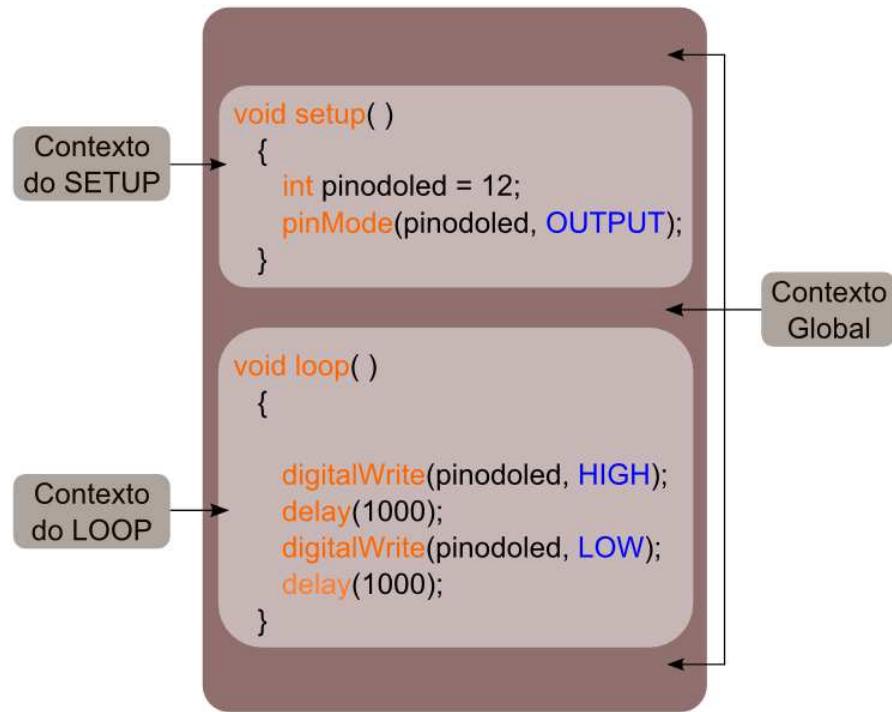
Antes de declarar um variável é importante ter a noção de qual lugar ela deverá ser declarada. Uma variável para ser “reconhecida”, ser visível, em qualquer lugar da programação deverá ser declara no escopo correto.

Mas o que é um escopo? Na programação escopo é o contexto, é o limite máximo na qual poderá ser utilizado um recurso.



A figura acima exemplifica como funciona o escopo. Se criarmos uma variável no contexto global ela irá “valer”, vai “ser visível”, em toda a programação. Diferente se criarmos uma variável no contexto do setup ou do loop, pois ela só será visível dentro de um ou de outro destes escopos.

No exemplo da figura abaixo, é declarada uma variável no escopo do **setup**. O problema é que ela é chamada no contexto do loop, mas isso resultará em erro.



Quando tentarmos passar a programação, resultará no erro exposto na figura abaixo. A variável não será reconhecida pois foi declarada em um escopo diferente.

A screenshot of the Arduino IDE interface. The title bar says "sketch_jun04a | Arduino 1.0.5-r2". The code editor window contains the following sketch:

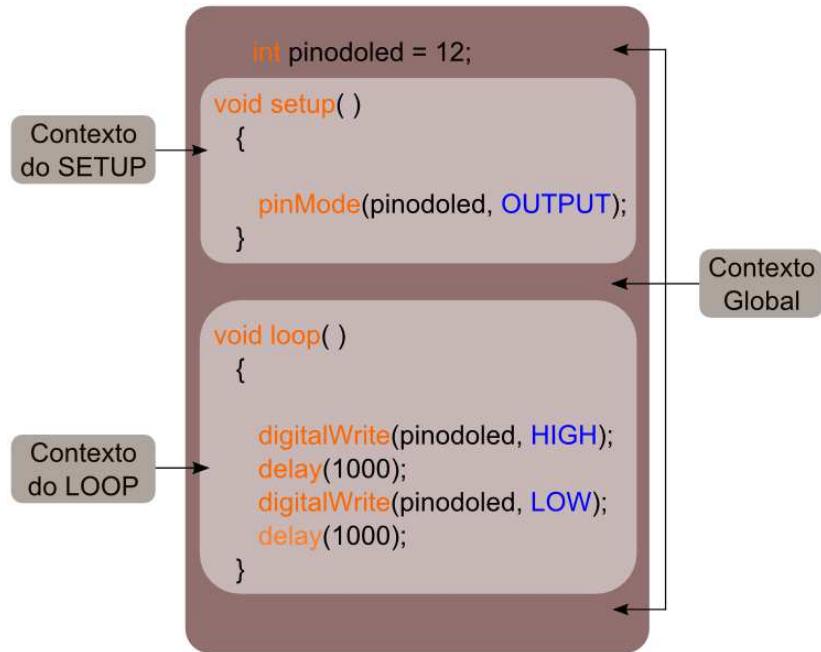
```
void setup()
{
    int pinodoled = 12;
    pinMode(pinodoled, OUTPUT);
}

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

The status bar at the bottom right shows "Arduino Uno on COM3". In the bottom right corner of the code editor, there is an orange error message box with the text "'pinodoled' was not declared in this scope". Below the code editor, the serial monitor window shows the error message: "sketch_jun04a: In function 'void loop()': sketch_jun04a:11: error: 'pinodoled' was not declared in this scope".

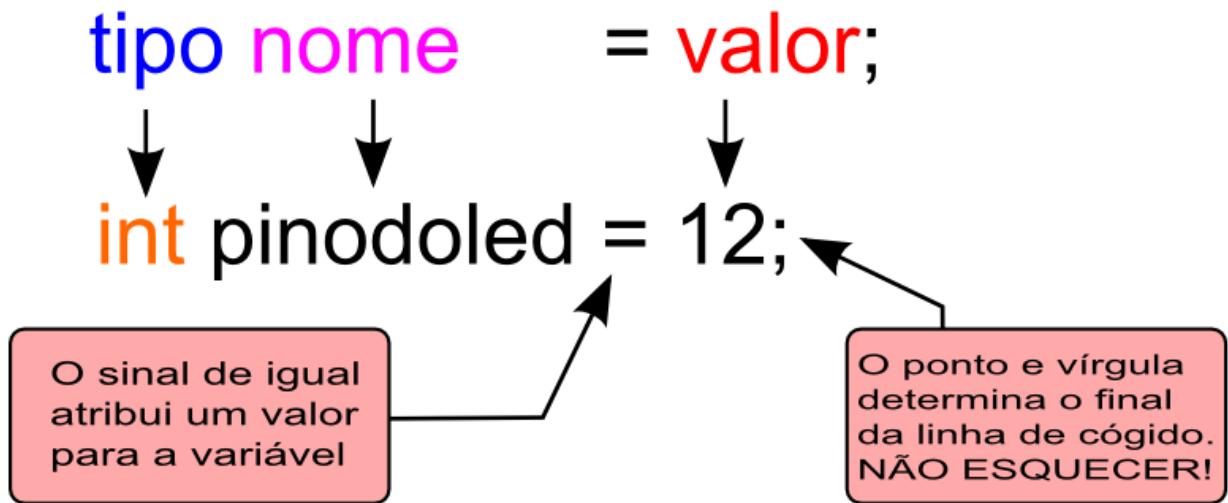
A solução, neste caso é declarar a variável no escopo global, para que ela seja visível pelo outros escopos. Veja

a figura:



Aprimorando o código - usando Variáveis e Constantes - tipos:

No exemplo anterior, declaramos uma variável **int** de nome **pinodoled** e declaramos seu valor como **12**. Vamos entender melhor cada parte da declaração de uma variável e seus tipos.



Na figura acima declaramos a variável e já determinamos o seu valor. Mas poderíamos apenas declará-la, sem nenhum valor.

A primeira parte da instrução da declaração da variável é o tipo. Variáveis podem ser de vários tipos. Usamos um tipo de variável de acordo com a informação que queremos armazenar dentro dela. No exemplo anterior usamos o tipo **int** o que significa que podemos armazenar um número inteiro. Mas também poderíamos usar no lugar de **int** uma outra variável chamada **byte**.

```
byte → 0 ~ 255
char → 'A'
int → -32.768 ~ 32.676
string → "muitos caracteres"
```

A variável de tipo **byte** pode armazenar números de 0 até 255, utilizando, como o próprio nome sugere, apenas 1 byte de memória. É um tipo ideal para armazenar, por exemplo, os pinos digitais que usamos no arduino que são do 0 até o 13. Na imagem acima alguns exemplos de tipos de variáveis.

Vamos deixar o nosso código da seguinte forma:

The screenshot shows the Arduino IDE interface. The title bar reads "sketch_jun04a | Arduino 1.0.5-r2". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for save, upload, and search. The code editor window contains the following code:

```
byte pinodoled;

void setup( )
{
    pinodoled = 12;
    pinMode(pinodoled, OUTPUT);
}

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

The status bar at the bottom displays "Done compiling." and "Binary sketch size: 1.084 bytes (of a 32.256 byte maximum)". The page number "19" is also visible in the bottom left corner.

Trabalhando com um botão

fonte de pesquisa:

<http://www.arduinoecia.com.br/2013/05/ligando-uma-lampada-com-rele.html>

<http://fritzing.org/projects/acionamento-de-um-rele-com-o-arduino>

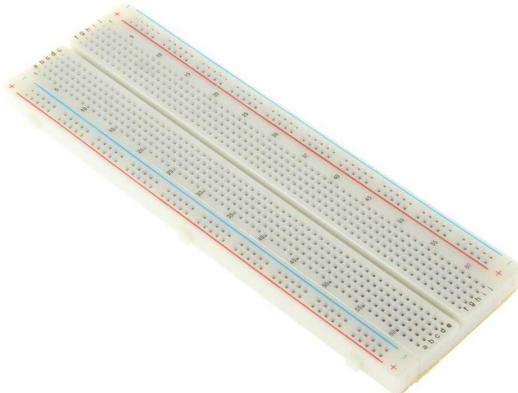
Componentes necessários:



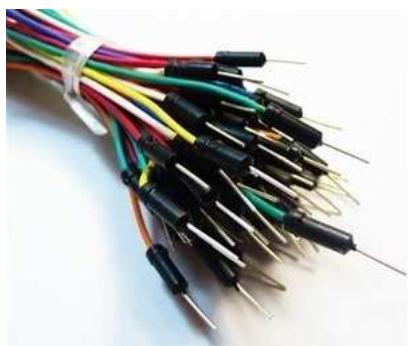
1 placa Arduino



1 cabo USB



1 protoboard



Fios diversos - jumpers



1 Chave toque | push button | interruptor momentâneo | botão de pressão | chave tactil (=>nomes correlatos)

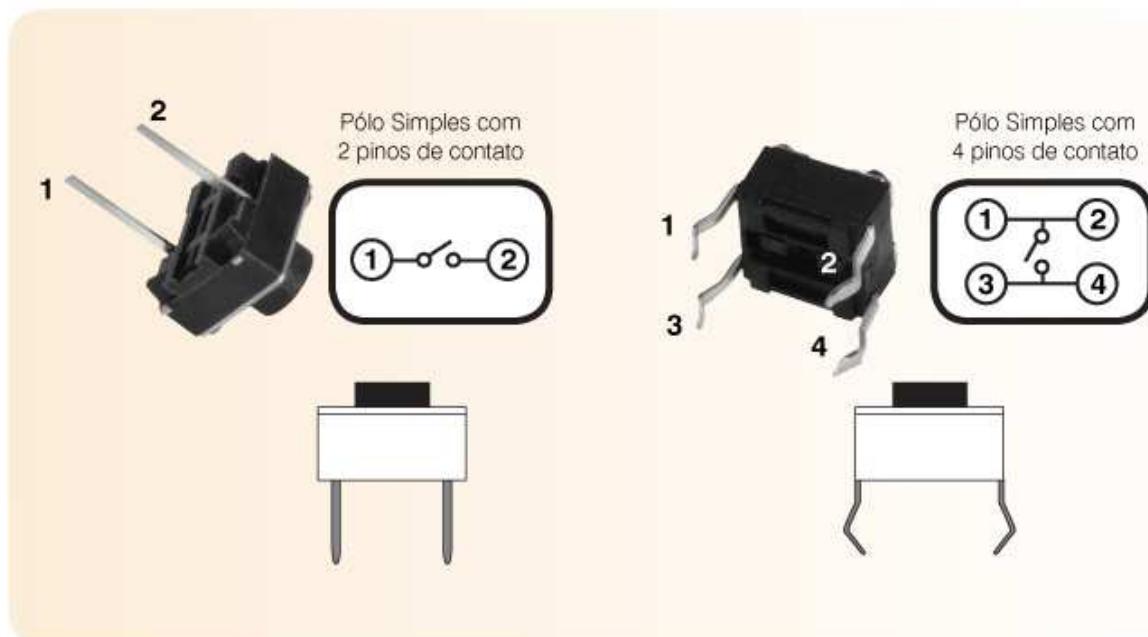


1 Led



1 resistor 220r (vermelho, vermelho, marrom)

Uma primeira observação: nos kits que compramos pela internet geralmente vem junto o botão (push button) de 4 pinos de contato. Se for usar esse botão neste nosso projeto, fique atento a forma como ele deve ser ligado. Veja o esquema abaixo:



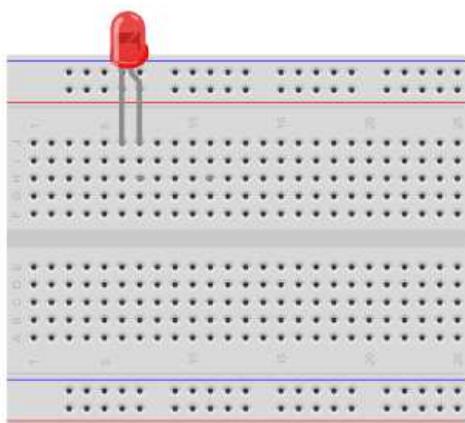
Fonte: http://www.dreaminc.com.br/sala_de_aula/9b-interruptores-mini-botao-de-pressao/

Observe que no botão de 4 pinos, na parte de baixo dele há um pequeno sulco. De um lado deste sulco então os pinos 1 e 2 que são interconectados tal como demonstra a figura acima. Do outro lado os pinos 3 e 4.

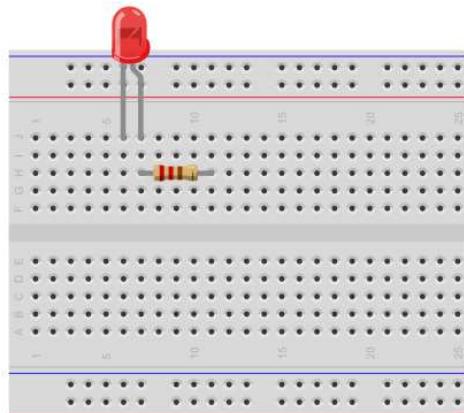
Uma segunda observação: os pinos do arduino além de fornecer energia fazem também a leitura da variação da corrente elétrica como já foi mencionado anteriormente. No caso de um botão ou de um sensor o que faz um microcontrolador como o que tem no arduino, não é “enxergar”, não é “sentir”, o botão em si, mas sim interpretar a energia fornecida.

A montagem:

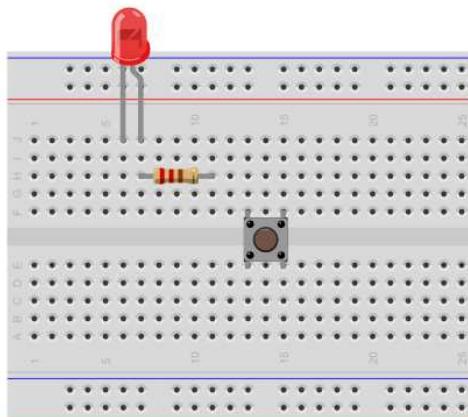
A montagem na protoboard é bem simples. Conectamos o led na coluna **6** da linha **J** tal como mostra a figura. A perninha do polo positivo do led ficará na coluna **7**.



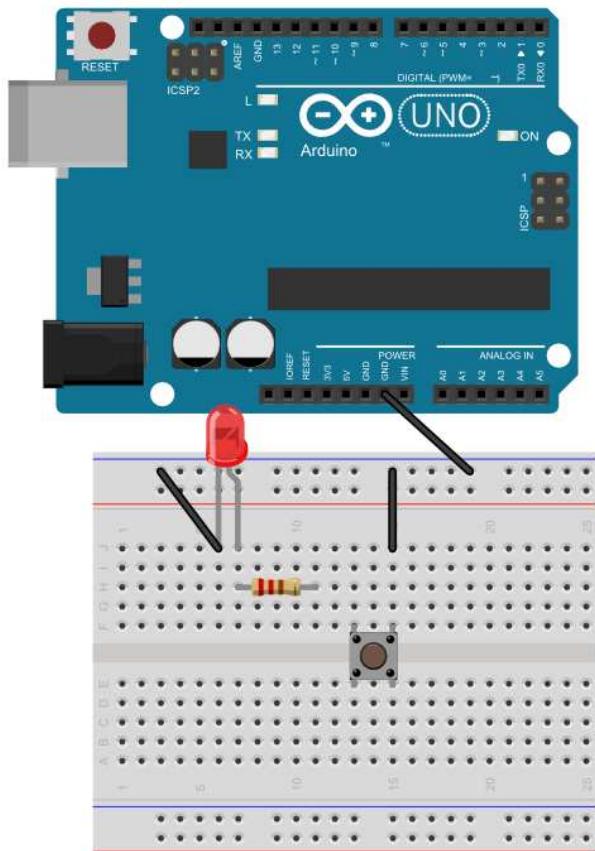
Adicionamos o resistor com uma perninha na coluna **7**, na linha **H**



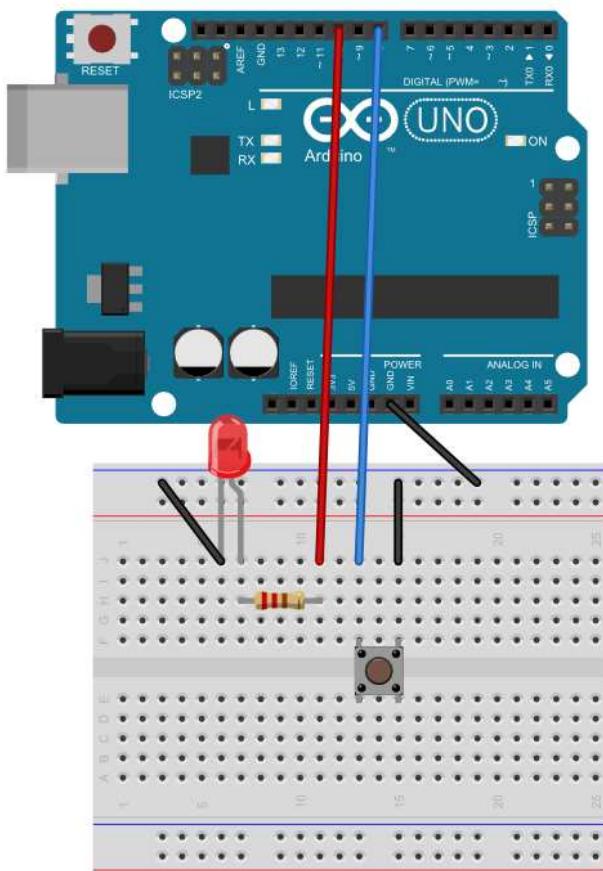
Depois colocamos o botão. Neste exemplo estamos utilizando o botão com 4 pinos. Caso tenha alguma dúvida sobre isso releia o tópico anterior. Vamos colocá-lo na linha **F** com um pino na coluna **13** e o outro na coluna **15**.



Vamos alimentar a linha azul da protoboard com um fio vindo do **GND** do Arduino. Usamos outro fio para alimentar a coluna **6** e outro para a coluna **15**.



Por último conectamos o pino **10** a coluna **11** no resistor (fio vermelho) e pino **8** a coluna **13**.



O código:

Digite o seguinte código na IDE do arduino, teste e passe a programação:

```
void setup()
{
    pinMode(8, OUTPUT);
    pinMode(10, INPUT);
    digitalWrite(10, 1);
}

void loop()
{
    int leitura = digitalRead(10);
    if (leitura == 0)
    {
        digitalWrite(8, 1);
    }
    else {digitalWrite(8, 0);}
}
```

Compreendendo o código...

Esse código apresenta uma função diferente e trabalha um dos operadores lógicos que ainda não vimos. Além disso, para esse tipo de montagem com botão, utilizamos na programação o conceito de resistores de elevação (PULL-UP). Esse conceito será explicado depois.

No **setup** dizemos para o arduino que usaremos o pino **8** como saída (**OUTPUT**). Esse pino terá a função de ligar e desligar o led. Colocamos o pino **10** como entrada configurando-o para leitura do botão. E escrevemos no pino **10** o valor **1**. Poderíamos ter escrito **HIGH** que teria o mesmo efeito. Com isso estamos ativando o **resistor de elevação** para esse pino (conceito que será explicado posteriormente).

Dentro da função **loop** criamos uma variável chamada **leitura**. Atribuímos como valor para essa variável o que está sendo lido no pino 10.

```
int leitura = digitalRead(10);
```

A função **digitalRead**, faz exatamente o oposto da **digitalWrite**. Enquanto a **digitalWrite** escreve um valor, a função **digitalRead** lê um valor em determinado pino.

Depois geramos uma condicional com a função **if** dando um parâmetro para verificar se é verdadeiro. Neste caso o parâmetro (o que está entre as aspas) é a comparação com o valor da variável **leitura**. É comparada se ela é igual a zero.

```
if (leitura == 0)
{
    digitalWrite(8, 1);
}
```

Ou seja, só será efetivada o que está dentro do corpo desta função (o que está dentro das chaves{ }) se a condição do parâmetro for verdadeira. Do contrário a programação segue para a a função **else**.

No parâmetro da função **if** quando comparamos a variável **leitura** são utilizados dois sinais de igual (==). *Isso porque nesta linguagem de programação usamos só um sinal de igual (=) quando queremos atribuir um valor, mas quando queremos comparar, fazer um teste de igualdade, usamos dois sinais de igual (==). Mais adiante explicaremos sobre operadores de comparação.*

Na função **else** estamos dizendo ao arduino para escrever no pino digital **8** o valor **LOW**.

```
else {digitalWrite(8, 0);}
```



Duas formas de escrever
o mesmo código!



```
else
{
    digitalWrite(8, 0);
}
```

Emitindo som com arduino

fonte de pesquisa:

<http://electronicpiece.blogspot.com.br/2012/02/buzzer-arduino.html>
<http://www.comofazerascoisas.com.br/projeto-arduino-como-emitir-sons-com-o-buzzer.html>
<http://www.ciandorobocomarduino.com/2013/09/como-usar-o-buzzer-som-no-arduino.html>
<http://www.arduinoecia.com.br/2013/06/sons-no-arduino.html>

Componentes necessários:

- 1 placa Arduino
- 1 cabo USB
- 1 protoboard
- Fios diversos - jumpers

1 resistor 220 Ω (vermelho, vermelho, marrom)

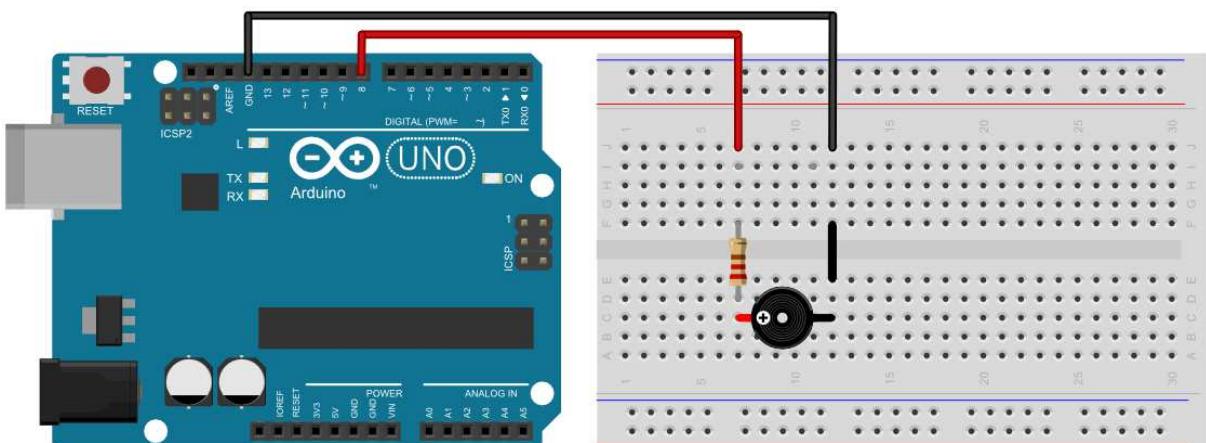


1 buzzer



A montagem:

A montagem na protoboard é muito simples. O único detalhe é que o buzzer é um componente que tem polaridade, mas é fácil de identificar qual é o seu fio positivo pois ele é indicado com o sinal “+”. Conectamos o **positivo** do buzzer no **resistor**. E conectamos o resistor no pino **8** do arduino. O outro pino do buzzer deve ser ligado no **GND**.



O código:

```
const int buzzer = 8;

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

void loop()
{
    tone(buzzer,1500);
    delay(500);
    noTone(buzzer);
}
```

Compreendendo o código...

Esse código apresenta duas funções novas. Na biblioteca do arduino encontramos a função **tone** e **noTone**. A primeira responsável por gerar sons a partir da frequência de 31Hz (som muito grave) até sons inaudíveis ao ouvido humano.

A função **tone** tem a seguinte estrutura:

tone(pino, frequência);

ou

tone(pino, frequência, duração);

Em **pino** escrever o pino do arduino que será conectado o buzzer, no caso colocamos a variável **buzzer** que continha o valor do pino que estamos usando: **8**. Na frequência colocamos um número com valor entre **31** e **10.000**. Acima de **10.000** o som já começa a ficar imperceptível dependendo do tipo de buzzer. Na nossa programação colocamos o valor de **1.500**. Convém testar os valores. Temos a opção de colocar ou não a duração, o tempo que o som irá tocar em milisegundos.. Nesta programação optamos por não colocar nada.

noTone(pino);

Já a função **noTone** simplesmente para a geração do som executado pela função **tone**. O único parâmetro para esta função é o pino que deve ser desativado.

Lendo a intensidade luminosa

fonte de pesquisa:

<http://robotpig.net/UAVs-blogs/arduino-and-processing- 1772>

<http://communityofrobots.com/tutorial/kawal/how-use-ldr-arduino>

<http://labdegaragem.com/profiles/blogs/tutorial-como-utilizar-a-mini-fotocelula-ldr>

<http://www.comofazerascoisas.com.br/projeto-arduino-sensor-de-luz-ldr-com-leds.html>

Componentes necessários:

1 placa Arduino

1 cabo USB

1 protoboard

Fios diversos - jumpers

1 resistor de $10\text{K}\Omega$

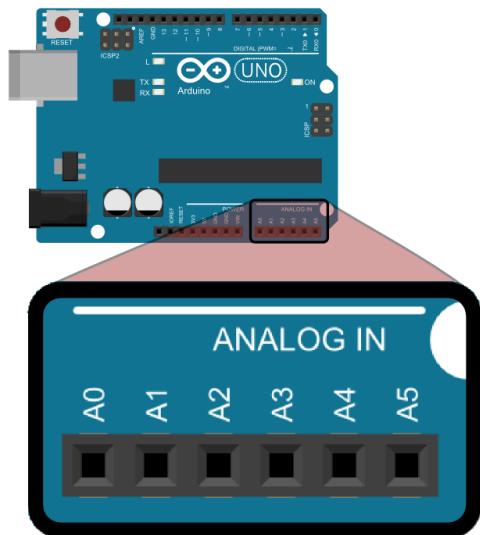


1 LDR

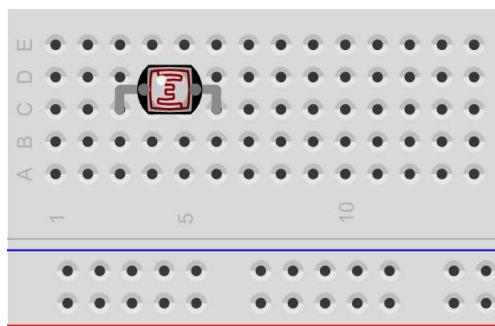


A montagem:

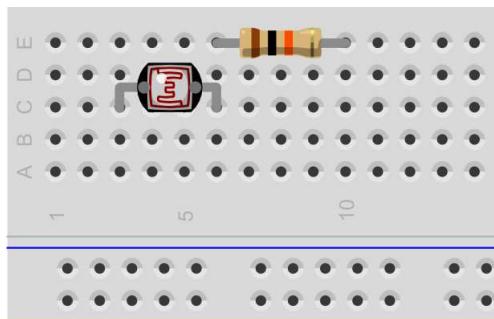
Nesta montagem usaremos os **pinos analógicos** do arduino.



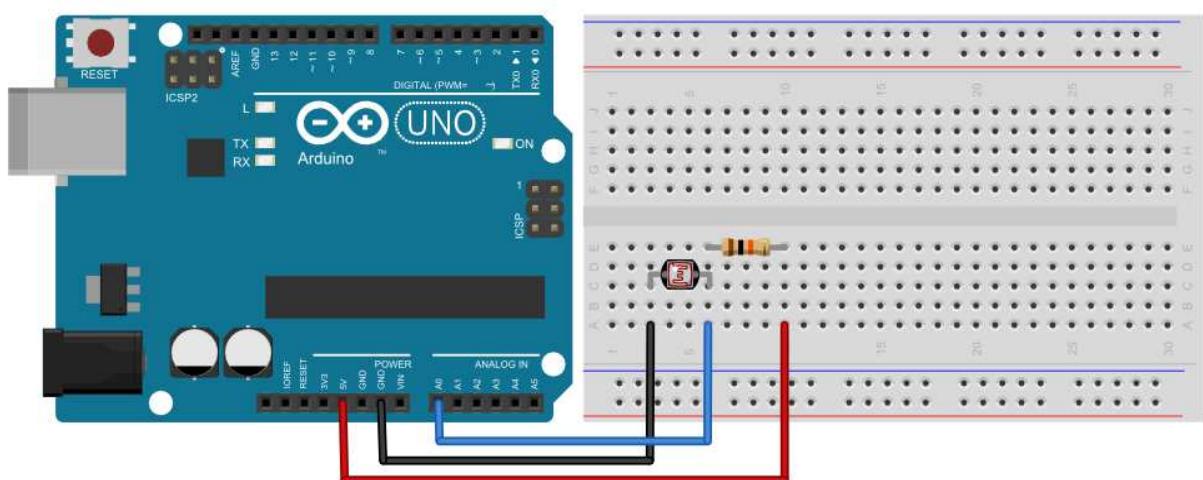
Vamos instalar primeiro o componente **LDR**. Vamos colocá-lo na linha **C** com um terminal na coluna **3** e o outro na coluna **6**. Tal como demonstra a figura abaixo:



Na sequencias colocaremos o resistor de **10k** na linha **E** com um dos terminais na coluna **6**, onde já tem um dos terminais do **LDR**, e o outro terminal na coluna **10**. É importante que um dos terminais, tanto do **LDR**, quanto do **resistor**, estejam na mesma coluna, neste caso a **6**.



Por último fazemos as ligações entre os pinos e o arduino com os jumpers. A coluna **3** no **GND**, a coluna **6** no pino analógico **A0** e a coluna **10** no **5v**.



O código:

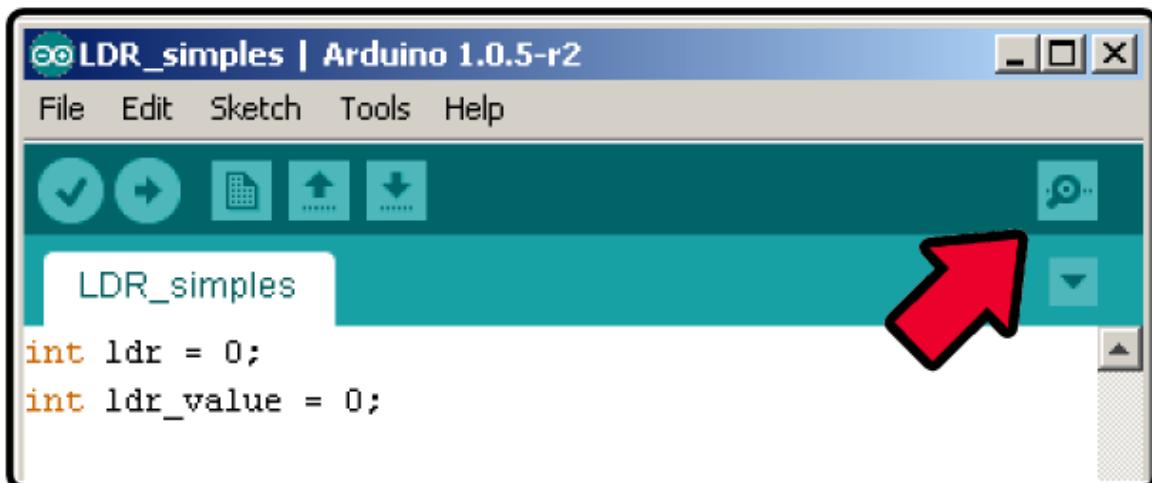
O código também trás elementos novos:

```
int ldr_value = 0;

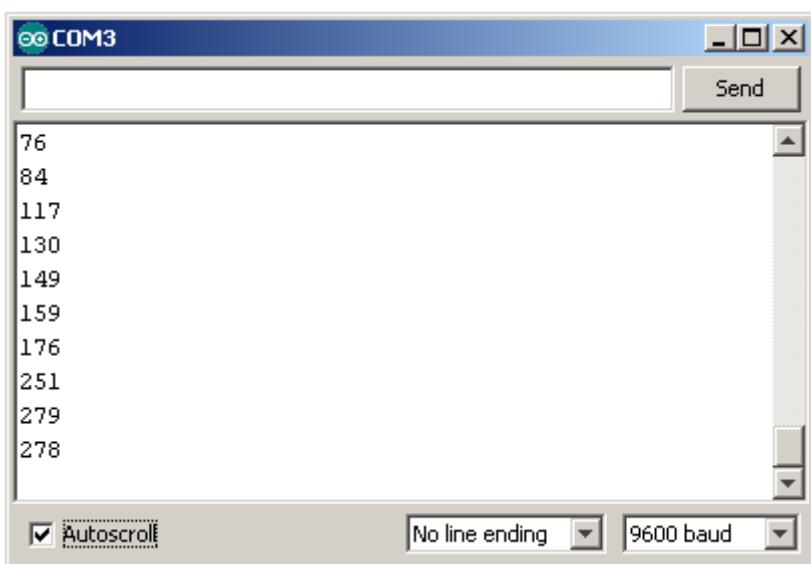
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    ldr_value = analogRead(A0);
    Serial.println(ldr_value);
    delay(500);
}
```

Agora clique no Serial Monitor para visualizar o valor da intensidade luminosa. O ícone do Serial Monitor fica no canto direito, conforme mostra a figura. Também é possível acessá-lo digitando ctrl+shift+M.



Ao aproximar a mão do LDR, impedindo sua exposição a luz, veremos uma alteração dos valores que são gerados.



Compreendendo o código...

Começamos declarando uma variável de tipo inteira (**int**) chamada de **ldr_value** e determinamos seu valor como **zero**.

```
int ldr_value = 0;
```

Dentro da **setup** inicializamos a porta de comunicação serial com o **Serial.begin** informando ao arduino que queremos utilizar a comunicação serial com a taxa de transferência de **9600** bits por segundo.

```
Serial.begin(9600);
```

Aqui cabe entendermos o que é **Serial**:

Serial é um tipo de comunicação de dados usados entre dois dispositivos eletrônicos. Por exemplo: entre dois computadores, ou entre um computador e uma impressora, ou entre dois celulares. O nome serial se deve a natureza deste tipo de comunicação que envia e recebe informação em fila, em série, os bits são enviados um de cada vez. Para quem está acostumado com o uso do RCX sabe que precisa ligar a torre na porta serial do computador, na verdade a uma porta padrão RS-232. Esse tipo de conexão foi sendo substituído pelo USB, que também é serial, aliás, USB significa **Universal Serial Bus**.

Pesquise sobre o assunto em http://pt.wikipedia.org/wiki/Comunica%C3%A7%C3%A3o_serial.



Porta serial padrão RS-232 de um computador.

A comunicação serial possibilita o Arduino se comunicar com um computador ou com outros dispositivos. Usamos ela para diversas aplicações simples e complexas. Podemos utilizá-la para fazer comunicação via bluetooth por exemplo. A comunicação ocorre através dos pinos digitais 0 (RX) e 1 (TX), assim como uma conexão USB. Por tanto, toda vez que usamos a função **Serial** os pinos 0 e 1 **não poderão** ser utilizados.

Alguns funções que usamos para declarar a comunicação serial:

1. **Serial.begin(velocidade)** - Essa é uma função de configuração. É a primeira função a ser utilizada quando vai trabalhar com a comunicação serial. Geralmente utilizamos o valor **9600** como parâmetro. Essa é a taxa de transferência em bits por segundo (baud) para transmissão de dados pelo padrão serial. Para comunicação com um computador use uma destas taxas: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 57600, 115200.
2. **Serial.print(dado)** - Essa função envia um determinado dado pela porta serial. Esse dado pode ser uma variável, pode ser uma expressão, um número.
3. **Serial.println(dado)** - Essa função faz o mesmo que a **Serial.print** com o diferencial enviar também para a porta serial a indicação de mudança de linha.
4. **Serial.read()** - Essa função lê os dados que estão entrando pela porta serial.

Voltando ao código:

```
ldr_value = analogRead(A0);
```

Mudamos o valor da variável **ldr_value** dizendo que ela deve armazenar a leitura do pino analógico **A0**.

IMPORTANTE: Agora estamos usando a função **analogRead** porque estamos lendo os pinos analógicos e não os digitais. Na sequência explicaremos com mais detalhes.

```
Serial.println(ldr_value);
```

Executamos a função **Serial.println** para enviar pela porta serial a variável **ldr_value** que, como foi explicado

anteriormente, contém o valor da leitura do LDR instalado na porta analógica A0.

delay(500);

A função delay, nesta programação, indica o tempo de espera até ser feita uma nova leitura do LDR.

PROJETOS MAIS COMPLEXOS

Protegendo o arduino

fonte de pesquisa:

<https://www.youtube.com/watch?v=j3kn4-po5hs>

http://www.dreaminc.com.br/sala_de_aula/entendendo-o-atomo/



Recordando conceitos de eletrônica

Disponível em: <http://www.dreaminc.com.br/>

Todo componente eletrônico precisa de cuidados especiais obedecendo os limites das suas características e composição. Com o arduino não é diferente. O arduino Uno, que é a referência para este tutorial, tem um limite de tensão e corrente que não podemos esquecer!



O arduino pode fornecer o máximo de **40mA** em uma porta! Isso quer significar que não é possível ligar componentes eletrônicos que demandem mais corrente que isso! Isso é extremamente importante observar para que o Arduino não seja danificado. O microcontrolador do arduino vai esquentar e queimar. Para termos a exata ideia do que isso representa vejamos as ilustrações abaixo:



Podemos alimentar diretamente o **led** em um arduino porque ele demanda **20mA** de corrente, como demonstra na figura acima (alguns leds consomem apenas **7mA**). Mas esse relé que está na segunda figura, não pode ser alimentado diretamente pelos **pinos digitais** arduino porque precisa de **96mA** o dobro que ele pode fornecer

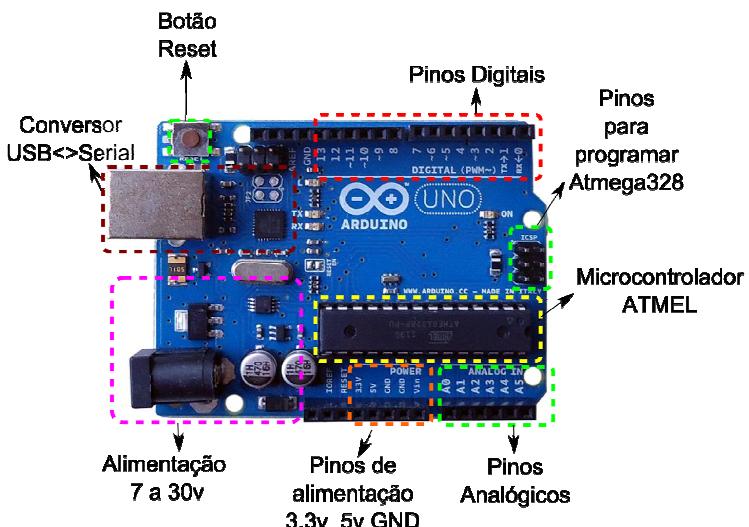
sem danificar o arduino. E um motor **jamais** pode ser alimentado pelos pinos do arduino porque demanda uma corrente muito elevada! Dependendo do motor pode ultrapassar **1A**.

Como saberei a corrente de um componente? As especificações técnicas dos componentes eletrônicos são encontradas nos datasheets. Datasheet é um documento que apresenta de forma resumida, todos os dados e características técnicas de um equipamento ou produto. Veja o exemplo do datasheet de um relé: <http://www.metaltex.com.br/downloads/A.pdf>.



Outro fato importe: embora o arduino possa oferecer 40mA em uma porta, temos que ter o cuidado de não utilizar mais do que **200mA** no conjunto. Um exemplo não podemos usar mais do que 5 portas digitais do arduino, fornecendo uma corrente de 40mA cada uma delas por que excederia os 200mA. As portas digitais do arduino são usadas para **controlar** os dispositivos e **não como fonte** de alimentação.

Mas e os pinos de alimentação?



Os pinos de alimentação do arduino fornecem 3,3v e 5v de tensão, mas tem como limite **200mA** de corrente. Neste caso um relé pode ser conectado pelos pinos de alimentação por que precisa de **96mA** para funcionar. Mas o motor **jamais** poderá ser alimentado diretamente ao pino de alimentação por exceder o limite da corrente.

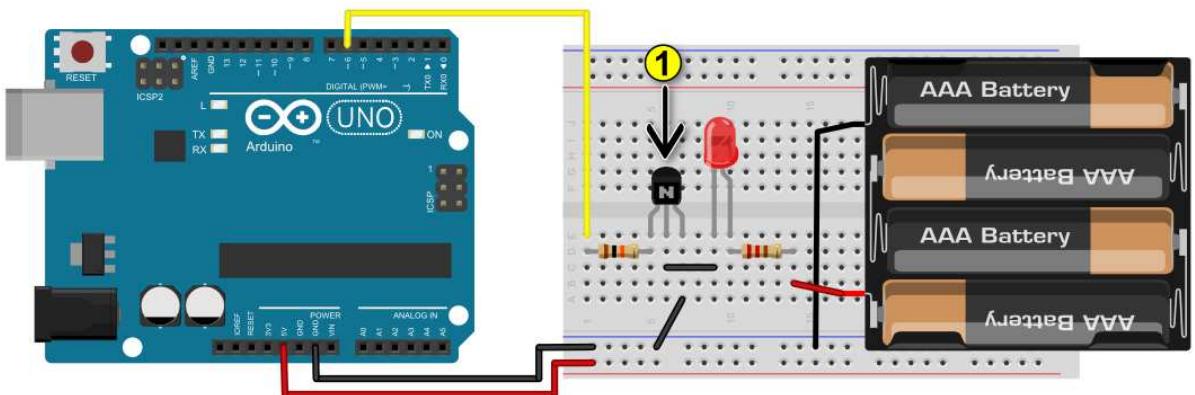
Então como ligamos componentes eletrônicos que necessitam de corrente maior do que o arduino pode fornecer? Criando um circuito de proteção e ativação.

Círculo de proteção e circuito de ativação

Utilizamos um circuito de proteção para impedir que o arduino seja danificado por uma determinada montagem. Ao ativar um motor dc, por exemplo, temos que pensar de que forma a corrente reversa não vai danificar o arduino. Há dispositivos que precisam ser acionados com auxílio de um circuito de ativação, além do circuito de proteção. Veja o esquema abaixo:



Como montamos um circuito de proteção? Fazemos um circuito de proteção com um transistor. O transistor funciona como uma espécie de chave que vai mediar a interação entre o arduino e o circuito de ativação ou o dispositivo a ser acionado. Veja um exemplo:



Nesta montagem estamos usando o transistor **bc546** (número 1, indicado pela seta) que está intermediando o fornecimento de energia para o led. A fonte de energia do led são as pilhas a direita, desta forma o led não entra em contato diretamente com a porta do arduino. Por isso, mesmo se houvesse um curto circuito, o transistor é que seria danificado e não o arduino. A porta 6 do arduino vai ativar o transistor e este é que deixará passar corrente permitindo ligar o led.

Mas, e se fosse um motor que demanda mais corrente que um led? Teríamos que usar um outro componente, isso porque esse transistor (bc546) também queimaria se ligássemos um motor diretamente nele. Para isso é que usamos o circuito de ativação. Que poderia ser um circuito contendo um relê.

Acionando um relé

fonte de pesquisa:

<http://www.arduinoecia.com.br/2013/05/ligando-uma-lampada-com-rele.html>

<http://fritzing.org/projects/acionamento-de-um-rele-com-o-arduino>

Componentes necessários:

1 placa Arduino

1 cabo USB

1 protoboard

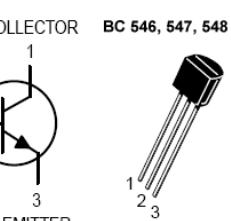
Fios diversos - jumpers



Rele 5v



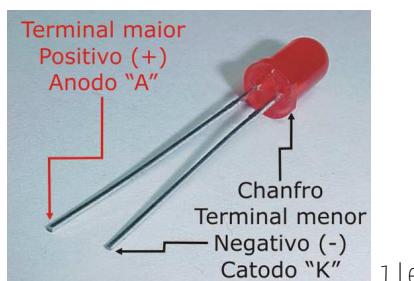
1 Diodo 1N4007 (ou similar)



1 Transistor BC548 (ou equivalente NPN)



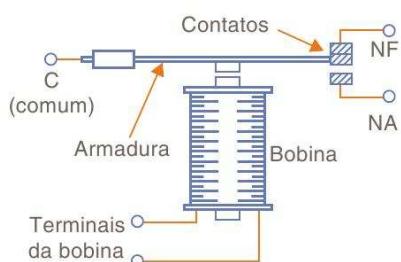
2 resistores de 10 K



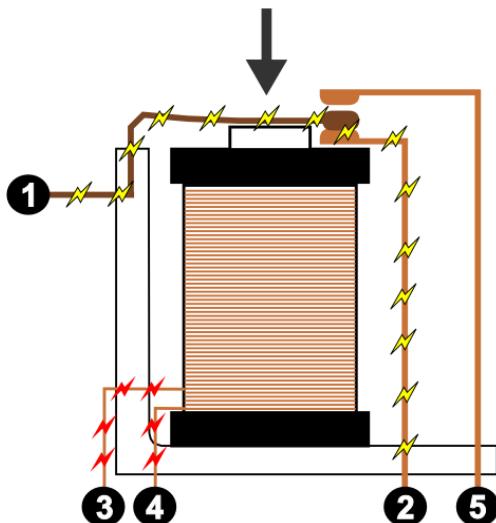
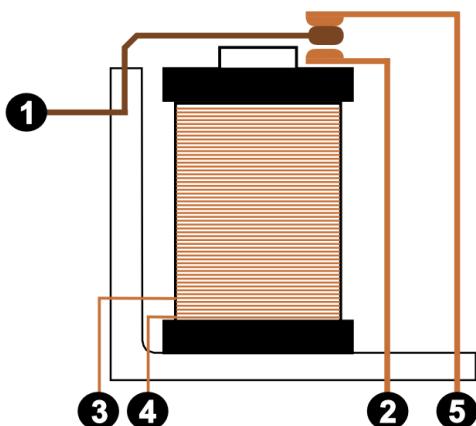
1 Led

Esquema de funcionamento de um relé

Disponível em: http://2.bp.blogspot.com/-CivHe8KfW0Q/UU0FpHF1VI/AAAAAAAQE/99JAI-EA2bg/s1600/figura_1_testando_reles.jpg



Um relé funciona como se fosse uma chave. Uma bobina ao ser percorrida por uma corrente, gera um campo magnético no seu núcleo que atrai um ou vários contatos elétricos, permitindo ligar, desligar ou comutar um circuito elétrico externo.



- 1** Comum - ligar o fio que irá ao led
- 2** NA (NO) - contato que fica **Normalmente Aberto**
- 3** Terminal da bobina
- 4** Terminal da bobina
- 5** NF (NC) - contato que fica **Normalmente Fechado**

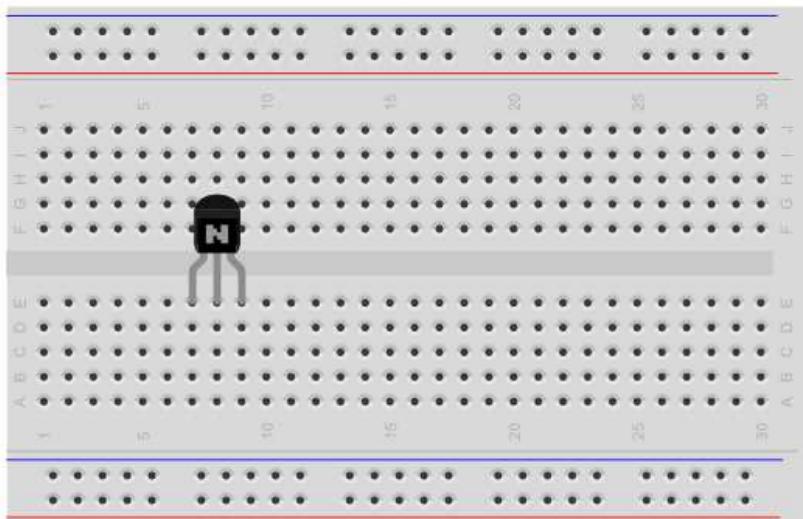
Acima temos uma representação do que acontece com os relés. O relé que usaremos tem 5 pinos | terminais. Quando deixamos passar corrente entre os terminais da bobina (3 e 4) ela é ativada, transformando-se em um eletroímã que “puxa” a placa metálica, a armadura, fazendo passar corrente entre o comum (1) e o terminal NA (2).

Ativando o relé com o Arduino

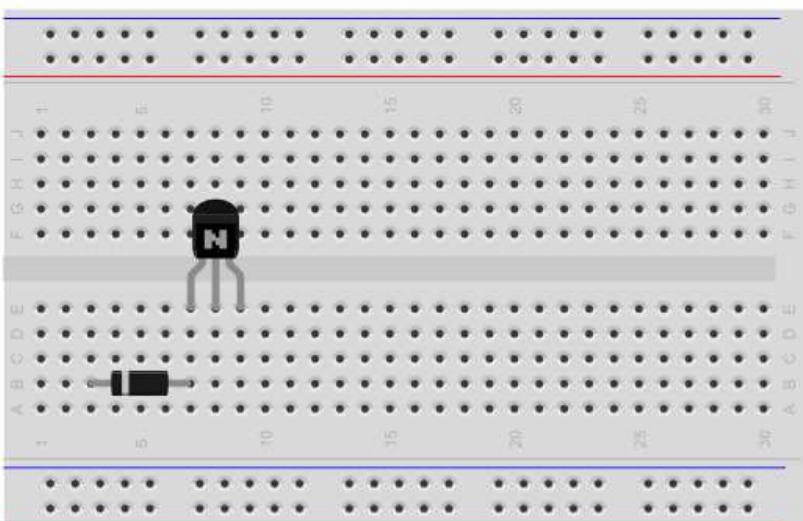
Disponível em: http://4.bp.blogspot.com/-KZSHBJTXZYUYIgO2ueq8I/AAAAAAAQAO8/saFsT32QTfE/s1600/Rele+e+transistor_LED.jpg

Faça a montagem dos componentes na protoboard tal como dispõe a figura abaixo. A montagem deve ser exatamente igual para funcionar corretamente.

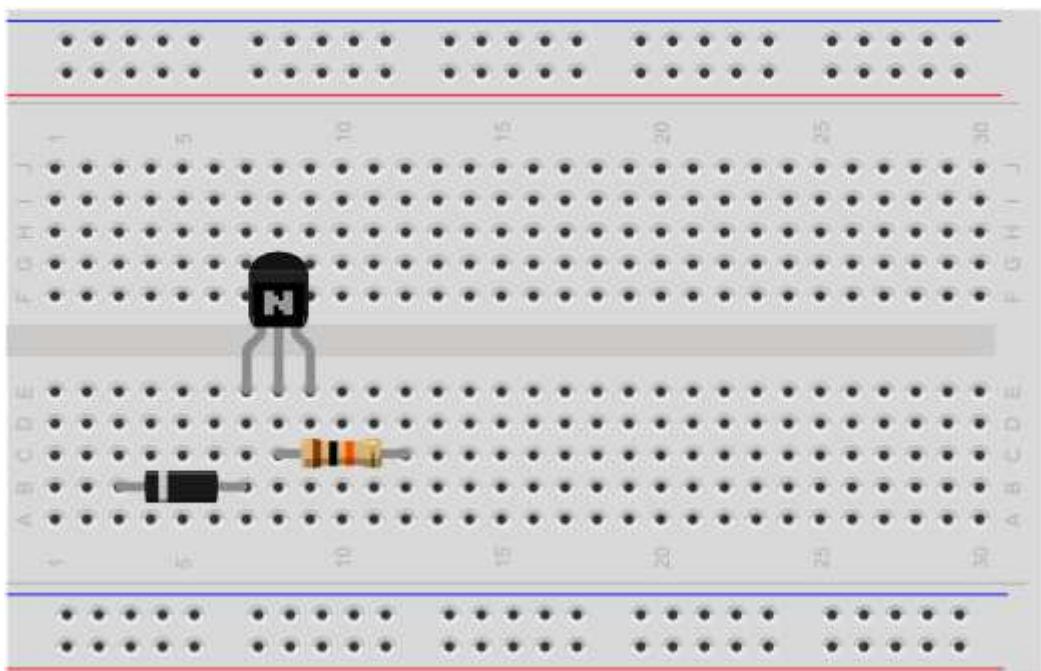
Primeiro coloque o transistor na linha **E** com a parte “arredondada” virada para trás e a parte “plana” voltada para frente. **Cuidado** para posicionar corretamente o transistor! Ele deve ficar encaixado nas colunas 7, 8 e 9.



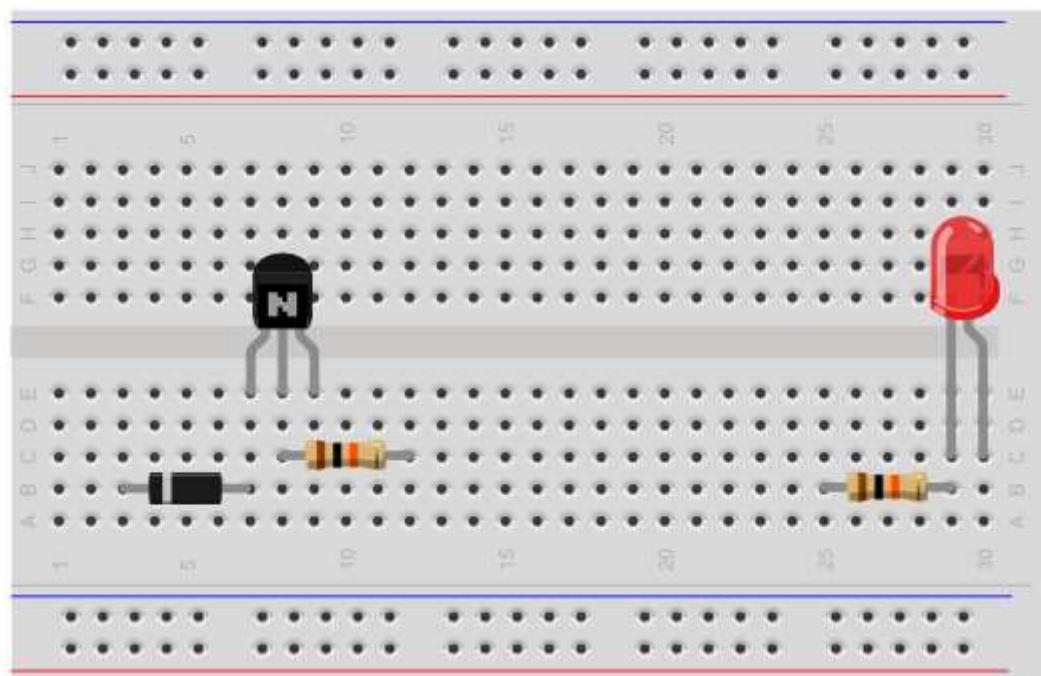
Acrecente o **diodo** na linha **B** conforme indica a figura abaixo. O perninha do catodo deve estar conectado na coluna **3** e o anodo na **7** na mesma coluna que a primeira perninha do transistor. Relembrando: **A, anodo, +** e **positivo** são sinônimos. Da mesma forma, **K, catodo, -** e **negativo**, também são sinônimos. O catodo fica no lado onde tem uma linha.



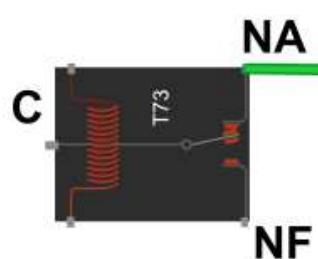
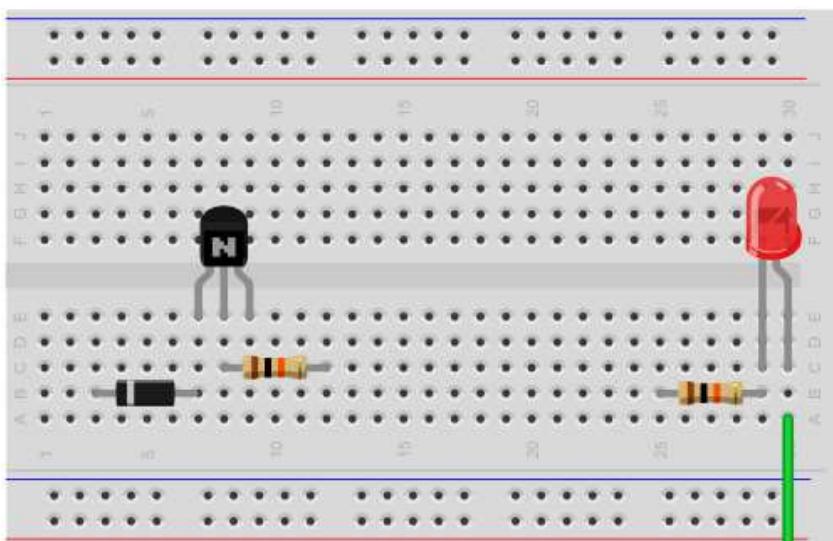
Colocamos um dos resistores na linha **C** no terminal **8** e no **12**.



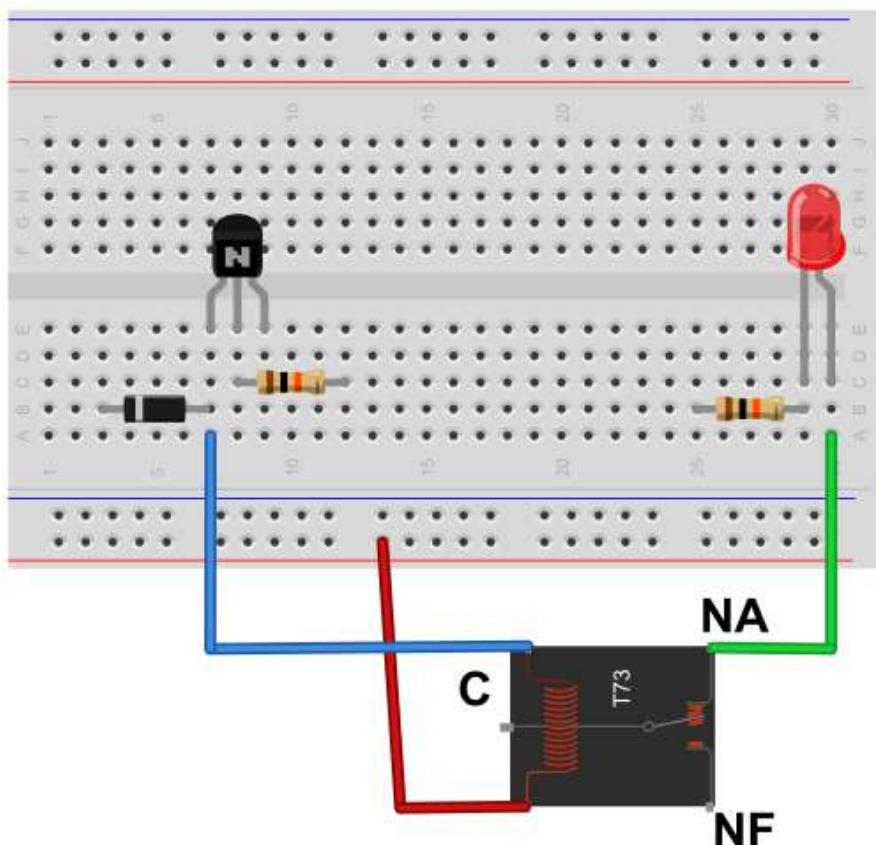
Agora colocamos o led na linha **C** com o catodo na coluna **29** e o anodo na coluna **30**. O resistor colocamos na linha **B** com uma perinha na coluna **25** e a outra na coluna **29**, a mesma do catodo do led.



Agora vamos trabalhar com o relé. Esse componente tem 5 pinos. Vamos usar apenas 4. Caso tenha dificuldade com os pinos, releia sobre os relés, logo acima. Vamos enrolar uma ponta de fio no pino **NA** do relé e ligar a outra ponto na coluna **30**, a mesma do anodo do led.

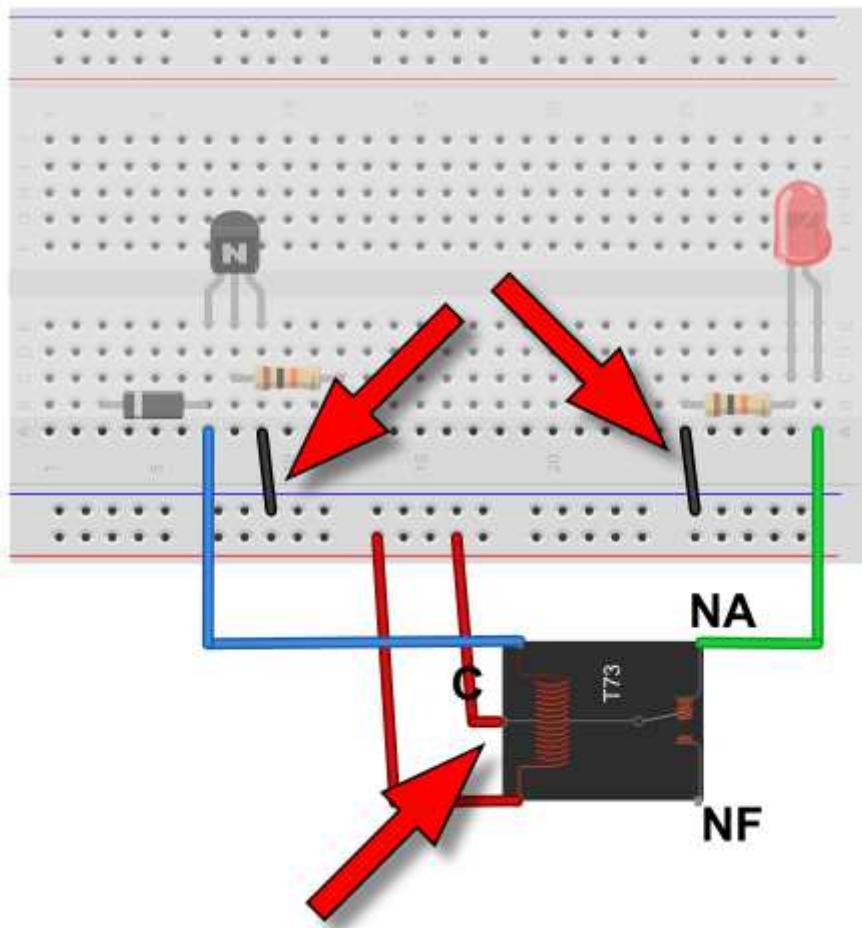


Agora conecte um fio na coluna **7** da protoboard até o terminal da bobina do relé. Conecte outro fio da linha vermelha da protoboard para o outro terminal da bobina do relé, conforme mostra a figura.

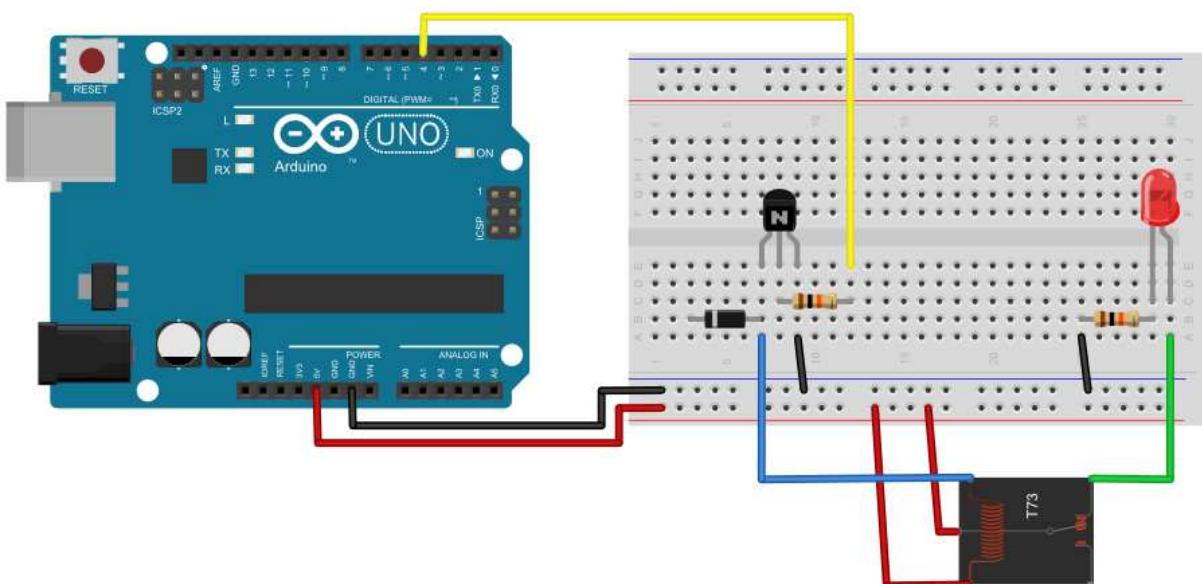


Conecte um fio na linha vermelha da protoboard ao terminal **comum** do relé. Também conecte um fio entre a

coluna **9** e a linha azul da protoboard e um outro entre a coluna **25** e a linha azul. Veja as indicações das setas na figura abaixo.



Por fim vamos colocar um fio saindo do **4** do arduino para a coluna **12** da protoboard. Além disso, alimentamos a protoboard colocando o **GND** do arduino na linha azul da protoboard e o **5v** na linha vermelha.



Vamos explicar a função de cada um dos componentes deste circuito: A função do diodo é impedir uma corrente reversa que poderia danificar o arduino. O transistor, funciona como um chave que será ativada quando

mandarmos energia do arduino para ele, através do pino 4. Entre o fio amarelo (que vem do pino 4) e o transistor, temos um resistor de 10k. A função deste resistor é limitar a corrente que irá para o transistor. Embaixo temos o relé. Logo acima dele, temos um outro resistor e um led. Este resistor tem a função de limitar a corrente para não queimar o led.

O código:

O código para ligar um relé é muito simples. Ele é praticamente igual ao código para ligar o led pisca-pisca. A diferença maior é o valor do **delay**. Digite o código abaixo na IDE do arduino e faça o teste.

```
int sinalparaorele = 4; //define a porta a ser utilizada para o acionamento do rele

void setup()
{
    pinMode(sinalparaorele, OUTPUT); //Define o pino como saída
}

void loop()
{
    digitalWrite(sinalparaorele, HIGH); //Aciona o rele
    delay(5000); //Aguarda 5 segundos
    digitalWrite(sinalparaorele, LOW); //Desliga o rele
    delay(5000); //Aguarda 5 segundos e reinicia o processo
}
```

Usando um potenciômetro

fonte de pesquisa:

<http://www.ajudino.com/2013/04/3-utilizando-potenciometro-no-arduino.html>

<http://www.comofazerascoisas.com.br/projeto-arduino-como-controlar-um-led-com-potenciometro.html>

<http://deivson-robotica.blogspot.com.br/2012/01/ligar-led-com-valor-do-potenciometro.html>

http://www.pnca.com.br/index.php?option=com_content&view=article&id=67:pwm&catid=42:saiab-mais&Itemid=150

Componentes necessários:

1 placa Arduino

1 cabo USB

1 protoboard

Fios diversos - jumpers

1 led

1 resistor de 150Ω (150 ohms)  (marrom, verde, marrom)

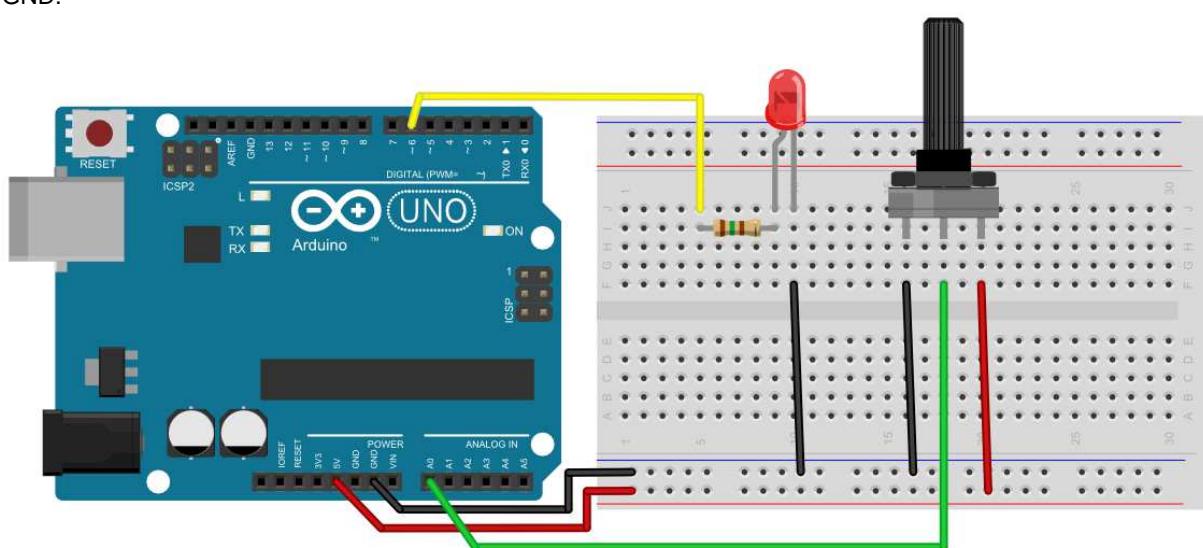


1 potenciômetro 10KΩ

A montagem:

Essa montagem não é difícil. Os 3 pinos do potenciômetro se encaixam perfeitamente na protoboard. Mas fique atento para verificar exatamente em qual coluna eles realmente estão encaixados por que eles “somem” debaixo do potenciômetro. O pino que fica no meio do potenciômetro deverá ser conectado a pino analógico **A0**. O pino da esquerda ligado ao **GND** e o pino da direita ao **5v**.

A montagem do led já foi feita. Relembrando: colocar o anodo do led conectado ao resistor e deste para o pino digital **6**. O Catodo no led vai para o **GND**.



O 1º código - programando um pisca-pisca:

Vamos utilizar nesta montagem 2 diferentes códigos. Começamos com o mais fácil de entender:

```
int led = 6;  
int potenciometro = A0;  
int tempo = 0;  
  
void setup() {  
    pinMode(led,OUTPUT);  
}  
  
void loop() {  
  
    tempo = analogRead(potenciometro);  
  
    digitalWrite(led, HIGH);  
    delay(tempo);  
    digitalWrite(led, LOW);  
    delay(tempo);  
}
```

Compreendendo o 1º código...

Iniciamos este código declarando três variáveis de tipo **int**. A variável **led** para indicar o pino digital onde o led estará conectado. A variável **potenciometro** que indicará o pino analógico onde o potenciômetro está conectado. E a variável **tempo** que armazenará o valor de leitura do potenciômetro.

```
int led = 6;  
int potenciometro = A0;  
int tempo = 0;
```

Depois configuramos o pino que será conectado o led como pino de saída de dados, dentro do **setup**:

```
pinMode(led,OUTPUT);
```

Dentro do **loop**, primeiro determinamos que o valor da variável tempo seja alterado para o que o pino analógico está lendo. Lembrando que o pino analógico é a variável que está dentro do parênteses:

```
tempo = analogRead(potenciometro);
```

Na sequência entra a programação do led pisca-pisca. Escrevemos no pino (representado pela variável **led**) o valor **HIGH** deixando-o em nível alto o que acende o led. Determinamos que o tempo de espera, o **delay**, será a leitura do potenciômetro (que está armazenado na variável **tempo**). Depois escrevemos no pino “**led**” o valor **LOW** deixando-o em nível baixo o que apaga o led. E mais uma vez determinamos o tempo conforme a leitura do potenciômetro.

```
digitalWrite(led, HIGH);
delay(tempo);
digitalWrite(led, LOW);
delay(tempo);
```

Há um conceito importante subentendido nesta programação: a diferença entre pinos analógicos e pinos digitais. Em que consiste esta diferença?

Como já foi descrito no início desta publicação, os **pinos digitais** trabalham por padrão com duas condições possíveis sendo colocados em **nível alto** (true ou HIGH ou 1 com uma saída de 5v) ou em **nível baixo** (false ou LOW ou 0 com uma saída de 0v). Fogem a esse padrão os pinos digitais **PWN** que explicaremos adiante.

pinos digitais
2 estados

Algumas vezes necessitamos que o Arduino interprete com mais precisão a tensão aplicada a um pino. Para isso utilizamos a Leitura Analógica. Os **pinos analógicos** permitem que o Arduino traduza a tensão em valores que variam de 0 até 1023. Isso é realizado por um circuito interno, chamado Conversor Analógico Digital (CAD).

pinos analógicos
1024 estados

Os pinos analógicos têm uma sensibilidade e uma precisão muito maior para vários tipos de projetos. Nesta programação estamos usando essa sensibilidade para fazer o ajuste do tempo.

Quando giramos o ômbolo do potenciômetro totalmente para a direita, no sentido horário, ajustamos o tempo em

1024 milionésimos de segundo fazendo com que o **led** pisque neste intervalo de tempo. A medida que fazemos o giro no sentido contrário diminuímos esse intervalo de tempo aumentando a velocidade das piscadas do led.

O 2º código - programando a intensidade do led com PWN:

Vamos escrever um novo código utilizando a mesma montagem. Desta vez utilizaremos do conceito de **PWN** que explicaremos a seguir. Primeiro o código:

```
int led = 6;
int potenciometro = A0;
int valor = 0;

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

void loop()
{
    valor = analogRead(potenciometro);
    analogWrite(led, valor / 4);
}
```

Compreendendo o 2º código...

Iniciamos este código declarando três variáveis de tipo **int** exatamente como fizemos no código anterior, apenas substituindo a variável **tempo** pela **valor**. Relembrando: a variável **led** para indicar o pino digital onde o led estará conectado. A variável **potenciometro** que indicará o pino analógico onde o potenciômetro está conectado. E a variável **valor** que armazenará o valor de leitura do potenciômetro.

Então configuramos o pino que será conectado o led como pino de saída de dados, dentro do **setup** exatamente igual como o código anterior.

Dentro da função **loop** programamos para que a variável **valor** armazene o que está sendo lido pelo potenciômetro:

```
valor = analogRead(potenciometro);
```

Depois vem a novidade.

```
analogWrite(led, valor / 4);
```

Usamos a função **analogWrite** que vai escrever um valor analógico em um pino. Nesta linha de programação estamos acionando a função **analogWrite** determinando que no pino **6** (que é o valor da variável **led**, portanto onde está led leia-se “6”) deve ser escrito o resultado de uma operação matemática “**valor /4**”.

Poder parecer estranho em um primeiro momento, mas não estamos dividindo a palavra **valor** por **4**. É o conteúdo do que está armazenado na variável **valor** que será dividido por 4. A variável **valor** armazena a leitura do potenciômetro que está no pino analógico.

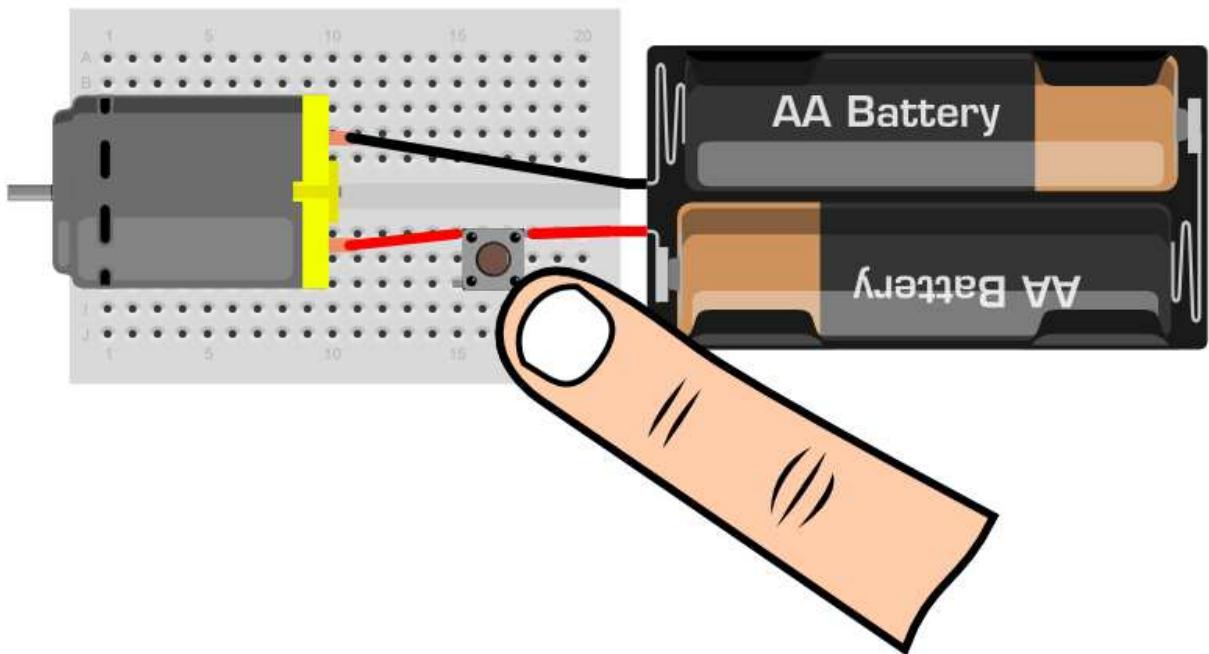
Mas porque estamos usando uma função analógica, **analogWrite**, em um pino **digital**? Porque estamos usando um pino PWN.

A Modulação por Largura de Pulso, ou **PWM** (do inglês Pulse Width Modulation), é utilizada em alguns pinos digitais do arduino. No arduino Uno os pinos que permitem o PWM são os de número 3, 5, 6, 9, 10 e 11. Esses pinos estão identificados na placa do arduino com um sinal de ~ ou com a sigla **pwm**.

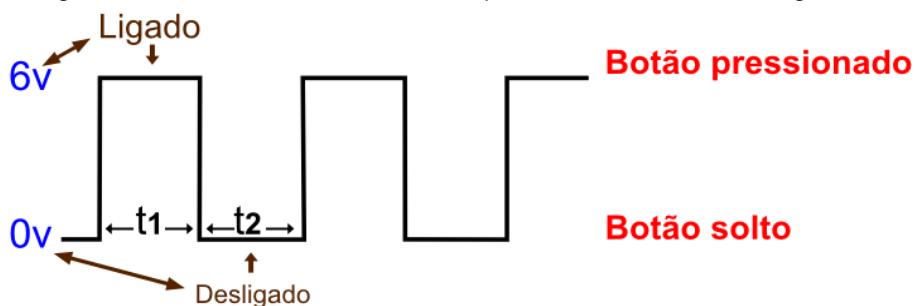
Já vimos que o padrão dos pinos digitais permitem apenas **2** estados: em **nível alto** (true ou HIGH ou 1 com uma saída de 5v) ou em **nível baixo** (false ou LOW ou 0 com uma saída de 0v). Também vimos que os pinos analógicos permitem **1024** estados. O pinos **PWN** permitem **255** estados. São os mesmos pinos digitais mas que oferecem uma saída de tensão variável, ao invés de 5v ou nada.

PWN? Me explica melhor o que é isso?

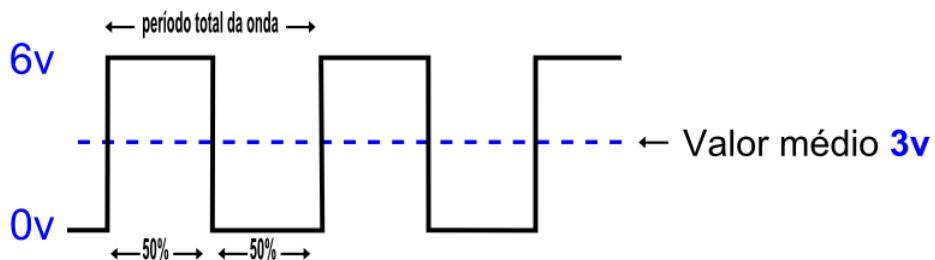
Para entender melhor o funcionamento do **PWN** vamos imaginar um circuito formado por um pack de 4 pilhas, um motor e um interruptor com uma velocidade muito rápida de acionamento. Neste, temos um interruptor que quando acionado faz com que o motor receba 6v e funcione com 100% de potência. Quando o interruptor não está precionado, o motor não recebe energia e simplesmente não funciona.



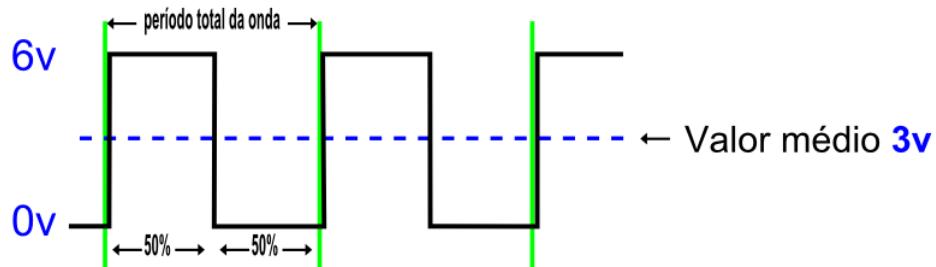
Vamos supor que consigamos pressionar e soltar o interruptor um grande número de vezes por segundo com intervalos regulares, de tal forma que metade do tempo ele fica ligado e metade desligado. O resultado seria uma onda quadrada como mostra a figura abaixo:



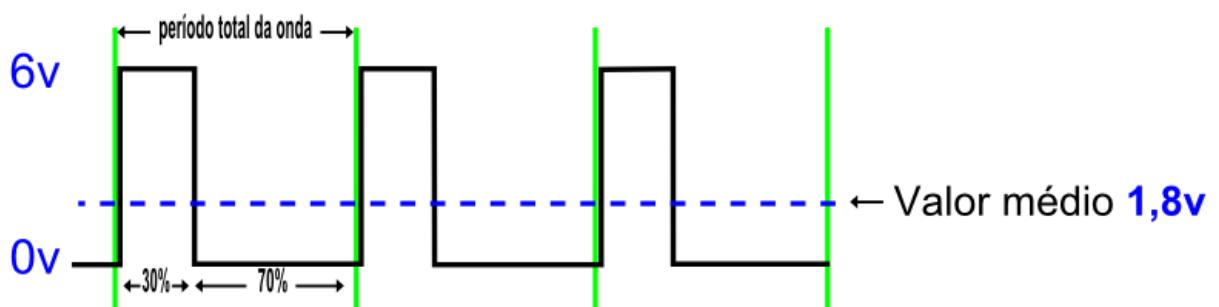
No exemplo o tempo t_1 corresponde ao tempo que o interruptor fica precionado e t_2 o tempo que ele fica livre. Como neste caso t_1 é igual a t_2 , durante a metade do tempo o motor recebe a tensão de 6v e na outra metade ele recebe 0v. A tensão média, desta figura, aplicada ao motor é neste caso de 3v, ou seja, 50% da tensão vinda da bateria.



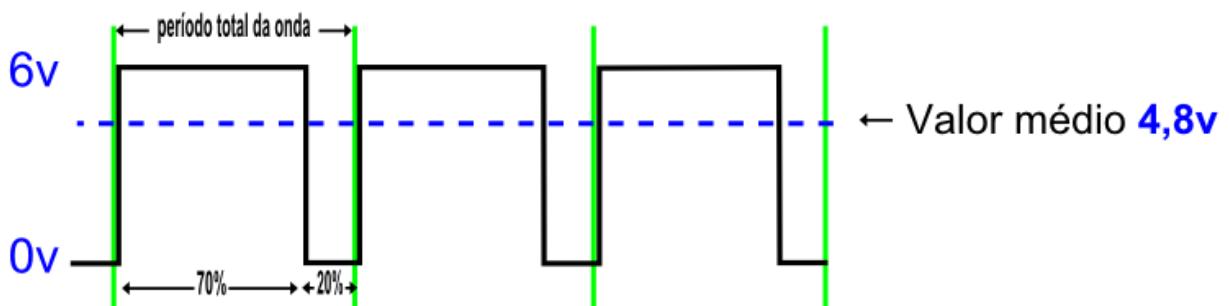
É claro que não é possível usar um interruptor em um circuito com PWM, pois não conseguiríamos pressioná-lo na velocidade necessária. O arduino executa esta técnica gerando os pulsos que são entregues sempre com uma mesma taxa (aproximadamente 500 por segundo).



No arduino, ou em outro circuito que trabalhe com pwn, para diminuir a velocidade do motor, é reduzido a largura dos pulsos, mantendo-o menos tempo ligado, conforme a figura a seguir. O ciclo ativo é de 30% por que o tempo ativo corresponde a 30% do período da onda. O resultado prático disso é que a média da tensão “recebida” pelo motor fica em 1,8v.



Já na próxima figura o ciclo ativo é de 80% e o motor irá girar mais rápido que no exemplo anterior porque permanece mais tempo “ligado” no 6v.



Para entender melhor sobre PWM pesquise nos sites:

http://www.pnca.com.br/index.php?option=com_content&view=article&id=67:pwm&catid=42:sabre-mais&Itemid=150

<http://googolplex.com.br/arduino/entendendo-pwm-no-arduino>

http://books.google.com.br/books?id=C2U3AgAAQBAJ&pg=PA90&lpg=PA90&dq=pulsos+por+segundo+arduino+pwn&source=bl&ots=GK7s_uFwpq&sig=GKZcPXIJ8jPz86YltspN-6NeOjg&hl=pt-BR&sa=X&ei=iTmoU6nMB6i0sQSj7YHADw&ved=0CF4Q6AEwCDgK#v=onepage&q=pulsos%20por%20segundo%20arduino%20pwn&f=false

<http://www.newtoncbraga.com.br/index.php/robotica/5169-mec071a>

O 3º código - utilizando a função map para converter valores analógicos em PWN:

Este código ampliaremos nosso conhecimento sobre a programação do arduino com a função **map**:

```
int led = 6;
int potenciometro = A0;
int valor = 0;
int luminosidade = 0;

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

void loop()
{
    valor = analogRead(potenciometro);
    luminosidade = map(valor, 0, 1023, 0, 255);
    analogWrite(led, luminosidade);
}
```

Compreendendo o 3º código...

Este código aproveita a estrutura existente do anterior. Declaramos apenas uma outra variável lá no início chamada de **luminosidade**. A **setup** permanece idêntica ao código anterior. A diferença está na **loop**. Aqui usamos a função **map**.

No código anterior utilizamos um cálculo matemático dividindo a variável **valor** por quatro. Relembrando:

analogWrite(led, valor / 4);

Porque fizemos isso? Porque a variável **valor** contém a leitura do pino analógico onde está conectado o potenciômetro. E, como já vimos, uma leitura analógica tem valores entre 0 e 1023 - 1024 estados, mas o pino **pwm** trabalha entre 0 e 255 - 256 estados. Ou seja os pinos analógicos têm 4 vezes mais estados que os **pwm**. Por isso dividimos a leitura do pino analógico por 4, para equiparar os valores em relação ao pino **pwm**.

Mas desta vez usamos a função **map** que remapeia um número a partir de uma faixa, para outra.

luminosidade = map(valor, 0, 1023, 0, 255);

Aqui estamos dizendo ao arduino que o valor da variável **luminosidade** é o resultado mapeamento do que do pino analógico **A0**.

Mais sobre a função map em <http://renatoaloi.blogspot.com.br/2013/12/funcao-map-do-arduino.html>

Melhorando o 3º código...

Para compreender melhor o que faz na prática a função **map** vamos implementar uma comunicação serial para visualizar a leitura e a conversão dos valores analógicos em pwm. Digite o código, passe para o arduino e abre o **serial monitor** para ver o resultado:

```
int led = 6;
int potenciometro = A0;
int valor = 0;
int luminosidade = 0;

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

void loop()
{
    valor = analogRead(potenciometro);
    luminosidade = map(valor, 0, 1023, 0, 255);
    analogWrite(led, luminosidade);
    Serial.print("valor lido no pino analógico = ");
    Serial.println(valor);
    Serial.print("valor modificado = ");
    Serial.println(luminosidade);
    delay(500);
}
```

Movimentando um Motor

Que tipo de motores usaremos...

Com o arduino temos várias possibilidades em nossas montagens com motores. Há projetos que usam até mesmo motores de elétricos de corrente alternada. Vamos usar em nossos experimentos motores DC e servomotores pela facilidade de uso.

Os motores DC. Os motores de corrente contínua (CC) ou motores DC (Direct Current), como também são chamados, são dispositivos que operam aproveitando as forças de atração e repulsão geradas por eletroímãs e imãs permanentes. São os motores facilmente encontrados em brinquedos e nos mais diversos eletrônicos.

As velocidades de operação dos pequenos motores são elevadas e para os casos em que necessitados de mais força e de um movimento mais lento, precisamos contar com recursos para a chamada redução. Essa redução é feita com polias ou com engrenagens.



Os servo motores. Servomotores são motores de posição frequentemente usados em aeromodelos, automodelos, e outros veículos radio-controlados em escala reduzida e também são muito utilizados em automação e robótica. Por este motivo, são fáceis de serem encontrados no mercado especializado de radio-controles.



Um servomotor tipo RC consiste essencialmente em um motor de corrente contínua com um circuito de controle de posição acoplado. Os servomotores **não dão uma volta completa em seu eixo**, eles possuem uma faixa ou de 90 ou 180 graus em seu eixo. Em contraste com os motores contínuos que giram indefinidamente, o eixo dos servo motores possui a liberdade predeterminada, mas são precisos quanto a posição.

Do servomotor sai três cabos geralmente nas cores: preto, vermelho e branco ou amarelo. Os cabos preto e vermelho são para alimentação e o branco ou amarelo é o cabo de controle. Na estrutura do servo motor já está acoplada uma caixa de redução que lhe permite um bom torque.

Os motores de passo. Motores de passo é um tipo de motor elétrico inventado por Marius Lave. e usado quando algo tem que ser posicionado muito precisamente ou rotacionado em um ângulo exato. Neste tipo de motor a rotação do balancete é controlado por uma série de campos eletromagnéticos que são ativados e desativados eletronicamente.

Motores de passo não usam escovas ou comutadores e possuem um número fixo de pólos magnéticos que determinam o número de passos por revolução. Os motores de passo mais comuns

possuem de 3 a 72 passos/revolução, significando que ele leva de 3 a 72 passos para completar uma volta. Controladores avançados de motores de passo podem utilizar modulação por largura de pulso para realizarem micropassos, obtendo uma maior resolução de posição e operação mais macia, em detrimento de outras características.

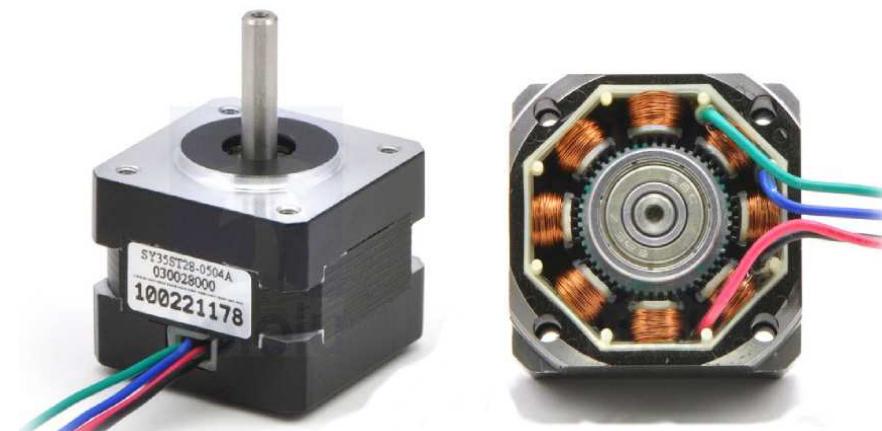
São usados em larga escala em impressoras, plotters, scanners, drivers de disquetes, discos rígidos e muitos outros aparelhos. Existem vários modelos de motores de passos disponíveis no mercado que podem ser utilizados para diversos propósitos. Poderemos utilizá-los para mover robôs, câmeras de vídeo, brinquedos ou mesmo uma cortina.

Esses tipos de motores tem no mínimo 4 fios.

Saiba mais em:

<http://www.rogercom.com/pparalela/IntroMotorPasso.htm>

<http://www.telecom.uff.br/pet/petws/downloads/tutoriais/stepmotor/stepmotor2k81119.pdf>



http://www.robomaster.com.br/2013/07/23/motor-de-passo-para-robotica/

Movimentando um Motor DC

1º montagem - controlando o motor com potenciômetro

fonte de pesquisa:

<http://arduinolivre.wordpress.com/2013/03/13/transistor-motor-dc-arduino-parte-2/>
<http://labdegaragem.com/profiles/blogs/tutorial-utilizando-arduino-e-transistor-para-controlar-motor-dc>
<http://www.comofazerascoisas.com.br/controlando-a-velocidade-de-um-motor-cc-no-arduino-com-potenciometro.html>
<http://www.funnyrobotics.com/2011/03/arduino-with-l298n-based-dual-motor.html>

Componentes necessários:

1 placa Arduino

1 cabo USB

1 protoboard

Fios diversos - jumpers

1 resistor de 1 KΩ (1000 ohms)  (marrom, preto, vermelho)



1 potenciômetro 10KΩ



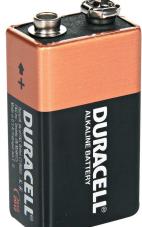
1 motor DC



1 Transistor Darlington TIP122



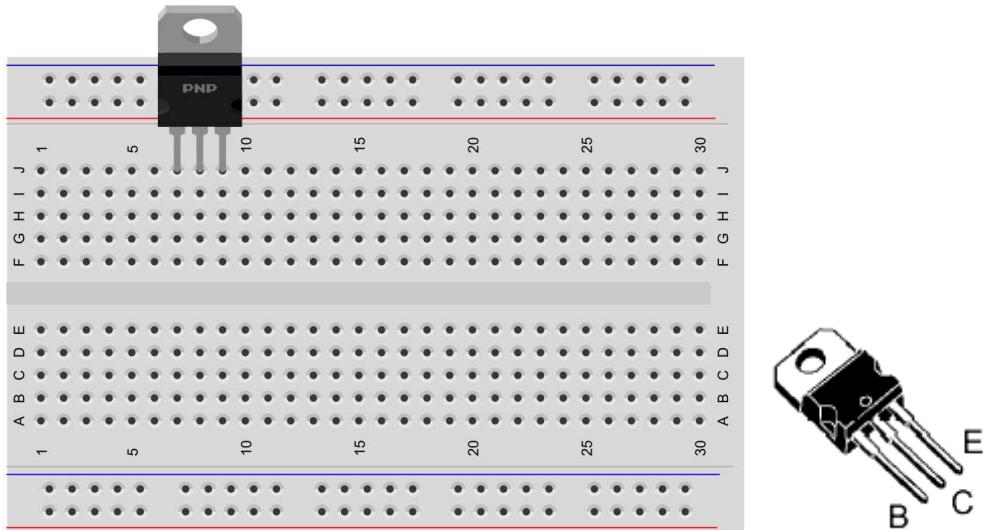
1 Clip de bateria 9v



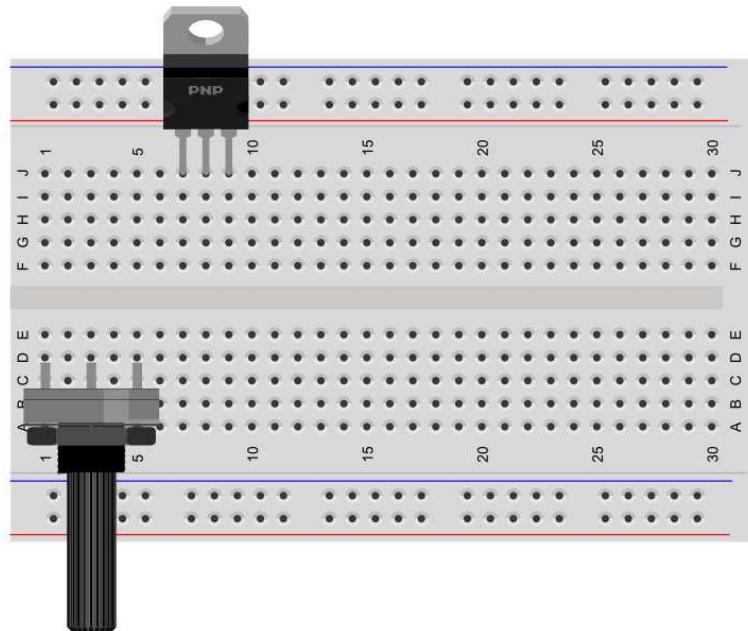
1 bateria 9v

A montagem:

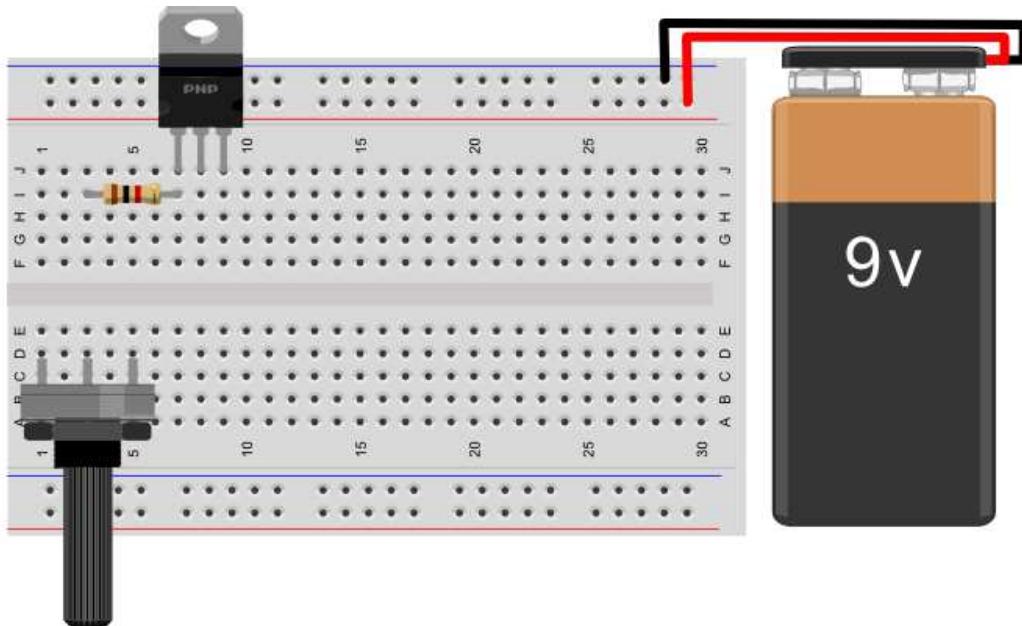
Vamos colocar o **TIP122** na linha **J** nas colunas **7, 8 e 9**. A parte mais “saliente” do transistor deve ficar para frente. De maneira que a base fique na coluna 7, o coletor na coluna 8 e o emissor na coluna 9. Se tiver dúvidas releia a apostila anterior sobre eletrônica básica ou consulte o site: <http://www.electronica-pt.com/componentes-eletronicos/transistor-tipos>. Veja a datasheet do transistor TIP122 em <http://www.adafruit.com/datasheets/TIP120.pdf>.



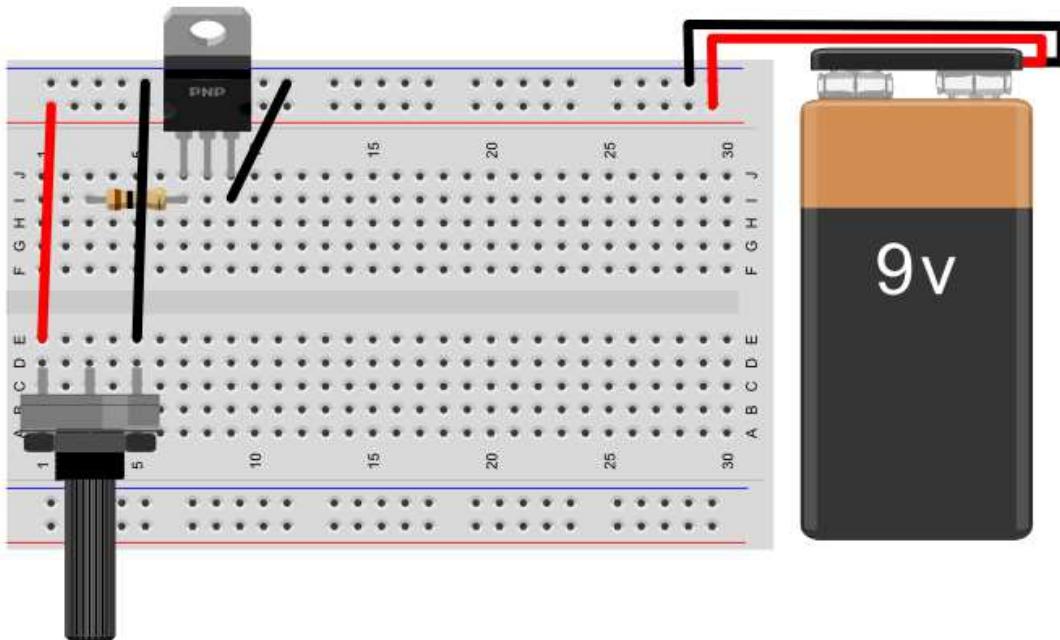
O potenciômetro colocamos na linha **B** nas colunas **1, 3 e 5**.



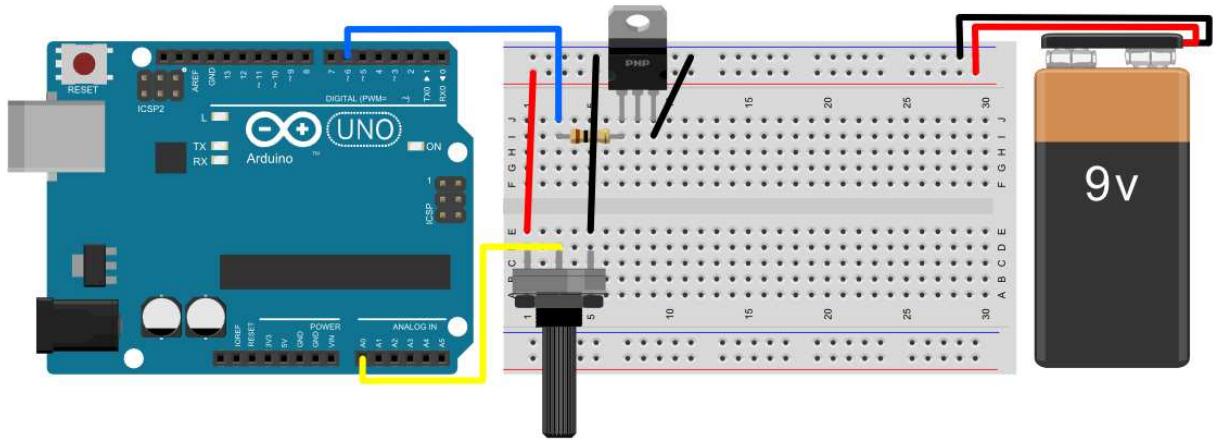
Colocamos um **resistor** com seus terminais na coluna **3** e **7** na linha **I** e alimentamos a protoboard com a bateria de 9v



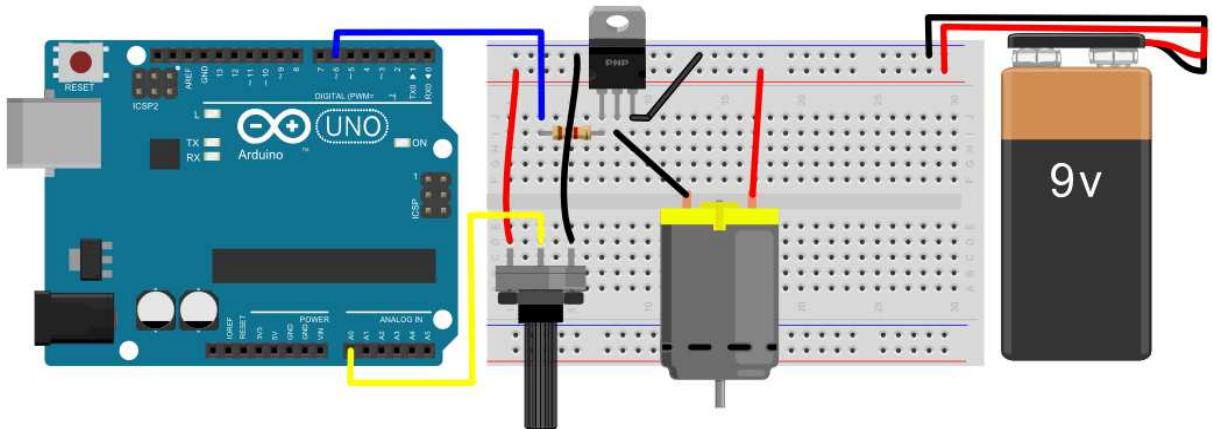
Alimentamos o pino direito(coluna 1) e o pino esquerdo (coluna 5) do potenciômetro, com o positivo e o negativo da bateria. O pino do meio, o da coluna 3, deve ficar livre. Alimentamos também a coluna 9 do TIP122 com o negativo vindo da bateria de 9v.



Conectamos a coluna 7, o pino base do TIP122, no pino 6 do arduino. E o terminal do meio do potenciômetro no pino analógico **A0** do arduino.



Por fim conectamos o motor conforme a figura:



Compreendendo a eletrônica

Já estudamos anteriormente que o motor não pode ser ligado diretamente no arduino porque demanda uma corrente muito maior que a porta do arduino pode fornecer. Relembrando: as portas do arduino individualmente suportam no máximo de 40mA e coletivamente no máximo 200mA. Dependendo do motor dc teremos veremos que demanda acima de 600mA até mais do que 1A.

Já estudamos também que para solucionar tal impasse usamos o circuito de proteção. No caso desta montagem o circuito de proteção é formado pelo **resistor** e pelo **TIP122**. O TIP122 funciona como uma chave e impede que o arduino seja danificado pelo motor. Só que ele pode demandar uma corrente de até 100mA e é por isso que se usa o resistor que limita a corrente que chega no TIP 122 mas não impede seu funcionamento.

Pesquise o datasheet para saber mais sobre esse componente =>
<http://www.adafruit.com/datasheets/TIP120.pdf>

O código

Aqui vem algo interessante no arduino. É possível utilizar o mesmo código para diversos tipo de montagem eletrônicas. Neste caso, podemos simplesmente repetir o código anterior:

Podemos melhorar o código renomeando as variáveis, já que não estamos trabalhando com um led. No lugar da variável **luminosidade** poderíamos colocar **velocidade**. Mas o efeito é o mesmo!!!

```

int led = 6;
int potenciometro = A0;
int valor = 0;
int luminosidade = 0;

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

void loop()
{
    valor = analogRead(potenciometro);
    luminosidade = map(valor, 0, 1023, 0, 255);
    analogWrite(led, luminosidade);
}

```

2º montagem - controlando o motor com botões push button

fonte de pesquisa:

<http://robotfeliciano.blogspot.com.br/2012/03/botoes-e-resistores-pull-down-e-pull-up.html>

<http://googolplex.com.br/arduino/resistores-pull-uppull-down>

<http://labdegaragem.com/profiles/blogs/tutorial-utilizando-arduino-e-transistor-para-controlar-motor-dc>

Componentes necessários:

1 placa Arduino

1 cabo USB

1 protoboard

Fios diversos - jumpers

1 resistor de 1 KΩ (1000 ohms)  (marrom, preto, vermelho)



2 Chaves toque | push button | interruptor momentâneo | botão de pressão | chave tactil (=>nomes correlatos)



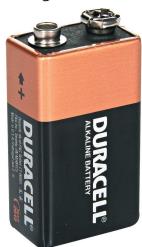
1 motor DC



1 Transistor Darlington TIP122



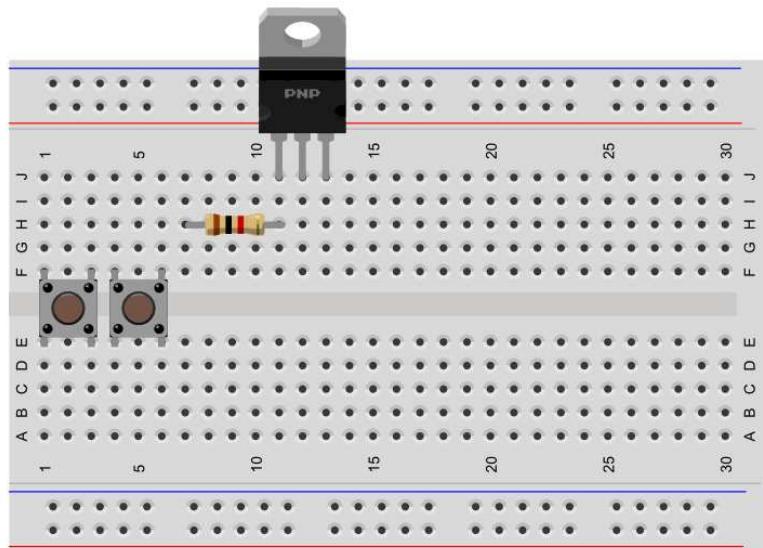
1 Clip de bateria 9v



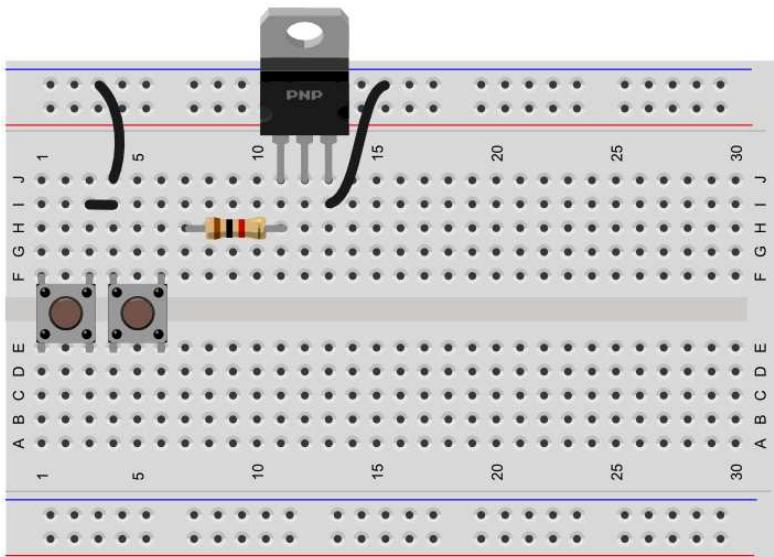
1 bateria 9v

A montagem:

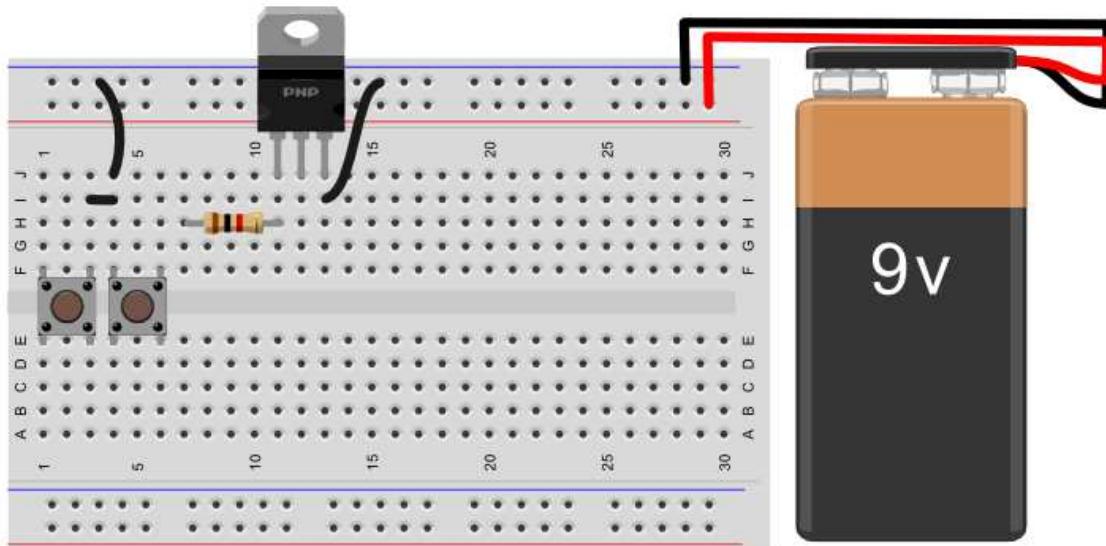
Vamos colocar os botões na linha **F** nas colunas. Um botão com seus terminais nas colunas **1** e **3** e o outro nas colunas **4** e **6**. O resistor alocamos nas colunas **6** e **11** da linha **H**. O TIP122 na linha **J** com seus terminais nas colunas **11**, **12** e **13**. Lembrando que ele deve ficar com sua parte mais “saliente” para frente e a parte mais plana para trás. Observe a figura:



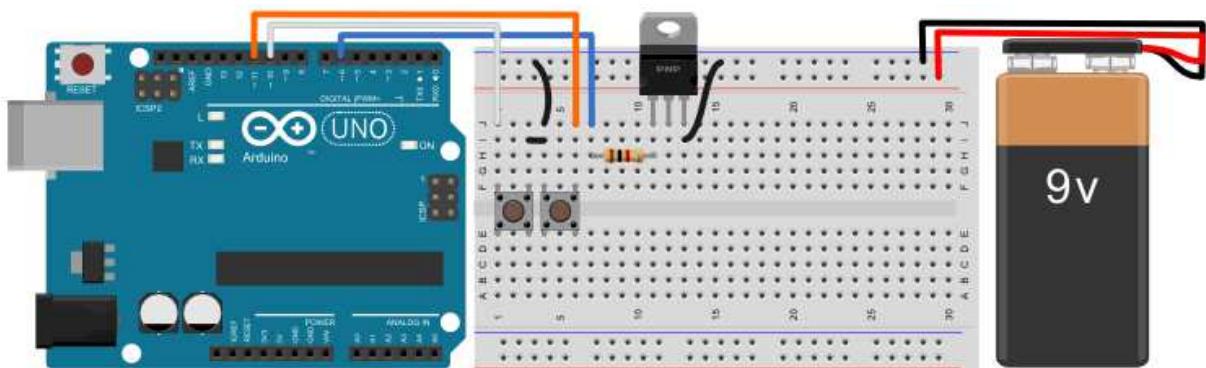
Ligamos as colunas **3** e **4** na linha azul da protoboard que será alimentada pelo **negativo**. Da mesma forma ligamos a coluna **13** na linha azul. Veja a figura:



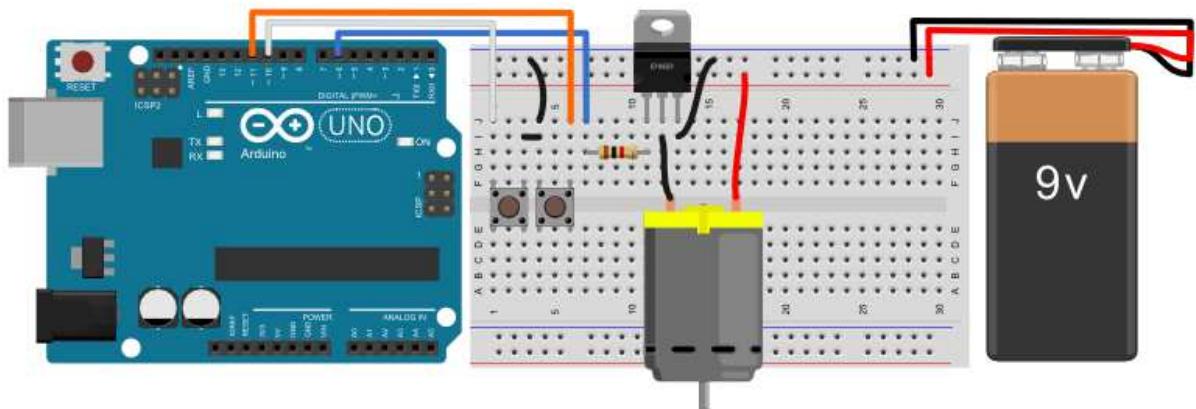
Alimentamos a protoboard com a bateria de 9v conforme demonstra a figura.



Conectamos o pino digital **11** do arduino na coluna **1** da protoboard. O pino **10** do arduino na coluna **6** da protoboard. O pino digital **6** do arduino na coluna **7** da protoboard. Veja a figura:



Por fim conectamos o motor. Um terminal na coluna **12** da protoboard e o outro terminal no positivo da bateria.



O código

Digite o seguinte código:

```

int led = 6;
int botao1 = 10;
int botao2 = 11;
byte valor1 = 0;
byte valor2 = 0;

void setup()
{
    pinMode (led, OUTPUT);
    pinMode(botao1, INPUT);
    pinMode(botao2, INPUT);
    digitalWrite(botao1, HIGH);
    digitalWrite(botao2, HIGH);
}

void loop()
{
    valor1 = digitalRead(botao1);
    valor2 = digitalRead(botao2);

    if (valor1 == HIGH && valor2 == LOW)
    {
        analogWrite(led, 255);
        delay(2000);
    } else {analogWrite(led, 0);}
}

```

```

if (valor2 == HIGH && valor1 == LOW)
{
    analogWrite(led, 100);
    delay(2000);
} else {analogWrite(led, 0);}
}

```

Compreendendo o código...

Começamos por declarar três variáveis do tipo **int**, ou seja, do tipo inteiras e outras duas variáveis do tipo **byte**. Utilizamos o tipo byte porque só serão armazenados dois tipos de valor: o 1 quando o botão estiver pressionado ou o 0 quando o botão estiver solto, desligado.

```

int led = 6;           ← Variável criada para armazenar o pino do led
int botao1 = 10;        ← Variável criada para armazenar o pino de um botão
int botao2 = 11;        ← Variável criada para armazenar o pino do outro botão
byte valor1 = 0;        ← Variável criada para armazenar o estado do botão 1 se ligado ou desligado (HIGH OU LOW, 1 OU 0)
byte valor2 = 0;        ← Variável criada para armazenar o estado do botão 2 se ligado ou desligado (HIGH OU LOW, 1 OU 0)

```

Dentro do **setup** fizemos 5 configurações. Determinamos que o pino do led (o pino **6**) será de saída, transmitirá energia. Determinamos que os pinos ligados aos botões serão de entrada, receberão energia. E ativa o resistor interno de elevação para os pinos dos botões. Explicaremos logo a seguir sobre pull-up e pull-down.

```

pinMode (led, OUTPUT);   ← Determina que o pino do led será de saída de dados
pinMode(botao1, INPUT);  ← Determina que o pino do botão 1 será de entrada de dados
pinMode(botao2, INPUT);  ← Determina que o pino do botão 2 será de entrada de dados
digitalWrite(botao1, HIGH); ← Ativa o pull-up para o pino do botão 1
digitalWrite(botao2, HIGH); ← Ativa o pull-up para o pino do botão 2

```

Iniciamos dentro da função **loop** fazendo a leitura e o armazenamento do estado dos botões, ou seja, se estão pressionados ou não.

```

valor1 = digitalRead(botao1); ← Armazena na variável valor1 a leitura do pino onde está o botão 1.
valor2 = digitalRead(botao2); ← Armazena na variável valor1 a leitura do pino onde está o botão 2.

```

Agora nós temos uma condicional. Embora já tenhamos abordado essa função, destacamos desta vez o uso de um operador booleano e outro de comparação. Operadores de comparação, com já sugere o nome, é usado para testar a comparação de dois valores. Já os operadores booleanos são usados para relacionar dados de uma pesquisa.

| Operadores de booleanos |
|------------------------------|
| && (e) (ou) ! (não) |

| Operadores de comparação |
|---|
| == (igual a) != (diferente de) < (menor que) > (maior que) <= (menor ou igual a) >= (maior ou igual a) |

A forma mais prática de entender o funcionamento destes operadores é usando-os nas programações. Por isso vamos voltar ao estudo desta programação e mais adiante a ilustraremos com outros exemplos.

Na prática estamos determinando como parâmetro para essa condicional que o botão 1 tem que estar pressionado, ligado, e o botão 2 tem que estar desligado



Sendo verdadeira essa condição é executado o que está dentro do corpo dela:

analogWrite(led, 255); ← Escreve o valor 255, o máximo, no pino 6 (que é o valor da variável led)
delay(2000); ← Espera o tempo de 2 segundos antes de continuar a programação

Não sendo a condição verdeira é executada a função **else**, que escreve o valor de **0** mantendo o led apagado.

else {analogWrite(led, 0);}

Na sequencia da programação escrevemos novamente um condição que faz exatamente o inverso do que já vimos. Ela testa se o 2º botão está pressionado, pesquisa se o 1º botão não está pressionado, e, se estas duas condições forem verdadeiras, acende o led com baixa potência. Do contrário, mantém o led apagado.

Movimentando um Servo Motor

1º montagem - controlando o motor com potenciômetro

fonte de pesquisa:

<http://blog.filipeflop.com/motores-e-servos/micro-servo-motor-9g-sq90-com-arduino-uno.html>
http://www.seucurso.com.br/index.php?option=com_content&view=article&id=218:controlando-um-servo-motor-com-arduino&catid=901:arduino&Itemid=65
http://www.ajudino.com/2013/05/4-utilizando-servo-motor-no-arduino_11.html
<http://sirleech.wordpress.com/2010/06/05/hello-coke-bot-arduino-servo-test/>
<http://electronicpiece.blogspot.com.br/2012/03/arduino-knob-servo-e-potenciometro.html>

Componentes necessários:

1 placa Arduino

1 cabo USB

1 protoboard

Fios diversos - jumpers



1 potenciômetro 10KΩ



1 Micro Servo Motor

datasheet: <http://alturn-usa.com/products/PDF/AAS-309BB.pdf>



1 suporte para 4 pilhas AA

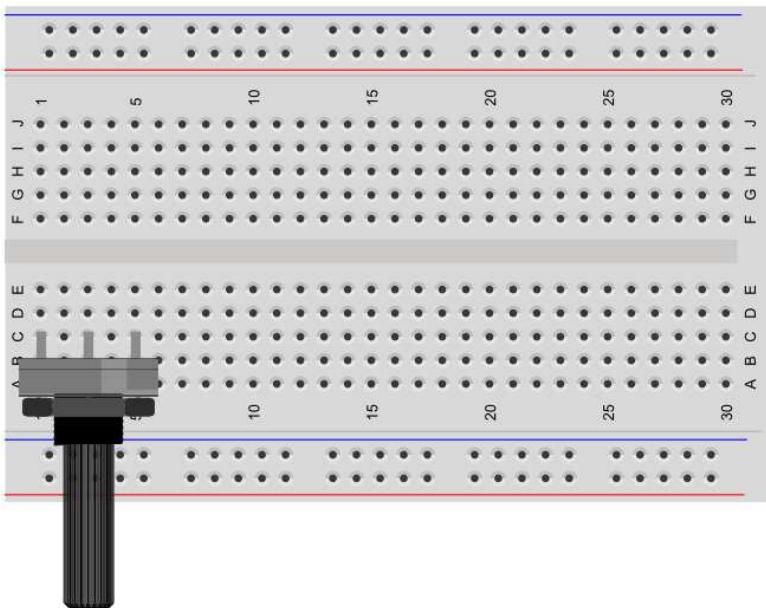


4 pilhas AA (1,5v)

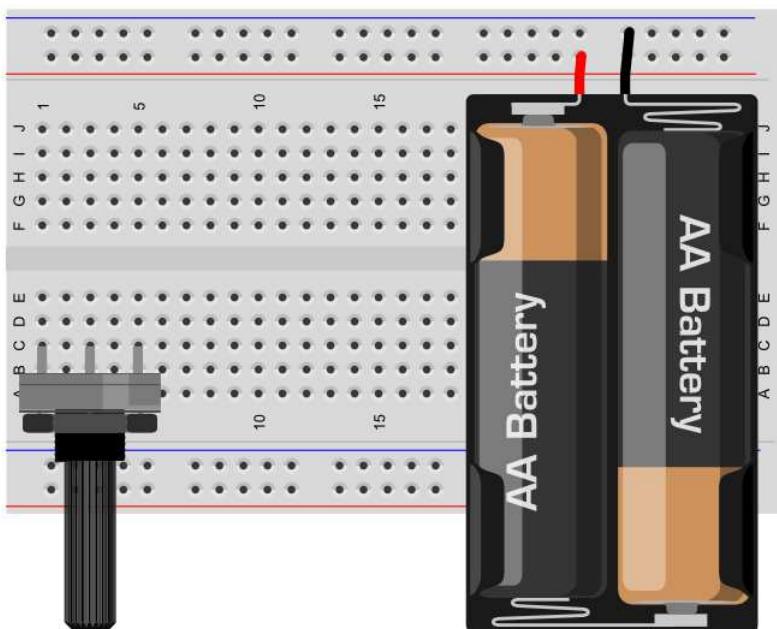
A montagem:

A montagem deste projeto é bem simples. Começamos colocando o terminais do potenciômetro na linha **A**, nas

colunas 1, 2 e 3 como mostra a figura.



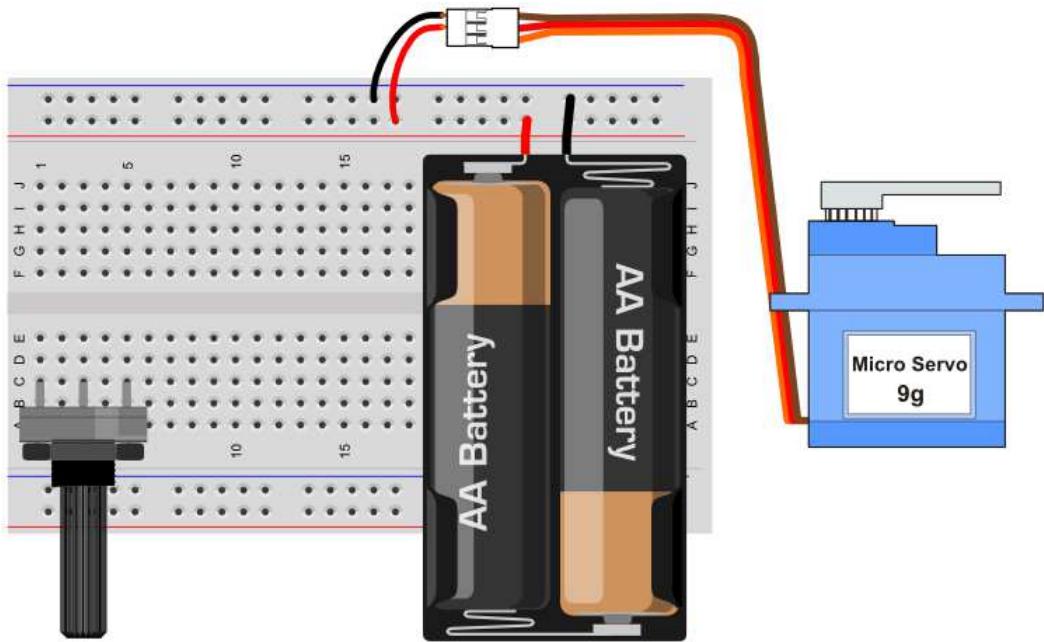
Depois alimentamos as linhas azul e vermelha da protoboard com, respectivamente, o negativo e positivo das pilhas.



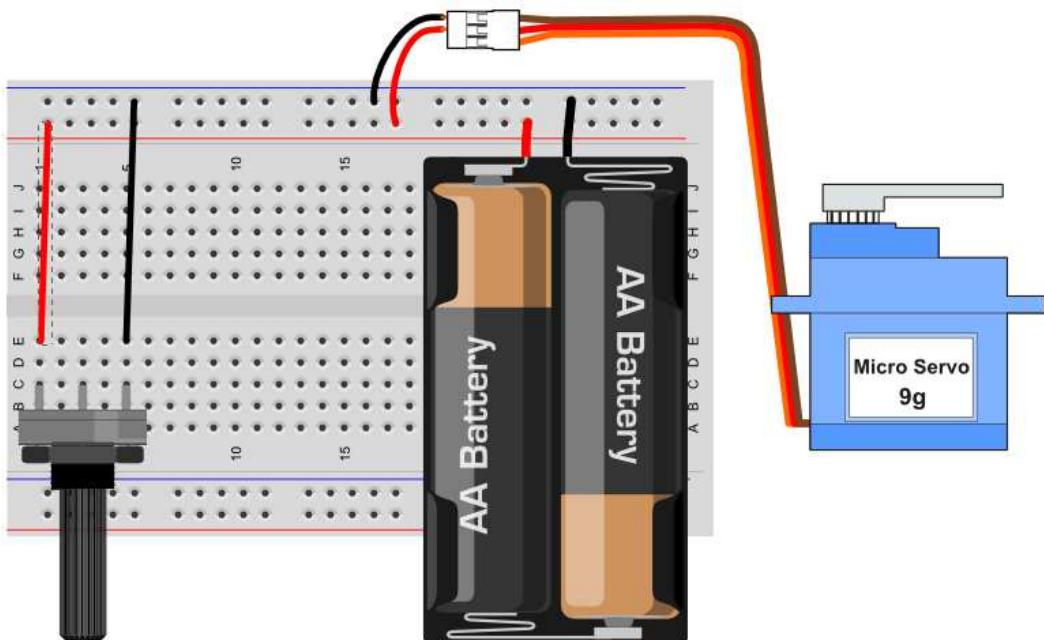
Vamos alimentar os fios do **Servomotor**. O fio vermelho vai conectado ao positivo da protoboard e o fio marrom no negativo. Observe a figura que ilustra a conexão dos fios do Servomotor:



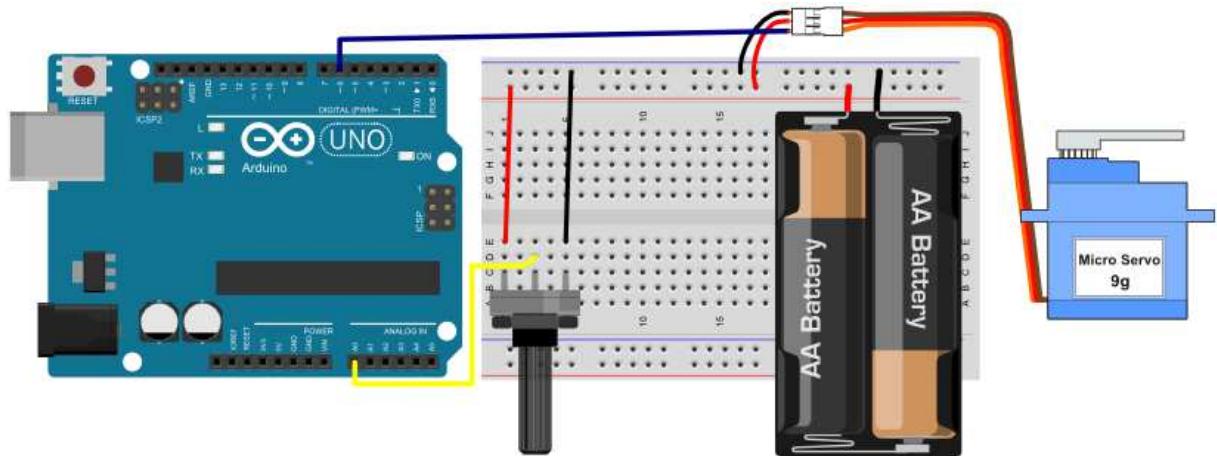
A montagem na protoboard ficará desta forma:



Conectamos a coluna 1 da linha no positivo e a coluna 5 no negativo da protoboard:

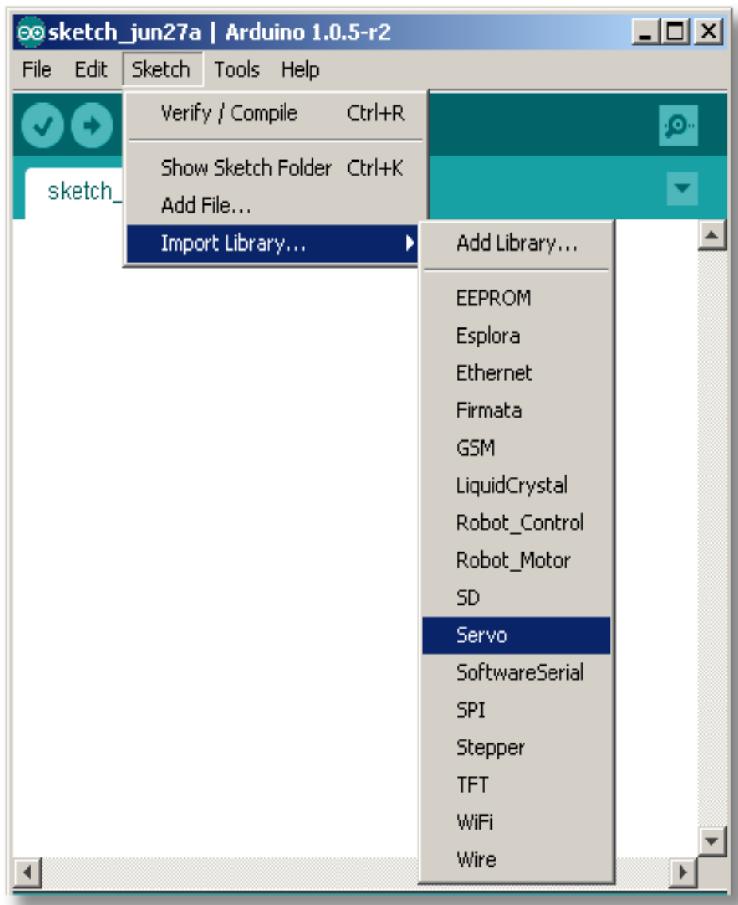


Conectamos o pino central do potenciômetro no arduino: ligamos um fio na coluna 3 da protoboard para o pino analógico **A0** do arduino. E conectamos o fio do sinal do **Servomotor** ao pino digital **6** do arduino.



O código:

Este código trará um novidade: a utilização da biblioteca. Vamos começar o nosso código inserindo a biblioteca para podermos controlar o servomotor. O jeito mais prático de inserir sem erros uma biblioteca é através do menu Sketch. Clique no menu **Sketch**, depois em **Import Library...** e depois em **Servo**. Conforme demonstra a figura:



Vai ser inserido o seguinte código: **#include <Servo.h>**

Agora vamos escrever as outras linhas do código:

Servo testaservo;

int potenciometro;

```

void setup()
{
    testaservo.attach(6);
}

void loop()
{
    potenciometro = analogRead(A0);
    potenciometro = map(potenciometro, 0, 1023, 0, 179);
    testaservo.write(potenciometro);
    delay(5);
}

```

Entendendo o código:

O ambiente de programação do Arduino pode ser estendido através da utilização de bibliotecas, assim como a maioria das plataformas de programação. Bibliotecas podem fornecer funcionalidades extras para, por exemplo, trabalhar com hardware específico (como é o caso do servomotor) ou manipulação de dados. Uma série de bibliotecas já vêm instalados com o IDE, mas também podemos fazer download de outras ou criar a nossa própria biblioteca. Para aprofundar sobre o assunto acesse este site:

<http://renatoaloi.blogspot.com.br/2012/11/criando-bibliotecas-no-arduino.html>. Ou esse em inglês:
<http://arduino.cc/en/Reference/Libraries>.

#include <Servo.h>

Essa linha de comando está dizendo para o compilador importar a biblioteca **Servo**. Com isso poderemos usar as classes e comandos disponíveis nesta biblioteca. Por se tratar de uma biblioteca baseada em classes, precisamos declarar uma variável para criar uma instância da classe, ou seja, criar um **objeto**. Esses conceitos não ficarão muito claros neste momento para quem não tem uma familiaridade com programação.

Servo testaservo;

Aqui criamos o objeto **testaservo**. Na prática definimos um nome para o **servomotor**. Vamos usá-lo para chamar os métodos durante a programação.

int potenciometro;

Definimos uma variável que vai armazenar o valor da porta analógica conectada ao potenciômetro.

```

void setup()
{
    testaservo.attach(6);
}

```

Dentro da **setup** chamamos o método **attach**. É o primeiro método da biblioteca **Servo** que devemos utilizar. Este método deve ser utilizado na função **setup()** para associar o

servomotor a um pino da placa Arduino. No caso de nossa montagem é o pino **6** que fica entre os parenteses. Repare que devemos usar o objeto (**testaservo**) que declaramos anteriormente para invocar o método.

potenciometro = analogRead(A0);

Setamos como valor para a variável

potenciometro a leitura da porta analógica **A0**. Onde está conectado o pino do potenciômetro.

potenciometro = map(potenciometro, 0, 1023, 0, 179);

Usamos a

função **map** e mapeamos o valor do potenciômetro que é de 0 até 1023 redimensionando-os para os valores aceitáveis pelo **Servomotor** que tem um giro máximo de 180º, portanto de 0 até 179.

testaservo.write(potenciometro);

Usamos depois da configuração o método **write**

da biblioteca servomotor para escrever o valor do posicionamento do servomotor.

delay(5);

Espera 5 milisegundos antes de continuar a programação.

Sensor ultrasônico HC-SR04

fonte de pesquisa:

<http://www.arduinoecia.com.br/2014/04/sensor-de-estacionamento-re-com-arduino.html>
<http://www.instructables.com/id/Simple-Arduino-and-HC-SR04-Example/?lang=pt>
<http://ferpinheiro.wordpress.com/2011/04/29/meu-primeiro-projeto-arduino-sensor-ultrasonico-hc-sr04/>
<http://blog.filipeflop.com/sensores/sensor-ultrassonico-hc-sr04.html>
<http://www.mecatronicaatual.com.br/educacao/1598-sensores-ultra-snicos>
<http://www.sabereletronica.com.br/artigos/1753-sensores-ultra-snicos>

Componentes necessários:

1 placa Arduino

1 cabo USB

1 protoboard

Fios diversos - jumpers

1 resistor 220Ω 

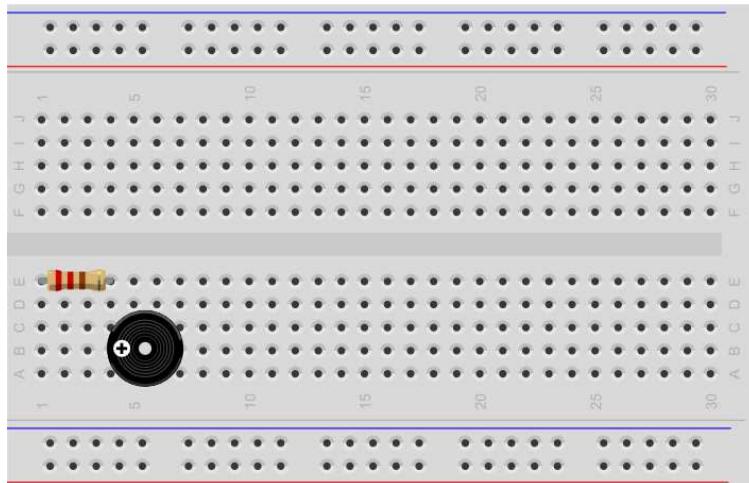
1 buzzer 



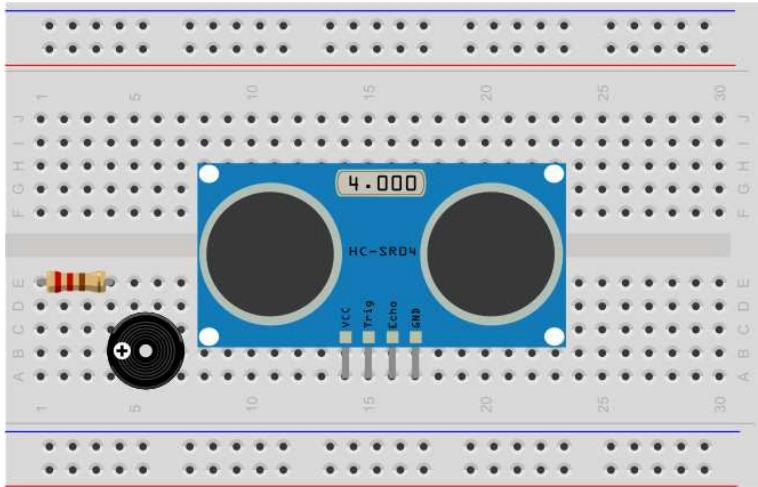
1 sensor ultrasônico HC-SR04

A montagem:

Coloque o **resistor** na linha **F** com seus terminais na coluna **1** e **4**. Nesta mesma linha coloque o **buzzer** nas colunas **4** e **7**. Não esqueça que o terminal positivo do **buzzer** vai conectado ao resistor, ou seja, na coluna **4**.

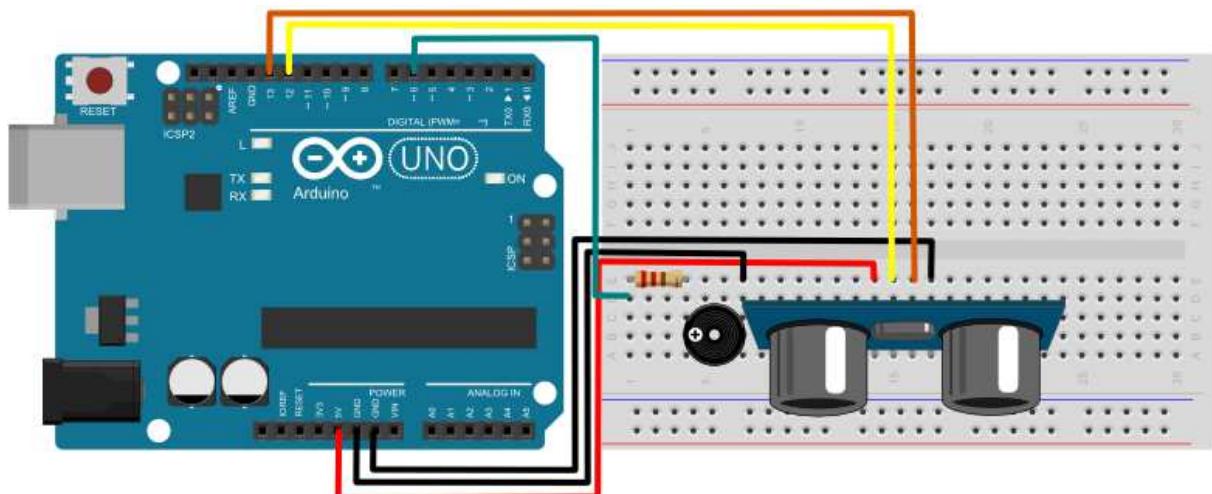


Conecte os terminais do **HC-SR04** na linha **A** nos pinos **14, 15, 16 e 17**. Veja a figura:



Por fim, vamos fazer a conexão com os pinos do arduino. Conectar um fio do pino **6** do arduino ao conector do resistor na coluna **1**. Conectar um fio do **GND** do arduino ao pino do **buzzer** na coluna **7**. Conectar o **5v** do pino de alimentação do arduino ao terminal **VCC** do sensor ultrasônico na coluna **14**.

Conectar o pino **12** do arduino ao terminal **TRIG** do sensor ultrasônico (coluna **15**). Conectar o pino **13** do arduino ao terminal **ECHO** do sensor ultrasônico (coluna **16**). Conectar o **GND** do arduino ao terminal **GND** do sensor ultrasônico (coluna **17**). Veja a demonstração da figura:



Como funciona o sensor ultrasônico:

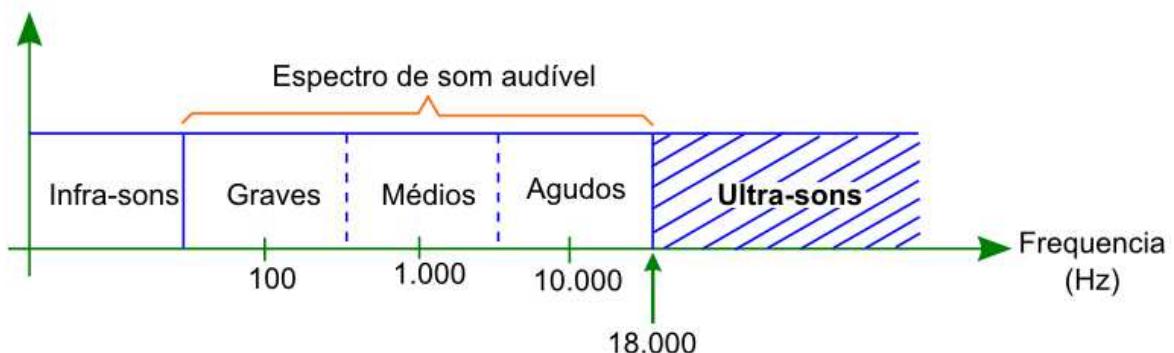
Antes de escrevermos o código para essa montagem convém entender como é o funcionamento básico deste sensor. O sensor ultrasônico HC-SR04 vai detectar um objeto além de poder determinar a distância que ele se encontra. O sensor é formado por um emissor e um receptor de ultrasons e se baseia no sonar dos morcegos.

Mas o que são ultrasons? Ultrasom é um tipo de onda sonora que não conseguimos escutar. Mas o que são ondas sonoras? De maneira geral as ondas sonoras se propagam a uma velocidade de 331,5 metros por segundo no ar em condições normais de temperatura e pressão. Com a elevação da temperatura, essa velocidade aumenta da ordem de 0,61 metros por segundo para cada grau Celsius. Atingindo nossos ouvidos, essas ondas podem pressioná-los, dando-nos a sensação sonora, se estiverem, entretanto, numa faixa bem definida de freqüências.

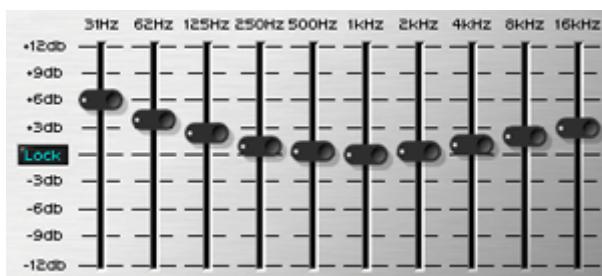
Efetivamente, partindo do zero, só podemos começar a ouvir alguma coisa quando o número de vibrações ultrapassar 16 por segundo ou **16 Hz**. À medida que as vibrações vão se tornando mais rápidas, vamos tendo a sensação de sons cada vez mais **agudos** até que em torno de 18.000 Hz (dependendo da pessoa), deixamos de ter qualquer sensação auditiva. Acima das 18 000 vibrações por segundo ou **18 kHz** é que estão os ultra-sons

conforme vemos no espectro da figura abaixo:

Espectro de sons

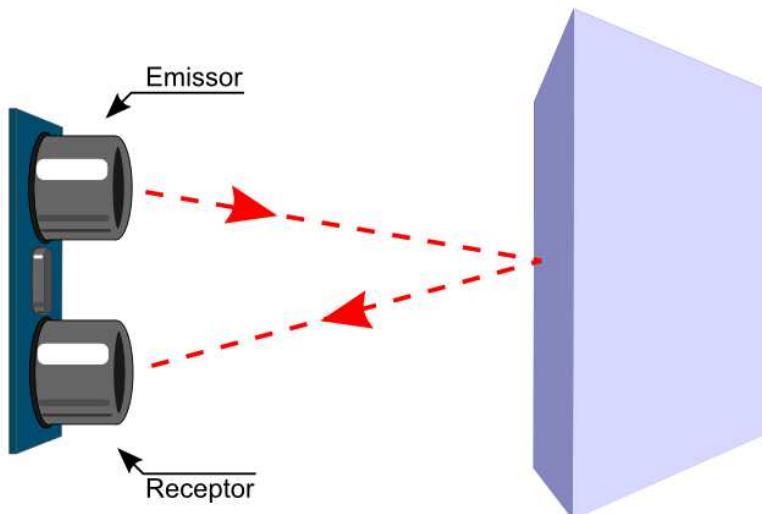


Num equalizador de um aparelho de som podemos alterar a intensidade das frequências sonoras:



Acima dos **18KHz** as vibrações existem, mas não podemos ouvi-las, e suas propriedades são as mesmas dos sons comuns. É claro que existem animais que podem ouvir bem acima dos 18 kHz, como os morcegos, os golfinhos e até mesmo seu cachorro. Alguns morcegos podem ouvir ultrasons de freqüências que ultrapassam os 200 kHz.

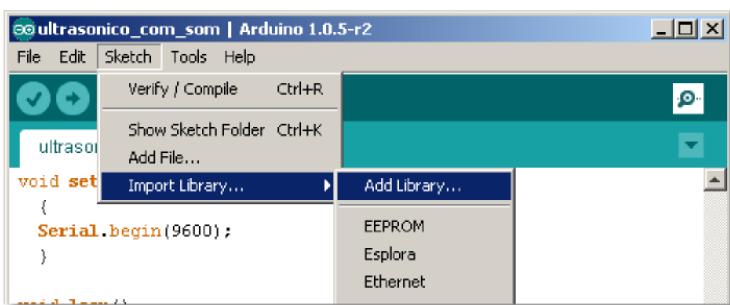
O funcionamento do sensor ultrasônico é assim: o sensor emite um sinal ultrasônico que reflete em um objeto e retorna ao sensor (eco). O sinal de retorno é captado, permitindo-se deduzir a distância do objeto ao sensor tomando o tempo de trânsito do sinal, ou seja, o tempo que o sinal leva para sair do emissor, refletir no objeto e ser “escutado” pelo receptor.



O 1º código:

Para usar esse sensor ultrasônico com facilidade usaremos uma biblioteca. Por padrão não vem instalada na IDE do Arduino nenhuma biblioteca para sensor ultrasônico, mas é fácil de instalar. Há várias bibliotecas que podemos usar com o sensor ultrasônico HC-SR04. A que vamos usar pode ser baixada aqui:
<http://www.4shared.com/zip/dJy8gpEOba/HCSR04Ultrasonic.html>.

Depois de baixar, crie uma nova pasta em seu computador e move o arquivo HCSR04Ultrasonic.zip dentro dela. Abra a IDE, clique no menu **Sketch**, depois em **Import Library** e em **Add Library**. Encontre o local que você salvou o arquivo e clique em **Open**. Reinicie a IDE do Arduino.



Reiniciado a IDE do arduino clique novamente em **Sketch**, **Import Library**. Deverá aparecer neste menu a biblioteca **HCSR04Ultrasonic**. Clique nela. Será inserida a linha de código:

```
#include <Ultrasonic.h>
```

Complete a programação digitando o código abaixo:

```
#define PINODOTRIG 12
#define PINODEOCHO 13

Ultrasonic ultrasonico(PINODOTRIG, PINODEOCHO);

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

void loop()
{
    float centimetro, polegada;
    long microsegundos = ultrasonico.timing();

    centimetro = ultrasonico.convert(microsegundos, Ultrasonic::CM);
    polegada = ultrasonico.convert(microsegundos, Ultrasonic::IN);
    Serial.print("MS: ");
    Serial.print(microsegundos);
```

```

Serial.print(", CM: ");
Serial.print(centimetro);
Serial.print(", IN: ");
Serial.println(polegada);
delay(1000);
}

```

Compreendendo o 1º código:

Como já foi explicado começamos o código determinando que o compilador da IDE deve utilizar a biblioteca Ultrasonic. Vamos utilizar algumas funções e métodos desta biblioteca.

#include <Ultrasonic.h>

Depois construímos duas variáveis através da função **#define** que vão armazenar os pinos conectados aos terminais Trig e Echo do sensor. No caso o pino digital **12** para o trig e o pino digital **13** para o echo. Poderíamos ter declarado essa variável de uma outra forma. Por exemplo com a função **int**. Perceba que na função **define** não vai o ponto e vírgula no final da linha de programação.

#define PINODOTRIG 12
#define PINODEOCHO 13

Agora estamos declarando o objeto **ultrasonico** colocando como parâmetros as variáveis PINODOTRIG e PINODEOCHO que acabamos de fazer. Veja a figura abaixo

Ultrasonic ultrasonico(PINODOTRIG, PINODEOCHO);

Dentro da função **setup** declaramos o uso da porta serial

Serial.begin(9600);

Agora criamos duas variáveis do tipo **float**. Uma com o nome de **centimetro** e outra com o nome de **polegada**. Uma variável de tipo **float** armazena valor em ponto flutuante, ou seja, um valor que possui casas decimais. Por isso não usamos a função **int**, pois ela só armazena números inteiros. Para poupar tempo e espaço declaramos as duas variáveis numa mesma linha de programação usando a vírgula.

float centimetro, polegada;

Depois criamos uma outra variável mas agora do tipo **long**. O tipo de dado Long armazena valores inteiros sinalizados de 32 bits (4 bytes) que compreendem a faixa de -2.147.483.648 a 2.147.483.647. A variável precisa ser do tipo **long** pelo tamanho de dados que ela vai trabalhar. O sensor ultrasônico trabalha com valores em milionésimos, portanto muito grandes. Não poderia ser uma variável do tipo **int** por exemplo, que trabalha na faixa entre -32.768 a 32.767.

Já criamos a variável **microsegundos** com um valor: **ultrasonico.timing()** que é a leitura que o receptor do ultrasônico está fazendo. Esta leitura é o tempo em milionésimos de segundos que o ultrasom leva para sair e voltar do sensor. O valor desta leitura é chamada pelo método **timing** da biblioteca ultrasonic. O objeto **ultrasonico** chama o **timing**.

long microsegundos = ultrasonico.timing();

Setamos na variável **centímetro** um valor. Este valor é o cálculo de conversão do tempo no formato de milionésimos de segundo em centímetros. Para isso chamamos o método **convert** no objeto **ultrasonico** e colocamos como parâmetro a variável **microsegundos** e a função **Ultrasonic::CM**.

centimetro = ultrasonico.convert(microsegundos, Ultrasonic::CM);

Setamos na variável **polegada** o valor do cálculo de conversão do tempo no formato de milionésimos de segundo em polegadas.. Para isso chamamos o método **convert** no objeto **ultrasonico** e colocamos como parâmetro a variável **microsegundos** e a função **Ultrasonic::IN**.

```
polegada = ultrasonico.convert(microsegundos, Ultrasonic::IN);
```

Mandamos para a porta serial os dados “MS: ” com a função **Serial.print**, como também mandamos a variável **microsegundos**.

```
Serial.print("MS: ");
Serial.print(microsegundos);
```

Enviamos os valores “, CM: ” e a variável **centimetro** com a função **Serial.print**.

```
Serial.print(", CM: ");
Serial.print(centimetro);
```

Enviamos os valores “, IN: ” e a variável **polegada** com a função **Serial.print**.

```
Serial.print(", IN: ");
Serial.println(polegada);
```

Mandamos aguardar 1 segundo:

```
delay(1000);
```

O 2º código:

Digite o código abaixo:

```
#include <Ultrasonic.h>

#define PINODOTRIG 12
#define PINODOECHO 13

const int buzzer = 3;
float centimetro;
int tic = 0;

Ultrasonic ultrasonico(PINODOTRIG, PINODOECHO);

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

void loop()
```

```

{
    long microsegundos = ultrasonico.timing();
    tic = 10;
    centimetro = ultrasonico.convert(microsegundos, Ultrasonic::CM);

    Serial.print("Distancia: ");
    Serial.print(centimetro);
    Serial.println(" centimetros.");
    delay(100);
    sirene();
}

void sirene()
{
    digitalWrite(buzzer, 0);
    if(centimetro <= tic)
    {
        digitalWrite(buzzer, 1);
        delay(300);
        digitalWrite(buzzer, 0);
        delay(300);
    }
}

```

Compreendendo o 2º código:

Como já foi explicado começamos o código determinando que o compilador da IDE deve utilizar a biblioteca

#include <Ultrasonic.h>

Ultrasonic.

Depois construímos duas variáveis através da função **#define**. Lembrando que elas são diferentes de outras variáveis que já criamos por não ter que colocar o **ponto-e-vírgula** no final! Essas variáveis vão armazenar os pinos conectados aos terminais Trig e Echo do sensor. O pino digital **12** para o trig e o pino digital **13** para o echo.

#define PINODOTRIG 12
#define PINODOECHO 13

Depois declaramos a constante **buzzer** que armazena o pino do onde está conectado o buzzer. Lembrando que a constante é declarada ao colocar **const** antes do tipo de uma variável. Neste caso o tipo é **int**.

const int buzzer = 6;

Declaramos a variável **centímetro** do tipo **float**. Como já foi explicado uma variável de tipo float armazena números decimais.

float centimetro;

Criamos a variável **tic** do tipo **int** com valor igual a zero. Essa variável vai ser usada como uma referência, que explicaremos depois.

int tic = 0;

Declarando o objeto **ultrasonico** colocando como parâmetros as variáveis PINODOTRIG e PINODEOCHO. Isso é feito com a classe **Ultrasonic** que vem da biblioteca Ultrasonic.

Ultrasonic ultrasonico(PINODOTRIG, PINODEOCHO);

Iniciamos a função **setup** iniciando a comunicação serial

Serial.begin(9600);

e declaramos o pino digital **6**, que é o valor armazenado na variável **buzzer**, como pino de saída de dados, **OUTPUT**.

pinMode(buzzer,OUTPUT);

Depois criamos uma outra variável mas agora do tipo **long**. O tipo de dado Long armazena valores inteiros sinalizados de 32 bits (4 bytes) que compreendem a faixa de -2.147.483.648 a 2.147.483.647. Esta variável criada precisa ser do tipo **long** pelo tamanho de dados que ela vai trabalhar. O sensor ultrasônico trabalha com valores em milionésimos, portanto muito grandes. Não poderia ser uma variável do tipo **int** por exemplo, que trabalha na faixa entre -32.768 a 32.767 (observação: miléssimos diferente de milionésimos).

Já criamos a variável **microsegundos** com um valor: **ultrasonico.timing()** que é a leitura que o receptor do ultrasônico está fazendo. Esta leitura é o tempo em milionésimos de segundos que o ultrasom leva para sair e voltar do sensor. O valor desta leitura é chamada pelo método **timing** da biblioteca ultrasonic. O objeto **ultrasonico** chama o **timing**.

long microsegundos = ultrasonico.timing();

Determinamos um valor para variável **tic**.

tic = 10;