

## ▶ UC15: Realizar programação em placas microcontroladoras e redes de sensores para IoT.

CARGA HORÁRIA: 60 HORAS

### Indicadores

1. Instala software para desenvolvimento em placas microcontroladoras conforme instruções do fabricante.
2. Emprega linguagem de programação de acordo com modelo de microcontrolador adotado e redes de sensores para IoT.
3. Desenvolve softwares de acordo a linguagem de programação selecionada.
4. Elabora código para microcontroladores conforme a necessidade de controle de sistemas eletrônicos.
5. Realiza a compilação e depuração do código desenvolvido de acordo com orientações técnicas da plataforma utilizada.
6. Testa o código desenvolvido de acordo com funcionalidades esperadas para o programa.

### Elementos da competência

#### CONHECIMENTOS

- IDEs: instalação para programação de microcontroladores.
- Drivers de acesso: instalação em computadores para dispositivos microcontrolados.
- Microcontroladores: integração de sensores e atuadores, placa microcontroladora, carga de programas pré-compilados para testes de microcontroladores.
- Programação: linguagem, compilação e carga do programa em placas microcontroladas.
- Testes: programas carregados em plataformas de microcontroladores.

#### HABILIDADES

- Organizar materiais, ferramentas, instrumentos, documentos e local de trabalho.
- Analisar as etapas do processo de trabalho.
- Comunicar-se de maneira assertiva.
- Elaborar documentos técnicos.
- Administrar as etapas do processo de desenvolvimento de software embarcado.
- Planejar e codificar scripts para execução em microcontroladores.
- Interpretar e solucionar erros e problemas que impeçam a execução de um programa.
- Mediar conflitos nas situações de trabalho.
- Planejar e configurar sistemas embarcados.

---

#### ATITUDES/VALORES

- Atitude colaborativa com membros da equipe, parceiros e clientes.
  - Postura profissional no ambiente de trabalho.
  - Proatividade na resolução de problemas.
  - Zelo na apresentação pessoal e postura profissional.
  - Zelo na execução de procedimentos técnicos.
  - Zelo pela higiene, limpeza e conservação na utilização dos equipamentos, instrumentos e ferramentas.
- 

#### 1. Recomendações bibliográficas:

KARVINEN, Kimmo; KARVINEN, Tero. Primeiros Passos com Sensores. Primeira edição. São Paulo: Novatec, Outubro 2014.

BANZI, Massimo; SHILOH, Michael. Primeiros Passos com o Arduino. Segunda edição. São Paulo: Novatec, Maio 2015.

#### 2. O que é Open Source Hardware?

De modo geral, os Open Hardwares são plataformas cujo design, código ou tecnologia podem ser replicados, customizados e distribuídos gratuitamente. Ele é parte fundamental da cultura *maker*, que tem a colaboração e a sustentabilidade como dois de seus pilares fundamentais.

Existem dois tipos de licenças, com características específicas, as *copyleft* e as permissivas. Vamos falar mais sobre elas abaixo, mas, de modo geral, elas seguem os seguintes princípios: Hardware pode ser modificado, usado comercialmente e distribuído; hardware pode ser modificado e usado de forma privada; a licença e os direitos precisam ser incluídos no hardware; os autores não provêm garantias.

##### 2.1. Hardware ou software?

Os *softwares* livres ficaram muito populares no início nos anos 2000. O Linux, por exemplo, é um sistema operacional de software livre, que pode ser distribuído gratuitamente. Embora a ideia do Open Hardware seja similar à do software livre, são coisas diferentes. O hardware é a parte física da máquina, as peças que a compõem, como as placas operacionais e de memória. O software, diferentemente, engloba os programas que fazem com que o computador, como aplicativos e sistemas operacionais.

#### 3. Computação física

A computação física (ou *physical computing*) refere-se ao uso de hardware e software para criar sistemas interativos que podem sentir e responder ao mundo físico. Esses sistemas utilizam sensores, atuadores, microcontroladores (como o Arduino, Esp32 ou Raspberry Pi) e outras tecnologias para interagir com o ambiente.

##### 3.1. Sensores e Atuadores

Sensores captam informações do ambiente, como temperatura, luz, som ou movimento. Atuadores transformam essas informações em ações físicas, como acender uma luz, mover um motor ou gerar som.

Exemplo: Uma lâmpada que acende automaticamente quando alguém entra na sala. Neste caso, um sensor de movimento detecta a presença da pessoa, e um atuador (no caso, a lâmpada) responde acendendo.

### 3.2. Microcontroladores

Microcontroladores são pequenos computadores integrados usados para controlar dispositivos em tempo real. Plataformas como Arduino, Esp32 e Raspberry Pi são comuns na computação física.

Exemplo: Em um projeto de irrigação automática, o microcontrolador pode monitorar os níveis de umidade do solo com sensores e ativar uma bomba de água quando o solo estiver seco.

### 3.3. Interatividade

A computação física é frequentemente utilizada para criar dispositivos interativos, onde a interação humana desempenha um papel central. Isso inclui desde brinquedos interativos até instalações de arte digital.

Exemplo: Um tambor digital que, ao ser tocado, emite sons através de sensores de pressão conectados a um microcontrolador.

## 4. Alguns Microcontroladores usados em computação física e Internet das Coisas

A ideia é antes de aprofundarmos no Arduino, vermos algumas definições de microcontroladores que existem e são bem utilizados, vale lembrar, que existem muito mais do que estamos abordando aqui, vamos lá.

### 4.1. Arduino Nano

O Arduino Nano é uma versão pequena e portátil da plataforma Arduino. Ele é ótimo para projetos compactos e funciona bem em protótipos.

#### 4.1.1. Características:

- Microcontrolador: ATmega328.
- Tamanho: 43 x 18 mm.
- Portas: 22 pinos de I/O (6 analógicos e 14 digitais).
- Conectividade: Sem Wi-Fi ou Bluetooth.
- Comunicação: Porta USB Mini-B.
- Tensão: 5V.

**Aplicação:** Projetos de automação simples como controle de LEDs ou sensores de temperatura.

Imagem – Arduino Nano:



### 4.2. Arduino Mega 2560

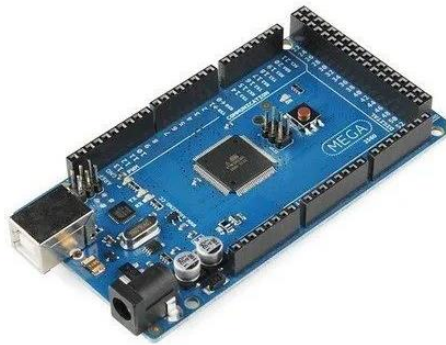
O Arduino Mega é uma versão maior do Arduino, com mais portas e memória, ideal para projetos mais complexos.

#### 4.2.1. Características:

- **Microcontrolador:** ATmega2560.
- **Portas:** 54 pinos digitais, 16 entradas analógicas.
- **Memória:** 256 KB de Flash.
- **Conectividade:** Sem Wi-Fi ou Bluetooth.
- **Comunicação:** USB padrão.
- **Tensão:** 5V.

**Aplicação:** Impressoras 3D e robótica complexa.

Imagem Exemplo – Arduino Mega:



#### 4.3. ESP32

O ESP32 foi desenvolvido pela Espressif Systems e se destaca por possuir Wi-Fi e Bluetooth integrados, sendo ideal para automação residencial, sistemas de monitoramento, e dispositivos IoT em geral.

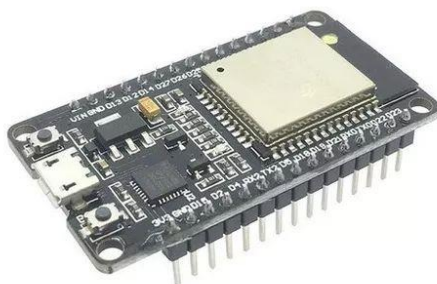
##### 4.3.1. Características Técnicas:

- **Processador:** Dual-core Xtensa LX6 com até 240 MHz.
- **Memória:** 520 KB SRAM e 4 MB Flash (ou mais, dependendo do módulo).
- **Conectividade:** Wi-Fi 802.11 b/g/n e Bluetooth 4.2 (BLE).
- **Portas:** GPIOs configuráveis para PWM, I2C, SPI, UART e ADC.
- **Tensão:** 3.3V.
- **Outras Funcionalidades:**
  - Modos de baixo consumo de energia.
  - Criptografia para comunicação segura.
  - Sensores embutidos, como sensores de temperatura e toque em alguns modelos.

##### 4.3.2. Aplicações Típicas do ESP32:

- **Automação Residencial:** Controle de luzes, trancas e dispositivos elétricos por meio de aplicativos móveis.
- **Sistemas de Monitoramento:** Monitoramento de temperatura e umidade em tempo real.
- **IoT Industrial:** Integração de sensores e atuadores em processos produtivos.
- **Wearables:** Dispositivos vestíveis que monitoram atividades físicas ou biometria.

Imagem Exemplo – ESP32:



#### 4.4. ESP32-CAM

O **ESP32-CAM** é um microcontrolador poderoso com Wi-Fi, Bluetooth e câmera integrada, muito usado para projetos de visão computacional.

##### 4.4.1. Características:

- **Processador:** Dual-core Xtensa LX6.
- **Conectividade:** Wi-Fi e Bluetooth.
- **Câmera:** OV2640 (2 MP).
- **Tensão:** 3.3V.

**Aplicação:** Sistemas de monitoramento remoto e câmeras de segurança.

Imagem Exemplo – ESP32-CAM:



#### 4.5. ESP8266

O **ESP8266** é um microcontrolador econômico com Wi-Fi integrado, ideal para automação residencial e projetos de IoT.

**Características:**

- **Processador:** 32 bits a 80 MHz.
- **Conectividade:** Wi-Fi.
- **Memória:** 4 MB Flash.
- **Tensão:** 3.3V.

**Aplicação:** Controle de lâmpadas e dispositivos remotos via internet.

Imagem Exemplo – ESP8266:



#### 4.6. Raspberry Pi

O **Raspberry Pi** é um microcomputador completo, capaz de executar sistemas operacionais e com diversas interfaces de entrada e saída. Ele é ideal para projetos mais robustos, como servidores e media centers.

**4.6.1. Características:**

- **Processador:** Broadcom ARM Cortex.
- **Memória:** 1 a 8 GB de RAM (dependendo do modelo).
- **Conectividade:** Wi-Fi, Bluetooth, portas USB e HDMI.
- **Sistemas Operacionais:** Linux (Raspbian).

**Aplicação:** Servidores web, sistemas de automação e robótica avançada.

Imagem Exemplo – *Raspberry Pi*:





## 5. Introdução ao Arduino

O Arduino faz parte do conceito de hardware e software livre e está aberto para uso e contribuição de toda sociedade. O conceito Arduino surgiu na Itália, em 2005, com o objetivo de criar um dispositivo que fosse utilizado em projetos/protótipos construídos de uma forma menos dispendiosa do que outros sistemas disponíveis no mercado.

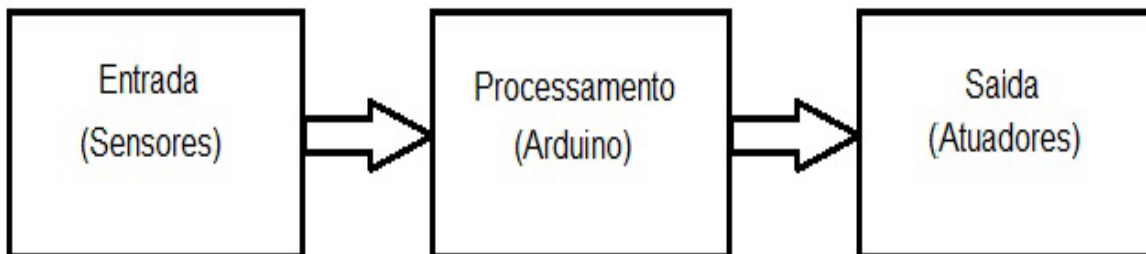
Ele pode ser usado para desenvolver artefatos interativos *stand-alone* ou conectados ao computador, utilizando diversos aplicativos, tais como: *Adobe Flash*, *Processing*, *Max/MSP*, *Pure Data* ou *SuperCollider*.

O Arduino foi projetado com a finalidade de ser de fácil entendimento, de fácil programação e de fácil aplicação, além de ser multiplataforma, podendo ser configurado em ambientes Linux, Mac OS e Windows. Além disso, um grande diferencial deste dispositivo é ser mantido por uma comunidade que trabalha na filosofia *open-source*, desenvolvendo e divulgando gratuitamente seus projetos.

O equipamento é uma plataforma de computação física: são sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que percebam a realidade e respondem com ações físicas. Ele é baseado em uma placa microcontrolada, com acessos de Entrada/Saída (I/O), sobre a qual foram desenvolvidas bibliotecas com funções que simplificam a sua programação, por meio de uma sintaxe similar a das linguagens C e C++.

O Arduino utiliza o microcontrolador Atmega. Um microcontrolador (também denominado MCU) é um computador em um chip, que contém um microprocessador, memória e periféricos de entrada/saída. Ele pode ser embarcado no interior de algum outro dispositivo, que, neste caso, é o Arduino, para que possa controlar suas funções ou ações.

Em resumo, o Arduino é um kit de desenvolvimento, que pode ser visto como uma unidade de processamento capaz de mensurar variáveis do ambiente externo, transformadas em um sinal elétrico correspondente, através de sensores ligados aos seus terminais de entrada. De posse da informação, ele pode processá-la computacionalmente. Por fim, ele pode ainda atuar no controle ou no acionamento de algum outro elemento eletroeletrônico conectado ao terminal de saída. A Figura abaixo apresenta um diagrama de blocos de uma cadeia de processamento utilizando o Arduino.



Uma vez que o Arduino é baseado em um microcontrolador e, portanto, é programável, torna-se possível criar diversas aplicações diferentes com uma certa facilidade. Além disso, o próprio equipamento pode ser reutilizado, através de uma nova programação. Por sua vez, a sua programação é simplificada pela existência de diversas funções que controlam o dispositivo, com uma sintaxe similar a de linguagens de programação comumente utilizadas (C e C++).

Inicialmente vou mostrar como instalar o mesmo no computador para que se possa fazer a interação física do Arduino, e mostrar como funciona na IDE Arduino do computador, após isso, irei mostrar no simulador TinkerCad.

## 6. Anatomia de uma placa Arduino

### 6.1. Portas Digitais

O Arduino Uno oferece 14 portas digitais que podem ser utilizadas tanto para entrada (input) como para saída (output) e que podem ser utilizadas para comandar 14 dispositivos externos. Estas portas vão de 0 a 13, observe com atenção a figura:



### 6.2. Portas PWM

Estas portas simulam uma porta analógica e são em número de seis, note que elas além do número da porta têm também uma pequena onda senoidal como mostrado na figura. As portas digitais que também podem ser usadas como portas PWM são as de número: 11, 10, 9, 6, 5 e 3. Mas tenha em mente que elas também são portas digitais.



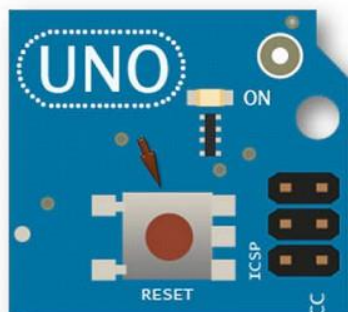
### 6.3. Portas RX e TX

Essas duas portas embora possam ser utilizadas como Portas digitais também são utilizadas pelo Arduino para comunicação serial tanto para entrada como para saída de dados. É conveniente evitar uso destas portas como portas digitais, mas não proibido.



### 6.4. Reset

Este botão, mostrado na figura abaixo, tem como única função reinicializar o Arduino Uno, mas, note que ele muda um pouco seu posicionamento em outras placas.



### 6.5. Microcontrolador

É onde tudo acontece, é o cérebro desta do Arduino, onde fica gravado o código desenvolvido e que será executado, mas note que quando se grava um código o anterior é descartado, sempre fica apenas o último código gravado.

Este processador permite que o Arduino funcione de forma autônoma, ou seja, uma vez transferido o código para ele não existe mais a necessidade de uma conexão com o computador.

A figura 1 mostra o tipo de processador mais comumente encontrado, mas existem outros como mostrado na figura 2. A grande vantagem do processador da figura 1 é a facilidade com que pode ser removido e transferido para outra placa.

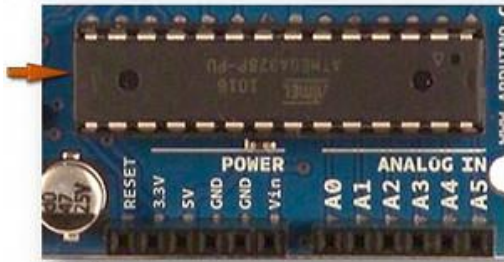


Figura 1



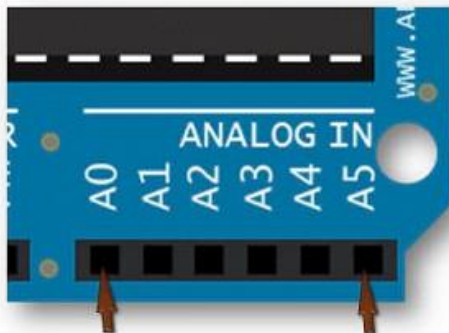
Figura 2

#### 6.6. Portas Analógicas

Essas portas são unicamente para entrada de dados e comumente usadas para comunicação com sensores que podem ser utilizados para determinar:

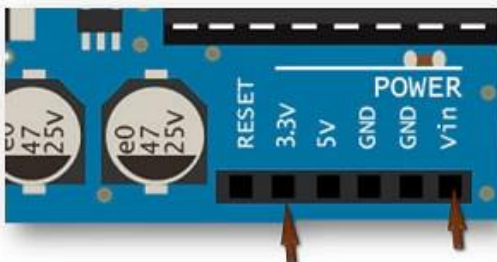
- Temperatura;
- Quantidade de luz;
- Umidade;
- Dentre outras ações.

As portas analógicas são em número de 6 e vão de A0 até A5.



#### 6.7 Pinos de Energia

Estes pinos fornecem energia para dispositivos externos como mostrado na figura ao lado e explicados abaixo: Observe com atenção a figura, ela mostra estes pinos.



- 3.3V: este pino fornece a 3.3 volts a dispositivos externos e está marcado na placa;
- 5V: fornece 5 volts a dispositivos externos e também está indicado na placa.
- GND: fornece potencial de terra a dispositivos externos, o seja 0 volt e são em número de dois e como pode observar também se encontram bem identificado na placa.

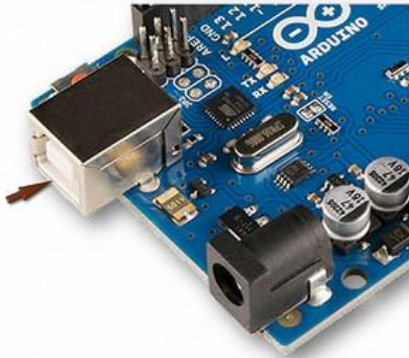


- Vin: este pino fornece ao dispositivo externo a mesma tensão que está sendo recebida pelo Pino de alimentação externa. Note que é o último pino mostrado na figura.

Deve ser observado que estes pinos fornecem uma corrente muito baixa, portanto devem ser usados para pequenas cargas e nunca para ligar um motor, por pequeno que ele possa ser dentre outros dispositivos de alto consumo. De maneira geral aguentam alimentar apenas um LED, a base de um transistor ou coisa do gênero.

#### 6.8. Porta USB

Esta é a porta usada para estabelecer uma conexão entre a placa de Arduino e o PC. É ela que permite o envio de códigos para o processador, permite conexão com a serial e também é usada para a alimentação da placa. Observe a figura:



#### 6.9. Pino para Alimentação externa

Este é o pino para a alimentação externa da placa, ou seja, quando não estiver sendo usado a porta USB para conexão com o computador, e é usando-o que vamos alimentar a placa. Ele é que permite a autonomia desta placa. Podemos alimentar esta placa com tensão entre 6 e 12 volts sem problemas.



### 7. Arduino em Windows

#### 7.1 Placa Arduino e um cabo USB AB



## 7.2. - Download do software do Arduino

Faça download da última versão do software do Arduino(<http://multilogica-shop.com/Download>). Ao terminar, descompacte o arquivo e mantenha a estrutura de pastas e sub-pastas. Se quiser guarde esta pasta no drive C: do seu computador. Dentro desta pasta existe um arquivo chamado arduino.exe que é o ponto de entrada do programa do Arduino, a IDE (Integrated Development Environment).

## 7.3. Conectando o Arduino

O Arduino Uno isolado usa a energia do computador através da conexão USB, não sendo necessária energia externa. Conecte a placa Arduino ao computador usando o cabo USB AB. O LED verde de energia (PWR) deve acender.

## 7.4. Instalando os drivers

Drivers para Arduino Uno ou Arduino Mega 2560 com Windows 7, Vista ou XP:

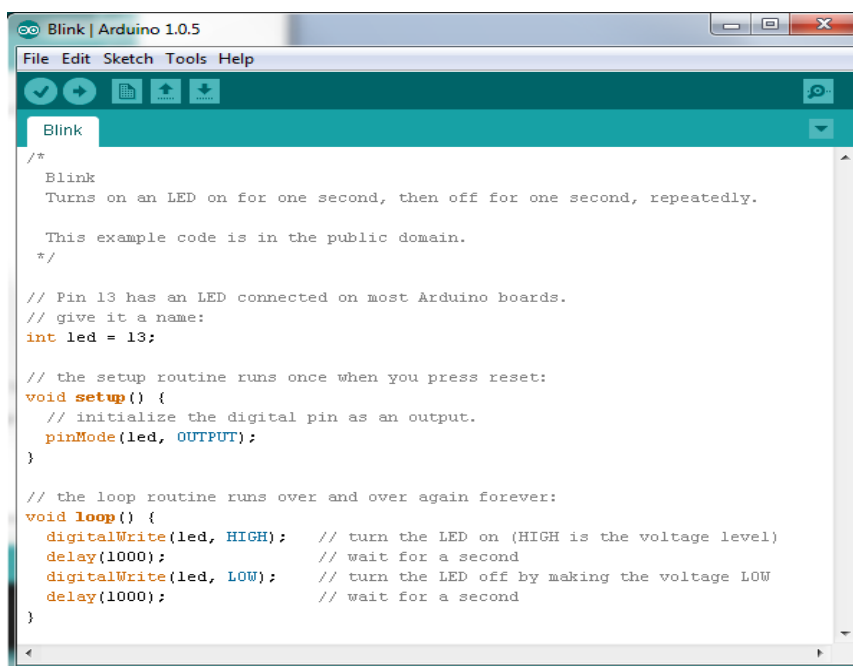
- Conecte a placa ao computador e aguarde o Windows iniciar o processo de instalação do driver. Depois de alguns momentos o processo vai falhar. Clique em concluir e dispense a ajuda do assistente.
- Clique no Menu Principal e abra o Painel de Controle.
- Dentro do Painel de Controle, navegue até Sistema e Segurança. Na sequência clique em Sistema, selecione Hardware e depois clique em Gerenciador de Dispositivos.
- Procure por Portas (COM & LPT), onde você deve ver uma opção Arduino UNO (COMxx).
- Clique com o botão da direita em Arduino UNO (COMxx) e escolha a opção Atualizar Driver.
- Depois escolha a opção Instalar de uma lista ou local específico (Avançado), e clique em avançar.
- Finalmente navegue e escolha o driver arduino.inf localizado na pasta Drivers do software do Arduino que você baixou.
- O Windows vai finalizar a instalação do driver a partir deste ponto.

## 7.5. Abrindo o programa Arduino

Clique duas vezes na aplicação do Arduino, o arquivo arduino.exe. Caso o programa carregue com o idioma que não é da sua preferência você pode alterar na sessão de preferências do programa.

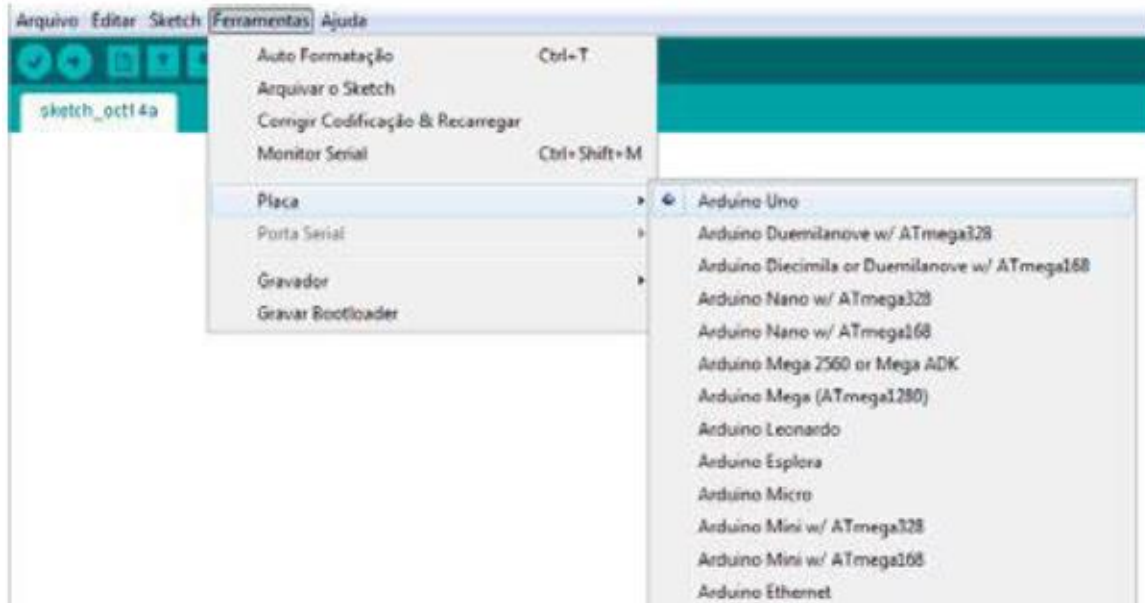
## 7.6. Exemplo Piscar

Abra o exemplo Piscar (blink): Arquivo > Exemplos > 01.Basics > Blink



### 7.7. Selecione sua placa

Você deve selecionar qual a sua placa Arduino: Ferramentas > Placa > Arduino Uno.



### 7.8. Selecione a porta

Selecione agora a porta serial que conectará o Arduino: Ferramentas > Porta Serial.

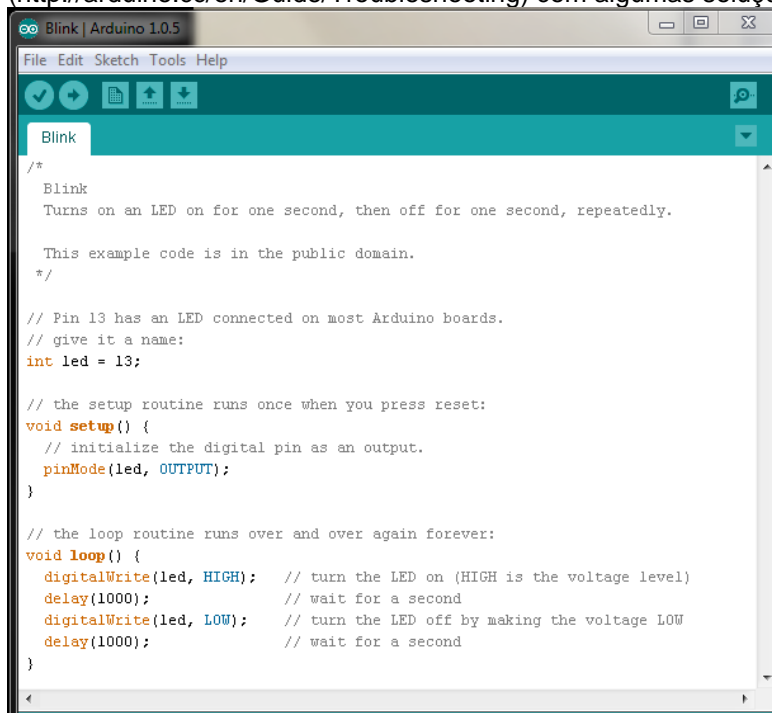
Você deve selecionar a mesma porta que utilizou para configurar o sistema no passo 4.

### 7.9. Carregue o programa

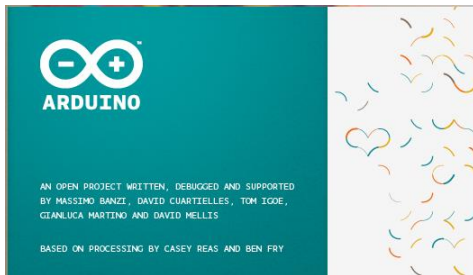
Agora simplesmente clique no botão Carregar da janela do programa. Espere alguns segundos. Você deve ver os LEDs RX e TX da placa piscarem. Se o processo foi executado normalmente você verá uma mensagem de “Transferência concluída”.

Depois de alguns segundos você verá o LED do pin 13 piscar em laranja. Neste caso, parabéns! Seu Arduino está pronto e instalado.

Se você tiver problemas na instalação pode acessar a página oficial do Arduino (<http://arduino.cc/en/Guide/Troubleshooting>) com algumas soluções.



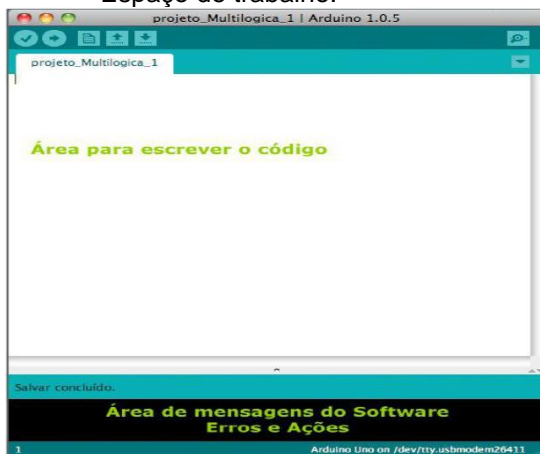
## 8. Software Arduino



Para executar o programa entramos na pasta do Arduino guardada no computador e procuramos o ícone. Clique duas vezes para abrir o programa.

O programa do Arduino também é conhecido como IDE Arduino (*Integrated Development Environment*) pois além do entorno de programação consiste também em um editor de código, um compilador e um depurador.

Espaço de trabalho:



### 8.1. Sketches

Softwares escritos usando Arduino são chamados de *Sketches*. Estes Sketches são escritos no editor de texto da IDE do Arduino e são salvos com a extensão de arquivo .ino. Este editor tem características de cortar/colar e para buscar/substituir texto. A área de mensagem dá feedback ao salvar e exportar arquivos e também exibe informações de erros ao compilar Sketches. O canto direito inferior da janela exibe a placa atual e a porta serial. Os botões da barra de ferramentas permitem que você verifique, carregue, crie, abra e salve Sketches ou abra o monitor serial.

**Nota:** Nas versões do IDE antes de 1.0 os Sketches são salvos com a extensão .pde. É possível abrir esses arquivos com a versão 1.0, mas você será solicitado a salvar o Sketch com a extensão .ino.



#### Verificar

Verifica se seu código tem erros.



#### Carregar

Compila seu código e carrega para a placa Arduino.



#### Novo

Cria um novo Sketch.



### Abrir

Apresenta um menu de todos os sketches já existentes.



### Salvar

Salva seu Sketch.



### Monitor Serial

Abre o monitor serial.

## 8.2. Biblioteca Arduino

O ambiente Arduino pode ser estendido através da utilização de bibliotecas, assim como a maioria das plataformas de programação. Bibliotecas fornecem funcionalidades extras para uso em sketches. Por exemplo, para trabalhar com hardware ou manipulação de dados.

Algumas bibliotecas já vêm instaladas com a IDE Arduino, mas você também pode fazer download ou criar a sua própria.

Para usar uma biblioteca em um sketch, selecione em sua IDE Arduino:

Sketch> Importar Biblioteca.

Dentro da programação você inclui as funcionalidades de uma biblioteca já existente a partir do comando:

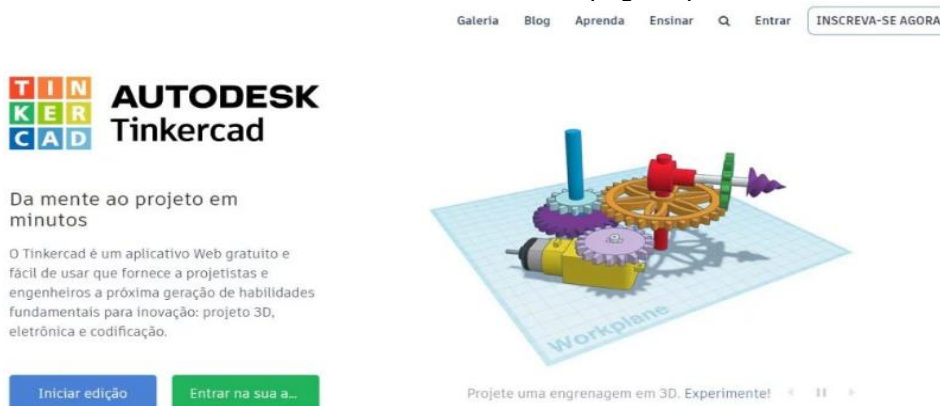
```
#include <LiquidCrystal.h>
```

## 9. Simulador Tinkercad

O Tinkercad é uma ferramenta online de design de modelos 3D em CAD, onde podemos criar projetos e enviar para uma impressora 3D por exemplo, e de simulação de circuitos elétricos analógicos e digitais, desenvolvida pela Autodesk. Por ser gratuita e de fácil manipulação, encontramos nela uma oportunidade de ensino de Programação Embarcada, que é o principal objetivo de nossa aula neste semestre.

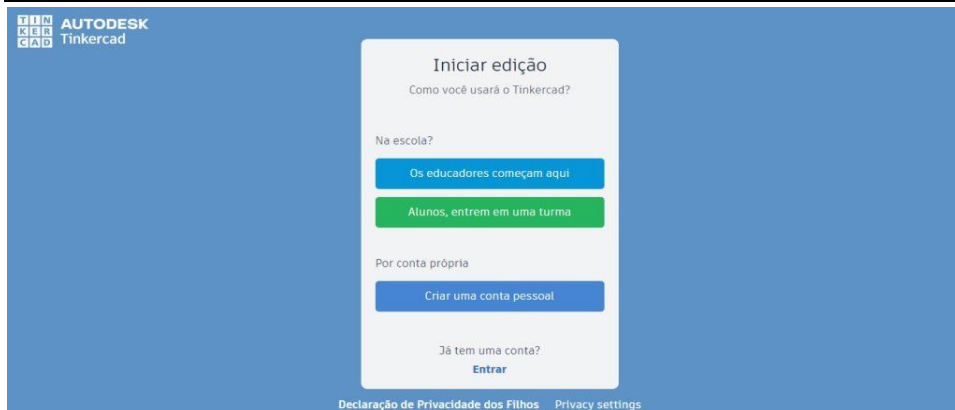
Através do link abaixo, você será redirecionado à página do Tinkercad: <https://www.tinkercad.com/>

Acessando-o, você deve se encontrar na página que está ilustrada na imagem abaixo:

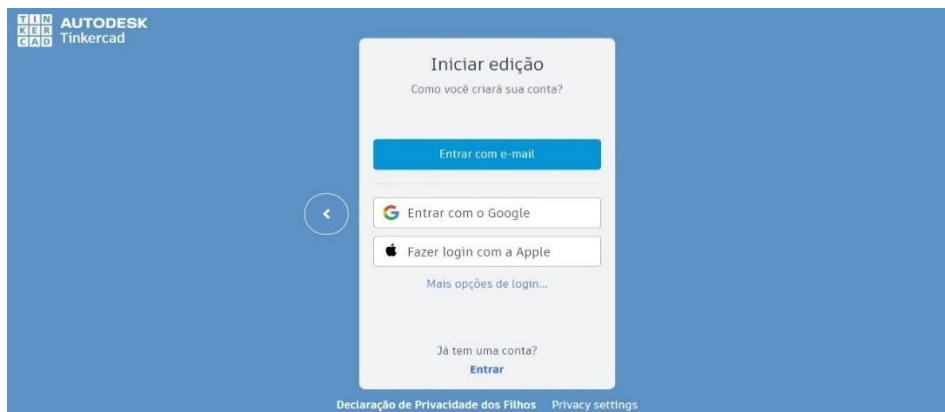


Perceba que, na imagem acima, temos dois botões no canto inferior esquerdo. “Iniciar a edição” lhe dará algumas opções para começar no Tinkercad, como educador, aluno ou por conta própria. Já o botão verde, “Entrar na sua aula”, irá te redirecionar para uma página que irá pedir um código de acesso para a sala de aula de um professor. Neste post, seguiremos, primeiro, pela opção “inscreva-se agora”.

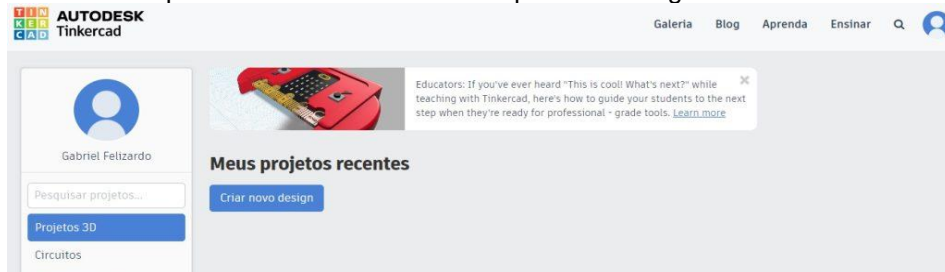




Conforme vimos acima, iremos escolher a opção “Criar uma conta pessoal”.

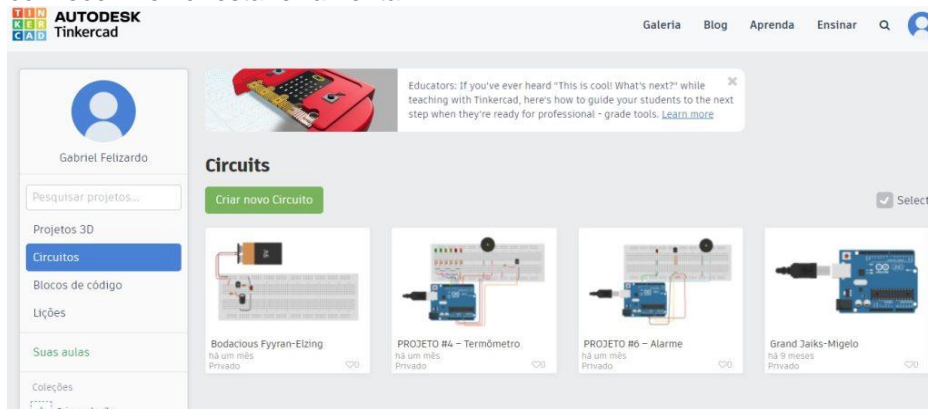


Agora, iremos criar a nossa conta pela opção “Entrar com o Google”. Depois deste passo, você só precisa preencher seus dados. Por fim, com todos estes passos concluídos, iremos ser direcionados para nosso dashboard. Você pode vê-lo logo abaixo:

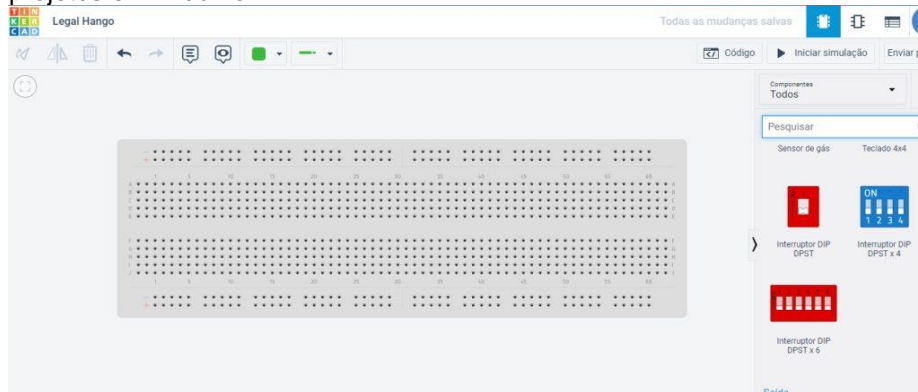


## Circuitos

Conforme a imagem abaixo, seguiremos pela aba “Circuitos” para começarmos a conhecer melhor esta ferramenta.



Após acessar “Criar novo circuito”, estaremos no ambiente de prototipação para nossos projetos em Arduino.



## 10. Programando o Arduino

Arduino se programa em uma linguagem de alto nível semelhante a C/C++ e geralmente tem os seguintes componentes para elaborar o algoritmo, isso tanto no simulador (TinkerCad) como na IDE do Arduino:

- Estruturas (veremos a seguir)
- Variáveis (veremos a seguir)
- Operadores booleanos, de comparação e aritméticos (no decorrer do curso)
- Estrutura de controle (no decorrer do curso)
- Funções digitais e analógicas (no decorrer do curso)

Veja a referência estendida (<http://arduino.cc/en/Reference/HomePage?from=Reference.Extended>) para características mais avançadas da linguagem Arduino e a página das bibliotecas (<http://arduino.cc/en/Reference/Libraries>) para interação com tipos específicos de hardware, no site oficial do Arduino.

### 10.1. Estruturas

São duas funções principais que deve ter todo programa em Arduino.

A função `setup()` é chamada quando um programa começa a rodar. Use esta função para inicializar as suas variáveis, os modos dos pinos, declarar o uso de livrerias, etc. Esta função será executada apenas uma vez após a placa Arduino ser ligada ou ressetada.

```
setup(){
}
```

Após criar uma função `setup()` que declara os valores iniciais, a função `loop()` faz exatamente o que seu nome sugere, entra em looping (executa sempre o mesmo bloco de código), permitindo ao seu programa fazer mudanças e responder. Use esta função para controlar ativamente a placa Arduino.

```
loop(){
}
```

### 10.2. Variáveis

Variáveis são expressões que você pode usar em programas para armazenar valores como a leitura de um sensor em um pino analógico. Aqui destacamos algumas:

#### - Variáveis Booleanas

Variáveis booleanas, assim chamadas em homenagem a George Boole, podem ter apenas dois valores: verdadeiro (true) e falso (false).

**Exemplo:** `boolean running = false;`

#### - Int

Inteiro é o principal tipo de dado para armazenamento numérico capaz de guardar números de 2 bytes. Isto abrange a faixa de -32.768 a 32.767 (valor mínimo de  $-2^{15}$  e valor máximo de  $(2^{15}) - 1$ ).

**Exemplo:** `int ledPin = 13;`

#### - Char

Um tipo de dado que ocupa 1 byte de memória e armazena o valor de um caractere ASCII. Caracteres literais são escritos entre aspas.

**Exemplo:** `char myChar = 'A';`

### 11. Vamos praticar?

A ideia agora é iniciarmos o manuseio do Arduino com a Linguagem de Programação, para isso vamos usar exemplos de aplicações envolvendo o hardware e o software onde teremos a entrada, o processamento e a saída de dados, lembrando que iremos fazer primeiramente no TinkerCad e posteriormente fisicamente utilizando a IDE do Arduino.

#### 11.1. Hello World

Este exemplo mostra a experiência mais simples que você pode fazer com um Arduino para verificar uma saída física: **piscar um LED**.

Quando você está aprendendo a programar, na maioria das linguagens de programação, o primeiro código que você escreve diz “*Hello World*” na tela do computador. Como a placa Arduino não tem uma tela substituiremos esta função fazendo piscar um LED.

##### 11.1.1. O que vou aprender?

- Ativar uma saída digital
- Acender um LED em ON/OFF
- Temporizar um sinal de saída
- Sintaxe de um programa Arduino

##### 11.1.2. Conhecimentos prévios

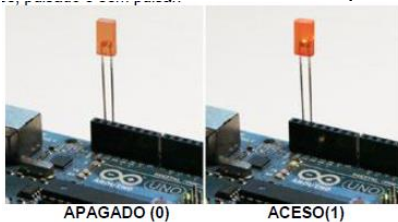
###### 11.1.2.1. Sinal digital

As entradas e saídas de um sistema eletrônico serão consideradas como sinais variáveis. Em eletrônica se trabalha com variáveis que são tomadas na forma de tensão ou corrente, que podem simplesmente ser chamados de sinais.

Os sinais podem ser de dois tipos: digital ou analógico.

Em nosso caso como o que nos interessa no momento é o sinal digital, ele se caracteriza por ter dois estados diferentes e portanto também podem ser chamadas de binárias (em lógica seria valores Verdadeiro (V) e Falso (F), ou poderiam ser 1 ou 0 respectivamente).

Um exemplo de um sinal digital é o interruptor da campainha da sua casa, porque ele tem somente dois estados, pulsado e sem pulsar.



###### 11.1.2.2. Função pinMode()

Configura o pino especificado para que se comporte ou como uma entrada (input) ou uma saída (output).

Sintaxe:

**pinMode(pin, mode)**

`pinMode(9, OUTPUT);` // determina o pino digital 9 como uma saída.

###### 11.1.2.3. Função digitalWrite()

Escreve um valor HIGH (ligar) ou um LOW (desligar) em um pino digital.

Sintaxe:

**digitalWrite(pin, valor)**

`digitalWrite(9,HIGH)` // o LED na porta 9 acenderá

#### 11.1.2.4. Função delay()

É um temporizador, onde o valor passado(em milissegundos) será o tempo em que o Arduino não irá executar atividades até que este tempo passe.

Sintaxe:

**delay(valor em milissegundos)**

delay(1000); // neste caso o Arduino ficará 1 segundo sem executar atividades.

#### 11.1.2.5. Polaridade de um LED

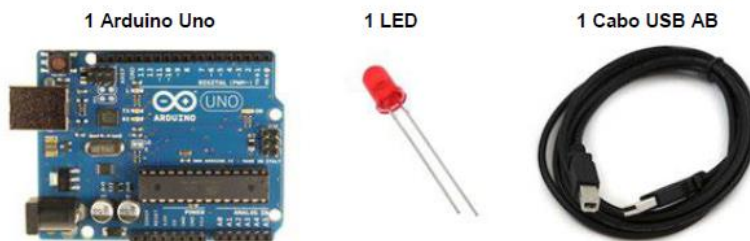
O LED (*Light Emitting Diode*) é um diodo que emite luz quando energizado. Os LEDs apresentam muitas vantagens sobre as fontes de luz incandescentes como um consumo menor de energia, maior tempo de vida, menor tamanho, grande durabilidade e confiabilidade. O LED tem uma polaridade, uma ordem de conexão. Ao conectá-lo invertido não funcionará corretamente. Revise os desenhos para verificar a correspondência do negativo e do positivo.

São especialmente utilizados em produtos de microeletrônica como sinalizador de avisos. Também é muito utilizado em painéis, cortinas e pistas de led. Podem ser encontrados em tamanho maior, como em alguns modelos de semáforos ou displays.

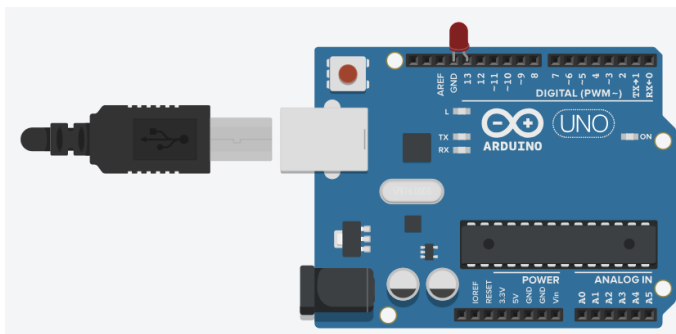
#### 11.1.2.6. Conexão da placa Arduino com o computador

Item 6.3 desta nota de aula.

#### 11.1.2.7. Material necessário



#### 11.1.2.8. Diagrama



#### 11.1.2.9. Código fonte

No programa a seguir, o primeiro comando é o de inicializar o pino 13 como saída através da linha `pinMode(13, OUTPUT);`

No loop principal do código, você liga o LED com esta linha de comando:

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

Este comando direciona 5 volts ao pino 13 e o acende. Você desliga o LED com o seguinte comando:

**`digitalWrite(13, LOW);`**

Este comando retira os 5 volts do pino 13, voltando para 0 e desligando o LED. Entre desligar e ligar você precisa de tempo suficiente para que uma pessoa veja a diferença, então o comando `delay()` informa o Arduino não fazer nada durante 1000 milissegundos, ou um segundo. Quando você usa o comando `delay()`, nada mais acontece neste período de tempo.

```

1  /*
2  Piscar/Led -> "Hello World - Arduino"
3  Acende o Led por um segundo, e depois apaga pelo mesmo tempo,
4  de forma repetida.
5  */
6
7  //Variável para o pino 13
8  int led = 13;
9
10 //Executa somente 1 vez quando o Arduino liga
11 void setup(){
12     //Indica que o pino é de Saída
13     pinMode(led, OUTPUT);
14 }
15
16 void loop(){
17     digitalWrite(led, HIGH); //Acende o Led
18     delay(1000); //Pausa de 1 segundo
19     digitalWrite(led, LOW); //Apaga o Led
20     delay(1000); //Pausa de 1 segundo
21 }
22

```

Agora façam primeiramente no TinkerCad, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

## 11.2. Semáforo

Agora nós iremos criar um circuito para simular um semáforo de trânsito. O semáforo será constituído por três LED's: um vermelho, um amarelo e um verde.

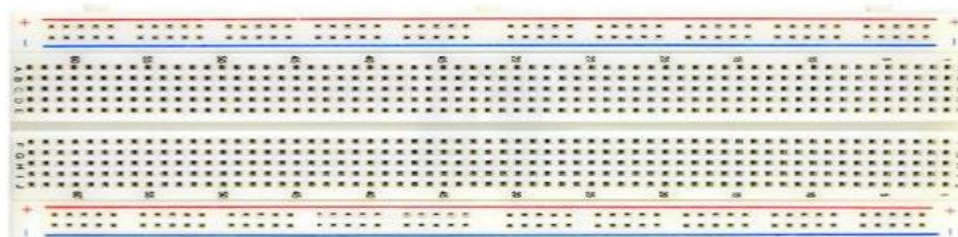
### 11.2.1. O que vou aprender?

- O que é uma protoboard;
- Cabear um circuito;
- Ler uma entrada digital e escrever uma saída digital.

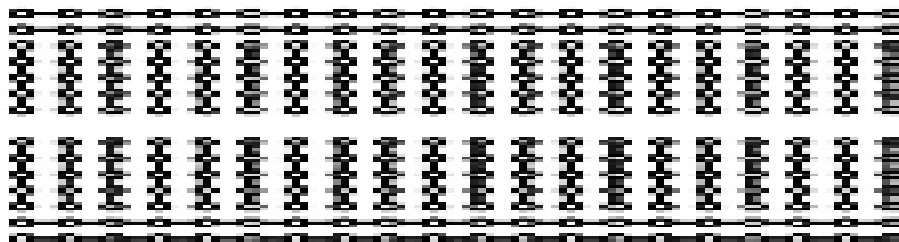
### 11.2.2. O que é uma Protoboard

É uma placa reutilizável usada para construir protótipos de circuitos eletrônicos sem solda.

Uma protoboard é feita por blocos de plástico perfurados e várias lâminas finas de uma liga metálica de cobre, estanho e fósforo.



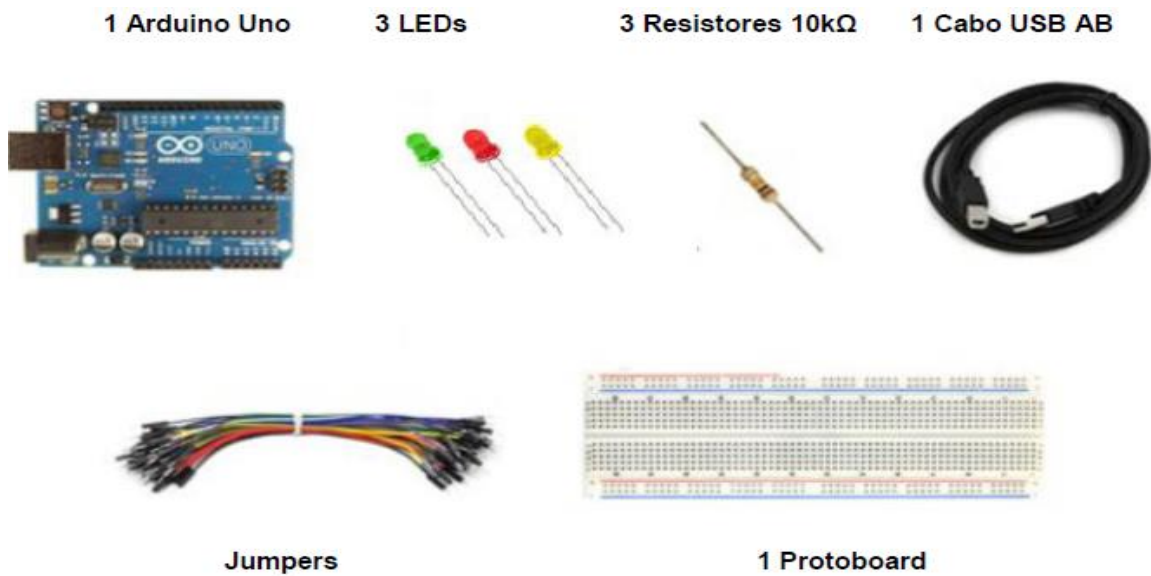
Protoboard



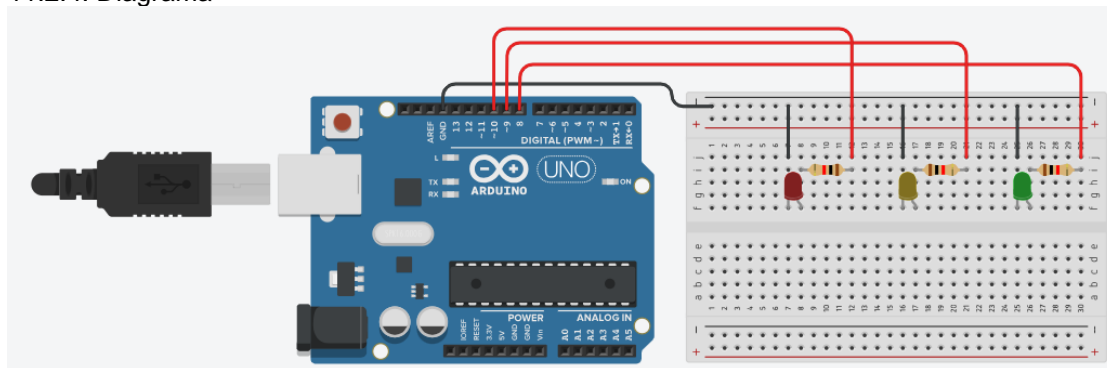
Conexões abertas



### 11.2.3. Material necessário



### 11.2.4. Diagrama



### 11.2.5. Código Fonte

```

1  /*
2  Semáforo simples.
3  */
4  int ledDelay = 5000; //Variável de tempo das paradas
5
6  int vermelho = 10; //Variável de Luz Vermelha
7  int amarelo = 9; //Variável de Luz Amarela
8  int verde = 8; //Variável de Luz Verde
9
10 void setup(){
11     //Inicializando os pinos
12     pinMode(vermelho, OUTPUT);
13     pinMode(amarelo, OUTPUT);
14     pinMode(verde, OUTPUT);
15 }
16
17 void loop(){
18     digitalWrite(vermelho, HIGH); //Acende o vermelho
19     delay(ledDelay); //Aplicando a pausa de tempo
20     digitalWrite(vermelho, LOW); //Apagando o Vermelho
21     delay(500); //Pausa para transição
22     digitalWrite(amarelo, HIGH); //Acende o Amarelo
23     delay(2000); //Pausa de atenção
24     digitalWrite(amarelo, LOW); //Apagando o Amarelo
25     delay(500); //Pausa para transição
26     digitalWrite(verde, HIGH); //Acende o Verde
27     delay(ledDelay); //Aplicando o tempo que ficará verde
28     digitalWrite(verde, LOW); //Desliga o Verde
29     delay(500); //Pausa de transição
30 }

```

Agora façam primeiramente no TinkerCad, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

### 11.3. Botão

O botão é um componente que conecta dois pontos do circuito quando está pressionado. Neste exemplo quando o botão está pressionado o LED se acende.

#### 11.3.1 O que vou aprender?

- Cabear um circuito;
- Condicional if/else;
- Comunicação serial;
- Função `digitalRead()`, ler uma entrada digital.

#### 11.3.2. Conhecimentos prévios

##### 11.3.2.1. Função `digitalRead()`

Lê o valor de um pino digital especificado, HIGH ou LOW.

Sintaxe:

```
digitalRead(pin)
buttonState = digitalRead(9); // Leitura do estado de um botão no pino 9.
```

##### 11.3.2.2. Condicional If else (Se e Senão)

###### 11.3.2.2.1. Condicional If else

**If**, que é usado juntamente com um operador de comparação, verifica quando uma condição é satisfeita, como por exemplo um input acima de um determinado valor. O formato para uma verificação **if** é:

```
if (algumaVariavel == 50) {
    // faça alguma coisa se verdade
} else{
    // faça outra coisa se falso
}
```

O programa checa se algumaVariavel (colocar acentos em nomes de variáveis não é uma boa ideia) é maior que 50. Se for, o programa realiza uma ação específica. Colocado de outra maneira, se a sentença que está dentro dos parêntesis é verdadeira o código que está dentro das chaves roda; caso contrário o programa salta este bloco de código, e vai para o **else**, que é a negação do IF, realizando outra tarefa.

A sentença que está sendo verificada necessita o uso de pelo menos um dos operadores de comparação:

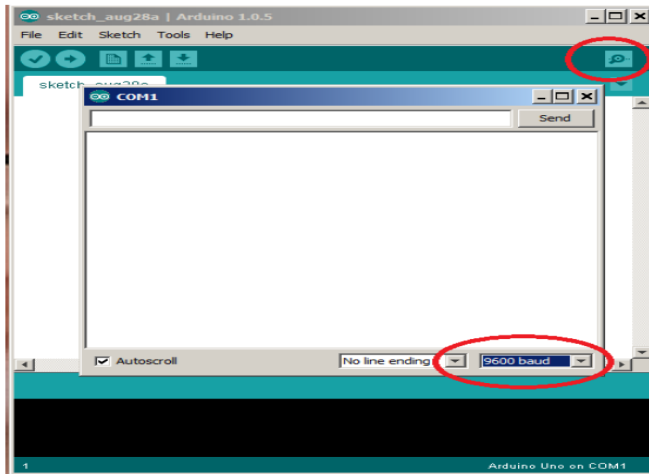
```
x == y (x é igual a y)
x != y (x é não igual a y)
x < y (x é menor que y)
x > y (x é maior que y)
x <= y (x é menor ou igual a y)
x >= y (x é maior ou igual a y)
```

#### 11.3.3. *Serial.print*

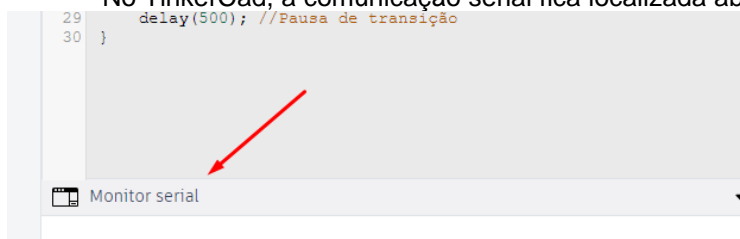
Exibe dados seriais sendo enviados da placa Arduino para o computador. Para enviar dados para a placa, digite o texto e clique no botão "enviar" ou pressione enter.

A comunicação entre a placa Arduino e seu computador pode acontecer em várias velocidades padrão pré-definidas. Para que isso ocorra é importante que seja definida a mesma velocidade tanto na *Sketch* quanto no Monitor Serial.

Na Sketch esta escolha é feita através da função `Serial.begin`. E no Monitor Serial através do menu *drop down* do canto inferior direito, isso na IDE Arduino.



No TinkerCad, a comunicação serial fica localizada abaixo do código escrito, vejam:

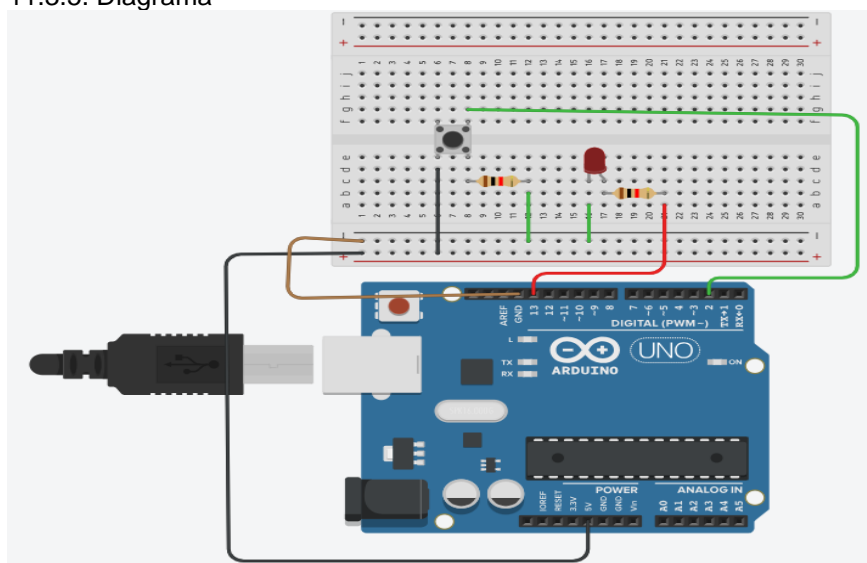


#### 11.3.4. Material necessário

1 Arduino Uno    1 PushButton    1 LED    1 Resistor 10k    Cabo USB/AB



#### 11.3.5. Diagrama



### 11.3.6. Código Fonte

```

1  /*
2     Botão para acionar LED
3  */
4  //Declarar as variáveis de controle
5  int v_botao = 2;
6  int v_ledRed = 13;
7  String v_situacao;
8
9  //Declarar a variável de estado do botão
10 int v_estado;
11
12 void setup() {
13     Serial.begin(9600);
14     pinMode(v_botao, INPUT);
15     pinMode(v_ledRed, OUTPUT);
16 }
17
18 void loop() {
19     //Leitura do estado do botão
20     v_estado = digitalRead(v_botao);
21
22     //Validação
23     if (v_estado == HIGH){
24         //Acender o LED
25         digitalWrite(v_ledRed, HIGH);
26         v_situacao = "LED ligado :";
27     }else{
28         //Apagar o LED
29         digitalWrite(v_ledRed, LOW);
30         v_situacao = "LED desligado :";
31     }
32     Serial.println(v_situacao); //Mostrando o resultado na Serial
33     delay(1000);
34 }

```

Agora façam primeiramente no TinkerCad, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

### 11.4. Som no Arduino

Agora nós iremos ver como emitir sons com o Arduino e mais algumas definições importantes tanto em programação convencional como embarcada, iremos fazer uma escala musical, utilizando os recursos explicados a seguir.

#### 11.4.1. O que vou aprender

- O que é um **Buzzer**;
- Função **tone()**;
- O que é um **Array**;
- Estrutura de repetição **for**.

#### 11.4.2. O que é um Buzzer

O *buzzer* é um transdutor ou espécie de alto-falante que já possui um oscilador interno para produzir sons (semelhantes ao de uma sirene). Os transdutores são dispositivos de alta impedância, fabricados com um material à base de uma cerâmica, denominada titanato de bário. Eles são muito sensíveis podendo ser usados como fones ou pequenos alto-falantes em sistemas de aviso ou alarmes.

#### BUZZER



#### 11.4.3. Função tone()

A função `tone()` possui 2 sintaxes: `tone(pino, frequência)` e `tone(pino, frequência, duração)`, onde pino referencia qual é o pino que irá gerar a frequência (ligado ao positivo do buzzer), a frequência é definida em hertz e a duração (opcional) é em milissegundos. Caso opte pela sintaxe

sem duração é necessário usar a função `noTone(pino)` para parar a frequência enviada pelo pino definido

Exemplo:

`tone(pino, frequência, duração)`

onde a **frequência** do tom é setada em hertz, e a **duração**, em milissegundos.

#### 11.4.4. O que é um Array?

Array é um tipo especial de variável que pode armazenar uma série de elementos de dados ao mesmo tempo. Além disso, cada valor de um *array* estará sempre relacionado à uma chave de identificação. Portanto, esta estrutura de dados é também conhecida como variável indexada, vejam a imagem abaixo:

```
int[] arr = {1, 2, 3, 4, 5};
```

index	→	0	1	2	3	4
value	→	1	2	3	4	5
		arr[0]	arr[1]	arr[2]	arr[3]	arr[4]

#### 11.4.5. Estrutura de repetição *for*

A estrutura *for* é usada para repetir um bloco de instruções entre chaves. Um contador de incremento é geralmente usado para incrementar e finalizar o loop. A instrução *for* é útil para qualquer operação repetitiva e geralmente é usada em combinação com matrizes para operar em coleções de dados / pinos, a seguir temos a sintaxe:

```
for (inicialização; condição; incremento) {
    // afirmações e código desejado para repetição;
}
```

#### Parâmetros:

- inicialização:** acontece primeiro e exatamente uma vez;
- condição:** cada vez que o loop é testado; se for verdade, o bloco de instruções e o incremento forem executados, a condição será testada novamente. Quando a condição se torna falsa, o loop termina;
- incremento:** executado sempre através do loop quando a condição for verdadeira.

#### 11.4.6. Material necessário

**1 Arduino Uno**



**1 Buzzer**



**1 Cabo USB AB**



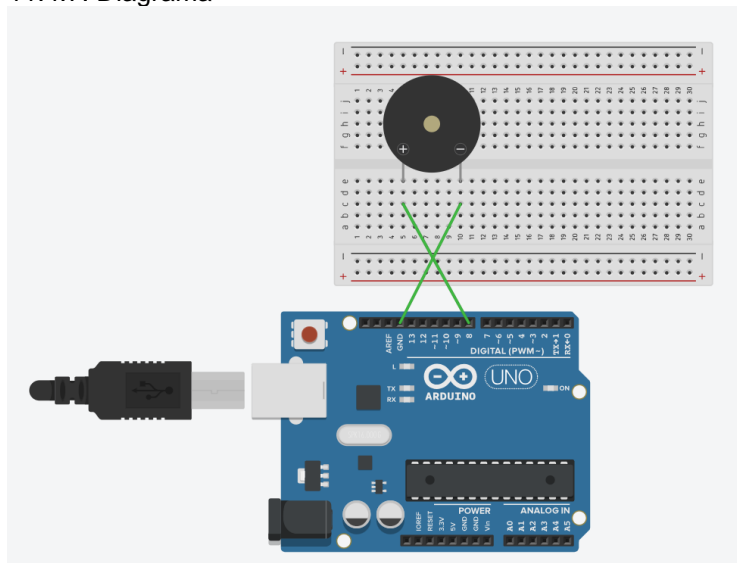
**Jumpers**



**1 Protoboard**



#### 11.4.7. Diagrama



#### 11.4.8. Código fonte

```

1  /*
2  Buzzer - Notas musicais com buzzer
3  */
4
5  //Notas Musicais
6  int DO  = 262; //Nota Do
7  int RE  = 294; //Nota Ré
8  int MI  = 330; //Nota Mi
9  int FA  = 349; //Nota Fa
10 int SOL = 392; //Nota Sol
11 int LA  = 440; //Nota La
12 int SI  = 494; //Nota Si
13 int DO_2 = 523; //Nota Dó2
14
15 int pinoBuzzer = 8; //Pino do Buzzer
16
17 //Array para armazenar as notas
18 int melodia[] = {
19   DO, RE, MI, FA, SOL, LA, SI, DO_2
20 // 0  1  2  3  4  5  6  7
21 };
22
23 int tempo[] = {
24   300, 300, 500, 500, 200, 250, 600, 120
25 // 0  1  2  3  4  5  6  7
26 };
27
28 void setup() {
29   // configura pino do buzzer como saída
30   pinMode(pinoBuzzer, OUTPUT);
31 }
32
33 void loop() {
34   for(int i=0 ; i<8; i++) //Estrutura for para percorrer o array
35   {
36     tone(pinoBuzzer, melodia[i]); //Toca de acordo com a nota
37     delay(tempo[i]); //Duração do som
38   }
39
40   for(int i=7 ; i>=0; i--) //Estrutura for para percorrer o array
41   {
42     tone(pinoBuzzer, melodia[i]); //Toca de acordo com a nota
43     delay(tempo[i]); //Duração do som
44   }
45 }

```

Agora façam primeiramente no *TinkerCad*, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

### 11.5 Medindo distância com sensor ultrassônico

Neste projeto, iremos fazer um sensor que mede a distância de um determinado objeto e veremos o resultado na Serial Monitor.

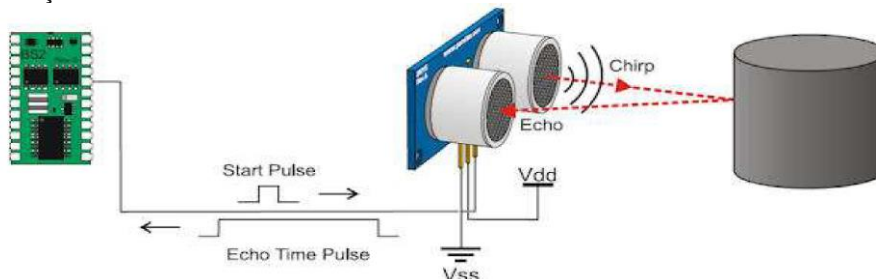
A comunicação seria no computador é vista em uma tela à parte, que pode ser acessada pelo atalho Ctrl+Shift+M (Serial monitor) na IDE do Arduino.

#### 11.5.1 O que vou aprender?

- O que é um sensor ultrassônico.

#### 11.5.2 O que é um sensor ultrassônico?

O sensor é usado para medir distâncias. A distância do sensor de ultrassom comum de medição é de cerca de 2 cm a 3-5m. O módulo sensor funciona assim:



O sensor ultrassônico é composto de um emissor e um receptor de ondas sonoras. Podemos compará-los a um alto-falante e um microfone trabalhando em conjunto. Entretanto, ambos trabalham com ondas de altíssima frequência, na faixa dos 40.000 Hz (ou 40KHz). Isto é muito, muito acima do que os nossos ouvidos são capazes de perceber. O ouvido humano consegue, normalmente, perceber ondas na entre 20 e 20.000 Hz e por isto o sinal emitido pelo sensor ultrassônico passa despercebido por nós.

O sinal emitido, ao colidir com qualquer obstáculo, é refletido de volta na direção do sensor. Durante todo o processo, o aparelho está com uma espécie de “cronômetro” de alta precisão funcionando. Assim, podemos saber quanto tempo o sinal levou desde a sua emissão até o seu retorno. Como a velocidade do som no ar é conhecida, é possível, de posse do tempo que o sinal levou para ir até o obstáculo e voltar, calcular a distância entre o sensor e o obstáculo. Para isto vamos considerar a velocidade do som no ar (340 m/s) na seguinte equação:

$$d = ( V * t ) / 2$$

Onde:

d = Distância entre o sensor e o obstáculo (é o que queremos descobrir);

V = Velocidade do som no ar (340 m/s);

t = Tempo necessário para o sinal ir do sensor até o obstáculo e voltar (é o que o nosso módulo sensor ultrassom mede);

A divisão por dois existe pois o tempo medido pelo sensor é na realidade o tempo para ir e voltar, ou seja, duas vezes a distância que queremos descobrir.

O funcionamento do sensor ultrassônico trabalha com dois pinos que são utilizados para alimentar o sensor, um deles é utilizado para disparar o sinal ultrassônico e o outro para medir o tempo que ele leva para retornar ao sensor. Existem alguns sensores ultrassônicos, que possuem apenas três pinos e utilizam um único pino para disparar o pulso e medir o tempo de resposta. Se seu sensor tiver apenas três pinos as conexões e o código do Arduino devem ser devidamente ajustados.

**VCC:** Alimentação do módulo com +5 V.

**Trig:** Gatilho para disparar o pulso ultrassônico. Para disparar coloque o pino é HIGH por pelo menos 10us.

**Echo:** Gera um pulso com a duração do tempo necessário para o eco do pulso ser recebido pelo sensor.

**Gnd:** Terra.



Vale lembrar que existem limitações para o funcionamento do sensor ultrassônico. Primeiro você tem que levar em consideração que tipo de obstáculo está querendo detectar. Se o obstáculo for muito pequeno pode ser que ele não gere um sinal de retorno suficiente para ser percebido pelo sensor. Se o obstáculo não estiver posicionado bem a frente do sensor você pode ter medidas imprecisas ou até mesmo não acusar a presença do mesmo. E por fim, a faixa de distância que o sensor trabalha fica entre 2cm e 3-5m e isto pode variar de sensor para sensor.

#### 11.5.3 Material necessário

**1 Arduino Uno**

**1 sensor ultrassônico**

**1 Cabo USB AB**

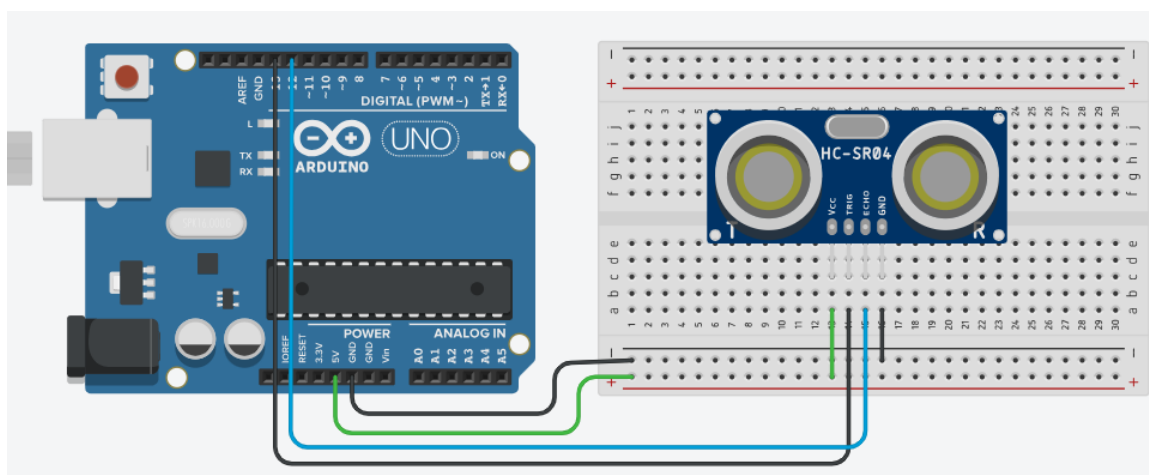


**Jumpers**



**1 Protoboard**

#### 11.5.4 Diagrama



#### 11.5.5 Código Fonte

Vocês verão a distância calculada pelo código no Monitor Serial. Mas como foi calculada essa distância?

Anteriormente, falamos que o sensor emite pulsos de ondas. Esses pulsos são enviados pelo pino Trigger a cada 10us (microssegundos) e retornam para o Echo. A onda ultrassônica

percorre o ar em uma velocidade de 343,5 m/s. Portanto, o cálculo feito para encontrar a distância medida é:

$$(343,5 \text{ m/s} * 100 \text{ cm}) / (1.000.000) = 0,3435$$

$$0,3435 / 2 = 0,017175$$

Multiplicamos a velocidade por 100 para a leitura ser feita em centímetros e após isso dividimos por 1 milhão (equivalente aos 10us). Esse valor é o resultado da distância de ida mais a da volta da onda, por isso, ainda é preciso dividi-lo por 2. Assim, encontramos a variável distância, que é igual a duração do pulso vezes o valor 0,017175.

```
1  /*
2     Trabalhando com sensor ultrassônico
3  */
4
5  const int PinoTrigger = 13;    //O Trigger emite o pulso
6  const int PinoEcho = 12;      //O Echo recebe o pulso
7
8  int duracao;    //Armazenamento do valor lido
9  int distancia; //Distância calculada
10
11 void setup () {
12     pinMode(PinoTrigger, OUTPUT);
13     pinMode(PinoEcho, INPUT);
14     Serial.begin(9600);
15 }
16
17 void loop() {
18     digitalWrite(PinoTrigger, HIGH);
19     delayMicroseconds(10);
20     digitalWrite(PinoTrigger, LOW);
21
22     duracao = pulseIn(PinoEcho, HIGH); //Armazena o valor lido
23     distancia = duracao*0.017175; //Calculo de tempo em distância
24     Serial.print(distancia);
25     Serial.println("cm");
26     delay(100);
27 }
```

Agora façam primeiramente no *TinkerCad*, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

## 12. Introdução a Internet das Coisas

A “Internet das Coisas” (em inglês, *Internet of Things*, ou *IoT*) corresponde a uma revolução tecnológica que tem como objetivo conectar itens usados no dia a dia, como eletrodomésticos, meios de transporte, tênis, roupas e até maçanetas, à rede mundial de computadores.

A *IoT* emergiu dos avanços de várias áreas como sistemas embarcados, microeletrônica, comunicação e sensoriamento.

De fato, tem recebido bastante atenção tanto da academia quanto da indústria, devido ao seu potencial de uso nas mais diversas áreas das atividades humanas.

A Internet das Coisas, em poucas palavras, nada mais é que uma extensão da Internet atual, que proporciona aos objetos do dia a dia (quaisquer que sejam), mas com capacidade computacional e de comunicação, se conectarem à Internet.

A conexão com a rede mundial de computadores viabilizará, primeiro, controlar remotamente os objetos e, segundo, permitir que os próprios objetos sejam acessados como provedores de serviços.

Estas novas habilidades, dos objetos comuns, geram um grande número de oportunidades tanto no âmbito acadêmico quanto no industrial. Todavia, estas possibilidades apresentam riscos e acarretam amplos desafios técnicos e sociais. A IoT tem alterado aos poucos o conceito de redes de computadores, neste sentido, é possível notar a evolução do conceito ao longo do tempo como mostrado a seguir.

Para Tanenbaum [Tanenbaum 2002], “Rede de Computadores é um conjunto de computadores autônomos interconectados por uma única tecnologia”. Entende-se que tal tecnologia de conexão pode ser de diferentes tipos (fios de cobre, fibra ótica, ondas eletromagnéticas ou outras).

Em 2011, Peterson definiu em [Peterson and Davie 2011] que a principal característica das Redes de Computadores é a sua generalidade, isto é, elas são construídas sobre dispositivos de propósito geral e não são otimizadas para fins específicos tais como as redes de telefonia e TV.

Já em [Kurose and Ross 2012], os autores argumentam que o termo “Redes de Computadores” começa a soar um tanto envelhecido devido à grande quantidade de equipamentos e tecnologias não tradicionais que são usadas na Internet. Os objetos inteligentes, definidos mais adiante, possuem papel fundamental na evolução acima mencionada. Isto porque os objetos possuem capacidade de comunicação e processamento aliados a sensores, os quais transformam a utilidade destes objetos.

Atualmente, não só computadores convencionais estão conectados à grande rede, como também uma grande heterogeneidade de equipamentos tais como TVs, Laptops, automóveis, smartphones, consoles de jogos, *webcams* e a lista aumenta a cada dia. Neste novo cenário, a pluralidade é crescente e previsões indicam que mais de 40 bilhões de dispositivos estarão conectados até 2020 [Forbes 2014]. Usando os recursos desses objetos será possível detectar seu contexto, controlá-lo, viabilizar troca de informações uns com os outros, acessar serviços da Internet e interagir com pessoas. Concomitantemente, uma gama de novas possibilidades de aplicações surgem (ex: cidades inteligentes (Smart Cities), saúde (Healthcare), casas inteligentes (Smart Home)) e desafios emergem (regulamentações, segurança, padronizações). É importante notar que um dos elementos cruciais para o sucesso da IoT encontra-se na padronização das tecnologias. Isto permitirá que a heterogeneidade de dispositivos conectados à Internet cresça, tornando a IoT uma realidade. Também é essencial frisar que nos últimos meses e nos próximos anos serão vivenciados os principais momentos da IoT, no que tange as definições dos blocos básicos de construção da IoT.

Na IoT, eventualmente, a unidade básica de hardware apresentará ao menos uma das seguintes características [Ruiz et al. 2004, Loureiro et al. 2003]:

- i) unidade(s) de processamento;
- ii) unidade(s) de memória;
- iii) unidade(s) de comunicação e;
- iv) unidade(s) de sensor(es) ou atuador(es).

Aos dispositivos com essas qualidades é dado o nome de objetos inteligentes (*Smart Objects*). Os objetos, ao estabelecerem comunicação com outros dispositivos, manifestam o conceito de estarem em rede, como discutido anteriormente.

### 12.1. Perspectiva histórica da IoT

Por volta de 2005, o termo bastante procurado (tanto pela academia quanto indústria) e que apresenta relação com a IoT foi Redes de Sensores Sem Fio (RSSF) (do inglês Wireless Sensor Networks – WSN). Estas redes trazem avanços na automação residencial e industrial [Kelly et al. 2013, Da Xu et al. 2014], bem como técnicas para explorar as diferentes limitações dos dispositivos (e.g., memória e energia), escalabilidade e robustez da rede [Loureiro et al. 2003].

Nos anos seguintes (entre 2008 e 2010), o termo Internet das Coisas ganhou popularidade rapidamente. Isto se deve ao amadurecimento das RSSFs e ao crescimento das expectativas sobre a IoT.

A IoT foi identificada como uma tecnologia emergente em 2012 por especialistas da área [Gartner 2015]. Em 2012, foi previsto que a IoT levaria entre cinco e dez anos para ser adotada pelo mercado e, hoje, é vivenciado o maior pico de expectativas sobre a tecnologia no âmbito acadêmico e industrial. Também pode-se notar o surgimento das primeiras plataformas de IoT



que têm gerado uma grande expectativa de seu uso. Estes fatos certamente servem de incentivo para despertar a curiosidade do leitor para a área, bem como indica o motivo do interesse da comunidade científica e industrial para a IoT.

## 12.2. Blocos Básicos de Construção da IoT

A IoT pode ser vista como a combinação de diversas tecnologias, as quais são complementares no sentido de viabilizar a integração dos objetos no ambiente físico ao mundo virtual. A Figura abaixo apresenta os blocos básicos de construção da IoT sendo eles:



**Identificação:** é um dos blocos mais importantes, visto que é primordial identificar os objetos unicamente para conectá-los à Internet. Tecnologias como RFID, NFC (*Near Field Communication*) e endereçamento IP podem ser empregados para identificar os objetos.

**Sensores/Atuadores:** sensores coletam informações sobre o contexto onde os objetos se encontram e, em seguida, armazenam/encaminham esses dados para data *warehouse*, clouds ou centros de armazenamento. Atuadores podem manipular o ambiente ou reagir de acordo com os dados lidos.

**Comunicação:** diz respeito às diversas técnicas usadas para conectar objetos inteligentes. Também desempenha papel importante no consumo de energia dos objetos sendo, portanto, um fator crítico. Algumas das tecnologias usadas são WiFi, Bluetooth, IEEE 802.15.4 e RFID.

**Computação:** inclui a unidade de processamento como, por exemplo, microcontroladores, processadores, responsáveis por executar algoritmos locais nos objetos inteligentes.

**Serviços:** a IoT pode prover diversas classes de serviços, dentre elas, destacam-se os Serviços de Identificação, responsáveis por mapear Entidades Físicas (EF) (de interesse do usuário) em Entidades Virtuais (EV) como, por exemplo, a temperatura de um local físico em seu valor, coordenadas geográficas do sensor e instante da coleta;

Serviços de Agregação de Dados que coletam e sumarizam dados homogêneos/heterogêneos obtidos dos objetos inteligentes; Serviços de Colaboração e Inteligência que agem sobre os serviços de agregação de dados para tomar decisões e reagir de modo adequado a um determinado cenário; e Serviços de Ubiquidade que visam prover serviços de colaboração e inteligência em qualquer momento e qualquer lugar em que eles sejam necessários.

**Semântica:** refere-se à habilidade de extração de conhecimento dos objetos na IoT. Trata da descoberta de conhecimento e uso eficiente dos recursos existentes na IoT, a partir dos dados existentes, com o objetivo de prover determinado serviço. Para tanto, podem ser usadas diversas técnicas como *Resource Description Framework* (RDF), *Web Ontology Language* (OWL) e *Efficient XML Interchange* (EXI).

## 12.3. Dispositivos e Tecnologias de Comunicação

Esta seção aborda em mais detalhes a arquitetura básica dos dispositivos inteligentes e as tecnologias de comunicação, principalmente as soluções de comunicação sem fio que tendem a se popularizar no ambiente de IoT.

### 12.3.1. Arquiteturas básica dos dispositivos

A arquitetura básica dos objetos inteligentes é composta por quatro unidades: processamento/memória, comunicação, energia e sensores/atuadores. A figura a seguir apresenta uma visão geral da arquitetura de um dispositivo e a interligação entre seus componentes, os quais são descritos a seguir:

(i) **Unidade(s) de processamento/- memória:** composta de uma memória interna para armazenamento de dados e programas, um microcontrolador e um conversor analógico-digital para receber sinais dos sensores. As CPUs utilizadas nesses dispositivos são, em geral, as mesmas utilizadas em sistemas embarcados e comumente não apresentam alto poder computacional. Frequentemente existe uma memória externa do tipo flash, que serve como memória secundária, por exemplo, para manter um “log” de dados. As características desejáveis para estas unidades são consumo reduzido de energia e ocupar o menor espaço possível.

(ii) **Unidade(s) de comunicação:** consiste de pelo menos um canal de comunicação com ou sem fio, sendo mais comum o meio sem fio. Neste último caso, a maioria das plataformas usam rádio de baixo custo e baixa potência. Como consequência, a comunicação é de curto alcance e apresentam perdas frequentes. Esta unidade básica será objeto de estudo mais detalhado na próxima seção.

(iii) **Fonte de energia:** responsável por fornecer energia aos componentes do objeto inteligente. Normalmente, a fonte de energia consiste de uma bateria (recarregável ou não) e um conversor AC-DC e tem a função de alimentar os componentes. Entretanto, existem outras fontes de alimentação como energia elétrica, solar e mesmo a captura de energia do ambiente através de técnicas de conversão (e.g., energia mecânica em energia elétrica), conhecidas como *energy harvesting*.

(iv) **Unidade(s) de sensor(es)/atuador(es):** realizam o monitoramento do ambiente no qual o objeto se encontra. Os sensores capturam valores de grandezas físicas como temperatura, umidade, pressão e presença. Atualmente, existem literalmente centenas de sensores diferentes que são capazes de capturar essas grandezas. Atuadores, como o nome indica, são dispositivos que produzem alguma ação, atendendo a comandos que podem ser manuais, elétricos ou mecânicos.

### 13. Sistemas operacionais de Tempo Real para IoT

Um Sistema Operacional da Internet das Coisas é um sistema operacional projetado para atuar dentro das restrições específicas dos dispositivos da Internet das Coisas, incluindo restrições de memória, tamanho, potência e capacidade de processamento. Na verdade, este é um Sistema Operacional Embarcado, mas por definição são projetados para permitir a transferência de dados pela internet.

IoT OS controlam sistemas em carros, luzes de trânsito e ruas, *Smart TVs*, caixas eletrônicas, controles de avião, terminais de ponto de venda, câmeras digitais, sistemas de navegação GPS, elevadores, receptores de mídia digital e medidores inteligentes entre muitas outras possibilidades. Embora o IoT OS seja uma evolução do sistema operacional embarcado, o IoT traz seu próprio conjunto adicional de restrições que precisam ser resolvidas.

#### 13.1. Quais são os parâmetros para selecionar um IoT OS adequado?

Seguem alguns parâmetros a ser considerados para selecionar um IoT OS:

- **Footprint:** Como os dispositivos são restritos, esperamos que o sistema operacional requiera pouca memória, energia e requisitos de processamento. A sobrecarga devida ao sistema operacional deve ser mínima;
- **Portabilidade:** O sistema operacional isola os aplicativos das especificações do hardware. Normalmente, o sistema operacional é portado para diferentes plataformas de hardware e interfaces para o pacote de suporte de placa (BSP) de maneira padrão, como o uso de chamadas POSIX;
- **Conectividade:** O OS suporta diferentes protocolos de conectividade, como Ethernet, Wi-Fi, BLE, IEEE 802.15.4 e muito mais;
- **Escalabilidade:** O sistema operacional deve ser escalável para qualquer tipo de dispositivo. Isso significa que desenvolvedores e integradores precisam estar familiarizados com apenas um sistema operacional para dispositivos edge e gateways;
- **Confiabilidade:** Essencial para sistemas de missão crítica. Frequentemente, os dispositivos estão em locais remotos e precisam funcionar por anos sem falhas. A confiabilidade também implica que o sistema operacional deve cumprir as certificações para determinados aplicativos;

- **Segurança:** O sistema operacional possui complementos que trazem segurança ao dispositivo por meio de inicialização segura, suporte a SSL, componentes e drivers para criptografia.

### 13.2. Alguns Sistemas Operacionais

- **RIOT-OS:** Tem sua origem em 2008 como um SO para redes de sensores wireless e atualmente se autodenomina o “Sistema Operacional Amigável para a Internet das Coisas”. Com um *footprint* próximo de 1,5kB de RAM e 5kB de flash, possui um escalonador de tempo real baseado em prioridades e uma API (parcialmente) implementada no padrão POSIX. Roda em microcontroladores de 8, 16 e 32 bits e possui suporte a gerenciamento de energia. Conectividade é um dos pontos fortes deste sistema operacional, incluindo o suporte aos padrões e pilhas de protocolo IPv6, 6LoWPAN, RPL, UDP e CoAP. Possui um porte para Linux onde é possível estudar e aprender sobre sua API, e suporta ferramentas comuns de desenvolvimento como GCC e GDB. Possibilita o desenvolvimento de aplicações em C/C++ e implementa algumas funcionalidades interessantes como um terminal de comandos, alguns algoritmos de criptografia e bibliotecas para manipulação de estruturas de dados. Liberado sob licença LGPL, possui suporte a diferentes arquiteturas de CPU incluindo x86, AVR, ARM7, ARM Cortex-M e MSP430.
- **CONTIKI:** Foi criado em 2002 por Adam Dunkels (o mesmo criador das pilhas de protocolo uIP e lwIP) e hoje é mantido por uma equipe grande de desenvolvedores ao redor do mundo. O projeto se autodenomina o “Sistema Operacional de Código Aberto para a Internet das Coisas”. Com um *footprint* de aproximadamente 10kB de RAM e 30kB de flash, possui suporte abrangente da pilha de protocolos TCP/IP, além dos protocolos IPv6 focados em baixo consumo como 6LoWPAN, RPL e CoAP. Projetado para rodar em sistemas com baixíssimo consumo de energia, ele provê alguns mecanismos interessantes para estimar e analisar o consumo de energia da aplicação.
- **NUTTX:** É um sistema operacional de tempo real de código aberto para microcontroladores com uma ênfase em compatibilidade com padrões de mercado (POSIX e ANSI) e baixo consumo de recursos (CPU, RAM e flash). Criado por Gregory Nutt (daí o seu nome) em 2007 e liberado sob licença BSD, suporta diversas arquiteturas de microcontroladores, de 8, 16 e 32 bits, incluindo ARM, AVR, 8051, MIPS e z80. Possui funcionalidades comuns em sistemas operacionais de tempo real como determinismo, preempção e herança de prioridade.
- **ZEPHYR:** Lançado este ano (2016) pela Linux Foundation em parceria com grandes empresas como a Intel e a NXP, o Zephyr é um sistema operacional de tempo real para dispositivos de IoT com foco em conectividade, modularidade e segurança. É um sistema operacional multithread com um escalonador preemptivo baseado em prioridades e possui diversas funcionalidades para economia de energia. Utiliza um conceito interessante chamado fiber, que é basicamente uma tarefa de alta prioridade (comparada às tarefas comuns) escalonada de modo colaborativo. Possui baixo footprint (começando em 8kB de RAM) e suporta diversos protocolos e padrões de mercado como Bluetooth, Bluetooth LE, WiFi, 802.15.4, 6Lowpan, CoAP, IPv4, IPv6 e NFC. Liberado sob licença Apache, no momento são suportadas as arquiteturas ARM Cortex-M, Intel x86, e ARC, incluindo portes para as placas Arduino 101, Arduino Due, Intel Galileo Gen 2 e a Freedom Board FRDM-K64F da NXP.
- **BRILLO:** Este é outro sistema operacional que vem com bastante força, já que tem a empresa Google por trás. Anunciado em 2015, o Brillo é uma plataforma de IoT que inclui uma versão reduzida do Android e um protocolo chamado Weave para conexão entre os dispositivos IoT. Desenvolvido em parceria com a Nest (aquela empresa fabricante de termostatos digitais que a Google comprou por US\$ 3,2 bilhões!), possui suporte às arquiteturas ARM, Intel e MIPS. Diferentemente dos sistemas operacionais apresentados anteriormente, o Brillo não foi feito para microcontroladores, já que possui um footprint mínimo de 32MB de RAM e 128MB de flash.

- **MBED OS:** Vem com a força da ARM e suporta apenas microcontroladores ARM Cortex-M. É um sistema operacional modular, seguro e com foco em conectividade. É capaz de garantir isolamento de memória entre as tarefas através da MPU (Memory Protection Unit) existente em microcontroladores Cortex M3/M4, e suporta diversos protocolos e padrões de comunicação como Ethernet, WiFi, IPv6, 6LoWPAN, TLS, Bluetooth Low Energy.

#### 14. Protocolos de comunicação IoT

Os protocolos de comunicação disponíveis para implementar sistemas de Internet das coisas são bem diversificados. Os projetistas de equipamentos eletrônicos e desenvolvedores possuem um leque interessante de opções para incorporar em seus projetos.

Alguns protocolos clássicos como WiFi e Bluetooth são bastante conhecidos da comunidade de desenvolvedores e usuários.

Algumas alternativas emergentes, porém, como Thread e Z-Wave, ambas focadas em automação residencial, ainda são menos difundidas.

Com a popularização do conceito de IoT, no entanto, uma série de protocolos começam a aparecer no dia-a-dia tecnológico e você deve conhecer os principais deles. Vamos ver alguns protocolos.

##### 14.1. Transporte de Telemetria da Fila de Mensagens (MQTT)

O MQTT, também conhecido como protocolo de assinatura/publicação, é um protocolo de mensagens leve e o mais preferível para dispositivos IoT. Ele coleta dados de vários dispositivos e supervisiona dispositivos remotos. Ele é executado sobre o *Transmission Control Protocol* (TCP) e oferece suporte à troca de mensagens orientada a eventos por meio de redes sem fio. O MQTT é usado principalmente em dispositivos que exigem menos memória, por exemplo, sensores no carro e *smartwatches*. Iremos fazer projetos com ele.

##### 14.2. Protocolo de comunicação máquina a máquina (M2M)

Refere-se a um protocolo aberto para indústria. O M2M foi criado para gerenciar dispositivos IoT remotamente. Esses protocolos econômicos usam redes públicas. O M2M cria um ambiente onde duas máquinas se comunicam mutuamente e trocam dados. Tal protocolo reforça o automonitoramento das máquinas e permite que os sistemas se adaptem de acordo com o ambiente variável. É usado principalmente para casas inteligentes, veículos e caixas eletrônicos.

##### 14.3. Bluetooth

O Bluetooth é amplamente usado para comunicação de curto alcance e é um protocolo IoT padrão para transmissão de dados sem fio. Sua versão de baixo consumo de energia é *Bluetooth Low Energy* (BLE). A versão mais recente, BLE 5.0, suporta aplicações de baixa taxa de dados e um alcance estendido de até 150 metros. Recursos como balizamento e serviços de localização ajudaram a implementá-lo em uma ampla gama de aplicações de fitness e automotivas. Pode suportar topologia em estrela. As versões mais recentes suportam topologia de malha, estendendo a rede usando a rede de muitos para muitos dispositivos adequada para aplicações de automação residencial.

##### 14.4. Wi-fi

WiFi é um protocolo de comunicação sem fio. O WiFi usa a topologia de rede em estrela e o ponto de acesso pode ser usado como um gateway para a Internet. Cada ponto de acesso pode se conectar a um máximo de 250 dispositivos, e a maioria das soluções disponíveis comercialmente suportam até 50 dispositivos. O 802.11-b/g/n opera em 2,4 GHz e fornece taxa de dados de 150-200 Mbps no ambiente doméstico ou no escritório, normalmente em uma faixa de 50 metros. O padrão 802.11-ac mais recente funciona em 5 GHz e fornece uma taxa de dados de 500 Mbps-1 Gbps.

##### 14.5. Celular

Muitas aplicações de IoT usam redes celulares existentes, como 3G, 4G LTE e 5G para comunicação de dados. O 3G usa 2100 MHz e oferece uma taxa de dados de 384 Kbps-10Mbps, e o 4G LTE oferece uma alta taxa de dados de 3Mbps-10 Mbps a 2700 MHz. Eles são inadequados para a maioria dos aplicativos de IoT devido ao alto consumo de energia e altos custos de implementação.

#### 14.6. NFC

*Near Field Communication* (NFC) é um protocolo de comunicação de rádio de alcance ultracurto. Ele usa o padrão ISO/IEC 18000-3 e a banda de frequência ISM de 13,56 MHz. Ele fornece uma taxa de dados de 100-420 Kbps e um alcance de até 20 cm. Alguns dispositivos NFC podem ler (em conformidade com ISO 15693) etiquetas RFID passivas de alta frequência, que também funcionam em 13,56 MHz. O NFC fornece comunicação *full-duplex* na faixa de detecção de substratos metálicos e não metálicos. Ele é usado para pagamento sem contato, sincronização rápida e aplicativos de acesso a conteúdo digital.

#### 15. ESP32

O ESP32 é um dispositivo IoT (Internet das Coisas) que consiste de um microprocessador de baixa potência dual core Tensilica Xtensa 32-bit LX6 com suporte embutido à rede WiFi, Bluetooth v4.2 e memória flash integrada. Essa arquitetura permite que ele possa ser programado de forma independente, sem a necessidade de outras placas microcontroladoras como o Arduino, por exemplo. Dentre as principais características deste dispositivo, podemos citar: baixo consumo de energia, alto desempenho de potência, amplificador de baixo ruído, robustez, versatilidade e confiabilidade.

##### 15.1. ESP8266 ou ESP-01

Posteriormente, a ESP8266 ganhou espaço na cultura *maker* no ano de 2014, a partir disso, ela tornou-se um grande sucesso pela sua capacidade de fazer conexões WiFi.

Sendo assim, esse chip também possui um poder de processamento muito superior ao popularmente conhecido Arduino UNO.

Além disso, na data de lançamento seu preço já era abaixo de 10 dólares. Dessa maneira, por contar com todos esses recursos, se tornou famoso no mundo dos desenvolvedores.

Nesse sentido, vamos ver a seguir algumas versões mais avançadas que surgiram no decorrer do desenvolvimento da linha ESP.

##### 15.2. ESP12 e NODE MCU

Em seguida, a fabricante acertou novamente em disponibilizar, o chip chamado de NODE MCU, ou seja, uma evolução do ESP8266. Tal versão, agora acompanhava também, uma conexão USB e um layout que facilita muito a utilização da placa com a protoboard.

##### 15.3. ESP32 e o Advento da Internet Das Coisas

Essencialmente, já se sabe que, em time que está ganhando, não se mexe. E foi exatamente isso que a *Espressif* pensou, ao lançar o modelo ESP32.

Acompanhando o modelo *StandAlone* e também o modelo de desenvolvimento, a linha ESP dessa vez chegou com um super processador dual core, com até 240MHZ

de velocidade, contando ainda com conexão *WiFi* e *Bluetooth BLE*.

Isso quer dizer que, além do chip possuir conexão com a internet sem fio, agora possuía também conexão *bluetooth* e processador com dois núcleos.

Vale lembrar que, em todos os casos, o firmware dessa placa é baseado no RTOS, que permite fazer o gerenciamento de multitarefa e os gerenciamentos dos núcleos da placa.

Como resultado, não demorou muito para essa placa se tornar a queridinha dos *makers*.

Definitivamente, hoje, a placa começa a se tornar uma referência em Internet das Coisas.

##### 15.4. Características principais

- Chip com WiFi embutido: padrão 802.11 B/G/N, operando na faixa de 2.4 a 2.5GHz
- Modos de operação: Client, Access Point, Station+Access Point
- Microprocessador dual core Tensilica Xtensa 32-bit LX6
- Clock ajustável de 80MHz até 240MHz
- Tensão de operação: 3.3 VDC
- Possui SRAM de 512KB
- Possui ROM de 448KB
- Possui memória flash externa de 32Mb (4 megabytes)
- Corrente máxima por pino é de 12mA (recomenda-se usar 6mA)
- Possui 36 GPIOs
- GPIOs com função PWM / I2C e SPI
- Possui Bluetooth v4.2 BR / EDR e BLE (Bluetooth Low Energy)



### 15.5. Comparativo entre ESP32, ESP8266 e Arduino R3

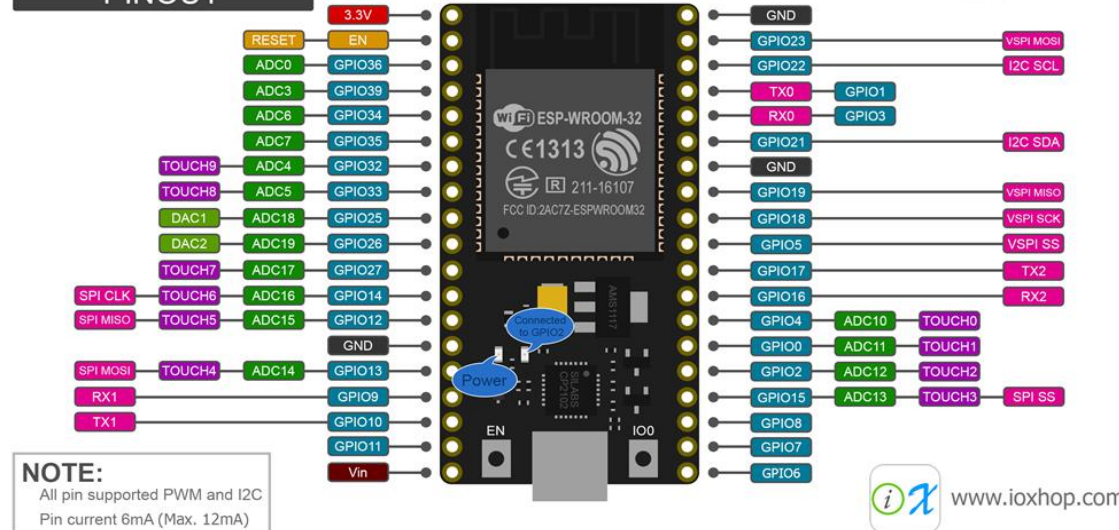
	ESP32	ESP8266	ARDUINO UNO R3
Cores	2	1	1
Arquitetura	32 bits	32 bits	8 bits
Clock	160MHz	80MHz	16MHz
WiFi	Sim	Sim	Não
Bluetooth	Sim	Não	Não
RAM	512KB	160KB	2KB
FLASH	16Mb	16Mb	32KB
GPIO	36	17	14
Interfaces	SPI / I2C / UART / I2S / CAN	SPI / I2C / UART / I2S	SPI / I2C / UART
ADC	18	1	6
DAC	2	0	0

### 15.6. Configurando IDE do Arduino (Windows)

Este é o diagrama do ESP que estamos usando na nossa montagem. Trata-se de um chip que tem muito recurso e poder. São diversos pinos que você escolher se quer trabalhar como analógico digital, digital analógico ou ainda se quer trabalhar a porta como digital.

## NodeMCU-32S

### PINOUT



#### 15.6.1. Portas GPIO

GPIO, acrônimo de *General Purpose Input/Output*, são pinos responsáveis pela entrada e saída de sinais digitais. Elas são capazes, individualmente, de fornecer um máximo de 12 mA de corrente.

#### 15.6.2. Portas Touch

Esses pinos funcionam como sensores capacitivos, ou seja, reagem quando há uma alteração em seu estado. Um simples exemplo é tocá-lo: isso iniciará um processo de troca de calor entre dois corpos distintos, onde o sensor perceberá e emitirá sinais para a ESP. Essa placa possui 10 pinos *touch*.

#### 15.6.3. ADC

Os ADC (*Analog Digital Converter*), são famosos na área da eletrônica. Eles convertem grandezas analógicas (velocímetro de ponteiro do carro, por exemplo) em grandezas digitais (um velocímetro digital, por exemplo). Segundo o diagrama do ESP, ele possui 16 pinos com essa capacidade.

#### 15.6.4. Demais pinos

O NodeMCU ESP32 conta com 3 portas GND, 1 porta de 3.3V e 1 porta de 5V (que no diagrama está identificada por Vin, visto que pode ser usada para alimentação da placa caso ela não esteja conectada pelo cabo USB). Há também outros pinos, como o RX e TX, CLK, MISO e outros, que serão explorados em outros artigos aqui no Blog. Note que alguns deles acumulam diversas funções em um pino só.

### 16. WiFi Scan

Nesta introdução ao ESP32 vamos fazer um programa que serve como um exemplo de como buscar as redes WiFi disponíveis próximas ao ESP-32, assim como a intensidade do sinal de cada uma delas e também se precisa ser informada senha. A cada escaneamento, também podemos descobrir qual a rede com a melhor intensidade de sinal, mas antes vamos ver algumas definições.

#### 16.1. O que é um sinal WiFi?

Uma Rede WiFi emprega ondas de rádio para estabelecer a comunicação entre dispositivos. Esses dispositivos podem incluir computadores, telefones celulares, tablets ou roteadores de rede. O roteador de rede sem fio é a interface entre uma conexão com fio à Internet ou outra rede Ethernet e os dispositivos conectados sem fio.

O roteador decodifica os sinais de rádio recebidos dos usuários da rede WiFi e os transmite para a Internet. Por outro lado, os dados recebidos da Internet são convertidos de dados binários em ondas de rádio para distribuição aos dispositivos que estão usando a rede.

As ondas de rádio que compreendem os sinais WiFi utilizam as bandas de frequências de 2,4 GHz e 5 GHz. Elas são mais altas do que as frequências usadas por televisões ou telefones celulares e permitem que mais dados sejam transportados do que as frequências mais baixas.

Os sinais WiFi usam os padrões de rede 802.11 ao transmitir dados. Existem diferentes protocolos usados na rede WiFi. Alguns dos mais comuns que você verá são 802.11n, que é usado na banda de 2.4GHz e 802.11ac usado principalmente para transmissão de 5GHz.

#### 16.2. O que é uma boa força de sinal Wifi?

A força do sinal WiFi em toda a área de cobertura da rede impacta diretamente a capacidade dos usuários de realizar várias atividades em tempo hábil. Antes de investigar a força de sinal adequada para determinados usos da sua rede WiFi, vamos discutir como é medida a força do sinal WiFi.

O método mais consistente de indicar a intensidade do sinal é através da medida dBm. Este termo significa decibéis relativos a um miliwatt e é expresso como um número negativo de 0 a -100. Portanto, um sinal de -40 é mais poderoso do que um sinal de -80, já que -80 é maior que 0 e, portanto, um número menor.

A escala dBm é logarítmica e não linear, o que significa que as mudanças entre as intensidades do sinal não são escalonadas de maneira suave e gradual. Nesta escala, uma diferença de 3 dBm leva à redução pela metade ou ao dobro da força do sinal anterior.

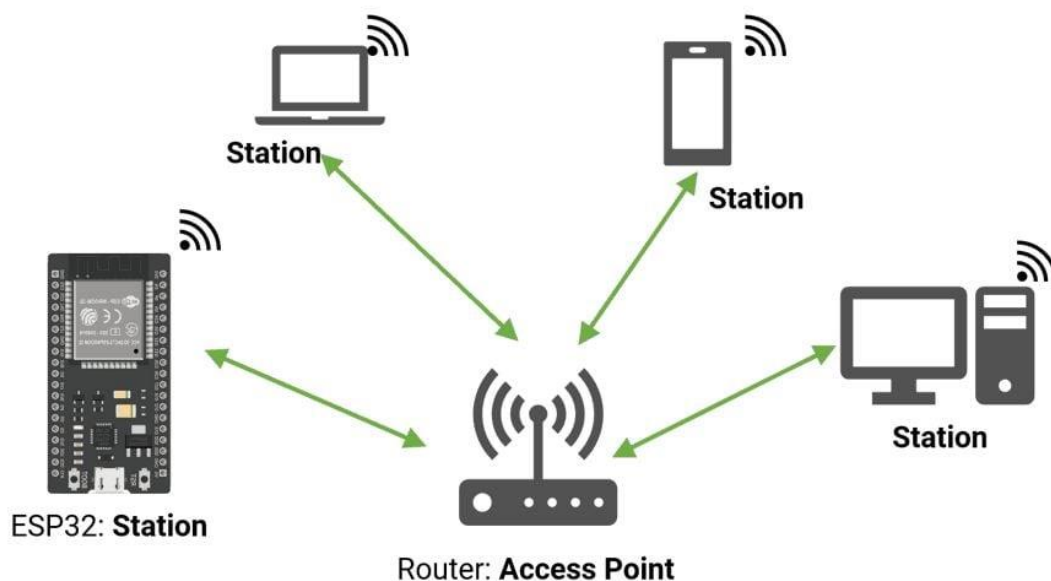
Os níveis de ruído de fundo que podem afetar seu desempenho de WiFi também são expressos em dBms. No caso de níveis de ruído, um valor próximo de zero indica um alto nível de ruído. O ruído medido a -10 é maior que o de -40.

A tabela a seguir indica a intensidade mínima de sinal para se ter em sua rede WiFi para realizar as atividades necessárias.

Força do sinal	Qualificador	Usos Adequados
<b>-30 dBm</b>	Excelente	Esta é a intensidade de sinal máxima alcançável e será apropriada para qualquer cenário de uso.
<b>-50 dBm</b>	Excelente	Este excelente nível de sinal é adequado para todos os usos da rede.
<b>-65 dBm</b>	Muito Bom	Recomendado para o uso de smartphones e tablets.
<b>-67 dBm</b>	Muito Bom	Esta intensidade do sinal será suficiente para voz sobre IP e streaming de vídeo.
<b>-70 dBm</b>	Aceitável	Esse nível é a intensidade mínima do sinal necessária para garantir a entrega confiável de pacotes e permitir que você navegue na Web e troque e-mails.
<b>-80 dBm</b>	Ruim	Permite conectividade básica, mas a entrega de pacotes não é confiável.
<b>-90 dBm</b>	Muito ruim.	Cheia de ruído que inibe a maior parte da funcionalidade da rede.
<b>-100 dBm</b>	Extremamente ruim	Ruído total.

### 16.3. Noções básicas sobre o ESP32 WiFi primeiros passos

O ESP32 tem funcionalidades Wi-Fi embutidas. Ele tem três modos Wi-Fi. Antes disso, vamos entender os termos estação Wi-Fi e ponto de acesso



- **Estação Wi-Fi:** É um dispositivo que pode se conectar a outras redes Wi-Fi, como roteadores Wi-Fi. Também é chamado de nó ou cliente sem fio.
- **Access Point:** Dispositivo como Roteador Wi-Fi, que permite que outros dispositivos Wi-Fi se conectem a ele. Geralmente, Access points são conectados a uma conexão com fio como **Ethernet**.
- **SoftAP (*Software Enabled Access Point*):** Dispositivos que não são especificamente roteadores, mas podem ser habilitados como um ponto de acesso chamado SoftAP. Por exemplo, Smartphone *Hotspot*.

#### 16.4. Modos Wi-Fi ESP32

Os modos Wi-Fi são definidos na WiFi.h biblioteca. Precisamos incluí-la.

```
#include <WiFi.h>
```

Precisamos usar WiFi.mode()a função para definir o modo.

1	<b>Modo Estação:</b> O ESP32 se conecta a um ponto de acesso.	Modo WiFi(WIFI_STA)
2	<b>Modo de ponto de acesso:</b> as estações podem se conectar ao ESP32	Modo WiFi(WIFI_AP)
3	<b>Ponto de acesso e modo Hotspot:</b> ambos ao mesmo tempo	Modo WiFi(WIFI_AP_STA)

#### 16.5. Código fonte

Iremos implementar um projeto que busca as redes disponíveis no alcance do ESP32.

```
01-BuscaRedesDisponiveis
/*
  Professor Marcos Costa
  ESP32 WiFi Scanning
  Verifica as conexões disponíveis e se há necessidade
  de informar senha.
*/

//Biblioteca WiFi
#include "WiFi.h"

void setup() {
  // Iniciando a comunicação serial:
  Serial.begin(115200); //Velocidade -> Comunicação Serial
  Serial.println("Iniciando WiFi ...");
  WiFi.mode(WIFI_STA); // Modo de conexão, neste caso iremos procurar roteadores
  Serial.println("Configuração concluída!");
}

void loop() {
  Serial.println("Procurando ...");

  int qtdeRedes = WiFi.scanNetworks(); //Quantidade de redes encontradas
  Serial.println("Verificação concluída.");

  if (qtdeRedes == 0){
    Serial.println("Nenhuma rede encontrada.");
  }else{
    Serial.print("Total de rede(s) encontrada(s): ");
    Serial.println(qtdeRedes);
  }
}
```

```
//Estrutura de repetição para mostrarmos todas as redes encontradas
for (int i = 0; i < qtdeRedes; i++){
    Serial.print(i + 1); //Sequencia numérica
    Serial.print(": ");

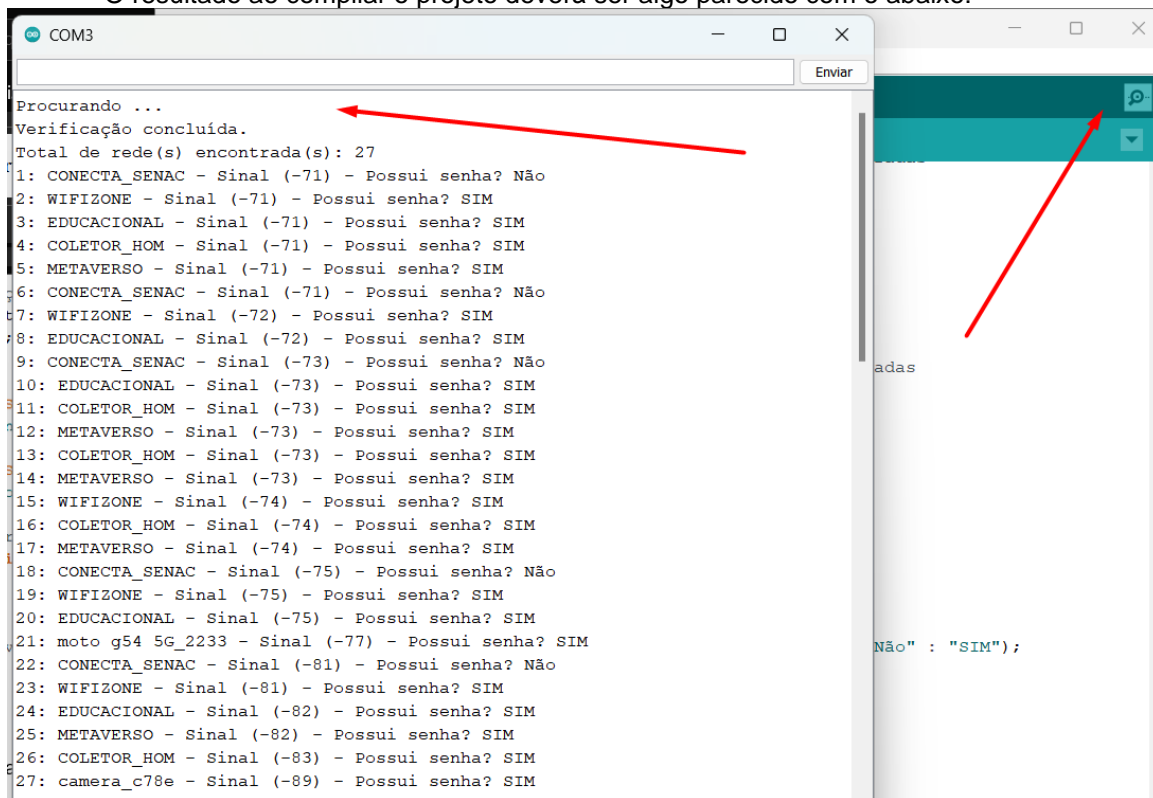
    Serial.print(WiFi.SSID(i)); //Nome da rede encontrada
    Serial.print(" - Sinal ");

    Serial.print(WiFi.RSSI(i)); //Intensidade do Sinal
    Serial.print(" - Possui senha? ");

    //WIFI_AUTH_OPEN verifica se a rede possui senha
    Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? "Não" : "SIM");
    delay(10);
}

//Tempo para próxima verificação
delay(5000);
}
```

O resultado ao compilar o projeto deverá ser algo parecido com o abaixo:



### 16.5. Aprimorando o código anterior

Agora, para mais uma prática em nosso curso, vamos fazer a conexão com a rede que estiver aberta, ou seja, não possuir senha para autenticação e com melhor sinal.

Vamos utilizar o mesmo projeto que estamos trabalhando, só acrescentar algumas funcionalidades, observando o código a seguir, podemos verificar que não teremos mudanças bruscas, somente uma função for que fará este trabalho e informará a melhor rede sem senha para se acessar, vejamos:



02-BuscaRedesDisponiveisEMelhorSinalAberto §

```

/*
  Professor Marcos Costa
  ESP32 WiFi Scanning
  Verifica as conexões disponíveis e se há necessidade
  de informar senha e conectar a melhor rede aberta disponível
*/

//Biblioteca WiFi
#include "WiFi.h"

void setup() {
  // Iniciando a comunicação serial:
  Serial.begin(115200); //Velocidade -> Comunicação Serial
  Serial.println("Iniciando WiFi ...");
  WiFi.mode(WIFI_STA); // Modo de conexão, neste caso iremos procurar roteadores
  Serial.println("Configuração concluída!");
}

void loop() {
  Serial.println("Procurando ...");

  int qtdeRedes = WiFi.scanNetworks(); //Quantidade de redes encontradas
  Serial.println("Verificação concluída.");

  if (qtdeRedes == 0){
    Serial.println("Nenhuma rede encontrada.");
  }else{
    Serial.print("Total de rede(s) encontrada(s): ");
    Serial.println(qtdeRedes);

    //Estrutura de repetição para mostrarmos todas as redes encontradas
    for (int i = 0; i < qtdeRedes; i++){
      Serial.print(i + 1); //Sequencia numérica
      Serial.print(": ");

      Serial.print(WiFi.SSID(i)); //Nome da rede encontrada
      Serial.print(" - Sinal ");

      Serial.print(WiFi.RSSI(i)); //Intensidade do Sinal
      Serial.print(" - Possui senha? ");

      //WIFI_AUTH_OPEN verifica se a rede possui senha
      Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? "Não" : "SIM");
      delay(10);
    }

    //Verificando a rede aberta com melhor sinal
    Serial.println("Verificando a rede aberta com melhor sinal");
    for (int i = 0; i < qtdeRedes; ++i) {
      if (WiFi.encryptionType(i) == WIFI_AUTH_OPEN) {
        Serial.println("Rede aberta com melhor sinal:");
        Serial.print("Nome da Rede: ");
        Serial.print(WiFi.SSID(i));
        Serial.print(" - Sinal: ");
        Serial.print(WiFi.RSSI(i));
        Serial.println();
        break; //Finaliza o loop
      }
    }

    //Tempo para próxima verificação
    delay(5000);
  }
}

```

Parte acrescida ao projeto anterior

### 16.6. Conectando o ESP32 em uma rede aberta.

Agora, melhorando mais ainda nosso projeto, vamos conectar nosso ESP32 na rede aberta encontrada, para isso teremos que modificar um pouco nosso projeto, acrescentando uma rotina de conexão, mas além disso, vamos armazenar a mesma, para que em verificações posteriores, caso esteja conectado, não faça o mesmo trabalho. Veja:

```
03-BuscaRedesDisponiveisEMelhorSinalAbertoEConectar
```

```
/*
  Professor Marcos Costa
  ESP32 WiFi Scanning
  Verifica as conexões disponíveis e se há necessidade
  de informar senha e conectar a melhor rede aberta disponível
  */

//Biblioteca WiFi
#include "WiFi.h"

String wifiConnected = ""; //Variável para controle do nome da rede conectada

void setup() {
  // Iniciando a comunicação serial:
  Serial.begin(115200); //Velocidade -> Comunicação Serial
  Serial.println("Iniciando WiFi ...");
  WiFi.mode(WIFI_STA); // Modo de conexão, neste caso iremos procurar roteadores
  Serial.println("Configuração concluída!");
}

void loop() {
  Serial.println("Procurando ...");

  int qtdeRedes = WiFi.scanNetworks(); //Quantidade de redes encontradas
  Serial.println("Verificação concluída.");

  if (qtdeRedes == 0){
    Serial.println("Nenhuma rede encontrada.");
  }else{
    Serial.print("Total de rede(s) encontrada(s): ");
    Serial.println(qtdeRedes);

    //Estrutura de repetição para mostrarmos todas as redes encontradas
    for (int i = 0; i < qtdeRedes; i++){
      Serial.print(i + 1); //Sequencia numérica
      Serial.print(": ");

      Serial.print(WiFi.SSID(i)); //Nome da rede encontrada
      Serial.print(" - Sinal ");

      Serial.print(WiFi.RSSI(i)); //Intensidade do Sinal
      Serial.print(" - Possui senha? ");

      //WIFI_AUTH_OPEN verifica se a rede possui senha
      Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? "Não" : "SIM");
      delay(10);
    }

    //Verificando a rede aberta com melhor sinal
    Serial.println("-----");
    Serial.println("Verificando a rede aberta com melhor sinal");
  }
}
```

```
for (int i = 0; i < qtdeRedes; ++i) {
  if (WiFi.encryptionType(i) == WIFI_AUTH_OPEN) {
    Serial.print("Rede aberta com melhor sinal: ");
    Serial.print(WiFi.SSID(i));
    Serial.print(" - Sinal: ");
    Serial.println(WiFi.RSSI(i));
    Serial.println("-----");

    Serial.println();

    if (wifiConnected == ""){
      //Conectando a rede aberta com melhor sinal
      Serial.print("Conectando a melhor rede aberta encontrada: ");
      Serial.println(WiFi.SSID(i));

      //Efetua a conexão com a rede wifi
      WiFi.begin(WiFi.SSID(i));

      //Variável para controlar as tentativas de conexão
      int tentativas = 0;

      while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        tentativas++;
        //Foi tentado conectar 100 vezes, como não conseguir, para o processo
        if (tentativas == 100) break; //Mantém aguardando conexão por 15 segundos
        delay(100);
      }

      Serial.println("");

      if (WiFi.status() == WL_CONNECTED) {
        wifiConnected = WiFi.SSID(i);
        Serial.println("WiFi conectado!" + wifiConnected);
      } else {
        wifiConnected = "";
        Serial.println("Nao foi possivel conectar.");
      }
      break; //Finaliza o loop
    }else{
      Serial.println("ESP32 ja conectado a Rede: " + wifiConnected);
    }
    Serial.println("-----");
  }
}

//Tempo para próxima verificação
delay(5000);
}
```

Com estas atividades, concluímos a parte de conhecimento WiFi e algumas práticas de manuseio do ESP32 com esta tecnologia, vale salientar que foram práticas que poderiam ser desenvolvidas de outras maneiras e com lógicas diferentes, isso cabe a cada estudante, mudar e manusear de forma diferente o conteúdo.

#### 16.7. Conectando o Esp32 em uma rede WiFi

Agora, a ideia é colocarmos o ESP32, em um rede WiFi e fazermos um Ping em alguns sites, com isso iremos colocar nosso ESP na internet, iremos fazer um programa onde iremos configurar a rede e em seguida pingar sites.

#### 16.7.1. O que vou aprender?

- O que é uma biblioteca;
- O que é *Array*;
- Funções;
- Constante.

#### 16.7.2. Conhecimentos prévios

##### 16.7.2.1. O que é uma biblioteca?

*Uma biblioteca nada mais é do que um conjunto de instruções desenvolvidas para executar tarefas específicas relacionadas a um determinado dispositivo. Por exemplo, pensando em um LCD e sua biblioteca *LiquidCrystal.h*. Essa biblioteca tem funções desenvolvidas especificamente para executar tarefas como, configurar o LCD, limpar a tela, mover o texto para direita e esquerda e etc.*

A utilização de uma biblioteca facilita o desenvolvimento, tornando o código mais simples e organizado. É obrigatório o uso dessa biblioteca? Não. Mas então teríamos que escrever, muito mais linhas para que o LCD funcionasse.

Bibliotecas podem ser desenvolvidas por qualquer pessoa. Podemos ver muitas bibliotecas da Adafruit e Sparkfun, dentre as outras milhares espalhadas pelo GitHub. Podemos encontrar bibliotecas ótimas, ruins e até mesmo algumas que não funcionam. Uma dica interessante ao avaliar uma biblioteca é olhar o número de estrelas e issues no GitHub e ver também se ela está disponível oficialmente na IDE Arduino.

E foram as bibliotecas que ajudaram a tornar o Arduino tão popular, onde crianças e adultos não precisam ser engenheiros ou desenvolvedores de software, para criarem seus projetos e colocarem sua criatividade em prática. O próprio “jeitão” de escrever código para Arduino segue o princípio da facilidade. Por exemplo a função `digitalWrite(pino, valor)`. Com apenas essa função ligamos ou desligamos um pino do Arduino.

##### 16.7.2.2. O que é um *Array*?

*Array* é um tipo especial de variável que pode armazenar uma série de elementos de dados ao mesmo tempo. Além disso, cada valor de um *array* estará sempre relacionado à uma chave de identificação. Portanto, esta estrutura de dados é também conhecida como variável indexada, vejamos a imagem abaixo:

```
int[] arr = {1, 2, 3, 4, 5};
```

index →	0	1	2	3	4
value →	1	2	3	4	5
	<code>arr[0]</code>	<code>arr[1]</code>	<code>arr[2]</code>	<code>arr[3]</code>	<code>arr[4]</code>

##### 16.7.2.3. Funções

Uma função é simplesmente um bloco de código separado que recebeu um nome. Entretanto, funções também podem receber parâmetros e/ou retornar dados. Nesse caso, você não passou nenhum dado à função, nem fez com que ela retornasse dados. No caso de nosso exercício iremos utilizar duas funções, a `conectaWifi()` e a `verificaSite()`.

##### 16.7.2.4. Constantes

Uma constante, nada mais é que um valor armazenado em memória que não tem seu valor alterado durante todo o ciclo de vida do algoritmo, recebido o valor, permanece com ele até o final de sua execução, e caso exista a tentativa de mudança desse valor, o programa exibirá um erro.

#### 16.7.3 Código fonte

```
04-ConectaRoteadorEPing
/*****
Professor Marcos Costa
Conectando a um roteador e
Fazendo um Ping com Esp32 para verificar se
o Esp32 está na Web validando dois sites.
*****/

#include <WiFi.h>
#include <ESP32Ping.h>
```

```
//Constantes que terão o dados da Rede WiFi
const char* ssid = "IoT-Senac";
const char* password = "Senac@2025";

//"Banco de Dados" com os sites para pings
const char* site[] = {"www.senactaboa.com.br", "www.google.com.br", "www.marcoscosta.eti.br"};

void setup() {
    //Iniciar a comunicação serial
    Serial.begin(115200);
    //Chamada da função para conexão com WiFi
    conectaWiFi();
    Serial.println("=====");
    //Chamada da função para verificações dos Pings
    verificaSite();
}

void loop(){
    //Não possui execução
}

void conectaWiFi() {
    Serial.print("Conectando em ");
    Serial.println(ssid);

    //Inicia a conexão de acordo com os parâmetros do roteador escolhido
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi conectado.");
    Serial.println("Endereço de IP: ");
    Serial.println(WiFi.localIP());
}

void verificaSite() {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("WiFi não está conectado. Não será possível realizar o ping.");
        return;
    }

    for (int x = 0; x < sizeof(site) / sizeof(site[0]); x++) {
        Serial.print("Pingando o site: ");
        Serial.println(String(site[x]));

        //Variável booleana que recebe True ou False do Ping realizado de até 5 segundos
        bool success = Ping.ping(site[x], 5);

        if (!success) {
            Serial.println("O Ping falhou!");
        } else {
            Serial.println("Ping no site foi realizado com sucesso!");
        }

        Serial.println("=====");
    }
}
```



O resultado aproximado da execução:

```
COM3

..
WiFi conectado.
Endereço de IP:
10.0.0.132
=====
Pingando o site: www.senactaboao.com.br
O Ping falhou!
=====
Pingando o site: www.google.com.br
Ping no site foi realizado com sucesso!
=====
Pingando o site: www.marcoscosta.eti.br
Ping no site foi realizado com sucesso!
=====
```

#### 16.8. Prática Número 1 em grupos.

Para terminarmos esta etapa, iremos realizar uma tarefa que irá demandar um trabalho em equipe, portanto montem seus times, e vocês devem realizar as etapas abaixo, levando em consideração todos os códigos que realizamos até o momento, tudo deve ser documentado em um arquivo do Microsoft Word, subir em um repositório do GitHub de um dos componentes do grupo e compartilhando no Microsoft Teams o link do mesmo. Todos os programas que serão abordados abaixo estão em nosso repositório no GitHub em: <https://github.com/marcossousa33dev/T01IOTUC15>

##### 18.8.1. Objetivo:

O objetivo deste exercício é consolidar os conhecimentos adquiridos sobre conectividade WiFi no ESP32, analisando redes disponíveis, conectando-se automaticamente à melhor rede aberta e validando a conectividade com a internet via ping, montar um documento com as especificidades abaixo e um outro ponto, vocês deverão fazer a gravação de vídeo dos projetos em funcionamento e subir o mesmo, ou em um canal do YouTube, Instagram ou LinkedIn. Neste documento, deverá ter todos os links que fizerem, para isso, se organizem para que o tempo de aula seja suficiente para tal.

##### 18.8.2. Parte 1 – Análise de Redes WiFi

Compile e carregue o primeiro programa no ESP32, observe e registre a saída no monitor serial, respondendo as questões abaixo:

- Quantas redes WiFi foram encontradas?
- Qual é a rede com o sinal mais forte?
- Quais redes possuem senha?

##### 18.8.3. Conexão Automática à Melhor Rede Aberta

Compile e carregue o segundo programa, verifique se o ESP32 conseguiu identificar a melhor rede aberta e responda:

- O ESP32 conseguiu encontrar e conectar a uma rede aberta? Se sim, qual?
- O que acontece se nenhuma rede aberta for encontrada?

##### 18.8.4. Validação de Conectividade com Ping

Compile e carregue o último programa no ESP32, verifique no monitor serial se a conexão com a rede foi estabelecida, observando os resultados dos pings para os sites listados responda:

- O ESP32 conseguiu conectar ao WiFi configurado?
- Todos os sites responderam ao ping? Se algum falhou, qual foi?

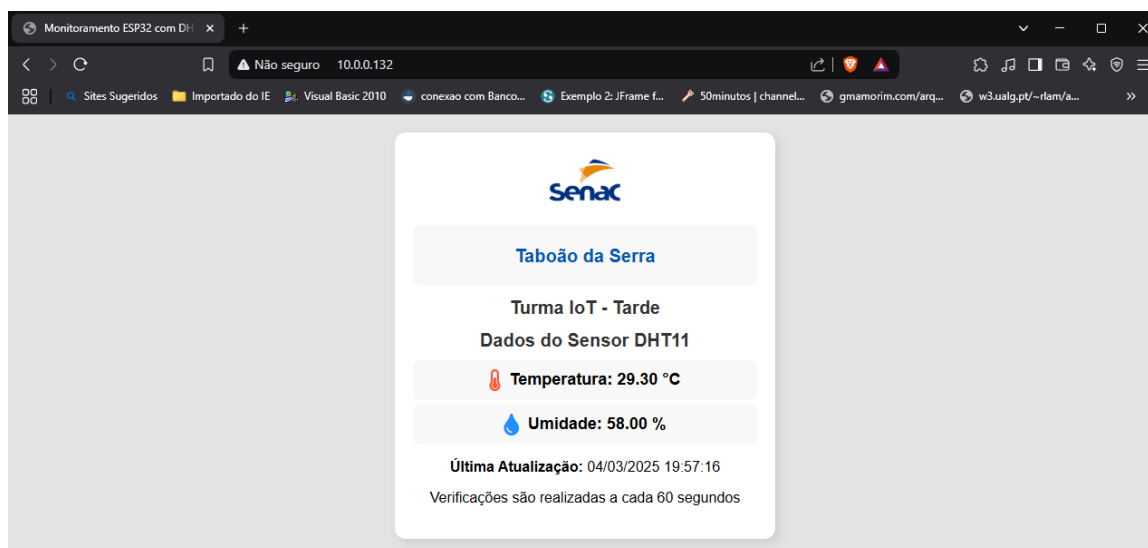
##### 18.8.5. Desafio Extra

Modifique o terceiro programa para que, caso o ESP32 não consiga conectar a uma rede WiFi específica, ele tente procurar e se conectar à melhor rede aberta disponível, realizem as alterações

no código, teste e verifique no monitor serial se o ESP32 consegue alternar entre redes caso a primeira tentativa falhe, e explique as mudanças que você fez no código, este código deverá estar no repositório que irão postar o documento que relatamos no começo, também podem colocar o código do mesmo no arquivo do Microsoft Word que criarem.

### 16.9. Implementação placa Esp32 com monitoramento

Vamos neste projeto, integrar o sensor DHT11(sensor de temperatura e umidade) e fornecer os resultados de medição em uma página HTML responsiva, que será acessada de qualquer dispositivo, podendo ser um computador, tablet ou celular, desde que esteja na mesma rede que nosso esp32.



Visualização no computador



Visualização no celular

#### 16.9.1. O que vou aprender?

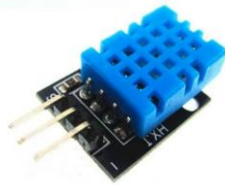
- O que é o sensor DHT11;
- Biblioteca DHT.h e time.h;
- O que é o pool.ntp.org;
- O que é o WiFiServer;
- O que é o WiFiClient;
- O que é o HTML.

#### 16.9.2. Conhecimentos prévios

##### 16.9.2.1. Bibliotecas DHT.h e time.h

O Módulo DHT11 é um Sensor de temperatura e ao mesmo tempo um Sensor de Umidade. Como o próprio nome sugere é utilizado para medir a temperatura nas escalas de 0 a 50° graus celsius e a umidade do ar nas faixas de 20 a 90%. Especialmente, vem sendo empregado no desenvolvimento de projetos eletrônicos e robóticos através de placas microcontroladoras, entre estas, Arduino, ARM, AVR, PIC e etc.

O Sensor de Umidade e Temperatura DHT 11 na prática detecta a umidade e a temperatura, enviando estas informações para a placa microcontroladora, que deve estar programada para realizar alguma ação quando atingida determinada temperatura ou umidade. Um exemplo a ser destacado de sua utilização é por meio da placa Arduino, onde é possível programar a placa microcontroladora para ligar, por exemplo, o ar-condicionado quando o ambiente atingir determinada temperatura, ou ligar a função desumidificar quando atingir determinada umidade. (Para tal função pode ser necessário acrescentar mais acessórios a placa microcontroladora).



Sensor DHT11

##### 16.9.2.2. Bibliotecas DHT.h e time.h

A biblioteca DHT.h é amplamente utilizada em projetos com microcontroladores, como Arduino e ESP8266/ESP32, para a leitura de sensores de temperatura e umidade da família DHT, incluindo os modelos DHT11 e DHT22. Ela fornece funções que facilitam a comunicação com esses sensores, permitindo a obtenção de dados ambientais de forma simples e eficiente. Para utilizá-la, é necessário instalar a biblioteca correspondente na IDE do Arduino ou em outros ambientes compatíveis.

Já a biblioteca time.h é uma biblioteca padrão da linguagem C e C++, usada para manipulação de tempo e datas. Ela permite acessar a data e hora do sistema, calcular diferenças entre tempos, medir intervalos de execução e formatar a exibição de informações temporais. Em microcontroladores, essa biblioteca pode ser utilizada em conjunto com protocolos como o NTP (Network Time Protocol) para obter a hora exata a partir da internet.

Enquanto a DHT.h é específica para sensores de temperatura e umidade, a time.h é uma biblioteca genérica que lida com funções de tempo e data. Ambas são úteis em diferentes contextos e podem ser combinadas em projetos que envolvem monitoramento ambiental e registro de dados com marcação temporal.

##### 16.9.2.3. O que é pool.ntp.org.

O pool.ntp.org é um serviço global de servidores de tempo baseado no protocolo NTP (*Network Time Protocol*). Ele fornece sincronização precisa de data e hora para dispositivos conectados à internet, como servidores, computadores e microcontroladores, garantindo que mantenham um relógio atualizado.

Esse serviço funciona como um conjunto distribuído de servidores NTP, onde os dispositivos solicitam a hora exata e recebem uma resposta de um servidor disponível no momento. O pool é mantido por voluntários e distribuído geograficamente para reduzir a latência e melhorar a precisão. Quando um dispositivo consulta pool.ntp.org, ele é direcionado automaticamente para um servidor próximo e funcional.

O serviço é amplamente utilizado em sistemas operacionais, servidores e dispositivos IoT, como ESP32, para garantir a precisão do relógio interno, especialmente em sistemas que não

possuem um RTC (*Real-Time Clock*) integrado. Para obter maior precisão regional, é possível utilizar subdomínios como `br.pool.ntp.org` para o Brasil ou `us.pool.ntp.org` para os Estados Unidos.

#### 16.9.2.4. O que é o WiFiServer.

O WiFiServer é uma classe da biblioteca WiFi do ESP32 usada para criar um servidor TCP que permite a comunicação via Wi-Fi. Ele é comumente utilizado para criar servidores web, permitindo que dispositivos como ESP32 recebam conexões de clientes, como navegadores ou aplicativos de rede.

Essa classe funciona escutando conexões em uma porta específica, geralmente a porta 80 para servidores web. Quando um cliente se conecta, o WiFiServer aceita a conexão e cria um objeto WiFiClient para gerenciar a comunicação. Isso permite enviar e receber dados, como páginas HTML ou comandos em projetos IoT, que é nosso caso.

O WiFiServer é muito utilizado para criar interfaces de controle remoto, dashboards para monitoramento e APIs locais para automação residencial.

#### 16.9.2.5. O que é o WiFiClient.

O WiFiClient é uma classe utilizada em dispositivos como o ESP8266 ou ESP32 para facilitar a comunicação via rede Wi-Fi, permitindo que o dispositivo atue como um cliente em uma rede TCP/IP. Ele pode ser usado para se conectar a servidores e enviar ou receber dados pela rede. A principal função do WiFiClient é criar uma conexão TCP com um servidor, seja para enviar ou receber dados. Por exemplo, quando você deseja acessar uma página web, se conectar a um servidor de banco de dados via HTTP.

O WiFiClient atua como um cliente em uma comunicação de rede, estabelecendo uma conexão com um servidor remoto através de um endereço IP e uma porta. Isso permite que o dispositivo envie ou receba dados de forma simples.

#### 16.9.2.5. O que é o HTML.

HTML, ou *HyperText Markup Language*, é a linguagem de marcação padrão usada para criar e estruturar páginas na web. Ele permite que você organize o conteúdo de uma página, como texto, imagens, links e outros elementos multimídia, de uma maneira que os navegadores possam entender e exibir corretamente.

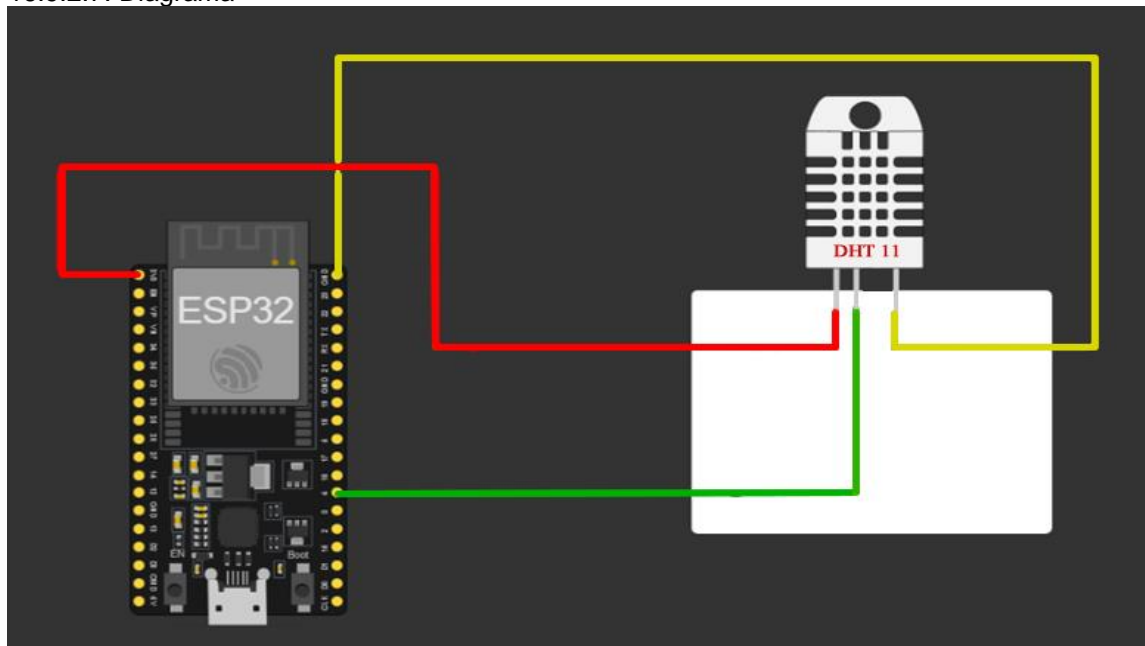
Quando usamos HTML no ESP32, geralmente estamos criando um servidor web que permite interagir com o dispositivo através de uma interface web. Isso é útil para monitorar sensores, controlar dispositivos conectados ao ESP32 e exibir informações em tempo real, a seguir estão alguns pontos importantes sobre o uso de HTML no ESP32:

- Servidor Web: O ESP32 pode ser configurado como um servidor web que responde a solicitações HTTP. Isso permite que você acesse páginas web hospedadas no ESP32 usando um navegador, que será nosso caso.
- Conteúdo HTML: As páginas web servidas pelo ESP32 são escritas em HTML, que pode incluir elementos como texto, imagens, formulários e botões para interagir com o dispositivo, também teremos em nosso projeto.
- Integração com Código: O HTML pode ser integrado com código JavaScript e CSS para criar interfaces mais dinâmicas e estilizadas. Além disso, o código do ESP32 pode ser escrito para responder a eventos gerados na página web, como cliques de botões ou envio de formulários.

#### 16.9.2.6. Material necessário



### 16.9.2.7. Diagrama



### 16.9.2.8. Código fonte do projeto

```
05-MonitorHTMLEsp32DHT11 §
/*
  Professor Marcos Costa
  Verifica as conexões disponíveis e se há necessidade
  de informar senha.
*/

// Inclui a biblioteca DHT para interagir com o sensor DHT11
#include "DHT.h"

// Inclui a biblioteca WiFi para permitir a comunicação Wi-Fi
#include <WiFi.h>

// Inclui a biblioteca time para permitir a captura de data e hora
#include "time.h"

#define DHTPIN 4 //Define o pino de conexão do sensor DHT
#define DHTTYPE DHT11 // Define o tipo de sensor como DHT11

//Constantes que terão o dados da Rede WiFi
const char* ssid = "IoT-Senac";
const char* password = "Senac@2025";

const char* ntpServer = "pool.ntp.org"; // Servidor usado para sincronizar relógio de dispositivos conectados a internet

const long  gmtoffset_sec = -10800; // Ajuste para seu fuso horário (exemplo: -3h para Brasil) neste caso em segundos
const int   daylightOffset_sec = 0; // Ajuste para horário de verão, se necessário

WiFiServer server(80); // Criação do servidor Web que "escuta" a porta 80 (HTTP)
DHT dht(DHTPIN, DHTTYPE); // Cria um objeto DHT com o pino e tipo especificados

// Variáveis para armazenar Umidade e Temperatura ambiente
float umidade, temperatura;

void setup(void) {
  Serial.begin(115200); // Inicia a comunicação serial com uma taxa de transmissão de 115200 bps
  Serial.print("Conectando-se a: ");
  Serial.println(ssid);

  WiFi.begin(ssid, password); // Inicia a conexão Wi-Fi com as credenciais fornecidas

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi conectada.");
  Serial.println("Endereço de IP de acesso de qualquer browser: ");
  Serial.println(WiFi.localIP());

  configTime(gmtoffset_sec, daylightOffset_sec, ntpServer); // Configura o relógio NTP
  dht.begin(); // Inicializa o sensor DHT
  delay(1000);
  server.begin();
}
```



```
String getDateTime() {
    /*
     * Retornar a data e a hora atuais como uma string
     * formatada no formato "DD/MM/YYYY HH:MM:SS"
     */
    struct tm timeinfo;
    //O &timeinfo é um ponteiro para a estrutura tm,
    //onde a data e hora serão armazenadas.
    if (!getLocalTime(&timeinfo)) {
        Serial.println("Falha ao obter o tempo");
        return "Erro ao obter hora";
    }

    char buffer[30];

    strftime(buffer, sizeof(buffer), "%d/%m/%Y %H:%M:%S", &timeinfo);
    return String(buffer);
}

void loop(void) {
    temperatura = dht.readTemperature(); // Lê o valor da temperatura
    umidade = dht.readHumidity();         // Lê o valor da umidade do ar

    WiFiClient client = server.available();

    if (client) {
        Serial.println("Nova requisição.");
        String currentLine = "";

        //Enquanto estiver conectado
        while (client.connected()) {
            if (client.available()) {
                char c = client.read();
                Serial.write(c);
                if (c == '\n') {
                    if (currentLine.length() == 0) {
                        String dateTime = getDateTime(); // Obtém a data e hora atual
                        // Cabeçalhos HTTP
                        client.println("HTTP/1.1 200 OK");
                        client.println("Content-type:text/html");
                        client.println();
                        client.println("<!DOCTYPE html>");
                        client.println("<html lang='pt-BR'>");
                        client.println("<head>");
                        client.println("<meta charset='UTF-8'>");
                        client.println("<meta name='viewport' content='width=device-width, initial-scale=1.0'>");
                        client.println("<meta http-equiv='refresh' content='60'>"); // Atualiza a cada 60 segundos
                        client.println("<title>Monitoramento ESP32 com DHT 11</title>");
                        // Importação do Font Awesome para ícones
                        client.print("<link href='https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css'>");
                        client.println("<rel='stylesheet'>");
                        client.println("<style>");
                        client.print("body { font-family: Arial, sans-serif; text-align: center; background-color: #e3e3e3;");
                        client.println("margin: 0; padding: 0; }");

                        client.println(".container { max-width: 90%; width: 380px; margin: 20px auto; padding: 20px; background: white;");
                        client.println("box-shadow: 4px 4px 10px rgba(0,0,0,0.1); border-radius: 12px; }");

                        client.println(".logo { width: 50%; max-width: 120px; display: block; margin: 0 auto 15px; }");
                        client.println("h1 { font-size: 20px; color: #333; margin-bottom: 10px; }");
                        client.println(".data-section { display: flex; align-items: center; justify-content: center; margin: 10px 0; }");

                        client.println(".icon { font-size: 24px; margin-right: 10px; }");
                        client.println(".temp-icon { color: #ff5733; }");
                        client.println(".umidade-icon { color: #1e90ff; }");

                        client.println(".info-box { background: #f8f8f8; padding: 10px; border-radius: 8px; margin: 5px 0;");
                        client.println("display: flex; align-items: center; justify-content: center; font-size: 18px; font-weight: bold; }");
                        client.println("</style>");

                        client.println("</head>");
                        client.println("<body>");

                        client.println("<div class='container'>");

                        // Logo do Senac
                        client.print("<img src='https://i0.wp.com/centralblogs.com.br/wp-content/uploads/2020/09/'>");
                        client.println("senac-logo.png?w=696&ssl=1' alt='Logo Senac' class='logo'>");
                        client.println("<div class='info-box'>");
                        client.println("<h1 style='color: #0056b3;'>Taboão da Serra</h1>");
                        client.println("</div>");
                        client.println("<h1>Turma IoT - Tarde</h1>");
                        client.println("<h1>Dados do Sensor DHT11</h1>");

                        // Temperatura
                        client.println("<div class='info-box'>");
                        client.println("<i class='fas fa-thermometer-half icon temp-icon'></i>");
                        client.print("Temperatura: ");
                        client.print(temperatura);
                        client.println(" °C</div>");

                        // Umidade
                        client.println("<div class='info-box'>");
                        client.println("<i class='fas fa-tint icon umidade-icon'></i>");
```

```

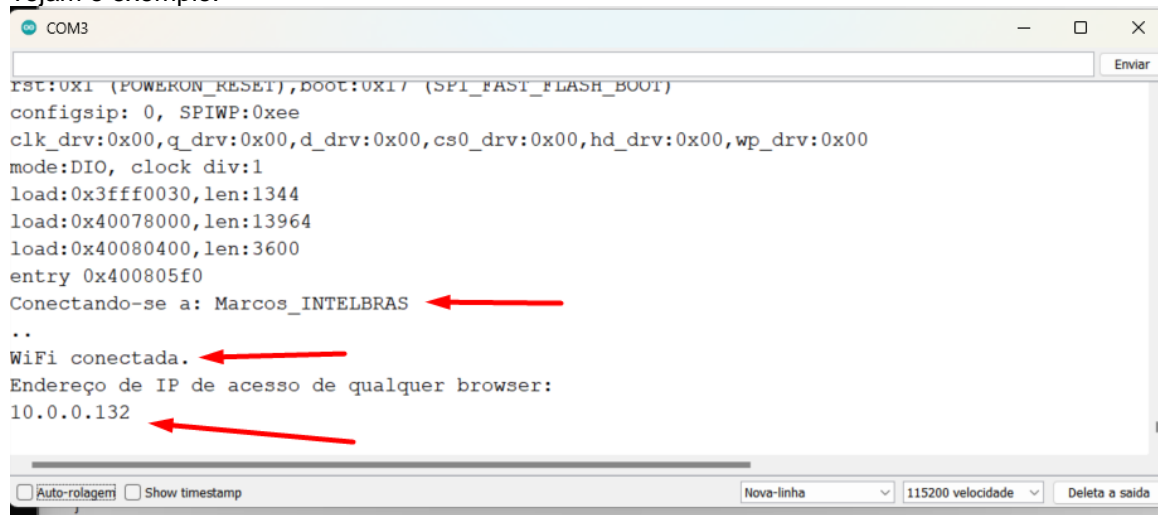
client.print("Umidade: ");
client.print(umidade);
client.println(" %</div>");

// Última atualização
client.println("<p><strong>Última Atualização:</strong> " + dateTime + "</p>");
client.println("<p>Verificações são realizadas a cada 60 segundos</p>");
client.println("</div>"); // Fecha container
client.println("</body>");
client.println("</html>");

break;
} else {
    currentLine = "";
}
} else if (c != '\r') {
    currentLine += c;
}
}
}
client.stop();
}
}

```

Após esta implementação, compilamos o projeto e verificamos que na saída serial da IDE do Arduino será exibido o IP do nosso ESP32, que neste momento fará o papel de um *WebServer*, vejam o exemplo:



```

COM3
Enviar
rst:0x1 (POWERON_RESET),boot:0x1 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13964
load:0x40080400,len:3600
entry 0x400805f0
Conectando-se a: Marcos_INTELBRAS
..
WiFi conectada.
Endereço de IP de acesso de qualquer browser:
10.0.0.132

```

Após isso, iremos abrir o browser em um dispositivo que esteja na mesma rede do ESP32 e passaremos este IP, após alguns segundos a página é carregada com as informações de umidade, temperatura e horário de verificação.

