Curso Básico de Arduino

Marco Polo Moreno de Souza

WWW.MARCOPOLO.UNIR.BR

Primeira impressão, setembro de 2017

ARDUINO™

Sumário

Prefácio da primeira edição

Apostila preparada para o Curso Básico de Arduino, ministrado no Laboratório Didático de Física e Química do Departamento de Física da Universidade Federal de Rondônia, Campus Ji-Paraná, entre 15 de setembro e 13 de outubro de 2017. Dúvidas, sugestões e correções podem ser enviadas para o e-mail marcopolo@unir.br.

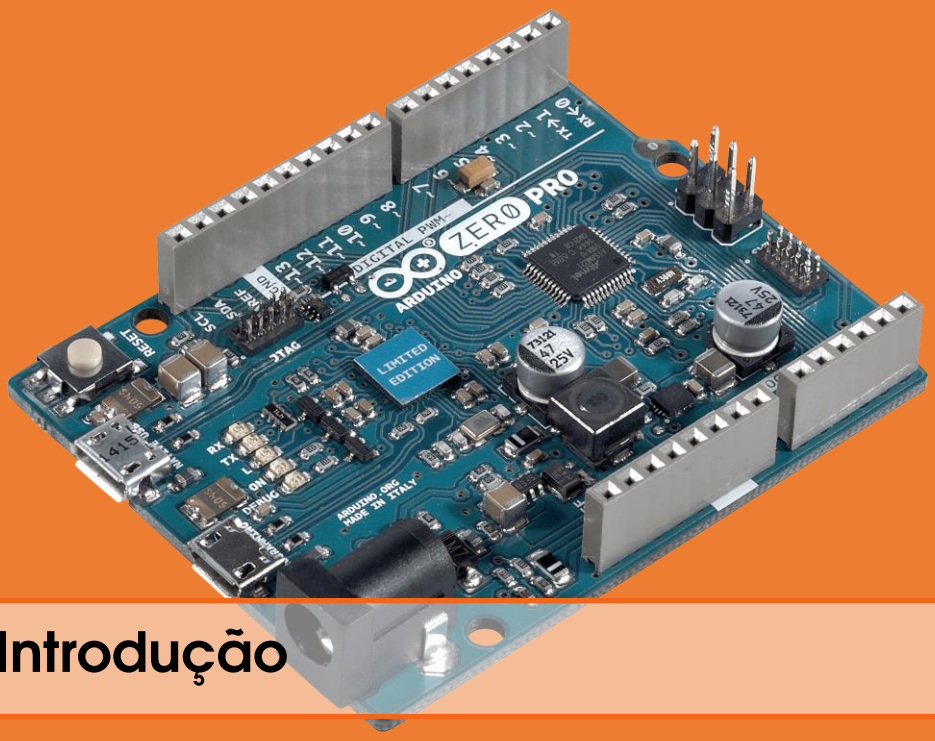
Ji-Paraná/RO, setembro de 2017.
Marco Polo Moreno de Souza.

Prefácio da segunda edição

Essa apostila, melhorada a partir de uma versão de 2017, foi criada para servir de suporte ao projeto "Arduino para Meninas", uma iniciativa financiada pelo CNPq voltada ao estímulo das meninas e mulheres para o ingresso na área das ciências exatas, engenharias e computação. Para mais informações sobre o projeto, acesse `www.arduino.unir.br`.

Para dúvidas, críticas e sugestões à respeito desta apostila, envie um e-mail para marcopolo@unir.br.

Ji-Paraná, RO, 08 de fevereiro de 2019.
Marco Polo Moreno de Souza.



1. Introdução

1.1 O que é Arduino?

Quando falamos em “Arduino”, podemos estar nos referindo ao hardware Arduino (isto é, seus vários modelos de placas), ao software Arduino (isto é, o ambiente onde desenvolvemos o código fonte e também a linguagem Arduino) ou, ainda à ideia Arduino (isto é, a de ser uma plataforma de prototipagem eletrônica de baixo custo e de fácil uso). A definição mais geral de Arduino engloba esses três aspectos.

1.1.1 O hardware

Hoje há uma variedade enorme de modelos de placas, sendo o Arduino Uno R3 o modelo mais famoso (Fig. 1.1). É por essa placa que quase todos entram no mundo Arduino. Há placas maiores, com maior poder de processamento, maior capacidade de armazenar dados e mais pinos de entrada e saída de dados, como o Arduino Mega 2560 e, ainda, há aquelas placas minúsculas, criadas para atuarem em projetos pequenos, onde espaço ou peso são fundamentais, como o Arduino Nano. Há também placas para propósitos específicos, como IoT (Internet das Coisas) e eletrônica *wireless* (sem fio), como as placas Arduino Yún e a Lilypad Arduino Simple, respetivamente (ver Fig. 1.2).

Na Seção 4.1 entraremos em detalhes no hardware do Arduino Uno R3.

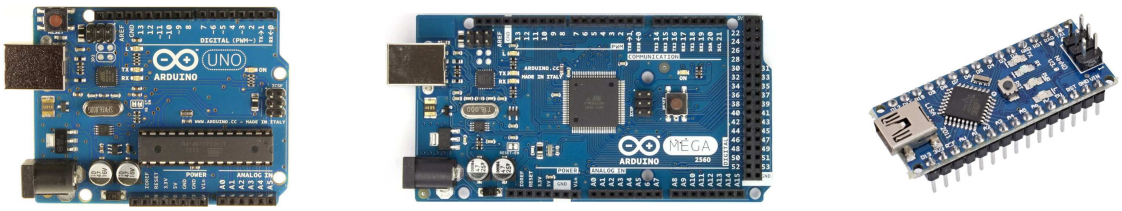


Figura 1.1: As placas Arduino Uno R3, Arduino Mega 2560 e Arduino Nano.

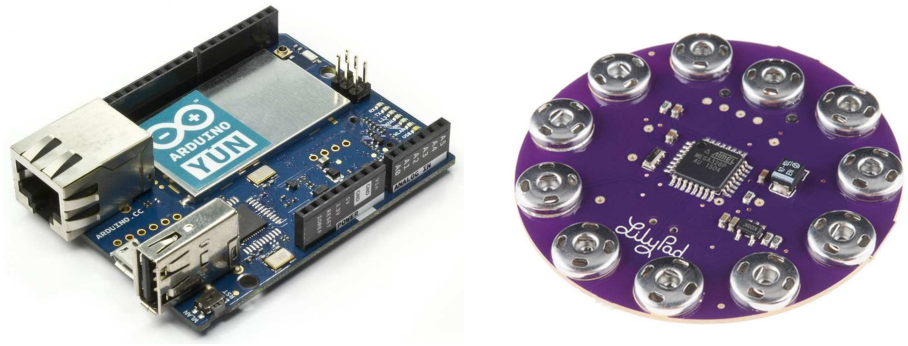


Figura 1.2: As placas Arduino Yún e Lilypad Arduino Simple.

1.1.2 O software

É através de um IDE (ambiente de desenvolvimento integrado, Fig. 1.3) que programamos as placas de Arduino: as variáveis são declaradas, as instruções são inseridas dentro das funções `setup()` e `loop()`, a eletrônica é implementada, o hardware é conectado e, bingo, seu sensor de luminosidade começa a funcionar. A programação é, para a maior parte das pessoas, a parte mais difícil para os iniciantes em Arduino. A programação é um conjunto de instruções para fazer algo, como calcular o número π com 20 casas decimais, resolver um sistema de equações, permitir o acesso de um cliente a um servidor remoto ou monitorar a quantidade de vagas disponíveis em um estacionamento de um shopping, por exemplo. No Arduino, esse conjunto de instruções se chama "sketch". Em outros contextos, são simplesmente chamados de código-fonte, ou ainda programa.

Uma breve introdução sobre a linguagem do Arduino pode ser encontrada no Capítulo 5.2; sobre a IDE, veja a Seção 4.2.

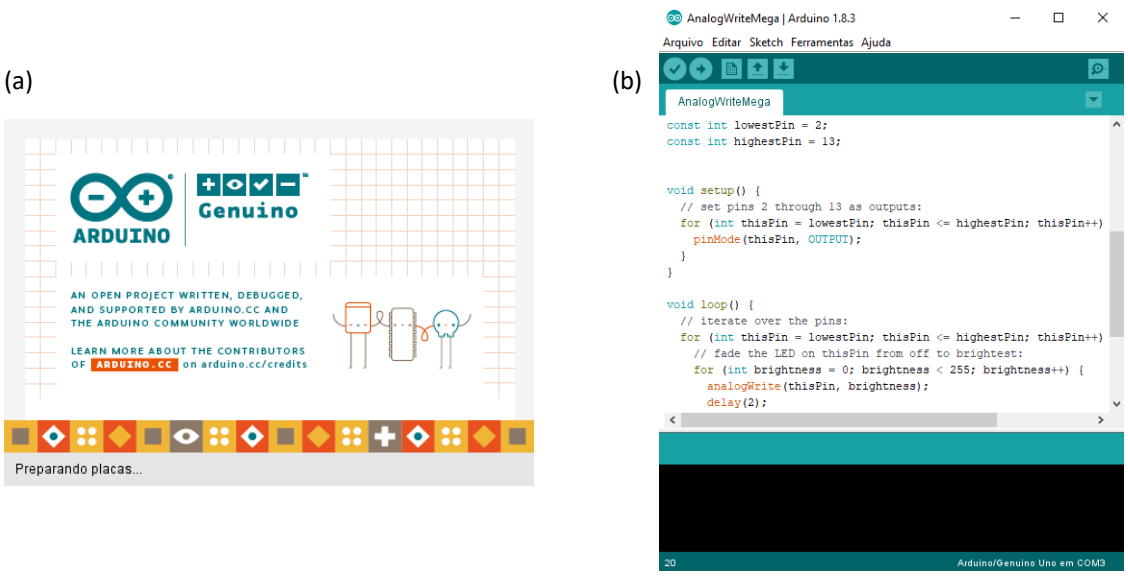


Figura 1.3: (a) Iniciando a IDE do Arduino no Windows 10. (b) Sketch aberta com o código "AnalogWriteMega", exemplo contido no IDE.

1.1.3 A ideia

O Arduino foi concebido para ser duas coisas: barato e fácil, em pé de igualdade em relação de importância. Ambos os aspectos levaram a uma rápida disseminação após 2005, quando a primeira placa foi produzida na Itália [**arduino-cc**]. Atualmente a placa mais famosa, a Arduino Uno R3,

pode ser adquirida no Mercado Livre por 40 reais, e um kit básico, contendo, além da placa, leds, resistores, uma protoboard, jumpers, um relé, um servo-motor e um potenciômetro, gira em torno de 120 reais. Há inclusive diversas outras placas que não são Arduino na forma mais abrangente da palavra, porém incorporam os aspectos da simplicidade e, principalmente, do baixo custo. Dentre algumas, podemos citar os microcontroladores ESP32 e ESP8266 (Seção 11.2) e o Raspberry Pi 3 B+ (Seção 11.1, Fig. 1.4), este último um computador completo que não apenas possui processador quad-core com clock em 1,4 GHz, 1 Gb de RAM, wifi, bluetooth, como também é capaz de rodar diversos sistemas operacionais, como o Raspbian (desenhado para as particularidades dessa placa), o Linux Ubuntu e o Windows 10 IoT Core (versão desse Windows para sistemas embarcados), por exemplo.

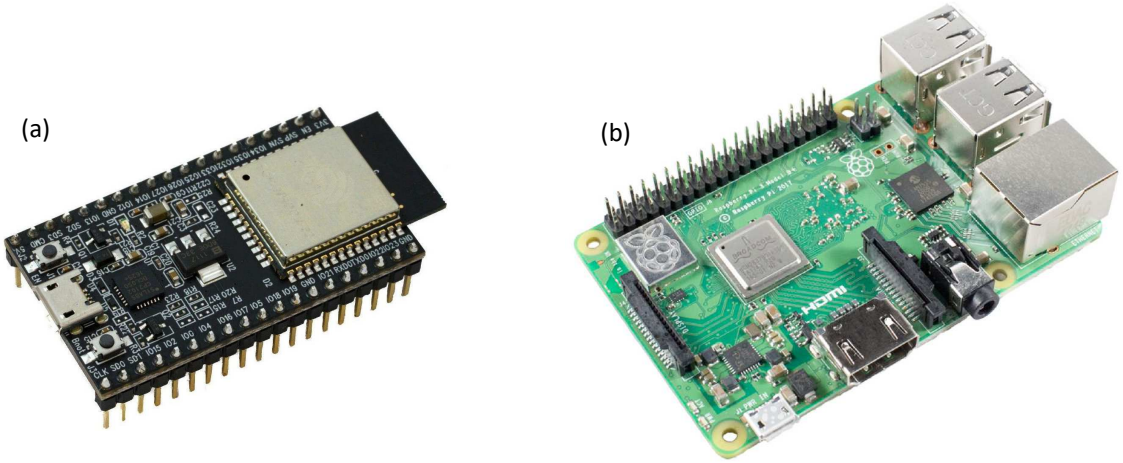


Figura 1.4: (a) Microcontrolador Esp32. (b) Raspberry Pi 3, modelo B+, lançado em 2018.

A primeira placa de Arduino foi desenhada para ajudar estudantes de Design que, em geral, não possuem conhecimentos básicos de eletrônica e programação, para criarem os mais diversos protótipos. Esse conceito foi extrapolado para fora do ambiente educacional; hoje em dia, é bastante comum encontrar o Arduino em laboratórios de pesquisa e na indústria como, por exemplo, para controlar a frequência de lasers de diodo em armadilhas de átomos frios e em impressoras 3D.

1.2 Como o Arduino funciona?

As placas de Arduino possuem microcontroladores capazes de processar informações. Assim, é interessante lembrar que todo o processamento é realizado no microcontrolador da placa, por mais que o código fonte seja digitado na IDE rodando em um notebook com Windows conectado via USB com a placa, por exemplo. Então a velocidade do processamento estará limitada à memória RAM, à memória flash e ao processador de cada tipo de placa. Uma vez que o código foi carregado para a placa, lá ele permanece, à princípio, indefinidamente. A partir desse momento um computador já não é mais necessário, a menos que seja necessário carregar outro código na placa, que é escrito sobre o anterior, ou que você não tenha uma fonte de energia própria, que deve fornecer tensão entre 7 e 12 V. Um pouco mais de detalhe sobre esse processo por ser encontrado na Seção 5.2.

A criação de um projeto Arduino envolve, de forma simplificada, os seguintes passos:

1. Conexão do computador com a placa Arduino via porta USB.
2. Criação do código fonte. Corresponde à digitação das instruções na IDE do Arduino.
3. Conexão dos componentes eletrônicos às portas de entrada e saída de dados da placa.

4. Verificação de erros e carregamento do código na placa.

Um exemplo de criação de um projeto simples pode ser encontrado na Seção 3.

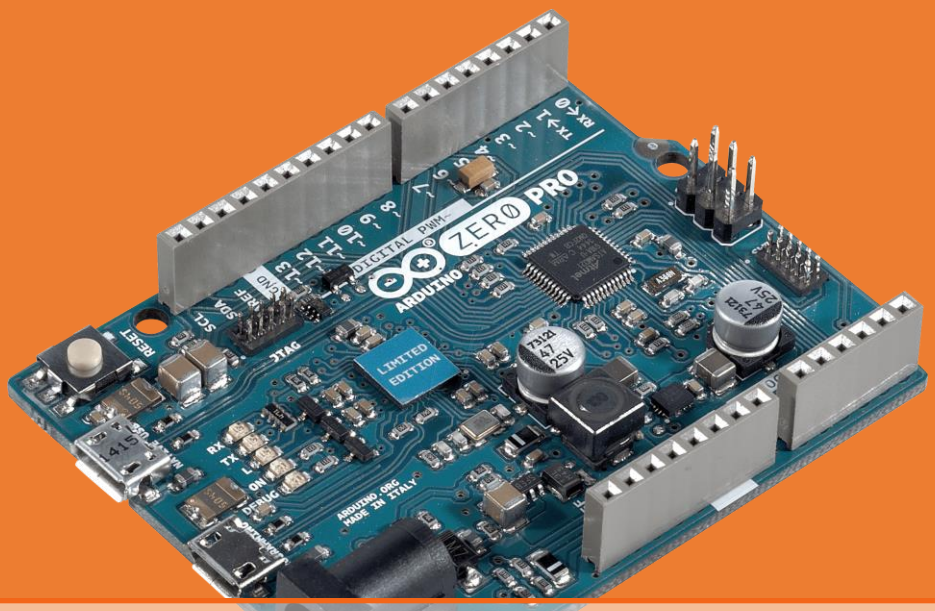
1.3 Aplicações

Como a aplicação do Arduino é vasta, esta seção não chega nem perto de esgotar todas as possibilidades. Listamos abaixo algumas:

- Impressão 3D.
- Automação residencial.
- Robótica.
- Controle de experimentos em laboratórios de pesquisa.
- Entretenimento.

1.4 Link para consulta

Site oficial do projeto Arduino: `www.arduino.cc`. É uma das fonte de consulta mais completa.



2. Instalação e configuração

Aqui mostraremos a instalação, a configuração e as principais funcionalidades do IDE do Arduino, primeiramente com base no ambiente do Microsoft Windows 10. Na Seção 2.3 apresentamos a instalação no Linux Mint 19.1, Cinnamon.

2.1 Download e instalação no Windows

O ambiente onde programamos o Arduino (IDE - *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) é um software livre que pode ser baixado no site `https://www.arduino.cc/en/Main/Software`. Atualmente ele está na versão 1.8.3. Para instalá-lo no seu computador, basta clicar no executável baixado (`arduino-1.8.3-windows.exe`) e proceder como na instalação de qualquer outro software.

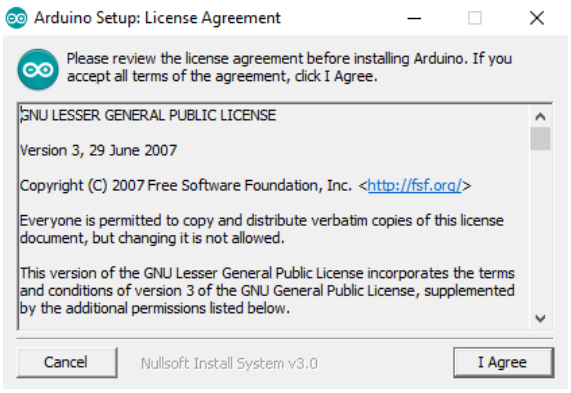


Figura 2.1: Instalação da IDE do Arduino.

Em seguida, será perguntado se você deseja instalar as portas `COM` e `LPT`, e o Driver do Arduino. Clique em `Instalar` nas três vezes. Após isso, a instalação estará completa.

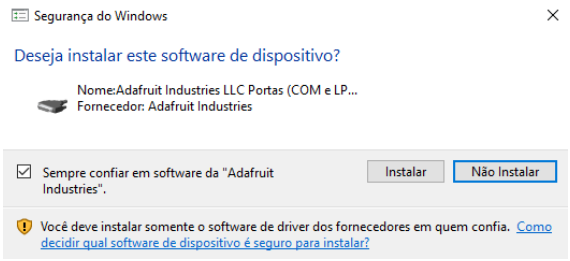


Figura 2.2: Instalação das portas `COM` e `LPT`.

2.2 Configuração

Agora vem a configuração, que é uma etapa importante. Primeiramente, clique no ícone do Arduino que foi criado na área de trabalho. Se o Firewall do Windows bloquear algum recurso, será mostrado um alerta de segurança. Nesse caso, clique em `Permitir acesso`. Finalmente, aparecerá uma janela com a IDE do Arduino. Muito provavelmente o software estará no idioma do seu Windows. Observe que o sketch já vem com um esqueleto de um código a ser criado, com as funções `setup()` e `loop()` e alguns comentários em inglês. Falaremos sobre as funções na linguagem do Arduino na Seção. 5.9.

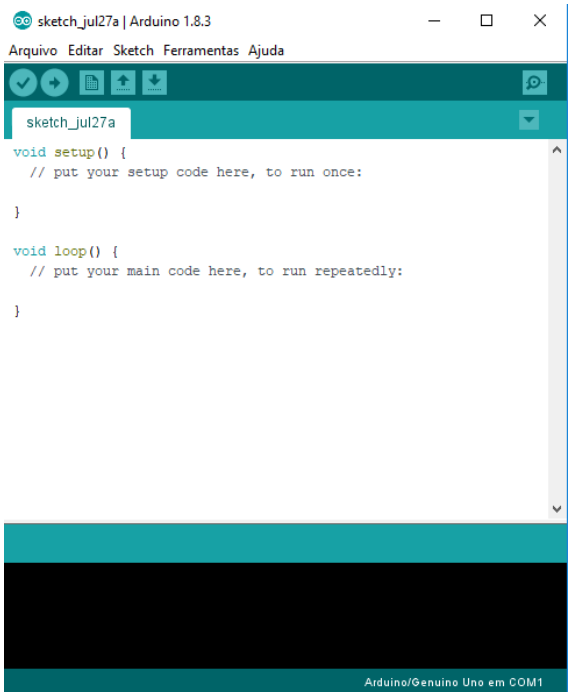


Figura 2.3: IDE do Arduino.

Agora você deve escolher a placa do Arduino que você vai trabalhar. Para isso, clique em `Ferramentas` ⟶ `Placa:` ``Arduino/Genuino Uno'' e faça a escolha apropriada.

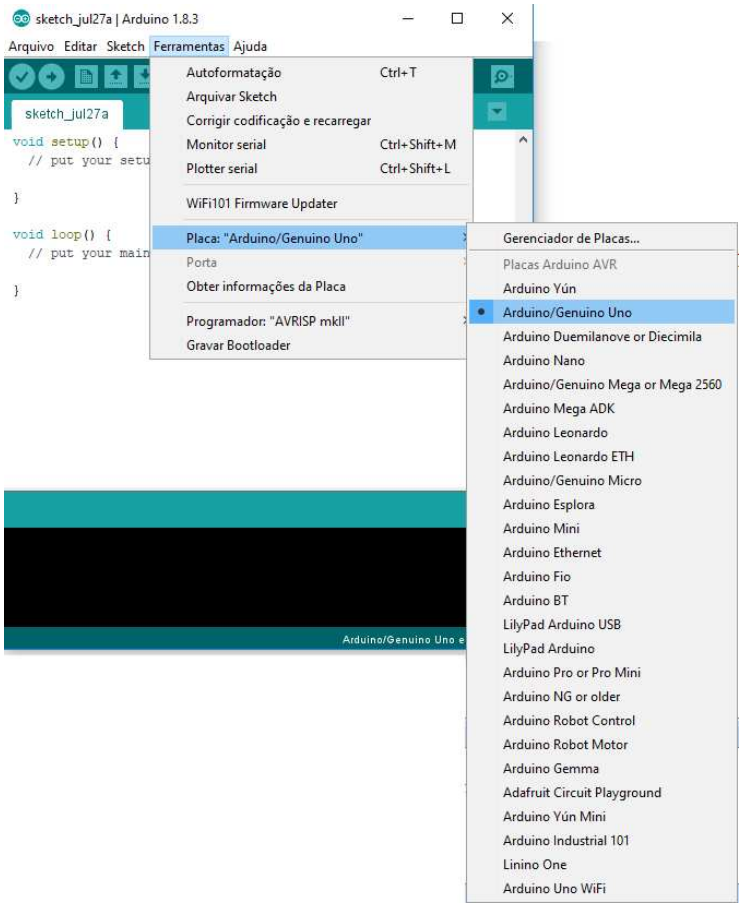


Figura 2.4: Escolhendo a placa a ser usada. Nesse caso, temos apenas a `COM4` disponível.

A próxima escolha diz respeito à porta USB do computador. Para isso, plugue o cabo USB da sua placa em uma porta USB do computador. Em seguida, clique em `Ferramentas` ⟶ `Porta` e selecione uma das portas disponíveis.

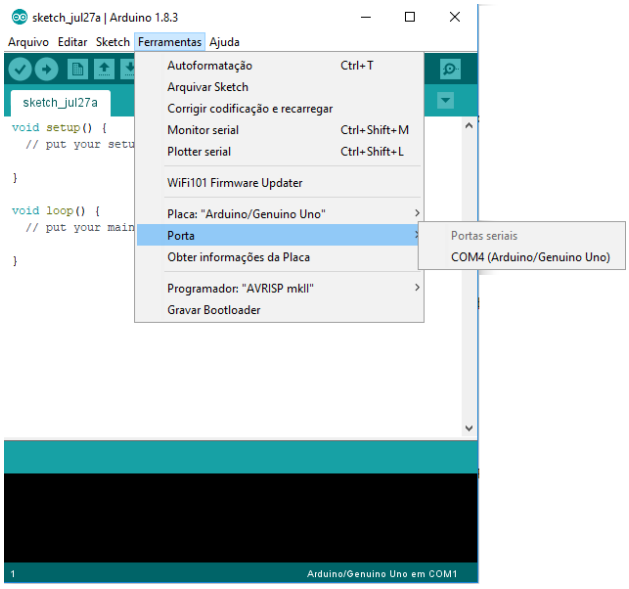


Figura 2.5: Escolhendo a porta a ser usada.

Por fim, vamos testar a comunicação entre o Arduino e o computador usando um código já pronto. Clique em `Arquivo` ⟶ `Exemplos` ⟶ `01.Basics` ⟶ `Blink`.

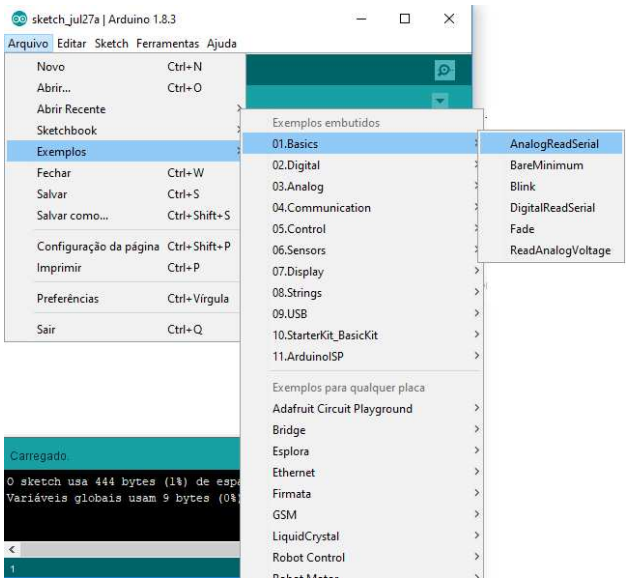


Figura 2.6: Abrindo o código `Blink`.

Em seguida clique em `Carregar`. Se tudo der certo, será possível ver uma pequena luz amarela acendendo e apagando na placa do Arduino.

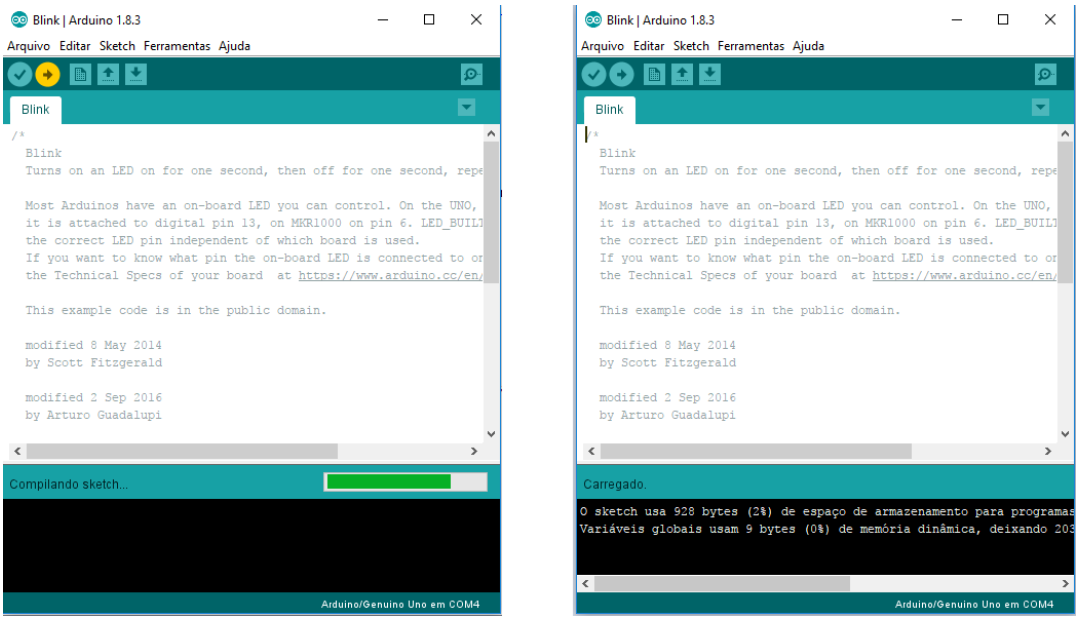


Figura 2.7: Do lado esquerdo, temos o código sendo carregado. O processo costuma ser mais rápido em computadores com SSD e mais lentos naqueles que ainda usam HDs. Do lado direito vemos o código carregado com sucesso.

2.3 Instalação no Linux Mint

É possível instalar através do comando `sudo-apt-get-install` no terminal, mas a versão do repositório está muito desatualizada (versão 1.0.5). Assim, recomendamos baixar o software no endereço `https://www.arduino.cc/en/Main/Software` e fazer a instalação pelos passos abaixo:

- Após baixar, descompacte o arquivo. Observe, pela Fig. 2.8, que estamos usando a versão 1.8.8 do Arduino.

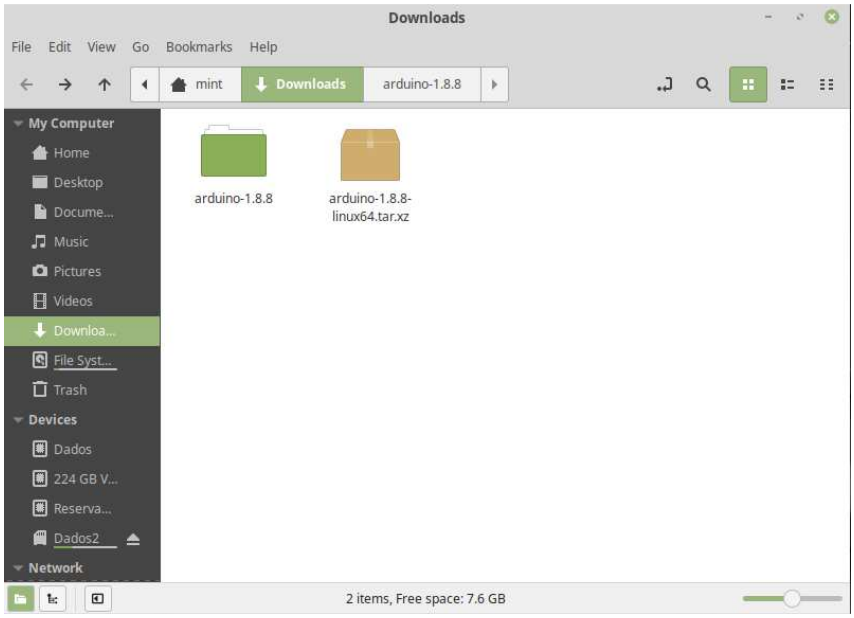


Figura 2.8: Arquivo baixado e pasta descompactada.

- Em seguida, acesse a pasta criada (em geral, `/Downloads/arduino-1.8.8`), abra o terminal e digite `./install.sh`.
- Seu IDE já estará instalado, e um ícone do programa estará na área de trabalho. Conecte o cabo USB da placa no computador, abra o IDE do Arduino e no menu `Ferramentas`, na opção `Placa:`, escolha a versão correspondente à sua placa.
- Ainda no menu `Ferramentas`, na opção `Porta`, escolha uma das portas disponíveis.
- Agora vamos testar um código exemplo. Clique no menu `Arquivo` → `Exemplos` → `01.Basics` → `Blink`.
- Por fim, clique no botão `Carregar`. Será possível ver uma pequena luz na placa acender e apagar em intervalos de um segundo.
- Pode acontecer de haver um erro de permissão negada: `can´t open device"/dev/ttyAM0": permission denied`. Nesse caso, com o terminal aberto, digite o `comando sudo chmod a+rw/dev/ttyACM0` e, em seguida, tente carregar o código novamente.

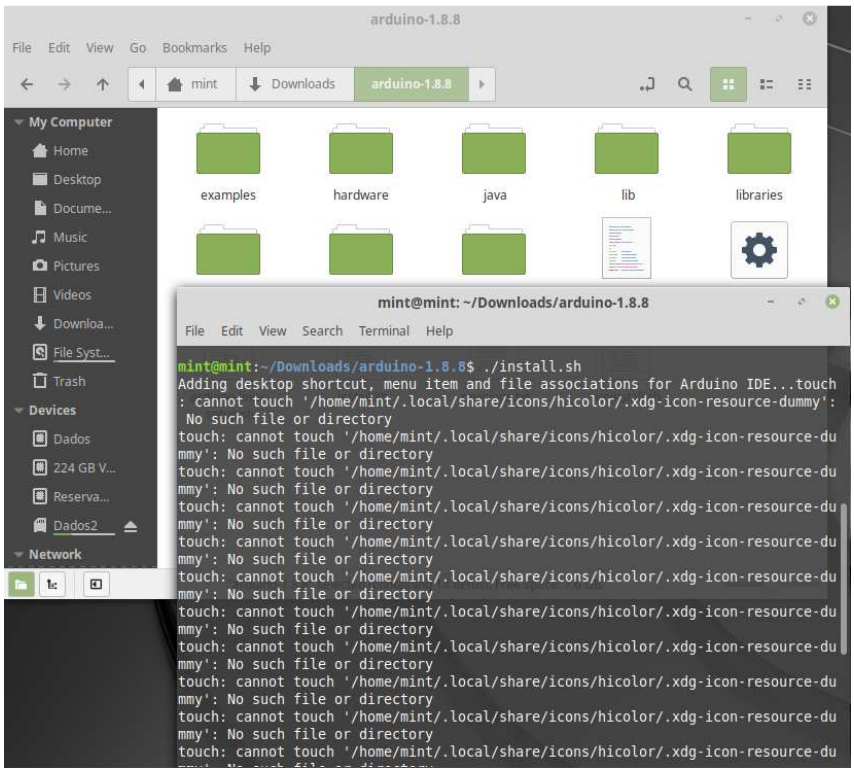


Figura 2.9: Instalação da IDE do Arduino.

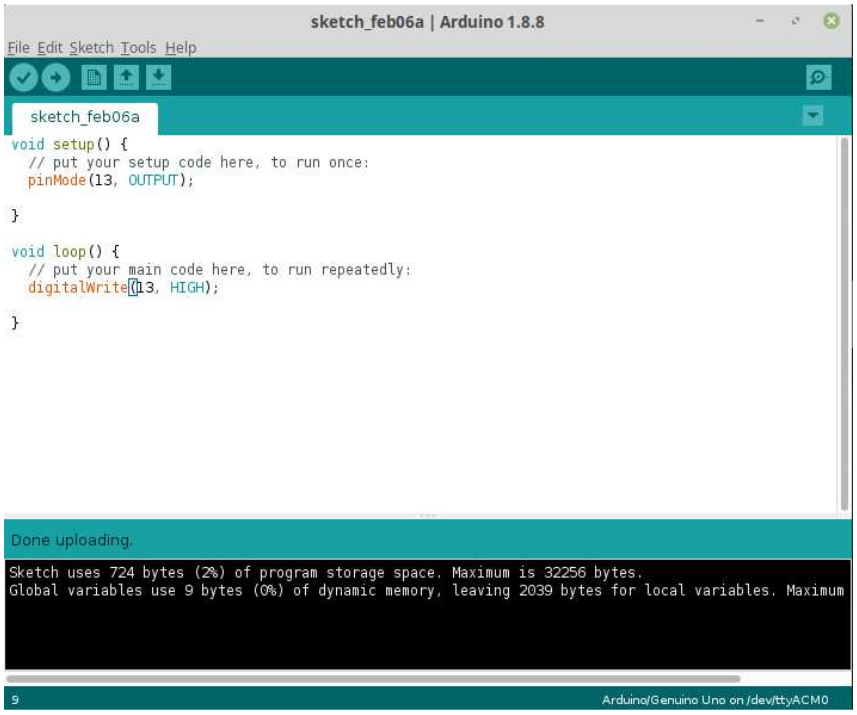
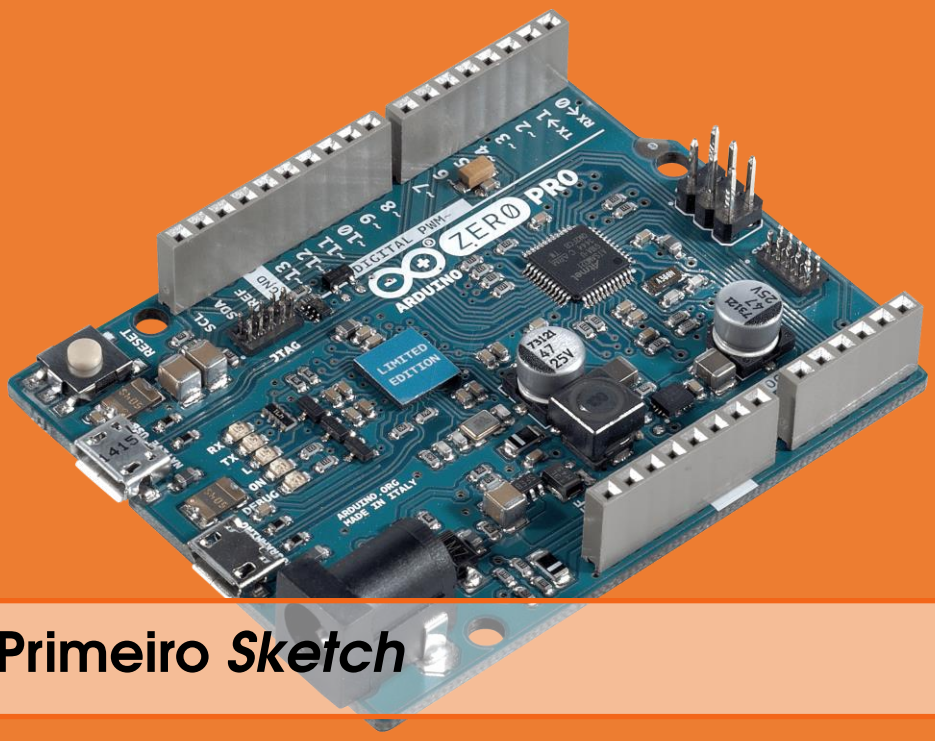


Figura 2.10: IDE aberto.



3. Primeiro *Sketch*

Nosso primeiro Skecht será extremamente simples: vamos fazer um LED acender e apagar e um intervalo de 1 segundo. Para isso, execute o procedimento abaixo:

- Abra a IDE do Arduino.
- Digite o código abaixo no Sketch, exatamente como está escrito. Nos primeiros contatos com o Arduino, é importante manter a prática de digitar ao invés de copiar a colar o código, para que seja adquirida a familiaridade com as funções mais importantes. Observe que a numeração à esquerda serve para identificar a linha do código, e não deve ser digitada.

```
1   //Primeiro código!
2
3   void setup() {
4     pinMode(13, OUTPUT);
5   }
6
7   void loop() {
8     digitalWrite(13, HIGH);
9     delay(1000);
10    digitalWrite(13, LOW);
11    delay(1000);
12  }
13
14  /*
15  Fim do Código
16  */
```

- Plugue os terminais do LED nas portas `13` e `GND` do Arduino. Observe que os LEDs possuem duas pernas com tamanhos diferentes. A perna maior geralmente é o positivo, enquanto que a menor é o negativo. Assim, conecte a *perna maior* na porta `13` (que fornece 5 V, de forma

que podemos entender como o terminal positivo) e a *perna menor* na porta `GND` (que é o terra, subtendida como o terminal negativo).

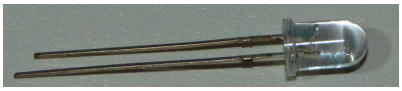


Figura 3.1: LED branco de alto brilho.

- Plugue o Arduino na entrada USB do computador e clique no botão `carregar` da IDE do Arduino.

Caso não haja nenhum erro de digitação, o código será carregado para a memória do Arduino e, logo após, o LED acenderá e apagará em intervalos de 1 segundo.

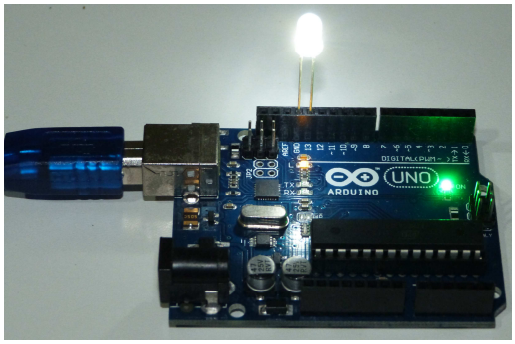


Figura 3.2: Arduino carregado com o código acima.

3.1 Sobre o código

Agora vamos tentar entender um pouco sobre o código listado na página anterior. O Arduino trabalha com uma linguagem de programação que pode ser considerada um dialeto da linguagem C, isto é, sua sintaxe obedece exatamente esta última linguagem, porém possuindo funcionalidades adicionais, o que inclui novas funções e novas bibliotecas. Além disso, algumas funções da linguagem C podem não funcionar na linguagem do Arduino, que poderá possuir uma versão alternativa para a mesma função. Um exemplo é a função `printf()` da linguagem C, que na linguagem do Arduino vai aparecer como `Serial.print()`. Tirando essas particularidades, as duas linguagens são essencialmente idênticas, o que significa que a forma de declarar variáveis, bibliotecas, funções é exatamente a mesma. Quem já possui noções de programação em C levará pouquíssimo tempo para se acostumar à linguagem do Arduino.

Agora vamos analisar cada linha de comando do código.

- Linha `1`. Aqui temos um comentário, uma funcionalidade que aparece na maioria das linguagens de programação. A função do comentário é deixar o código claro tanto para outras pessoas como para o próprio desenvolvedor. É bastante comum um programador não se lembrar o que um determinado bloco de instruções faz após alguns meses, mesmo entre os experientes. Nesse sentido, o comentário serve como um lembrete. Outro uso é como um cabeçalho, para indicar o autor e a data de desenvolvimento do código. Observe que a IDE do Arduino ignora todas as linhas comentadas no código. Para comentar uma linha, adicionamos duas barras no começo da linha. Também é possível comentar várias linhas com os comandos $/*$ e $*/$, como mostrado no final do código.

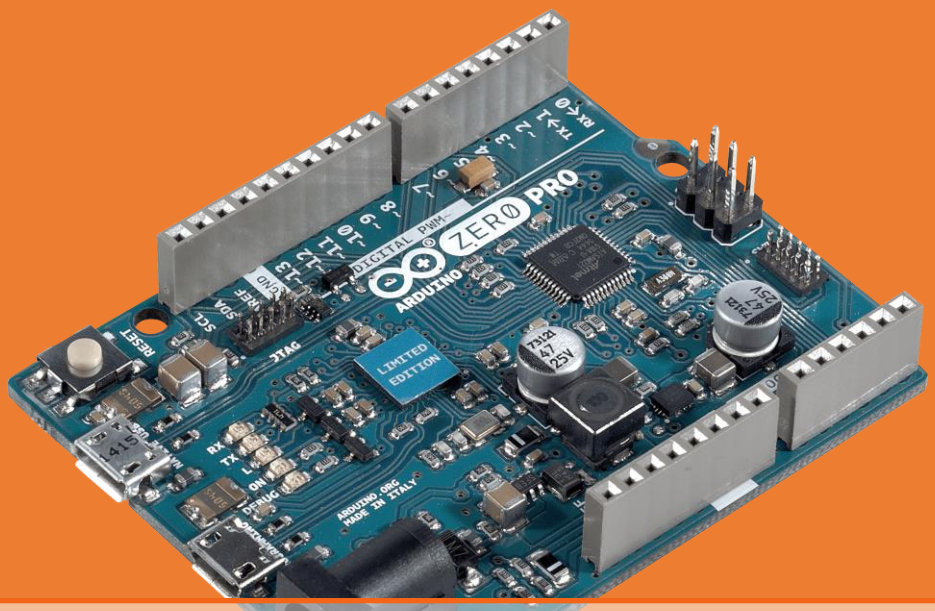
- Linha `3`. Aqui temos a função `setup()`. O comando `void` na frente da função significa que essa função não retorna valores (na linguagem Pascal, por exemplo, o `setup()` seria na verdade um "procedimento", que é semelhante a uma função, mas que não retorna valores). Por hora, podemos dizer que a função `setup()` executa um conjunto de instruções (tudo o que está entre as chaves - linha `4`) *uma única vez*. Mais sobre funções pode ser lido na Seção 5.9.

- Linha `4`. A função `pinMode()` habilita uma porta do Arduino para entrada ou saída de dados. Nesse caso, estamos habilitando a porta `13` para saída de tensão.

- Linha `7`. A função `loop()` também é do tipo `void`. A diferença é que ela executa as instruções dentro das chaves (linhas `8` - `11`) *infinitas vezes*.

- Linha `8`. A função `digitalWrite()` gera uma tensão em uma porta específica do Arduino. Essa tensão pode ser apenas 5 V (`HIGH`) ou 0 (`LOW`). Para tensões entre 0 e 5 V, é necessário um *shield* específico. Observe que primeiro precisamos habilitar a porta com o comando `pinMode`. Nesse projeto, o objetivo da função `digitalWrite()` é gerar uma diferença de potencial de 5 V entre os terminais do LED, de forma a acendê-lo.

- Linha `9`. A função `delay(n)` força uma pausa de `n` milissegundos na execução das linhas de comando. Aqui, fizemos uso dela para que, após 1000 milissegundos (1 segundo) aceso, o LED apague com a instrução da linha `10`: `digitalWrite(13, LOW);`. Sem o uso da função `delay()` nas linhas `9` e `11`, o LED iria acender e apagar em uma frequência tão alta (em uma frequência próxima do *clock* da placa) que seria impossível ver a luz piscar.



4. Arduino: hardware e software

Neste capítulo vamos falar sobre as principais características da placa do Arduino bem como do seu IDE (ambiente de desenvolvimento integrado).

4.1 A placa do Arduino Uno R3

Nessa seção falaremos sobre a placa do Arduino Uno R3, uma das mais populares, sendo considerada uma placa de entrada no mundo do Arduino. As principais partes dessa placa são:

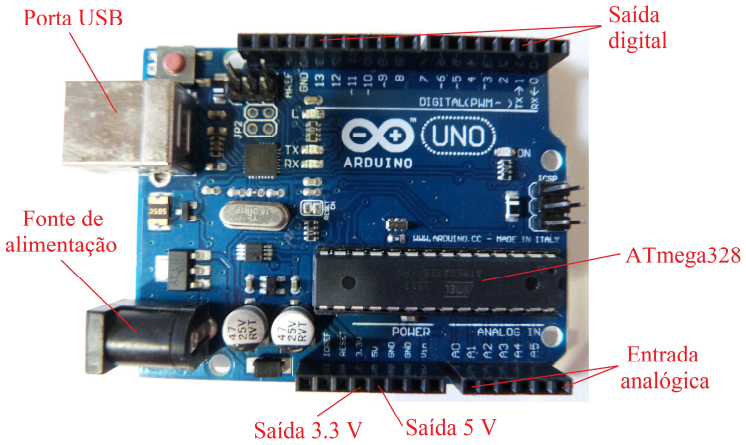


Figura 4.1: Placa do Arduino UNO.

- **Microcontrolador ATmega328**. É o cérebro da placa, onde todo o processamento de dados é feito. Ele é basicamente uma CPU com clock de 20 MHz, 8 bits e 32 kB de memória flash. Além disso, é nesse controlador que os programas são armazenados.
- **Porta USB**. É por onde a comunicação com o computador é feita, e que também serve como fonte de alimentação da placa.

- **Fonte de alimentação**. Usada quando desejamos operar a placa sem um computador. Naturalmente, um computador é necessário em um primeiro momento para colocar o programa dentro da placa.

- **Pinos**. Essa placa possui uma série de pinos, que podem servir tanto para entrada e saída de dados como também como uma fonte de tensão fixa. Os pinos numerados como `A0`, `A1`, `A2`, `A3`, `A4` e `A5` são de entrada analógica. Os pinos denotados por `GND` são o terra e os pinos de `1` a `13` são para saída da tensão. Desses, alguns possuem também o símbolo $\sim$, que representam uma saída do tipo PWM (modulação por largura de pulso), que seria uma forma de simular uma saída analógica.

4.2 A IDE do Arduino

Aqui vamos fazer uma rápida passagem sobre as funcionalidades do ambiente de desenvolvimento integrado do Arduino, ou apenas o software do Arduino.

Na figura abaixo mostramos todos os botões da IDE.

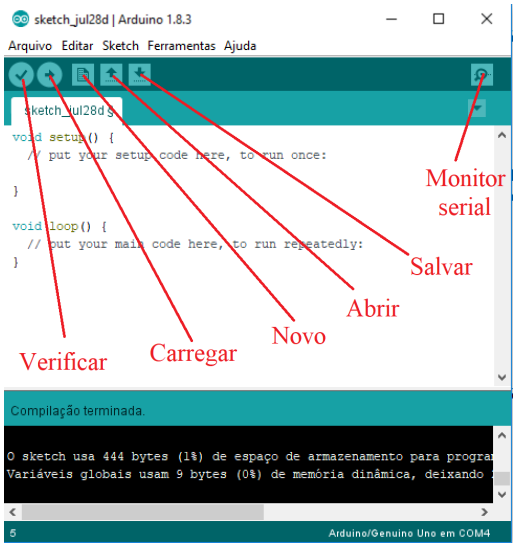


Figura 4.2: Botões da IDE do Arduino.

- **Botão `Verificar`**. Compila o sketch, verificando os erros de sintaxe do seu código. Na compilação, a IDE do Arduino traduz o código escrito em linguagem C para a linguagem de máquina (sequência de bits - código binário), preparando para o armazenamento na placa do Arduino.

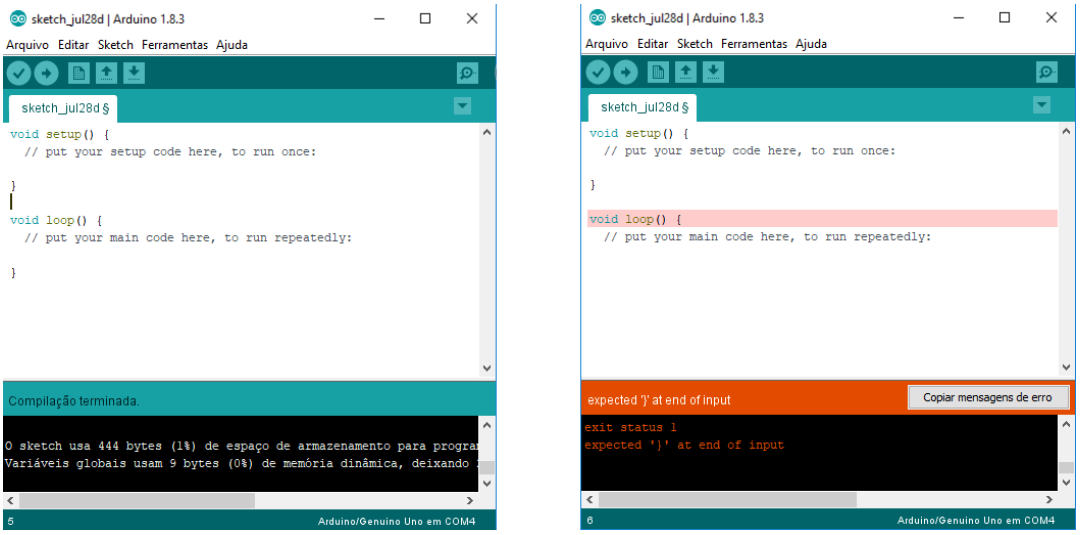


Figura 4.3: Códigos após um clique no botão `Verificar`. Na esquerda vemos uma compilação finalizada com sucesso, enquanto que na direita o IDE retornou uma mensagem de erro, que foi causada pela ausência de uma chave fechando a função `loop()`.

- **Botão `Carregar`**. Compila e, em caso de ausência de erros de sintaxe, armazena o código na placa do Arduino. Em resumo, esse botão executa os mesmos procedimentos do botão `Verificar` e ainda armazena o código binário no Arduino, o que faz com que a placa "rode" o seu código logo em seguida.
- **Botão `Novo`**. Cria uma nova sketch.
- **Botão `Abrir`**. Abre uma sketch salva previamente.
- **Botão `Salvar`**. Salva a sketch.
- **Botão `Monitor serial`**. Abre o monitor serial. O Monitor serial é usado, por exemplo, para ler o valor armazenado em uma dada variável. Ele é um recurso extremamente importante para testar o código e procurar por erros de lógica. Observe que o Monitor serial só pode ser aberto quando a placa do Arduino está conectada no computador.

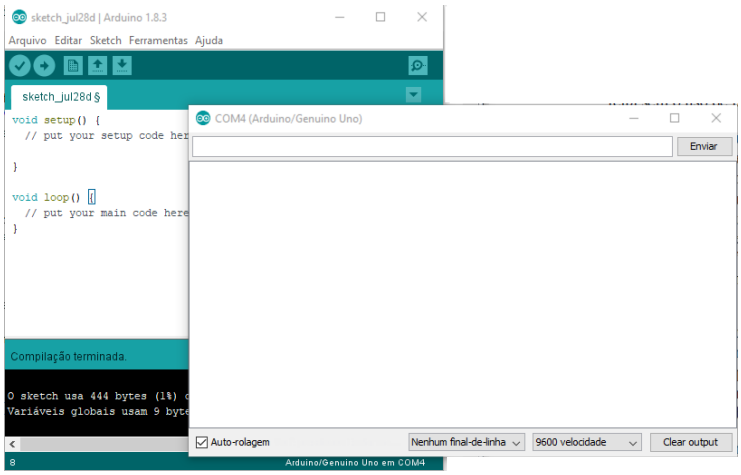
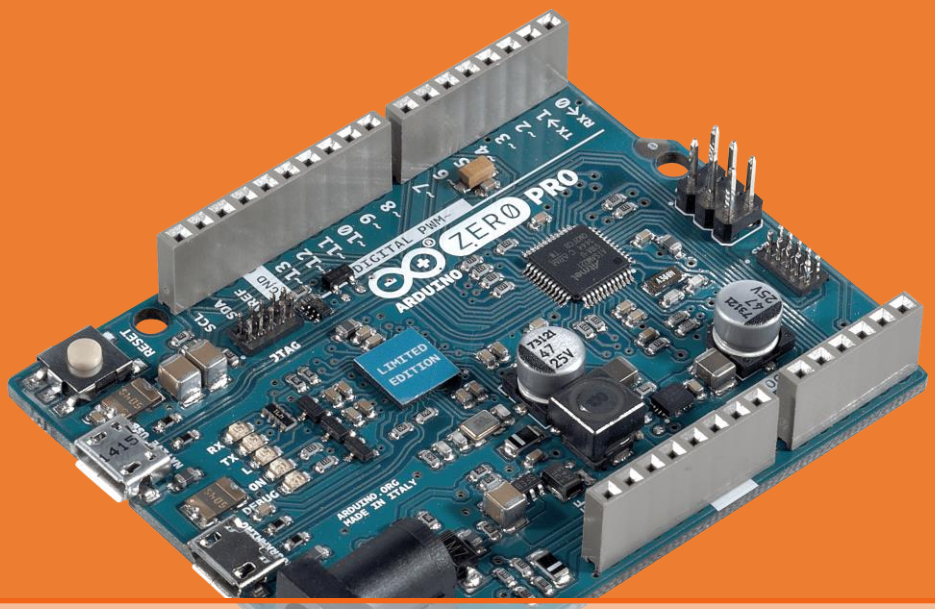


Figura 4.4: IDE com o Monitor serial aberto.



5. A Linguagem do Arduino

Nesse capítulo abordaremos os principais tópicos de uma linguagem de programação, como declaração e tipos de variáveis, estrutura de condição, estrutura de repetição, funções e bibliotecas, com ênfase na linguagem do Arduino.

5.1 Funcionamento de uma linguagem de programação

Uma linguagem de programação é uma linguagem formal (no sentido amplo do termo, contexto que inclui matemática, ciência da computação e linguística) onde uma determinada tarefa pode ser executada mediante o uso de um conjunto de instruções, que nessa apostila as vezes chamaremos de comandos. Por exemplo, se um cientista precisa saber a quantidade de elementos nulos de uma matriz muito grande, como uma 1000×1000, o que é uma tarefa árdua para ser feita sem o uso de um computador, ele poderá proceder da seguinte forma.

- **Criação de um algoritmo**. A primeira tarefa do cientista é pensar como um computador poderia contar a quantidade de zeros da matriz. Ele poderia decidir criar uma variável que iria percorrer, linha por linha, todos os elemento da matriz. Cada vez que a variável encontrasse um zero, por comparação por exemplo, ele poderia incrementar uma outra variável em uma unidade, criada especificamente para este fim. Essa etapa é a criação da "receita de bolo", que é o algoritmo. Um exemplo de algoritmo famoso é o "método dos trapézios", que consiste em realizar a integração numérica de uma função dividindo o intervalo em vários trapézios, calculando a área de cada um deles e realizando a soma total.

- **Criação do código**. Aqui, o cientista precisa implementar o algoritmo em alguma linguagem de programação. Então, ele deverá criar uma variável para ler a matriz (entrada de dados), seja diretamente do teclado ou de um arquivo externo, depois outra para contar a quantidade de zeros (processamento de dados) e, por fim, uma última para informar a quantidade de zeros (saída de dados). Observe que aqui temos três subetapas principais: entrada, processamento e saída de dados.

A escolha da linguagem geralmente é feita com base em uma das três dimensões: conhecimento do programador, dificuldade de implementação e velocidade de processamento. Se o problema é simples de ser implementado em uma dada linguagem que o programador conhece bem, ele naturalmente a escolherá, desde que o código seja executado rapidamente (por exemplo, em um minuto ou menos). Se o problema demanda muito tempo para ser executado, o programador deverá pesar se vale a pena uma implementação difícil em uma linguagem veloz. Por exemplo, sistemas algébricos computacionais, como o Maple e o Mathematica possuem uma linguagem de programação de fácil implementação, porque possuem muitos pacotes prontos voltados à solução de inúmeros problemas da matemática, da física e da engenharia. No Maple é possível resolver equações diferenciais numericamente com um único comando (`dsolve`). Em compensação, elas são extremamente lentas quando comparadas com linguagens como C e Fortran, cuja implementação é um pouco mais difícil e demorada. Por fim, temos como exemplo a linguagem CUDA-C, de implementação muito mais difícil que as duas anteriores, que ainda requer um hardware específico (uma placa de vídeo da Nvidia), mas que é extremamente veloz por realizar cálculos em paralelo no núcleos CUDA da placa.

A maior parte das linguagens de programação, como C, Fortran, Pascal, Java, PHP e Python, executa as linhas de comando em sequência, grosso modo, de cima para baixo. É assim que o iniciante em programação deve pensar na hora de escrever seu código.

5.2 A linguagem Arduino

A "linguagem" Arduino é, atualmente, uma das mais usadas no mundo, segundo o ranking publicado pela IEEE em julho de 2017 (*Institute of Electrical and Electronics Engineers*, ver figura abaixo). Entretanto, é consenso nos fóruns especializados que o Arduino não é uma linguagem propriamente dita, e sim um "dialeto" da linguagem C, contendo funções específicas para a programação no microcontrolador AVR.

De forma simplificada, o processo de compilação começa quando o IDE do Arduino faz pequenas alterações no sketch, criando um código em C++. Em seguida, o compilador avr-gcc converte o código de C++ para linguagem de máquina (ou código objeto), para em seguida armazená-lo na placa do Arduino via USB como um único arquivo Intel hexadecimal. Um pouco mais de detalhe sobre esse processo pode ser encontrado na Ref. [**github-wiki**].

Segundo o site oficial do Arduino (`www.arduino.cc`), os programas escritos em Arduino podem ser divididos em três partes: estrutura, variáveis e funções.

5.3 Estrutura

A estrutura básica de todo código Arduino é composta pelas funções `setup()` e `loop()`, como pode ser visto nas linhas de comando abaixo. A diferença entre essas funções é que todos os comandos localizados dentro da função `setup()` são executados apenas uma vez, ao passo que aqueles localizados dentro da função `loop()` são executados repetidamente, até que o programa seja encerrado. Por exemplo, uma instrução que faz a leitura contínua de dados em um experimento deve estar dentro de `loop()`, enquanto que uma declaração de variáveis deveria estar em `setup()`. Observe que um programa cuja variável seja declarada dentro de `loop()` ainda funcionará, porém uma eficiência menor, visto que haverá intermináveis redeclarações de variáveis desnecessárias. Abaixo mostramos os blocos de instruções dentro dessas duas funções.

```
1 void setup()
2 {
3  Comando a1;
4  Comando a2;
```

```
5    ...
6
7   }
8
9   void loop()
10  {
11   Comando b1;
12   Comando b2;
13   ...
14  }
```

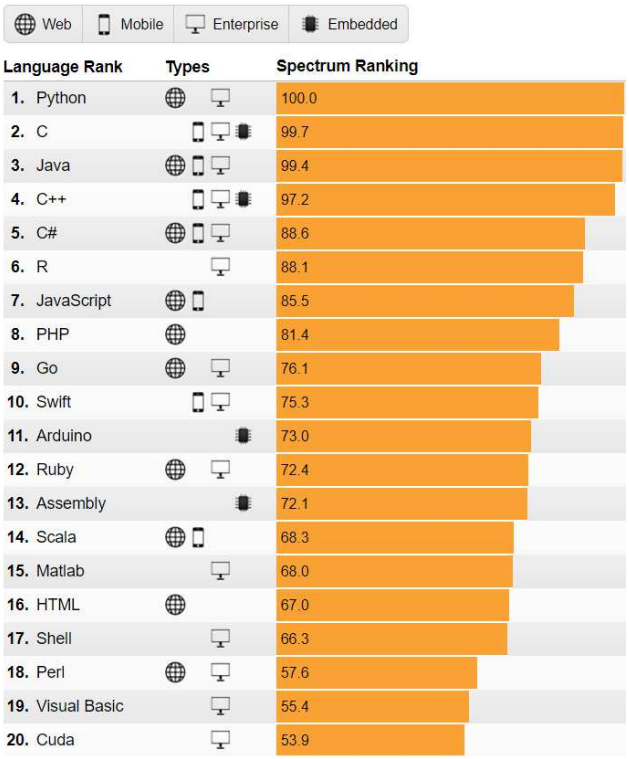


Figura 5.1: Ranking de uso das linguagens de programação em 2017, com base em pesquisa do IEEE.

5.4 Variáveis

As variáveis servem para armazenar um determinado tipo de dado, que pode ser um número ou caractere.

5.4.1 Variáveis numéricas

Os principais tipo de variáveis numéricas são `int`, `float` e `double`, como mostrado na tabela abaixo. De forma resumida, se estamos trabalhando com números inteiros, o melhor é usar a variável do tipo `int`. Se números com ponto são necessários (6.7, 3.14159, etc), podemos usar o `float`. Para uma melhor precisão, usa-se o `double`, porém o tempo de execução do programa aumenta, pois é necessária mais memória para armazenar os dados nesse tipo de variável. Abaixo mostramos os três tipos básicos de variáveis numéricas, os valores máximos e mínimos suportados e a precisão em quantidade de casas decimais.

Tipo	Característica	Alcance	Precisão
`int`	inteiros	de -32767 a 32767	-
`float`	reais	de -3.4×10^{38} a -3.4×10^{38}	6 dígitos
`double`	reais	de -1.7×10^{308} a -1.7×10^{308}	10 dígitos

5.4.2 Variáveis não-numéricas

Nesse caso, o tipo mais usado é o `char`, que armazena tanto letras como números e símbolos (%, $*$, &, etc). Esse tipo de variável armazena um único caractere; para mais, considere usar um vetor (veja a Seção 5.11.1).

5.4.3 Declaração de variáveis

A declaração de uma variável pode ser entendida pelo exemplo 5.1. Nesse caso, declaramos uma variável com nome **num** do tipo `int`, uma variável com nome **delta** do tipo `float` e uma variável de nome **H** do tipo `char`. Quando declaramos as variáveis antes da função `setup()`, elas são "variáveis globais", de forma que elas podem ser usadas tanto dentro da função `setup()` como dentro da função `loop()`. Variáveis declaradas dentro de uma função são "variáveis locais", de forma que elas não podem ser usadas fora dela.

■ Exemplo 5.1

```
1  int num;
2  float delta;
3  char H;
4
5  void setup()
6  {
7    ...
8  }
```

5.5 Entrada e saída de dados

O armazenamento de dados em uma variável é feito com o comando de atribuição, representado pelo símbolo $=$. Assim, quando escrevemos

```
1  num = 17;
```

estamos atribuindo o valor inteiro 17 à variável **num**, que deve ter sido previamente declarada. Uma declaração com uma atribuição em uma única linha pode ser realizada da seguinte forma:

```
1  int num = 17;
```

O símbolo $=$ tem o papel unicamente de uma atribuição. A igualdade matemática no sentido comparativo é representada pela dupla igualdade $==$ (ver Seção 5.6).

O valor de uma variável pode ser mostrado no monitor serial (saída de dados) com as funções `Serial.print()` ou `Serial.println()`. A diferença entre elas é que a última pula uma linha após a escrita do valor da variável no monitor serial.

Abaixo temos um código ilustrando o uso da função `Serial.println()`. Para o uso dessa função, é necessário o carregamento da função `Serial.begin()` dentro da função `setup()`.

O valor de 9600 se refere à taxa de transmissão de dados pela porta USB do computador. Observe que, como a chamada da função `Serial.println()` foi feita dentro da função `loop()`, o valor da variável `num` aparecerá no monitor serial até o programa ser finalizado. Observe também que o monitor serial pode ser usado apenas quando o programa for carregado na memória do Arduino, de forma que este deve estar conectado com o computador via porta USB.

```
1  int num = 17;
2
3  void setup()
4  {
5    Serial.begin(9600);
6  }
7
8  void loop()
9  {
10   Serial.println(num);
11   delay(2000);
12 }
```

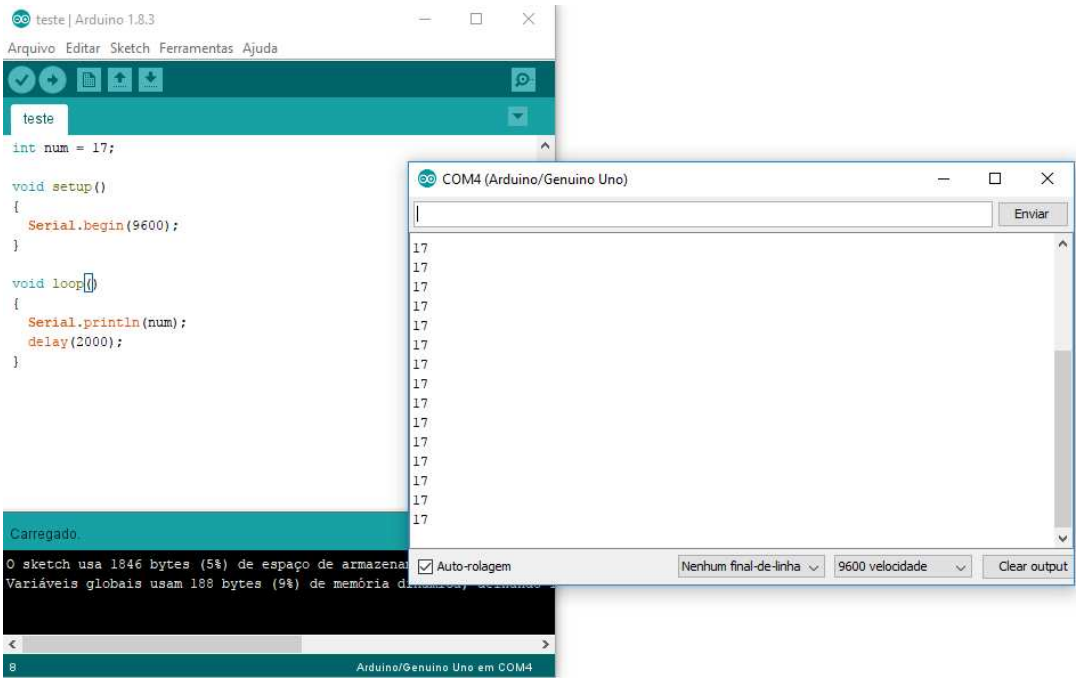


Figura 5.2: Monitor serial mostrando o valor armazenado na variável **num**.

Também é possível ler uma variável inserida no monitor serial. Isso é possível com a função `Serial.parseInt()`, que retorna o número inteiro digitado no monitor serial. O monitor serial fica em espera por até 1 segundo, e caso esse tempo seja ultrapassado, a função retorna o valor zero. Esse tempo de espera pode ser personalizado com a função `Serial.setTimeout()`, que faz o monitor ficar em espera pelo argumento em milissegundos.

■ **Exemplo 5.2** Abaixo temos um código que calcula o valor de $(a+b)^3$, onde a e b são inteiros digitados pelo usuário.

```
1  int a,b,resultado;
```

```
2
3   void setup()
4   {
5     Serial.begin(9600);
6     Serial.setTimeout(3000);
7
8     Serial.println("Este␣programa␣calcula␣o␣valor␣de␣(a␣+␣b)^3.");
9
10    Serial.print("a␣=␣");
11    a = Serial.parseInt();
12    Serial.println(a);
13
14    Serial.print("b␣=␣");
15    b = Serial.parseInt();
16    Serial.println(b);
17
18    resultado = (a+b)*(a+b)*(a+b);
19    Serial.print("(a␣+␣b)^3␣=␣");
20    Serial.print(resultado);
21  }
22
23  void loop()
24  {
25
26  }
```

5.6 Operações

Agora vamos dar início ao processamento de dados com as operações aritméticas. O exemplo abaixo é bastante ilustrativo.

■ Exemplo 5.3

```
1   int a;
2   int b;
3   int soma;
4   int produto;
5
6   void setup()
7   {
8     Serial.begin(9600);
9     a = 3;
10    b = 4;
11    soma = a + b;
12    produto = a * b;
13
14    Serial.println(soma);
15    Serial.println(produto);
16  }
17
```

```
18  void loop()
19  {
20
21  }
```

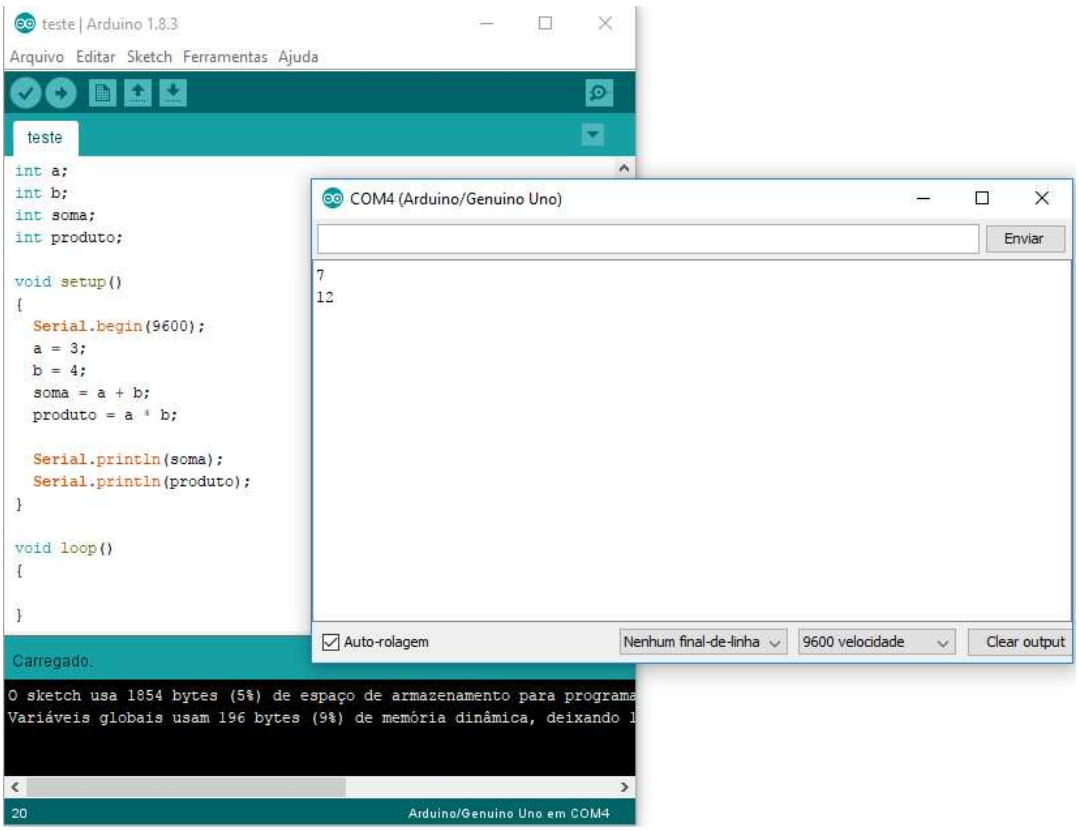


Figura 5.3: Execução do código acima.

Comentário do código:

- **Linhas 1 - 4**. Aqui são declaradas as variáveis **a**, **b**, **soma** e **produto**, todas do tipo `int`.
- **Linhas 9 - 10**. Atribuição do valor 3 à variável **a** e 4 à variável **b**.
- **Linhas 11 - 12**. À variável **soma** é atribuído o valor da soma de **a** com **b**, e à variável **produto** é atribuído o valor do produto de **a** com **b**. Essas duas linhas contém todo o processamento de dados desse código.
- **Linhas 14 - 15**. Por fim, é exibido no monitor serial o valor das variáveis **soma** e **produto**.

Observe que, agora, preferimos incluir a chamada da função `Serial.println()` dentro da função `setup()`, de forma que os valores das variáveis **soma** e **produto** aparecem uma única vez no monitor serial.

Lista resumida de operações aritméticas:

Operação	Descrição
+	adição
-	subtração
*	multiplicação
/	divisão
%	resto da divisão

Há, também, as operações de comparação e booleanas, listadas nas tabelas abaixo.

Operadores de comparação	
Operação	Descrição
==	igual
!=	diferente
<	menor
>	maior
<=	menor ou igual
>=	maior ou igual

Operadores booleanos	
Operação	Descrição
&&	e
||	ou
!	não

5.7 Estrutura de condição

A estrutura de condição permite obtermos uma quebra no fluxo de execução do programa. Isso pode ser necessário quando o programa executa tarefas diferentes dependendo do valor que uma ou mais variáveis possui. Um interessante exemplo é o da solução de uma equação do segundo grau. Se estivermos restritos ao corpo dos números reais, sabemos que algumas dessas equações não possuem solução.

Uma equação do segundo grau pode ser escrita como

$$ax^2 + bx + c = 0,$$

onde $a \neq 0$, b e c são números reais. Sabemos que, se $\Delta = b^2 - 4ac > 0$, a equação possui duas soluções; se $\Delta = 0$ a equação possui uma solução; e se $\Delta < 0$ não há solução real. Essas informações podem ser resumidas através do fluxograma da Fig. 5.4, onde é possível observar a quebra do fluxo em três caminhos possíveis.

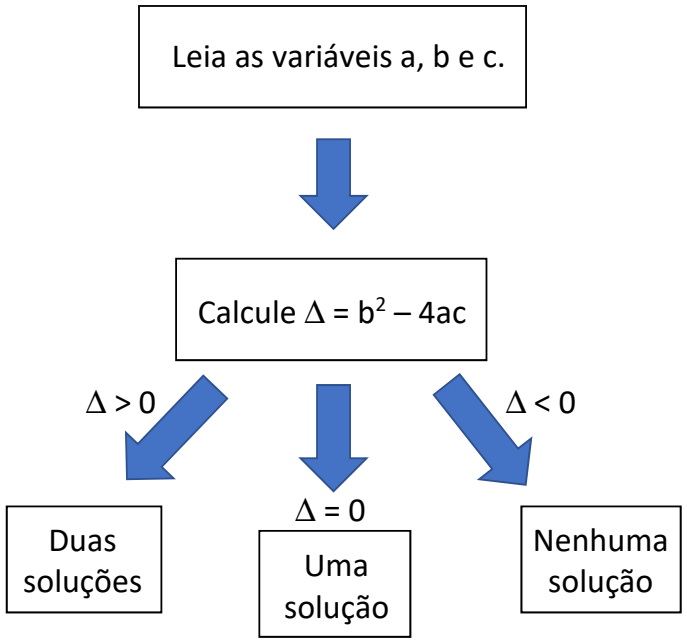


Figura 5.4: Fluxograma representando a quebra de caminho conforme o valor de Δ em uma equação do segundo grau.

Assim, nosso código deve ter uma instrução que determina o que o programa deve fazer dependendo do valor de Δ.

Uma estrutura de condição é formada pelos comandos `if-else`, cuja sintaxe é dada por

```
1  if (condição)
2  {
3   instrução a1;
4   instrução a2;
5   ...
6  }
7  else
8  {
9   instrução b1;
10  instrução b2;
11  ...
12 }
```

O esboço de código acima pode ser lido da seguinte forma: se (`if`) uma determinada condição for satisfeita, execute as instruções entre chaves (instruções a1, a2, etc), caso contrário (`else`), ou seja, se a condição não for satisfeita, execute o segundo conjunto de instruções entre chaves (instruções b1, b2, etc).

Veja abaixo um código que soluciona uma equação do segundo grau. A linha 12 calcula o valor de Δ com base nos coeficientes a, b e c, e nas linhas 15, 19 e 25 determinamos o que acontece no caso de Δ ser negativo, nulo ou positivo.

■ Exemplo 5.4

```
1  //declaração de variáveis
2  float a, b, c, delta, x1, x2;
3
```

```
4  void setup()
5  {
6    Serial.begin(9600);
7    a = 1;
8    b = 5;
9    c = 4;
10
11   //calcula o valor de delta
12   delta = b*b - 4*a*c;
13
14   //estrutura de condição para calcular as soluções
15   if (delta < 0)
16   {
17     Serial.print("A equação não possui soluções");
18   }
19   if (delta == 0)
20   {
21     x1 = -b/(2*a);
22     Serial.print("Solução: x1 = ");
23     Serial.print(x1);
24   }
25   if (delta > 0)
26   {
27     x1 = -b/(2*a) + sqrt(delta)/(2*a);
28     x2 = -b/(2*a) - sqrt(delta)/(2*a);
29     Serial.println("Solução:");
30     Serial.print("x1 = ");
31     Serial.println(x1);
32     Serial.print("x2 = ");
33     Serial.println(x2);
34   }
35 }
36
37 void loop()
38 {
39
40 }
```

5.8 Estrutura de repetição

A estrutura de repetição permite que uma sequência de instruções seja repetida quantas vezes o programador desejar. Observe que a função `loop()` já faz algo semelhante; a diferença é que com a última o programador não escolhe quantas vezes as instruções são repetidas. Os principais comandos de repetição são o `for()` e o `while()`.

5.8.1 Comando `for()`

A sintaxe do comando `for()` é a seguinte:

```
1  for(início; condição para continuar o loop; incremento)
2  {
```

```
3   instrução a1;
4   instrução a2;
5   ...
6 }
```

As instruções entre os parênteses determinam quantas vezes as instruções entre as chaves são executadas. Para isso, uma determinada variável (um contador) é inicializada e incrementada, com os comandos entre as chaves sendo repetidos até que a condição deixe de ser satisfeita.

No exemplo 5.5, a variável "contadora" `k` começa com o valor 2 e é incrementada por 1 a cada passo do loop (a instrução `k++` é o mesmo que `k = k + 1`). Quando `k` ultrapassa o valor de 15, a condição `k<=15` deixa de ser satisfeita e a repetição cessa. Assim, o programa abaixo imprime no monitor serial (comando `Serial.println()`) o quadrado do valor de k, com k variando de 2 a 15: 4, 9, 25, ...

■ **Exemplo 5.5**

```
1  int k;
2
3  void setup()
4  {
5    Serial.begin(9600);
6
7    for(k=2; k<=15; k++)
8    {
9      Serial.println(k*k);
10   }
11 }
12
13 void loop()
14 {
15
16 }
```

Exercício 5.1 Escreva um programa que mostre no monitor serial todos os números pares entre 150 e 300.

Exercício 5.2 Escreva um programa que mostre no monitor serial o valor aproximado da seguinte série: $\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\cdots$.

Exercício 5.3 Escreva um programa que mostre no monitor serial a matriz identidade $I_{10\times10}$ abaixo:

```
1  1 0 0 0 0 0 0 0 0 0
2  0 1 0 0 0 0 0 0 0 0
3  0 0 1 0 0 0 0 0 0 0
4  0 0 0 1 0 0 0 0 0 0
5  0 0 0 0 1 0 0 0 0 0
6  0 0 0 0 0 1 0 0 0 0
7  0 0 0 0 0 0 1 0 0 0
8  0 0 0 0 0 0 0 1 0 0
```

```
 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 1
```

5.8.2 Comando `while()`

A sintaxe do comando `while()` é

```
while(condição)
{
 instrução a1;
 instrução a2;
 ...
}
```

que pode ser "lido" como "enquanto uma condição for satisfeita, execute o bloco de instruções entre as chaves". Com esse comando, o exemplo 5.5 ficaria assim:

```
int k;

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

  k = 2;

  while(k<=15)
  {
    Serial.println(k*k);
    k++;
  }
}

void loop()
{

}
```

Observe que a variável contadora `k` foi previamente inicializada e que o incremento `k++` foi colocado como um comando dentro das chaves. O comando `while()` é mais usado quando não se sabe ao certo quantas vezes um determinado bloco de instruções deve ser repetido para que uma determinada tarefa seja executada.

Exercício 5.4 Refaça os exercícios 5.1, 5.2 e 5.3 usando a função `while()`.

5.9 Funções

Em linguagens de programação, funções são blocos de instruções que podem ser chamados em qualquer parte do corpo do programa. Com esses blocos de instruções, a função pode ou retornar um valor ou executar uma tarefa. As funções são usadas quando os blocos são chamados mais de uma vez, o que evita que eles tenham que ser digitados duas ou mais vezes, melhorando a estética do código, deixando-o mais organizado e , assim, mais fácil de ser corrigido, compreendido e melhorado.

As funções podem ser criadas pelo programador como também podem ser usadas as predefinidas pelo Arduino (`pinMode()`, `digitalWrite()`, `delay()`, `for()`, etc).

5.9.1 Criação e chamada de funções

Em geral, as funções são criadas antes da estrutura `setup()`. A sintaxe é

```
tipo-do-retorno nome(tipo variável1, tipo variável2, ...)
{
 declaração de variáveis; (quando necessário)

 instrução 1;
 instrução 2;
 ...
 return variável; (opcional)
}
```

Se a função retornar um valor, é obrigatório que seja determinado o tipo do retorno, que pode ser um número inteiro, um número real ou mesmo um caractere. As variáveis dentro do parênteses são as variáveis de entrada, cujos tipos também devem ser explicitados.

■ **Exemplo 5.6** Função de nome "func", com retorno do tipo `float`, com variáveis de entrada "var1" e "var2", ambas do tipo inteiro:

```
float func(int var1, int var2)
```

Já as funções que não retornam valores são do tipo "void". Em algumas linguagens de programação, como Pascal, elas são chamadas de procedimentos (*procedures*), e não de funções. No exemplo 5.7 temos a declaração da função que escolhemos chamar de "pul", e que não retorna valor. Ela também não possui variáveis de entrada.

■ **Exemplo 5.7**

```
void pul()
```

É possível que variáveis sejam declaradas para uso exclusivo dentro da função. Nesse caso, chamamos elas de "variáveis locais", em contraposição àquelas declaradas antes de `void setup()`, que são "variáveis globais". A regra é que variáveis locais são "enxergadas" apenas dentro da função onde foram declaradas, enquanto que as locais são enxergadas em todo o escopo do programa.

A chamada de uma função pode ser realizada em qualquer parte do programa, até mesmo dentro da própria função (atributo este conhecido como recursividade, característico da linguagem C).

■ **Exemplo 5.8** Seja uma função que retorna o sinc de um número real. O "sinc" é uma função matemática muito usada na teoria da difração, aprecendo também em física atômica, sendo definida pela razão do seno de um número pelo próprio número:

$$\mathrm{sinc}(x) = \frac{\sin(x)}{x}$$

Vamos criar um programa que retorna o sinc de 4:

```
float sinc(float x)
{
  float f;
  f = sin(x)/x;
  return f;
```

```
6   }
7
8   void setup()
9   {
10    Serial.begin(9600);
11    Serial.println(sinc(4));
12  }
13
14  void loop()
15  {
16
17  }
```

Observe, dentro da função sinc(), que declaramos uma variável local `f`, que foi usada apenas para retornar o valor da função. Na linha `11` temos a chamada da função dentro de `println()`.

■ **Exemplo 5.9** Programa que informa se um aluno foi aprovado ou não, com base na sua média final.

```
1   void situacao(float nota)
2   {
3     if (nota>=7)
4     {
5       Serial.println("Aluno aprovado");
6     }
7      if (nota<7)
8     {
9       Serial.println("Aluno reprovado");
10    }
11  }
12
13  void setup()
14  {
15    Serial.begin(9600);
16    float nota = 8.5;
17    situacao(nota);
18  }
19
20  void loop()
21  {
22
23  }
```

Exercício 5.5 Crie uma função quer retorna o menor valor entre dois números inteiros. Naturalmente, a função deve ter duas variáveis inteiras de entrada.

Exercício 5.6 Crie uma função quer retorna o fatorial de um número inteiro.

5.9.2 Funções predefinidas do Arduino

Abordaremos aqui apenas as funções mais usadas por iniciantes no mundo do Arduino. Para uma lista completa, consulte o site oficial do Arduino.

- `pinMode()`: Essa função habilita uma determinada porta da placa do Arduino para entrada ou saída de dados. Sua sintaxe é

```
1  pinMode(porta, modo);
```

onde a "porta" pode ser qualquer uma da placa e o modo pode ser para entrada (`INPUT`) ou saída (`OUTPUT`) de dados.

■ **Exemplo 5.10** Porta `10` habilitada para saída de tensão:

```
1  pinMode(10, OUTPUT);
```

■ **Exemplo 5.11** Porta `A5` habilitada para entrada de tensão:

```
1  pinMode(A5, INPUT);
```

- `digitalWrite()`: Coloca ou 0 ou 5 V de tensão em uma porta específica. É obrigatório que a porta esteja habilitada para saída de tensão com o comando `pinMode()`. No Arduino Uno R3, as portas de `0` a `13` podem ser usadas com esse propósito. Sua sintaxe é

```
1  digitalWrite(porta, valor);
```

Os valores assumidos podem ser HIGH (5 V) ou LOW (0 V).

■ **Exemplo 5.12** Porta `13` com tensão de 5 V:

```
1  digitalWrite(13, HIGH);
```

- `digitalRead()`: Realiza a leitura de tensão de uma determinada porta. É obrigatório que a porta esteja habilitada para entrada de tensão com o comando `pinMode()`. No Arduino Uno R3, as portas que podem ser usadas são as de `A0` a `A5`. Sintaxe:

```
1  digitalRead(porta);
```

Normalmente, o valor será armazenado em alguma variável. Além disso, o valor da leitura será sempre um número entre 0 e 1023, correspondendo aos limites 0 e 5 V, respectivamente.

■ **Exemplo 5.13** Armazenamento na variável **tensao** do valor lido na porta `A0`:

```
1  tensao = digitalRead(A0);
```

- `analogWrite()`: Faz uma determinada porta ficar sob tensão em modulação por largura de pulso (*Pulse Width Modulation* - PWM). Em outras palavras, a função simula uma saída analógica entre 0 e 5 V ao fazer a tensão variar rapidamente entre esses extremos, de forma que o valor médio é uma função contínua. Assim, não é, de fato, uma saída analógica, e sim uma saída digital modulada. Uma saída de tensão verdadeiramente *contínua* entre esses valores de tensão é possível com *shields* específicos. Segundo o site do Arduino, a frequência da modulação varia dependendo da porta, podendo ser 490 Hz ou 980 Hz. Apenas as portas indicadas pelo símbolo $\sim$ podem ser usadas como PWM.

A sintaxe do comando é

```
1   analogWrite(porta, valor);
```

O atributo valor regula a largura temporal do pulso em relação ao período de repetição. Os valores devem estar entre 0 (0%) e 255 (100% do período). Isso faz com que, *na média*, a tensão varie continuamente entre 0 e 5 V, sendo ideal para regular a intensidade de um LED ou para controlar a velocidade de um motor DC. A figura 5.5 ilustra o papel do atributo valor na geração dos pulsos.

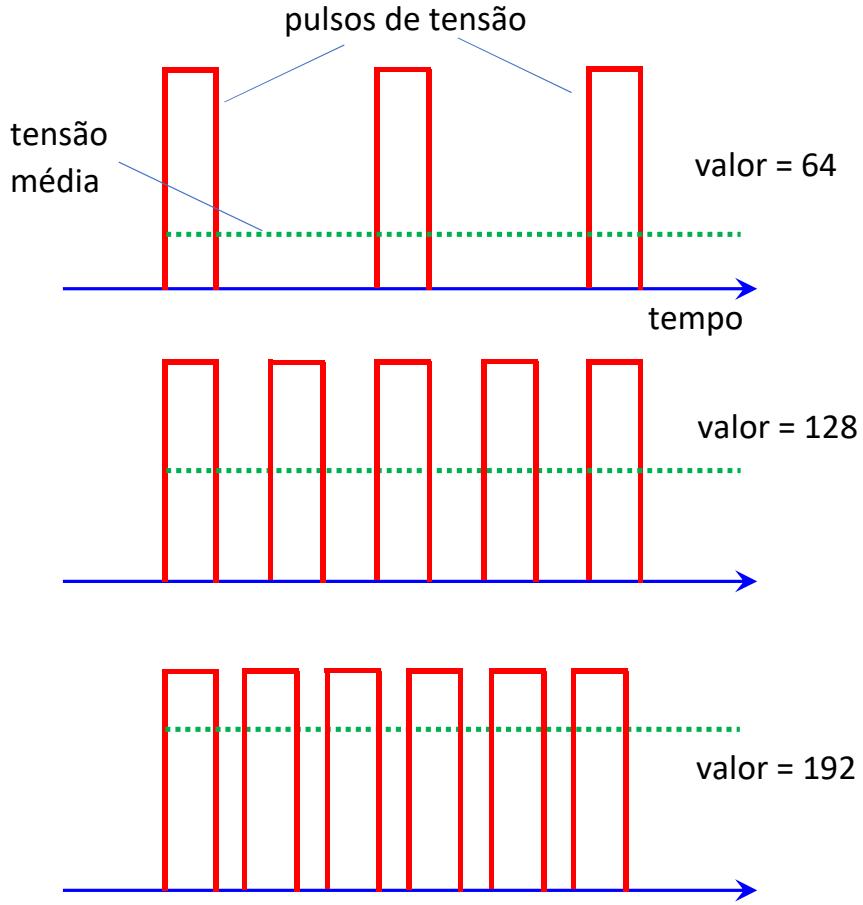


Figura 5.5: Ilustração da saída de tensão em modulação por largura de pulso (PWM). Temos representado o atributo valor em 64, 128 e 196, o que corresponde a pulsos com largura temporal de 25%, 50% e 75% do período, respectivamente. A linha verde tracejada seria a média do trem de pulsos quadrados (linhas vermelhas), que pode ser ajustada entre 0 e 5 V.

- `delay()`: Faz o fluxo do programa parar por alguns milissegundos. Assim, para fazer o programa esperar 2 segundos, basta incluir o código `delay(2000)`. É ideal para exibir o valor de uma variável no monitor serial de forma mais lenta.
- `delayMicroseconds()`: Idem anterior, só que em microssegundos.
- `sin()`, `cos()` e `tan()`: Funções matemáticas trigonométricas. O argumento deve estar em radianos.
- `sqrt()`, `pow()` e `abs()`: Funções matemáticas para raiz quadrada, potência e valor absoluto.

■ **Exemplo 5.14** O código abaixo exibe os valores numéricos de $\sqrt{5}$, 7^3 e $|-20|$.

```
1   void setup()
2   {
3     Serial.begin(9600);
4
5     Serial.println(sqrt(5));
6     Serial.println(pow(7,3));
7     Serial.println(abs(-20));
8   }
9
10  void loop()
11  {
12
13  }
```

Para o uso desses comandos não é necessário o carregamento da biblioteca (ver Seção 5.10) `<math.h>`, como na linguagem C.

5.10 Bibliotecas

A maneira mais fácil de entender as bibliotecas é pensá-las como um tipo de função. A diferença é que o bloco de instruções não fica no escopo do programa, e sim na pasta `libraries` do Arduino. As bibliotecas são aplicadas na chamada de blocos de instruções que são usados muitas vezes em diferentes programas. Um exemplo é a biblioteca `math.h` da linguagem C, que contém as funções `sqrt()`, `pow()`, `atan()`, etc. As bibliotecas também podem ser criadas pelo programador.

Uma biblioteca padrão é chamada com a diretiva include antes das estruturas `setup()` e `loop()`:

```
1   #include<nome-da-biblioteca>
```

Já uma biblioteca criada pelo usuário segue a sintaxe abaixo:

```
1   #include "nome-da-biblioteca.h"
```

Para a criação de uma biblioteca é necessário um IDE da linguagem C/C++, como o Dev-C++, ou mesmo um editor de texto simples, como o Bloco de Notas do Windows. Usando este último, basta digitar o código no Bloco de Notas e salvar com a extensão `h`, que indica uma biblioteca na linguagem C. Feito isso, basta localizar a pasta `libraries`, geralmente em `C:\Program Files (x86)\Arduino\libraries`. Para ficar mais organizado, a biblioteca pode ficar nesse endereço dentro de uma outra pasta, com o nome que lembre a biblioteca.

■ **Exemplo 5.15** Abaixo temos um programa que calcula o sinc de 4 (compare com o Exemplo 5.7), mas agora usando uma biblioteca. Para isso, criamos a biblioteca `funcaosinc.h`, que foi colocada dentro da pasta `Funcao_sinc`, localizada na pasta `libraries` do Arduino. Os códigos da biblioteca e do sketch estão mostrados abaixo.

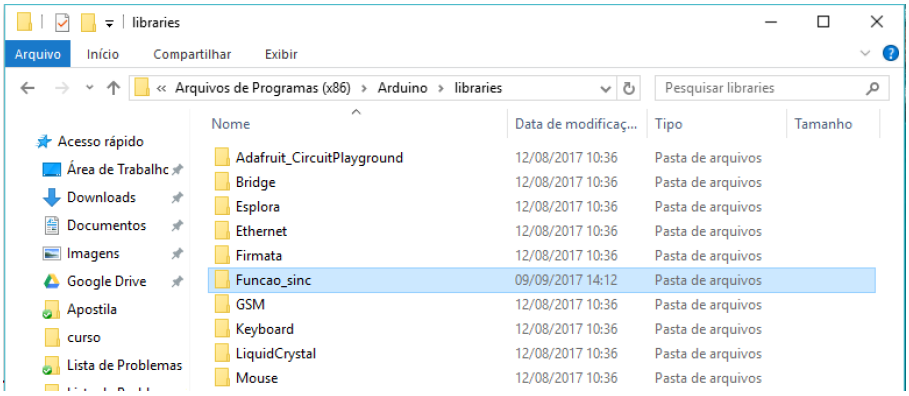


Figura 5.6: Pasta criada para a biblioteca funcaosinc.h.

```
1  //Código da biblioteca
2  float sinc(float x)
3  {
4    float f;
5    f = sin(x)/x;
6    return f;
7  }
```

```
1  //Código do sketch
2  #include "funcaosinc.h"
3
4  void setup()
5  {
6    Serial.begin(9600);
7    Serial.println(sinc(4));
8  }
9
10 void loop()
11 {
12
13 }
```

Exercício 5.7 Escreva uma biblioteca que execute as mesmas instruções das funções dos exercícios 5.5 e 5.6.

5.11 Vetores e matrizes

Em diversas linguagens de programação, existe o que chamamos de variáveis multidimensionais, capazes de armazenar não apenas um número ou um caractere, mas sim vários deles.

5.11.1 Vetores

Um vetor é uma variável unidimensional. A forma mais fácil de entendê-la é imaginando que ela possui vários "espaços" que podem ser acessados ou preenchidos com valores numéricos ou não. Cada espaço é uma componente do vetor. A Fig. 5.7 ilustra um vetor com 5 componentes.

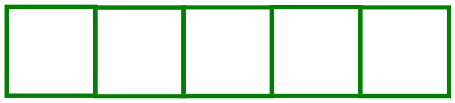


Figura 5.7: Vetor com 5 componentes.

A declaração de uma variável vetorial é bastante semelhante ao de uma variável convencional; o que muda é que agora precisamos informar o tamanho do vetor, isto é, a quantidade de componentes que ele possui:

```
tipo-da-variável nome-da-variavel[tamanho-do-vetor];
```

■ **Exemplo 5.16** Variável de nome vet, do tipo real com 9 componentes:

```
float vet[9];
```

Isso quer dizer que podemos preencher os nove espaços com números reais do tipo `float`. Uma mistura de tipos no mesmo vetor (por exemplo, componentes inteiras com reais) não é permitida.

Para acessar um único componente do vetor, basta saber a posição correspondente, lembrando que a primeira posição é representada pelo número `0` e a última, por `n-1`, onde `n` é o tamanho do vetor.

■ **Exemplo 5.17** Agora vamos declarar um vetor de nome v, de 5 componentes inteiras, atribuir valores às componentes e, por fim, apresentar no monitor serial a soma primeira com a última componente.

```
void setup()
{
 //Declaração do vetor
 int v[5];
 int soma;

 //Atribuição de valores às componentes do vetor
 v[0] = 3;
 v[1] = 8;
 v[2] = 11;
 v[3] = -7;
 v[4] = 60;

 //Soma da primeira com a última componente
 soma = v[0] + v[4];

 //Mostrando no monitor serial a o resultado da soma
  Serial.begin(9600);
  Serial.println(soma);
}

void loop()
{

}
```

O código acima pode ser melhor compreendido com o fluxograma da figura abaixo.

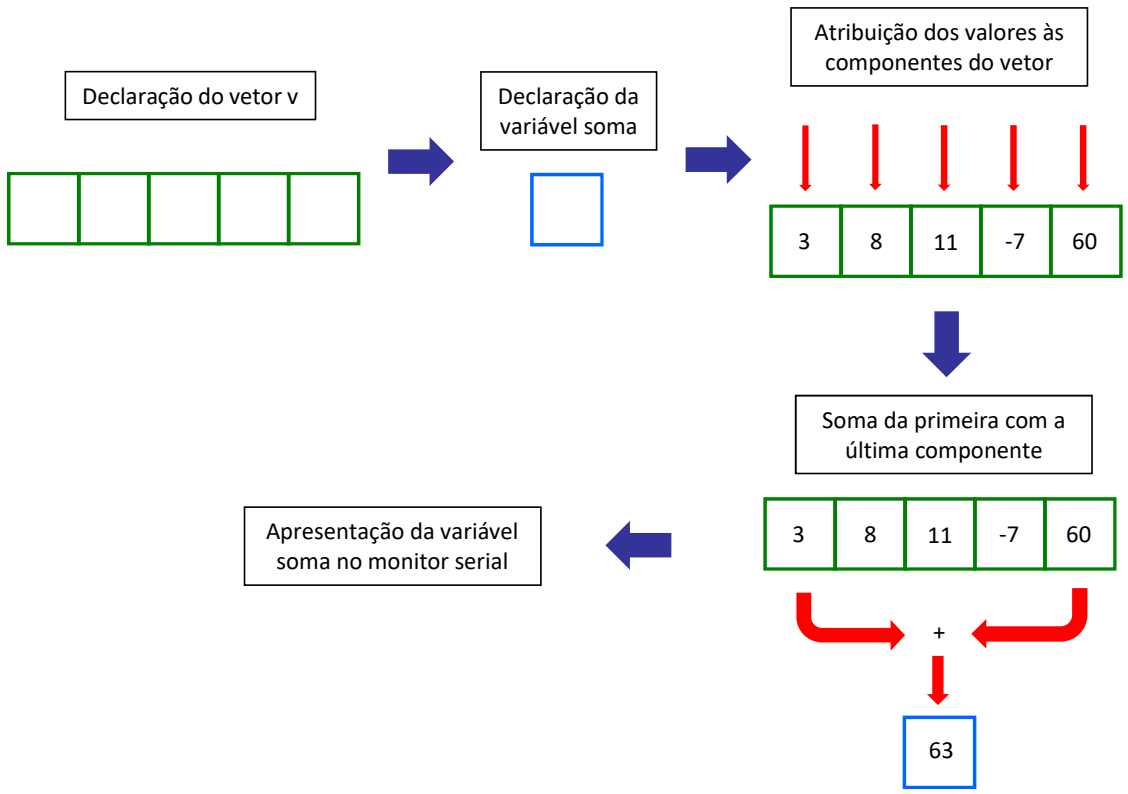


Figura 5.8: Fluxograma do Exemplo 5.17.

Lembre-se, sempre, de efetuar as operações usando as componentes do vetor, e não o vetor em si. O mesmo vale para a declaração e o acesso.

O uso de vetores, quando apropriado, tem muitas vantagens. Imagine que uma pessoa precise monitorar e regular a temperatura de 12 salas de uma empresa. Poderiam ser declaradas 12 variáveis para cada sala:

```
1  float T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12;
```

É bastante óbvio que essa não é uma boa escolha, já que é possível fazer da seguinte forma:

```
1  float T[12];
```

onde `T[0]` é a temperatura da sala 01, `T[1]` da sala 02 e assim por diante. Na Seção 7.1, que descreve um projeto envolvendo o controle de 5 leds, há um exemplo de uso de um vetor.

5.11.2 Matrizes

A generalização de vetores para duas ou mais dimensões é chamada de matriz. Assim, em vez de um único índice, matrizes usam vários.

■ **Exemplo 5.18** Declaração de uma matriz de nome mat de 3 dimensões, de tamanho 4 x 5 x 6, de variáveis do tipo char:

```
1  char mat[4][5][6];
```

A forma como manipulamos matrizes é análoga à forma como usamos os vetores.

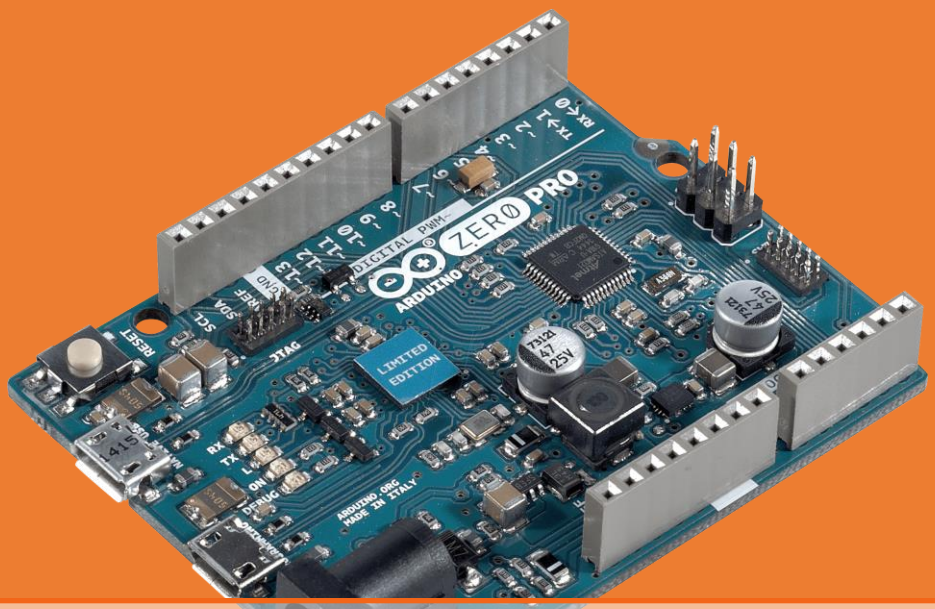
■ **Exemplo 5.19** Escreva um programa que mostre no monitor serial a matriz identidade 10 x 10 abaixo:

```
1  0  0  0  0  0  0  0  0  0
0  1  0  0  0  0  0  0  0  0
0  0  1  0  0  0  0  0  0  0
0  0  0  1  0  0  0  0  0  0
0  0  0  0  1  0  0  0  0  0
0  0  0  0  0  1  0  0  0  0
0  0  0  0  0  0  1  0  0  0
0  0  0  0  0  0  0  1  0  0
0  0  0  0  0  0  0  0  1  0
0  0  0  0  0  0  0  0  0  1
```

```
int ident[10][10];
int k,p;

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

   //Cria a matriz identidade
 for (k=0; k<=9; k++)
  for (p=0; p<=9; p++)
  {
   if (k == p) ident[k][p] = 1;
   if (k != p) ident[k][p] = 0;
  }

 //Escreve no monitor serial a matriz identidade
 for (k=0; k<=9; k++)
 {
  Serial.println();
  for (p=0; p<=9; p++)
  {
   Serial.print(ident[k][p]);
   Serial.print("␣");
  }
 }
}

void loop()
{

}
```

Exercício 5.8 Escreva um programa que mostre no monitor serial a matriz abaixo.

```
1  1 0 0 0 0 0 0
2  1 2 0 0 0 0 0
3  1 1 3 0 0 0 0
4  1 1 1 4 0 0 0
5  1 1 1 1 5 0 0
6  1 1 1 1 1 6 0
7  1 1 1 1 1 1 7
```



6. Componentes eletrônicos

Apresentaremos aqui o funcionamento dos dispositivos básicos na eletrônica usados nessa apostila, em particular:

- A matriz de linhas
- Jumpers
- Resistores
- Fontes
- Capacitores
- LED's
- Fototransistores
- LDR's
- Relés
- Potenciômetros
- PZT's

6.1 A matriz de linhas

A matriz de linhas, *protoboard* ou matriz de contato é uma placa de ensaios para montagens de circuitos eletrônicos. Ela é composta por furos com conexões internas, pelo qual é possível fazer a ligação entre diversos dispositivos, evitando solda.

Os furos da placa são conectados obedecendo algumas regras. Como mostrado na Fig. 6.1, os pinos centrais (denotados pelas linhas `a`, `b`, `c`, ..., `j`) possuem ligações na vertical. Por exemplo,

todos os furos indicados pelas coordenadas (10, a), (10, b), (10, c), (10, d) e (10, e) são conectados e, portanto, nos circuitos, todos eles estarão sob a mesma tensão. De forma análoga, todos os furos da linha indicada por + (e também −) estão conectados entre si. Esses dois últimos conjuntos de furos são muito usados para conectar uma fonte de tensão (furos +) e o terra (furos −), apesar de isso não ser uma obrigação.

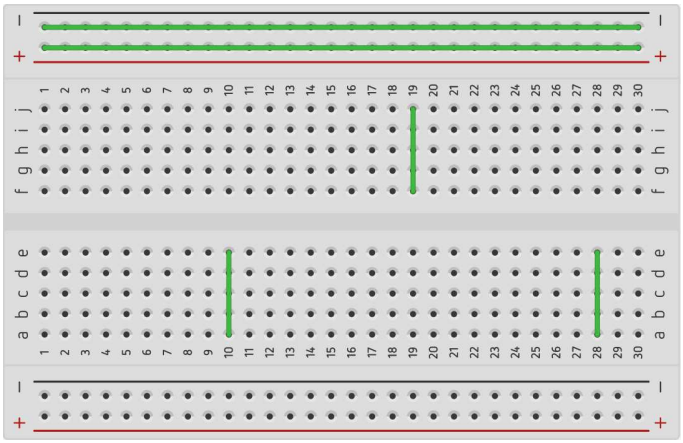


Figura 6.1: Representação de uma matriz de linhas com alguns furos que estão conectados entre si.

No exemplo baixo temos um circuito RC nas representações convencional de livros didáticos e na matriz de linhas.

■ **Exemplo 6.1**

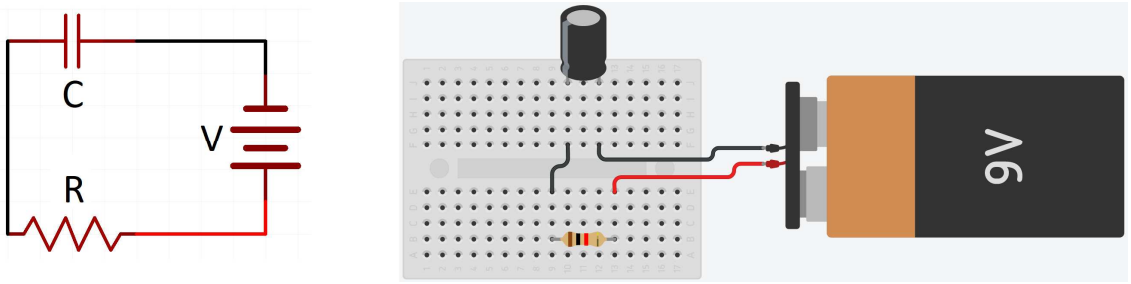


Figura 6.2: Duas representações do mesmo circuito. Legenda: R - resistor, C - capacitor, V - fonte de tensão.

Abaixo temos um circuito mais complexo, feito de forma aleatória para exemplificar o uso da matriz de linhas.

■ **Exemplo 6.2**

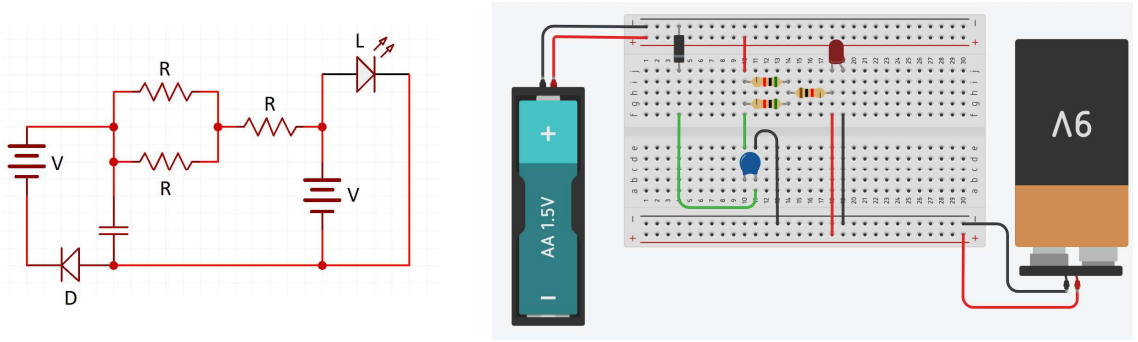


Figura 6.3: Duas representações do mesmo circuito. Legenda: R - resistor, C - capacitor, V - fonte de tensão, L - LED, D - Diodo.

6.2 Jumpers

Os jumpers são os fios que usamos para conectar os componentes eletrônicos na protoboard. Basicamente, há jumpers de três tipos: macho-macho, macho-fêmea e fêmea-fêmea. Em circuitos muito complexos, a quantidade de jumpers pode ser muito grande e uma padronização pode facilitar na hora de modificar o projeto. Em geral, fios vermelhos são usados para o terminal positivo da tensão, os fios pretos são reservados para o terra e os fios de outras cores são para leitura e escrita de dados.

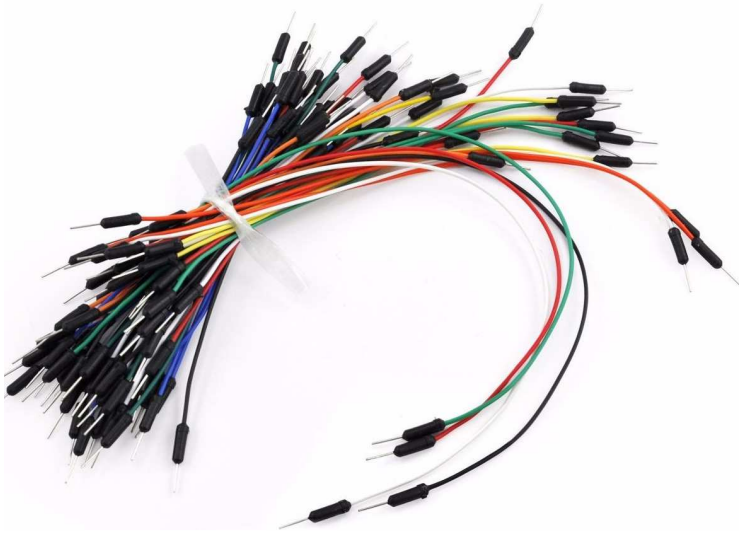


Figura 6.4: Jumpers macho-macho de diversas cores.

6.3 Resistores

O resistor, designado pela letra R, é um componente eletrônico que obedece à lei de Ohm quando atravessado por por uma corrente elétrica i, de forma que podemos escrever

$$V = iR, \tag{6.1}$$

onde V representa uma queda de tensão elétrica no sentido da corrente.

Figura 6.5: Resistor e sua representação em circuitos elétricos.

6.4 Fontes

Fontes de tensão, ou simplesmente fontes, são os dispositivos que geram uma diferença de potencial elétrico (ou, novamente, tensão) entre seus terminais. As tensões podem ser contínuas, quando o valor da tensão permanece constante no tempo, ou alternada, quando a tensão oscila periodicamente.

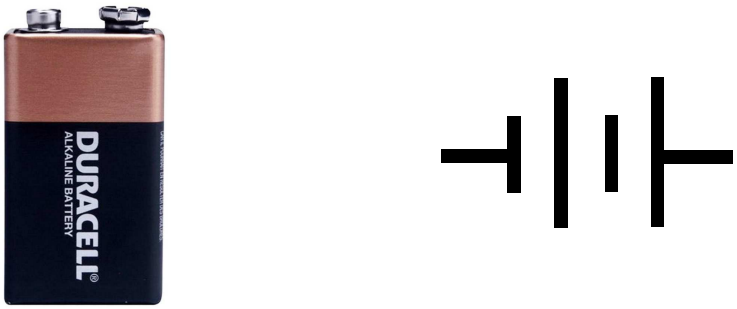


Figura 6.6: Exemplo de fonte de tensão (uma bateria de 9 V) e um exemplo de representação de fonte contínua (há muitas variações).

■ **Exemplo 6.3** Exemplo de fontes contínuas: pilhas AA e AAA (1,5 V), baterias de 9 V, fontes de tensão DC sintonizáveis e placas de energia solar.

■ **Exemplo 6.4** Exemplo de fontes alternadas: tomadas residenciais (110 ou 220 V de tensão senoidal rms) e geradores de funções (quadrada, triangular ou senoidal com amplitude variável).

As fontes de tensão, junto com os capacitores carregados, são as fontes de energia do circuito elétrico, sem os quais não haverá corrente elétrica.

Não tente usar uma tomada residencial como fonte de tensão em nenhum exemplo desta apostila. A tensão elevada gera correntes altas, levando a risco de choques elétricos e incêndios devido ao excesso de calor em fios de pouca espessura. Priorize o uso de baixas tensões contínuas (15 V ou menos).

6.5 Corrente, tensão, resistência e lei das malhas

O conhecimento da física de circuitos elétricos a nível de Ensino Médio é fundamental para a criação de projetos Arduino.

Segundo a lei de Ohm, a corrente i em materiais dito "ômicos" é proporcional a diferença de potencial V aplicado nos seus terminais:

$$i = \frac{V}{R}. \tag{6.2}$$

Então, quanto maior sua resistência, menor a corrente elétrica que o atravessa. Esse é um ponto muito importante para ser levado em conta na criação dos projetos, visto que diversos componentes eletrônicos suportam uma certa corrente máxima.

6.5.1 Associação de resistores

Dois ou mais resistores em série (veja a Fig. 6.7) se comportam como se fosse um só, mas com resistência equivalente R_{eq} igual à soma das resistências individuais. Matematicamente:

$$R_{eq} = R_1 + R_2 + R_3 + \cdots \tag{6.3}$$

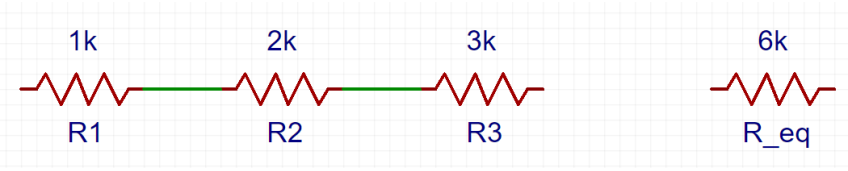


Figura 6.7: Três resistores em série, com valores de 1 kΩ, 2 kΩ e 3 kΩ, e o resistor equivalente, com resistência 6 kΩ. Em alguns softwares e livros, o símbolo Ω pode estar omitido.

Para resistores em paralelo (veja a Fig. 6.8), a resistência equivalente é o inverso da soma dos inversos das resistências individuais: Matematicamente:

$$\frac{1}{R_eq} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \cdots \tag{6.4}$$

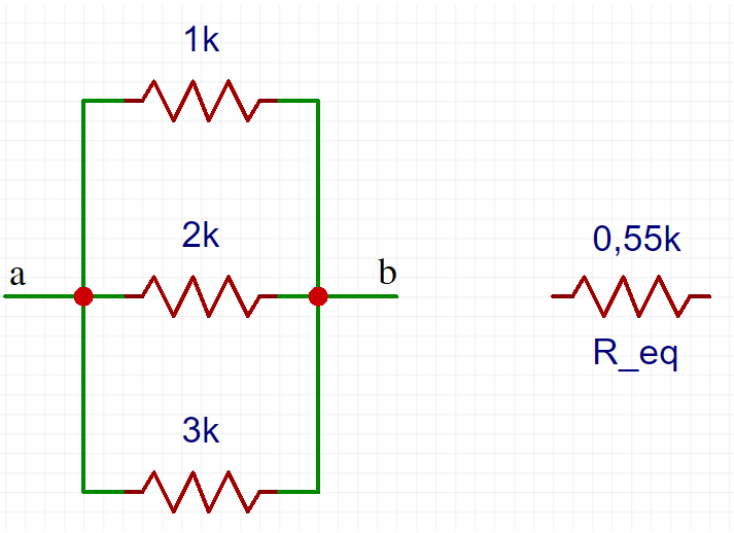


Figura 6.8: Três resistores em paralelo, com valores de 1 kΩ, 2 kΩ e 3 kΩ, e o resistor equivalente, com resistência de aproximadamente 0,55 kΩ, tomando por base os terminais a e b indicados.

Em seus projetos, você pode combinar resistores até obter uma resistência apropriada. Assim, se um resistor de 6 kΩ não estiver disponível, você pode combinar três resistores em série como indicado na Fig. 6.7. É possível também obter tensões diferentes das fontes disponíveis (veja o Exemplo 6.6.).

6.5.2 Lei das malhas

A lei das malhas permite calcular as correntes no circuito. Ela pode ser enunciada da seguinte forma: quando percorremos o circuito fechado até completá-lo, a soma das diferenças de potencial é zero.

■ **Exemplo 6.5** Calcule a corrente no circuito abaixo.

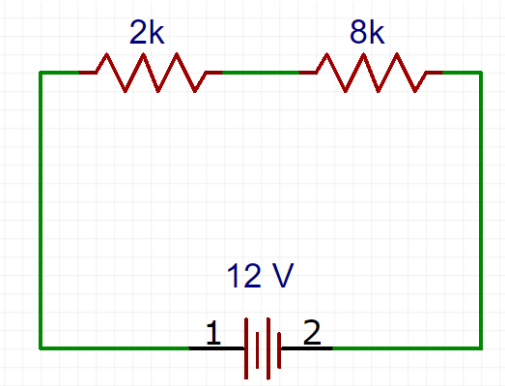


Figura 6.9: Exemplo de circuito.

Percorrendo a malha no sentido horário a partir da fonte, temos:

$$\begin{aligned} 12 - i \times 2000 - i \times 8000 &= 0 \\ 10000i &= 12 \\ i &= \frac{12}{10000} \\ i &= 1,2\text{mA} \end{aligned}$$

■ **Exemplo 6.6** Vamos supor que queremos uma tensão de exatamente 2,7 V nos terminais de um dado dispositivo, identificado pela letra D no circuito da Fig. 6.10. Qual é valor do resistor R que devemos usar, supondo que não podemos alterar o valor da tensão da fonte? A resistência do dispositivo vale 50 Ω.

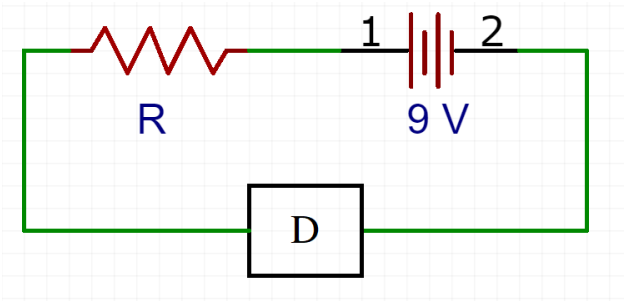


Figura 6.10: Exemplo de circuito.

Pela lei das malhas, a corrente que atravessa o dispositivo D deve ser

$$\begin{aligned} 2,7 &= i \times 50 \\ i &= 54\text{mA} \end{aligned}$$

Então, novamente pela lei de malhas, temos

$$\begin{aligned} 9 - 0,054 \times 50 - 0,054 \times R &= 0 \\ 0,054R &= 9 - 0,054 \times 50 \\ R &\approx 166\ \Omega \end{aligned}$$

Esse valor de resistência pode ser obtido com um potenciômetro (ver Seção 6.6.7).

6.6 Mais componentes eletrônicos

6.6.1 Capacitores

Um capacitor é um dispositivo eletrônico de dois terminais composto por dois condutores separados por uma região não-condutora, que armazena energia elétrica no campo elétrico nessa região. Em um capacitor ideal, se uma diferença de potencial elétrico V é aplicada em seus terminais, cargas Q de mesmo módulo e sinais opostos são criadas em suas placas segundo a equação

$$Q = CV,$$

onde C é a "capacitância" do dispositivo (dada geralmente em unidades de μF), uma constante que depende do seu material e da sua geometria.

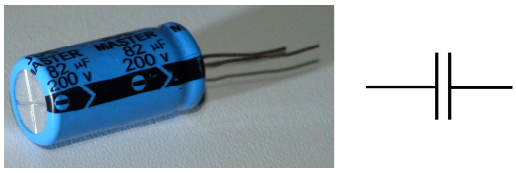


Figura 6.11: Um capacitor de 82 μF e sua representação em circuitos eletrônicos.

Os capacitores são usados em circuitos eletrônicos principalmente como filtros (passa banda e passa baixa), estabilizadores de tensão e também osciladores eletrônicos, sintonizando onda de rádio em frequências específicas. Com capacitores também é possível bloquear corrente contínua enquanto é permitida a passagem de corrente alternada.

Aqui vamos aproveitar para apresentar um simples projeto onde mediremos a tensão em um dado ponto de um circuito RC (circuito com um capacitor e um resistor) em função do tempo, conforme chaveamos uma fonte de tensão de 5 V (veja a figura abaixo). Vamos ligar e desligar a fonte em intervalos pré-determinados, de forma que o capacitor seja periodicamente carregado e descarregado. Para isso, precisamos de uma porta de saída digital (usaremos a porta `11` do Arduino). A medida de tensão será feita entre os terminais do resistor e do capacitor (por razões óbvias), e assim precisamos também de uma porta de entrada analógica (usaremos a `A5`).

O tempo de carga de um capacitor é da ordem da constante característica τ de um circuito RC, definida por

$$\tau = RC.$$

Escolhemos usar um resistor de $R = 3,8$ kΩ e um capacitor de 82 μF, de forma que obtemos $\tau \approx 300$ ms. Assim, para ver a carga e a descarga completa, vamos modular a tensão de 5 V em 500 ms.

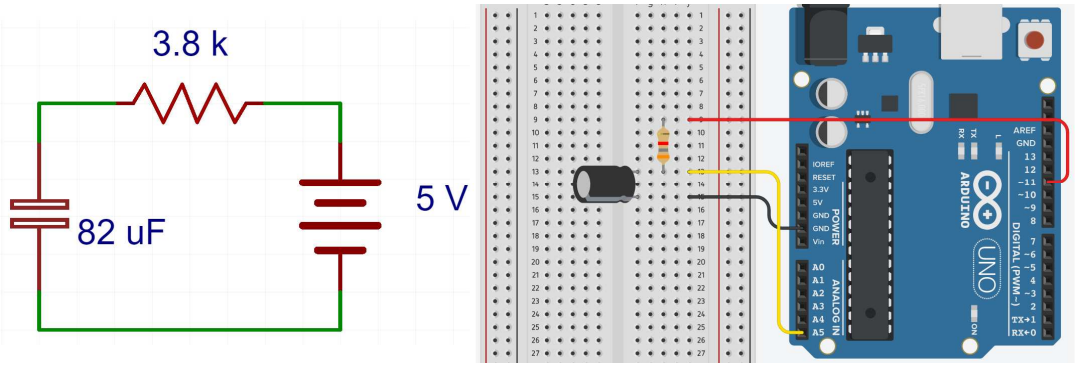


Figura 6.12: Na esquerda temos a representação de um circuito RC com fonte e na direita as conexões usando o Arduino. Desenhados nos sites `https://easyeda.com` e `https://www.tinkercad.com`, respectivamente.

O código está mostrado abaixo. As portas de entrada e saída estão definidas nas linhas 1 e 2 e escolhemos pegar 500 pontos em cada carga e descarga (linha 3) com intervalos de 1 segundo (linha 4). Nas linhas 10-12 habilitamos as portas e o monitor serial. O chaveamento é feito dentro da função `loop()`. Iniciamos com a fonte em 0 V (linha 17) e, 500 vezes (linha 18), armazenamos o valor lido na porta `A5` (linha 20) e convertemos em tensão (linha 21), para em seguida apresentar no monitor serial (linha 22). O mesmo processo é feito quando a tensão está em 5 V (linhas 26-33), reiniciando o ciclo.

```
1  int saida = 11;
2  int entrada = A5;
3  int pontos = 500;
4  int tempo = 1;
5  int valor, k;
6  float tensao;
7
8  void setup()
9  {
10   pinMode(saida, OUTPUT);
11   pinMode(entrada, INPUT);
12   Serial.begin(9600);
13 }
14
15 void loop()
16 {
17   digitalWrite(saida, LOW);
18   for (k=1; k<=pontos; k++)
19   {
20      valor = analogRead(entrada);
21      tensao = float(valor)*5/1023;
22      Serial.println(tensao);
23      delay(tempo);
24   }
25
26   digitalWrite(saida, HIGH);
27   for (k=1; k<=pontos; k++)
```

```
28      {
29          valor = analogRead(entrada);
30          tensao = float(valor)*5/1023;
31          Serial.println(tensao);
32          delay(tempo);
33      }
34  }
```

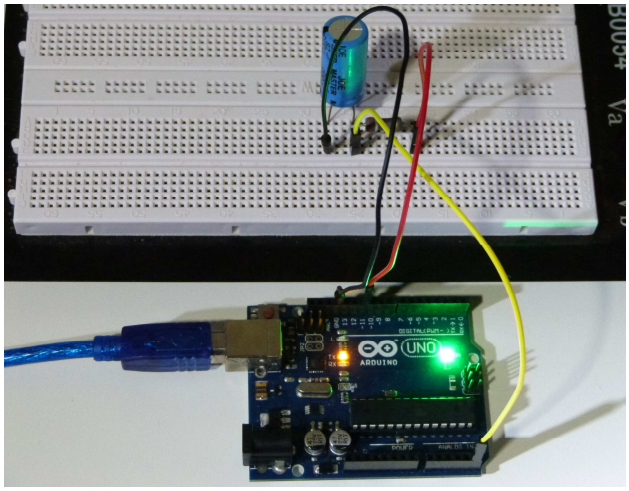


Figura 6.13: Arduino executando o código acima.

Para ficar mais ilustrativo, copiamos as tensões lidas no monitor serial (ver figura 6.14) e colamos em um software gráfico, onde pode ser observado claramente (figura 6.15) os ciclos de carga e descarga do capacitor.

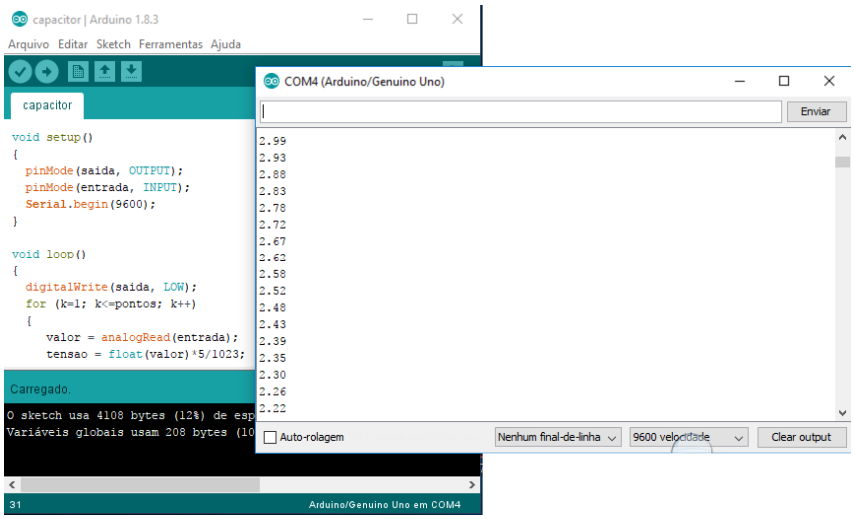


Figura 6.14: Monitor serial apresentando as tensões lidas.

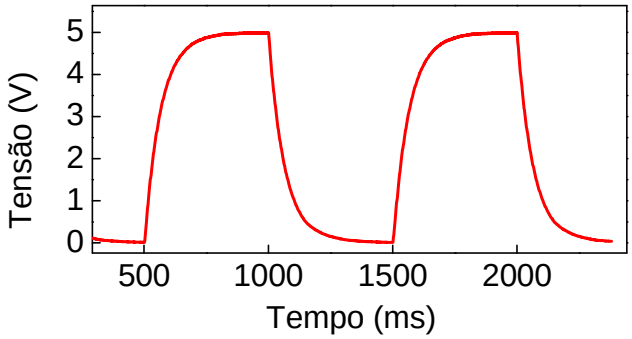


Figura 6.15: Carga e descarga de um capacitor conectado a um resistor e a uma fonte de 5 V modulada em 1,0 Hz.

Exercício 6.1 Introduza um LED no circuito acima e faça uma modificação no código para que o LED acenda quando o capacitor fique com carga acima de 80% da sua carga máxima.

6.6.2 Diodos

Diodos são componentes que permitem a passagem da corrente em apenas um sentido. São importantes para evitar danos a dispositivos que não aceitam corrente reversa. Tem aplicações também em filtros passa banda e passa baixa, e em retificadores elétricos.

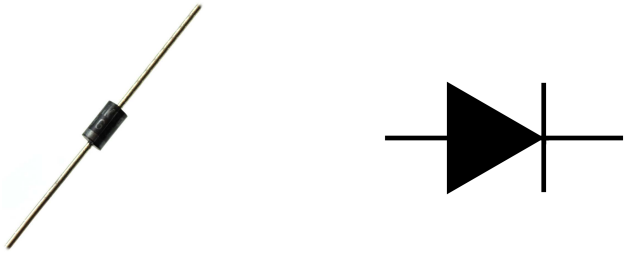


Figura 6.16: Um diodo e sua representação em circuitos elétricos.

6.6.3 LEDs

LED's (*Light-Emitting Diode* - diodo emissor de luz) são dispositivos semicondutores que emitem luz quando uma diferença de potencial elétrico é aplicado em seus terminais. O efeito por trás é a eletroluminescência, onde a luz é emitida via recombinação elétron-buraco. Eles são encontrados em praticamente todas as cores, e também no infravermelho e no ultravioleta, sendo que o desenvolvimento na cor azul resultou no Prêmio Nobel de Física em 2014.

Na figura abaixo temos um LED e sua representação em circuitos eletrônicos. Na perna maior deve ser conectado o polo positivo, e na menor o negativo (ou o terra). Projetos envolvendo LED's podem ser encontrados no Cap. 7.

6.6.4 Fototransistores

Fototransistores são transistores sensíveis à luz. Na prática, eles são dispositivos que convertem luz em corrente elétrica. Assim, um uso bastante comum é em fotodetectores. Sua eficiência depende do comprimento de onda da luz incidente, que por sua vez depende do semicondutor do qual ele é feito. Por exemplo, fotodiodos de silício são eficientes na faixa 190 - 1100 nm, enquanto que os de InGaAs (índio arseneto de gálio) operam melhor entre 800 nm e 2,6 μm. Um projeto envolvendo um fototransistor como um sensor de luz se encontra na Sec. 7.3.

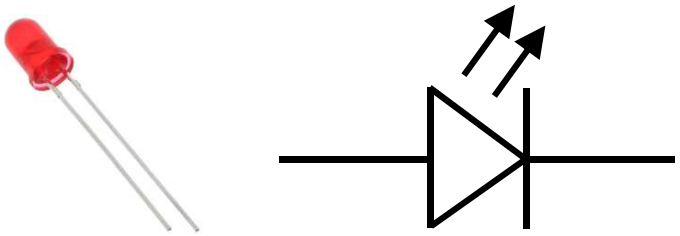


Figura 6.17: Um LED vermelho e sua representação em circuitos eletrônicos.

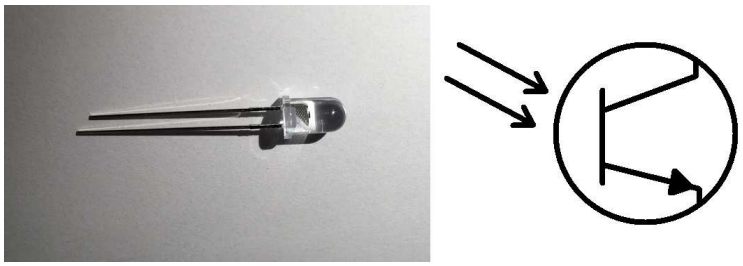


Figura 6.18: Um fototransistor e sua representação em circuitos eletrônicos.

6.6.5 LDRs

LDR's (*Light-Dependent Resistor* - resistores dependentes da luz) são dispositivos cuja resistência varia com a intensidade da luz que incide sobre eles. Na verdade, a resistência *diminui* com o aumento da intensidade da luz. Abaixo temos um LDR e sua representação em circuitos. Um projeto envolvendo um LDR como um sensor de luz se encontra na Sec. 7.2.

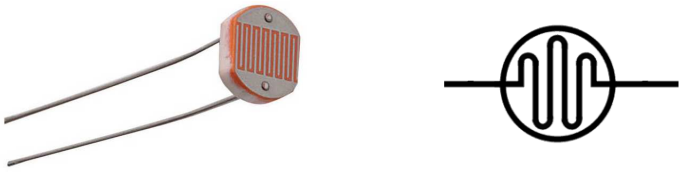


Figura 6.19: LDR e sua representação em circuitos eletrônicos.

6.6.6 Relés

O relé é uma chave eletrônica. Nesse dispositivo, basicamente, algumas portas podem ser chaveadas (isto é, ter sua tensão regulada ou em 0 V ou em uma tensão máxima) dependendo da tensão em uma terceira porta.

Para ilustrar o seu funcionamento, usaremos o relé de um canal de 5 V em um projeto onde controlaremos a luz de dois LED's. Esse relé suporta uma corrente máxima de 10 A e uma tensão máxima de 240 V em corrente alternada, e possui seis portas de tensão, conforme mostra a tabela e a figura abaixo. Duas dessas portas $(+/-)$ são de alimentação para o seu funcionamento, devendo ser de 5 V para esse relé. A porta `S` é a que é usada para o chaveamento das portas `NO` e `NC`. Quando a porta `S` está em `HIGH`, a porta `NO` está com tensão enquanto que a porta `NC` está sem. Quando a porta `S` está em `LOW`, as tensões nas portas `NO` e `NC` se invertem. Na porta `COM` deve ser ligado o positivo que alimentará as portas `NO` e `NC`. Assim, se tivermos 220 V no `COM`, as portas `NO` e `NC` serão chaveadas (digitalmente) entre 0 e 220 V.

$+/-$	Alimentação em 5 V e terra
`S`	*Switch* (chave)
`COM`	*Comum* (Tensão comum)
`NO`	*Normal Open* (aberta normal)
`NC`	*Normal Closed* (normal fechada)

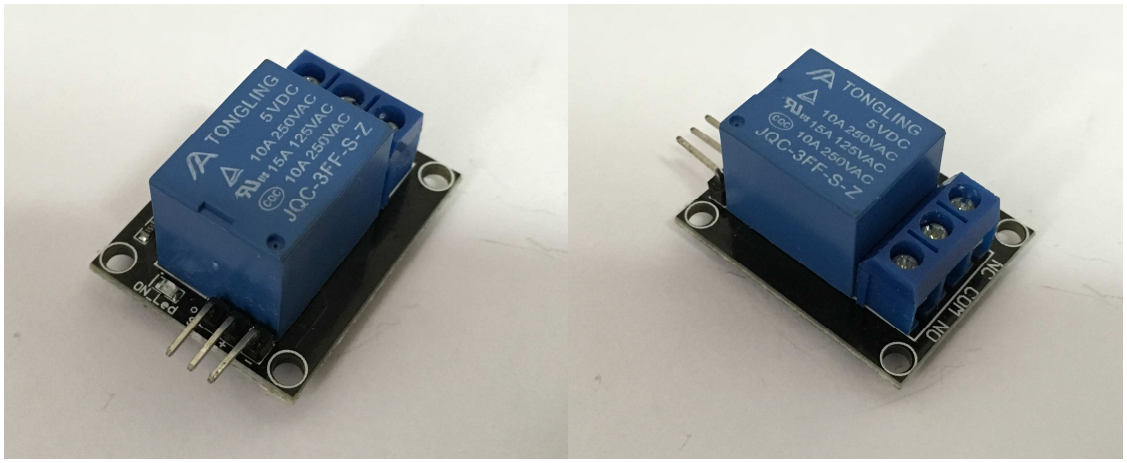


Figura 6.20: Relé de um canal de 5 V. À esquerda podemos ver as portas $+/-$ e `S`, e na direita, as portas `NO`, `COM` e `NC`.

> ! As portas $+/-$ e `COM` podem (e normalmente são) originadas de fontes externas ao Arduino. A única porta que deve obrigatoriamente ser ligada ao Arduino é a `S`, pode onde o chaveamento é feito.

Agora vamos ao projeto. Faremos dois LED's acederem e apagarem de forma alternada. É claro que isso pode ser feito com duas portas digitais do Arduino operando de forma independente. O que faremos aqui, entretanto, é usar apenas uma porta do Arduino (a `3`) e um relé para chavear as tensões nos terminais dos LED's. Abaixo podemos ver o esquema das ligações. Escolhemos alimentar os LED's com 3,3 V, indicado na figura pelo fio laranja.

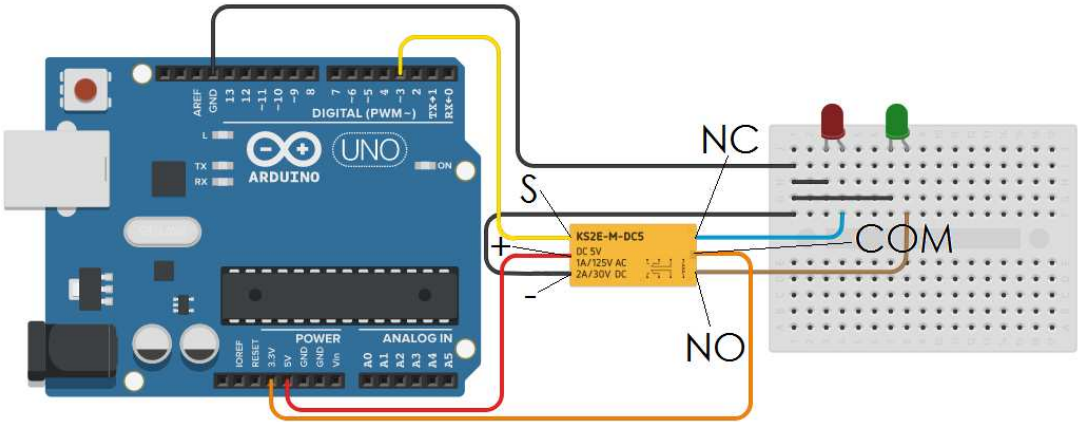


Figura 6.21: Esquema das ligações do Arduino com o relé e os LED's. Desenhado no site `https://www.tinkercad.com`.

Abaixo temos o código. Basta apenas colocar 0 (`LOW`) ou 5 V (`HIGH`) na porta `3` de forma alternada, que o relé fará o resto, chaveando a tensão nos terminais dos LED's.

```
1  void setup()
2  {
3    pinMode(3, OUTPUT);
4  }
5
6  void loop()
7  {
8    digitalWrite(3, HIGH);
9    delay(1000);
10   digitalWrite(3, LOW);
11   delay(1000);
12 }
```

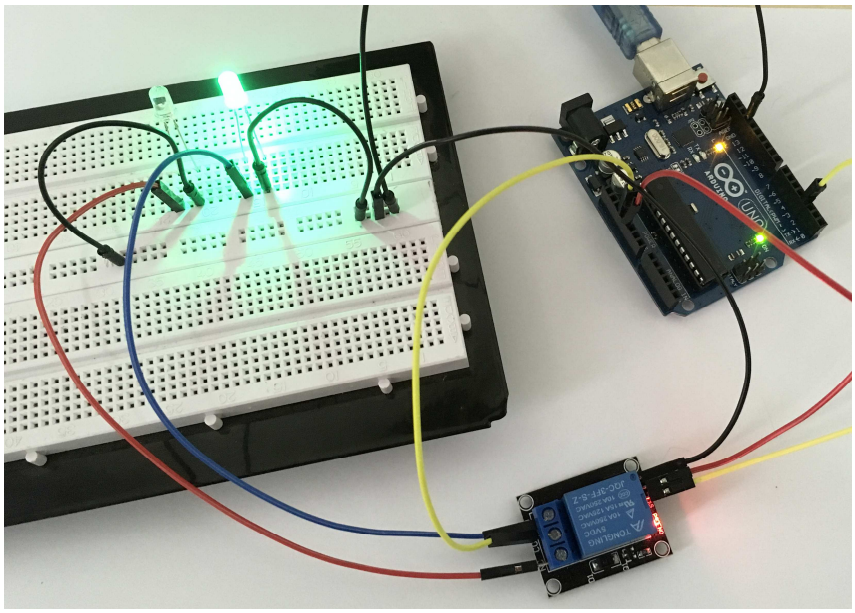


Figura 6.22: Arduino executando o código acima.

6.6.7 Potenciômetros

O potenciômetro é resistor variável de três terminais, que pode ser ajustado com o auxílio de um botão girante (ver figura abaixo). Na realidade, quando apenas dois dos seus terminais são usados, ele age como um simples resistor variável (reostato), enquanto que quando todos os três são usados, ele age como um divisor de tensão ajustável.

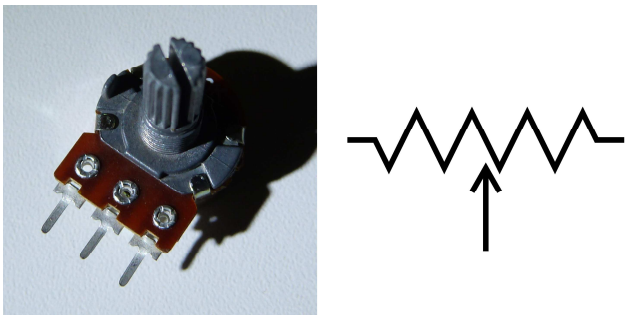


Figura 6.23: Um potenciômetro de 10 kΩ e sua representação em circuitos eletrônicos.

Em um potenciômetro de 10 kΩ, a resistência entre o pino central e o pino de uma das extremidades varia entre 0 e 10 kΩ conforme o botão é girado. Dessa forma, é possível usar esse dispositivo para obter uma saída de tensão ajustável quando conectamos uma tensão positiva em um dos pinos das extremidades e o terra no pino da outra extremidade, enquanto usamos o pino central como uma fonte de tensão que pode ser regulada. Assim, se usarmos 5 V e o terra nas extremidades, teremos uma tensão ajustável entre 0 e 5 V.

No projeto esquematizado abaixo usamos o raciocínio discutido no parágrafo anterior para ler a tensão gerada pelo pino central de um potenciômetro a partir de uma das portas de entrada analógica do Arduino (escolhemos a `A0`).

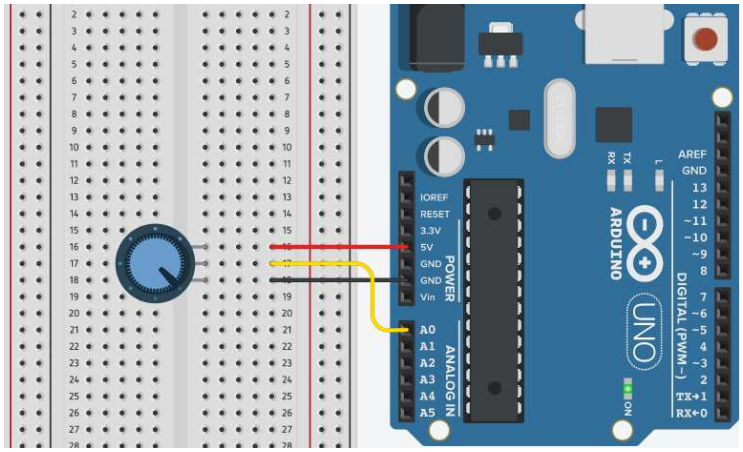


Figura 6.24: Conexões entre o potenciômetro, a matriz de linhas e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

O código, que está discutido abaixo, é simples. Declaramos duas variáveis (**valor** e **tensao** - linhas 1 e 2) e habilitamos a porta `A0` para entrada de dados (linha 7). Em seguida, armazenamos a tensão lida pela porta `A0` na variável **valor** (linha 12) e convertemos para uma tensão entre 0 e 5 V, usando para isso a variável **tensao** (linha 13). Por fim, mandamos imprimir o valor da variável **tensao** no monitor serial (linhas 15-17).

```
1  int valor;
2  float tensao;
3
4  void setup()
5  {
6    Serial.begin(9600);
```

```
7     pinMode(A0, INPUT);
8   }
9
10  void loop()
11  {
12    valor = analogRead(A0);
13    tensao = float(valor)*5/1023;
14
15    Serial.print("Tensão = ");
16    Serial.print(tensao);
17    Serial.println(" V");
18  }
```

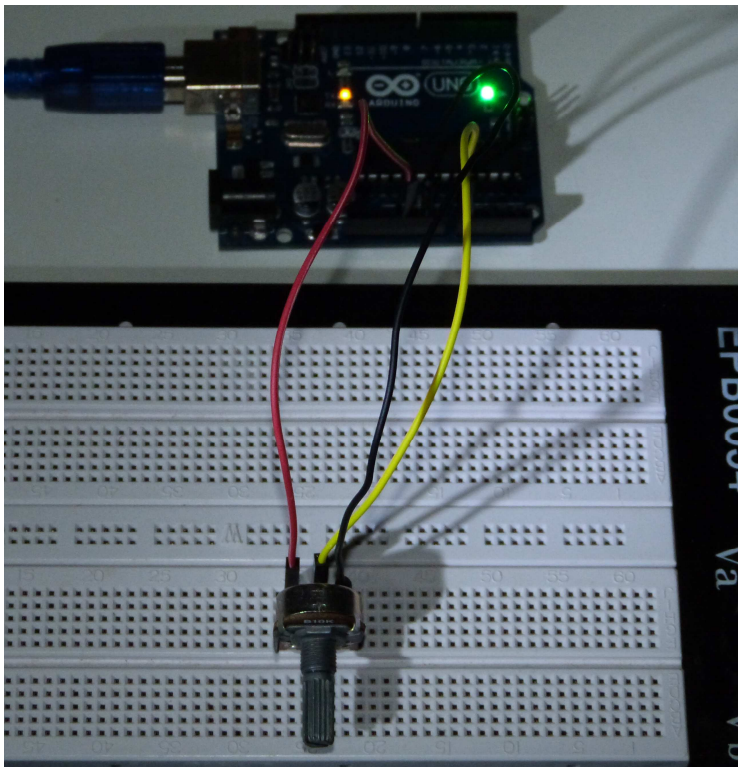


Figura 6.25: Arduino executando o código acima.

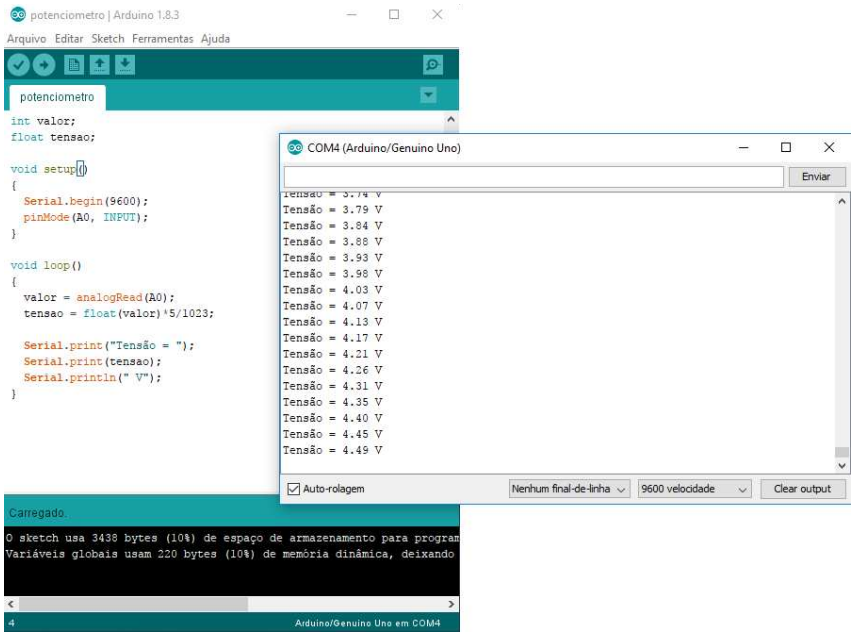


Figura 6.26: Conforme giramos o botão do potenciômetro, podemos observar a tensão sendo variada com o auxílio do monitor serial.

6.6.8 PZTs

O PZT (ou piezo) é uma cerâmica que tem propriedades piezoelétricas. Esses materiais sofrem expansão quando uma tensão elétrica é aplicada sobre ele e, de forma inversa, geram tensão quando ele é submetido a uma pressão. Abaixo temos um PZT típico facilmente encontrado no mercado.

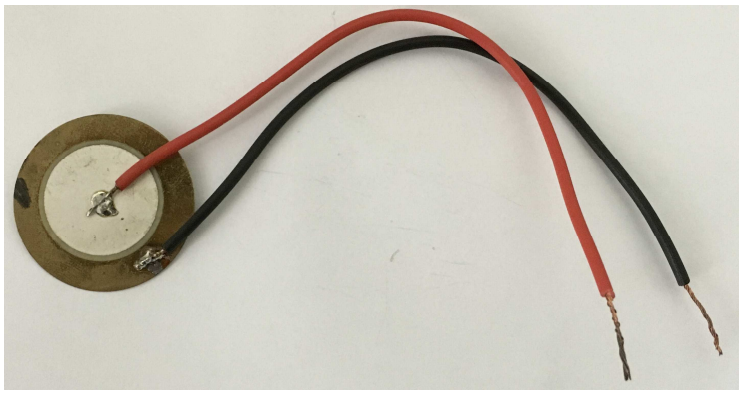


Figura 6.27: PZT com fios conectando seus terminais.

7. Projetos envolvendo Luz

Abordaremos aqui três projetos:

- Sincronizando LEDs
- Sensor de luminosidade com LDR's
- Sensor de luminosidade fototransistores
- Controlando a luminosidade de um LED

7.1 Sincronizando LEDs

Nesse projeto vamos colocar cinco LEDs piscando em sincronia, da seguinte forma: o primeiro LED acenderá enquanto os outros estarão apagados. Em seguida, o primeiro LED apaga enquanto o segundo acende, e assim sucessivamente, formando uma espécie de propagação da luz entre os LEDs, mantendo apenas um LED aceso por vez.

Primeiramente vamos fazer as conexões entre os LEDs, a matriz de linhas e as portas do Arduino. Como os LEDs são independentes, precisaremos de 05 portas de saída. Escolheremos as portas `11`, `9`, `7`, `5` e `2`. Cada perna maior dos LEDs deve estar conetado a uma dessas portas, mas as pernas menores podem compartilhar o mesmo terra. Então as conexões vão ficar como mostrado na figura abaixo.

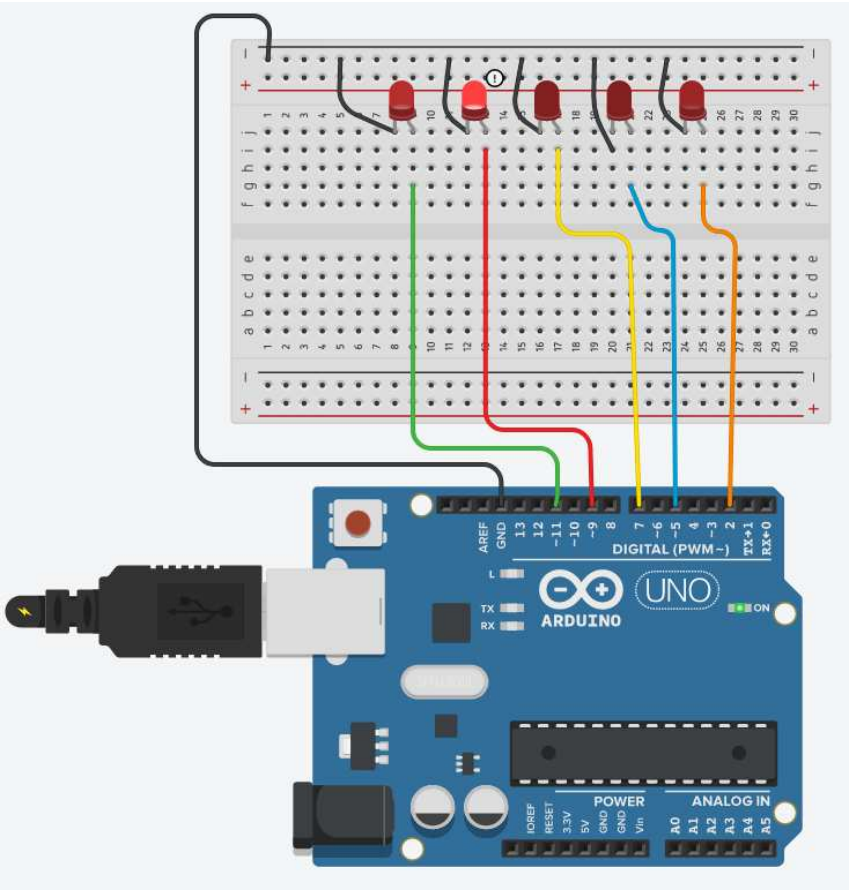


Figura 7.1: Conexões entre os LEDs, a matriz de linhas e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

Agora vamos decidir como escrever o código. Vamos definir como 200 ms o tempo no qual cada LED ficará aceso. Então teremos no código a seguinte linha de comando: `int tempo = 200;`. Os números das portas serão armazenados em um vetor de 05 componentes inteiras (ver Seção 5.11.1), de forma que colocaremos no código o comando `int pino[5];`. Para fazermos os LEDs acenderem alternadamente, colocaremos os comandos `digitalWrite(pino[k], HIGH);` e `digitalWrite(pino[k], LOW);` dentro de um loop `for`, deparados pela chamada da função `delay()`. O código completo segue abaixo.

```
1  int tempo = 200;
2  int pino[5];
3
4  void setup()
5  {
6    pino[0] = 11;
7    pino[1] = 9;
8    pino[2] = 7;
9    pino[3] = 5;
10   pino[4] = 2;
11
12   for(int k=0; k<=4; k++)
13   {
14     pinMode(pino[k], OUTPUT);
```

```
15      }
16    }
17
18    void loop()
19    {
20      for(int k=0; k<=4; k++)
21      {
22        digitalWrite(pino[k], HIGH);
23        delay(tempo);
24        digitalWrite(pino[k], LOW);
25      }
26    }
```

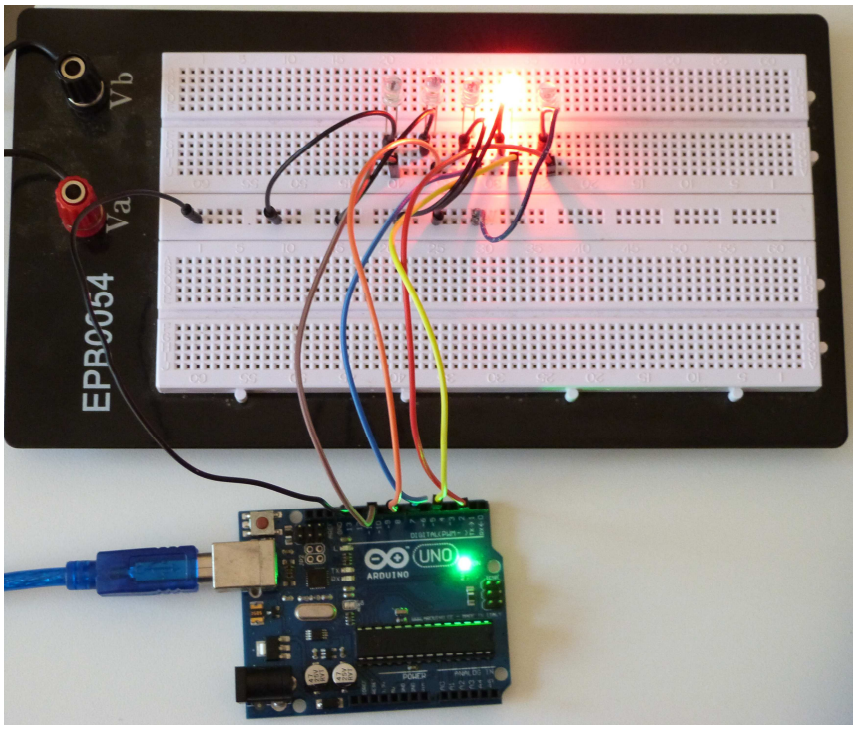


Figura 7.2: Arduino executando o código acima.

Exercício 7.1 Nesse projeto fizemos os LEDs acenderem alternadamente, mas apenas em um sentido. Modifique o código para observar o pulso de luz percorrer o LED nos dois sentidos de forma alternada, isto é, primeiro da esquerda para a direita, depois da direita para a esquerda, e assim por diante.

7.2 Sensor de luminosidade com um LDR

Nesse projeto faremos um sensor de luminosidade com o auxílio de um LDR (esse dispositivo foi abordado na Seção 6.6.5). Como o LDR é basicamente um resistor dependente da luminosidade, a diferença de potencial elétrico entre seus terminais varia dependendo da intensidade da luz que incide sobre ele. Então a ideia aqui é usar uma das portas de entrada analógica (como a `A1`) para ler a tensão em um de seus terminais, tendo como referência o terra. Essa tensão lida será, portanto, proporcional à intensidade da luz incidente no LDR.

Naturalmente, precisamos de um resistor para ser ligado em série com o LDR. Um dos terminais do resistor será ligado no terra (tensão 0) e o outro na porta A1 (tensão variável, dependendo da luz que incide no LDR). A figura abaixo ilustra as conexões. Uma escolha importante é o valor do resistor, que dará a sensibilidade do sensor. Tenha em mente que o LDR usado nesse projeto possui uma resistência de ~ 50 kΩ na luz ambiente, e de ~ 0 kΩ sob a luz de uma lanterna, de forma que o ideal é que o resistor tenha alguns kΩ de resistência. Escolhemos portanto um resistor de ≈ 2 kΩ.

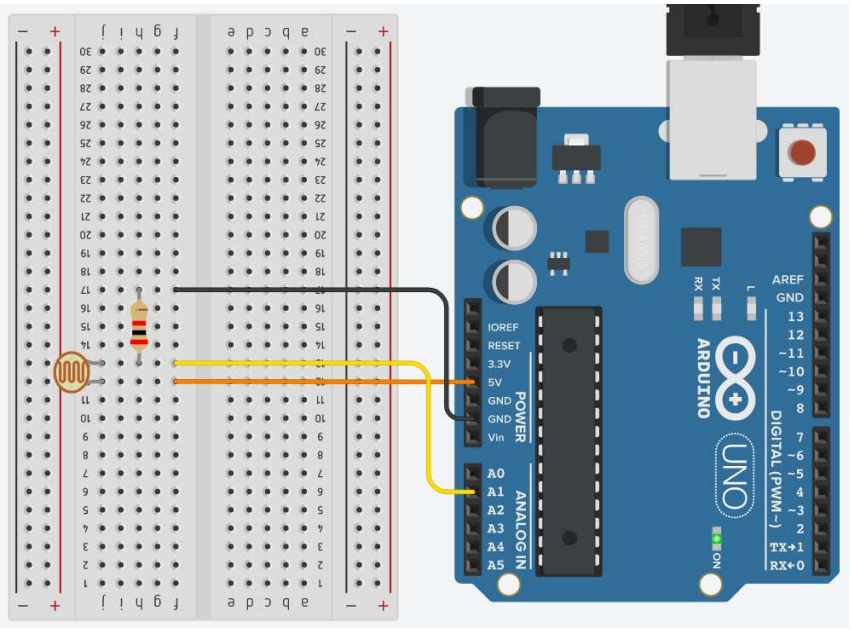


Figura 7.3: Conexões entre os LEDs, a matriz de linhas e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

O código desse projeto, que é bem simples, segue abaixo. Basicamente, a tensão lida na porta `A1` é armazenada na variável **valor** (linha 12) e, na linha 13, o valor armazenado nessa variável é exibida no monitor serial.

```
1  int porta = A1;
2  int valor;
3
4  void setup()
5  {
6    Serial.begin(9600);
7    pinMode(porta, INPUT);
8  }
9
10 void loop()
11 {
12   valor = analogRead(porta);
13   Serial.println(valor);
14 }
```

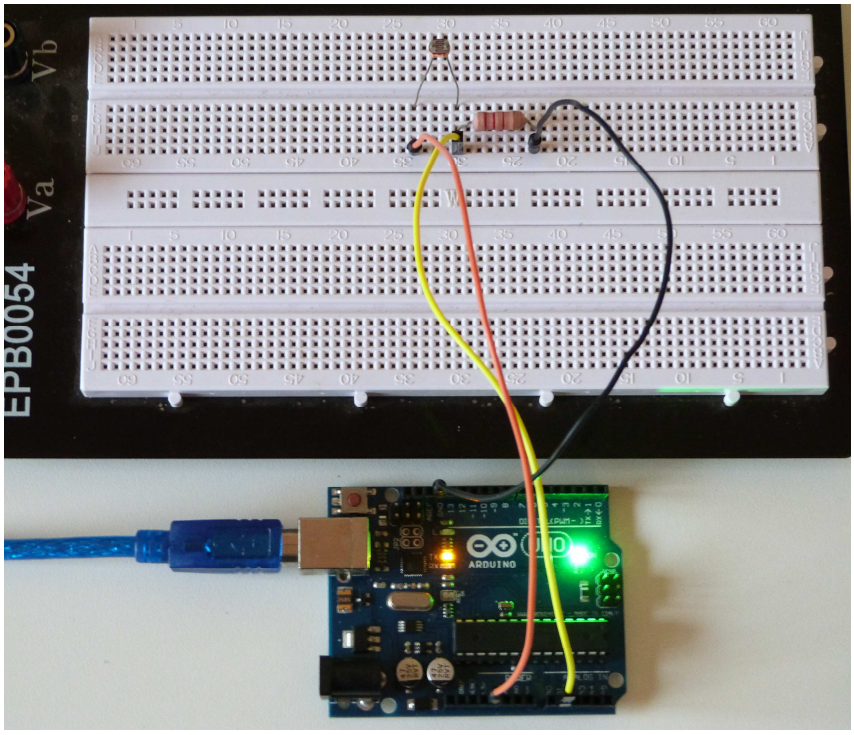


Figura 7.4: Arduino executando o código acima.

Exercício 7.2 Modifique o circuito acima para mudando o resistor por um potenciômetro, de forma que esse projeto possa funcionar em todos os ambientes mediante ajuste nesse componente.
Exercício 7.3 Modifique o código e o circuito acima para incluir um LED, que deve acender se o ambiente estiver escuro e apagar se o ambiente estiver claro. Como "claro" e "escuro" são bastante relativos, você deve decidir quando um ambiente pode ser considerado claro ou escuro (dica: crie uma variável para decidir entre a interface claro/escuro).

7.3 Sensor de luminosidade com um fototransistor

Agora faremos um projeto de um sensor de luminosidade usando o fototransistor PT334-6C da Everlight (ver Fig. 7.5). Como ele é um fototransistor de silício, sua faixa de operação está entre 400 e 1100 nm, com o pico em 940 nm, de forma que ele pode ser usado para detecção de luz visível.

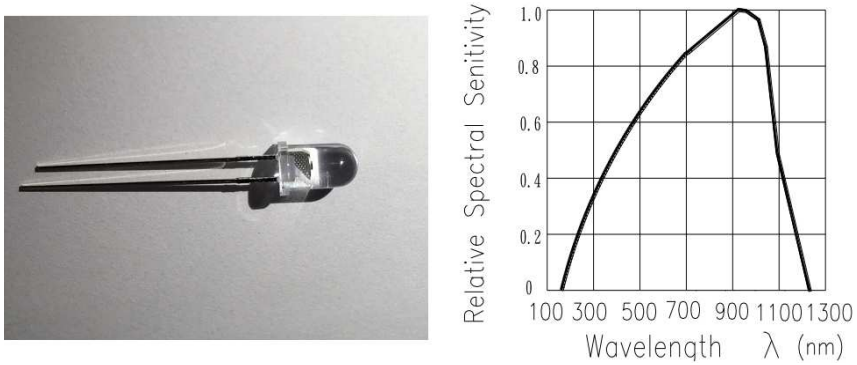


Figura 7.5: Fototransistor PT334-6C (à esquerda) e sua sensibilidade espectral (à direita). Fonte: *datasheet* em `http://www.everlight.com/file/productfile/pt334-6c.pdf`.

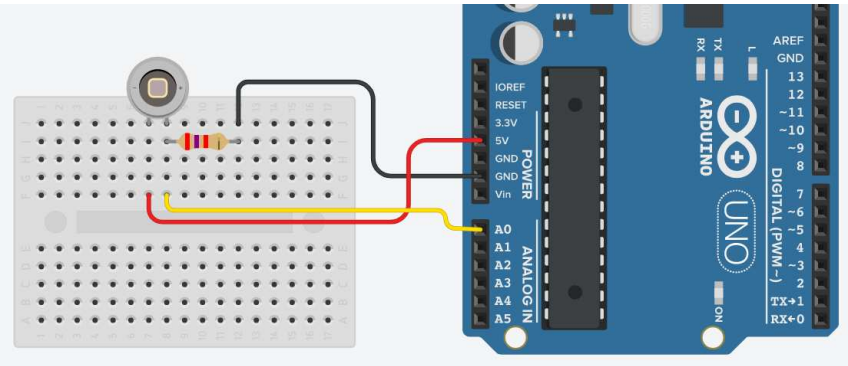


Figura 7.6: Conexões entre o fototransistor, a matriz de linhas, o resistor e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

Abaixo temos o esquema das ligações. Basicamente colocamos o fototransistor em série com um resistor de 2,7 kΩ, com a perna menor do fototransistor conectado na porta de 5 V do Arduino e o terminal do resistor no terra. Entre o resistor e o fototransistor lemos a tensão na porta `A0`.

E abaixo temos o código, que é bem simples. O valor lido pela porta `A0` é armazenado na variável **tensao** e posteriormente impressa no monitor serial.

```
1  int tensao;
2
3  void setup ()
4  {
5    pinMode(A0, INPUT);
6    Serial.begin(9600);
7  }
8  void loop ()
9  {
10   tensao = analogRead(A0);
11   Serial.println(tensao);
12   delay(100);
13 }
```

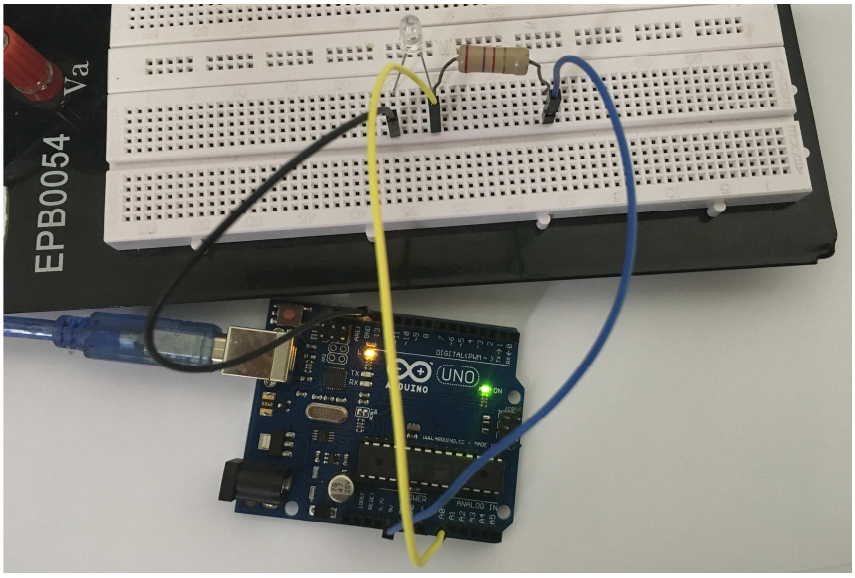


Figura 7.7: Arduino executando o código acima.

7.4 Controlando a luminosidade de um LED

Neste projeto vamos controlar a luminosidade de um LED usando uma das portas `PWM` do Arduino. Conforme visto na Seção 5.9.2, essas portas geram uma tensão modulada em uma dada frequência. Ainda que a amplitude de tensão continue sendo 5 V, a tensão média varia conforme variamos a largura dos pulsos. Em altas frequências, isto é, acima de $\sim$ 60 Hz, o olho humano não consegue distinguir entre um LED aceso ou apagado, de forma para nós o LED estará sempre aceso. A sensação de luminosidade do LED sera proporcional à tensão média entre os terminais do dispositivo.

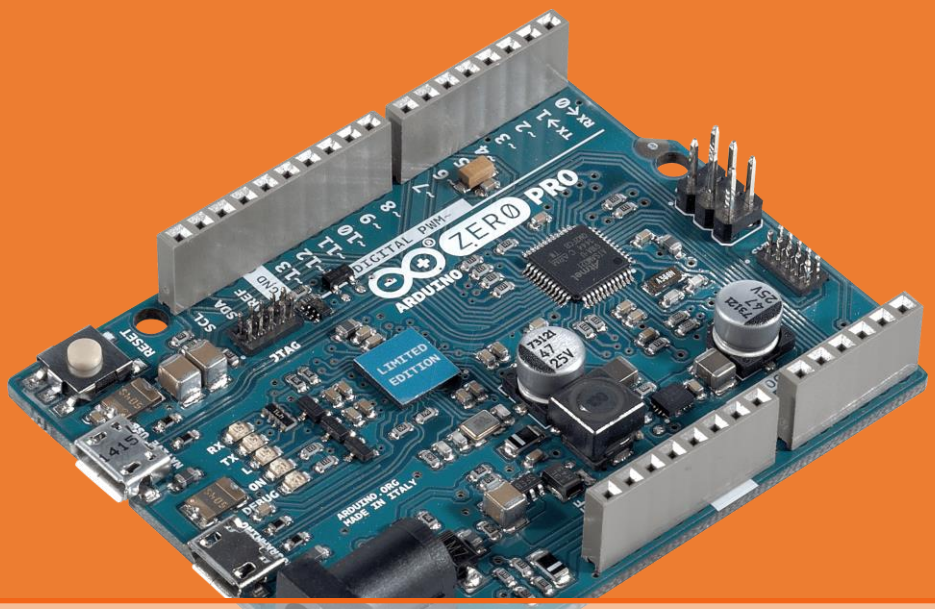
Esse projeto é extremamente simples. Basta ligar os terminais do LED em uma porta de saída `PWM` (vamos escolher a `11`) e o terra. Vamos fazer as ligações diretamente no Arduino, como mostrado na figura abaixo.

Agora vem a parte do código, que está mostrado abaixo. Vamos usar a função `Serial.parseInt()` para ler um inteiro que o usuário vai digitar e armazenar na variável **valor** (linha 18), converter o range dessa variável de 0 a 100 para 0 a 255 (que corresponde a 0 - 5 V, linha 23) e jogar na porta `PWM` (linha 24).

```
1   int valor;
```

```
2  int brilho;
3  int porta = 11;
4
5  void setup()
6  {
7    pinMode(porta, OUTPUT);
8    Serial.begin(9600);
9
10   Serial.println("Digite um valor entre 0 (brilho mínimo) e 100
       (brilho máximo):");
11   Serial.println();
12 }
13
14 void loop()
15 {
16   if (Serial.available() > 0)
17   {
18     valor = Serial.parseInt();
19
20     Serial.print("Valor lido: ");
21     Serial.println(valor);
22
23     brilho = valor*2.55;
24     analogWrite(porta, brilho);
25   }
26 }
```

Exercício 7.4 Modifique o código acima para fazer a luminosidade do LED variar periodicamente com o tempo. Dica: use a uma função periódica para isso.



8. Projetos envolvendo Temperatura

Neste capítulo apresentaremos projetos envolvendo temperatura e calor:

- Medindo temperatura com o Sensor LM35
- Medindo temperatura com o Sensor DS18B20
- Medindo umidade

8.1 Medindo temperatura com o Sensor LM35

Este projeto faz uso do sensor de temperatura LM35 (ver figura abaixo). O LM35 é um sensor de baixo custo; suas especificações técnicas estão mostradas na tabela abaixo:

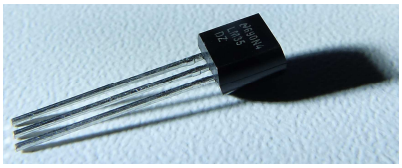


Figura 8.1: Sensor de temperatura LM35.

Tensão de alimentação:	4 a 30 V
Faixa de medição:	-55 ^{o}C a 150 ^{o}C
Precisão:	$\pm 0,5$
Corrente:	menor que 60 μA
Fator de escala:	10 mV por ^{o}C

Tabela 8.1: Fonte: *Datasheet* do sensor LM35: `http://www.ti.com/lit/ds/symlink/lm35.pdf`.

Há dois modos de operar esse sensor. No modo básico, que é o que usaremos aqui, o pino V_S (ver figura 8.2) deve ser conectado a uma tensão entre 4 e 20 V, o pino V_{OUT} é o pino de dados (que no nosso caso, conectaremos a uma das portas de entrada analógica do Arduino) e o pino GND ao terra. Esse modo de operação permite um *range* de temperatura entre 2 e 150 ^{o}C. O outro modo possui um range maior, porém necessita de um resistor específico para operar. Com essas informações, fizemos as conexões que estão ilustradas na figura 8.3: o pino V_S foi na porta de 5 V, o pino V_{OUT} foi em uma porta analógica (escolhemos a `A2`) e o pino GND foi em uma porta `GND` do Arduino.

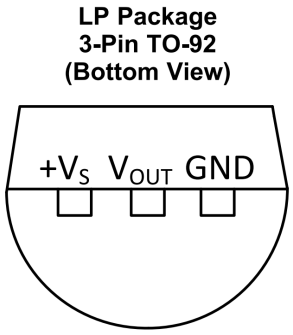


Figura 8.2: Terminais do sensor LM35. Note que essa é uma visão de baixo. V_S, V_{OUT} e GND se referem à tensão de alimentação, tensão de saída (dados) e ao terra, respectivamente. Fonte: Texas Instruments - `http://www.ti.com/lit/ds/symlink/lm35.pdf`.

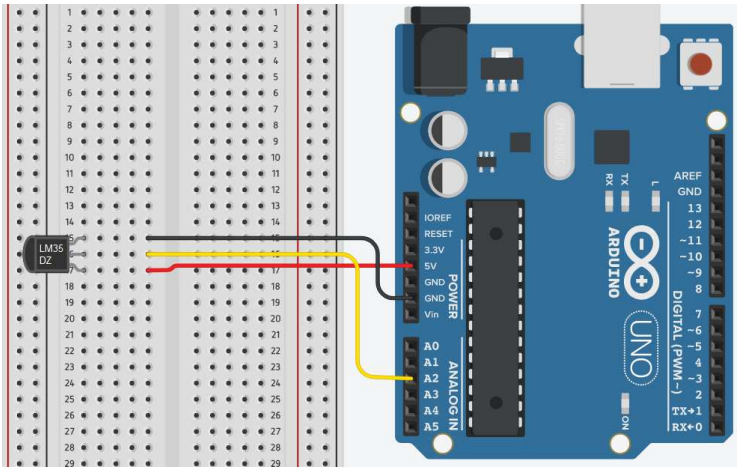


Figura 8.3: Conexões entre os terminais do sensor LM35DZ, a matriz de linhas e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

Agora passemos ao código, que é relativamente simples. Basta habilitar a porta `A2` para entrada de dados (`INPUT`) e armazenar em uma variável (criamos a variável **valor**) a tensão lida pelo pino V_{OUT} por meio da função `analogRead()`. O único trabalho é a conversão da tensão lida em temperatura. Pelo *datasheet*, sabemos que a tensão de saída aumenta em 10 mV para cada 1 ^{o}C de temperatura que aumenta. Sabemos, também, que o zero da função `analogRead()` corresponde a 0 V e 1023 corresponde a 5 V. Assim, a conversão entre tensão e temperatura é dada pela fórmula

$$\text{temperatura} = \left(\frac{100 \times 5}{1023}\right) \times \text{tensão}$$

Assim, apresentamos abaixo o código.

```
1  int porta = A2;
2  int tempo = 1000;
3  int valor;
4  float temperatura;
5
6  void setup()
7  {
8    Serial.begin(9600);
9    pinMode(porta, INPUT);
10 }
11
12 void loop()
13 {
14   valor = analogRead(porta);
15   temperatura = float(valor)*100*5/1023;
16
17   Serial.print("Temperatura = ");
18   Serial.print(temperatura, 1);
19   Serial.println(" oC");
20   delay(tempo);
21 }
```

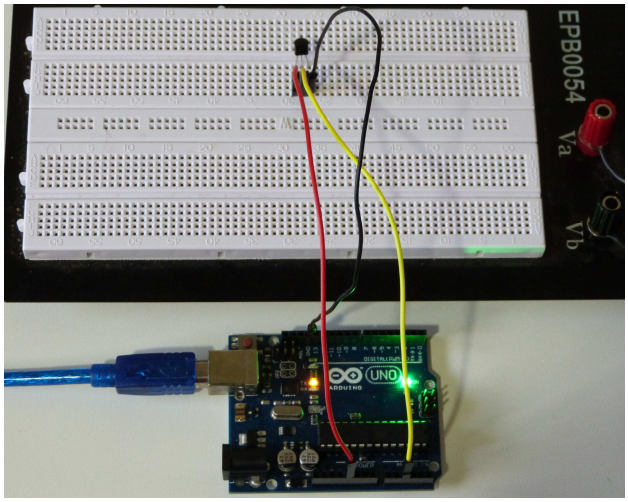


Figura 8.4: Arduino executando o código acima.

Exercício 8.1 Introduza um LED no circuito acima e faça uma modificação no código para que o LED acenda quando a temperatura medida ultrapasse 45^oC.

8.2 Medindo temperatura com o Sensor DS18B20

Este projeto faz uso do sensor de temperatura DS18B20 (ver figura abaixo). Esse sensor é baseado no chip DS18B20, e possui as seguintes especificações relevantes:

Tensão de alimentação:	3 a 5,5 V
Faixa de medição:	-55 ^{o}C a 125 ^{o}C
Precisão:	$\pm 0,5$ ^{o}C entre -10 ^{o}C e 85 ^{o}C
Fio vermelho:	Alimentação
Fio preto:	Terra
Fio amarelo:	Dados

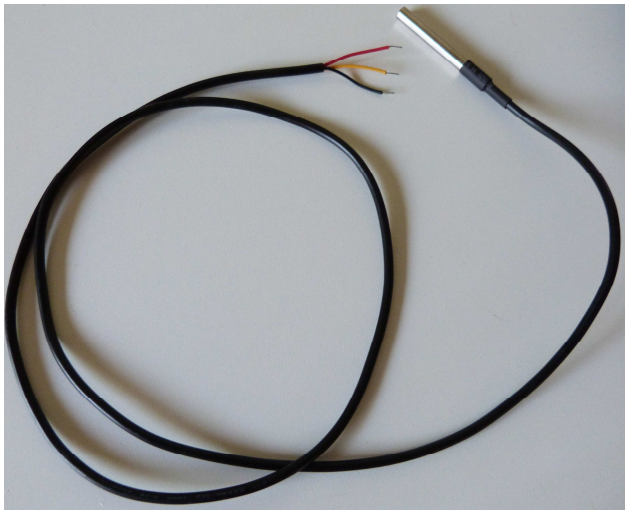


Figura 8.5: Sensor de temperatura DS18B20.

Mais detalhes podem ser consultados em seu *datasheet* no site `http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf`.

As conexões são relativamente simples. Como mostrado na tabela acima, o fio preto deve ir no terra (`GND`), o fio vermelho na alimentação em 5 V e o fio amarelo em uma porta de saída de tensão digital (escolhemos a `10`). Entre os fios amarelo e vermelho deve estar conectado um resistor de 4,7 kΩ (colocamos dois em série, sendo um de ≈ 3,7 kΩ e outro de ≈ 1,0 kΩ). Abaixo temos uma ilustração com essas conexões.

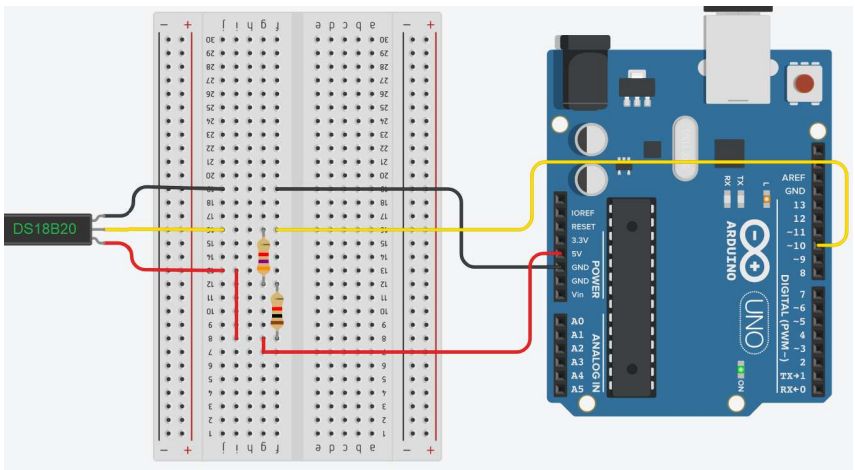


Figura 8.6: Conexões entre os fios do sensor DS18B20, a matriz de linhas, os resistores e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

Já a parte do código é bem mais complicada. O chip DS18B20 se comunica com o Arduino via duas bibliotecas: a `OneWire.h` e a `DallasTemperature.h`. Conforme mencionado na Seção 5.10, essas duas bibliotecas devem ser baixadas (`http://www.hacktronics.com/code/DallasTemperature.zip` e `http://www.pjrc.com/teensy/arduino_libraries/OneWire.zip`) e colocadas na pasta de bibliotecas do Arduino em seu computador. Essas duas bibliotecas trabalham em conjunto em dispositivos feitos pela Maxim/Dallas, como sensores de temperatura e botões e memória.

Abaixo mostramos o código. As instruções dentro da função `void` servem para obter o endereço dos fios do sensor de temperatura.

```
1  #include <OneWire.h>
2  #include <DallasTemperature.h>
3
4  int porta = 10;   //Porta do fio amarelo (dados)
5  int tempo = 2000; //tempo de atualização da temperatura
6
7  //Instruções ligadas às bibliotecas acima
8  OneWire oneWire(porta);
9  DallasTemperature sensors(&oneWire);
10 DeviceAddress sensor1;
11
12 void setup()
13 {
14   Serial.begin(9600);
15
16   sensors.begin();
17   Serial.println("Localizando␣sensor␣DS18B20␣...");
18   if (!sensors.getAddress(sensor1, 0))
19      Serial.println("Sensor␣não␣encontrado!");
20   Serial.print("Endereço␣do␣sensor:␣");
21   enderecoSensor(sensor1);
22   Serial.println();
23   Serial.println();
24 }
25
26 //função que escreve o endereço do sensor
27 void enderecoSensor(DeviceAddress deviceAddress)
28 {
29   for (int k=0; k<8; k++)
30   {
31     if (deviceAddress[k]<16) Serial.print("0");
32     Serial.print(deviceAddress[k], HEX);
33   }
34 }
35
36 void loop()
37 {
38   sensors.requestTemperatures();
39   float temperatura = sensors.getTempC(sensor1);
40
```

```
41    Serial.print("Temperatura: ");
42    Serial.print(temperatura, 1);
43    Serial.println(" oC");
44
45    delay(tempo);
46  }
```

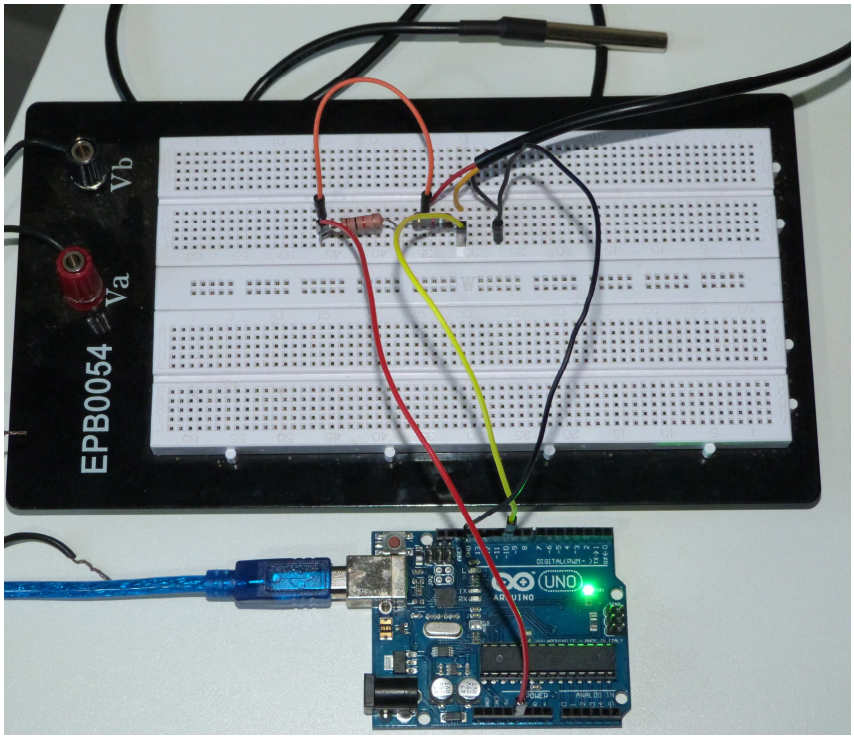


Figura 8.7: Arduino executando o código acima.

Exercício 8.2 Introduza um LED no circuito acima e faça uma modificação no código para que o LED acenda quando a temperatura medida ultrapasse 45^oC.

8.3 Medindo umidade

Neste projeto usaremos o sensor de umidade higrômetro para monitorar a umidade de um determinado meio (ver Fig. 8.8). Ele consiste de uma sonda e de um chip comparador, modelo LM393, que possui dois comparadores de tensão independentes, conformes especificações do seu *datasheet*.

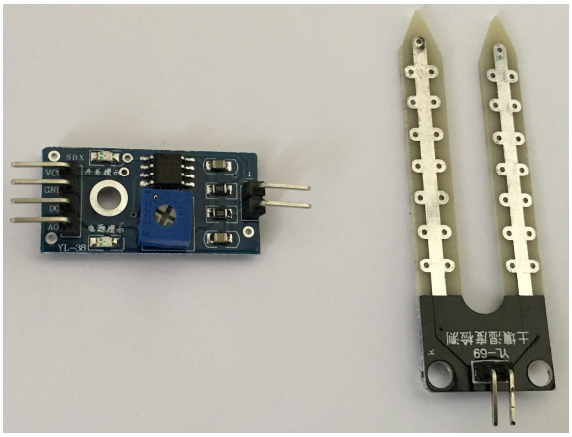


Figura 8.8: Sensor de umidade, contendo o chip comparador (à esquerda) e a sonda (à direita).

O esquema das ligações é bem simples (ver Fig. 8.9). A dupla de pinos do chip comparador deve ser ligado aos terminais da sonda, enquanto que três do conjunto de quatro pinos devem ser ligados às portas da placa Arduino, seguindo a nomenclatura que é bastante óbvia: o `VCC` é o pino positivo de alimentação (5 V), `GND` é o terra e `A0` é o pino para leitura dos dados.

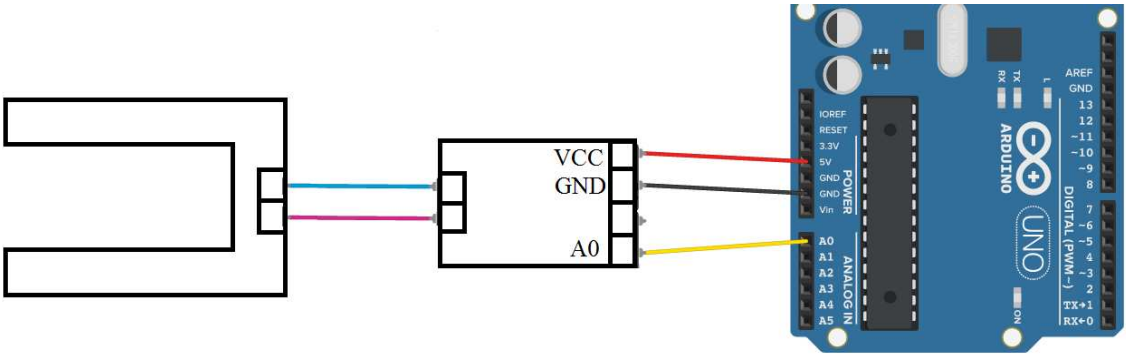


Figura 8.9: Diagrama das ligações entre o sensor de umidade e o Arduino.

O código também é extremamente simples, como pode ser visto abaixo. Basicamente, a leitura é feita com a função `analogRead()` e posteriormente armazenada na variável **valor**. Em seguida, é feita uma conversão para que o intervalo da variável **valor** (0 a 1023, devido ao retorno da função `analogRead()`) seja convertido em um intervalo de 0 a 100, com esses extremos denotando umidade nula e altíssima umidade, respectivamente. Como pôde ser observado na linha `13`, houve uma inversão do valor lido, porque o sensor é programado de fábrica para retornar um valor alto em baixa umidade e um valor baixo em alta umidade.

Há duas formas de regular a sensibilidade da leitura da umidade. A primeira é usando o potenciômetro do chip comparador, que pode ser ajustado com uma chave estrela. A outra forma é alterando o valor `100` da linha `13` do programa.

```
1  int valor;
2  float umidade;
3
4  void setup()
5  {
```

```
6     pinMode(A0, INPUT);
7     Serial.begin(9600);
8   }
9
10  void loop()
11  {
12    valor = analogRead(A0);
13    umidade = 100-valor*100.0/1023.0;
14
15    Serial.print("Umidade: ");
16    Serial.println(umidade);
17  }
```

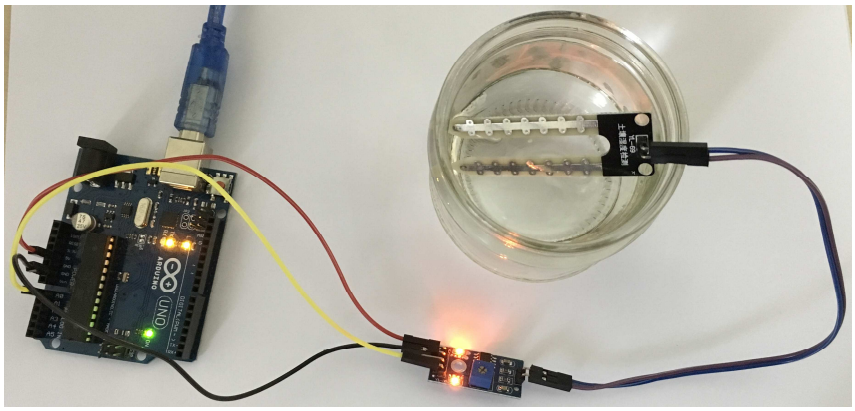
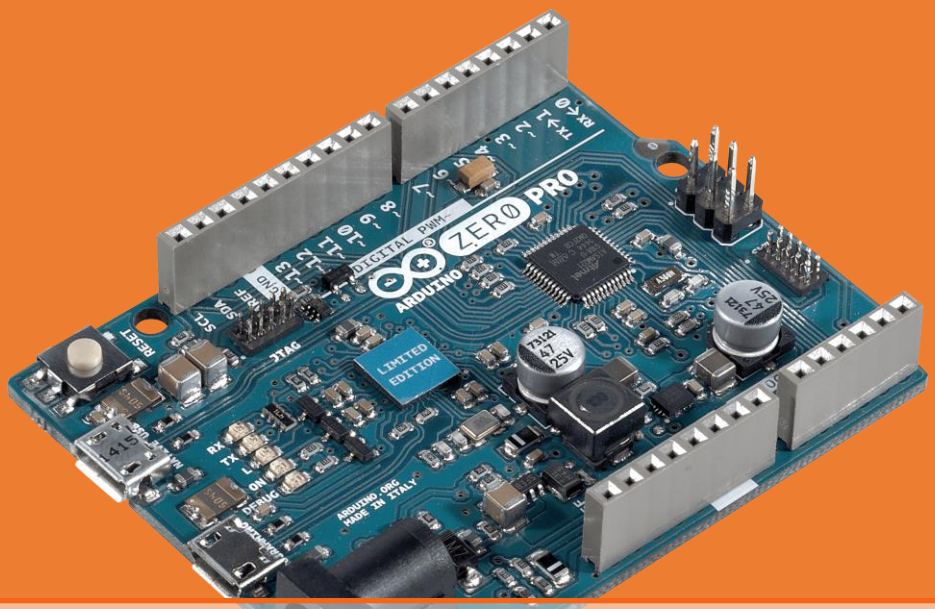


Figura 8.10: Arduino executando o código acima.



9. Mais opções de saída de dados

Abordaremos aqui os seguintes dispositivos:

- Monitor LCD
- Leitor de cartão SD

9.1 O monitor LCD

O monitor LCD, ou display LCD, é uma excelente alternativa ao uso do monitor serial, visto que possibilita uma independência quase completa do Arduino em relação ao computador. Neste projeto, usaremos um dos mais simples displays LCD, com tamanho de 16×2 caracteres (16 colunas e duas linhas - ver figura abaixo).

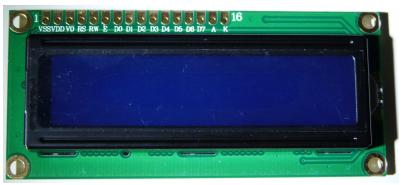


Figura 9.1: Monitor LCD tamanho 16×2.

Esse display possui um total de 16 pinos, sendo que neste projeto usaremos apenas 12 deles. Na tabela abaixo podemos ver as funções desses pinos. Basicamente há dois pinos de alimentação em 5 V junto com dois pinos terra, um pino que controla a leitura e a escrita de dados, oito pinos responsáveis pelos 8 bits dos caracteres a serem escritos na tela do LCD e um pino responsável pela regulagem do contraste da tela.

Pino	VSS	VDD	V0	RS	RW	E	A	K
Função	Terra	5 V	Contraste	Registro	Leitura/escrita	Habilitar	5 V	Terra

Pino	D0	D1	D2	D3	D4	D5	D6	D7
Função	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7

Com base nessas informações, mostramos abaixo as conexões necessárias para exibir um texto simples no display LCD. Note que colocamos um potenciômetro conectado ao pino V0 para uma regulagem do contraste do display (o que não é obrigatório). Além disso, deixamos metade dos pinos dos bits (D0 a D3) desconectados.

- VSS - `GND`
- VDD - `5V`
- V0 - Pino central de um potenciômetro
- RS - Porta `12`
- RW - GND
- E - Porta `11`
- D0 - Sem conexão
- D1 - Sem conexão
- D2 - Sem conexão
- D3 - Sem conexão
- D4 - Porta `5`
- D5 - Porta `4`
- D6 - Porta `3`
- D7 - Porta `2`
- A - `5V`
- K - `GND`

Abaixo mostramos uma ilustração dessas conexões.

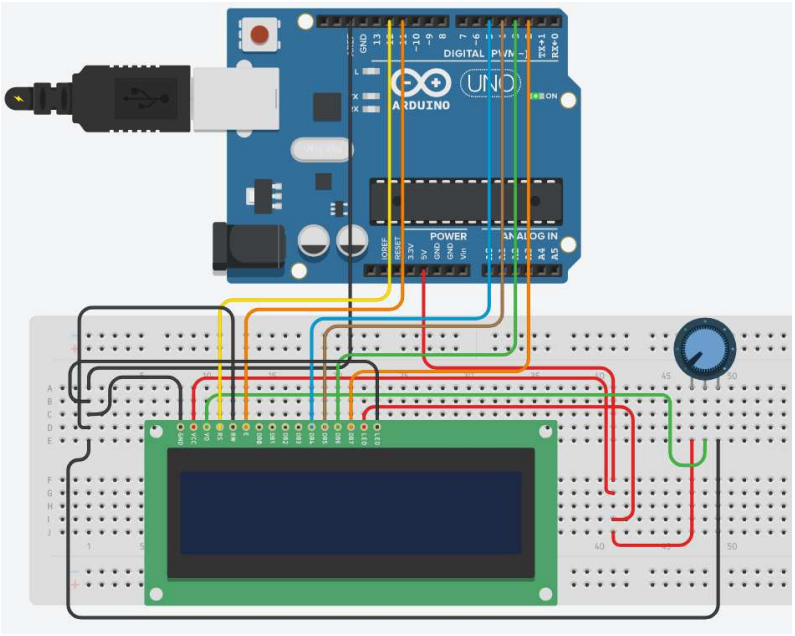


Figura 9.2: Conexões entre o monitor LCD, a matriz de linhas, o potenciômetro e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

Abaixo mostramos o código. Para trabalhar com o display LCD é necessário o uso da biblioteca `LiquidCrystal.h`, que é padrão do Arduino (não precisa ser instalada). Nas linhas `3-8` escolhemos as portas do Arduino e na linha `10` chamamos a função `LiquidCrystal` (que está contida na biblioteca mencionada). Dentro da função `setup()` temos 3 chamadas de função diferentes: `lcd.begin()`, `lcd.print()` e `lcd.setCursor()`. As duas primeiras são equivalentes às já conhecidas funções `Serial.begin()` e `Serial.print()` (ver Seção 5.5). Já a última serve para posicionar o cursor em um dado ponto do display. Fizemos uso dela para centralizar a string **"DEFIJI - UNIR"** na linha de baixo da tela, deslocando a linha e a coluna em uma unidade em relação ao ponto (0, 0) - ver figura 9.3.

```
1  #include <LiquidCrystal.h>
2
3  int RS = 12;
4  int EN = 11;
5  int D4 = 5;
6  int D5 = 4;
7  int D6 = 3;
8  int D7 = 2;
9
10 LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);
11
12 void setup()
13 {
14   lcd.begin(16, 2);
15   lcd.print("Curso de Arduino");
16   lcd.setCursor(1, 1);
17   lcd.print("DEFIJI - UNIR");
18 }
19
```

```
20  void loop()
21  {
22
23  }
```

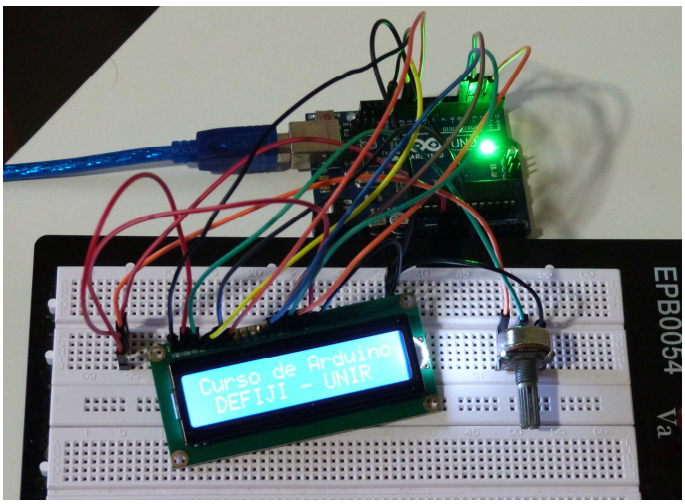


Figura 9.3: Arduino executando o código acima.

Exercício 9.1 Modifique o código acima para que o Arduino crie uma palavra (a sua escolha) que se mova no display LCD.

9.2 O leitor de cartão SD

Se com o display LCD conseguimos uma certa independência de um computador, com o leitor de cartão conseguimos uma razoável independência de uma pessoa por perto. Isso porque um dos usos do leitor de cartão é gravar dados que necessitem de um longo período para serem finalizados, como, por exemplo, a coleta da temperatura a cada hora do dia em uma cidade.

Neste projeto usaremos o MicroSD Card Adapter, mostrado na figura abaixo. Ele possui seis pinos, sendo dois para alimentação (5 V) e quatro para transferência de dados. O esquema das ligações, mostrado na tabela abaixo, segue a biblioteca `SD.h`, que é padrão do Arduino e portanto não necessita de instalação. Das quatro portas de dados, apenas a porta `CS` pode ser escolhida para ser diferente; todas as outras devem seguir a tabela.

Pino do Cartão	MISO	MOSI	SCK	CS
Porta do Arduino 12	11	SCK 13	digital à escolha	

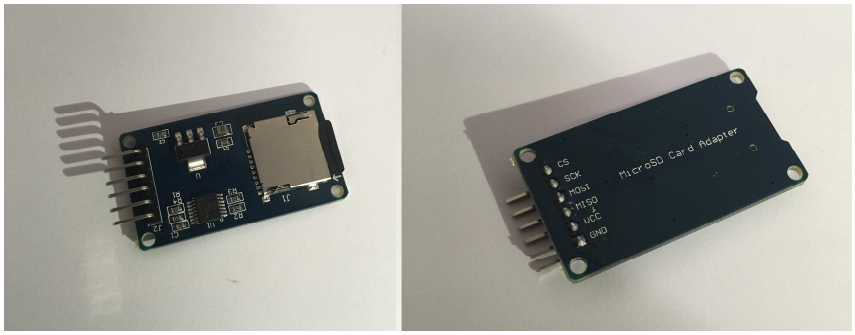


Figura 9.4: Adaptador para cartão micro SD usado no projeto.

Abaixo mostramos o diagrama das ligações.

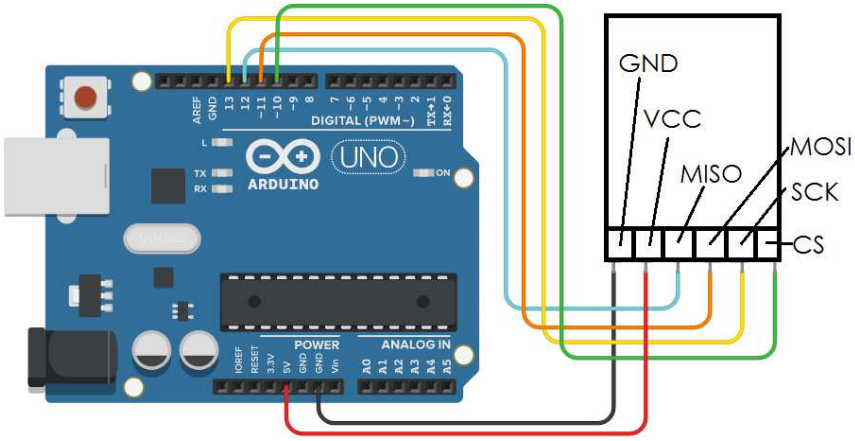


Figura 9.5: Diagrama das ligações entre o Adaptador de cartão micro SD e as portas do Arduino.

Agora vamos ao código. Para trabalharmos com arquivos, precisamos do seu nome interno e externo. O nome interno é usado apenas dentro do código, enquanto que o externo é o nome do arquivo onde os dados serão lidos/gravados, por exemplo, `texto.txt`. Para o nome interno usamos a sintaxe abaixo, que deve ser declarado, em geral, fora das funções `setup()` e `loop()`:

```
1  File nome-interno;
```

Para habilitar o uso do cartão SD, usamos a instrução abaixo:

```
1  SD.begin(10);
```

que tem o papel análogo ao de `Serial.begin()` para o monitor serial.

9.2.1 Gravando dados no cartão

Para gravar dados, primeiramente habilitamos o arquivo para escrita, conforme a sintaxe abaixo:

```
1  nome-interno-escolhido = SD.open("nome-externo.extensão",
       FILE_WRITE);
```

onde precisamos do nome interno declarado previamente e do nome externo do arquivo. Em seguida, as informações já podem ser gravadas no arquivo com a sintaxe abaixo:

```
1  nome-interno-escolhido.println("Algum texto ou variável ...");
```

Por fim, o arquivo precisa ser fechado de acordo com a seguinte sintaxe:

```
1   arquivo.close();
```

Abaixo mostramos o código completo, que cria um arquivo com nome externo `texto.txt` e grava nele as frases "Olá Arduino!" e "Funciona!". Após executar o código, confira o conteúdo do arquivo no seu computador, conectando o cartão de memória nele. Deverá haver o arquivo `texto.txt`.

```
1   #include<SD.h>
2
3   File arquivo;
4
5   void setup()
6   {
7     Serial.begin(9600);
8     SD.begin(10);
9
10    arquivo = SD.open("texto.txt", FILE_WRITE);
11    arquivo.println("␣");
12    arquivo.println("Olá␣Arduino!");
13    arquivo.println("Funciona!");
14    arquivo.close();
15  }
16
17  void loop(void)
18  {
19
20  }
```

9.2.2 Lendo dados do cartão

Algumas vezes precisamos ler os dados previamente gravados no cartão, e em seguida exibi-los no monitor serial ou em um display LCD. O código completo para a leitura de um arquivo chamada `dados.txt` se encontra abaixo. Antes de tudo, é preciso criar um arquivo no cartão SD com o nome `dados.txt`, e é preciso inserir algum conteúdo nele. Dessa vez, escolhemos o nome interno **arq** para o arquivo. Observe que o código para a leitura é um pouco mais elaborado, visto que precisamos ler caractere por caractere. Assim, os comandos das linhas `12-15` escrevem no monitor serial os caracteres lidos enquanto ainda existirem caracteres no arquivo.

```
1   #include<SD.h>
2
3   File arq;
4
5   void setup()
6   {
7     Serial.begin(9600);
8     SD.begin(10);
9
10    arq = SD.open("dados.txt");
11
12    while (arq.available())
13    {
14      Serial.write(arq.read());
```

```
15      }
16
17      arq.close();
18  }
19
20  void loop(void)
21  {
22
23  }
```

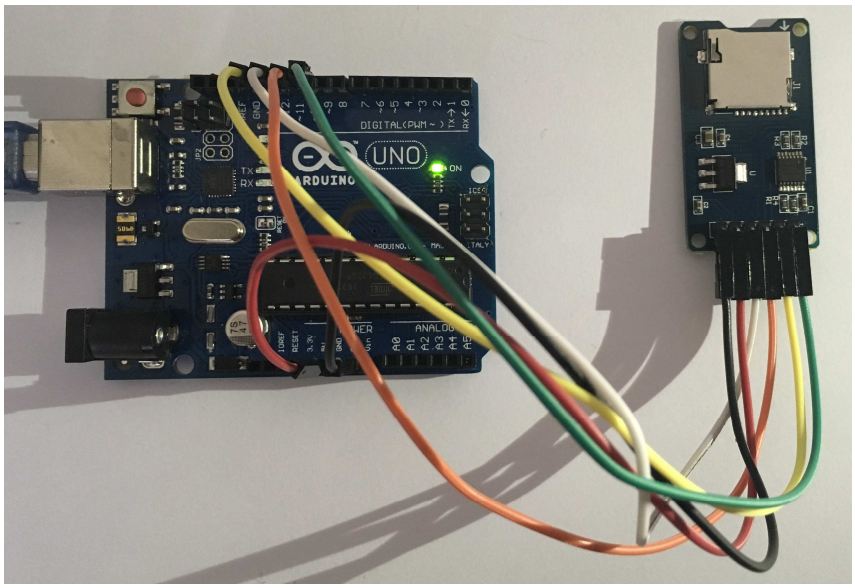
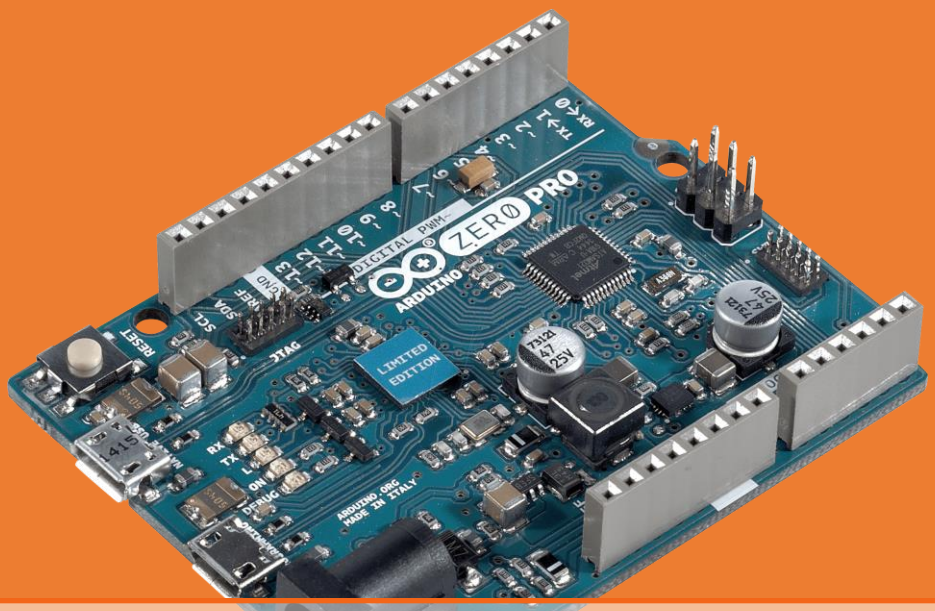


Figura 9.6: Arduino executando os códigos acima.

9.3 O *shield* wifi

9.4 O *shield* bluetooth



10. Projetos envolvendo Movimento

Abordaremos aqui os seguintes projetos:

- Medindo distância entre objetos
- Gerando sons com o PZT
- Movimento com o motor de passo

10.1 Medindo distância entre objetos

Este projeto usa o sensor de movimento ultrassônico modelo HC-SR04 (ver Fig. 10.1). Segundo o seu *datasheet*, ele funciona com base em uma alimentação em 5 V e é capaz de medir desde 2 cm até 4 m de distância, com incerteza de 3 mm. Seu princípio de funcionamento é razoavelmente simples: um emissor (cilindro esquerdo, `T`) emite pulsos sonoros curtos ($\sim$ 10 μs) em uma frequência de 40 kHz (ultrassom) que batem no objeto a ser medido e são refletidos, sendo captados pelo receptor (cilindro esquerdo, `R`). A distância é medida através da velocidade do som no ar, 340 m/s, e do tempo gasto no percurso.

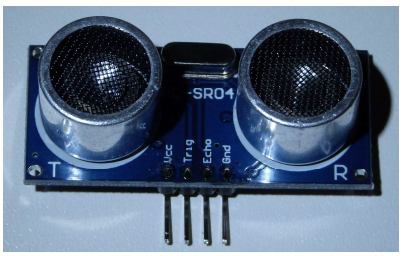


Figura 10.1: O *shield* ultrassônico modelo HC-SR04.

Abaixo temos o diagrama das ligações. O pino `VCC` deve ser ligado na porta de 5 V do Arduino, o `GND` no terra e pinos `Trig` e `Echo` em duas portas de saída digital.

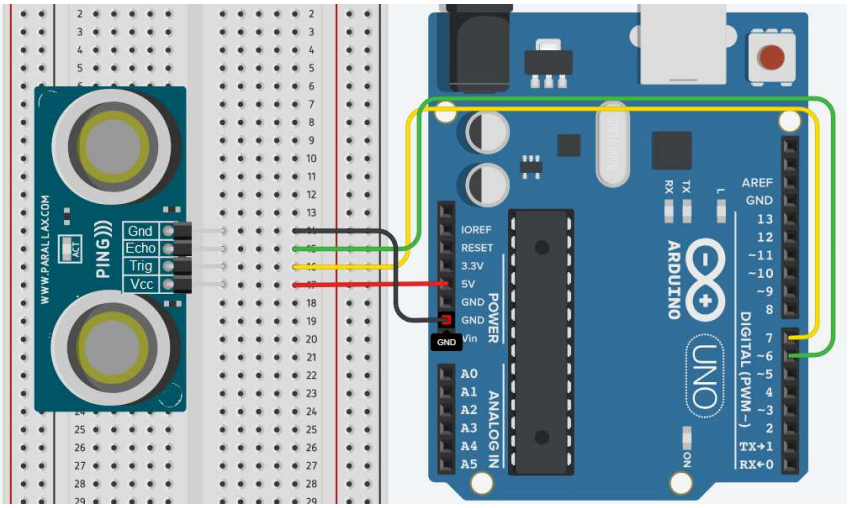


Figura 10.2: Conexões entre o *shield* ultrassônico, a matriz de linhas e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

O código, mostrado abaixo, faz uso da biblioteca `Ultrasonic.h`, que deve ser baixada e instalada na pasta `libraries` do Arduino, conforme instruções da Sec. 5.10. Primeiramente definimos as portas do Arduino que serão conectadas com os pinos `Trig` e `Echo`, o que é feito com a chamada da função `ultrasonic()`. Em seguida, o intervalo de tempo decorrido entre a emissão e a detecção dos pulsos ultrassônicos é medido através da função `ultrasonic.timing()`, que por fim é usada para medir a distância. O código completo segue abaixo.

```
1  #include <Ultrasonic.h>
2
3  int portaTrig = 7;
4  int portaEcho = 6;
5  int tempo;
6  float distancia;
7
8  Ultrasonic ultrasonic(portaTrig, portaEcho);
9
10 void setup()
11 {
12   Serial.begin(9600);
13 }
14
15 void loop()
16 {
17   tempo = ultrasonic.timing();
18   distancia = ultrasonic.convert(tempo, Ultrasonic::CM);
19
20   Serial.print("Distância = ");
21   Serial.print(distancia);
22   Serial.println(" cm");
23   delay(500);
24 }
```

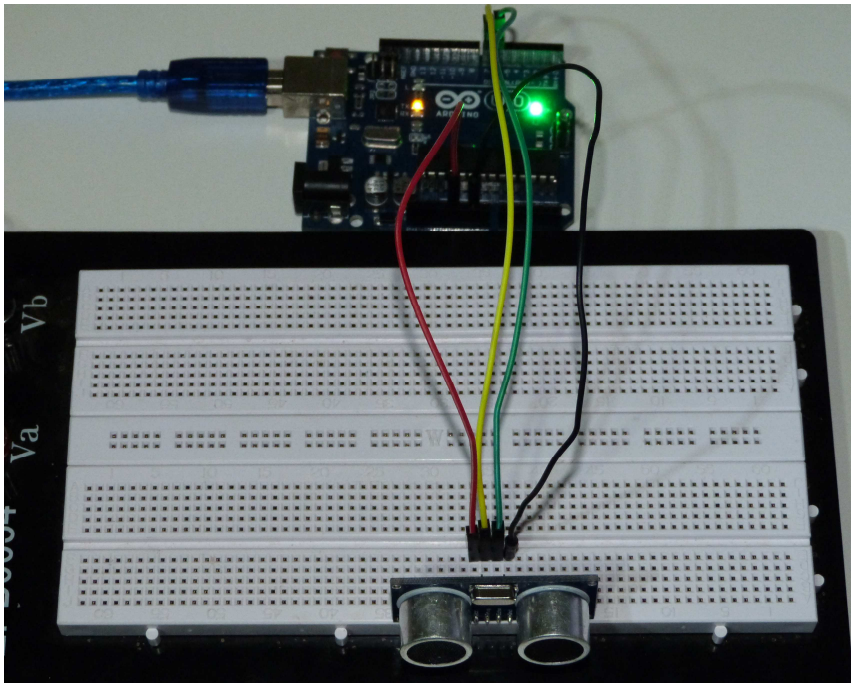


Figura 10.3: Arduino executando o código acima.

Exercício 10.1 Inclua um LED no circuito acima e modifique o código para que esse LED acenda toda vez que um objeto ficar a menos de 5 cm de distância do sensor.

10.2 Gerando sons com o PZT

Neste projeto usaremos um PZT (ver Sec. 10.2) para gerar sons com o auxílio do Arduino. A ideia é inserir uma tensão modulada nos terminais do PZT, de forma que a sua vibração gere uma onda de pressão que se propague no ar, chegando aos nossos ouvidos. Para isso, tenha em mente que a audição humana é capaz de detectar sons entre 20 Hz (grave, fronteira com o infrassom) e 20 kHz (agudo, fronteira com ultrassom) de frequência.

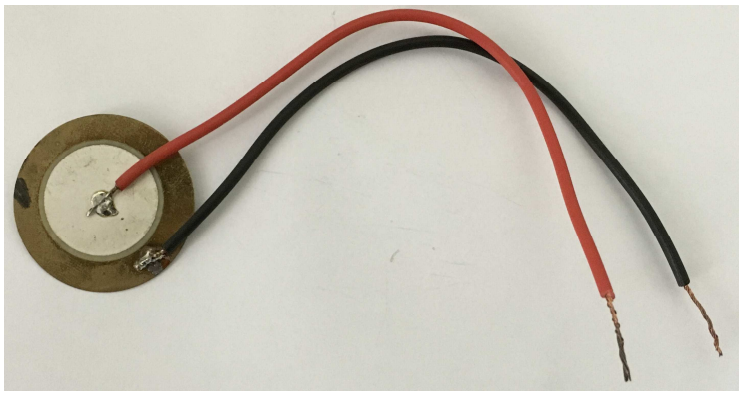


Figura 10.4: PZT com fios conectando seus terminais.

A ideia é usar um potenciômetro para sintonizar a frequência do som. O esquema das ligações segue na Fig. 10.5. Os terminais do PZT são conectados nas portas `13` e `GND` do Arduino, e um

potenciômetro é usado para colocar uma tensão específica (entre 0 e 5 V) na porta de entrada analógica `A5`. A tensão nessa porta é usada para que possamos escolher a frequência da modulação da tensão no PZT, de forma a produzir diversos tipos de sons. O código completo está mostrado abaixo.

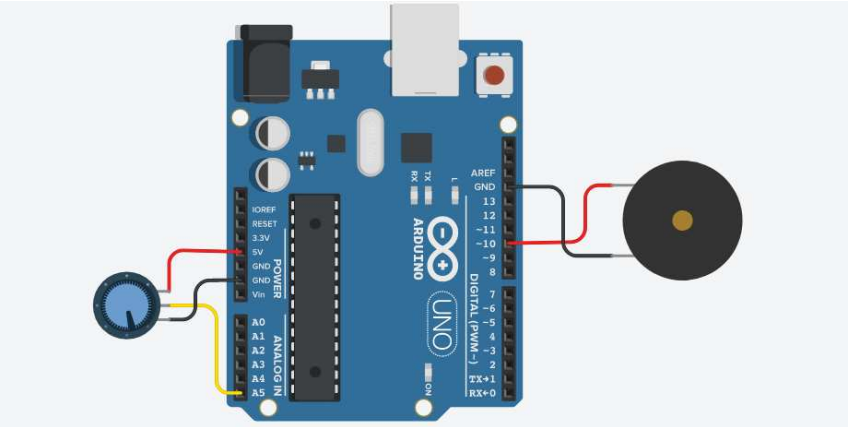


Figura 10.5: Esquema das ligações do Arduino com o PZT (à direita) e o potenciômetro (à esquerda). Desenhado no site `https://www.tinkercad.com`.

Agora vamos ao código, que é bastante simples. O valor da tensão lida pelo pino central do potenciômetro é armazenado na variável **valor**, que é usada como argumento da função `delayMicroseconds()`. Dessa forma, é possível regularmos a frequência da modulação da tensão. Como a função `analogRead()` retorna um valor inteiro entre 0 e 1023, a frequência da onda sonora gerada vai estar compreendida entre 489 Hz e 500 kHz (inaudível), seguindo uma função hiperbólica com o ângulo de giro do potenciômetro.

```
1  int valor;
2
3  void setup()
4  {
5    pinMode(10, OUTPUT);
6    pinMode(A5, INPUT);
7    Serial.begin(9600);
8  }
9
10 void loop()
11 {
12   valor = analogRead(A5);
13   Serial.println(valor);
14
15   digitalWrite(10, HIGH);
16   delayMicroseconds(valor);
17   digitalWrite(10, LOW);
18   delayMicroseconds(valor);
19 }
```

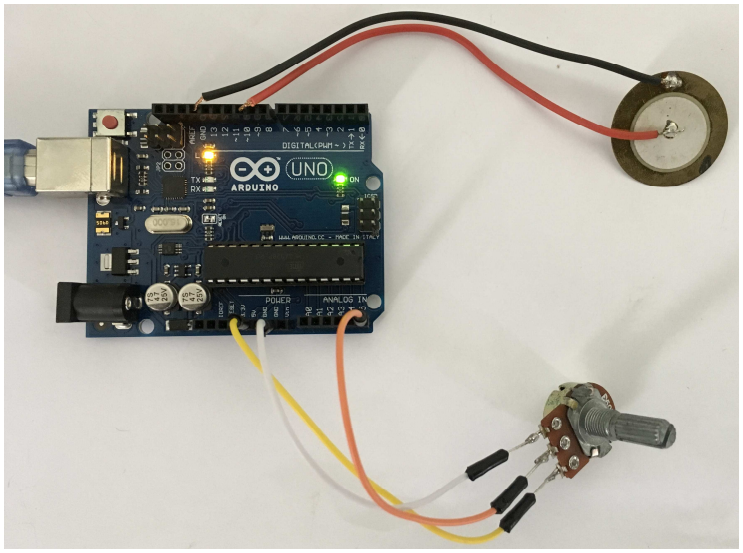


Figura 10.6: Arduino executando o código acima.

Exercício 10.2 Modifique o código e as ligações acima para que um LED varie sua luminosidade de acordo com a frequência da onda sonora emitida pelo PZT.

Exercício 10.3 Modifique o código e as ligações acima para o PZT gerar as sete notas musicas em sequência. Você pode considerar que as sete notas correspondem as seguintes frequências: 264 Hz (dó), 297 Hz (ré), 330 Hz (mi), 352 Hz (fá), 396 Hz (sol), 440 Hz (lá) e 495 Hz (si).

10.3 Movimento com o motor de passo

O motor de passo é um tipo de motor onde é possível controlar precisamente a rotação do balancete. Seu funcionamento parte do princípio onde uma rotação completa (360^o) é dividida em várias partes iguais, que seria o "passo" do motor. Assim, o passo corresponde ao ângulo mínimo que o balancete pode girar, que varia conforme as especificações do motor.

Nesse projeto usaremos o motor de passo 28BYJ-48 (em conjunto com o seu driver, ver figura 10.7), cuja volta completa é composta por 2048 passos, dando um ângulo de $\approx 0{,}18^o$ por passo.

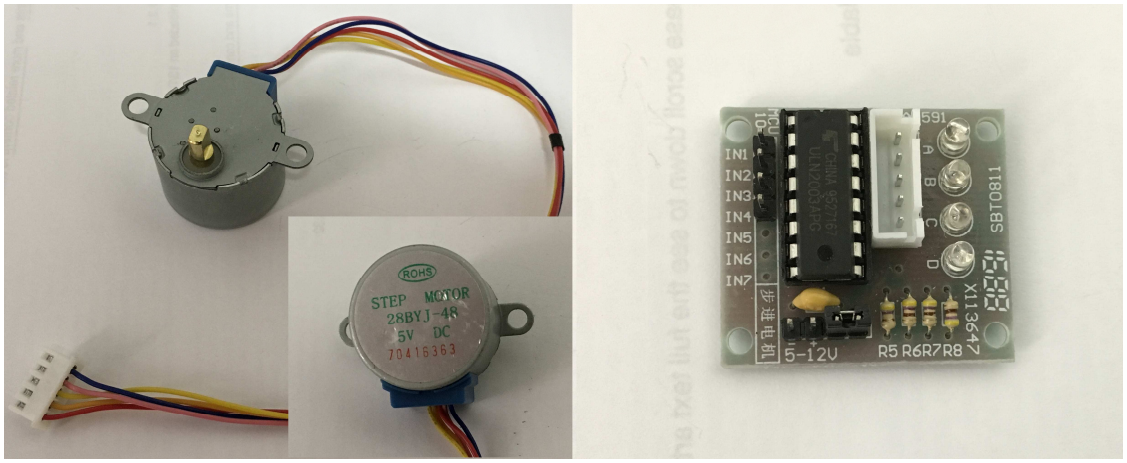


Figura 10.7: Motor de passo modelo 28BYJ-48 (à esquerda) e o driver controlador (à direita). O controle pelo Arduino é feito através dos pinos `IN1` até `IN4`, que é passado ao motor de passo através do soquete branco.

Para controlar o motor de passo, usamos o driver baseado no microcontrolador ULN2003. O uso desse driver é obrigatório principalmente porque o Arduino UNO não é capaz de inverter a tensão entre seus pinos, de forma que não seria possível fazer o motor girar em dois sentidos sem ter que mudar o circuito a todo instante. O esquema das ligações dos seus pinos com os do Arduino é mostrado na tabela e na figura abaixo, onde decidimos usar as portas `8` a `11`.

IN1, IN2, IN3 e IN4	Portas digitais do Arduino
$+/-$	Alimentação (5 V em $+$ e `GND` em $-$)

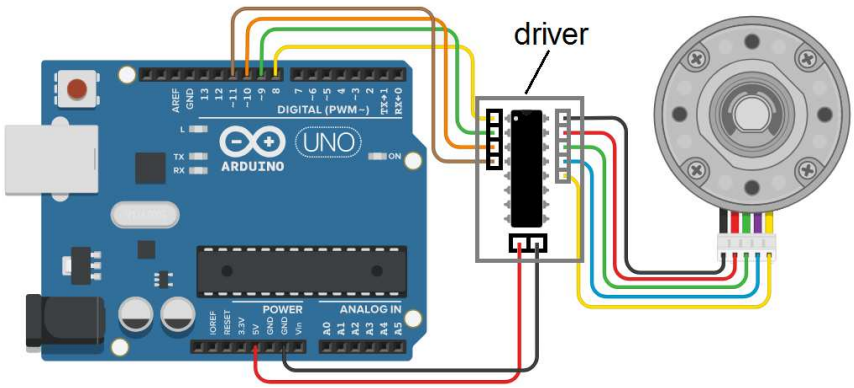


Figura 10.8: Conexões entre o motor de passo, o driver e as portas do Arduino. Desenhado no site `https://www.tinkercad.com`.

Agora vamos ao código necessário para controlar o motor de passo. Primeiramente precisamos declarar a biblioteca `Stepper.h`, que é padrão do Arduino é portanto já está na sua pasta `libraries`. Essa biblioteca trabalha em conjunto com a função **myStepper()**. É nessa função que são definidas as quatro portas digitais do Arduino que vão controlar o motor. A sua sintaxe é

```
1 Stepper myStepper(passos-por-revolução, IN1,IN3,IN2,IN4);
```

Assim, no código abaixo escolhemos 500 passos por revolução e as portas `8`, `9`, `10` e `11` para serem ligadas às portas `IN1`, `IN3`, `IN2` e `IN4`, respectivamente.

Com a função `setSpeed()` controlamos a velocidade do motor, e com a função `step()` controlamos a quantidade de passos que o motor deve dar. Dessa forma, o motor no código abaixo tem sua velocidade fixada em 60 e ele dá 1/4 de volta (giro de 90^o) a cada 2 segundos.

> A função `step()` também aceita números negativos em seu argumento. Nesse caso, o motor irá girar no sentido oposto.

```
1  #include<Stepper.h>
2
3  Stepper myStepper(500, 8,10,9,11);
4
5  void setup()
6  {
7    myStepper.setSpeed(60);
8  }
9
10 void loop()
11 {
12   myStepper.step(512);
13   delay(2000);
14 }
```

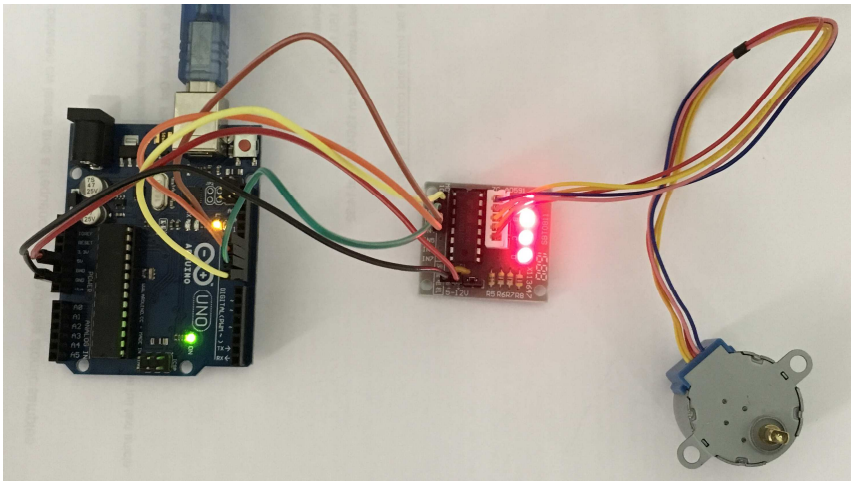


Figura 10.9: Arduino executando o código acima.

Exercício 10.4 Faça uma modificação no código acima para que o motor dê meia volta em um sentido, espere 1 segundo, e depois dê uma volta completa no outro sentido, esperando novamente 1 segundo. Deve fazer isso continuamente.

Exercício 10.5 Faça uma modificação no código acima para que o motor dê uma volta completa com velocidade crescendo linearmente desde 0 até a sua velocidade máxima.

Exercício 10.6 Faça uma modificação no código e no circuito acima para que o motor dê uma volta completa e espere 1 segundo, repetindo o processo. Enquanto o motor estiver girando, um LED deve ficar aceso.

11. Outras placas

11.1 Raspberry Pi

O Raspbetty Pi é na verdade um computador completo em uma placa com tamanho aproximado do Arduino Uno R3, fundado na Inglaterra em 2012 pela Raspberry Pi Foundation. Todos os componentes de um computador convencional estão soldados na placa: processador, memória RAM, entradas USB, HDMI, etc, e o sistema operacional roda em um cartão memória adquirido à parte. Há uma série de placas com poderes de processamento e preços distintos, sendo o Raspberry Pi 3 B+ (ver Fig. 11.1) o modelo mais recente e mais poderoso, lançado em 2018. Seu preço no site do revendedor oficial do Brasil é de R$ 279,00.

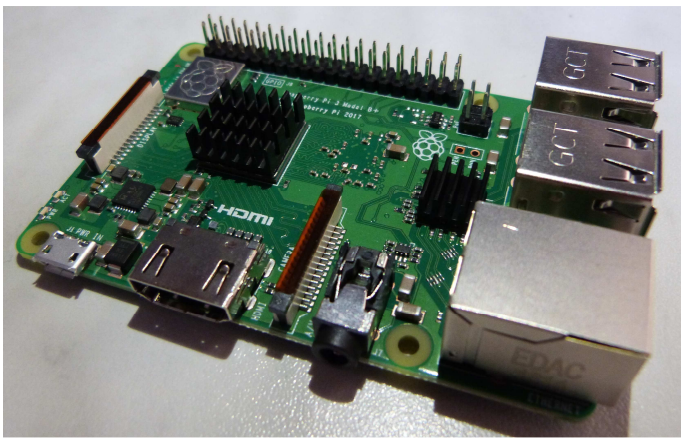


Figura 11.1: Raspberry Pi 3 B+.

11.1.1 O Hardware

Seu hardware é o seguinte:

- Processador Broadcom BCM2837B0, com 64 bits em 1.4 GHz

- Memória 1GB LPDDR2 SDRAM
- Rede sem fio 2,4 GHz e 5 GHz IEEE 802.11.b/g/n/ac
- Porta Ethernet (rede com fio) Gigabit USB 2.0 em 300 Mbps
- 40 pinos GPIO
- 1 porta HDMI
- 4 portas USB 2.0
- Entrada para câmera e para tela sensível ao toque
- Porta de saída de som
- Entrada para cartão micro USB
- Alimentação em 5 V/2,5 A

Os 40 pinos GPIO podem ser usados como entrada e saída de dados, assim como no Arduino. O mouse e o teclado podem ser conectados nas portas USB e um monitor pode ser conectado na saída HDMI.

Como pode ser visto na Fig. 11.1, os pinos são “para fora”, e não para dentro como no caso do Arduino Uno R3. Assim, é preciso ter um cuidado adicional com curtos que podem ser ocasionados ao conectar dois desses pinos com algum objeto metálico. Isso pode queima a placa.

11.1.2 O software

Como já mencionado, a placa Rasbperry roda um sistema operacional (SO). Os sistemas suportados são os seguintes:

- **NOOBS** – Como o nome já indica, é um sistema de fácil instalação e operação, indicado para novatos.
- **Raspbian** – Rapsberry + Debian. Sistema operacional oficial da placa.
- **Snappy Ubuntu Core** – Voltado para desenvolvedores.
- **Windows 10 IOT Core** – Windows 10 para IoT.
- **OSMC** – Open Source Media Center. Sistema operacional para mídia.
- **LibreELEC**
- **PiNet**
- **RIC OS** – Distribuição não Linux.
- **Weather Station**
- **IchogoJam RPi**

A instalação do NOOBS é bem simples. Bsta baixar o SO através do link `https://www.raspberrypi.org/downloads/noobs/`, que vem dentro de uma pasta zipada. Em seguida, basta extraí-la, copiar o conteúdo para o cartão de memória e ligar a placa, que deve estar conectada ao mouse, ao teclado e ao monitor.

O SO NOOBS (Fig. 11.2) já vem com diversos programas voltados à programação e ao ambiente acadêmico, além da suíte LibreOffice e de alguns jogos simples. Podemos destacar o famoso software de modelagem Mathematica (Fig. 11.3), o compilador gcc e diversas IDE's para programação, como Scratch, Python e Java. Há também o navegador Cromium e programas padrão para visualização de imagens, vídeos e arquivos pdf.

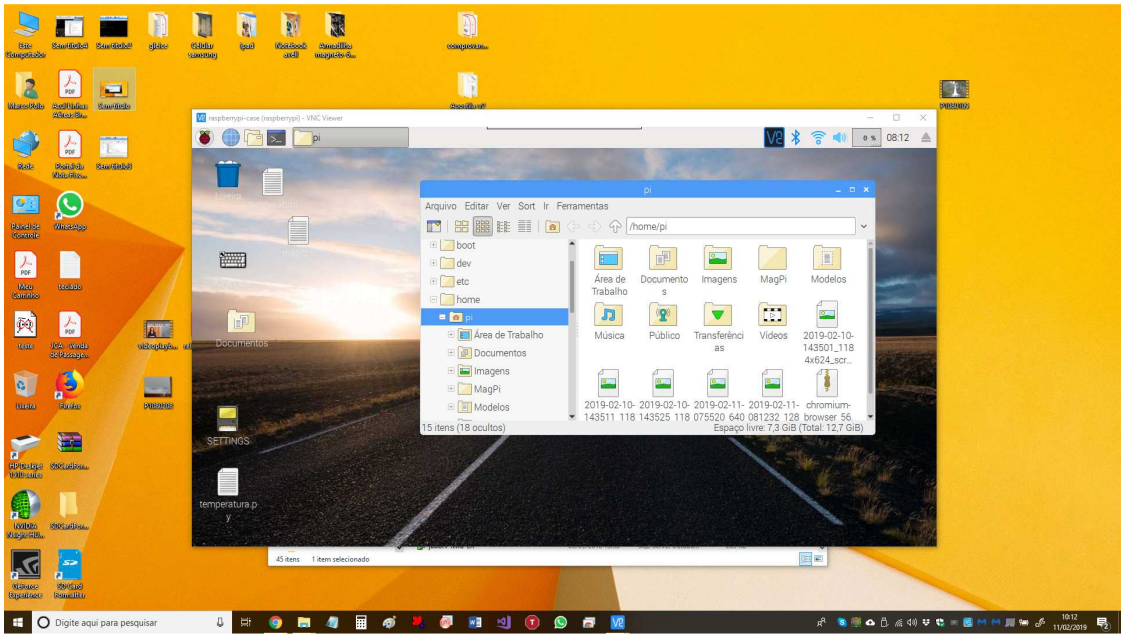


Figura 11.2: NOOBS rodando no Raspberry, mas sendo visualizado pelo Windows através de software de acesso remoto.

Não é necessária nenhuma preparação para a compilação de programas escritos em C, já que, como mencionado, o sistema vem com gcc. Basta execução dos comandos `gcc nome-do-arquivo.c -o nome-do-executável` e `./nome-do-executável` no terminal e pronto (Fig. 11.4).

O Raspberry pode ser acessado remotamente, como mostrado na Fig. 11.2. Isso elimina a necessidade do mouse, do teclado e de um monitor, tornando a configuração da placa mais prática. O acesso se dá por SSH ou pelos softwares como TeamViewer ou VNC Viewer, por exemplo, este último já incluso no SO. Ele pode ser acessado em `Menu` → `Internet` → `VNC Viewer`. Em seguida, basta criar uma conta, instalar o VNC também no computador ou no tablet e pronto, seu Raspberry pode ser acessado tanto na rede local como em qualquer parte do mundo.

11.1.3 Acendendo e apagando um LED

Nesse projeto simples, exemplificaremos o uso do Raspberry para executar um programa que acende e apaga um LED e intervalos de 1 segundo, como no primeiro Sketch apresentado no Cap. 3. O Raspberry Pi, por ser um computador, pode ser usado para programar uma placa Arduino; entretanto, usaremos o próprio processador da placa Raspberry para isso.

Conecte os terminais de um LED em dois fios jumper fêmea-fêmea, e estes aos terminais 6 e 12 da placa. Observe a numeração das portas na Fig. /reffig11-5. Veja que, assim como na placa do Arduino, as portas do Raspberry tem usos diferentes. Não entraremos em detalhes aqui, porque

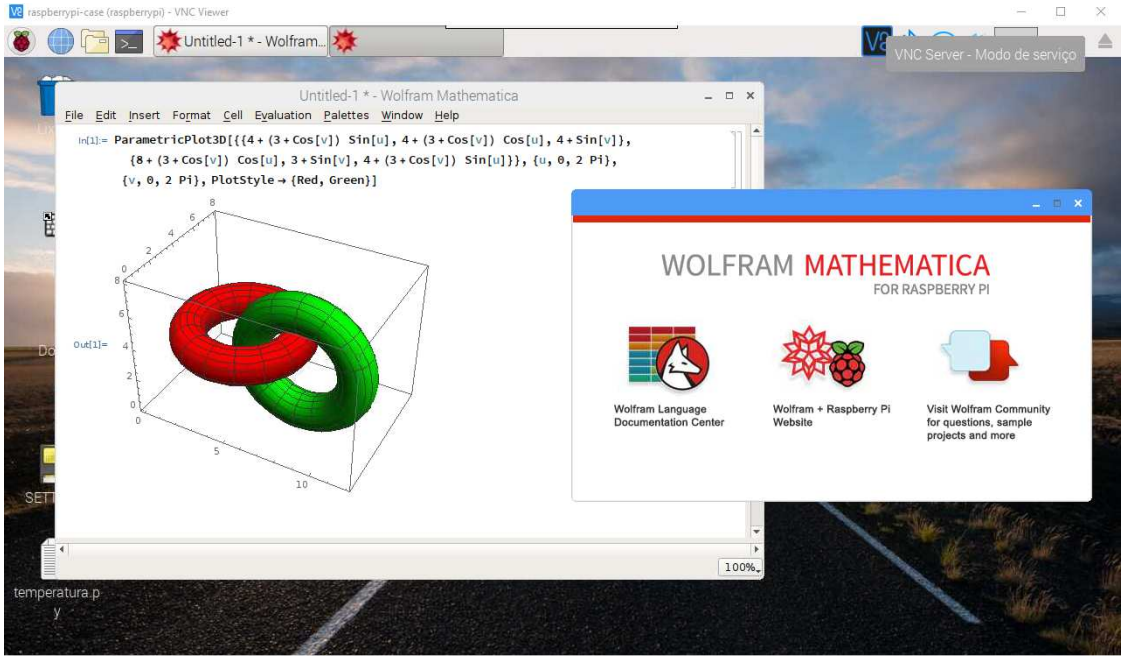


Figura 11.3: Superfícies paramétricas $x = 4+(3+\cos v)\sin u, y = 4+(3+\cos v)\cos u, z = 4+\sin v$ e $x = 8+(3+\cos v)\cos u, y = 3+\sin v, z = 4+(3+\cos v)\sin v$, com ambas as variáveis no intervalo $[0, 2\pi]$, criadas no Mathematica.

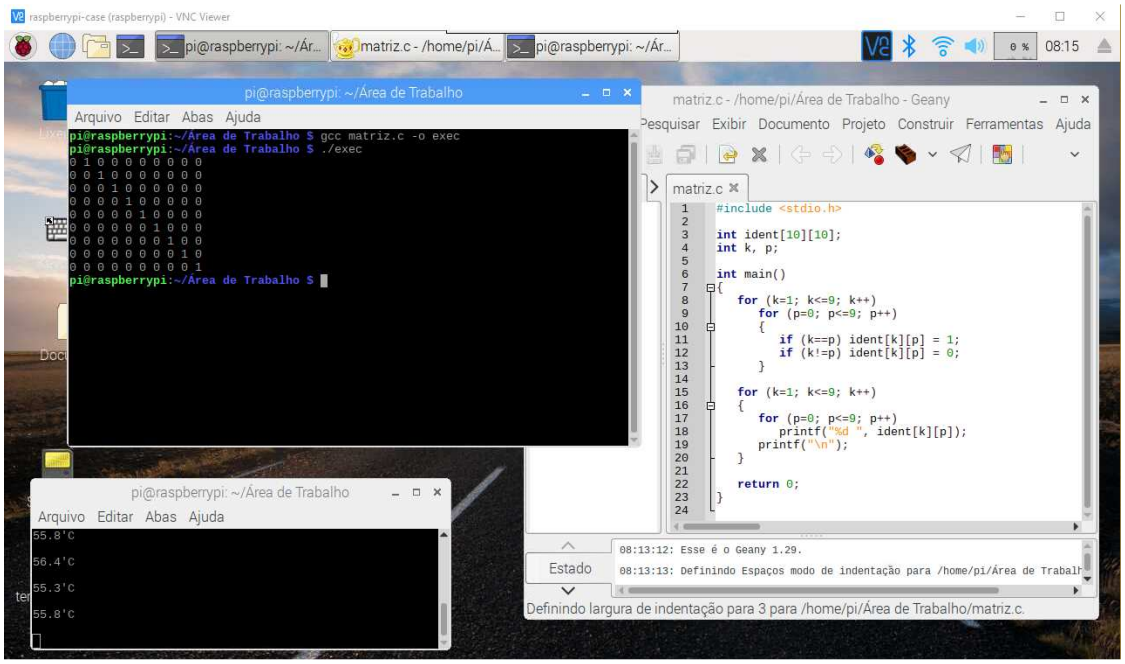


Figura 11.4: Código do Exemplo 5.19 sendo executado no Raspberry Pi, mas agora na linguagem C. Abaixo temos a monitoração da temperatura do processador.

todos os detalhes dessa placa daria assunto para outra apostila completa. Para esse projeto, mbasta saber que a porta 6 terra e a porta 12 é uma GPIO.

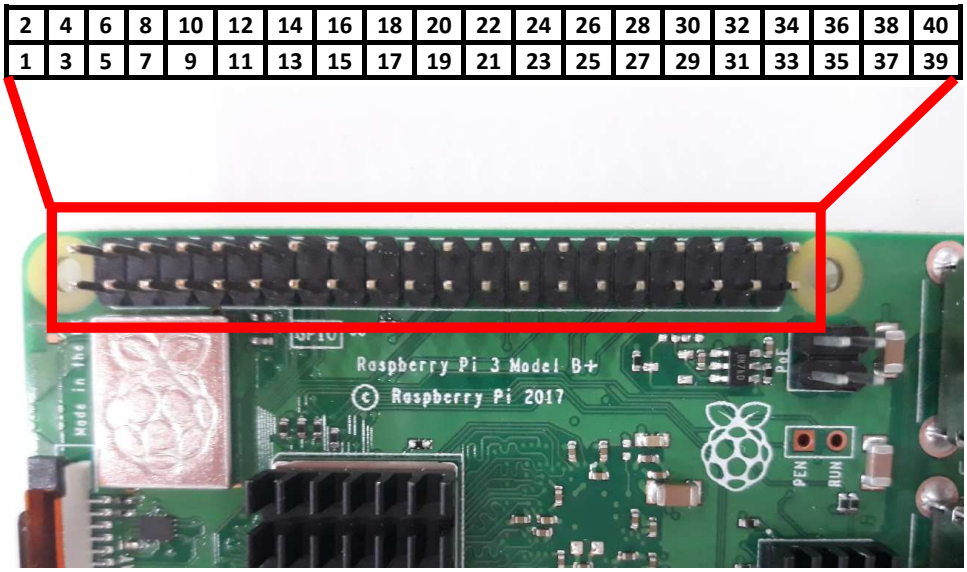


Figura 11.5: Ordem dos pinos GPIO da placa Raspberry Pi 3B+.

À respeito da programação, a melhor opção no Raspberry é usar o Python, uma linguagem simples porém poderosa, que é uma das mais usadas no mundo atualmente. Assim, abra o IDE do Python em `Menu` → `Desenvolvimento` → `Python 3 (IDLE)`, dentro do SO da placa. No menu `File`, clique em `New File` e lá, insira o código Python mostrado abaixo. Por fim, clique em `Run` → `Run Module` para executar o código. Outra opção é usar o terminal, inserindo o código em um editor de texto, salvando como `nome-do-arquivo.py` e digitando, no terminal, a instrução `python nome-do-arquivo.py`.

```
#Declaração da biblioteca da GPIO
import RPi.GPIO as GPIO

#Declaração da biblioteca time
import time
GPIO.setmode(GPIO.BOARD)

#Habilita o pino 12 para saida de dados
GPIO.setup(12, GPIO.OUT)

#Análogo da função while() ou mesmo loop() do Arduino
while(1):
  #Gera 3,3 V no pino 12 - análogo da função digitalWrite
  GPIO.output(12, 1)

  #Análogo da função delay()
  time.sleep(1)

  #Gera 0 V no pino 12
  GPIO.output(12, 0)
```

```
time.sleep(1)
```

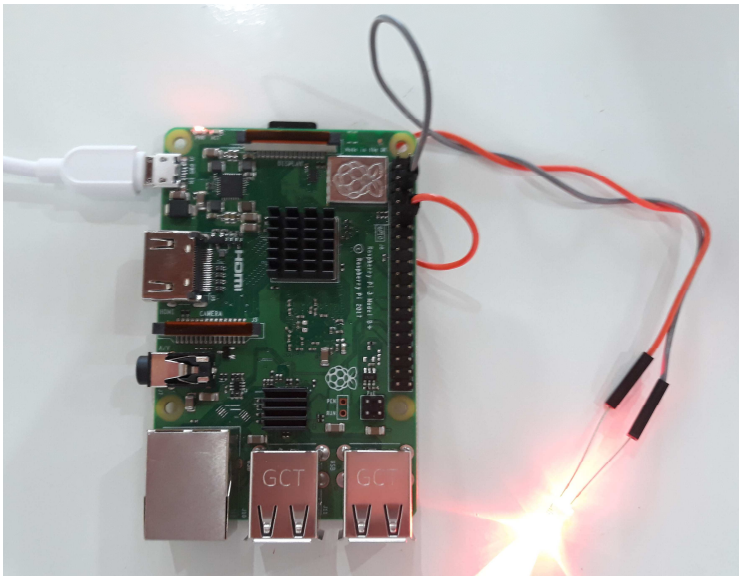


Figura 11.6: LED acendendo e apagando.

11.2 ESP 8266 e ESP32

As ESP8266 e ESP32 são pequenas placas microcontroladoras de baixo custo e baixo consumo de energia, criadas e desenvolvidas pela empresa chinesa Espressif Systems. A última é a sucessora da primeira, apresentando mais performance e possuindo bluetooth embutido, e ambas possuem wifi. Elas são voltadas tanto para dispositivos móveis, eletrônica e IoT, como também para o ambiente industrial, podendo trabalhar entre -40^o e 150^oC. Elas possuem aplicações em reconhecimento de voz, automação residencial e industrial, agricultura inteligente, medicina e mais.

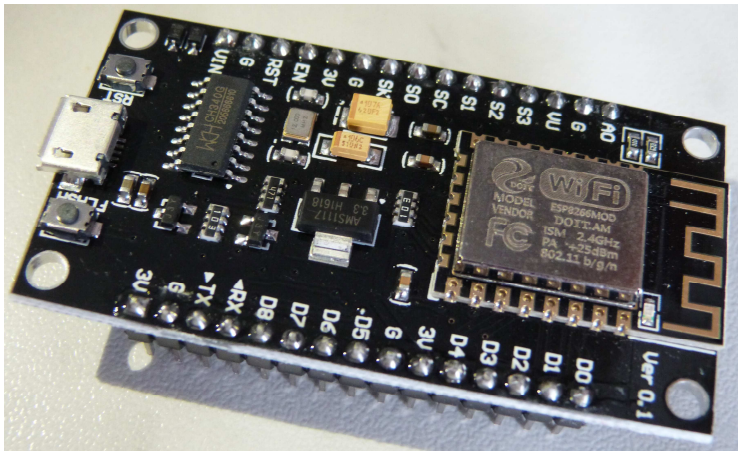


Figura 11.7: Placa ESP8266.

A partir deste ponto focaremos na placa ESP8266. Suas especificações técnicas são as seguintes:

- Microprocessador L106 32de 32 bits RISC Xtensa Diamond Standard 106Micro com clock em 80 MHz

- Memória RAM (32 kB instruções, 32 kB cache de instruções, 80 kB de dados e 16 kB do sistema)
- Wifi 802.11 b/g/n (2,4 GHz)
- 16 pinos GPIO

11.2.1 Instalação - ESP8266

A ESP8266 pode ser programa usando o IDE do Arduino após uma configuração simples. Primeiramente, com o IDE do Arduino aberto, clique em `Arquivo` → `Preferências`. Na janela que abrir, procure o campo `URLs Adicionais para Gerenciadores de Placas` e cole o endereço abaixo:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Clique no botão ao lado do campo e dê enter. Em seguida, no menu, acesse `Ferramentas` → `Placa: ''Arduino/Genuino UNO''` → `Gerenciador de Placas...` No campo de busca, digite `esp8266` e clique em instalar. Será baixado um arquivo com tamanho aproximado de 44 MB e a instalação estará finalizada. Em seguida, para trabalhar com a ESP8266 no IDE do Arduino, basta ir em `Ferramentas` → `Placa:` e selecionar a opção `NodeMCU 1.0 (ESP-12E Module)`.

11.2.2 Fazendo um LED piscar

Assim como no Raspberry Pi, faremos aqui o projeto mais simples, que consiste em deixar um LED piscando em intervalos de 1 segundo. O código é o mesmo usado no Cap. 3, que mostramos abaixo novamente:

```
1  void setup()
2  {
3    pinMode(13, OUTPUT);
4  }
5
6  void loop()
7  {
8    digitalWrite(13, HIGH);
9    delay(1000);
10   digitalWrite(13, LOW);
11   delay(1000);
12 }
```

A única mudança diz respeito às portas dessa placa, que são diferentes. Não entraremos em detalhes aqui; basta saber que a porta `13` (usada no código acima) corresponde à porta `D7` do ESP8266, assim como a porta `GND` está identificada apenas pela letra `G`. Basta conectar as pernas do LED e pronto.

O carregamento do código na placa (ver Fig. 11.8) é bem lento na ESP8266 quando comparado com o Arduino Uno R3. Isso está ligado ao fato de que a ESP8266 não foi otimizada para a linguagem do Arduino.

Figura 11.8: ESP8266 carregando o código que faz um LED piscar.

Bibliografia

[1] Site Oficial do Arduino. https://www.arduino.cc/ (Acessado em fevereiro de 2019).

[2] Arduino Wiki - Github. https://github.com/arduino/Arduino/wiki/Build-Process (Acessado em setembro de 2017).