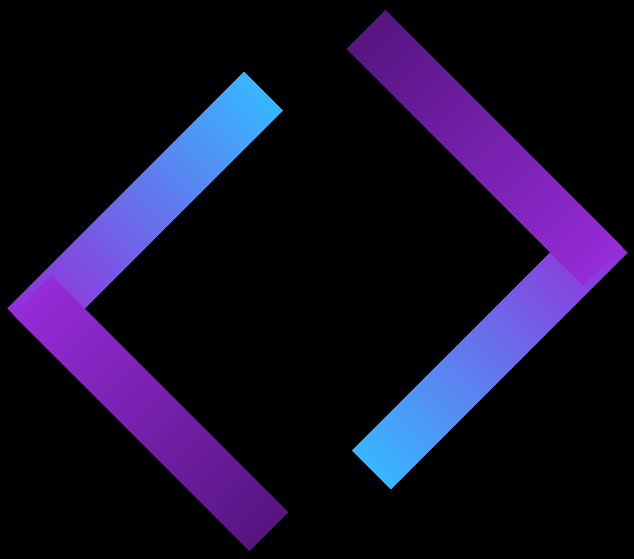
APOSTILA PET C3

Arduino

Conhecimento retido
é conhecimento
perdido.

PET C3
Ciências Computacionais



PETCode

Arduino

PET C3
Ciências Computacionais

PET CC Ciências Computacionais

PETCode

PETCode

Indíce

PETCode

Indíce

Introdução

Nesta apostila, você encontrará os principais conceitos para começar a desenvolver projetos com Arduino, desde os fundamentos da eletrônica até a programação e simulação de circuitos. Serão abordados temas essenciais como "O que é Arduino?", "Fundamentos de eletrônica" e "Lógica de programação em C/C++", além de tópicos mais avançados, como "Arquitetura do microcontrolador" e "Projetos práticos".
O objetivo desta apostila é auxiliar na compreensão dos conceitos fundamentais da eletrônica aplicados ao Arduíno, além de introduzir a programação em C/C++, a instalação e configuração das ferramentas necessárias e o desenvolvimento de projetos interativos.

O que é Arduino?

O básico

Começando do básico, o que é Arduino? Ele é uma plataforma de prototipagem eletrônica que une hardware (a parte física) e software (o código), permitindo criar modelos de projetos eletrônicos. Pense nele como uma ponte entre o mundo físico e o digital. Por exemplo, imagine uma lâmpada que pode ser acesa pelo seu celular. O Arduino seria o responsável por conectar a lâmpada (hardware) ao aplicativo (software), permitindo que você controle a luz de forma programada.

A grande vantagem do Arduino é a simplicidade, ele torna a eletrônica mais acessível, permitindo que qualquer pessoa, mesmo sem experiência, consiga desenvolver projetos. Diferente de outros microcontroladores, o Arduíno vem pronto para o uso, não há necessidade de adicionar nenhum componente na placa, basta conectá-lo ao computador via USB, escrever um código e enviar para a placa.

História

Em torno do ano de 2005, na Itália, um grupo de pesquisadores -Massimo Banzi, David Cuartielles, David Mellis, Gianluca Martino e Tom Igoe- queria desenvolver uma ferramenta eletrônica que fosse acessível e fácil de aprender. O objetivo era permitir que estudantes de diversas áreas tivessem contato com eletrônica e programação, sem precisar de conhecimentos avançados.

No mesmo ano surgiu o primeiro Arduíno, usando um microcontrolador ATmega8. O projeto teve grande sucesso, vendeu mais de 50 mil unidades e desde então a plataforma cresceu e novos modelos de Arduinos foram criados, falaremos mais sobre no ítem 5.

Hoje, o Arduino é utilizado no mundo inteiro, tanto para aprendizado quanto em projetos profissionais. Sua grande vantagem é a versatilidade: ele não se limita a uma única área de aplicação, sendo amplamente usado em automação, robótica, Internet das Coisas (IoT) e até mesmo em projetos artísticos e científicos.

Fundamentos de eletrônica

Componentes

Mesmo que o Arduino venha pronto, sem necessidade de acrescentar nada à placa, é essencial conhecer seus componentes e suas funções:

- Indutores: Armazenam energia em forma de campo magnético e são usados para filtrar sinais ou controlar variações de corrente.
- Capacitores: Armazenam energia em forma de campo elétrico. Servem para filtrar ruídos, suavizar sinais ou manter cargas temporárias.

- Resistores: Limitam a corrente elétrica para proteger outros componentes.
- LEDs: Pequenas luzes que acendem quando a corrente passa. Ótimos para testes e sinalizações.
- Botões: Permitem enviar comandos ao Arduino quando pressionados.
- Sensores: Captam informações do ambiente, como luz, temperatura, distância ou movimento.
- Protoboard: Uma placa usada para montar circuitos sem solda. Ideal para testes e aprendizado.
- Potenciômetro: Um tipo de resistor ajustável, usado para controlar a intensidade de sinais, como volume ou brilho.

Alguns conceitos importantes são Tensão, Corrente e Resistência:

- Tensão (voltagem): É a força que empurra os elétrons em um circuito, fazendo a corrente elétrica circular. É medida em volts (V).
-
- Corrente elétrica: É o fluxo de elétrons que percorre o circuito. Ela é medida em amperes (A).

Resistência: É a oposição à passagem da corrente elétrica. Componentes como resistores existem justamente para limitar a corrente. A resistência é medida em ohms (Ω).

Ligações e associações em série e paralelo

Os componentes estudados acima se conectam entre si, e essas conexões podem ser feitas em série ou em paralelo. O que define o tipo de ligação são os pontos em comum entre os componentes. Se eles tiverem apenas um ponto em comum, teremos uma ligação em série. Se tiverem dois pontos em comum, a ligação será em paralelo.

Pense assim: cada componente é uma pessoa, e seus braços são as partes responsáveis por fazer a conexão com outras pessoas, formando um circuito. Considere três pessoas: a Pessoa 1 dá uma mão para cada uma das outras duas, Pessoa 2 e Pessoa 3. Elas passam a ter apenas um ponto em comum entre si, ficando lado a lado. Nesse caso, temos uma ligação em série. Aqui, a corrente elétrica que circula por elas será a mesma, mas a tensão se divide conforme o "tamanho" (ou resistência) de cada uma.

Agora imagine que a Pessoa 1 dá as duas mãos para a Pessoa 2. Nesse caso, elas têm dois pontos em comum e ficam de frente uma para a outra. Isso representa uma ligação em paralelo. Aqui, a tensão é a mesma para ambas, e é a corrente que se divide conforme o tamanho das pessoas/resistência.

Diferente das pessoas, um componente de circuito pode "dar a mesma mão para mais de uma pessoa", ou seja, se ligar a vários outros componentes ao mesmo tempo. Assim, em um circuito real, é comum termos uma combinação de ligações em série e em paralelo. Esses pontos de conexão entre os componentes são usados para calcular a chamada Diferença de Potencial (ddp), mais conhecida como tensão.

Leis da eletricidade

Dentro do estudo da eletricidade, existem algumas leis importantes que ajudam a entender o funcionamento dos circuitos. As principais para essa apostila são a Primeira Lei de Ohm e as duas Leis de Kirchhoff.

Primeira lei de OHM

Essa lei diz que ***"a corrente que circula em um circuito é diretamente proporcional à diferença de potencial elétrico aplicada e inversamente proporcional à resistência"***. Mas calma, vamos por partes:

Um circuito tem três grandezas principais: tensão (V), corrente (I) e resistência (R). A Primeira Lei de Ohm estabelece a seguinte relação entre elas:

$$V = R \times I$$

Ou seja, a tensão (V) entre dois pontos do circuito é igual à multiplicação da resistência (R) entre esses pontos pela corrente (I) que passa por eles.

Essa fórmula é uma das mais usadas na eletrônica e vai aparecer bastante ao longo dos projetos.

Lei de Kirchhoff das correntes

Essa lei trata das correntes que entram e saem de um ponto (ou nó) em um circuito. Ela diz que:

"A soma das correntes que entram em um nó é igual à soma das correntes que saem."

É como imaginar água chegando e saindo de um cano: o que entra tem que sair. Nenhuma corrente se perde nesse ponto.

Lei de Kirchhoff das tensões

Essa lei se refere à tensão em um circuito fechado. Ela afirma que:

"A soma das tensões em um caminho fechado de um circuito é sempre zero."

Isso quer dizer que, ao dar uma "volta completa" pelo circuito, somando todas as quedas e subidas de tensão, o total sempre será zero. Essa ideia é muito útil para analisar circuitos mais complexos e entender como a energia elétrica é distribuída.

Instalação e Ferramentas

O que é a Arduino IDE?

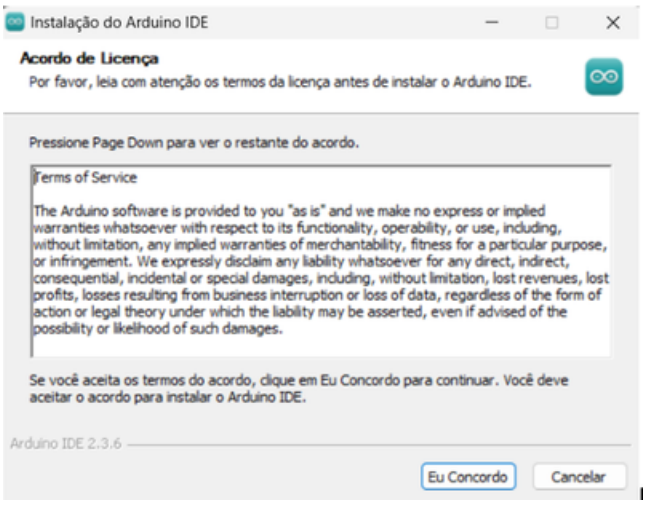
O Arduino IDE é o ambiente de desenvolvimento oficial usado para escrever, compilar e enviar códigos para placas Arduíno. Em outras palavras, é o programa que usamos no computador para programar o Arduíno.

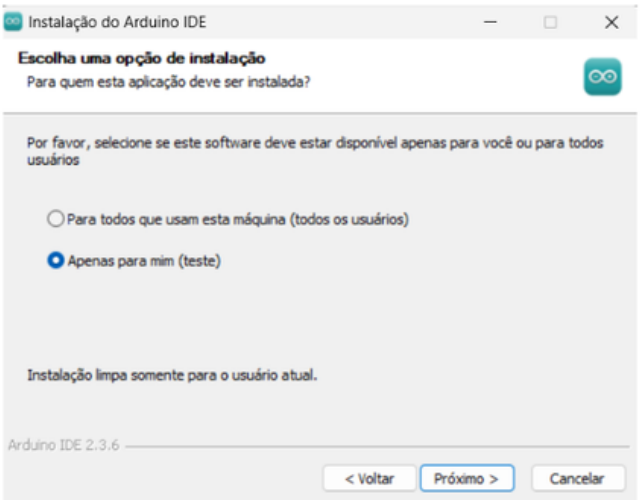
Para que possamos dar continuidade na apostila, iremos utilizar o software Arduino IDE para realizarmos práticas ao final. Aqui está o passo a passo para sua instalação

Instalação

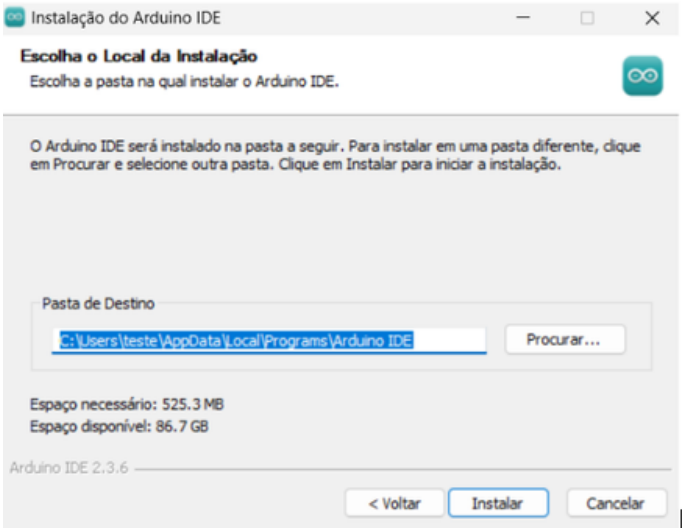
Entrar no site: https://www.arduino.cc/ensoftware/, onde aparecerá a seguinte tela:

Nela, iremos realizar o download do arquivo. Ao ser executado, aparecerá a seguinte tela:

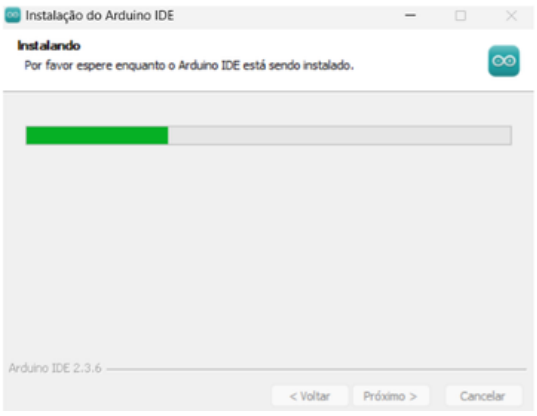




Selecione em qual usuário gostaria de instalar o programa e depois, em próximo

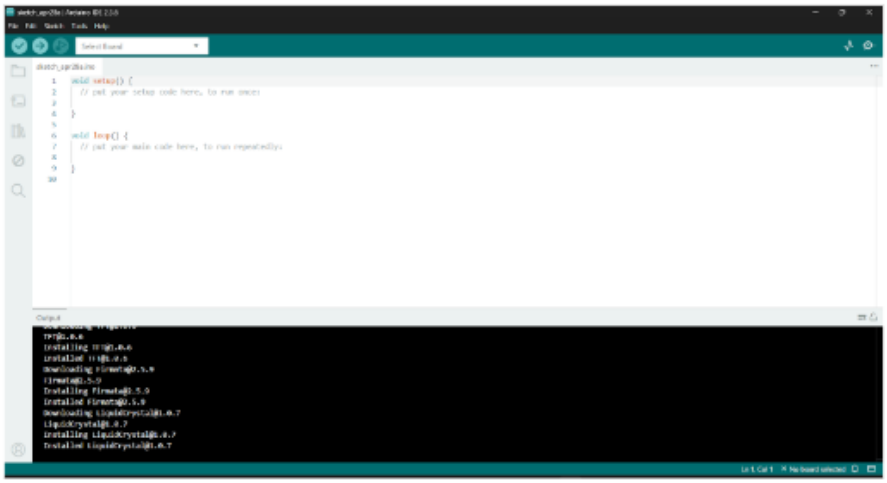


Confirme o local de instalação e clique em instalar, onde aparecerá a seguinte tela;



Após a instalação ser concluída, aparecerá a seguinte tela;

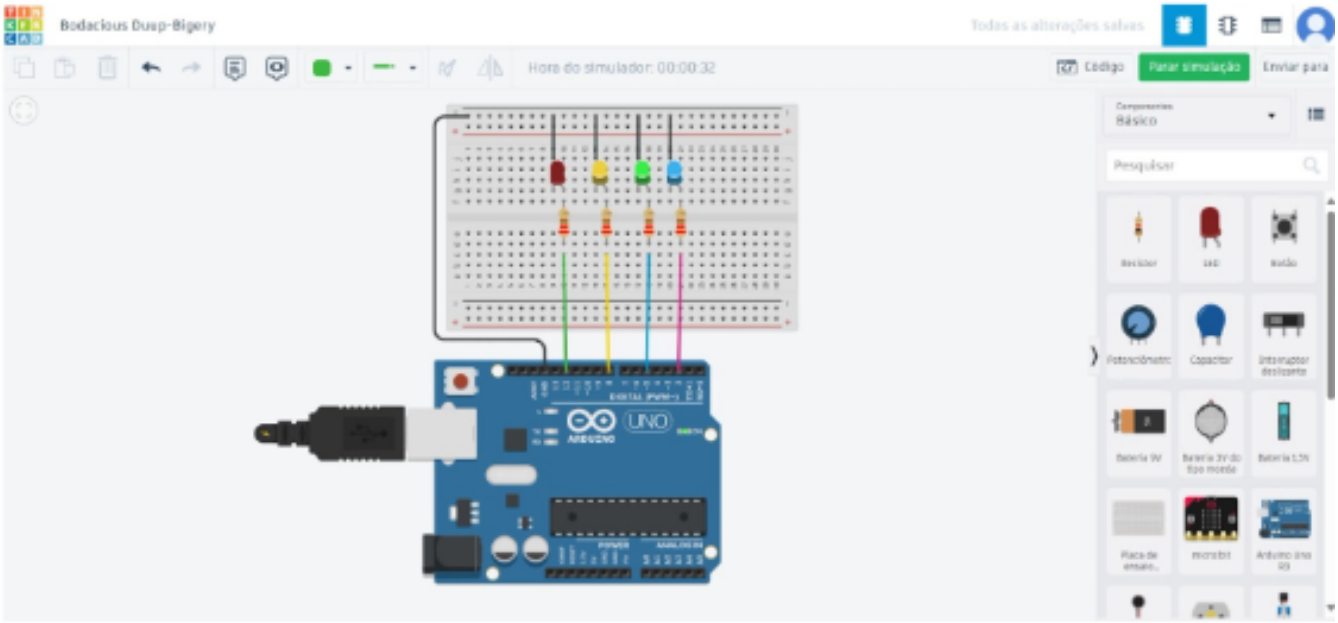
Pronto, agora que a ferramenta foi instalada, basta clicar em concluir e manter a caixa de seleção que o programa irá abrir;



Além do Arduino IDE, também existem diversas ferramentas que nos ajudam a praticar e implementar nossos projetos, até mesmo de forma simulada, como por exemplo o Tinkercad e o SimulIDE.

O que é o Tinkercad?

Tinkercad é uma plataforma online, gratuita, desenvolvida pela Autodesk. Além de modelagem 3D, ela oferece o Tinkercad Circuits, onde nele é possível montar circuitos eletrônicos virtuais, Programar microcontroladores como Arduino UNO usando blocos ou código C/C++ e visualizar simulações de componentes como LEDs, motores e sensores.

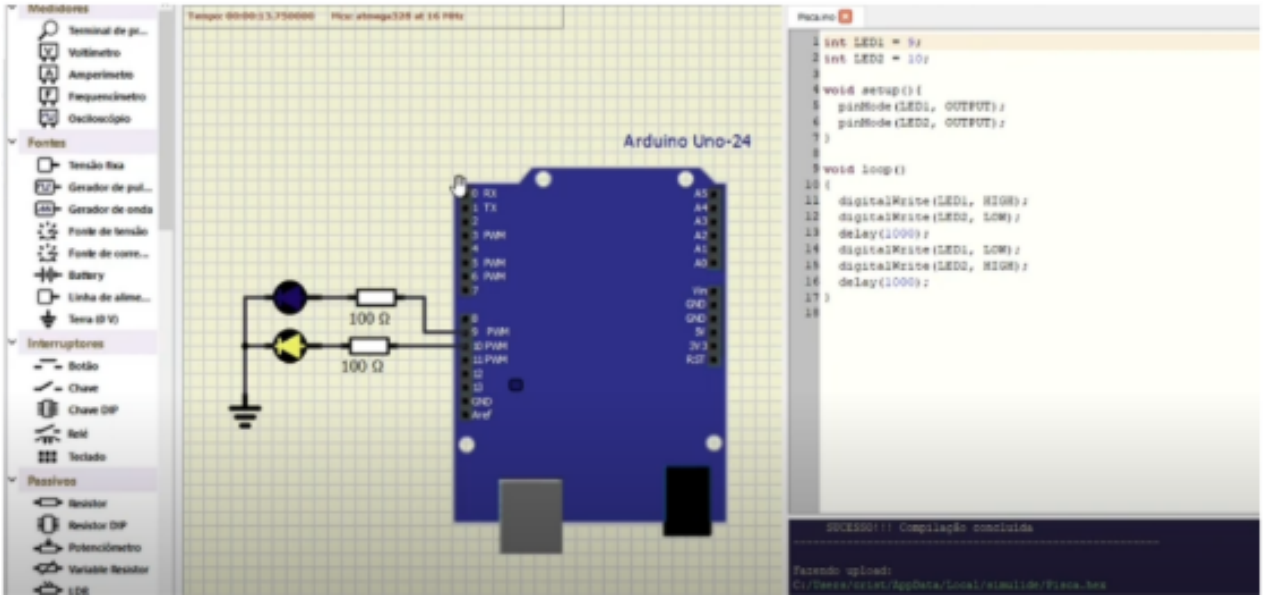


Exemplo de código sendo simulado para o controle de LEDs

O que é o SimulIDE?

O SimulIDE é um simulador de circuitos elétricos e eletrônicos, de instalação local, focado em simplicidade e simulação em tempo real, onde é possível simular circuitos analógicos (Um circuito analógico trabalha com sinais contínuos, ou seja, a tensão ou a corrente podem variar de forma suave dentro de um intervalo.) e digitais (Um circuito digital trabalha com sinais discretos, geralmente representados por dois estados: ligado ou desligado, 1 ou 0.), simular microcontroladores como o Arduino e ESP32, carregar programas em C/C++ compilados diretamente no microcontrolador simulado e visualizar a execução passo a passo de programas.

Ele possui certas vantagens em relação ao seu uso, como por exemplo, ter uma maior proximidade com a realidade nos testes, ter uma simulação de sinais elétricos detalhados das tensões, correntes, etc. e funcionar off-line.



Exemplo de código sendo simulado para o controle de LEDs

Em resumo…

Em resumo, enquanto o Tinkercad e SimulIDE são simuladores que imitam o funcionamento de um circuito real no computador, a Arduino IDE é focada apenas na programação real da placa.

No Tinkercad e no SimulIDE, você pode montar virtualmente circuitos com LEDs, motores, sensores e ver se eles funcionam antes mesmo de ter os componentes físicos. Isso é excelente para quem está começando ou para testar ideias sem risco de danificar nada.

Já no Arduino IDE, você precisa ter o hardware físico: a placa Arduino, fios, resistores, LEDs, etc. O código que você escreve será realmente carregado para o Arduino, e o circuito precisa estar montado corretamente no mundo real para funcionar.

Arquitetura do Microcontrolador

O que é arquitetura de um microcontrolador?

A arquitetura de um microcontrolador refere-se à organização interna e à forma como seus componentes funcionais são estruturados e interconectados. Isso inclui o modo como ele processa informações, acessa a memória e se comunica com periféricos.

Componentes do microcontrolador

CPU (Unidade Central de Processamento):

É o cérebro de qualquer sistema computacional, é o responsável por executar instruções dadas pelo usuário, processamento/decodificação de dados e controlar as operações internas do sistema. É com ele que será feita a comunicação entre hardware e software.

EX:A CPU é como um maestro em uma orquestra. Ela não faz tudo sozinha, mas coordena todos os "músicos" – como a memória, a placa de vídeo e outros componentes – para que tudo funcione junto, no ritmo certo. Sem o maestro, a música não sai como deveria. Sem a CPU, o computador também não funciona direito.

Memoria ROM (Read Only memory):

É uma memória de leitura não volátil que armazena instruções ou dados que não devem ser apagados com facilidade, ou seja, mantém a memória mesmo que o dispositivo esteja desligado, sendo utilizada para armazenar o firmware e pequenos softwares. No caso do arduino, a memória rom tem um tipo específico

geralmente sendo Flask que tem como características de ser regravável, gravada em blocos, além de ter uma rápida leitura com escritura moderada.

EX: UM livro de receitas antigo. Esse livro tem um conteúdo fixo que nunca muda, como as instruções para preparar um prato específico. Quando você precisa fazer a receita, é só seguir as instruções no livro, mas você não pode alterá-las.

Memória RAM (Random Access Memory):

É uma memória que por ter acesso aleatório, é volátil, assim sendo, tem uma rápida modelagem, podendo ser apagada e reescrita de constante. A RAM é usada para armazenar temporariamente dados e instruções enquanto o processador (CPU) está executando um programa. No arduíno a SRAM (Static Random Access Memory) é responsável por esse processo, sendo a principal responsável por armazenar as variáveis enquanto o código principal roda tendo uma característica de manter os dados enquanto estiver alimentada, sem refresh constante.

EX: Uma mesa de trabalho cheia de materiais temporários. Quando você está montando algo, você coloca as ferramentas, peças e documentos sobre a mesa para trabalhar. Porém, assim que termina o trabalho, você limpa a mesa e coloca tudo de volta no seu lugar

Memória EEPROM:

A EEPROM (Electrically Erasable Programmable Read-Only Memory) é uma memória interna não volátil, ou seja, não perde os dados quando a energia é desligada. No Arduino UNO, ela possui 1 KB de capacidade e é usada para armazenar dados que precisam ser mantidos entre reinicializações, como configurações de usuário, parâmetros de calibração, pontuações de jogos, entre outros. Ela pode ser lida e escrita diretamente por funções específicas da biblioteca EEPROM.

EX: Um quadro de avisos cheio de post-its. Você pode escrever informações nos post-its, remover e substituir sempre que necessário, e mesmo que a sala fique sem luz, os post-its continuam grudados no quadro.

Unidades de Entrada e Saída (Input/Output):

Unidades de entrada e saída são os pinos que permitem o Arduino interagir com o mundo real.Elas captam sinais do ambiente (entradas) e acionam dispositivos (saídas), tornando o microcontrolador útil em aplicações práticas como robótica, automação e IoT.

Pinos input são utilizados para receber informações do ambiente externo, como sensores de luz, temperatura, movimento e outros, além de botões e potenciômetros.

Pinos output são responsáveis pela comunicação com outros componentes do circuito como motores, LEDs, relés e displays.

Dentre os pinos, existem aqueles que têm funções extras como o PWM (Pulse Width Modulation) que permite saída analógica simulada, o ADC (Conversor Analógico-Digital) que converte sinal analógico em número digital, e o os I²C, SPI, UART no qual fazem protocolos de comunicação serial (usados em sensores, módulos etc.).

Temporizadores (Timers):

Os temporizadores são periféricos internos que permitem ao microcontrolador medir intervalos de tempo com precisão. No Arduino, existem três temporizadores principais: Timer0, Timer1 e Timer2. Cada um deles pode operar de forma independente e é capaz de gerar eventos após um determinado tempo, controlar sinais de saída, criar delays precisos e gerar sinais PWM (Pulse Width Modulation), utilizados por exemplo para controlar o brilho de LEDs ou a velocidade de motores. O Timer0, por exemplo, é usado internamente para a função millis(), que conta o tempo de execução do programa.

Conversor analogico digital:

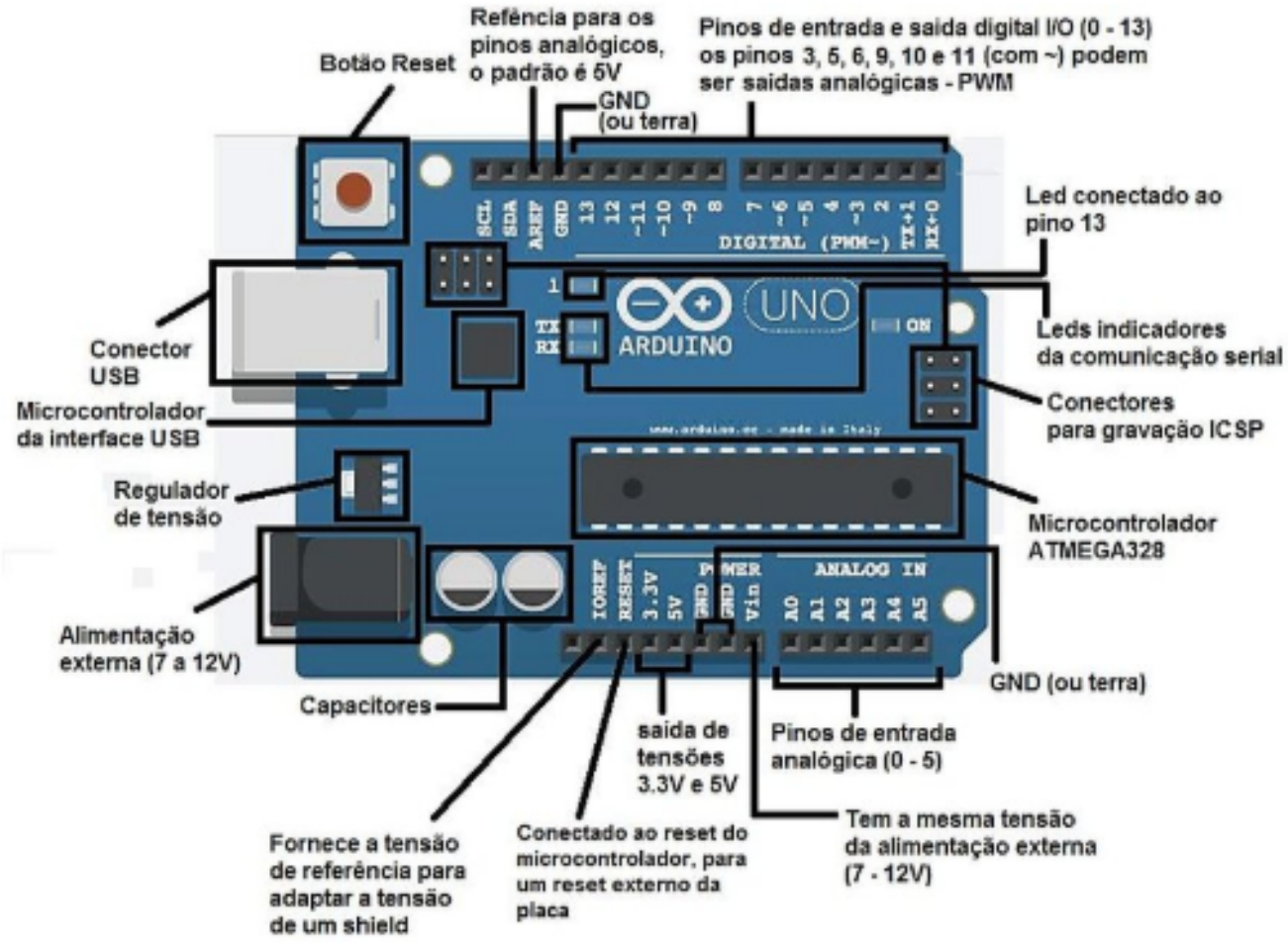
O ADC (Analog-to-Digital Converter) permite que o microcontrolador leia sinais analógicos (como variações de tensão de sensores) e os converta em valores digitais compreensíveis. No Arduino UNO, há 6 canais analógicos (A0 a A5), cada um capaz de ler tensões entre 0V e 5V e transformá-las em valores de 0 a 1023 (resolução de 10 bits). Isso é essencial para sensores como os de temperatura, luminosidade ou potenciômetros, pois seus sinais variam continuamente.

Comunicação Serial:

A comunicação seria via USART (Universal Synchronous/Asynchronous Receiver/Transmitter) permite que o Arduino troque dados com outros dispositivos, como computadores, módulos Bluetooth ou GPS. Essa comunicação ocorre usando apenas dois fios: um para enviar (TX) e outro para receber (RX). No Arduino UNO, essa interface é fundamental para a função Serial, usada para debugar e enviar dados ao monitor serial da IDE.

Principais tipos de placas e suas aplicações.

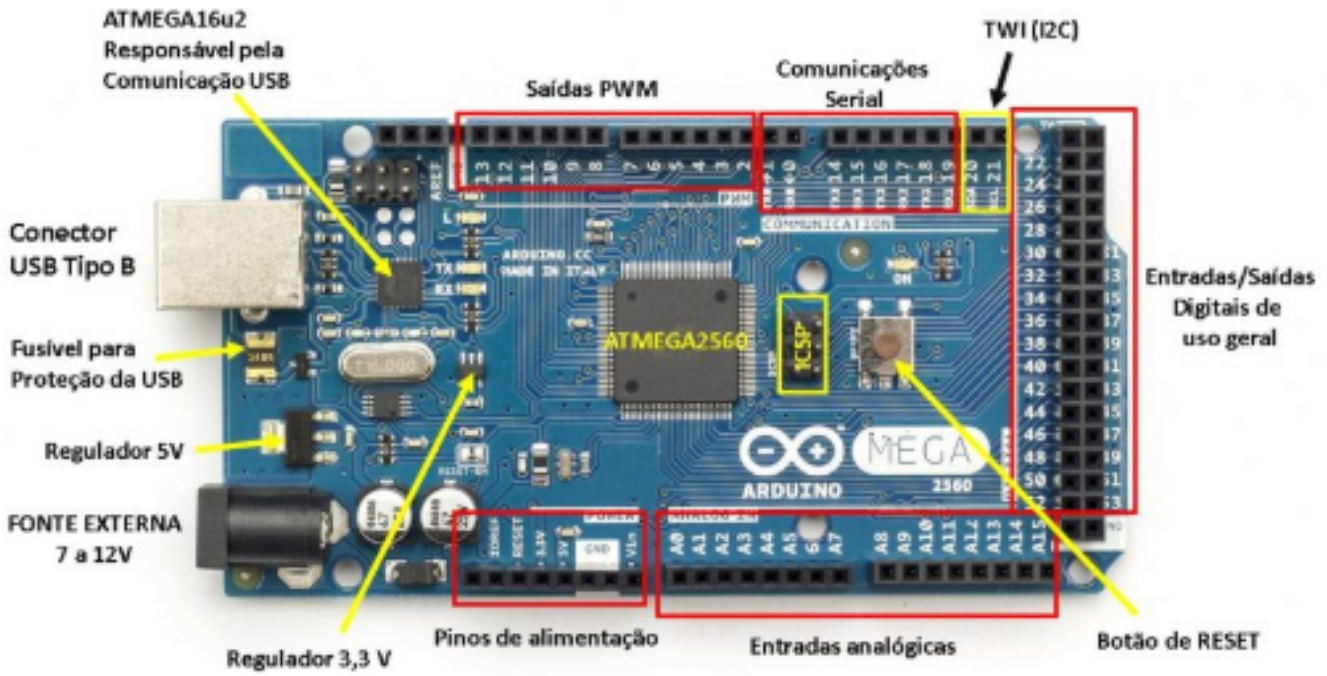
Placa arduino Uno:



- Microcontrolador: ATmega328P
- Pinos Digitais: 14 (6 com PWM)
- Entradas Analógicas: 6
- Memória Flash: 32 KB
- Comunicação: USB, UART, I2C, SPI
- Alimentação: 5V via USB ou fonte externa
- Uso Comum: Projetos iniciais, ensino, controle de LEDs, motores, sensores simples.

O Arduino Uno é ideal para iniciantes e projetos de pequeno a médio porte. Ele é amplamente utilizado em prototipagem e aprendizado, sendo uma excelente escolha para quem está começando a programar microcontroladores. Possui uma quantidade razoável de portas digitais e analógicas, além de ser compatível com a maioria dos shields (módulos de expansão). É comum vê-lo em projetos como sistemas de iluminação com LEDs, controle de motores simples, sensores de temperatura e umidade, e pequenos robôs.

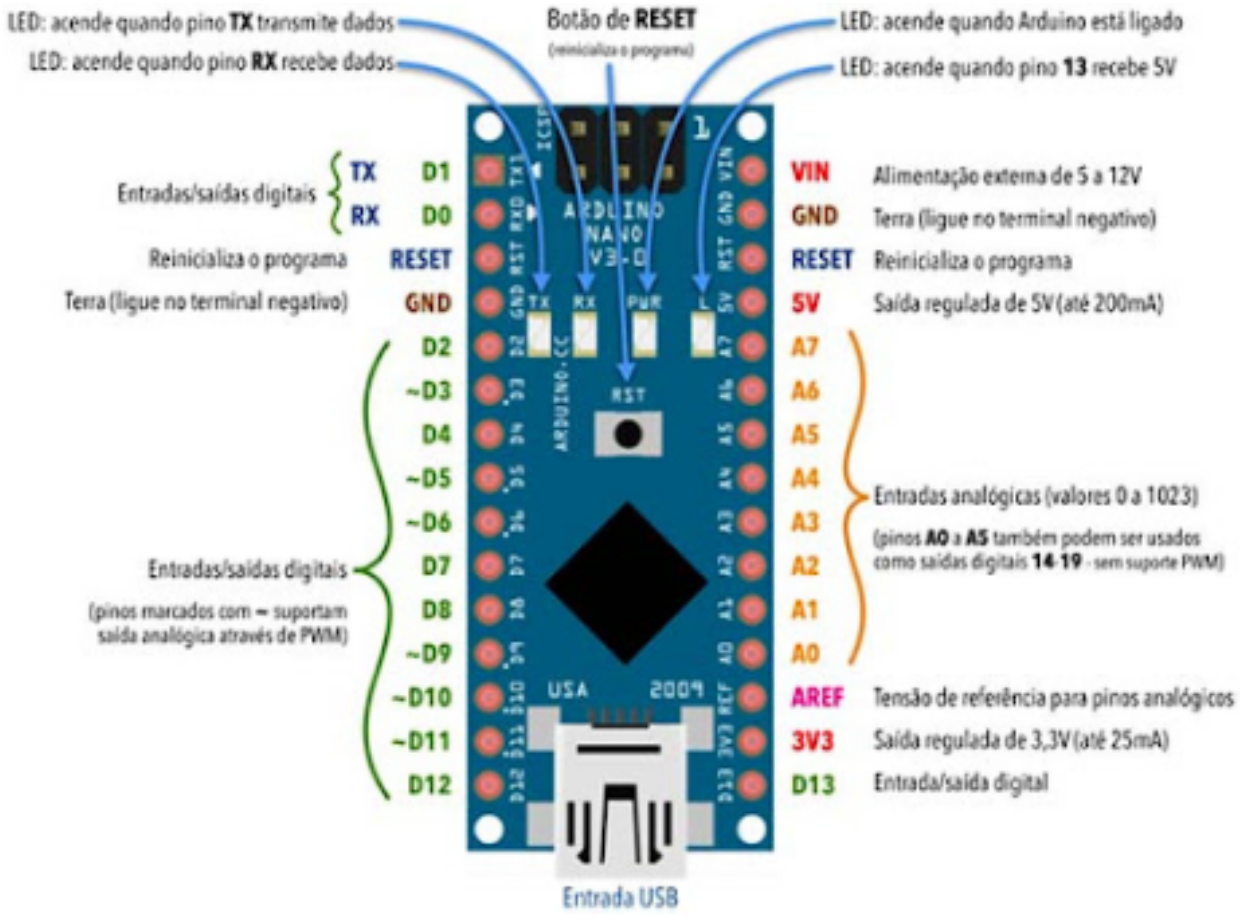
Placa Arduíno Mega:



- Microcontrolador: ATmega2560
- Pinos Digitais: 54 (15 com PWM)
- Entradas Analógicas: 16
- Memória Flash: 256 KB
- Mais UARTs (Serial): 4 portas
- Uso Comum: Robótica avançada, impressoras 3D, projetos com muitos sensores/motores.

O Arduino Mega é mais adequado para projetos complexos que exigem maior quantidade de entradas e saídas ou mais memória. Ele possui bem mais portas digitais e analógicas que o Uno, o que o torna ideal para aplicações como impressoras 3D, robôs com múltiplos sensores e motores, ou painéis de controle com muitos botões e indicadores. Sua capacidade de lidar com múltiplos dispositivos simultaneamente o torna uma ótima escolha quando o Uno começa a ficar limitado.

Placa Arduino Nano:



- Microcontrolador: ATmega328P (igual ao UNO)
- Formato: Muito menor, ideal para breadboards
- Pinos Digitais: 14 (8 com PWM)
- Entradas Analógicas: 8
- Alimentação: via USB mini ou VIN (externo)
- Uso Comum: Projetos compactos, vestíveis, embarcados.

O Arduino Nano é a versão compacta da plataforma. Seu tamanho reduzido o torna ideal para projetos onde o espaço é uma preocupação, como wearables, dispositivos portáteis ou circuitos que precisam ser integrados diretamente em uma placa de circuito impresso (PCB). Apesar de pequeno, o Nano tem praticamente as mesmas funcionalidades do Uno, o que o torna uma ótima opção para projetos definitivos e que não precisam de muitos periféricos conectados ao mesmo tempo.

Lógica de Programação em C/C++ em Arduíno

Programar em C ou C++ no Arduino pode parecer algo técnico à primeira vista, mas com um pouco de prática e imaginação, é como ensinar um pequeno robô a tomar decisões, lembrar coisas, repetir tarefas e interagir com o mundo. O Arduino é um microcontrolador que obedece ao que você programar. A linguagem C/C++ é a forma como você conversa com ele.
Aprender a lógica de programação para Arduino é entender como dar instruções claras e passo a passo, para que o Arduino realize desde tarefas simples, como acender um LED, até projetos mais complexos, como controlar sensores ou enviar mensagens via Wi-Fi.

Variáveis, tipos de dados e estruturas de controle.

Antes de tudo, é importante entender como o Arduino guarda informações. É aí que entram as variáveis – pense nelas como pequenas caixinhas onde podemos guardar valores, como números ou palavras. Cada variável tem um tipo, que define o que ela pode guardar. Por exemplo:

- Se quisermos saber a idade de uma pessoa, por exemplo, podemos usar o tipo int, que armazena números inteiros.
- Se quisermos representar uma temperatura com casas decimais, usamos o tipo float.
- Para letras ou símbolos individuais, usamos char.
- Já quando precisamos de algo que seja apenas verdadeiro ou falso, usamos bool.

Na prática, declarar variáveis é como dar nomes às nossas caixinhas e dizer o que elas vão guardar. Exemplos:

```
#include <stdio.h> // Bibliotecas

int idade = 18; //Declarando um número inteiro
bool ligado = false; //Declarando uma variável booleana
```

Essas informações ficam salvas na memória do Arduino enquanto o programa estiver rodando, mas guardar informações não é o bastante – o Arduino também precisa tomar decisões com base nelas. Para isso, usamos estruturas de controle. A mais comum é o **if**, que funciona como uma pergunta:

"Se isso for verdadeiro, então faça tal coisa."

```c
#include <stdio.h> //Biblioteca padrão de entrada/saída (necessária para usar printf)
#include <stdbool.h>//Biblioteca que permite o uso do tipo bool (verdadeiro/falso)

float temperatura = 32.5;//Declara uma variável do tipo float (número decimal) e já define o valor 32.5
bool Ventilador;//Declara uma variável booleana que indica se o ventilador está ligado ou não

//Define a função que liga o ventilador
void ligarVentilador() {
    Ventilador = true; //Atualiza o estado da variável para "ligado"
    printf("Ventilador ligado.\n");//Imprime a mensagem no console
}

//Define a função que desliga o ventilador
void desligarVentilador() {
    Ventilador = false;//Atualiza o estado da variável para "desligado"
    printf("Ventilador desligado.\n");// Imprime a mensagem no console
}

//Função principal do programa, onde tudo começa
int main() {
    //Verifica se a temperatura é maior que 30
    if (temperatura > 30) {
        ligarVentilador();//Se for, liga o ventilador
    } else {
        desligarVentilador();//Se não for, desliga o ventilador
    }

    return 0;//Encerra o programa
}
```

Além das decisões, também é comum querermos que o Arduino repita tarefas automaticamente, como piscar um LED várias vezes. Isso é feito com laços de repetição, como o for e o while. Com o while, conseguimos repetir enquanto uma condição for verdadeira:

```c
for (int i = 0; i < 5; i++) {

  acenderLed(i);

}

while (botaoPressionado == true) {

  tocarAlarme();

}
```

Manipulação de entradas e saídas.

Um dos pontos mais interessantes da programação com Arduíno é a possibilidade de interagir com o mundo real. Ele consegue perceber o ambiente ao seu redor (entradas) e reagir a ele (saídas).
Essas interações podem ser:

Entradas e saídas digitais: Possuem apenas dois estados, ligado (HIGH) ou desligado (LOW), as entradas são usadas para ler componentes como botões ou sensores de presença. Por exemplo:

```
int estado = digitalRead(2);
```

Isso verifica se o botão está pressionado (HIGH) ou não (LOW)
As saídas permitem ligar ou desligar algo. Por exemplo:

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

Isso liga o LED conectado ao pino 13.

Entradas e saídas analógicas: Aqui os valores variam de forma contínua, dentro de uma faixa. As entradas são usadas para ler valores variáveis, como, a posição de um potenciômetro, luz do ambiente e a temperatura de algum lugar. Por exemplo:

```
int valor = analogRead(A0);
```

Uso de bibliotecas.

Conforme os projetos ficam mais avançados, percebemos que algumas tarefas se repetem ou são complexas demais para fazer sempre do zero. Para isso, usamos as bibliotecas: conjuntos de códigos prontos que facilitam muito a programação. Por exemplo, para controlar um servo motor, módulos Bluetooth, sensores de temperatura, displays LCD ou comunicação via Wi-Fi usamos a biblioteca Servo.h:

```arduino
#include <servo.h>

Servo motor;

void setup() {
  motor.attach(9);
  motor.write(90); // Gira o motor para 90 graus
}
```

Além de usar as bibliotecas existentes, você pode criar suas próprias bibliotecas quando quiser organizar funções que usa com frequência. Isso ajuda a manter o código limpo e reaproveitável.

Criação de bibliotecas

Para criar uma biblioteca do zero vamos seguir alguns passos de boas práticas de programação. Uma biblioteca Arduíno é composta por três arquivos, um ".h" com as declarações, ".cpp" com a implementação e uma pasta com o nome da biblioteca onde os arquivos vão ficar. Vamos criar uma biblioteca chamada Ventilador com as funções ligar() e desligar() para exemplificar o passo a passo.

Criar a pasta

- Abra a pasta "documentos" no seu computador
- Vá para o diretório das bibliotecas do Arduíno, algo como:
- Documentos/Arduino/libraries/
- Crie uma nova pasta chamada "Ventilador"

Criar um arquivo Ventilador.cpp

Abra o editor de texto e salve o conteúdo da função, neste caso:

```arduino
#ifndef VENTILADOR_H //Evita duplicações no código.
#define VENTILADOR_H

#include <Arduino.h>

class Ventilador {
  public:
    Ventilador(int pino);//Cria o ventilador e diz em qual pino está ligado
    void ligar();
    void desligar();

  private:
    int _pino;
};

#endif
```

Criar um arquivo Ventilador.cpp

Abra o editor de texto e salve o conteúdo da função, neste caso:

```cpp
#include "Ventilador.h"

Ventilador::Ventilador(int pino) {
  _pino = pino;
  pinMode(_pino, OUTPUT);
}

void Ventilador::ligar() {
  digitalWrite(_pino, HIGH);
}

void Ventilador::desligar() {
  digitalWrite(_pino, LOW);
}
```

Testando a biblioteca

Abra a Arduino IDE e crie um novo sketch com o nome que quiser. Use o seguinte código para testar:

```cpp
#include <Ventilador.h>

Ventilador meuVentilador(9);  // Usa o pino 9

void setup() {
  meuVentilador.ligar();
  delay(2000);
  meuVentilador.desligar();
}

void loop() {
}
```

Dica: reinicie a Arduino IDE para que a biblioteca seja reconhecida.

Projetos Práticos e Boas Práticas

Exemplos passo a passo

Pisca-Pisca

```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(500);
}

}
```

- Definindo o led no pino 13 como saída
- Definir se o pino é de saída (led, OUTPUT)
- DigitalWrite(led, HIGH); Liga o LED
- delay Espera 1 segundo (1000 ms)
- DigitalWrite(led, LOW); Desliga o LED
- delay Espera ½ segundo (500ms)

Pisca-pisca com botão.

```
void setup() {

int led = 13;
int botao = 2;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(botao, INPUT_PULLUP);
}

void loop() {
  if (digitalRead(botao) == LOW) {
    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);
  } else {
    digitalWrite(led, LOW);
  }
}

}
```

- Define o pino do LED no pino 13
- Defino o botão no pino 2
- Definir se o pino é de saída (led, OUTPUT)
- Definir botão como entrada
- Definir a lógica do botão
- Em **if** ele define que o LED estará ligado enquanto o botão pressionado
- Em **else** mostra desligado quando o botão não está pressionado

Pisca-Pisca Automático

```
int led = 9;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  for (int brilho = 0; brilho <= 255; brilho++) {
    analogWrite(led, brilho);
    delay(10);
  }
  for (int brilho = 255; brilho >= 0; brilho--) {
    analogWrite(led, brilho);
    delay(10);
  }
}
```

- Definimos o Pino para ser PWM para colocar o LED
- Define o pino para ser saída
- Define a lógica do pino
- O primeiro for será para acender o brilho do LED até o pico de iluminação utilizando a função analogWrite.
- O Segundo for será para apaga na mesma velocidade

Fazendo um semáforo

Fazendo um semáforo

```
int verde = 8;
int amarelo = 9;
int vermelho = 10;

void setup() {
  pinMode(verde, OUTPUT);
  pinMode(amarelo, OUTPUT);
  pinMode(vermelho, OUTPUT);
}

void loop() {
  digitalWrite(verde, HIGH);
  delay(3000);
  digitalWrite(verde, LOW);

  digitalWrite(amarelo, HIGH);
  delay(1000);
  digitalWrite(amarelo, LOW);

  digitalWrite(vermelho, HIGH);
  delay(3000);
  digitalWrite(vermelho, LOW);
}
```

- Definimos os LEDs vermelho, verde e amarelo em seus respectivos pinos.
- Definimos como pinos de Saída
- Definimos a lógica para acender o verde e depois de 3 segundos desliga-lo, o amarelo fica ligado por 1 segundo e o vermelho fica ligado por 3 segundos.

Ligando o led por meio de Sensor (de presença)

```
int pinoPIR = 2;
int led = 13;

void setup() {
  pinMode(pinoPIR, INPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int estadoPIR = digitalRead(pinoPIR);

  if (estadoPIR == HIGH) {
    digitalWrite(led, HIGH);
    Serial.println("Movimento detectado!");
  } else {
    digitalWrite(led, LOW);
    Serial.println("Sem movimento");
  }

  delay(200);
}
```

- Definimos os pinos do LED e do sensor(PIR)
- Definimos o LED como saída e o PIR como entrada
- Colocamos a função Serial.begin(9600) que faz a comunicação entre o arduíno e o computador configurada a 9600 bits por ser o padrão
- Definimos o estado do Pino (se tem presença ou não)
- Caso o estado seja HIGH o pino acende e manda a mensagem para o computador (“Movimento detectado!”)
- Caso o estado seja LOW, o LED desliga e exibe a mensagem (“Sem movimento”)

dicas de melhores práticas de programação.

Depuração: Utilize a função Serial para a comunicação com o computador e verificar os processos, teste os códigos por partes a**o longo do processo.**

Organização: Busque definir funções ao invés de colocar tudo no loop()

```
void lerSensor();
void controlarLed();
void enviarSerial();
```

Deixe comentários claros para trechos importantes para que outras pessoas entendam seu código e você também depois de um tempo.

```
// Bom: explica a lógica
// Liga o LED se a leitura do sensor for maior que o limite
if (sensorValue > limite) digitalWrite(ledPin, HIGH);
```

Para mais projetos práticos:

https://github.com/DeniseValeriaVelarde/Tinkercad-Projetos

Wokwi

O Wokwi é uma plataforma online que permite simular projetos de eletrônica e programação com microcontroladores diretamente no navegador, sem a necessidade de hardware físico. Ele é amplamente utilizado para testar códigos e circuitos de forma rápida e prática, sendo especialmente útil para estudantes, educadores e entusiastas da eletrônica.

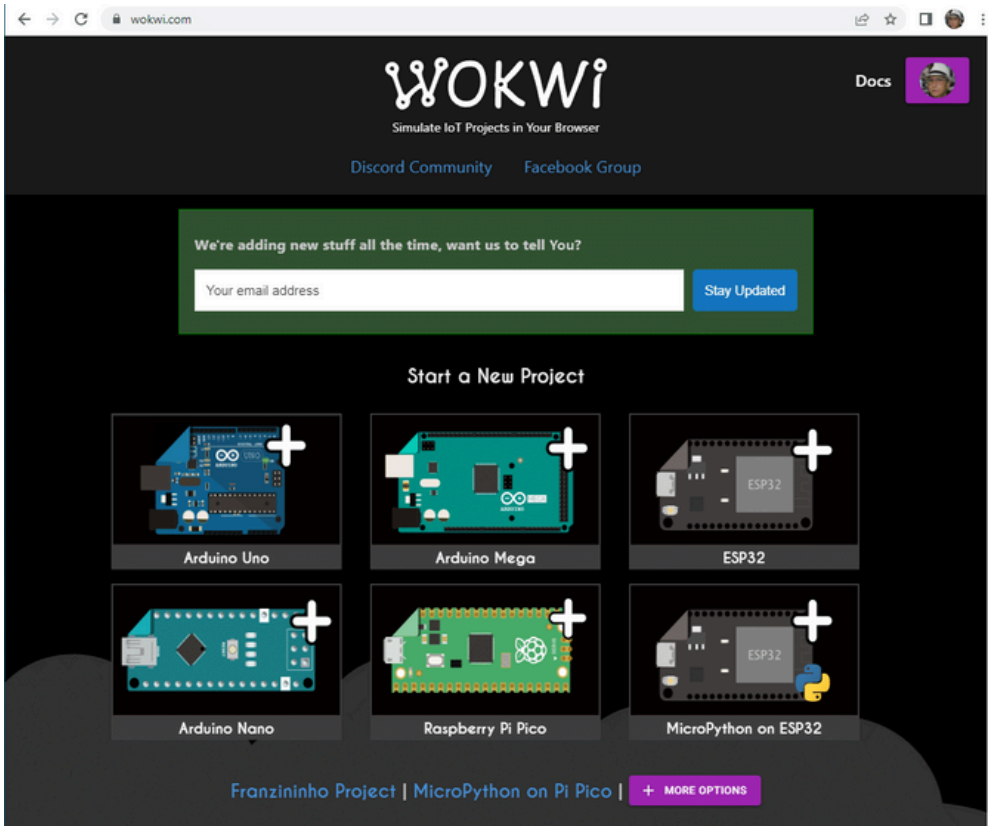
Com o Wokwi, você pode simular placas como Arduino Uno, Mega, Nano, ESP32, Raspberry Pi Pico, entre outras. A plataforma oferece uma vasta biblioteca de componentes eletrônicos, incluindo LEDs, displays LCD e OLED, sensores diversos, motores, botões, relés e muito mais. Isso permite criar e testar circuitos complexos, visualizar o comportamento dos componentes em tempo real e depurar o código de forma eficiente.

Uma das grandes vantagens do Wokwi é a possibilidade de compartilhar seus projetos facilmente por meio de links, facilitando a colaboração e o aprendizado em grupo. Além disso, a plataforma oferece integração com o Visual Studio Code por meio de uma extensão, permitindo simular e depurar projetos diretamente no ambiente de desenvolvimento.

No entanto, é importante notar que o Wokwi possui algumas limitações. Por exemplo, ele não simula correntes ou tensões reais, e há uma fonte de alimentação virtual ilimitada. Isso significa que, embora seja excelente para testar a lógica do código e a interação entre componentes, os resultados podem diferir do comportamento em um circuito físico real .

Em resumo, o Wokwi é uma ferramenta poderosa para prototipagem e aprendizado em eletrônica e programação de microcontroladores, oferecendo uma maneira acessível e prática de desenvolver e testar projetos sem a necessidade de hardware físico.

https://www.makerhero.com/blog/wokwi-simulador-de-arduino/


wokwi.com
WOKWI
Simulate IoT Projects in Your Browser
Docs
Discord Community
Facebook Group
We're adding new stuff all the time, want us to tell You?
Your email address
Stay Updated
Start a New Project
Arduino Uno
Arduino Mega
ESP32
Arduino Nano
Raspberry Pi Pico
MicroPython on ESP32
Franzininho Project | MicroPython on Pi Pico |
+ MORE OPTIONS

Recursos e Comunidade

Indicação de fóruns, sites, repositórios e materiais complementares para aprofundamento e suporte.

Se você quiser aprofundar seus conhecimentos em Arduino, programação ou eletrônica, aqui estão algumas indicações de sites, fóruns, vídeos e repositórios com conteúdo gratuito e acessível:

1. https://www.arduino.cc/
Documentação das placas, tutoriais, referências da linguagem e biblioteca oficial.
2. https://docs.arduino.cc/language-reference/
Lista de comandos, estruturas e exemplos em tempo real.
3. https://forum.arduino.cc/
Comunidade global com tópicos sobre problemas técnicos, projetos, dúvidas e novidades.
4. https://www.clubedohardware.com.br/forums/forum/255-eletr%C3%B4nica-digital-e-arduino/
Em português, é muito útil para iniciantes e estudantes brasileiros.
5. https://stackoverflow.com/questions/tagged/arduino
Perguntas e respostas mais técnicas sobre programação e bibliotecas.
6. https://www.reddit.com/r/arduino/
Compartilhamento de projetos, dicas e problemas do dia a dia.
7. https://github.com/arduino
Repositório oficial com bibliotecas, exemplos e contribuições.
8. https://github.com/tiimgreen/github-cheat-sheet
Lista com bibliotecas, tutoriais, simuladores e kits.

Para mais projetos práticos:

https://github.com/DeniseValeriaVelarde/Tinkercad-Projetos

PETCode

Créditos:

Denise Valéria Velarde Mamani
Lara Letittja Sague Lopez Guardiola Velloso
Luiz Fernando Viana Ciriaco

PET C3
Ciências Computacionais