Curso de Arduino

O que é?
Arduino é um microcontrolador (computador pequeno) utilizado para prototipagem eletrônica. Foi desenvolvido especialmente para designers e artistas plásticos, ou seja, deve ser fácil de utilizar.

Quais seus principais componentes?

- Portas digitais I/O;
- Portas digitais com PWM (Controle de largura de pulso);
- Portas analógicas;
- Processador (tipicamente 16 Mhz);
- Memória Flash (onde fica salva o programa);
- Memória Rom (onde ficam salvas as variáveis).

Dados importantes:

- Alimentação externa: de 6 – 20V (Ideal : 7 – 12 V)
- Tensão de trabalho nas portas: 5V
- Corrente nas portas: 50mA

O que são portas digitais I/O, PWM e analógicas?

Portas digitais são portas que só possuem dois estados possíveis:

- **LOW** – 0V aplicados a porta
- **HIGH** – 5V aplicados a porta

Estes estados podem ser tanto de leitura (**INPUT**) da tensão aplicada na porta como de escrita (**OUTPUT**). OBS. Como as portas digitais só admitem esses dois valores, quando a porta está no MODO **INPUT** e aplicamos uma tensão menor que 5V o arduino faz o arredondamento. Se > 2,5V então o estado é **LOW**, senão é **HIGH**.

Algumas portas digitais possuem a função PWM. Isso significa que a tensão é aplicada em pulsos como na figura ao lado. Este tipo de sinal pode ser usado para controlar a potência (energia por tempo) enviada para a porta e assim podemos controlar a velocidade de rotação de um motor, a intensidade de um LED etc. Este controle de potência simula uma porta analógica, ou seja, quando usamos a porta PWM para enviar um sinal com 50% do tempo HIGH e 50% LOW, isso equivale a aplicar uma tensão igual a 50% do valor da tensão daquela porta, ou seja, 50% 5V = 2,5V.

*Como o número de portas **PWM** é limitado, é uma boa prática de desenvolvimento, só utilizá-las com componentes que possam exigir o seu uso.*

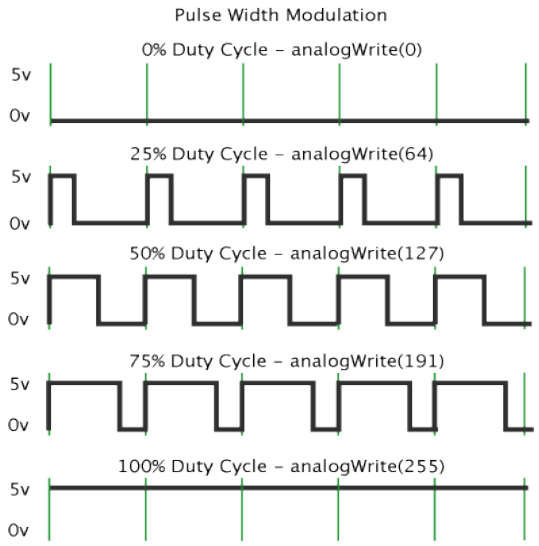


Fig 1. Tensão aplicada nas portas PWM em função do tempo.

As **portas analógicas** são aquelas que permitem a LEITURA (**INPUT**) de qualquer valor de tensão entre 0 e 5V. Cuidado para não ultrapassar o limite de 5V, isso pode danificar a porta ou até mesmo o arduino.

O que eu preciso para começar a utilizar?

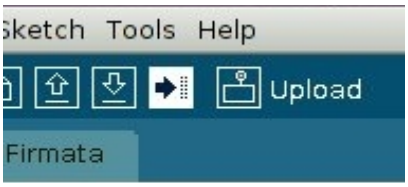
- Ter uma placa arduino;
- Instalar o software de programação (www.arduino.cc).

Um primeiro exemplo:

Objetivo: Fazer o LED embutido no arduino piscar

```
void setup() {
      pinMode(13, OUTPUT);
}
void loop() {
      digitalWrite(13, HIGH);
      delay(1000);
      digitalWrite(13, LOW);
      delay(1000);
}
```

Para passar o programa para o arduino (fazer UPLOAD) clique neste botão:



ATENÇÃO: Ao fazer o upload de um programa para o arduino as portas 0 e 1 não podem estar ligadas a nenhum dispositivo.

Entenda o que você fez:
→ **void setup(){ }** é a função de inicialização do arduino. Ela informa para o arduino os parâmetros iniciais no qual ele vai trabalhar, isso é equivalente a ligar o computador.

→ **void loop(){ }** é a função de ciclo do arduino. Após inicializar o arduino ele entra na função loop e executa instrução a instrução até o final. Ao chegar ao final da função ela retorna para o início da mesma e repete tudo de novo, indefinidamente.

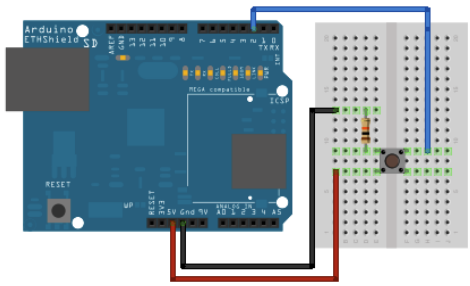
→ **pinMode(13, OUTPUT)**; esta instrução informa que a porta digital número 13 será utilizado como saída (**OUTPUT**). Se eu quiser que esta porta fosse uma porta de entrada eu usaria a opção **INPUT**. Atenção – apenas as portas digitais devem ser iniciadas com o comando pinMode(), as portas analógicas por só terem o modo INPUT não precisam ser iniciadas.

→ **digitalWrite(13, HIGH)**; esta instrução "liga" a tensão na porta **13**. Lembrando que ligar a tensão significa aplicar uma tensão de 5V. Para desligar a tensão, devemos usar a opção **LOW** no lugar do **HIGH**.

→ **delay(1000);** para o arduino por **1000 milisegundos** (1 segundo). Este valor pode ser qualquer valor inteiro.

Como eu faço para ler as portas digitais?

Objetivo: Utilizar um botão para acender ou apagar um LED.
Monte o seguinte circuito:



Use um resistor de 1 KOhms

Atenção: Em eletrônica há uma convenção de cores. Os fios vermelhos são sempre os fios ligados ao positivo (5V) e os fios pretos (em alguns casos marrom) são os fios ligados ao negativo (Ground - GND).

E escreva o seguinte código:

```
#define BOTAO 2
#define LED 13
void setup() {
      pinMode(LED, OUTPUT);
      pinMode(BOTAO, INPUT);
}
void loop() {
      int estado;
      estado = digitalRead(BOTAO);
      if ( estado == HIGH) {
            digitalWrite(LED, LOW);
      }
      else {
            digitalWrite(LED, HIGH);
      }
}
```

Entenda o que você fez:
Por motivos óbvios não vou falar sobre as funções já comentadas, apenas sobre as novas funções.

→ **#define BOTAO 2** esta definição é para nos poupar de ter que lembrar em que porta digital está ligada o BOTAO. Toda vez que eu escrever BOTAO em uma função, o arduino vai substituir a palavra pelo 2. Na outra definição **#define LED 13** estamos fazendo a mesma coisa, apenas atribuindo a palavra LED para porta 13.

→ **pinMode(BOTAO, INPUT);** agora estamos definindo que a porta onde está ligada o BOTAO é uma porta de leitura (INPUT). Isso é equivalente a escrever: pinMode(2, INPUT);

→ **int estado;** este comando cria uma variável (que será armazenada na memória RAM do arduino) com o nome **estado**. O termo **int** informa ao arduino que essa variável é um número inteiro, ou seja, sem vírgula. Esse inteiro pode ser positivo ou negativo e varia entre -32.768 to 32.767. Existem outros tipos de variáveis como: **float** (números fracionados), **double** (números fracionados com maior precisão), **string** (textos curtos), **char** (letras e números).

→ **estado = digitalRead(BOTAO);** estamos dizendo que o valor da variável estado é o mesmo do

resultado da função **digitalRead(BOTAO)**. A função **digitalRead(BOTAO)** lê o estado da porta BOTAO (no caso 2) e retorna **HIGH** se a tensão na porta for igual a 5V e **LOW** se for igual a 0V. Dentro do próprio compilador do arduino existem as seguintes definições: #define **HIGH** 1 e #define **LOW** 0. Ou seja, quando eu atribuo o estado **HIGH** em alguma função, na verdade eu estou atribuindo o valor 1 (na lógica de programação o valor 1 é atribuído ao "SIM/VERDADEIRO" e o 0 ao "Não/FALSO").

→ **if (estado == HIGH){** a função **if** ("se") é o que chamamos de operador condicional. Ele executa as instruções que estiverem entre **{ }** se a condição que estiver entre parênteses for verdadeira. No caso, a nossa condição é "estado == **HIGH**", esta condição quer dizer 'o valor da variável estado é exatamente igual a **HIGH**'. Na tabela abaixo estão todos os operadores condicionais que existem no arduino.

Condição	Significado
a == b	a é exatamente igual a b
a > b	a é maior que b
a < b	a é menor que b
a >= b	a é maior ou igual a b
a <= b	a é menor ou igual a b
a != b	a é diferente de b

Para resumir, o que fizemos nessa instrução quer dizer: "Se o valor da variável estado for HIGH" ele vai realizar o conjunto de operações até encontrar o primeiro '**}**'.

→ **else{** esta é uma continuação do comando **if** e quer dizer senão. Se a condição do **if** acima for falsa, ele vai executar as instruções entre **{ }**. **Atenção**: a instrução **else** só pode ser usada imediatamente após o **if.**

A linguagem de programação (Algoritmos)

Até aqui, a lógica dos códigos podem estar um pouco confusa, você pode está se preguntando porque eu usei determinada ordem? Como eu sei quando usar um if/else ou só um if. Para entender isso precisamos entender o que é um **algoritmo**.

*Um **algoritmo** é uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais pode ser executada mecanicamente num período de tempo finito e com uma quantidade de esforço finita.*

Estas instruções são cada linha do nosso código do arduino. No caso do código acima (o botão) escolhemos a linguagem C++ (que é o nome da linguagem utilizada pelo arduino) para escrever nosso algoritmo. A linguagem para se escrever um algoritmo pode ser qualquer uma, até mesmo o bom e velho português. Vou dar um exemplo de como isso pode ser feito para o caso do botão.

Antes de começar a escrever a escrever qualquer algoritmo, precisamos ter uma visão geral do que queremos que o nosso programa faça. No caso do sistema do botão, queremos que o LED fique aceso e enquanto o botão estiver apertado o LED fique apagado.

Para fazer o programa vamos utilizar:

```
Um BOTAO
Um LED
Vamos iniciar o programa:
      O LED só irá enviar informação (Aceso ou apagado)
      O BOTAO será só para receber informações (Apertado ou não)

Enquanto o programa estiver rodando (ou enquanto o arduino estiver
ligado):
      SE o BOTAO estiver APERTADO:
            O LED fica apagado
      SENÃO:
            O LED fica aceso
```

Este é o nosso algoritmo, o que eu fiz foi escrever em português o que o programa deve fazer a cada instante. SEMPRE que você for fazer um programa faça o algoritmo em português, isso vai fazer você programar melhor e mais rápido. Agora que temos que o algoritmo em português, precisamos traduzi-lo para a linguagem do arduino.

Dizer que eu vou usar um BOTAO e um LED é na verdade definir em que porta estes dois "caras" estão ligados, portanto:

```
#define BOTAO 2
#define LED 13
```

Agora vamos iniciar o nosso programa, no caso do arduino, precisamos da função setup() para iniciar o hardware:

```
void setup() {
```

Lembrando que precisamos depois fechar a {. Após isso, vamos informar ao arduino que aqueles dois dispositivos (LED e BOTAO) são de ENVIO (OUTPUT) e RECEBIMENTO (INPUT) de informação:

```
      pinMode(LED, OUTPUT);
      pinMode(BOTAO, INPUT);
```

Depois, quando dizemos "enquanto o programa estiver rodando" estamos no referindo a um programa que roda em *"looping",* que no arduino definimos da seguinte forma:

```
void loop() {
```

Em seguida, "SE o BOTAO estiver apertado", aqui nós dividimos o código em algumas estapas. Primeiro, criamos uma variável que vai armazenar o ESTADO do botão:

```
      int estado;
```

Depois, armazenamos na variável estado, o valor do botão:

```
      estado = digitalRead(BOTAO);
```

E agora sim, "SE o BOTAO estiver APERTADO":

```
      if ( estado == HIGH) {
```

O LED fica apagado:

```
                digitalWrite(LED, LOW);
```

SENÃO:

```
            else {
```

O LED fica ACESO:

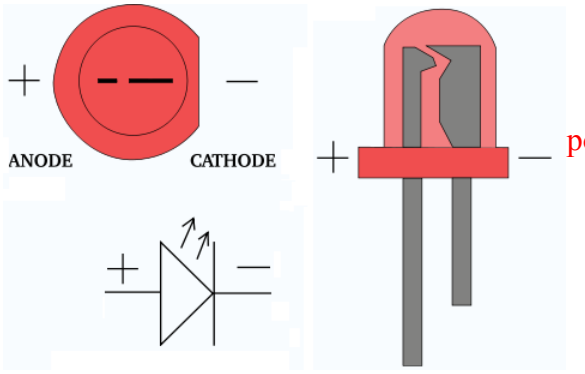
```
                digitalWrite(LED, HIGH);
```

Ficou mais fácil entender agora? TODO bom programador escreve seu algoritmo antes de passar para a linguagem da máquina. E uma triste realidade, **se você não é capaz de escrever um algoritmo em português você não será capaz de programar seu arduino**. Como você espera falar a linguagem da máquina se você não se comunica nem em português? Não quero desmotivar ninguém, apenas alertar para a importância do algoritmo. Já vi vários programadores experientes patinando em seus projetos que só conseguiram avançar quando "criaram vergonha na cara" e escreveram o algoritmo em português.

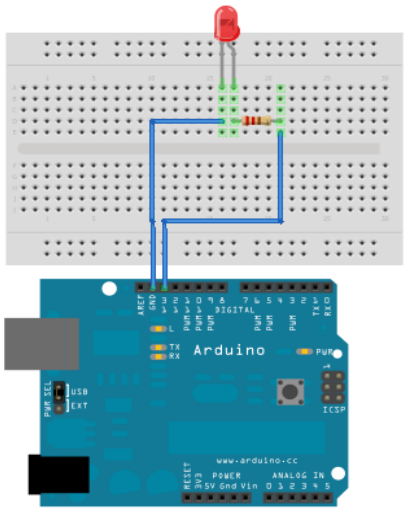
Usando as portas PWM

Como já foi dito, entre as utilidades das portas PWM está o controle da potência enviada para a porta digital. O objetivo do exemplo a seguir é fazer um LED acender e apagar de forma contínua até seu brilho máximo. Depois de ficar um determinado tempo no brilho máximo, vamos apagá-lo continuamente. Desta vez vamos utilizar um LED externo que será ligado em uma porta PWM (cada arduino tem uma forma diferente de marcar quais portas possuem a função PWM).

Monte o circuito e escreva o código:
Ao invés de fazer a ligação na porta 13 como está mostrado na figura, faça a ligação numa porta PWM a sua escolha. No código utilizarei a porta 11.



Cuidado ao ligar o LED! Ele possui um polo positivo e negativo. A "perna" maior do led é sempre o polo positivo.




```
#define LED 11
  void setup() {
        pinMode(LED, OUTPUT);
  }
  void loop() {
        for( int i = 0; i <= 255;  i++){
              analogWrite(LED, i);
              delay(30);
        }
        delay(2000);
        for( int i = 255; i >= 0; i--){
              analogWrite(LED, i);
```

```
            delay(30);
        }
    }
```

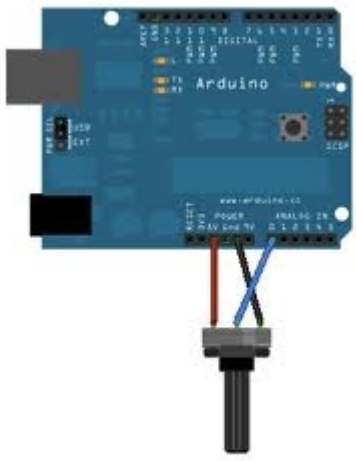
As duas novidades neste exemplo são as funções **for**() e **analogWrite**().

→ **for (int i=0; i<= 255; i++){** o comando **for**(), assim como o **if**(), é uma estrutura de controle. Ela deve ser chamada da seguinte forma **for**(início; condição; próximo passo){ e as instruções entre **{}** serão executadas enquanto a condição passada for verdadeira. O significa do for que escrevemos é começa com a variável i = 0 (veja que escrevemos **int** i = 0, toda variável deve ter seu tipo declarado) , enquanto essa variável for menor ou igual a 255 as instruções entre **{}** serão executadas e após executar um passo (passo é o processo de executar todas as instruções contidas entre **{}**) a é somado 1 a variável i (i++). No segundo for: **for(int i = 255; i >= 0; i--){** fazemos o caminho inverso, começamos com i=255 e as instruções serão executadas enquanto i >= 0 e a cada passa o valor de i é diminuído de 1 (i--).

→ **analogWrite(LED, i);** este é o comando de escrita na porta PWM. O primeiro parâmetro é a porta utilizada, no caso a porta LED, e o segundo é o valor da largura do pulso, com o seguinte detalhe: 0 corresponde a 0% e 255 corresponde a 100% da largura. Se o segundo valor for maior que 255 o arduino vai interpretar como 255.

Lendo um valor da porta analógica

Agora vamos utilizar um potenciômetro para controlar o brilho do LED. Adicione ao projeto anterior um potenciômetro, como mostrado na figura abaixo. Lembre que o fio vermelho e o fio preto podem ligados diretamente na *protoboard*.




```
#define LED 11
# define POT 0
void setup(){
  pinMode(LED, OUTPUT);
}
void loop(){
  int lido;
  lido = analogRead(POT);
  int ajustado;
  ajustado = map(lido, 0, 1023, 0, 255);
  analogWrite(LED, ajustado);
}
```

→ **analogRead(POT);** essa função só lê o valor da tensão na porta analógica e retorna um valor inteiro: 0 se a tensão for 0V e 1023 se a tensão for 5V. Então, se a tensão for, por exemplo, 2,5V ela vai retornar 512.

→ **map(lido, 0, 1023, 0, 255);** essa função é muito útil na hora de fazer ajustes em valores. Ela é utilizada da seguinte forma: **mapa**(valor, MinValor, MaxValor, MinConv, MaxConv), onde MinValor(MaxValor) é o menor(maior) valor que a variável valor pode assumir e MinConv(MaxConv) é o menor(maior) valor que o resultado pode ter. No código acima, o menor e maior valor lidos na porta analógica são 0 e 1023, respectivamente, e o menor e maior valor que queremos passar para a porta PWM são 0 e 255.

Enviando dados para o computador

Uma das habilidades mais importante do arduino é poder se comunicar com o computador através da comunicação serial. É através da comunicação serial é que iremos enviar e receber dados do computador.
Nosso próximo exemplo ensina a **enviar** o valor lido numa porta analógica para o computador. Mantendo o potenciômetro ligado ao arduino como no exemplo anterior, implementamos o código:

```
#define POT 0
void setup(){
      Serial.begin(9600);
}
void loop(){
      Serial.print("Valor do pot. = ");
      Serial.println( analogRead(POT));
      delay(300);
}
```

Após fazer o upload para o arduino clique no botão de comunicação serial como mostrado na figura ao lado. Ele vai abrir uma segunda janela onde será mostrado as informações recebidas do arduino.

→ **Serial.begin(9600);** é a função que inicia a comunicação entre o arduino e o computador. O valor 9600 informa a velocidade de transmissão dos dados, portanto, 9600 bits/s (~1Kb/s). Esse valor é chamado de "baud rate".

→ **Serial.print('Valor do pot. = ');** envia o texto que está entre **()** para o computador. Note que as ' ' não aparecem, as aspas servem para informar que o que está entre parênteses é um texto.

→ **Serial.println(analogRead(POT));** envia para o computador o valor lido na porta analógica e adiciona uma nova linha.

ATENÇÃO: A comunicação serial ocorre através de conexões feitas nas portas digitais 0 e 1. Portanto, se estivermos usando a comunicação serial, não podemos conectar nada a esta porta.

Recebendo dados do computador

Agora vamos usar o monitor serial para mandar o arduino ligar ou desligar o LED. Para isso vamos ligar utilizar o LED do próprio arduino.

```
#definde LED 13
void setup(){
      Serial.begin(9600);
      pinMode(LED, OUTPUT);
}
void loop(){
      char C ;
```

```
        if (Serial.available() > 0){
              C = Serial.read();
              if (C == 'H'){
                    digitalWrite(LED, HIGH);
              }else if (C == 'L'){
                    digitalWrite(LED, LOW);
              }
        }
}
```

Ao abrir o monitor serial, quando digitarmos H e enviarmos para o arduino, o LED irá acender, se digitarmos L o LED irá apagar. Qualquer outro valor o arduino não fará nada.

→ **char C;** criamos uma variável que é do tipo carácter. Nessa variável só podemos escrever caracteres.

→ **Serial.avaiable()** essa função informa quantos bits foram recebidos pela comunicação serial.

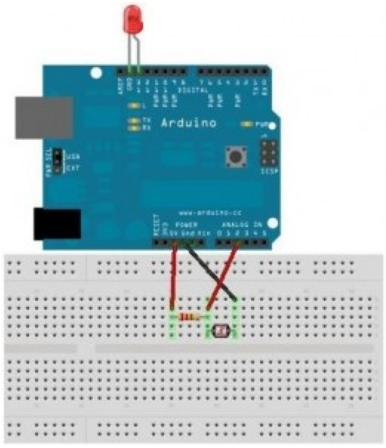
→ **C = Serial.Read();** Pega o valor que foi recebido pela comunicação serial e armazena na variável C.

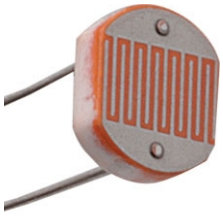
→ **else if (C == 'L'){** outra estrutura de controle é o **else if**() que é similar ao else . Contudo, as instruções dentro de **{}** só serão executadas se a condição do primeiro **if** for falsa e a do segundo for verdadeira.

A lista completa de funções, estruturas de controles etc podem ser obtidas em:
http://arduino.cc/playground/Portugues/Referencia

Componentes Básicos:

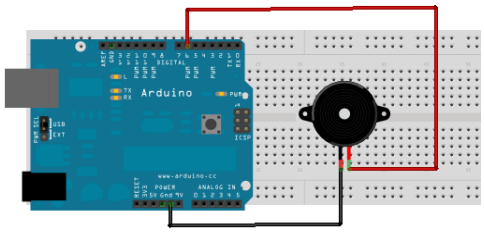
LDR



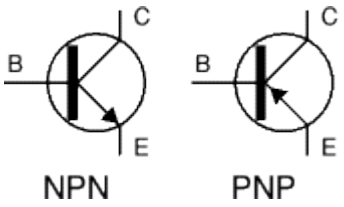


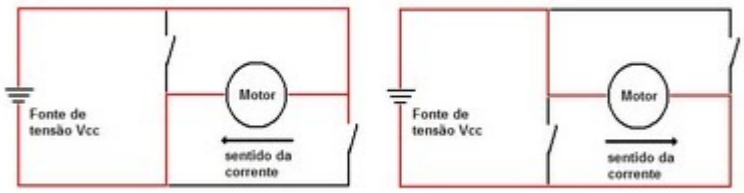
Buzzer

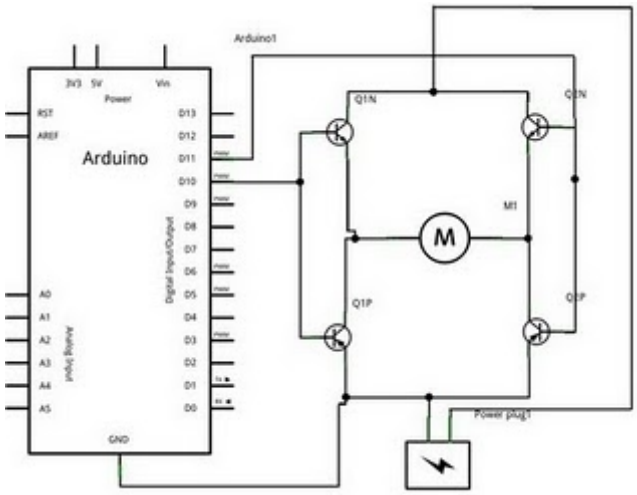
```
noTone(porta);
tone(porta, nota, duracao);
```

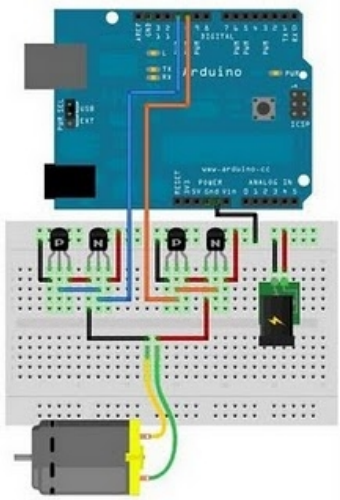


Transistores NPN e PNP
Ponte H





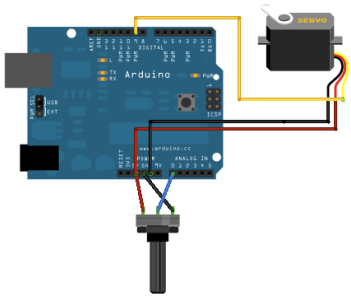




SERVOS

```
#include <Servo.h>
Servo MeuServo;
MeuServo.attach(porta);
MeuServo.write(angulo);
```

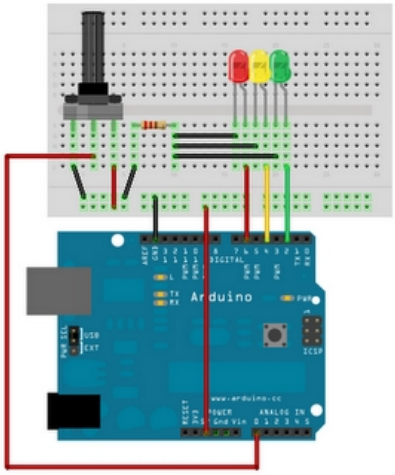
CUIDADO COM A ALIMENTAÇÃO!



SHIELDS E PERIFÉRICOS!

EXECUTANDO PROJETOS!

Nosso último projeto será criar um semáforo onde o tempo entre os sinais será determinado pelo potenciômetro.



1. Fazer o primeiro LED acender;
2. Fazer o primeiro LED apagar após um dado intervalo de tempo (dt) e acender o segundo LED;
3. Fazer o segundo LED apagar após um dado intervalo de tempo (dt) e acender o terceiro LED;
4. Verificar o valor do potenciômetro (via comunicação serial);
5. Usar o valor lido do potenciômetro para controlar o tempo que o primeiro LED fica aceso;
6. Repetir o passo 5 para os outros 2 LEDs.