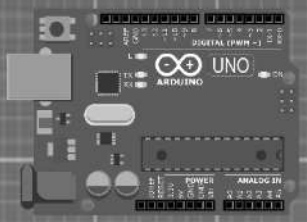
MARCELO EDUARDO DE OLIVEIRA
MATHEUS FERNANDO LIMA ZUCCHERELLI
GIOVANNI POLETTE DALLA LIBERA
RENATA LIMA ZUCCHERELLI DE OLIVEIRA
ADRIANO ROGÉRIO BRUNO TECH



INTRODUÇÃO À ROBÓTICA EDUCACIONAL COM ARDUINO

HANDS ON!

INICIANTE

PIRASSUNUNGA - SP
FACULDADE DE ZOOTECNIA E ENGENHARIA DE ALIMENTOS
2020

Introdução à Robótica Educacional com Arduino - HANDS ON!

INICIANTE

Autores

Marcelo Eduardo de Oliveira
Matheus Fernando Lima Zuccherelli
Giovanni Polette Dalla Libera
Renata Lima Zuccherelli de Oliveira
Adriano Rogério Bruno Tech

DOI: 10.11606/9786587023052

Pirassununga
Faculdade de Zootecnia e Engenharia de Alimentos
2020

UNIVERSIDADE DE SÃO PAULO

Reitor: Prof. Dr. Vahan Agopyan

Vice-Reitor: Prof. Dr. Antonio Carlos Hernandes

FACULDADE DE ZOOTECNIA E ENGENHARIA DE ALIMENTOS

Avenida Duque de Caxias Norte, 225

Pirassununga, SP

CEP 13635-900

http://www.fzea.usp.br

Diretora da FZEA: Profa. Dra. Elisabete Maria Macedo Viegas

Vice-Diretor da FZEA: Prof. Dr. Carlos Eduardo Ambrósio

Dados Internacionais de Catalogação na Publicação

Serviço de Biblioteca e Informação da Faculdade de Zootecnia e Engenharia de Alimentos da Universidade de São Paulo

O48i

Oliveira, Marcelo Eduardo de
Introdução à robótica educacional com Arduíno – hands on! : iniciante / Marcelo Eduardo de Oliveira, Matheus Fernando Lima Zuccherelli, Giovana Polette Dalla Libera, Renata Lima Zuccherelli de Oliveira, Adriano Rogério Bruno Tech. -- Pirassununga: Faculdade de Zootecnia e Engenharia de Alimentos, 2020.
51 p.

ISBN 978-65-87023-05-2 (e-book)
DOI: 10.11606/9786587023052

1. Eletrônica e Programação Básica. 2. Robótica. 3. Instrumentação Eletrônica. I. Zuccherelli, Matheus Fernando Lima. II. Libera, Giovana Polette Dalla. III. Oliveira, Renata Lima Zuccherelli de. IV. Tech, Adriano Rogério Bruno. V. Título.

Ficha catalográfica elaborada por Girlei Aparecido de Lima, CRB-8/7113

Introdução à Robótica Educacional com Arduino - HANDS ON!

INICIANTE

Apoio

Realização

Patrocínio

Santander

Prefácio

Este trabalho tem como objetivo apoiar alunos e iniciantes em programação e eletrônica básica no aprendizado simples e dirigido. Aqui, o iniciante terá disponível 15 atividades básicas utilizando o Arduino, com uma programação fácil e eficiente, além da montagem de todo o circuito em desenvolvimento de maneira simples e descomplicada.

Parafraseando uma Grande Mente que dizia, quando queremos ensinar devemos nos colocar no lugar daquele que irá receber o ensinamento, ou seja, devemos ser diretos e objetivos como a natureza, ela é transparente e absoluta e encerra em si, o TODO.

Os Autores.

Lista de Símbolos

Ω	ohm
KΩ	kiloohm

Lista de Figuras

Lista de Tabelas

Sumário

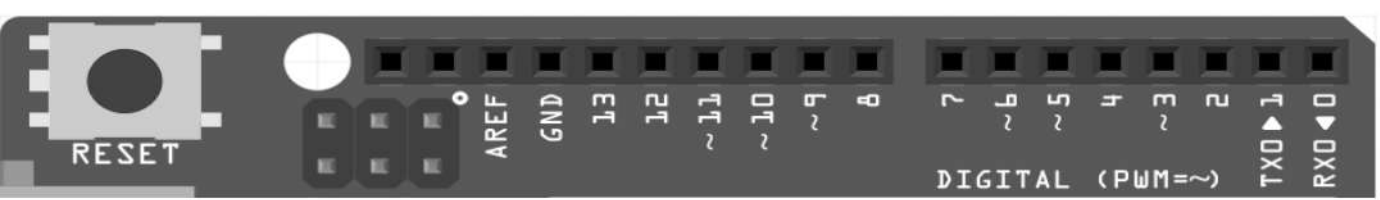
1. Introdução

1.1 Arduino

O Arduino é uma plataforma eletrônica *open source*, que tem como objetivo integrar hardware e software de maneira fácil, permitindo que pessoas com pouco conhecimento na área possam desenvolver as suas habilidades e aprendizado de maneira mais simples, aprendendo a eletrônica básica e programação. As placas do Arduino foram idealizadas para serem capazes de ler variáveis físicas, como entradas de luz em um sensor, o toque de um dedo em um botão ou até mesmo ler uma mensagem disponível em um meio virtual e transformá-la em uma saída, seja pelo acionamento de um motor, o ligar de uma luz ou de um LED. A sua programação é amigável e permite o envio de linhas de comando para o microcontrolador existente na placa, dizendo como atuar, o que fazer e como fazer.

E, para que tudo isso seja operacionalizado há a necessidade de interação entre o hardware e o usuário, que é feita por meio de uma linguagem de programação (Arduino), que é simples e objetiva (baseada na linguagem *Wiring*) e o Software Arduino (*Integrated Development Environment* – IDE, ou, em português, ambiente de desenvolvimento integrado), com base no ambiente de programação *Processing* (Arduino.cc, 2020). Desde sua criação, o Arduino tem sido usado em milhares de implementações, variando desde projetos simples até os mais complexos. Hoje o Arduino conta com uma grande comunidade mundial, envolvendo os próprios criadores, estudantes, programadores e profissionais, que reuniram suas contribuições a esta plataforma de código aberto e disponibilizaram uma grande quantidade de conhecimento e projetos acessíveis a todos, sejam iniciantes ou profissionais já em atuação.

O Arduino nasceu no *Ivrea Interaction Design Institute* como uma ferramenta simples e de fácil operacionalização para integração de hardware e software aplicados a prototipagem rápida e destinada a estudantes sem formação em eletrônica e programação. Ao alcançar uma comunidade maior, o Arduino começou a sofrer alterações para se adaptar aos novos desafios e necessidades, diferenciando sua



oferta de placas simples a produtos com aplicações voltadas a Internet das coisas (IoT – *Internet of Things*), com várias aplicações integrando impressão 3D e ambientes automatizados. As placas do Arduino, bem como o seu software de desenvolvimento são de código aberto, permitindo assim, que usuários possam implementar sistemas e controles de maneira independente e, se necessário adaptar o sistema para controles mais elaboradores, como controle e monitoramento de ambientes produtivos (Arduino.cc, 2019).

1.2 Por que usar Arduino

O Arduino, por possuir uma interface simples e acessível aos usuários, tem sido usado nas mais diferentes aplicações, variando de simples projetos aos mais elaborados projetos e aplicativos. O software Arduino é de fácil entendimento e de implementação para usuários iniciantes, tornando-se uma linguagem bem estrutura nas mãos de usuários experientes, além de sua interface rodar em ambientes variados, como o Mac, Windows e Linux.

O Arduino se tornou uma ferramenta simples e completa, permitindo que professores e alunos o usem para implementar atividades didáticas de baixo custo e, que permitam um aprendizado voltado a interação teoria - prática, onde o aluno pode na prática comprovar a teoria baseado nos experimentos de classe voltados para a química, física, programação e eletrônica, dentre outras mais, além de iniciá-los nos fundamentos da robótica.

O Arduino é uma ferramenta de apoio para que os professores possam inovar no ensino aplicando a tecnologia em sala de aula. Hoje, qualquer pessoa, com o auxílio de um professor ou de aulas disponíveis online, podem começar a aprender os conceitos básicos de programação e eletrônica, bastando para isso um tempinho para se dedicar, vendo vídeos educativos ou participando de atividades didáticas, ou mesmo, compartilhando ideias online com outros membros da comunidade Arduino. Atualmente, estão disponíveis no mercado outros microcontroladores e plataformas de microcontroladores para o aprendizado e para a construção de projetos práticos e didáticos, como o *Parallax Basic Stamp, o BX-24 da Netmedia, o Phidgets, o*

Handyboard do MIT - Massachusetts Institute of Technology, e muitos outros que oferecem funcionalidades bem parecidas. O Arduino, além de permitir um aprendizado simples e prático, também facilita a utilização dos microcontroladores voltados a projetos simples ou complexos, oferecendo vantagens para docentes, discentes e para os entusiastas interessados em aprender, aplicar e inovar através de uma ferramenta simples e objetiva (Arduino.cc, 2019). Basicamente, o Arduino possui:

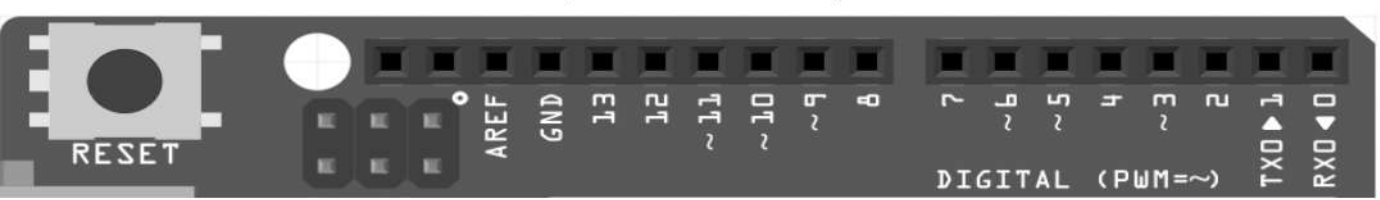
• Baixo custo: suas placas são relativamente baratas, quando comparadas com outras plataformas de microcontroladores. A versão mais barata do módulo Arduino pode ser montada manualmente e até os módulos pré-montados custam menos de R$ 50,00.

• Multiplataforma: O IDE do Arduino é multiplataforma, ou seja, pode rodas em sistemas operacionais, como: Windows, Macintosh OSX e Linux.

• Ambiente de programação simples e objetivo: A IDE do Arduino é de fácil uso e amigável, principalmente para os iniciantes, além de permitir que usuários avançados aproveitem toda sua funcionalidade. Para os professores, a sua programação é baseada no ambiente de programação *Processing*, o que torna o aprendizado da linguagem mais simples e clara para o aluno.

• Código aberto e extensível: O software Arduino, como já dito anteriormente, é *open source*, ou seja, de código aberto e disponível para aplicações mais complexas envolvendo programadores mais experientes. A linguagem pode ser expandida por meio das bibliotecas C/C++, e os interessados em implementar aplicações mais complexas podem migrar para linguagens mais avançadas, no caso AVR-C na qual a programação se baseia. Da mesma forma, é possível adicionar o código AVR-C diretamente aos seus programas em Arduino.

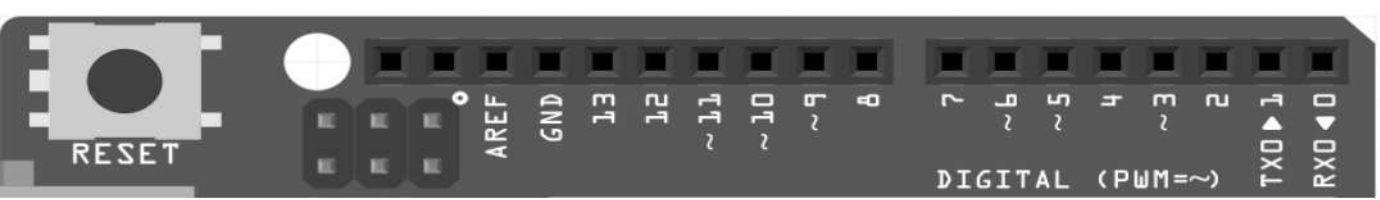
• Código aberto e hardware extensível: As placas do Arduino são publicadas sob uma licença *Creative Commons*, para que projetistas de circuitos mais experientes possam implementar suas próprias versões do módulo, estendendo-o e aprimorando-o.



1.3 Componentes básicos

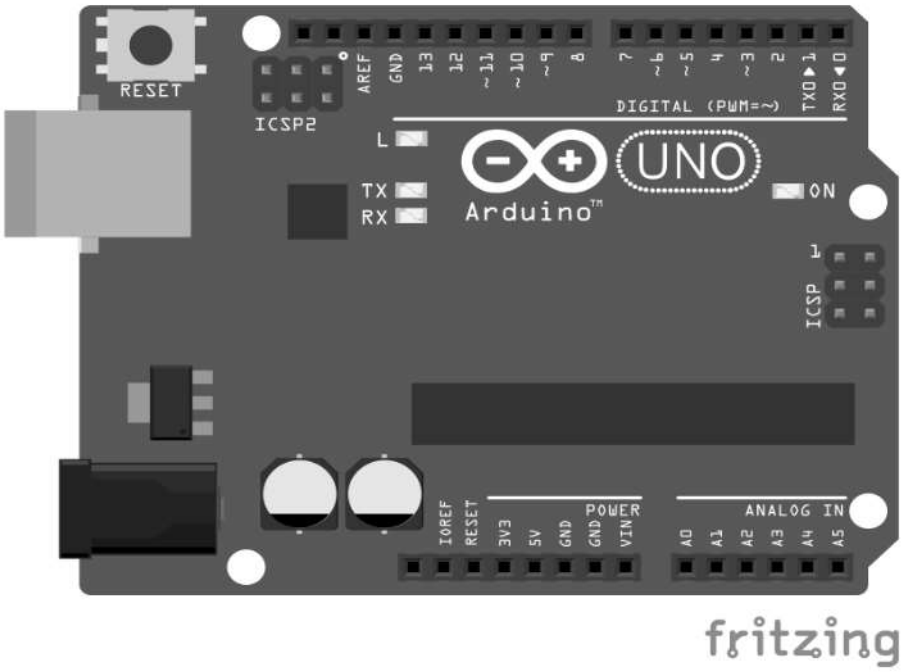
Os componentes básicos do *Arduino* são: **Microcontrolador:** é responsável por processar o *software* e manipular as portas de entrada e saída. É o cérebro do sistema e funciona como um computador dentro de um pequeno *chip*; **Conector USB:** é responsável pela conexão da placa ao computador, é por onde ocorre a transferência do programa implementado para dentro da placa (do circuito); **Pino de Entrada e Saída de dados:** é responsável por fazer com que a placa possa se comunicar com o ambiente externo. O *Arduino* possui 14 portas digitais, 6 pinos de entrada analógica e 6 pinos de saída analógica (PWM - Pulse Width Modulation); **Pinos de alimentação:** é responsável por fornecer diversos valores de tensão e são responsáveis por energizarem os componentes do projeto; **Botão *Reset*:** é o responsável por reiniciar ou *resetar* a placa; **Conversor *Serial*-USB e LED TX/RX:** é responsável pela comunicação entre o computador e o microcontrolador, sendo necessário um *chip* que faça a tradução de comunicação de um para o outro. O LED Tx/Rx acende quando alguma transmissão ou recepção está sendo realizada pela porta serial; **Conector de Alimentação:** é responsável por receber a fonte de alimentação externa e que pode ter uma tensão que varia de 7 a 20 *volts* e uma corrente mínima de 300 mA. Recomenda-se uma voltagem de 9 V; **LED de Alimentação:** indica se a placa está energizada e **LED Interno:** LED conectado ao pino digital (Vidadesilicio, 2020).

De uma maneira bem simples pode-se entender *Arduino* como sendo uma plataforma *open-source* de prototipagem eletrônica integrando hardware e *software* de maneira simples, flexíveis e de fácil uso, destinado as mais diversas áreas do conhecimento, principalmente a computação, eletrônica, a mecatrônica e robótica, mas podendo ser usada por qualquer pessoa interessada em criar objetos ou ambientes interativos, de maneira simples e rápida, ou seja, é uma plataforma composta basicamente por dois componentes: a Placa e a IDE. A placa pode ser entendida como o *hardware* do sistema que será estruturada para receber os projetos ou sistemas implementados e o IDE, que nada mais é que o *software* que irá operacionalizar o *hardware*, como um computador.

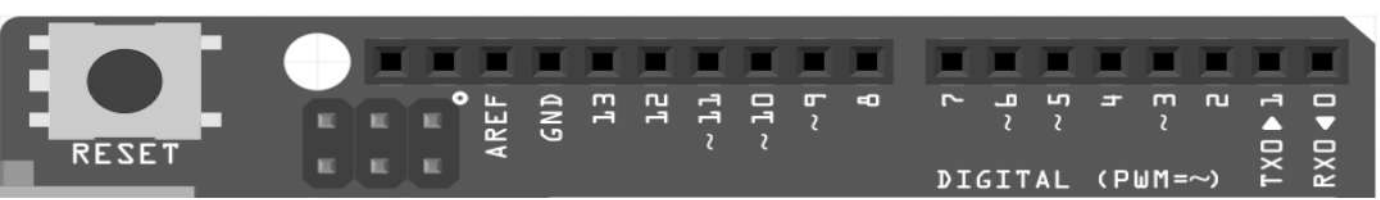


Atualmente, existem diversas placas de *Arduino*, mas para nossas implementações será usado o *Arduino* Uno. A Figura 1 mostra uma placa *Arduino* Uno.

Figura 1 - Arduino Uno



Fonte: Fritzing, 2019



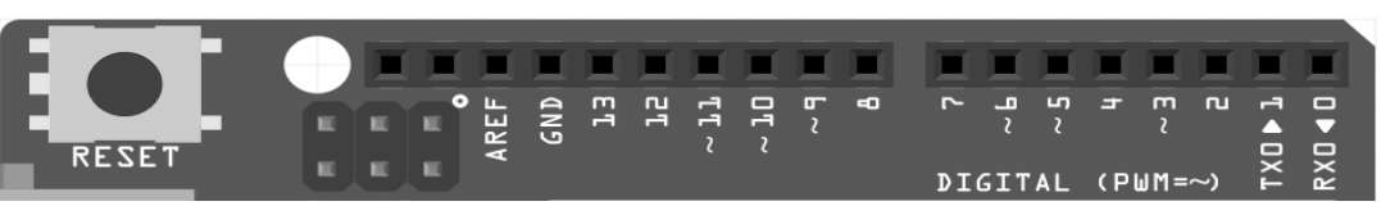
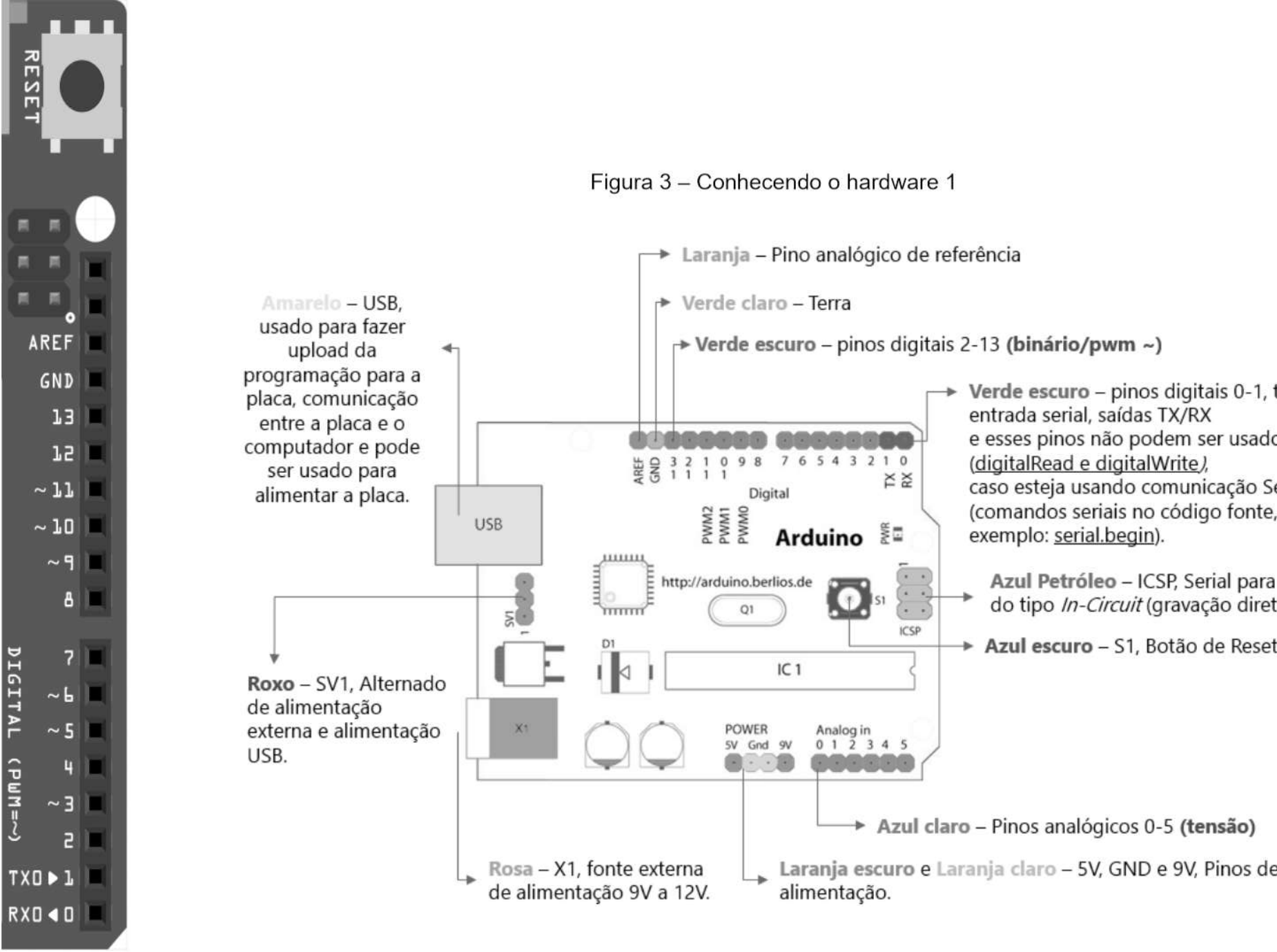


Figura 2 – Arduino Pinout Diagram

Fonte: https://www.circuito.io/blog/arduino-uno-pinout/

Figura 3 – Conhecendo o hardware 1



Fonte: Adaptado de Pictronics.com.br

Figura 4 – Conhecendo o hardware 2

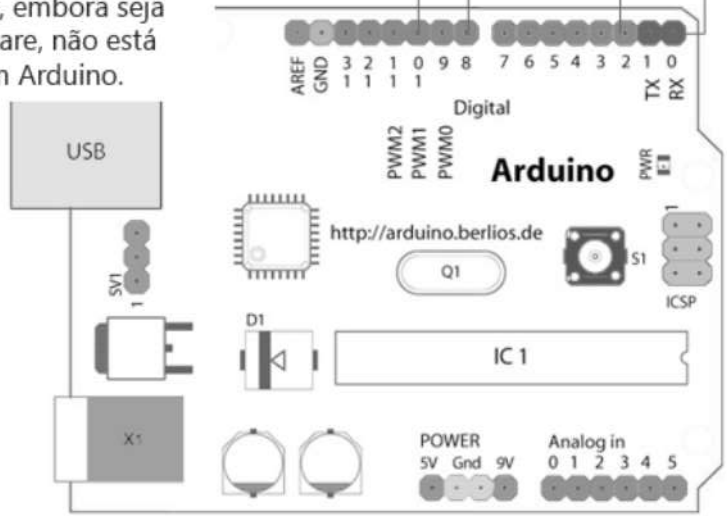
PWM nos 5, 6, 9, 10 e 11:
Forne na saída PWM de 8 Bits com a funçã alogWrite (). Em placas com um ATmega8, a saída PWM está disponível somente nos pinos 9, 10 e 11

analogWrite () == Imprime um valor de tensão
Ex: Controlar bilho do LED

SPI – Pino 10 (SS), Pino 11 (MOSI), Pino 12 (MISO) e Pino 13 (SCK):
Estes pinos são de apoio à comunicação SPI que, embora seja fornecido pelo hardware, não está incluído na linguagem Arduino.

Interrupções Externas – Pinos 2 e 3: attachInterrupt ()
Esses pinos podem ser configurados para disparar uma interrupção através de um valor de nível baixo, numa borda de subida ou descida, ou uma mudança de valor.
Pinos de chaveamento

Serial – Pino 0 (RX) e Pino 1 (TX):
Usados para receber (RX) e transmitir (TX) serial TTL.



PWM(do inglês **Pulse width modulation**) consiste em emular um sinal analógico através de pulsos digitais

Fonte: Adaptado de Pictronics.com.br

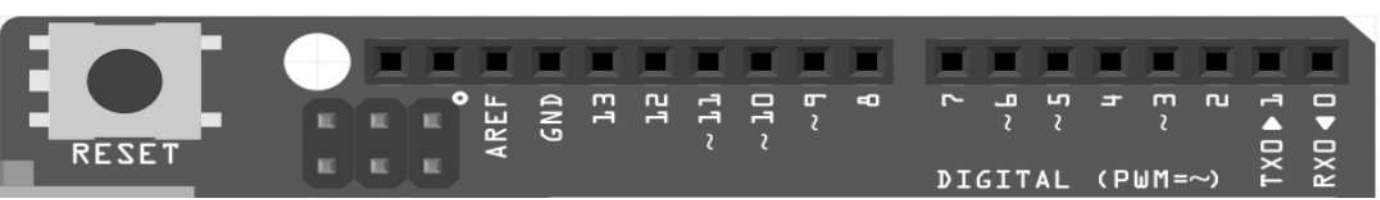
1.4 IDE Arduino

O IDE do *Arduino é* responsável pela implementação dos códigos e transferência desse código para o microcontrolador, que é responsável pelo processamento dos dados no Arduino, controlando o envio e recebimento de dados das portas digitais e analógicas de programação do micro controlador, ou seja, é o ambiente de desenvolvimento dos projetos confeccionados e que são gerenciados pelo *Arduino*. Uma das grandes facilidades da IDE é seu ambiente de programação que é baseada na linguagem C/C++.

O *software, que* é a IDE propriamente dito, ou seja, o ambiente de desenvolvimento do programa que será responsável por gerenciar e dar funcionalidade ao circuito, é implementada e executada em um computador (programação), conhecida como ***sketch***, que após concluída será transferida via *upload* para a placa de prototipagem *Arduino*, através de uma comunicação serial. O *sketch* implementado pelo analista dirá ao sistema ou à placa o que deve ser executado durante o seu funcionamento.

A IDE possui uma linguagem própria baseada na linguagem C e C++, como já descrito. O Ciclo de programação pode ser compreendida da seguinte maneira: Conexão da placa a uma porta USB do computador; Desenvolvimento de um *sketch* com comandos para a placa; *Upload* do *sketch* para a placa, utilizando a comunicação USB; Aguardar a reinicialização, após ocorrerá a execução do *sketch* criado. A partir do momento que foi feito o upload, o *Arduino* não precisa mais do computador: o *Arduino* executará o sketch criado, desde que seja ligado a uma fonte de energia (entre 9 e 12 *volts*). Aguardar a reinicialização, após ocorrerá a execução do sketch criado.

A partir do momento que foi feito o upload, o Arduino não precisa mais do computador, o Arduino executará o sketch criado, desde que seja ligado a uma fonte de energia (entre 9 e 12 volts).



2. Programando em C/C++

A linguagem de programação tem como objetivo transformar um algoritmo em um programa que roda em um computador. O Algoritmo nada mais é do que uma sequência de comandos estruturados com o objetivo de realizar uma tarefa ou um conjunto de tarefas bem definidas.

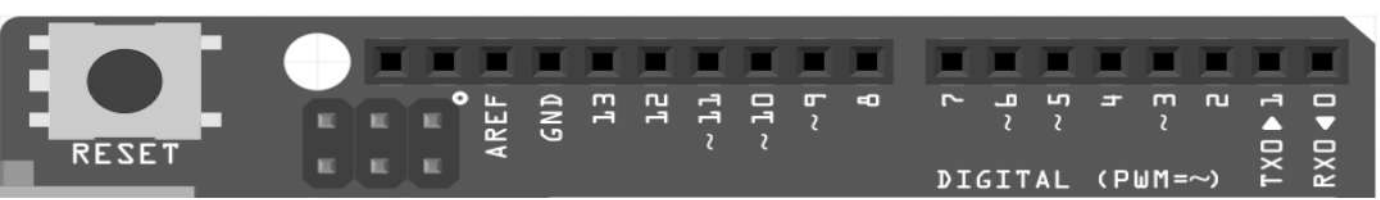
Hoje existem várias linguagens de programação e, dentre elas, podem ser citado o pascal, basic, Fortran, pyhton, C, C++, visual basic (basic), delphi (pascal), todas essas linguagens são ditas linguagens de alto nível e permitem aos analistas e programadores implementarem os seus códigos. Na programação no Arduino, isso não é diferente, tem-se a interação de um ambiente de programação (IDE), com o hardware, no qual uma sequência de comandos será executada com objetivos bem definidos e claros, como exemplo, controlar o acendimento de um led, a coleta de temperatura e umidade, por meio dos sensores.

Quando implementa-se um código para que ele possa executar uma tarefa, deve-se ter em mente um conceito muito simples, idealizado pelo biólogo austríaco Ludwig von Bertalanffy (1950), que definiu Sistema como sendo "um conjunto de partes interagentes e interdependentes que, conjuntamente, formam um todo unitário com determinado objetivo e efetuam determinada função" (REZENDE; ABREU, 2013) (FIGURA 2).

Figura 5 - Conceito de Sistemas

Fonte: Adaptado de Rezende e Abreu (2013).

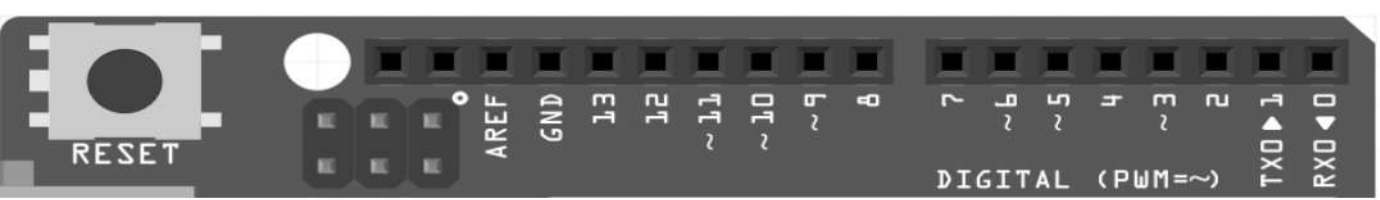
Esse conceito em computação auxilia na implementação de muitos programas, uma vez que fica claro como ocorre todo o fluxo de processamento dos dados, independentemente da linha de análise utilizado pelo analista ou programador.



Em síntese, todo programa inicia com os dados de entrada, ou seja, as variáveis que irão alimentar o sistema, o processamento nada mais é, do que um modelo matemático, ou um conjunto de **if** e **for** estruturados, simulando uma equação matemática. E, como consequência de um processamento têm-se a saídas ou os resultados da equação.

Com essa definição compreendida a construção de um programa fica mais fácil, dependendo do tipo de programa que se queira construir. Como ressalva, para um analista se formar, são necessários cinco anos de estudos nessa área.

Serão apresentados os principais conceitos de programação, de maneira simples e descomplicada, e, aqui, não há a presunção de formar nenhum programador profissional, mas sim, de introduzir os conceitos e comandos básicos que permitirão a todos compreender os comandos que serão utilizados durante as atividades que serão implementadas com o Arduino neste livro de introdução.



3. VARIÁVEIS

As variáveis são estruturas importantes em qualquer programação, independentemente da linguagem que se utilize para implementar os códigos ou programas. Elas podem ser entendidas como uma caixinha, onde os dados são armazenados temporariamente ou em definitivo e que são manipuladas durante a execução do programa. As variáveis devem ser inicializadas antes de usadas, mas, cabe aqui uma ressalva, dependendo da linguagem de programação isso não é necessário, o compilador da linguagem pode fazer isso automaticamente. Neste caso, onde serão implementados os códigos em linguagem C/C++ isso não ocorre, ou seja, deve-se iniciar as variáveis sempre que utilizadas pela primeira vez.

As variáveis são inicializadas quanto ao tipo e quanto ao valor, isso significa que terá de ser definida qual o uso que iremos atribuir a ela. As variáveis podem ser do tipo inteira, decimal, caractere, string, data e somente decimal positiva, assim, dependendo do tipo de dados que serão armazenados, tem-se um tipo definido para ela. Assim, as variáveis podem ser (TABELA 1).

Tabela 1 - Tipo de variáveis

Variável	Tipo
int	Inteiras
float	Decimal precisão simples
double	Decimal precisão dupla
unsigned	Decimal positiva
string	Cadeia de caracteres
char	Caracter
boolean	Lógica

Fonte: Autoria própria

Isso define que tipos de dados a variável poderá armazenar. Agora, as variáveis também podem ser inicializadas quanto aos valores. Exemplos:

Figura 6 - Tipo de variáveis

```
1 int A = 0;
2 float B = 12.00;
3 String C = "Hello Word";
```

Fonte: Autoria própria

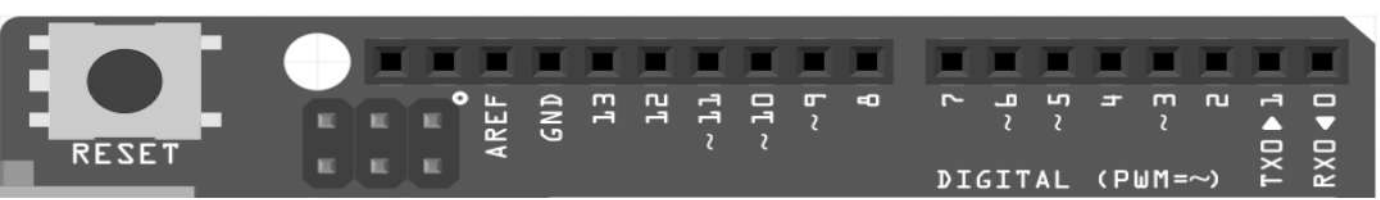
Uma outra forma de definir uma variável é com o comando #define.

Figura 7 – Comando Define

```
1 #define led 7
```

Fonte: Autoria própria

Desta forma, fica compreendido que sempre que se usar uma variável devemos saber com clareza o que vamos armazenar em seu interior.



3.1 Comandos e estruturas básicas usadas nas atividades a serem implementadas neste livro.

A seguir, serão apresentados os comandos básicos e a estrutura de programação do Arduino e, que serão usados nas atividades de 1 a 15, implementadas para que os alunos possam aprender e exercitar a programação e a eletrônica básica.

Basicamente, o Arduino possui a seguinte estrutura de blocos:

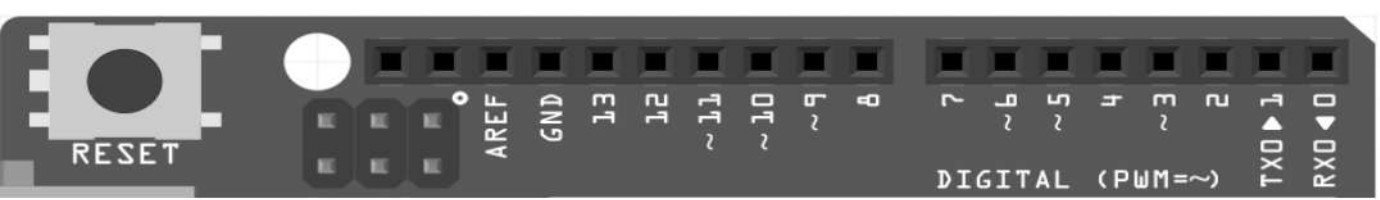
Figura 8 – Código Fonte

```
1 // Projeto LED -  nome do programa e o que ele faz
2 int led = 13;
3 void setup()
4 {
5   pinMode(led, OUTPUT);
6 }
7 void loop()
8 {
9   digitalWrite(led, HIGH);
10  delay(3000);
11  digitalWrite(led, LOW);
12  delay(3000);
13 }
14
```

Fonte: Autoria Própria

Esta é uma estrutura básica de implementação com a IDE (Arduino). Ela possui dois blocos padrões, uma é a função void setup(), que permite a configuração do ambiente relacionado às entradas e saídas de dados, em relação as portas analógicas e digitais. A outra, é a função **void loop()**, que executa em loop eterno os comandos e estruturas que estiverem em seu bloco de comando **{** comando **}** e, representa início e fim de bloco.

A primeira linha do código é reservada para a identificação do nome do programa e o que ele faz, as // indicam linha de comentário, o que vier após elas não



será executado pelo programa. Essas // podem ser usadas durante todo o código, como forma de documentar o que cada comando ou função faz.

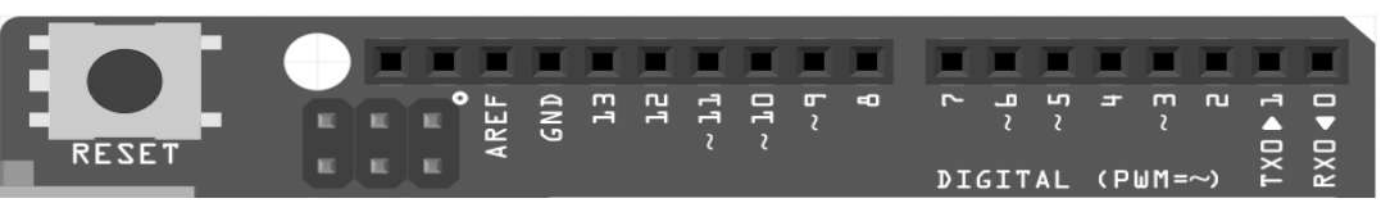
Após a primeira linha é possível inserir bibliotecas que permitirão ao compilador gerenciar outras funções. Essas bibliotecas são carregadas no IDE do Arduino para manipulação de Display, Sensores e Atuadores, dependendo da aplicação.

Entre a primeira linha e a função void setup(), serão inseridas as variáveis que serão utilizadas no durante o programa, ou seja, as variáveis são inicializadas quanto ao tipo e, se necessário, quanto ao valor, conforme discutido anteriormente.

Agora, serão tratadas as estruturas de condição, em C/C++ com em outras existem 2 estruturas básicas que são o if simples ou conjugado com if...else e a estrutura switch...case. Cada estrutura tem a sua aplicação própria, mas depende sempre do analista ou programador o seu uso. Muitos programadores preferem a estrutura switch...case para tratar com menus, sendo a estrutura if...else para tratamento com condições especificas, como intervalos, e seleções mais complexas.

Será iniciada pela mais simples: switch...case. A estrutura switch...case controla o fluxo dentro do programa, permitindo que os programadores especifiquem comandos diferentes para serem executados em situações diferentes, de acordo com os objetivos específicos. Essas especificidades serão tratadas individualmente na condição case. Em particular, uma instrução switch compara o valor de uma variável com os valores especificados nas suas instruções case.

Quando uma condição for verdadeira na estrutura case ela é executada, ou seja, uma série de comandos correspondentes àquela condição verdadeira é executada e ao término a estrutura finalizada. Portanto, essa estrutura executa somente uma opção dentre as possíveis existentes.



Estrutura de condição SWITCH...CASE:

Figura 9 – Estrutura de condição/case

```
switch (var)
{
    case label1:
     comandos; // linhas de programação que serão executadas
    break;
    case label2:
     comandos; // linhas de programação que serão executadas
    break;
    default:
     comandos; // linhas de programação que serão executadas
}
```

Fonte: Autoria Própria

Entendendo os parâmetros existentes na estrutura:

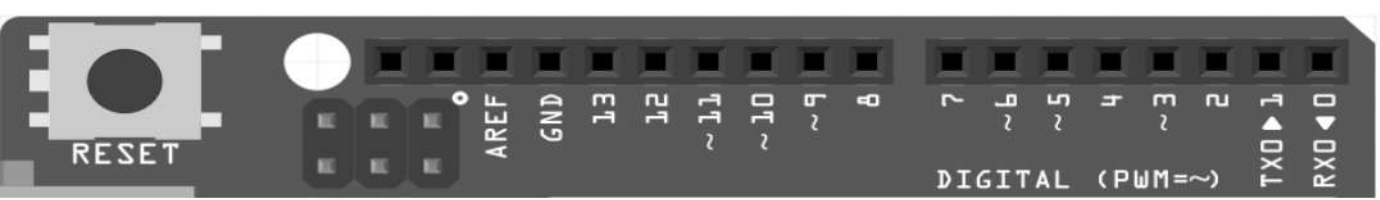
- var: uma variável cujo valor será comparado com várias condições (cases). Permitem dados do tipo: int ou char.
- label1, label2: constantes. Permitem dados do tipo: int, char.
- var: variável a ser comparada com os diferentes casos, case.
- label1, label 2, ...: um valor a ser comparado com a variável var.
- default: instrução padrão caso nenhum caso seja verificado.

Exemplo da estrutura switch...case:

Figura 10 – Exemplo de estrutura de decisão switch

```
switch (valor)
  {
     case  1:
      comandos; // linhas de programação que serão executadas
       break;
     case l2:
      comandos; // linhas de programação que serão executadas
       break;
    default:
      comandos; // se as anteriores forem falsas, executar o padrão
  }

```

Fonte: Autoria Própria



Observação: o comando break interrompe o case e o switch após todos os comandos serem executados para aquela condição verdadeira.

A próxima estrutura de repetição que iremos abordar é o **if...else** e suas variações. Essa estrutura é muito utilizada em programação por programadores experientes, uma vez que sua estrutura apesar de parecer complicada, é bem simples de implementar.

Será iniciado com a estrutura mais simples que é somente o **if.** A instrução **if** verifica uma condição e executa um comando ou uma série de comandos (instrução ou conjunto de instruções), se a condição for 'verdadeira'. A sua estrutura é bem simples:

Figura 11 – Estrutura IF

```
if (condition)
    {
      //um comando ou um conjunto de comandos
    }

```

Fonte: Autoria Própria

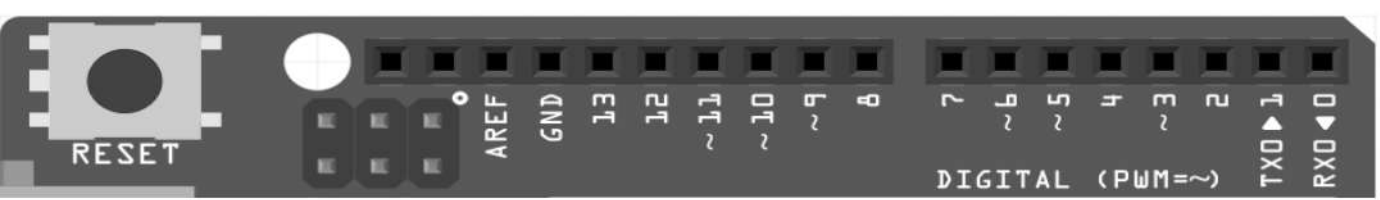
Os colchetes podem ser omitidos após uma instrução **if**. Se isso ocorrer, a próxima linha (definida pelo ponto e vírgula) se tornará a única instrução condicional. Mas, para evitar problemas de sintaxe, principalmente quando se está aprendendo é interessante colocar os colchetes.

Exemplo:

Figura 12 – Exemplo de estrutura IF

```
if (x > 120)
    {
      digitalWrite(LEDpin1, HIGH);
      digitalWrite(LEDpin2, HIGH);
    }

```

Fonte: Autoria Própria



Operadores de comparação:

Figura 13 – Operadores de Comparação

```
x == y (x é igual a y)
x != y (x é diferente de y)
x <  y (x é menor que y)
x >  y (x é maior que y)
x <= y (x é menor que ou igual a y)
x >= y (x é maior que ou igual a y)
```

Fonte: Autoria Própria

Agora, aprimorando a função **if,** utilizaremos o comando **else.** O comando else sempre deve vir acompanhado do comando **if**.

A estrutura **if ... else** permite maior controle sobre o fluxo de código (comandos), do que a instrução **if** básica, permitindo assim, que vários testes sejam agrupados. Uma cláusula **else** (caso exista) será executada se, e somente se, a condição na instrução **if** for falsa, caso contrário ela não será executada, ou seja, se isso faz um conjunto de comandos, caso contrário, faz outra sequência de comandos.

Após, um **else** pode-se continuar com outras estruturas **if...else**, de modo que vários testes possam ser executados ao mesmo tempo. Cada teste prosseguirá para o próximo, até que uma condição seja verdadeira. Quando um teste verdadeiro é encontrado, seu bloco de instrução é executado e o programa pula para a linha seguindo toda a construção **if ... else**. Se nenhuma condição for verdadeira, o bloco padrão, caso existe será executado, caso contrário a estrutura **if...else** será finalizada.

Exemplo 1: Estrutura simples IF...ELSE

Figura 14 – Exemplo de estrutura IF - ELSE

```
if (temperature >= 70)
{
      Serial.println("Deslique o sistema");
}
else
{
      Serial.println("Sistema em segurança");
}

```

Fonte: Autoria Própria

Exemplo 2: Estrutura composta IF...ELSE

Figura 15 – Exemplo de estrutura IF - ELSE

```
if (temperature >= 70)
{
     Serial.println("Deslique o sistema");
}
else if ((temperature >= 60) && (temperature < 70))
{
     Serial.println("Atenção... Sistema em aquecimento");
}
else
{
     Serial.println("Sistema em segurança");
}

```

Fonte: Autoria Própria

Agora, será trabalhado o comando de repetição **for**, este comando traz em seu escopo uma facilidade de implementação, quando comparado com outras estruturas de repetição.

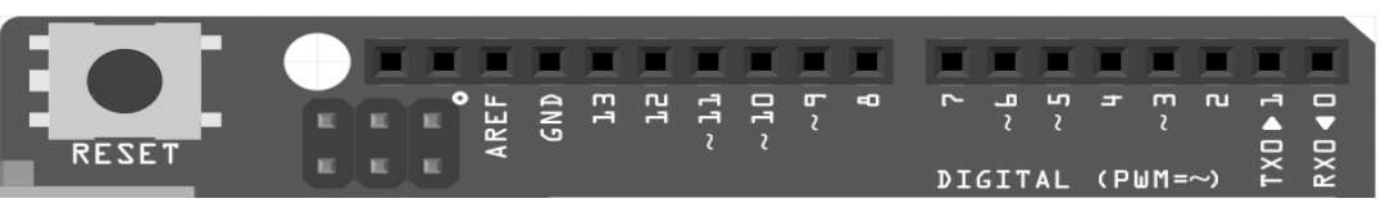
O comando **for** é usado para repetir um bloco de comandos existentes entre chaves (delimitação do escopo do **for**). Um contador de incremento é geralmente usado para incrementar e finalizar o loop (a repetição). A instrução **for** é útil para qualquer operação repetitiva e é frequentemente usada em combinação com vetores e matrizes, para manipulação com muitos dados.

Estrutura básica do **for**:

Figura 16 – Estrutura FOR

```
for (inicializaçàò; teste da condição; incremento / decremento)
 {
        comandos; // conjunto de instruções.
 }

```

Fonte: Autoria Própria



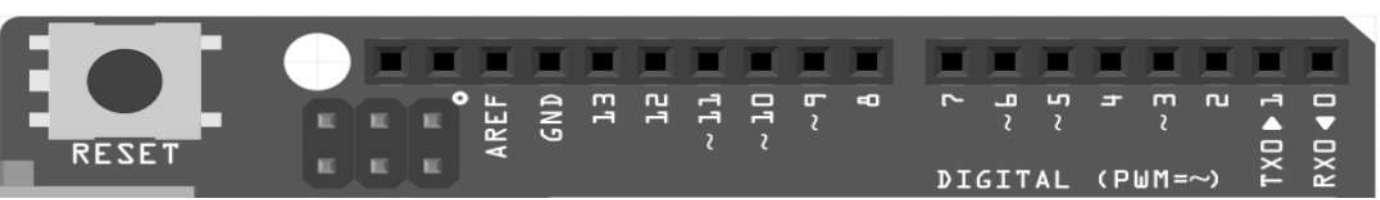
Exemplo de uma estrutura **for**:

Figura 17 – Exemplo de estrutura FOR

```
for (int a = 0; a <= 10; a = a + 1)  // inicializa; teste; incrementa ou decrementa
  {
    digitalWrite(13, HIGH);   // escreve na porta 13 sinal ALTO = 5 volts
    delay(3000);              // deixa acesso por 3s
    digitalWrite(13, LOW);    // escreve na porta 13 sinal BAIXA = 0 volts
    delay(3000);              // deixa apagado por 3s
  }

```

Fonte: Autoria Própria

Se necessário podem existir vários for, um dentro do outro, desde que, o programador tenho a lógica estruturada.



4. Atividades Práticas

4.1 - 15 Projetos com Arduino.

Atividade 1 - Por onde começo?

Objetivo da aula: Configurar o ambiente de programação do Arduino (Software IDE) para escrever o primeiro código.

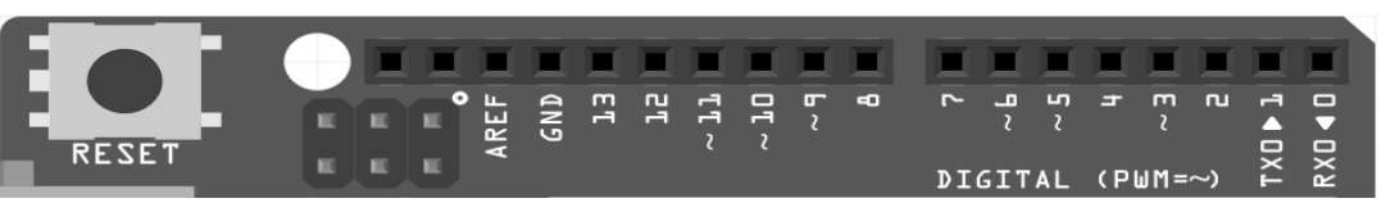
Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.

1º passo: Faça do download a IDE do Arduino acessando o endereço oficial: www.arduino.cc, na aba download e instale em seu desktop ou notebook.

Figura 18 – Página de download

Fonte: Arduino.cc



2º passo: Execute a IDE do Arduino pelo atalho criado na Área de Trabalho.

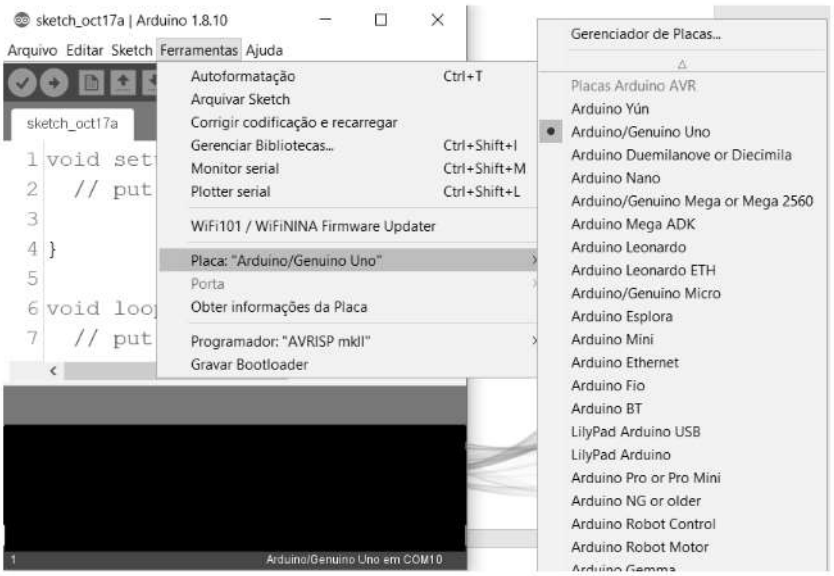
Figura 19 – Acessando a IDE do Arduino

Fonte: Arduino.cc

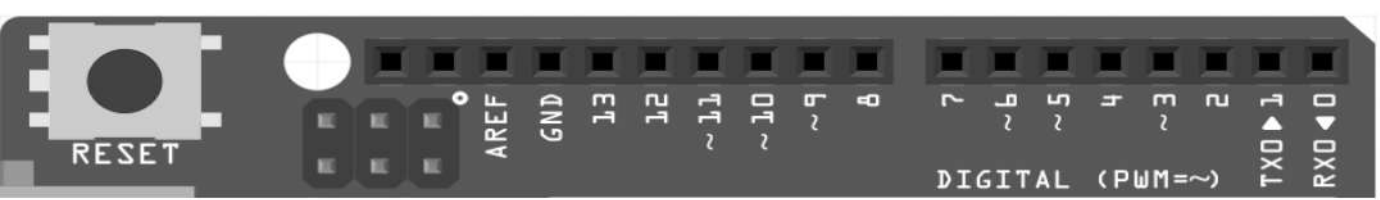
3º passo: Conecte a placa do **Arduino UNO** na porta USB de seu computador utilizando o **cabo USB A/B azul**. Em instantes o computador reconhecerá o Hardware.

4º passo: Selecione a placa: Vá até **Ferramentas > Placa**, e selecione **Arduino/Genuino UNO.**

Figura 20 - Configurando o Arduino identificando a placa Arduino



Fonte: Arduino.cc



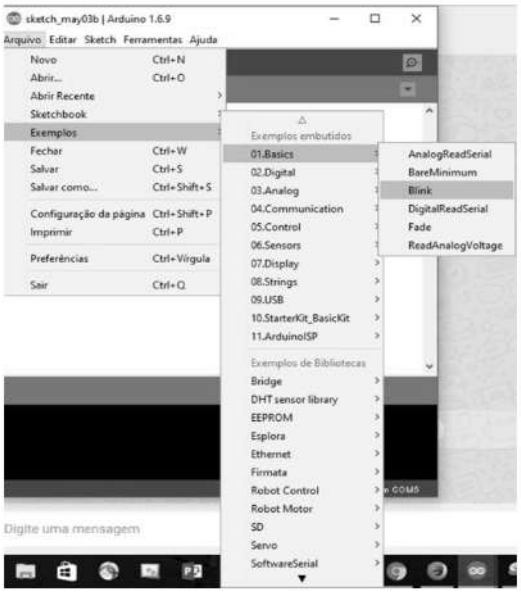
5º passo: Ainda no menu **Ferramentas**, em **Porta**, selecione a **porta COM** a qual o Arduino está utilizando:

Figura 21 – Configurando a porta de comunicação – Computador e o Arduino

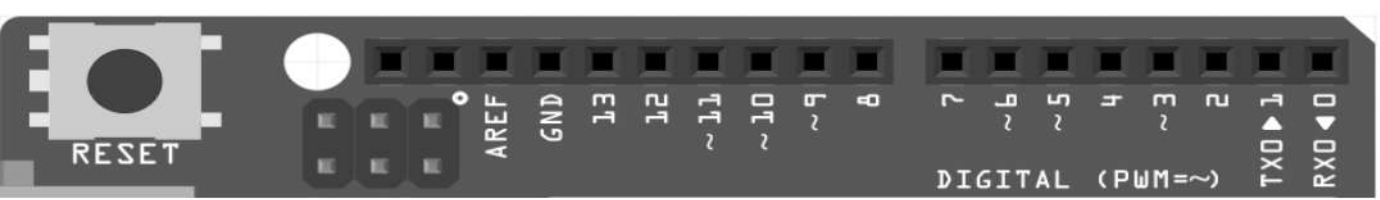
Fonte: Arduino.cc

6º passo: Nos menus, vá até **Arquivo>Exemplos>01.Basics>Blink** , para abrir um código de exemplo.

Figura 22 – Carregando um arquivo exemplo

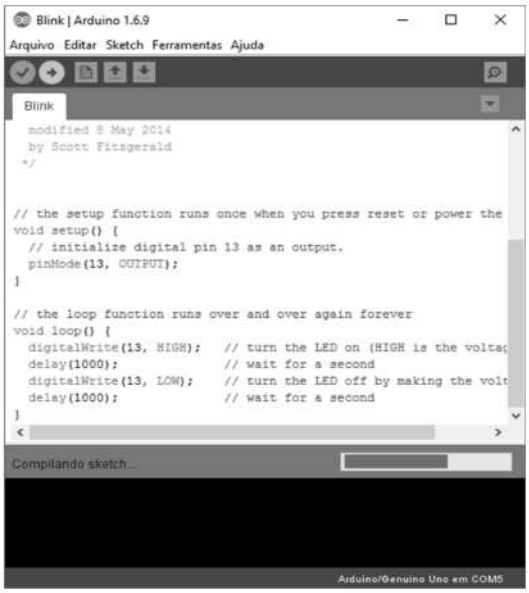


Fonte: Arduino.cc



7º passo: No canto superior esquerdo do programa, clique no botão **UPLOAD** para compilar e carregar o código na memória do Arduino.

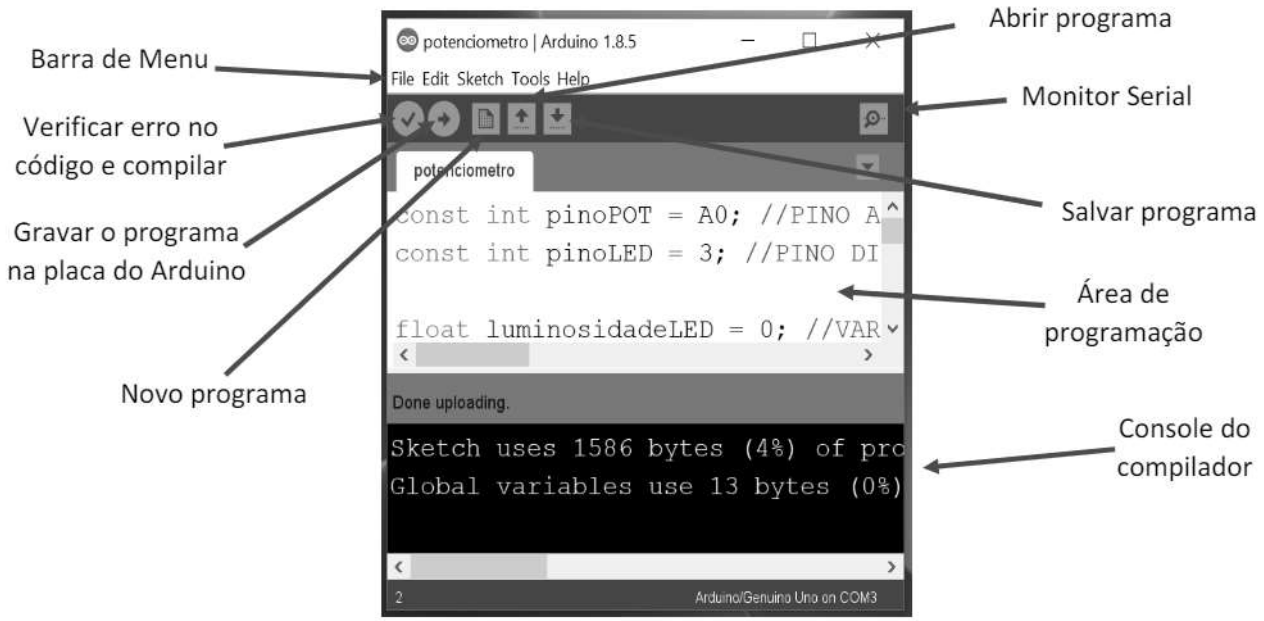
Figura 23 - Carregando o Blink para o Arduino (gravar o programa na placa)



Fonte: Arduino.cc

8º passo: IDE do Arduino (Conhecendo o ambiente de programação).

Figura 24 - IDE Arduino



Fonte: Arduino.cc

Atividade 2 - Hello World

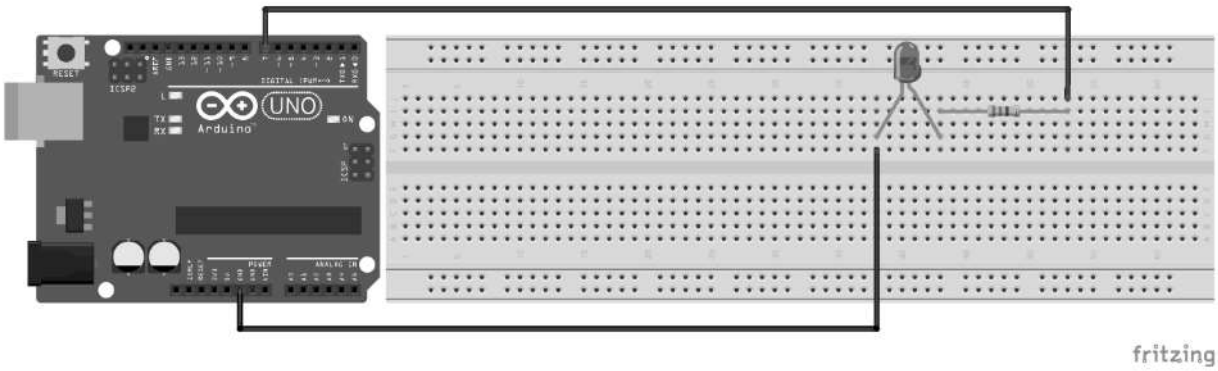
Objetivo da aula: Programar o Arduino para acender e apagar um LED.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 2 jumpers macho x macho;
- 1 led 5mm vermelho;
- 1 Resistor 330Ω;

Montagem do circuito:

Figura 25 – Montagem do circuito Helloword



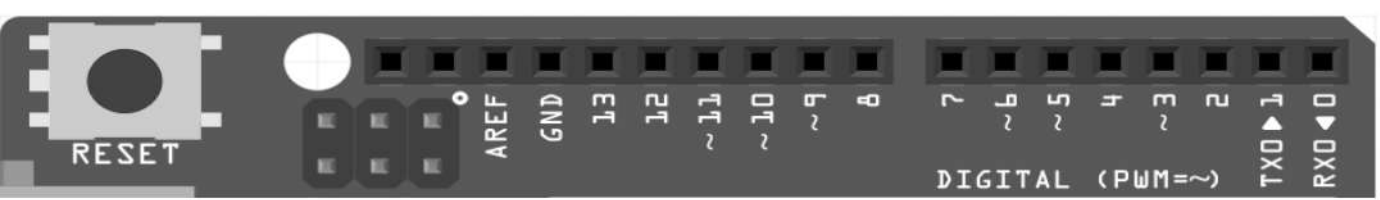
Fonte: Autoria Própria

Programa Hello World:

Figura 26 - Código fonte Hello World

```
#define led 7  // definindo a variável led para a porta digital 7

void setup()
{
pinMode(led, OUTPUT);   // configurando a porta 7 como saída de dados
}

void loop()
{
digitalWrite(led, HIGH); // enviando um sinal alto para a porta 7 - acendendo o led
delay(1000);   // tempo de espera - 1 segundo
digitalWrite(led, LOW); // enviando um sinal baixo para a porta 7 - acendendo o led
delay(1000);   // tempo de espera - 1 segundo

}

```

Fonte: Autoria Própria



Atividade 3 - Semáforo com Leds

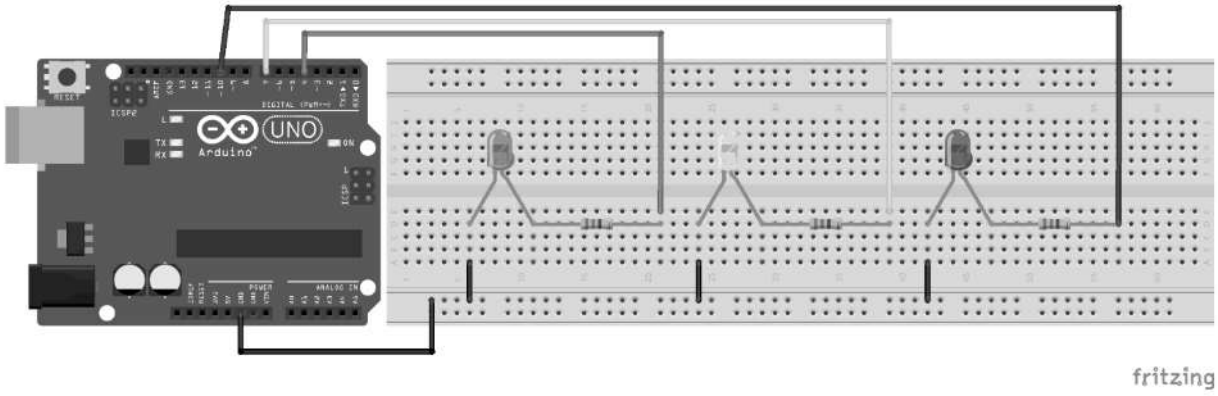
Objetivo da aula: Montar um circuito e programar o Arduino para controlar leds como um semáforo simples.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 7 jumpers macho x macho;
- 1 led 5mm verde;
- 1 led 5mm amarelo;
- 1 led 5mm vermelho;
- 3 Resistor 330Ω;

Montagem do circuito:

Figura 27 – Montagem do circuito Semáforo com leds



Fonte: Autoria Própria

Programa Semáforo:

Figura 28 – Código fonte Hello World

```
#define vermelho 10
#define verde 4
#define amarelo 7
void setup()
{
pinMode(vermelho, OUTPUT);
pinMode(verde, OUTPUT);
pinMode(amarelo, OUTPUT);
}
void loop()
{
digitalWrite(vermelho, HIGH);
delay(5000);
digitalWrite(vermelho, LOW);
digitalWrite(verde, HIGH);
delay(3000);
digitalWrite(verde, LOW);
digitalWrite(amarelo, HIGH);
delay(1000);
digitalWrite(amarelo, LOW);
}
```

Fonte: Autoria Própria

Atividade 4 - Sensor de Luminosidade LDR

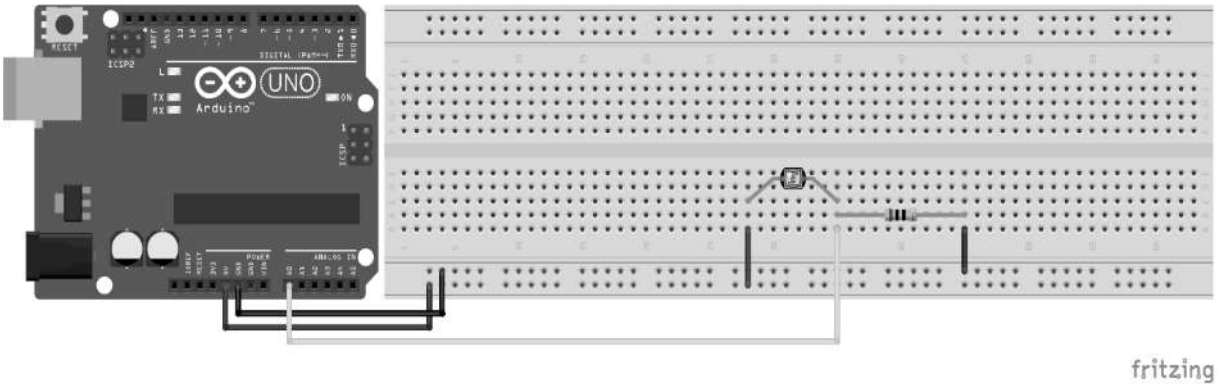
Objetivo da aula: Montar um circuito e programar o Arduino para fazer leitura de um LDR.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 5 jumpers macho x macho;
- 1 Resistor 10KΩ;
- 1 LDR.

Montagem do circuito:

Figura 29 – Montagem do circuito LDR



Fonte: Sensor LDR

Figura 30 - Código fonte Hello World

```
#define ldr A0   // definindo a variavel ldr como porta analogica A0
void setup()
{
pinMode(ldr, INPUT);

Serial.begin(9600); // iniciando a comunicação serial entre o Arduino e Computador.
}
void loop()
{
int valor = analogRead(ldr);
int luz = map(valor, 0, 1023, 100, 0);
// função "map(x,x_min,x_max,y_max,y_min)" mapeia um intervalo numérico em função de outro .
  Serial.print("Iluminação em: "); // escreve em uma linha no monitor serial algum argumento.
  Serial.print(luz);
  Serial.println("%"); // escreve seu argumento pulando linhas no monitor serial.
  delay(1000);
}
```

Fonte: Autoria Própria

Atividade 5 - Juntando tudo (LDR + Led)

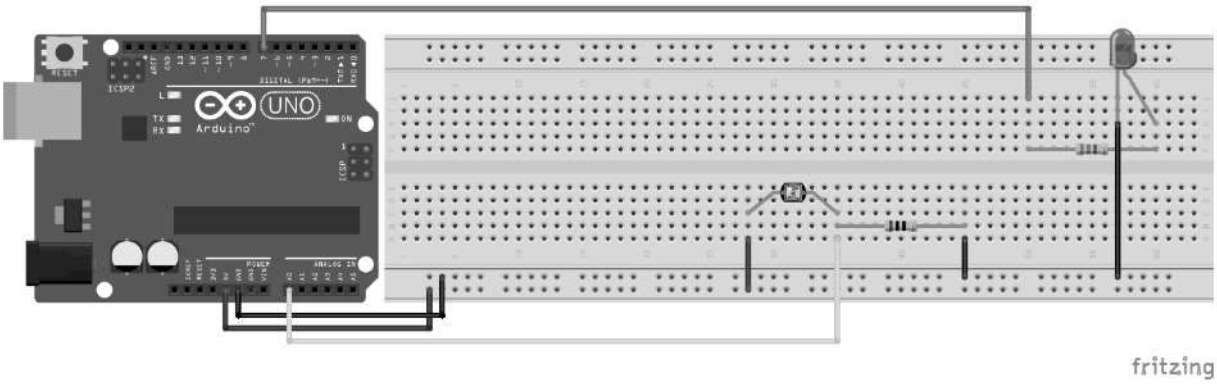
Objetivo da aula: Montar um circuito e programar o Arduino para fazer leitura de um LDR.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 1 led 5mm vermelho;
- 7 jumpers macho x macho;
- 1 Resistor 330Ω;
- 1 Resistor 10KΩ;
- 1 LDR.

Montagem do circuito:

Figura 31 – Montagem do circuito LDR + LED



Fonte: Autoria Própria

Figura 32 – Código Fonte LDR + LED

```
#define ldr A0
#define led 7
void setup()
{
pinMode(ldr, INPUT);
pinMode(led, OUTPUT);
Serial.begin(9600);
}
void loop()
{
    int valor = analogRead(ldr);
int luz = map(valor, 0, 1023, 100, 0);
Serial.print("Iluminacao em: ");
Serial.print(luz);
Serial.println("%");
if (luz >= 80)
{
digitalWrite(led, HIGH);
}
else
{
digitalWrite(led, LOW);
}
delay(1000);
}
```

Fonte: Autoria Própria

Atividade 6 - Piezo Elétrico (Buzzer)

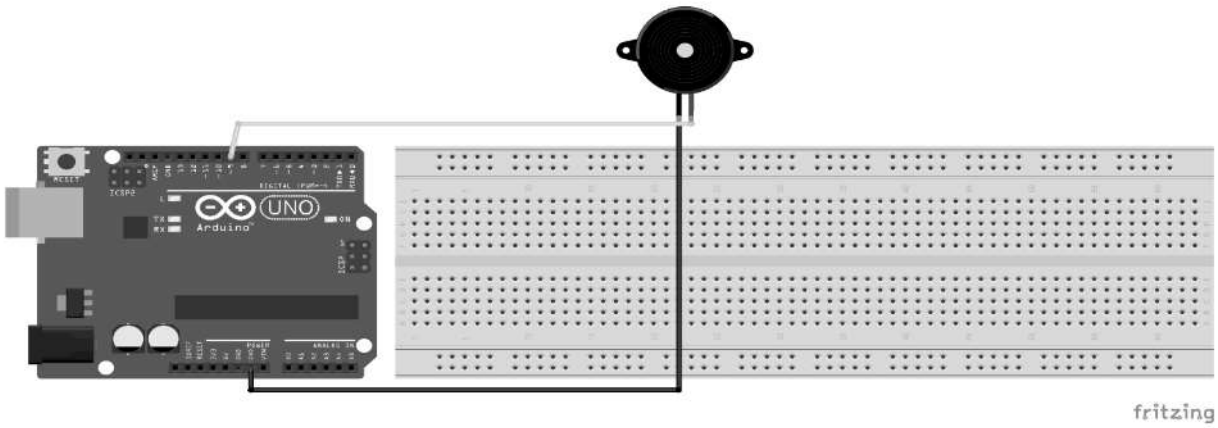
Objetivo da aula: Montar um circuito e programar o Arduino para emitir sons a partir de Piezo Elétrico (Buzzer).

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 2 jumpers macho x macho;
- 1 Buzzer.

Montagem do circuito:

Figura 33 – Montagem do circuito elétrico



Fonte: Autoria Própria

Figura 34 – Código Fonte Pino elétrico

```
#define buzzer 9
void setup()
{
pinMode(buzzer,OUTPUT);
}
void loop()
{
/* Declarando a frequência em
   Hertz de cada nota musical.*/
int nota_do = 262;
int nota_re = 294;
int nota_mi = 330;
int nota_fa = 349;
int nota_sol = 392;
int nota_la = 440;
int nota_si = 472;

/* A função "tone" reproduz um sinal
   de uma certa frequência no pino do buzzer. */
tone(buzzer, nota_do);
delay(500);
tone(buzzer, nota_re);
delay(500);
tone(buzzer, nota_mi);
delay(500);
tone(buzzer, nota_fa);
delay(500);
tone(buzzer, nota_sol);
delay(500);
tone(buzzer, nota_la);
delay(500);
tone(buzzer, nota_si);
delay(500);
}
```

Fonte: Autoria Própria

Atividade 7 - Juntando tudo Buzzer+LDR+Led

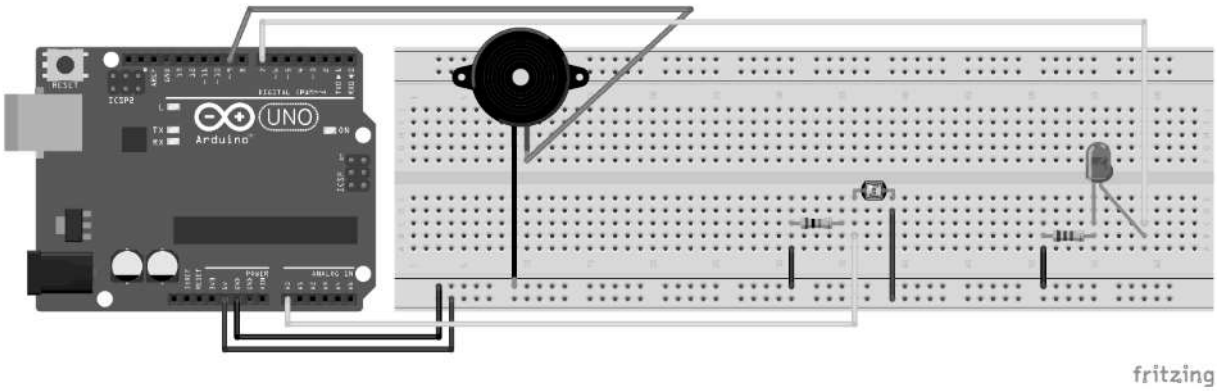
Objetivo da aula: Montar um circuito e programar o Arduino para mensurar a luminosidade ambiente e a converta em porcentagem, e caso o percentual de luminosidade esteja acima de 80%, ative um sinal luminoso e outro sonoro.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 1 led 5mm vermelho;
- 9 jumpers macho x macho;
- 1 Resistor 330Ω;
- 1 Resistor 10KΩ;
- 1 Buzzer.

Montagem do circuito:

Figura 35 – Montagem do circuito Buzzer LDR + LED

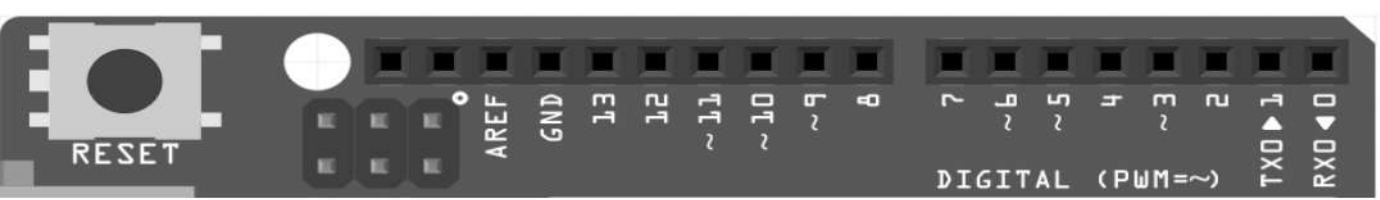


Fonte: Autoria Própria

Figura 36 – Código Fonte Buzzer + LDR + LED

```
#define ldr A0
#define led 7
#define buzzer 9
void setup()
{
pinMode(ldr, INPUT);
pinMode(led, OUTPUT);
pinMode(buzzer, OUTPUT);
Serial.begin(9600);
}
void loop()
{
int valor = analogRead(ldr);
int luz = map(valor, 0, 1023, 100, 0);
Serial.print("Iluminacao em: ");
Serial.print(luz);
Serial.println("%");
if (luz >= 80)
{
digitalWrite(led, HIGH);
digitalWrite(buzzer, HIGH);
}
else
{
digitalWrite(led, LOW);
digitalWrite(buzzer, LOW);
}
delay(1000);
}
```

Fonte: Autoria Própria



Atividade 8 - Sensor Ultrassônico (HC-SR04)

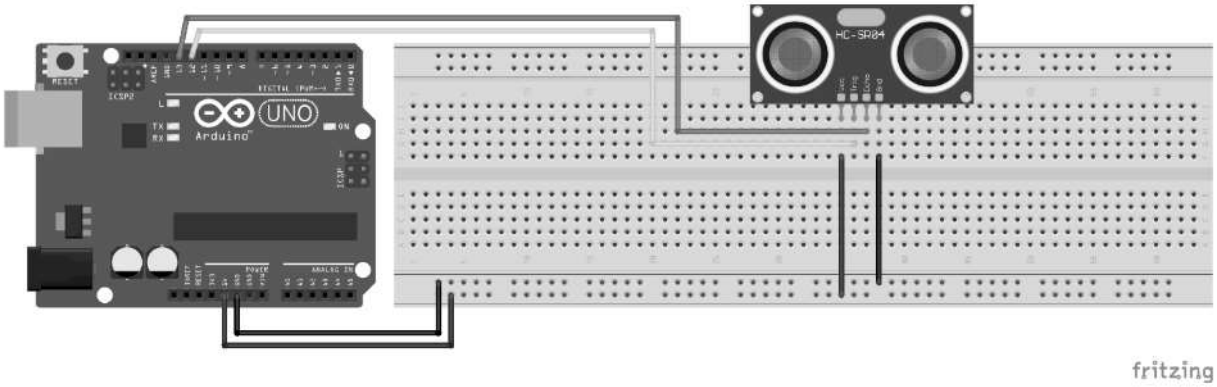
Objetivo da aula: Montar um circuito e programar o Arduino para mensurar leituras de distância em centímetros no monitor serial.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 6 jumpers macho x macho;
- 1 Sensor ultrassônico HC-SR04;

Montagem do circuito:

Figura 37 – Montagem do circuito Sensor Ultrassônico

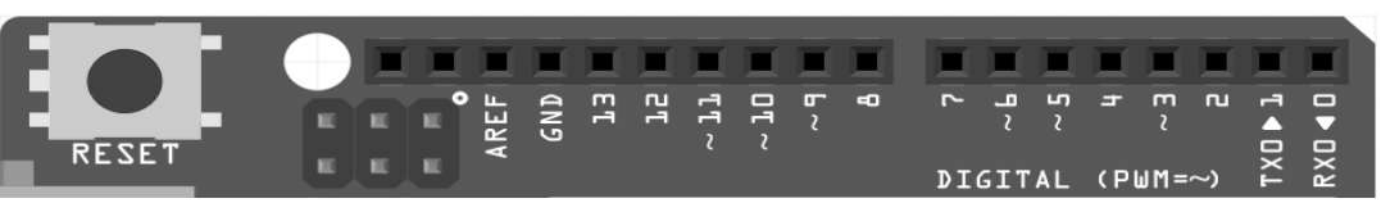


Fonte: Autoria Própria

Figura 38 – Código Fonte Sensor Ultrassônico

```
1 #include <Ultrasonic.h>
2 Ultrasonic ultrasonic(12, 13);
3
4 void setup()
5 {
6         Serial.begin(9600);
7 }
8 void loop()
9 {
10        Serial.print("Distancia em CM: ");
11        Serial.println(ultrasonic.distanceRead());
12        delay(1000);
13 }
```

Fonte: Autoria Própria



Atividade 9 - Servo motor

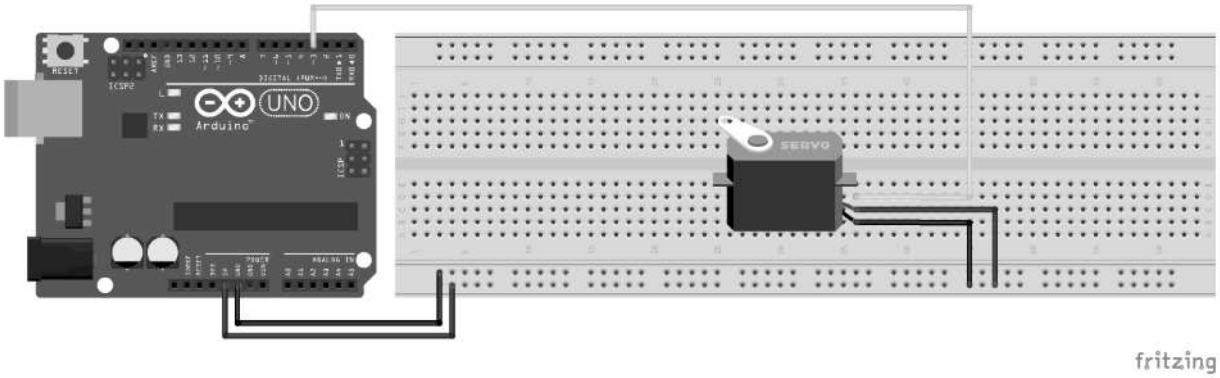
Objetivo da aula: Montar um circuito e programar o Arduino para que o servo motor movimente de 0º a 180º em loop.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 5 jumpers macho x macho;
- 1 Servo motor.

Montagem do circuito:

Figura 39 – Montagem do circuito Servo Motor



Fonte: Autoria Própria

Figura 40 – Código Fonte do circuito Servo Motor

```
1 #include <Servo.h>  // declarando a biblioteca do servo motor.
2 Servo servo1;       //cria o objeto servo1 da classe Servo.
3 void setup() {
4   servo1.attach(3); // declara que o fio amarelo de controle
5 }                   // do servo está no pino 3 do arduino.
6 void loop() {
7   int graus;
8   // girar de 0 a 180°.
9   for (graus = 0; graus <= 180; graus += 1) {
10    servo1.write(graus);
11    delay(15); }
12  // girar de 180° a 0°.
13  for (graus = 180; graus >= 0; graus -= 1) {
14    servo1.write(graus);
15    delay(15);  }
16 }
```

Fonte: Autoria Própria

Atividade 10 - Servo motor + Potenciometro

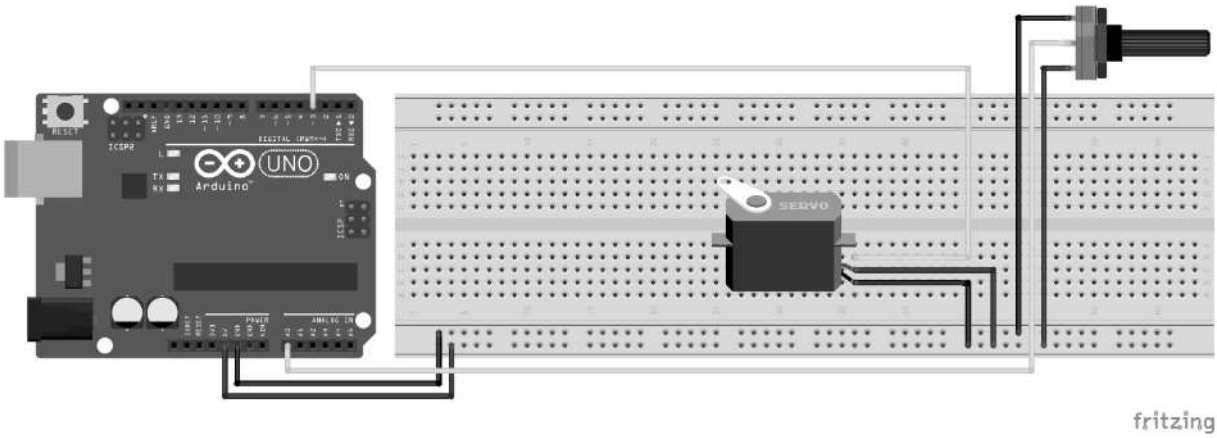
Objetivo da aula: Montar um circuito e programar o Arduino para controlar um servo a partir de um potenciômetro.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 5 jumpers macho x macho;
- 1 Servo motor;
- 1 Potenciômetro 50KΩ.

Montagem do circuito:

Figura 41 – Montagem do circuito Servo Motor + Potenciômetro



Fonte: Autoria Própria

Figura 42 – Código Fonte do circuito Servo motor + Potenciômetro

```
1 #include <Servo.h>
2 Servo servo1;
3
4  void setup()
5  {
6        servo1.attach(3);
7  }
8  void loop()
9  {
10       int angulo = analogRead(0);
11       angulo = map(angulo, 0, 1023, 0, 180);
12       servo1.write(angulo); //faz o servo girar em função do ângulo.
13       delay(20);
14 }
15
```

Fonte: Autoria Própria

Atividade 11 - Display LCD + Potenciômetro

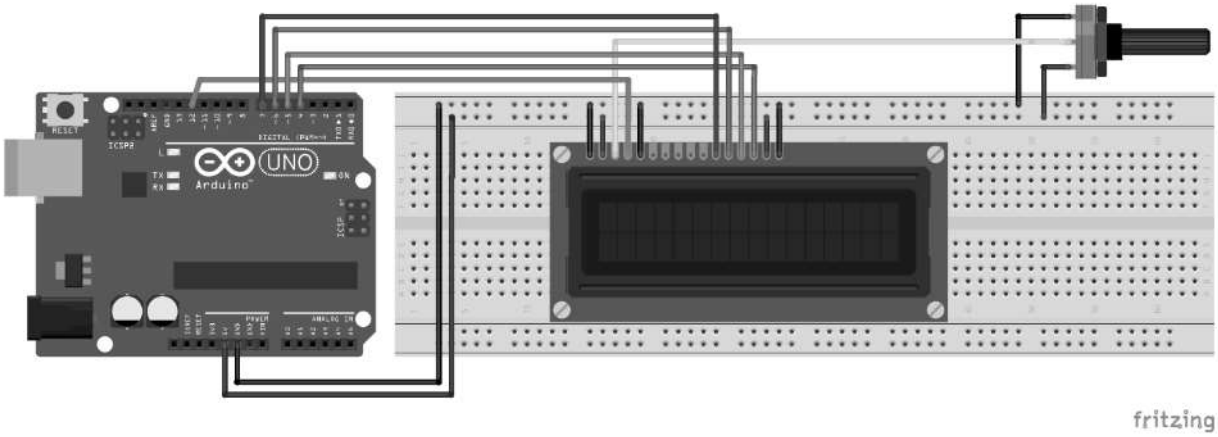
Objetivo da aula: Montar um circuito e programar o Arduino para mostrar na tela de um display LCD o seu nome e sobrenome.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 15 jumpers macho x macho;
- 1 Display LCD 16x2;
- 1 Potenciômetro 50KΩ.

Montagem do circuito:

Figura 43 – Montagem do circuito Display LCD + DOT



Fonte: Autoria Própria

Figura 44 – Código Fonte do circuito Dislpay LCD + DOT

```
#include <LiquidCrystal.h>
// declara os pinos de dados para o funcionamento do display.
LiquidCrystal lcd(12, 11, 7, 6, 5, 4);
void setup()
{
// declara o tamanho do display (16x2 caracteres).
lcd.begin(16, 2);
}
void loop()
{
  lcd.clear(); // limpa a tela do display
  lcd.setCursor(0, 0); // posiciona o cursor
  lcd.print("seu nome"); // escreve na tela
  lcd.setCursor(0, 1);
  delay(500);
  lcd.print("seu sobrenome");
  delay(5000);
```

Fonte: Autoria Própria

Atividade 12 - DHT11

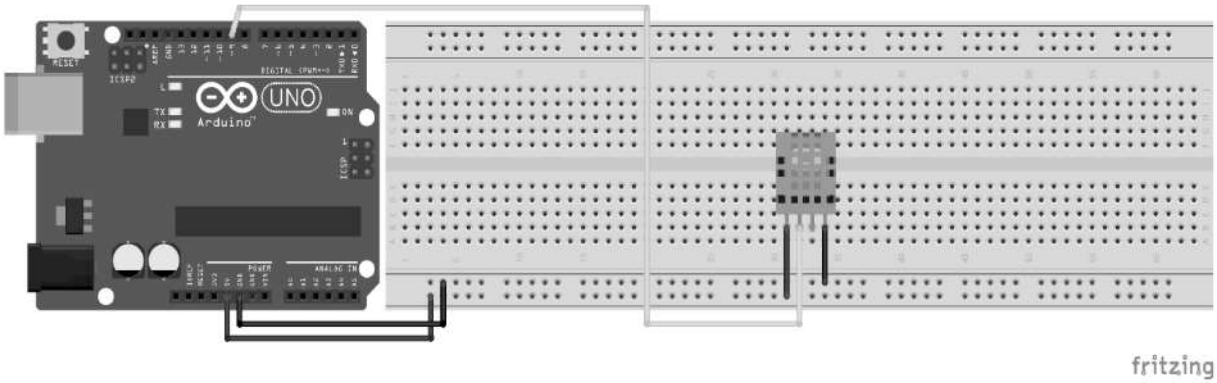
Objetivo da aula: Montar um circuito e programar o Arduino para mostrar na tela de um display LCD o seu nome e sobrenome.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 5 jumpers macho x macho;
- 1 DHT11.

Montagem do circuito:

Figura 45 – Montagem do circuito DHT 11

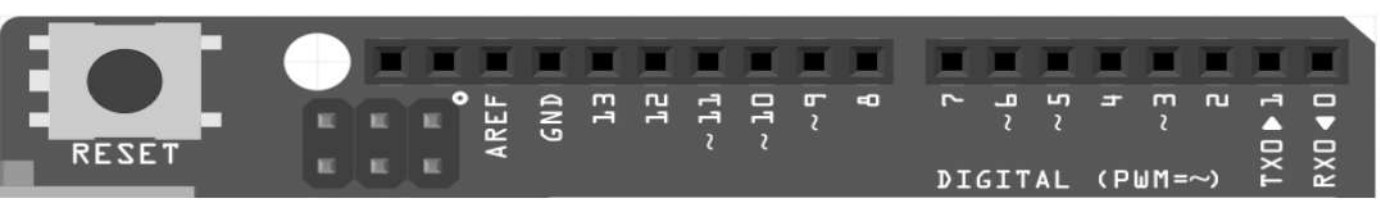


Fonte: Autoria Própria

Figura 46 – Código Fonte do circuito DHT 11

```
#include "DHT.h"
#define DHTPIN 9
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
  Serial.println("DHT11 teste!");
}
void loop()
{
  delay(2000);
  float u = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(u) || isnan(t))
{
    Serial.println("Falha ao ler o sensor!");
    return;
   }
  else
  {
    Serial.print("Temperatura = ");
    Serial.println(t);
    Serial.print("Umidade = ");
    Serial.println(u);
   }
}
```

Fonte: Autoria Própria



Atividade 13 - Juntando tudo (LDC+DHT11)

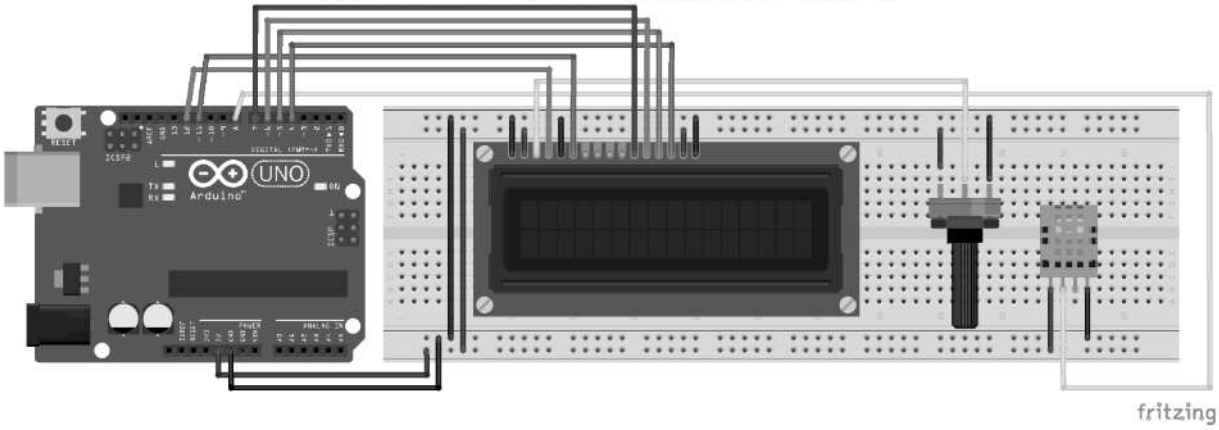
Objetivo da aula: Montar um circuito e programar o Arduino para mostrar na tela de um display LCD o seu nome e sobrenome.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 18 jumpers macho x macho;
- 1 DHT11.
- 1 Display LCD 16x2;
- 1 Potenciômetro 50KΩ

Montagem do circuito:

Figura 47 – Montagem do circuito LED + DHT 11



Fonte: Autoria Própria

Figura 48 – Código Fonte do circuito LED + DHT 11

```
#include <LiquidCrystal.h>
#include "DHT.h"
#define DHTPIN 8
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal lcd(12, 11, 7, 6, 5, 4);
void setup()
{
  Serial.begin(9600);
  lcd.begin(16, 2);
  dht.begin();
}
 void loop()
{
  delay(2000);
  float u = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(u) || isnan(t)) {
    Serial.println("Falha ao ler o sensor!");
    return;;
  }
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("U: ");
  lcd.setCursor(4, 0);
  lcd.print(u);
  lcd.setCursor(0, 1);
  lcd.print("T: ");
  lcd.setCursor(4, 1);
  lcd.print(t);
  delay(5000);
}
```

Fonte: Autoria Própria

Atividade 14 - Sensor de som

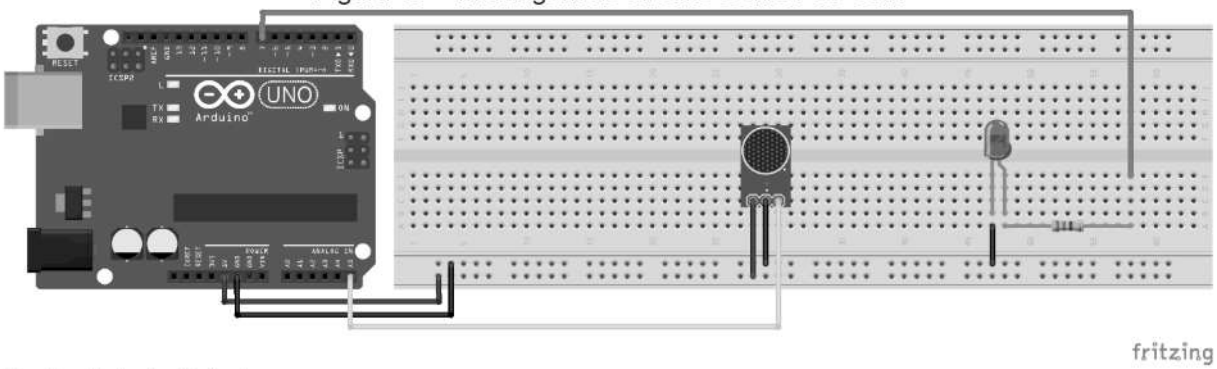
Objetivo da aula: Montar um circuito e programar o Arduino para receber o sinal sonoro e acender um Led quando a intensidade do sinal sonoro do ambiente for alto.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 7 jumpers macho x macho;
- 1 led 5mm vermelho;
- 1 Resistor 330Ω;
- 1 modulo sensor de som.

Montagem do circuito:

Figura 49 – Montagem do circuito Sensor de Som



Fonte: Autoria Própria

Figura 50 – Código Fonte do circuito Sensor de Som

```
1  #define led 7
2  void setup()
3  {
4      pinMode(A5, INPUT);
5      pinMode(led, OUTPUT);
6  }
7  void loop()
8  {
9      int valor = analogRead(A5);
10     int som = map(valor, 0, 1023, 100, 0);
11     if (som <= 70 ) {
12         digitalWrite(led, LOW); }
13    else
14    { digitalWrite(led, HIGH);  }
15    delay(50);
16 }
```

Fonte: Autoria Própria

Atividade 15 - Juntando tudo (DHT+Led)

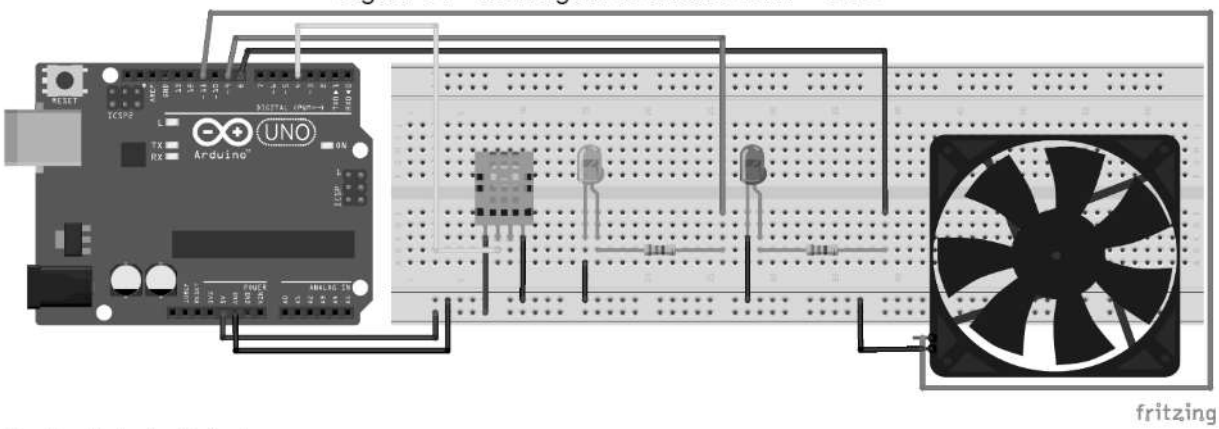
Objetivo da aula: Montar um circuito e programar o Arduino para simular um sistema de controle de temperatura ambiental onde caso a temperatura estiver acima de 27ºC acionar um cooler e ligar um led vermelho. Caso contrário, ligar led verde.

Lista de materiais:

- Notebook;
- Arduino UNO R3 + Cabo USB A/B.
- Protoboard;
- 7 jumpers macho x macho;
- 1 led 5mm vermelho;
- 1 led 5mm verde;
- 2 Resistor 330Ω;
- 1 cooler.

Montagem do circuito:

Figura 51 – Montagem do circuito DHT + LED



Fonte: Autoria Própria

Figura 52 – Código Fonte do circuito DHT + LED

```
#include "DHT.h"
#define DHTPIN 4
#define led_verde 9
#define led_vermelho 8
#define ventilador 11

#define DHTTYPE DHT11
//#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  pinMode(led_verde, OUTPUT);
  pinMode(led_vermelho, OUTPUT);
  pinMode(ventilador, OUTPUT);
  Serial.begin(9600);
  dht.begin();
  Serial.println("DHT11 ligado!!");
}
void loop() {
  delay(2000);
  float u = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(u) || isnan(t)) {
    Serial.println("Falha ao ler o sensor!");
    return;
  }
  if (t >= 27.0) {
    digitalWrite(led_vermelho, HIGH);
    digitalWrite(led_verde, LOW);
    digitalWrite(ventilador, HIGH);
  }
  else {
    digitalWrite(led_vermelho, LOW);
    digitalWrite(led_verde, HIGH);
    digitalWrite(ventilador, LOW);
  }
}
```

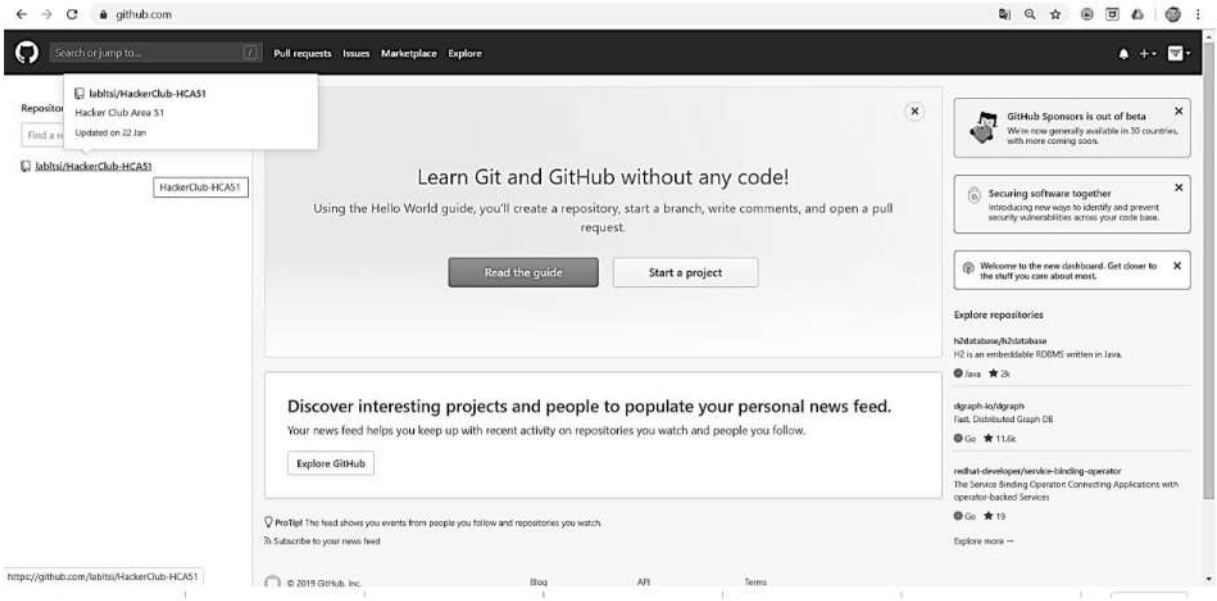
Fonte: Autoria Própria

5. Github

O GitHub é uma plataforma de hospedagem de código para controle de versão e colaboração. Permite que várias pessoas trabalhem juntas em projetos de qualquer lugar.

Foi criado um repositório, https://github.com/labltsi/HackerClub-HCA51, para um fluxo de trabalho *Pull Request* do GitHub, uma maneira popular de criar e revisar códigos. Assim todos podem contribuir na codificação dos códigos das atividades deste livro.

Figura 53 – Plataforma Github



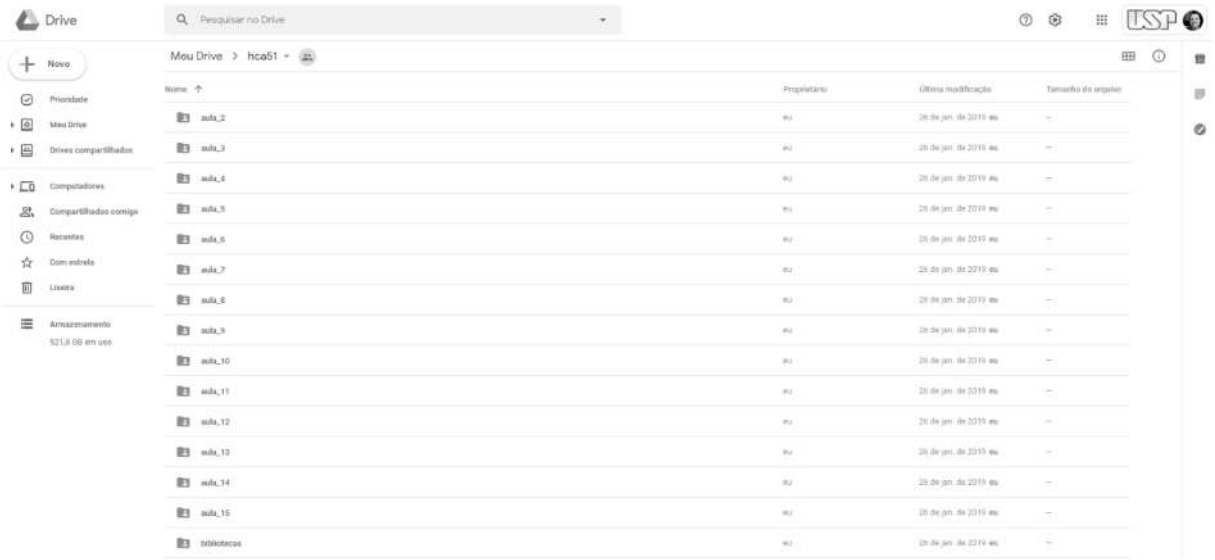
Fonte: Github.com

Acesse: http://bit.ly/2Q1L464

6. Google Drive

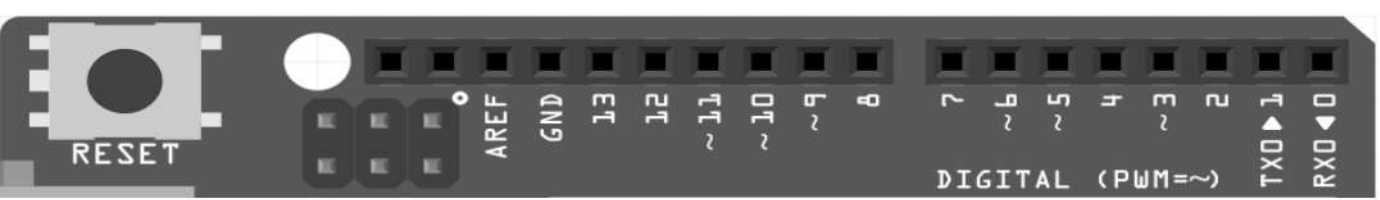
Para facilitar o processo de aprendizagem foram disponibilizados no Google drive do Hacker Club Área 51 os códigos e bibliotecas de todas as aulas deste livro.

Figura 54 – Serviço de Armazenamento Google Drive



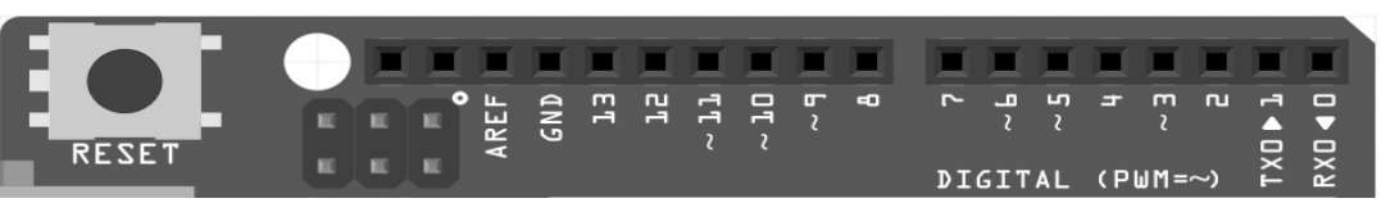
Fonte: drive.google.com

Acesse: http://bit.ly/2CwYmzr



7. Agradecimentos

Os Autores gostariam de agradecer a todos que direta ou indiretamente apoiaram a construção desta obra, em especial a Pró-Reitoria de Cultura e Extensão Universitária (PRCEU), a Comissão de Cultura e Extensão da FZEA (CCEx/FZEA) e aos alunos integrantes do Hacker Club e do Laboratório de Tecnologia e Sistemas de Informação (LTSI).



8. Referências

ARDUINO.CC. **What is Arduino?** Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 06 abr. 2020.

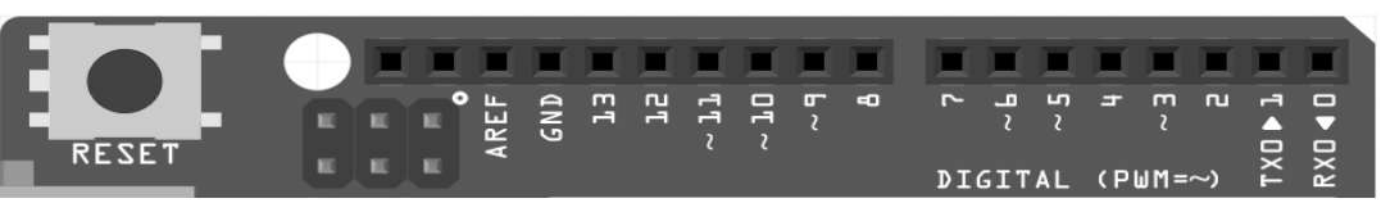
CIRCUITO.IO. **Arduino Uno Pinout**. Disponível em: <https://www.circuito.io/blog/arduino-uno-pinout/> Acesso em: 06 abr. 2020.

FRITIZING. **Software Fritzing BETA**. Disponível em: <http://fritzing.org/download/> Acesso em: 06 abr. 2020.

PICTRONICS. **Arduino Uno Pinout Diagram**. Disponível em: <http://www.pictronics.com.br/index.php?option=com_content&view=article&id=98:curso-basico-de-arduino-2&catid=43:eletronica-e-automacao&Itemid=2> Acesso em: 06 abr. de 2020.

VIDADESILICIO. **O que é e como funciona o Arduino Uno R3**. Disponível em: <https://portal.vidadesilicio.com.br/o-que-e-arduino-e-como-funciona/> Acesso em: 06 abr. 2020.

REZENDE, D. A.; ABREU, A. F. 2013. **Tecnologia da Informação aplicada a Sistemas de Informação Empresariais**. São Paulo: Atlas, 9 ed., 2013.



http://bit.ly/2NTEz2B

http://bit.ly/34MSezt

ISBN: 978-65-87023-05-2 (e-book)