

APOSTILA PARA ARDUINO

ENGENHARIA DE COMPUTAÇÃO

Mateus O. Pereira



UNIVERSIDADE DE PASSO FUNDO - UPF
CAMPUS I - PASSO FUNDO / RS

1. INTRODUÇÃO

Olá estudante de Engenharia da Computação. Bem vindo ao curso introdutório de Arduino. Esta será sua porta de entrada para futuras atividades com eletrônica e programação, incluindo componentes básicos, sensores, atuadores e até comunicação de dados. Nós, professores, desejamos um futuro de descobertas e experimentos, e que esta apostila seja uma plataforma de lançamentos para novas idéias e projetos.

Esta apostila fornece uma visão do Arduino, desde projetos básicos até atividades mais complexas. Entretanto, durante nossa disciplina ECP100, não seguiremos até o seu final.

2. OBJETIVOS

Com o estudo desta apostila, você será capaz de construir seus próprios projetos a partir de idéias. Testar seus conceitos. Após o término deste semestre, esperamos que esta apostila ainda seja de grande utilidade com valiosas informações sobre projetos do arduino para que, você, continue sua jornada no caminho do aprendizado em eletrônica e programação. E lembre-se: sua imaginação é o limite!

3. ELETRÔNICA BÁSICA

Antes de começarmos a trabalhar com o kit, precisamos nos certificar de que todos estão nivelados no mesmo conhecimento, para que, novos conceitos sejam introduzidos. Quem aqui nunca ouviu falar de tensão, corrente? Resistência? (além do chuveiro)... pilha... motor...

3.1. CONCEITOS

Os três conceitos básicos de eletrônica são: Tensão, Corrente e Resistência. Estas três grandezas são definidas como:

- Tensão: é a quantidade de energia transportada por uma corrente, em VOLTS (V). Pense na tensão como a força e o susto de um tapa quando você se encosta num fio da rede elétrica da sua casa.
- Corrente: é a quantidade de elétrons circulando por um condutor, em AMPÈRES (A). Pense na corrente como a dor ou queimadura de ter levado um choque.
- Resistência: é a intensidade que um material condutor se impõe ao fluxo dos elétrons, em OHMS (Ω). Pense na resistência sendo o quanto você suporta levar um choque.

Estas três grandezas se relacionam da seguinte forma:

- Para uma mesma resistência, quanto maior a tensão, maior será a corrente;
- Para uma mesma tensão, quanto maior a resistência, menor será a corrente; e

- Para uma mesma corrente, quanto maior a resistência, maior será a tensão.

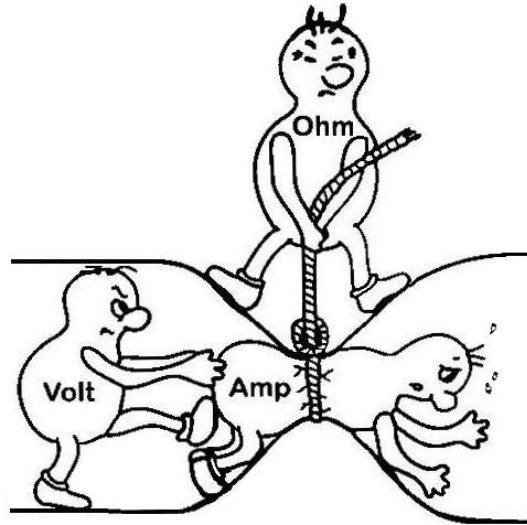
Matematicamente, temos:

$$V = R \times i$$

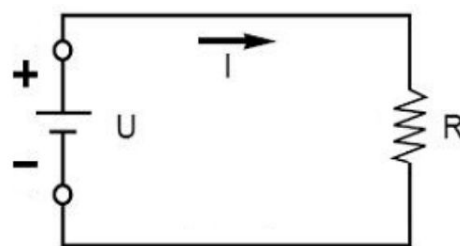
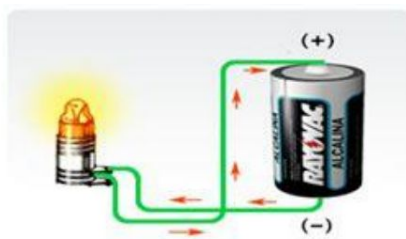
$$R = V / i$$

$$i = V / R$$

E, visualmente, temos:



A partir destas grandezas, podemos analisar um circuito elétrico. Um circuito elétrico é composto por diversos tipos de componentes, sendo estruturado, em geral, como uma fonte de energia e demais componentes que utilizam desta energia. Todos os componentes transformam a energia elétrica em outro tipo de energia, seja térmica, eletromagnética, mecânica, etc. Exemplos: resistências, lâmpadas, motores, etc...



Neste exemplo temos o circuito real (esquerda) e o seu esquemático (direita). Observe que a lâmpada está representada como uma resistência **R**. Convencionalmente, o sentido da corrente é do positivo (+) para o negativo (-). Entretanto, observe que, na real física do circuito, os elétrons se propagam do negativo (-) para o positivo (+).

Resumindo, o que precisamos saber:

- Tensão: É definida pela fonte de alimentação em uso. Seja uma rede elétrica 127v ou 220v. Seja uma fonte de computador com 3.3v, 5v e 12v. Cada componente possui seu valor (ou faixa) de operação. Ex: equipamentos 220v ou 127v; Arduino 5v; ARM 3.3v; Processadores 1.8v. Neste caso, vamos utilizar uma fonte adequada para cada um!
- Corrente: É definida pelo equipamento, dependendo do seu consumo. Devemos observar, então, se a fonte em questão é capaz de suprir tal demanda.

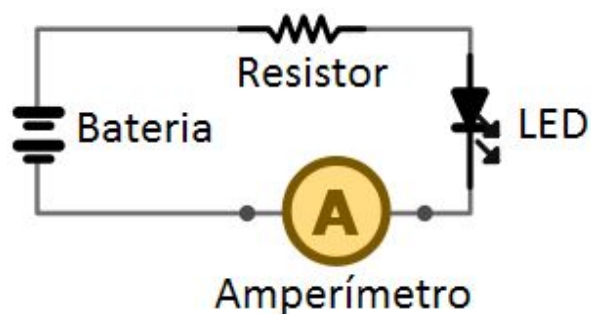
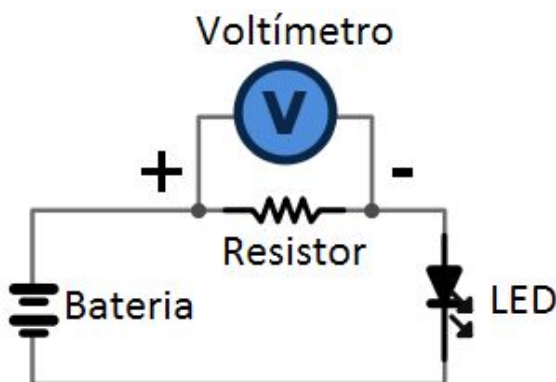
Devemos então tomar alguns cuidados: Sempre verificar se a fonte utilizada é adequada para tal componente. A tensão não é responsável por queimar ou danificar um componente. O que realmente o queima é a corrente! (o mesmo que nos dá choque). Porém, uma tensão acima do suportado pelo componente irá induzir uma corrente excessiva, e, esta sim irá queimá-lo



*Importante: No Arduino, é utilizado o valor de tensão padrão de 5 volts, chamado de nível alto ou **HIGH** em inglês. Por outro lado, o negativo possui 0 volts e é nomeado de terra, nível baixo ou **LOW** em inglês.*

3.2. INSTRUMENTOS

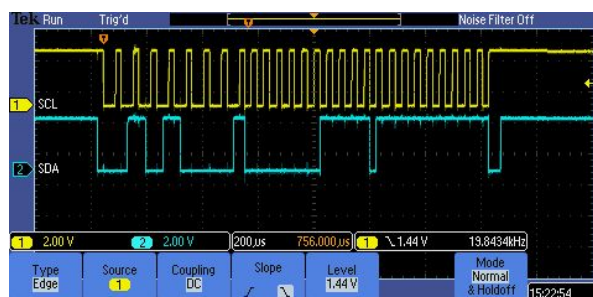
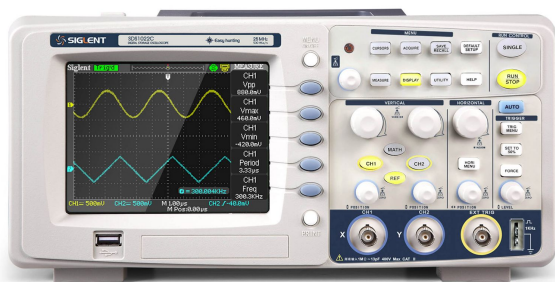
Para medir e saber sobre estas três grandezas, utilizamos instrumentos de medida. Estes instrumentos são voltímetros, amperímetros e ohmímetros, específicos para cada uma delas. O voltímetro é utilizado para medir a diferença de tensão entre dois pontos, ou seja, de uma referência até o ponto de teste. O **voltímetro** deve ser conectado em paralelo ao circuito. Ele mede a queda de energia no componente. O **amperímetro** é utilizado para medir o fluxo de corrente em uma determinada trilha do circuito, ou seja, um fio. Ele deve ser conectado em série ao circuito, e mede a quantidade de elétrons por segundo percorrendo o condutor.



Em geral, estes instrumentos estão todos juntos num único instrumento chamado **multímetro**. As imagens abaixo apresentam dois multímetros: digital (esquerda) e analógico (direita).



Além do multímetro, outro equipamento muito utilizado em eletrônica é o osciloscópio. De forma simplificada, este equipamento permite medir a tensão (como um voltímetro) e exibir o resultado em forma de gráfico, ao longo do tempo. A seguir apresentamos uma imagem do osciloscópio (esquerda) e um exemplo de comunicação digital na sua tela (direita).



Até então sabemos o básico de eletrônica para que todos estejam no mesmo nível de conhecimento, para que, daqui em diante, possamos utilizar o kit do Arduino de forma segura. Nas próximas seções iremos estudar os componentes eletrônicos e montá-los com o kit do arduino, incrementando o nível de dificuldade a cada passo.

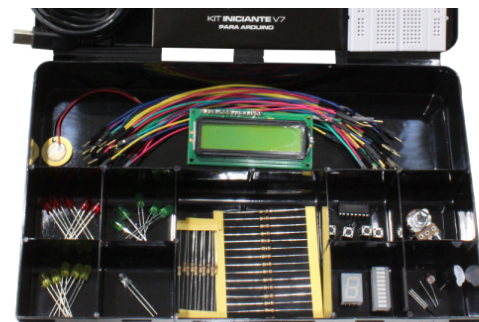
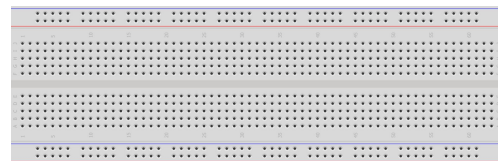
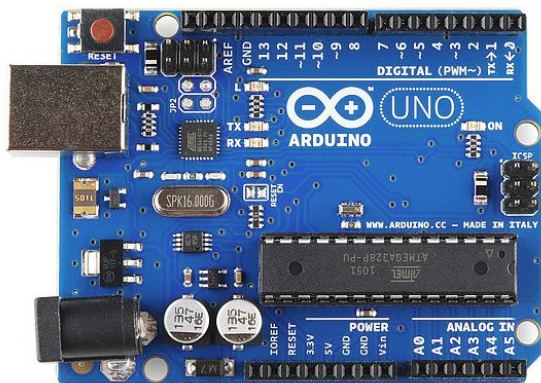
4. KIT ARDUINO

O Arduino tornou-se muito popular, pois é uma plataforma de aprendizado altamente versátil. No âmbito da eletrônica, já inclui o básico para o uso do microcontrolador; no âmbito da programação, possui uma interface simplificada para operação do componente, deixando para o usuário apenas a necessidade de implementar o necessário.

Iremos utilizar um kit com o arduino e diversos componentes. Podemos programar o arduino para interagir com o mundo externo (eletrônico, usuários) e assim construir uma aplicação (robôs, automação residencial, interface web, etc).

4.1. KIT ROBOCORE

O kit adquirido para esta disciplina é o kit robocore iniciante v7. Além dos componentes tradicionais do kit, vamos apresentá-los a outros mais, de acordo com o progresso da aula.



A imagem acima mostra o Arduino (esquerda), protoboard (acima) e os demais componentes do kit (abaixo).

4.2. IDE ARDUINO

No computador, iremos utilizar o ambiente de desenvolvimento para o arduino. Este programa possui uma forma simplificada de programá-lo, deixando para o usuário definir apenas as atividades a serem feitas na etapa de inicialização e, depois, na execução contínua.



Agora que já foi apresentado ao Arduino e a sua IDE de desenvolvimento, mãos à obra!

5. LIÇÕES

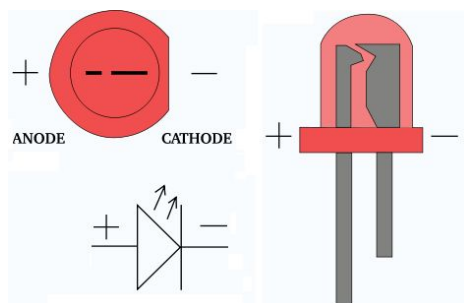
Nestas seções a seguir, vamos apresentá-lo a todos os componentes que se encontram no KIT, com projetos individuais e compostos.

5.1. PISCA-LED

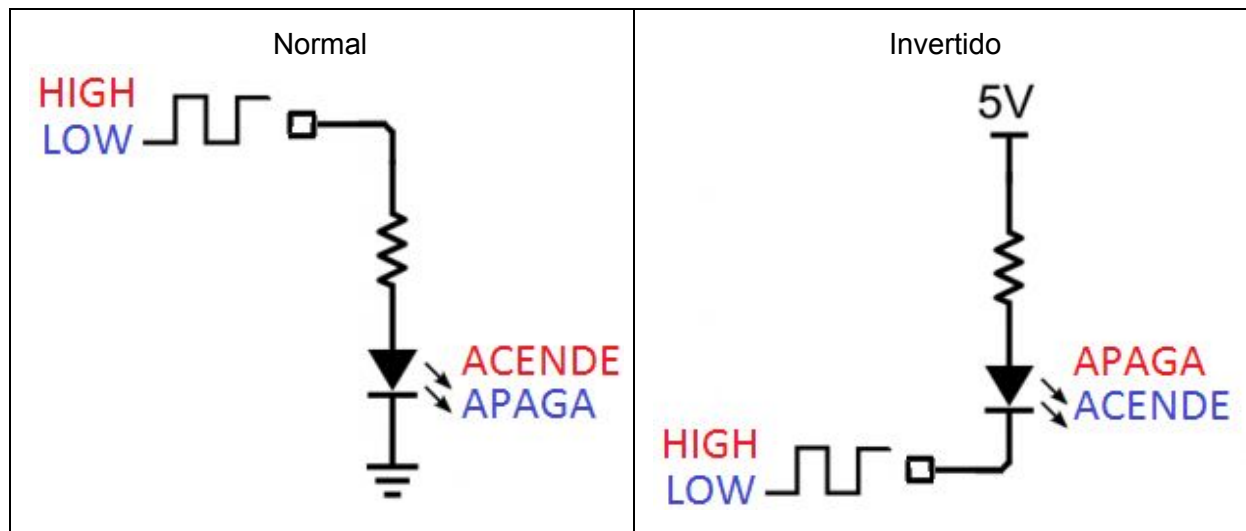
Esta é a mais famosa de todas as atividades. É o primeiro passo na união entre os conceitos de eletrônica e programação. Vamos conectar o LED a um dos pinos do Arduino e fazê-lo piscar. Primeiro, vamos conhecer os componentes que vamos utilizar:

- LED

Os LEDs estão disponíveis nas cores: verde, amarelo, vermelho e branco. Veja que o LED possui um pino maior que o outro, este é o positivo (+). O pino mais curto é o negativo (-).



Existem duas formas de se conectar um LED ao Arduino. O Arduino pode enviar o valor HIGH (positivo) para um pino e o outro já estar conectado ao LOW (negativo). Ou... o Arduino pode enviar o valor LOW (negativo) para um pino e o outro já está conectado ao HIGH (positivo). A imagem abaixo ilustra de forma clara o que acontece nestas duas condições.



Observe que o LED possui duas formas de ser conectado ao Arduino: normal e invertido. Nos nossos exemplos vamos montá-lo sempre de forma normal. Porém, existem situações ou até mesmo componentes comerciais que somente funcionam em modo invertido!

• RESISTOR

Existem dois valores de resistores diferentes neste kit: um de 300 ohms e um de 10000 ohms (a.k.a. 10k ohms). Os resistores são identificados pela sua sequência de cores:

4 Band - Code

COR	1ª BANDA	2ª BANDA	3ª BANDA	MULTIPLICADOR	TOLERANCIA
PRETO	0	0	0	1Ω	
MARROM	1	1	1	10Ω	±1% (F)
VERMELHO	2	2	2	100Ω	±2% (G)
LARANJA	3	3	3	1KΩ	
AMARELO	4	4	4	10KΩ	
VERDE	5	5	5	100KΩ	±0,5% (D)
AZUL	6	6	6	1MΩ	±0,25% (C)
VIOLETA	7	7	7	10MΩ	±0,1% (B)
CINZA	8	8	8		±0,05%
BRANCO	9	9	9	www.feiradeciencias.com.br	
DOURADO				0,1	±5% (J)
PRATEADO				0,01	±10% (K)

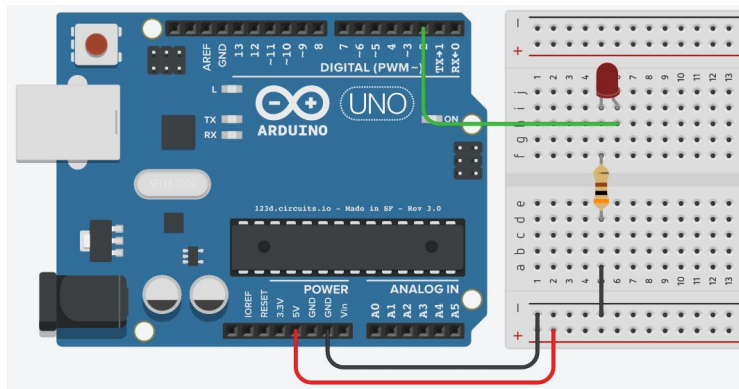
300 ohm 10k ohm



Observe que as cores para os dois resistores são idênticas! A diferença está na ordem em que são dispostas. De forma simplificada, temos: 300 ohms: 3, 0 seguido de um 0; 10k temos: 1, 0 seguido de três 0. Você vai precisar de:

Item	Quantidade
LEDs	x1
Resistores de 300 ohms	x1
Fios conectores	diversos
Protoboard e Arduino	x1

Monte o circuito abaixo. Observe bem as linhas e colunas onde os componentes estão sendo conectados! Elas estão numeradas por números e letras.




Após montado e verificado o circuito, conecte o arduino ao computador utilizando o cabo USB. É possível que os LEDs comecem a piscar ou se mantenham acesos. Isso acontece pois outro código, previamente gravado, está em execução na sua memória. Abra a interface do arduino e insira o seguinte código:

```
void setup(void)
{
    // escreva aqui seu codigo a ser executado uma vez, na inicializacao
    pinMode(2, OUTPUT);
}

void loop(void)
{
    // escreva aqui seu codigo a ser executado continuamente
    // enquanto o arduino estiver ligado
    digitalWrite(2, HIGH);
    delay(500);
    digitalWrite(2, LOW);
    delay(500);
}
```

Para gravar este código no arduino:

- Vá no menu Ferramentas → portas → escolha a porta disponível, que não seja COM1;
- Vá no menu Ferramentas → placa → escolha o Arduino UNO; e
- Clique no botão de upload:  ou tecele CTRL+U.

Se tudo correu bem, você deverá ver o LED piscar a cada 1 segundo. Neste tempo, ele fica meio segundo (500ms) aceso e meio segundo apagado.

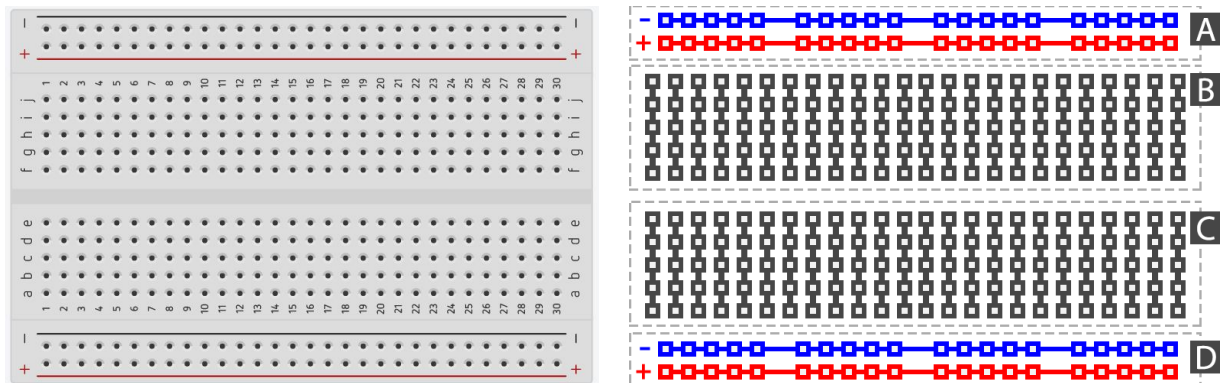
Ampliando o aprendizado, experimente:

- Adicionar mais um LED e piscá-lo de forma alternada.
- Alterar os valores dos delays.
- Alterar o modo e frequência em que os LEDs piscam.

5.2. MAIS LED'S

Neste exercício, vamos ligar mais LEDs juntos, e, mais interessante, fazê-los piscarem de forma independente, sem o uso de 'delay'. Em vez de fazer o Arduino parar e esperar a hora de acender ou apagar o LED, nós vamos marcar a hora em que devemos acender ou apagá-lo, e, quando essa hora chegar, executamos!

Antes, você deve estar se perguntando sobre essa protoboard. Como todos estes furos onde se encaixam os componentes fazem eles se ligarem? Pois bem, a resposta é simples: Cada coluna (número) é conectada entre todas as linhas (letras). E tudo isso é dividido em dois blocos, acima e abaixo. Além destes pinos, os dois segmentos de energia estão todos interligados horizontalmente. A imagem abaixo deixa mais claro como são conectados!

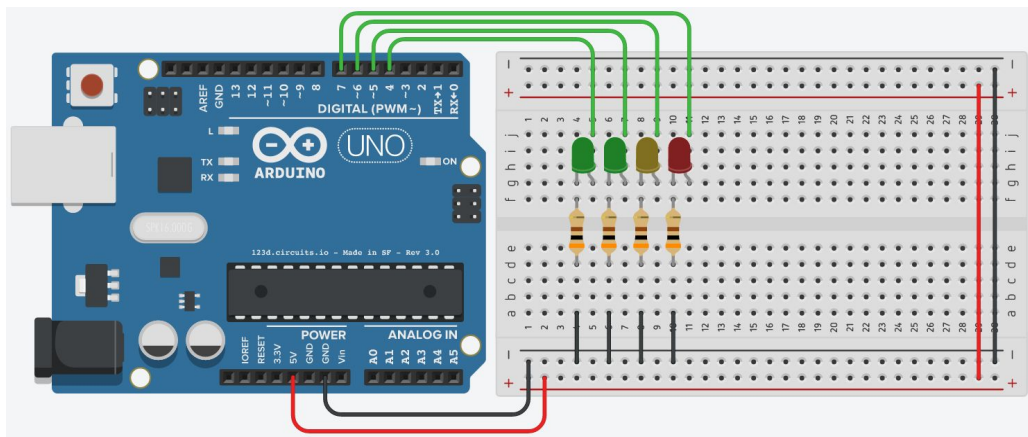


No KIT que estamos utilizando, a protoboard possui o dobro do tamanho. Ela também possui as mesmas conexões. Porém, atente que as linhas de energia (vermelho e azul / preto) não são contínuas durante todo o segmento. Elas são divididas meio a meio. Observe que as linhas possuem uma abertura no meio. Isso indica que, internamente, os contatos também são.

Para este experimento, vamos utilizar os seguintes componentes:

Item	Quantidade
LEDs (conforme as cores)	x4
Resistores de 300 ohms	x4
Fios conectores	diversos
Protoboard e Arduino	x1

Agora, monte o circuito conforme a imagem abaixo:



Após montado e verificado o circuito, conecte o Arduino ao computador utilizando o cabo USB. É possível que os LEDs comecem a piscar ou se mantenham acesos. Isso acontece porque outro código, previamente gravado, está em execução na sua memória.

Abra a IDE do Arduino e copie o código abaixo:

```
// colocamos os pinos como constantes:
int VD1 = 4;
int VD2 = 5;
int AM = 6;
int VM = 7;

// utilizamos para contar o tempo
long Tempo = 0;

void setup(void)
{
    // Configura os pinos como saída
    pinMode(VD1, OUTPUT);
    pinMode(VD2, OUTPUT);
    pinMode(AM, OUTPUT);
}
```

```

    pinMode(VM, OUTPUT);
}

void loop(void)
{
    // Faz os LEDs piscarem em sequencia
    if(millis() == Tempo + 250)
    {
        // acende o primeiro LED verde em 250 ms (1/4s)
        digitalWrite(VD1, HIGH);
    }

    if(millis() == Tempo + 500)
    {
        // acende o segundo LED verde em 500ms (1/2s)
        digitalWrite(VD2, HIGH);
    }

    if(millis() == Tempo + 1000)
    {
        // acende o LED amarelo em 1000ms (1s)
        digitalWrite(AM, HIGH);
    }


    if(millis() == Tempo + 2000)
    {
        // acende o LED vermelho em 2000ms (2s)
        digitalWrite(VM, HIGH);
    }

    if(millis() == Tempo + 5000)
    {
        // apaga todos!
        digitalWrite(VD1, LOW);
        digitalWrite(VD2, LOW);
        digitalWrite(AM, LOW);
        digitalWrite(VM, LOW);

        // o proximo ciclo comeca daqui 10 segundos
        // ou seja, o próximo Tempo sera o Tempo que
        // iniciamos mais 10000ms
        Tempo = Tempo + 10000;
    }
}

```

Para gravar este código no arduino:

- Vá no menu Ferramentas → portas → escolha a porta disponível, que não seja COM1;
- Vá no menu Ferramentas → placa → escolha o Arduino UNO
- Clique no botão de upload:  ou tecele CTRL+U.

Se tudo ocorreu bem, você deverá ver os LEDs piscarem: o da esquerda acende, após, o da direita, e assim por diante. Ao final, todos se apagam e o ciclo continua.

Ampliando o aprendizado, experimente:

- Adicionar mais LEDs ao conjunto. Lembre-se de dar nome aos pinos!
- Alterar os valores dos tempos.
- Alterar o modo em que os LEDs piscam, apagando um a um.

5.3. BOTÕES

Até então fizemos o Arduino enviar sinais para o mundo externo. Neste caso, utilizá-los para acender LEDs e determinar uma sequência de ativação. O próximo passo será, então, fazer com que o Arduino receba informações do mundo externo.

O meio mais simples para esta proposta é fazer com que o Arduino monitore um botão ligado aos seus pinos. Assim, podemos saber quando aquele botão foi pressionado ou solto. Veja a imagem de um botão.

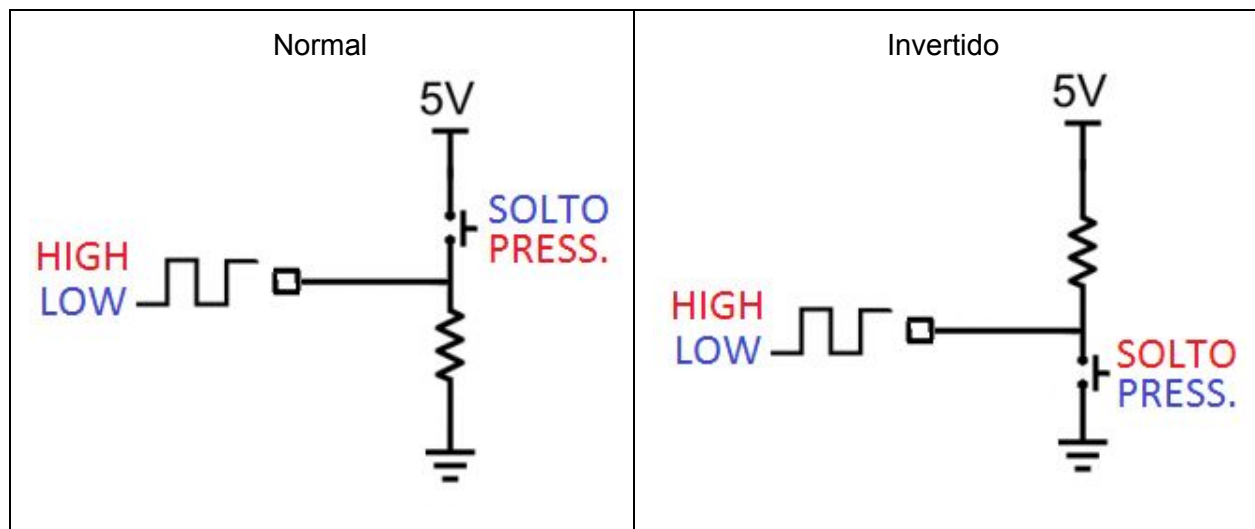


Importante: Observe que o botão possui 4 pinos. Estes pinos estão ligados em pares. Ou seja, quando o botão for pressionado, o par de um lado será ligado ao par do outro lado. O botão somente se encaixa na protoboard num sentido, sendo assim, os pares ficam lado a lado.

Aprendemos como acender um LED. O Arduino envia um valor de HIGH ou LOW para o seu pino e, assim, fará o LED acender ou apagar. Neste caso o que acontece de fato é que o pino passa a fornecer o valor de tensão de 5v (HIGH) ou 0v (LOW). De forma análoga, podemos pensar que o botão irá enviar estas informações para o pino. Assim, quando o pino estiver recebendo o valor de 5v, o resultado lido pelo Arduino será HIGH, enquanto que o pino conectado a 0v irá fornecer um resultado LOW.

Porém, o botão em si não envia os dois valores para o Arduino. Na verdade o comportamento do botão é apenas ligar ou desligar. Neste caso, podemos fazer o botão ligar o pino ao HIGH, bem como ligar ao LOW. Mas não ambos!! Vamos precisar de um resistor para

que o valor contrário seja fornecido ao pino, quando o botão estiver desligado. Vejamos a figura abaixo que ilustra duas formas de se conectar um botão ao Arduino:

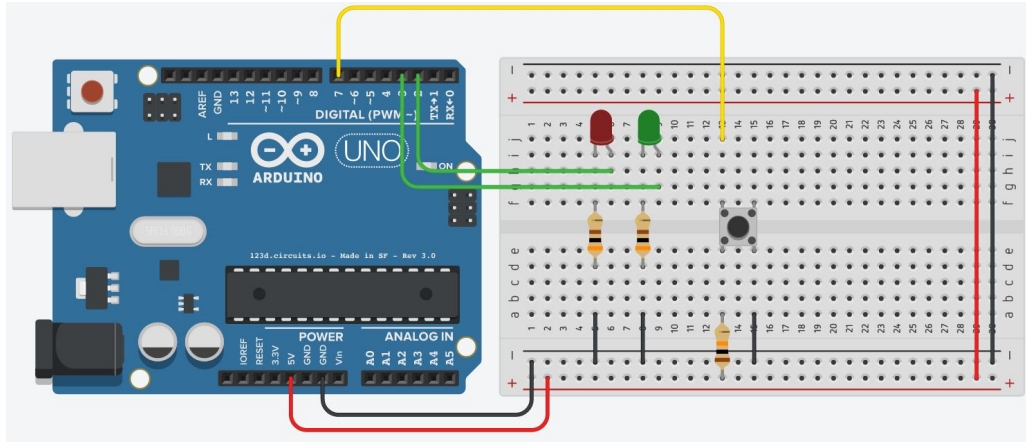


Neste caso à esquerda, observe que o botão está ligado em modo normal, e, à direita, em modo invertido. Apesar de não parecer intuitivo, mais adiante veremos a facilidade de se utilizar um botão de forma invertida.

Para este exercício, vamos montar dois LEDs. Um deles será piscante e o outro vai depender do botão. Para este experimento, vamos utilizar os seguintes componentes:

Item	Quantidade
LEDs (conforme as cores)	x2
Resistores de 300 ohms	x2
Botões	x1
Resistores de 10K ohms	x1
Fios conectores	diversos
Protoboard e Arduino	x1

Agora, monte o circuito conforme a imagem abaixo:



Importante: Observe que o resistor do LED é de 300 ohms e o resistor do botão é de 10K.

Após montado e verificado o circuito, conecte o arduino ao computador utilizando o cabo USB. É possível que os LEDs comecem a piscar ou se mantenham acesos. Isso acontece porque outro código, previamente gravado, está em execução na sua memória.

Importante: Evite pressionar os botões enquanto o Arduino ainda não foi carregado! Nunca se sabe se o último código carregado no Arduino utilizava o mesmo pino do botão como saída. Neste caso, poderá ocorrer curto-circuito e queimar este pino!

Abra a IDE do Arduino e copie o código abaixo:

```
// nomes dos pinos que vamos utilizar
int LED_VM = 2;
int LED_VD = 3;
int BTN_1 = 7;

// variavel para guardar o estado do botao:
// se ele estava solto ou pressionado na última
// vez que foi lido pelo Arduino
byte BotaoAnt = 0;

// Marca a hora de piscar o LED
long Tempo = 0;

void setup(void)
{
    // coloca os dois LEDs como saída
    pinMode(LED_VM, OUTPUT);
    pinMode(LED_VD, OUTPUT);

    // coloca o botao como entrada
```



```

    pinMode(BTN_1, INPUT);
}

// forevis!
void loop(void)
{
    // Passou da hora de piscar o LED?
    if(millis() > Tempo)
    {
        // inverte o LED verde
        if(digitalRead(LED_VD) == LOW)
        {
            // estava apagado, agora acende!
            digitalWrite(LED_VD, HIGH);
        }
        else
        {
            // estava aceso, agora apaga!
            digitalWrite(LED_VD, LOW);
        }

        // vamos inverter daqui mais 1/4s (250ms)
        Tempo = Tempo + 250;
    }

    // verifica se o botao foi pressionado! (ligou com 0v -> LOW)
    if(digitalRead(BTN_1) == LOW)
    {
        // Verifica se anteriormente estava solto (desligado)
        // neste caso, o resistor fornece 5v para o arduino -> HIGH
        if(BotaoAnt == HIGH)
        {
            // inverde o LED vermelho
            if(digitalRead(LED_VM) == LOW)
            {
                // estava apagado, agora acende!
                digitalWrite(LED_VM, HIGH);
            }
            else
            {
                // estava aceso, agora apaga!
                digitalWrite(LED_VM, LOW);
            }
        }
    }

    // Guarda o ultimo valor lido pelo botao
    BotaoAnt = digitalRead(BTN_1);
}

```

Ampliando o aprendizado, experimente:

- Adicionar mais botões: ESQ e DIR. Lembre-se: cada botão possui valor anterior!
- Alterar os valores dos delays, observe o resultado
- Alterar o modo em que os LEDs piscam

Vamos agora olhar em detalhes o que significam alguns comandos nestes códigos.

Comando	Descrição
<code>pinMode(Pino, Modo)</code>	Configura um determinado pino do Arduino para que opere de acordo com um modo específico: INPUT: entrada (padrão); OUTPUT: saída
<code>digitalWrite(Pino, Valor)</code>	Se o pino foi configurado em modo de saída (OUTPUT), então o Valor será enviado para o pino físico da placa, sendo: HIGH: alto, ligado ou 5 volts LOW: baixo, desligado ou 0 volts
<code>digitalRead(Pino)</code>	Se o pino foi configurado em modo de entrada (INPUT) ou não foi configurado, então este comando nos diz o valor imposto ao pino da placa, sendo: HIGH: alto, o pino está recebendo 5 volts no terminal; LOW: baixo, o pino está recebendo 0 volts no terminal. Se configurado como saída (OUTPUT), ele diz qual o valor foi enviado para o pino.
<code>delay(Tempo)</code>	Faz com que o Arduino pare por um tempo e depois continue a executar os demais comandos. O tempo especificado é em milissegundos (1s = 1000ms)
<code>millis()</code>	Este comando nos retorna quanto tempo, em ms, o Arduino está ligado. Este tempo inicia em 0 e aumenta em 1000 a cada 1s de execução. Logo, um minuto equivale a 60000ms. Pense nesta função como o relógio do arduino.
<code>if(condição)</code> { < comando ... > < comando ... > }	Este comando é utilizado para tomar decisões dentro do código. Com ele, o arduino analisa uma condição e decide se executa ou não determinados comandos, delimitados pelos { e }. Tal condição, sendo: LOW, 0, falso: não executa; HIGH, ≠0, verdadeiro: executa.
<code>else</code> { < comando ... > < comando ... > }	O <code>else</code> significa o caso contrario do <code>if</code> . Se a condição analisada no comando <code>if</code> estiver correta, então executa-se os comandos associados ao <code>if</code> . Caso ela esteja errada, executa-se os comandos associados ao <code>else</code> . Importante: Existe <code>if</code> sem <code>else</code> , mas não existe <code>else</code> sem <code>if</code> !

<pre>byte < nome > int < nome > long < nome ></pre>	<p>Estes comandos informam ao Arduino que estamos utilizando nomes para guardar informações e valores. Nestes exemplos, foram utilizados para lembrar o último estado do botão e o tempo em que desejamos monitorar. Os nomes byte, int, e long diferenciam na capacidade de armazenar informações.</p>
---	---

A partir destes exemplos você pode praticar projetos como:

- Acender LEDs em sequências
- Montar pares (ou trios) de LEDs em forma de sinal de trânsito e operar em tempos específicos
- Montar um pisca-pisca de natal e alternar a forma ou sequência de piscar quando pressionado um botão.

5.4. SERIAL

Uma das grandes facilidades do Arduino é que já vem embutido em sua placa um conversor USB ⇔ SERIAL. Assim, para transmitir e receber dados do computador, basta mexer na programação, sem nada de eletrônica. E já que falamos de “nada de eletrônica” então nem vamos montar nada! Utilize o mesmo circuito montado da lição anterior. (Agora, se você desmontou, terá que montar tudo de novo!)

Vamos fazer um teste serial bem simples! Copie o código abaixo para o Arduino:

```
// colocamos os pinos como constantes:
int BTN = 7;

void setup(void)
{
    // configura o pino do botão como entrada
    pinMode(BTN, INPUT);

    // configura a porta serial com velocidade de 9600
    Serial.begin(9600);


    // Manda uma mensagem inicial pro computador
    Serial.println("Arduino e Serial!");
    Serial.println("-----");
}

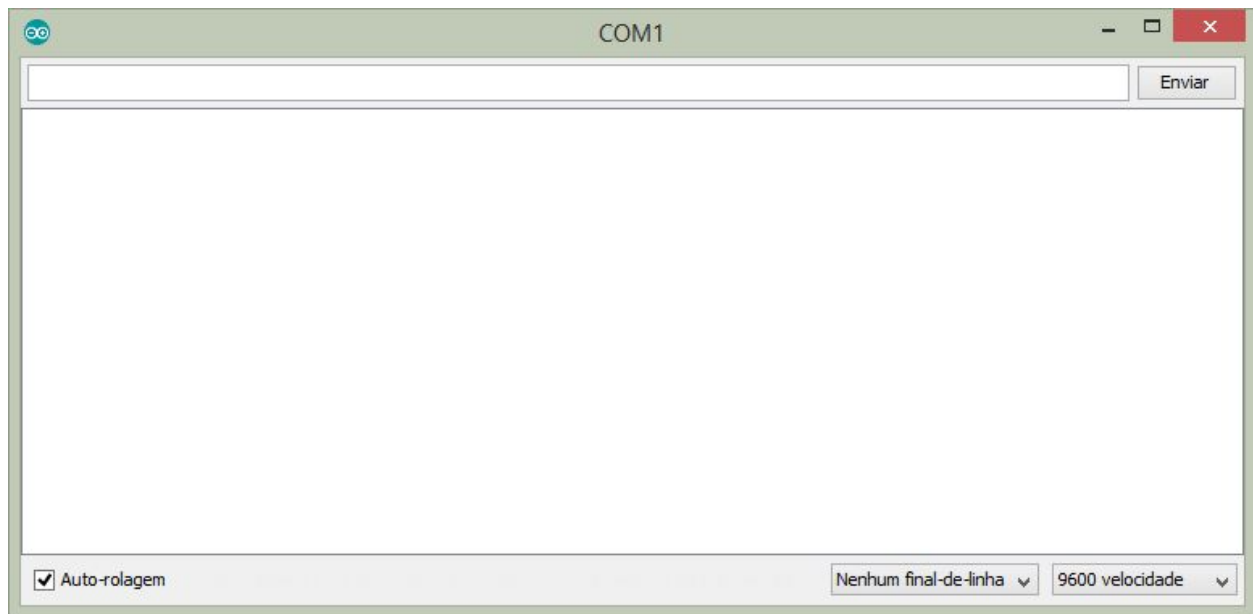
void loop(void)
{
    // verifica se o botão está pressionado
    if(digitalRead(BTN) == LOW)
    {
        // Envia uma mensagem dizendo que está
        Serial.println("Botão Pressionado!");
    }
}
```

```

    }
    else
    {
        // Envia uma mensagem dizendo que está solto.
        Serial.println("Botão Solto!");
    }
}

```

Após feito o upload para o arduino, abra o monitor serial, clicando no botão  ou pelo teclado: CTRL + SHIFT + M. Deve aparecer uma janela assim:



Vimos com este exemplo que o Arduino transmite continuamente a situação do botão. Neste caso, se ele está solto ou pressionado. Entretanto, esta solução não é muito eficiente, pois não precisamos receber a informação desta forma. Basta nos informar quando houver uma mudança. Faça um teste: remova todo o `else` e veja como se comporta.

Agora vamos trabalhar algo mais interessante com a serial. Mas, antes de já copiar o código, vamos primeiro dizer qual a idéia que ele irá executar. Os LEDs serão comandados pelo computador, para acender ou apagá-los. Será transmitida uma mensagem para o computador indicando se o botão foi pressionado ou solto. Além destas, caso seja enviado ao Arduino uma letra entre 'A' e 'Z', o Arduino irá transmiti-la de volta. Agora vamos ao código:

```

// nomes dos pinos que vamos utilizar
int LED_VM = 2;
int LED_VD = 3;
int BTN_1  = 7;

```

```

// variavel para guardar o estado do botao:
// se ele estava solto ou pressionado na ultima
// vez que foi lido pelo Arduino
byte BotaoAnt = 0;

void setup(void)
{
    // coloca os dois LEDs como saída
    pinMode(LED_VM, OUTPUT);
    pinMode(LED_VD, OUTPUT);

    // coloca o botao como entrada
    pinMode(BTN_1, INPUT);

    // Define a velocidade da comunicacao. Pode ser:
    // 1200, 4800, 9600, 19200, 38400, 57600, 115200.
    Serial.begin(9600);

    // Manda uma mensagem inicial pro computador
    Serial.println("Arduino e Serial!");
    Serial.println("-----");
}

void loop(void)
{
    // verifica se recebemos alguma coisa
    if(Serial.available())
    {
        // faz a leitura
        char Rx;
        Rx = Serial.read();

        // foi a letra '1' ?
        if(Rx == '1')
        {
            // alterna o led vermelho
            if(digitalRead(LED_VM) == LOW)
            {
                digitalWrite(LED_VM, HIGH);
            }
            else
            {
                digitalWrite(LED_VM, LOW);
            }
        }

        // foi a letra '>'?
        if(Rx == '>')
        {
            // acende o LED verde
            digitalWrite(LED_VD, HIGH);
        }
    }
}

```

```

    }

    // foi a letra '<'?
    if(Rx == '<')
    {
        // apaga o LED verde
        digitalWrite(LED_VD, LOW);
    }

    // Recebemos uma letra entre A e Z?
    if(Rx >= 'A' && Rx <= 'Z')
    {
        // transmite de volta
        Serial.println(Rx);
    }
}

// verifica se o botao está pressionado! (ligou 0v -> LOW)
if(digitalRead(BTN_1) == LOW)
{
    // Verifica se anteriormente estava solto
    // neste caso, o resistor manda o 5v para o arduino -> HIGH
    if(BotaoAnt == HIGH)
    {
        // Foi pressionado. Envia mensagem
        Serial.print("Botão pressionado aos ");
        Serial.print((float)millis()/1000);
        Serial.println(" segundos!");
    }
}
else
{
    // botão está solto. Resistor enviou o 5v -> HIGH
    // verificamos se anteriormente estava em LOW
    if(BotaoAnt == LOW)
    {
        // significa que o botão foi solto!
        // envia uma mensagem para o computador
        Serial.println("Botão solto!");
    }
}

// Guarda o ultimo valor lido pelo botao
BotaoAnt = digitalRead(BTN_1);
}

```

Neste exemplo, observe que o Arduino realiza duas tarefas em paralelo: monitorar o que chega da comunicação serial e monitorar o botão. É importante que uma tarefa não atrapalhe a outra. Imagine, agora, fazer um LED piscar junto com tudo isso? Se pensou em colocar um `delay()` para dar o tempo da piscada, atrapalhou todo o restante do código! Pois o `delay` trava o Arduino e ele não faz mais nada enquanto não terminar o prazo do `delay`. Para

contornar este problema, sempre utilizamos a `millis()` para marcar o tempo de determinadas tarefas!

A partir destes exemplos você pode praticar projetos como:

- Acender mais LEDs em sequências conforme uma letra digitada
- Adicionar um LED piscante independente das outras tarefas.
- Montar mais botões e transmitir seu estado para o computador

Vamos agora olhar em detalhes o que significam alguns comandos nestes códigos.

Comando	Descrição
<code>Serial.begin(vel);</code>	Configura a porta serial do arduino para funcionar de acordo com uma velocidade especificada. Em geral, o Arduino utiliza o padrão de 9600. Utilize 115200 caso deseje uma maior velocidade.
<code>Serial.print();</code> <code>Serial.println();</code>	Escreve uma mensagem na tela. O final <code>ln</code> indica que, após a mensagem, será transmitido um final de linha para continuar as próximas mensagens na linha abaixo. Poderá ser transmitido: Textos: "Isso é um texto" Letras: 'A', 'T', '1', '+' ... Números: 10, -5, 2.3 ... Variáveis: Tempo, <code>millis()</code> ...
<code>(float)millis()/1000</code>	Este comando obtém o tempo, em milissegundos, em que o Arduino está ligado, converte para um número com casa decimal (<code>float</code>) e, após, divide por 1000 para converter para segundos.
<code>Rx >= 'A' && Rx <= 'Z'</code>	Verifica se o caracter recebido (<code>Rx</code>) contém uma letra entre 'A' e 'Z'. Os dois símbolos & significam que as duas condições, de maior igual e menor igual, devem acontecer simultaneamente.

5.5. SINAL ANALÓGICO

Estudamos até agora que o Arduino opera em 5v. Assim, o Arduino utiliza os valores HIGH e LOW para representar, respectivamente, o pino em 5v e o pino em 0v (Terra ou GND). Porém, o mundo real não é "digital". Existem valores de tensão entre o GND e o 5v. Aliás, abaixo e acima destes limites também.

O Arduino possui alguns pinos especiais que podem medir estes valores. Entretanto, somente entre GND e 5v. A menor quantidade de informação armazenada pelo Arduino é o `bit`, com valores que podem ser apenas 0 e 1. Eis o porque somente temos os valores HIGH e LOW para os pinos! Imagine que queremos medir mais uma condição: a metade entre o GND e

5v, que seriam 2,5v. Precisamos de um “meio bit” para representar o 2,5v. Algo entre o HIGH e LOW. Mas isso não existe nos sistemas digitais.

Importante: conectar o arduino em valores de tensão abaixo de 0v (GND) ou acima de 5v pode queimar o pino permanentemente!

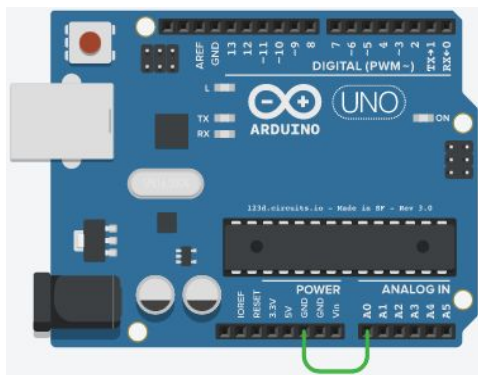
Por outro lado, podemos utilizar mais de um único bit para armazenar estes valores. Vamos imaginar que o pino pode nos fornecer mais valores, além do HIGH e LOW. Veja a tabela abaixo.

# de bits	# de valores	Valor mínimo (GND)	Valor máximo (5v)
1 (pino digital)	$2^1 = 2$	0 (LOW)	1 (HIGH)
2	$2^2 = 4$	0	3
3	$2^3 = 8$	0	7
8	$2^8 = 256$	0	255
10 (pino analógico)	$2^{10} = 1024$	0	1023
12	$2^{12} = 4096$	0	4095

Todos os pinos do Arduino que tem a função digital (a maioria), são apenas de 1 bit, e assim, podem nos fornecer apenas valores LOW e HIGH. Mas alguns outros pinos possuem função analógica, e podem nos fornecer valores entre o LOW e HIGH.

5.6. ENTRADA ANALÓGICA

O Arduino possui alguns pinos que são capazes de nos fornecer valores numéricos de acordo com sinais analógicos conectados a ele. Observe que eles estão nomeados de A0 até A5. Vamos fazer um pequeno exemplo sobre a entrada analógica. Para isso, você vai precisar do Arduino e apenas um fio.



Observe que conectamos o GND ao pino A0. Agora, copie este código para o Arduino e, após o upload, abra o monitor serial (CTRL + SHIFT + M). Certifique-se de que ele está em 9600bps para que a comunicação dê certo!

```
void setup()
{
    // configura a serial na velocidade de 9600
    Serial.begin(9600);
}

void loop()
{
    // armazena o valor lido
    int Valor;

    // faz a leitura do pino A0
    Valor = analogRead(A0);

    // envia pela serial o valor lido
    Serial.print("Valor: ");
    Serial.println(Valor);

    // espera um segundo até a proxima
    delay(1000);
}
```

Observe o monitor serial. Vai aparecer a mensagem: `Valor: 0` repetidas vezes. Experimente, agora, conectar o pino A0 no pino 5v. A mensagem que vai aparecer no monitor serial será `Valor: 1023`. Por fim, conecte o pino A0 agora ao pino 3v3. Qual mensagem apareceu? Você saberia explicar o porquê deste valor?

Basta fazer uma regra de três. Se 0 representa o 0v e 1023 representa o 5v, então quanto representa o 3v3? 675. Pois: $1023 * 3,3 / 5 = 675,18$ (mas não temos casa decimal, então cortamos o 0,18).







5.7. SAÍDA ANALÓGICA

Apesar de parecer intuitivo, no Arduino a saída analógica não tem nada a ver com o oposto de entrada analógica. Pra começar, os pinos utilizados como saída analógica não são os pinos A0-A5. Pegue o arduino e observe que alguns pinos digitais possuem uma marcação de um ~ ao lado do seu nome. Estes pinos são: 3, 5, 6, 9, 10 e 11. Somente estes pinos podem “simular” uma saída analógica.

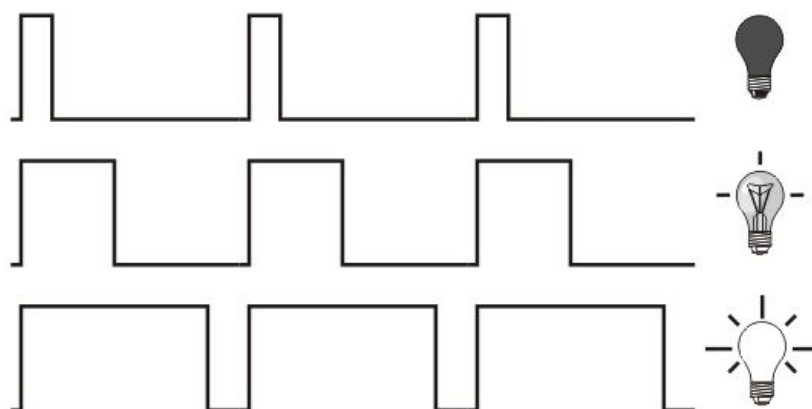
Não entraremos aqui no fundamento da saída analógica, até mesmo porque o Arduino não possui eletrônica interna para isso. Ele utiliza um outro recurso para contornar a ausência

desta saída. No Arduino, existem algumas saídas digitais que podem ser configuradas para alternar seu valor a certos intervalos de tempo. Vejamos uma analogia.

Imagine que estamos em um quarto e vamos acender e apagar uma lâmpada a intervalos regulares. O que acontece é que veremos o quarto iluminado, e escuro. Alternadamente. Agora vamos aumentar a velocidade em que se acende e apaga a lâmpada. Vamos vê-la piscando muito mais rápido, e, quanto mais rápido isso for feito, menos vamos conseguir enxergar a transição do aceso ao apagado. A um certo ponto, não vamos mais conseguir enxergar o quarto escuro ou iluminado, mas, sim, uma combinação entre estas duas condições. Caso elas sejam de tempos iguais, o resultado é o quarto com metade da iluminação total. Se utilizarmos 2 tempos acesos e um tempo apagado, o resultado será um quarto mais iluminado, com 66% de intensidade.

Valor A	Valor B	Média
100% 	0% 	50% 
50% 	0% 	25% 

Agora, troque a lâmpada acesa e apagada pelo pino do arduino, respectivamente em HIGH e LOW. Se o pino é alternado entre HIGH e LOW a tempos iguais, o resultado será $(0v + 5v) / 2 = 2,5v$! Se for alternado em 4 tempos LOW e 1 tempo HIGH, temos o resultado da saída em $(1*5v + 4*0v) / (1 + 4) = 1v$. Veja o exemplo abaixo.



<http://cdn.mikroe.com/ebooks/sites/3/2016/01/25153325/pic-microcontrollers-programming-in-c-chapter-03-image-68.gif>

```
// LED no pino 6
int LED = 6;

void setup()
{
    // configura o LED como saída
    pinMode(LED, OUTPUT);
}

void loop()
{
    // acende o LED com potência total (100% = 255)
    analogWrite(LED, 255);
    delay(2000);

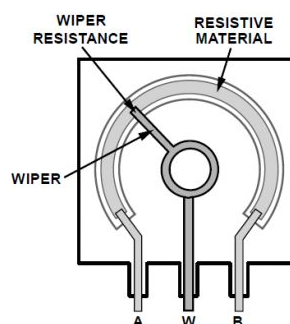
    // acende com metade da potencia
    analogWrite(LED, 128);
    delay(2000);

    // acende com 1/5 da potencia
    analogWrite(LED, 51); // 51 = 255 / 5
    delay(2000);
}
```

Neste exemplo o LED irá alternar o seu brilho. Experimente alternar os valores que são repassados ao comando `analogWrite()`.

5.8. POTENCIÔMETRO

Vimos nas lições anteriores que os resistores possuem diferentes valores. Neste kit, entretanto, apenas os valores de 300 ohms e 10k ohms. Além dos resistores fixos, temos resistores variáveis, chamados de trimpot ou potenciômetros. O valor da sua resistência varia de 0 até um valor máximo definido para cada potenciômetro. Este componentes são muito utilizados em controles de volume e equalizadores de aparelhos de som mais antigos.



Fonte: <https://www.quora.com/What-is-potentiometer>

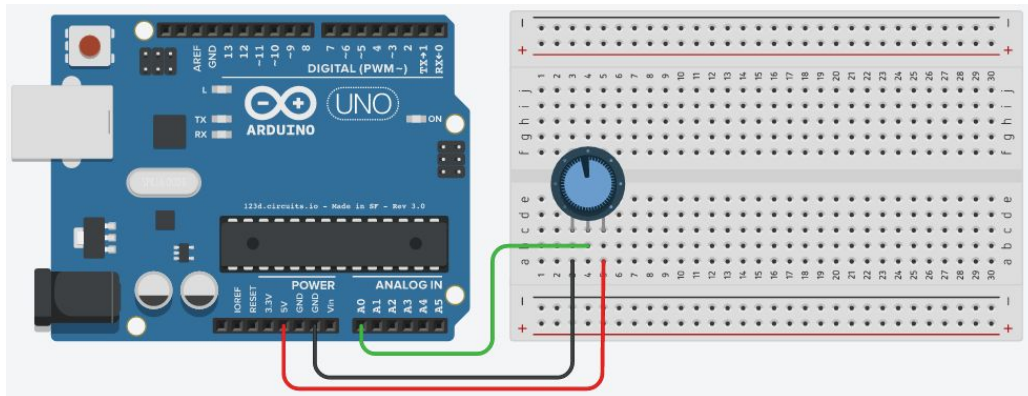
No kit do Arduino, o potenciômetro é de 10k ohms. Observe na figura que ele possui três pinos: A, W e B. Os pinos da extremidade (A e B) são como um resistor comum, no valor definido para este componente (neste kit, 10k). O pino central (w) é um ponto no meio do caminho entre A e B.

O valor do resistor no pino central varia conforme a alavanca é rotacionada. Todo o curso da alavanca é de 270 graus. Tomando como referência o pino A sendo o início, logo, consideramos B como sendo os 270 graus. A tabela abaixo então define alguns valores notáveis durante o curso.

Ângulo	$R_{A \rightarrow W}$	$R_{W \rightarrow B}$	Descrição
0	0	10k	Fim de curso sentido anti-horário
45°	1.66k	7.33k	Horizontalmente à esquerda
135°	5k	5k	Vertical (centralizado)
225°	7.33k	1.66k	Horizontalmente à direita
270°	10k	0	Fim de curso sentido horário

Observe na tabela que a soma das resistências de $A \rightarrow W$ e $W \rightarrow B$ são sempre os 10k do valor total do potenciômetro. Da mesma forma que a resistência é proporcional ao ângulo, podemos utilizá-lo para nos fornecer uma tensão, também proporcional. Neste caso, vamos

conectar os pinos A e B, respectivamente, ao GND (Terra) e ao 5v. O pino central, W, então irá fornecer uma tensão proporcional de 0v a 5v dependendo da posição da alavanca. Monte o circuito abaixo:



Agora, grave este código no Arduino:

```
void setup()
{
    // vamos usar a serial
    Serial.begin(9600);
}

void loop()
{
    // guarda o valor lido pelo conversor analogico
    int Pot;

    // faz a leitura do potenciometro
    Pot = analogRead(A0);

    // manda via serial
    Serial.print("Potenciometro: ");
    Serial.println(Pot);

    // aguarda um segundo
    delay(1000);
}
```

A partir deste exemplo você pode praticar fazendo alterações como:

- Remover o `delay()` e substituir pela `millis()`.
- Acender LEDs nas cores verde, amarelo e vermelho conforme o valor lido.
- Fazer a leitura do potenciômetro a cada 100ms, mas somente enviar pela serial caso o novo valor seja diferente do anterior (observe que nem sempre dá certo).
- Aumente este filtro para uma diferença de ± 5 entre cada leitura.

5.9. DISPLAY LCD

Um dos componentes mais interessantes de se utilizar é o display LCD. Com ele podemos exibir mensagens sem precisar da serial, e, assim, acompanhar a execução do que se passa no Arduino.

O display que acompanha este kit possui 2 linhas com 16 caracteres cada, chamado, assim de Display 16x2. Funciona apenas em modo texto, ou seja, apenas letras, números e outros símbolos pré definidos podem ser utilizados. Não é possível criar desenhos, pois, este não é um display gráfico.



Fonte: https://www.arduino.cc/en/uploads/Tutorial/lcd_photo.png

Existem diversos tamanhos de displays LCD, como: 8x2, 16x1, 20x4, 40x2, etc. Sempre com referência na quantidade de colunas e linhas. Para mais detalhes, visite o site oficial do arduino: <https://www.arduino.cc/en/Tutorial/HelloWorld>

Para esta lição, utilize o esquema e código fornecido pelo próprio tutorial do Arduino!

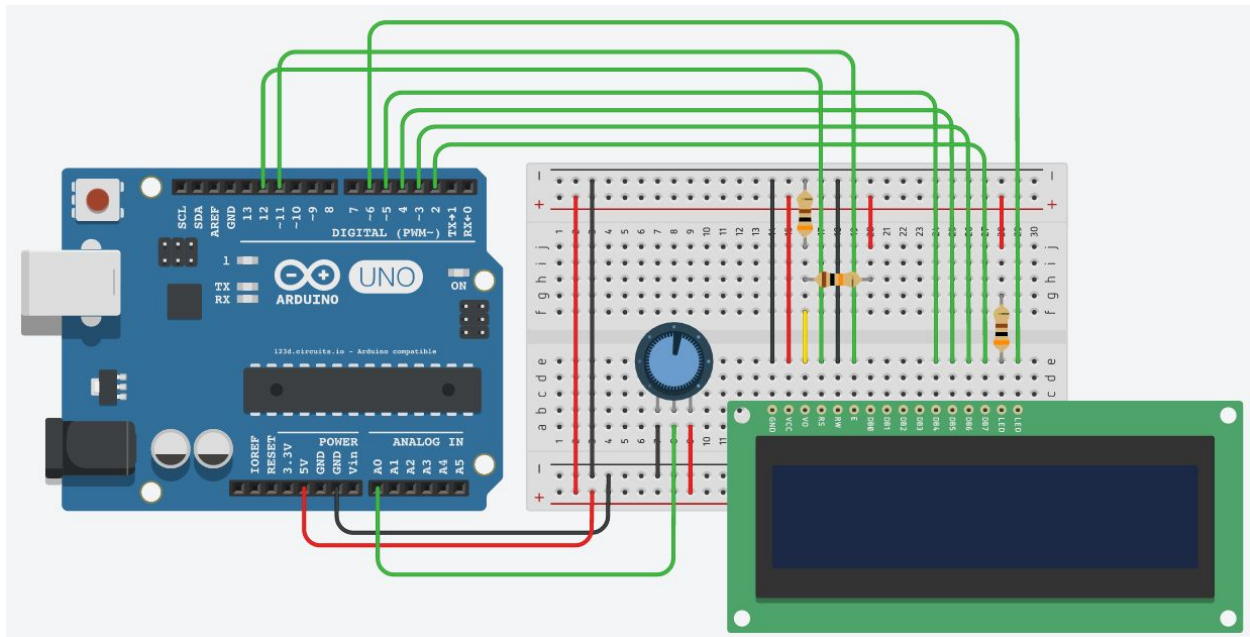
Importante: antes de montar o circuito, grave o código no Arduino!! Desta forma, assim que ligarmos o circuito o display já deve funcionar. Caso isso não aconteça, desligue o Arduino imediatamente! Por outro lado, se você deixar para gravar o código depois de montar o circuito, não terá como saber se foi montado errado ou se o display não ligou porque o Arduino não estava gravado.

5.10. DISPLAY LCD II

Vamos agora fazer outro projeto com o LCD! Além do exemplo acima, vamos controlar, também, o brilho da luz de fundo (backlight) do display. Você observou na lição anterior que o potenciômetro foi utilizado para ajustar o contraste da tela, deixando a imagem mais clara ou mais escura. Este ajuste é feito de acordo com o valor de tensão aplicado ao terceiro pino do LCD (pino Vo).

Em geral, não precisamos ficar sempre alterando este valor, então, podemos fazer um ajuste fixo, com resistores fixos. Para isso, vamos ligar um resistor de 10k entre o 5v e o pino Vo do LCD, e outro resistor de 300 ohms entre o pino Vo e o GND (0v). Desta forma, estes dois resistores formam um divisor de tensão que irá fornecer um valor de 0,15v para o pino Vo, suficiente para deixar a tela com um bom contraste.

Agora, podemos utilizar o potenciômetro para ajustar o brilho. Monte o circuito abaixo:



Resumindo, o conversor analógico, através de `analogRead()`; nos fornece um valor entre 0 e 1023. Porém, o comando `analogWrite()`; somente aceita valores entre 0 e 255. Logo, precisamos fazer uma conversão. Neste caso, basta dividir o valor lido por 4.

Observe que o LED do display (backlight) foi conectado de forma invertida (veja lição PISCA-LED, página 7). Assim, a saída em 0 irá acender o backlight ao máximo, enquanto a saída em 255 irá apagá-lo por completo. Para isso, então, vamos inverter o valor após dividi-lo por 4, fazendo: $255 - \text{Valor}$. Observe o código abaixo antes de gravá-lo, para entender o que o Arduino vai fazer.

```

// vamos utilizar os comandos para o display LCD
#include <LiquidCrystal.h>

// Pinos do LCD: RS, E, D4, D5, D6, D7
// RW sempre em GND
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// pino do LED de backlight
int LED = 6;

// hora, minuto e segundo ligados
byte Hora, Min, Seg;

// separador ':'
char Pontos = ':';

// intervalo para atualizar a tela
long TempoHora = 0;

// intervalo para ler o potenciometro
long TempoBrilho = 0;

void setup()
{
    // pino de backlight
    pinMode(LED, OUTPUT);

    // Inicializa o LCD
    lcd.begin(16, 2);

    // posiciona o cursor na 8a coluna da primeira linha
    lcd.setCursor(8, 0);
    lcd.print("00:00:00");
}

void loop()
{
    // deu hora de atualizar?
    if(millis() > TempoHora)
    {
        // daqui a meio segundo!
        TempoHora = TempoHora + 500;

        // verifica se os pontos foram escritos
        // e inverte para piscar na tela
        if(Pontos == ':')
        {
            // era ':' muda para ' '
            Pontos = ' ';
        }
        else
    }
}

```

```

{
    // era ' ' muda para ':'
    Pontos = ':';

    // aproveita para incrementar os segundos
    // uma vez a cada dois ciclos de 500ms
    Seg++;

    // caso tenha dado os 60 segundos
    if(Seg >= 60)
    {
        // fechou um minuto a mais.
        Seg = 0;
        Min++;

        // caso tenha dado os 60 minutos
        if(Min >= 60)
        {
            // fechou uma hora a mais
            Min = 0;
            Hora++;
        }
    }
}

// posiciona
lcd.setCursor(8, 0);

// escreve a hora no LCD: HH:MM:SS
if(Hora < 10) lcd.print('0');
lcd.print(Hora);
lcd.print(Pontos);
if(Min < 10) lcd.print('0');
lcd.print(Min);
lcd.print(Pontos);
if(Seg < 10) lcd.print('0');
lcd.print(Seg);
}

// faz a leitura do potenciometro
if(millis() > TempoBrilho)
{
    int Pot;

    // repete daqui 3/4s
    TempoBrilho = TempoBrilho + 750;

    // le o valor do potenciometro
    Pot = analogRead(A0);

    // divide por 4, pois, 0...255 equivale a 1/4 de 0..1023
    Pot = Pot / 4;
}

```

```

    // inverte pois o LED está ligado ao contrario
    Pot = 255 - Pot;

    // aciona o backlight
    analogWrite(LED, Pot);
  }
}

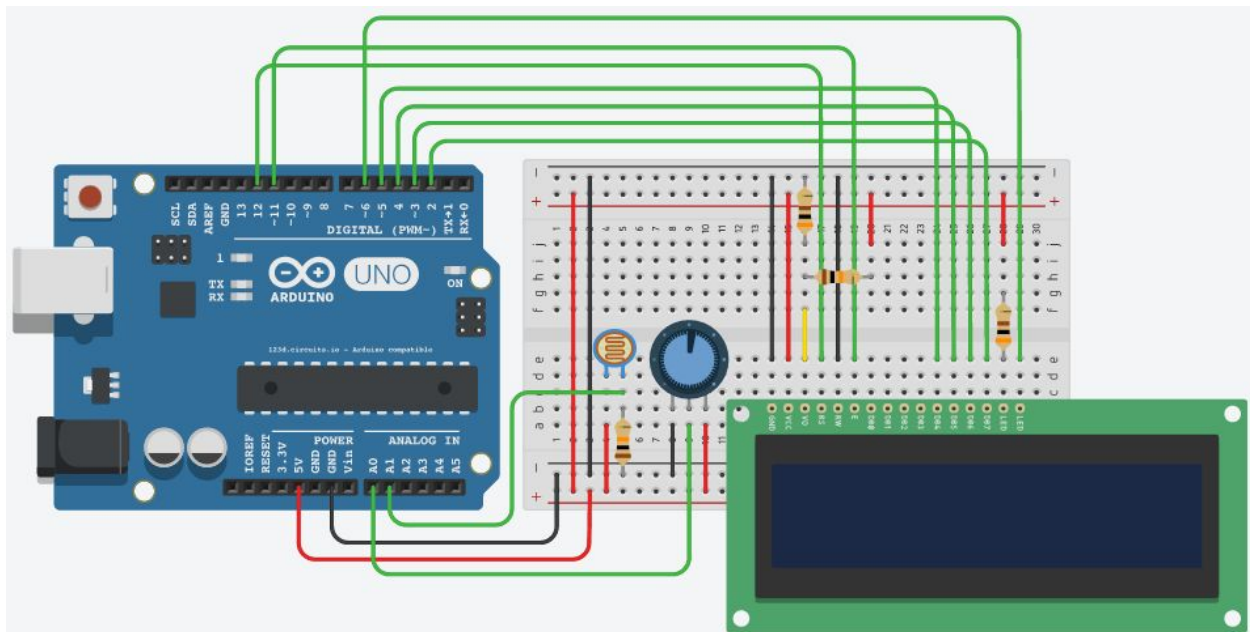
```

5.11. SENSOR DE LUZ LDR

Estudamos até então o uso de resistores de duas formas: limitar a corrente elétrica ao acionar um LED; e fornecer o nível lógico HIGH ou LOW quando o botão estiver solto. Existe um outro resistor no kit, chamado LDR, que muda o valor da sua resistência conforme a intensidade de luz. Se você pensou naquelas lâmpadas que acendem sozinhas quando escurece, acertou! Estes acendedores automáticos possuem este LDR como forma de saber se é dia ou noite (na prática se está claro ou escuro).



Monte o circuito abaixo, a partir do circuito da aula passada!



Este componente é utilizado de forma analógica, pois, a saída é um valor de tensão que pode variar conforme o valor do resistor e da alimentação. Logo, iremos utilizar o conversor analógico do Arduino para obter seu valor. Observe que o conversor analógico, de fato, faz a leitura do valor de tensão, e não o valor da resistência em si. Mais ainda, este valor de tensão não varia desde o 0v até os 5v. Desta forma, o valor lido pelo comando `analogRead()`; não irá cobrir toda a faixa de 0 a 1023 da mesma forma que acontecia com o potenciômetro.

Nesta lição, iremos trabalhar mais a fundo agora a relação entre o sensor LDR e o backlight do display. Não podemos assumir toda a faixa de 0 a 1023 para acionar o PWM de 0 a 255. Vamos precisar saber o valor mínimo e máximo do LDR e, a partir deles, fazer uma regra de três e encontrar o valor correto para o PWM. Primeiro, grave este código no Arduino:

```
// vamos utilizar os comandos para o display LCD
#include <LiquidCrystal.h>

// Pinos do LCD: RS, E, D4, D5, D6, D7
// RW sempre em GND
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// pino do LED de backlight
int LED = 6;

// hora, minuto e segundo ligados
byte Hora, Min, Seg;

// separador ':'
char Pontos = ':';

// intervalo para atualizar a tela
long TempoHora = 0;

// intervalo para ler o LDR
long TempoBrilho = 0;

// valores maximo e minimo para o LDR
int LdrMin = 320;
int LdrMax = 950;

void setup()
{
    // pino de backlight
    pinMode(LED, OUTPUT);

    // Inicializa o LCD
    lcd.begin(16, 2);

    // posiciona o cursor na 8a coluna da primeira linha
    lcd.setCursor(8, 0);
    lcd.print("00:00:00");
}
```

```

void loop()
{
    // deu hora de atualizar?
    if(millis() > TempoHora)
    {
        // daqui a meio segundo!
        TempoHora = TempoHora + 500;

        // verifica se os pontos foram escritos
        // e inverte para piscar na tela
        if(Pontos == ':')
        {
            // era ':' muda para ' '
            Pontos = ' ';
        }
        else
        {
            // era ' ' muda para ':'
            Pontos = ':';

            // aproveita para incrementar os segundos
            // uma vez a cada dois ciclos de 500ms
            Seg++;

            // caso tenha dado os 60 segundos...
            if(Seg >= 60)
            {
                // fechou um minuto a mais.
                Seg = 0;
                Min++;

                // caso tenha dado os 60 minutos
                if(Min >= 60)
                {
                    // fechou uma hora a mais
                    Min = 0;
                    Hora++;
                }
            }
        }

        // posiciona
        lcd.setCursor(8, 0);

        // escreve a hora no LCD: HH:MM:SS
        if(Hora < 10) lcd.print('0');
        lcd.print(Hora);
        lcd.print(Pontos);
        if(Min < 10) lcd.print('0');
        lcd.print(Min);
        lcd.print(Pontos);
    }
}

```

```

        if(Seg < 10) lcd.print('0');
        lcd.print(Seg);
    }

    // faz a leitura do potenciometro
    if(millis() > TempoBrilho)
    {
        int Ldr;
        long Pwm;

        // repete daqui 3/4s
        TempoBrilho = TempoBrilho + 750;

        // le o valor do LDR em A1
        Ldr = analogRead(A1);

        // faz a regra de três para o PWM
        Pwm = Ldr - LdrMin;           // valor mínimo em 'zero'
        Pwm = Pwm * 255;              // escala de 0 a 255 do PWM
        Pwm = Pwm / (LdrMax - LdrMin); // escala do LDR

        // verifica se está dentro dos nossos limites do PWM!
        if(Pwm < 10) Pwm = 10;        // menos que 10 o brilho é muito pouco!
        if(Pwm > 255) Pwm = 255;      // este o valor máximo do PWM!

        // inverte o PWM pois o LED está invertido!
        Pwm = 255 - Pwm;

        // aciona o backlight
        analogWrite(LED, Pwm);

        // escreve no display
        lcd.setCursor(0, 0);
        lcd.print("L:");
        if(Ldr < 1000) lcd.print('0');
        if(Ldr < 100)  lcd.print('0');
        if(Ldr < 10)   lcd.print('0');
        lcd.print(Ldr);
    }
}

```

Neste exemplo, o backlight não irá se acender muito bem. Isso acontece pois cada LDR possui um valor específico de resistência. Então o que devo fazer? Calibrar de acordo com o seu circuito! Observe no código que existem as seguintes variáveis:

```

// valores maximo e minimo para o LDR
int LdrMin = 320;
int LdrMax = 950;

```


Você deverá alterar estes valores para os valores que está observando no seu display. Deixe o LDR exposto à luz e veja qual será o valor MÁXIMO lido pelo conversor. Agora, tape o LDR para que fique no escuro e veja na tela qual será o valor MÍNIMO lido pelo conversor. Altere estas duas variáveis para estes valores que você leu e pronto!

6. HARDWARE

- 6.1. ok LEDs
- 6.2. ok BOTÕES
- 6.3. ok SERIAL
- 6.4. ok POTENCIÔMETRO
- 6.5. LEDs & PWM
- 6.6. ok DISPLAY LCD
- 6.7. ok SENSOR DE LUZ LDR
- 6.8. LM35
- 6.9. BUZZER

7. SOFTWARE

- 7.1. ok SETUP E LOOP
- 7.2. +- TIPOS DE DADOS
- 7.3. ok TOMADAS DE DECISÃO
- 7.4. REPETIÇÕES
- 7.5. VETORES
- 7.6. FUNÇÕES