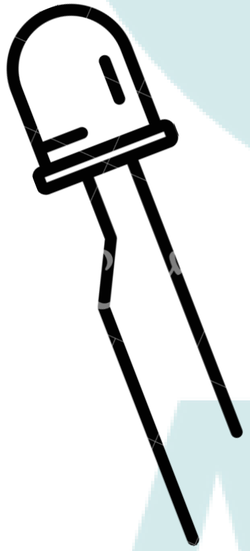
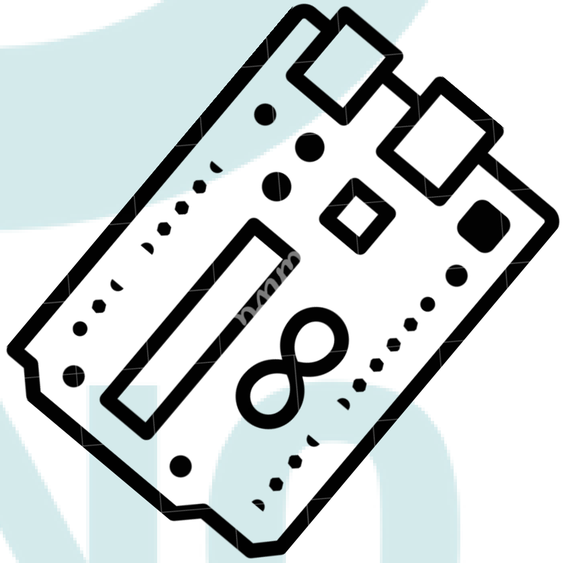
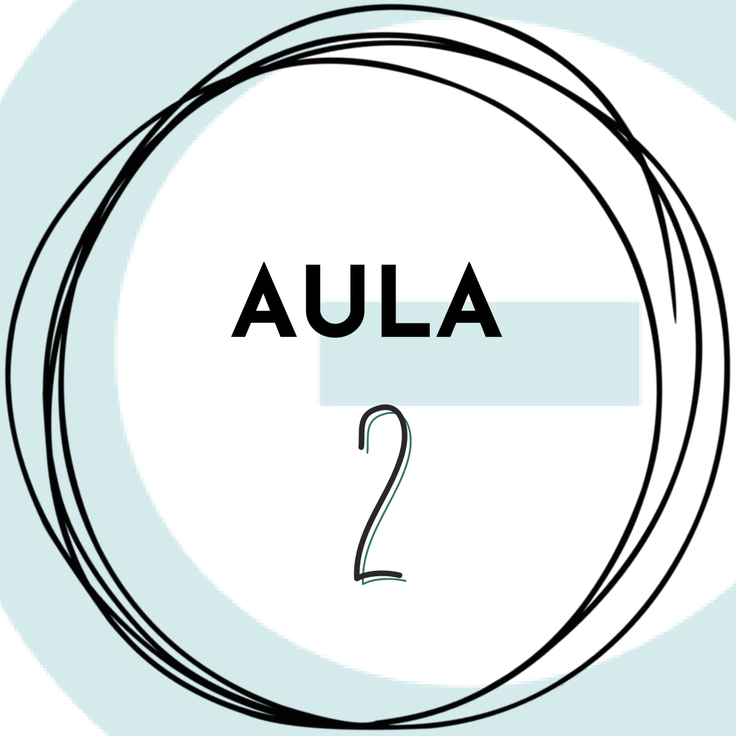
Arduino:
Conceitos, Programação e Aplicação Prática



UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CÂMPUS PATO BRANCO





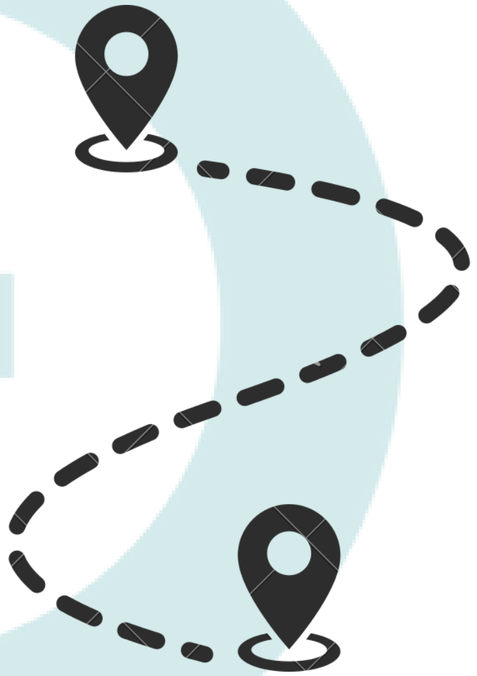
Eletrônica Básica e Noções Básicas de Programação

Coordenador
Fábio Favarim

Instrutor
Gabriel S. Folly

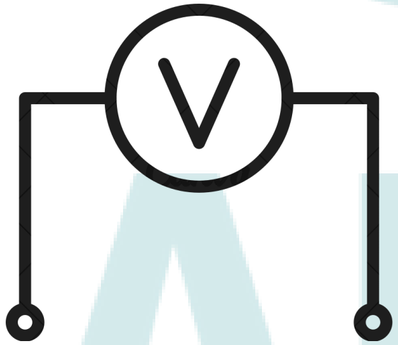
Roteiro

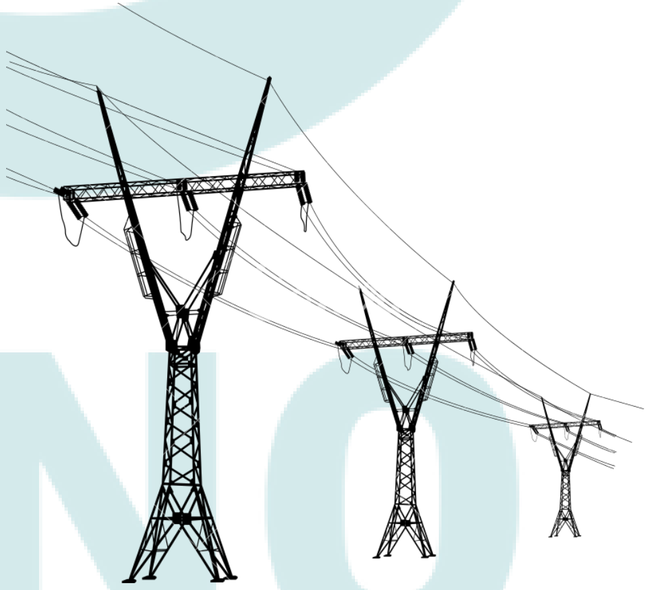
1. Eletrônica Básica
2. Estrutura do Algortimo
3. Demonstração Prática
4. Estado HIGH e LOW
5. Entrada e Saída
6. Laboratórios



Tensão Elétrica

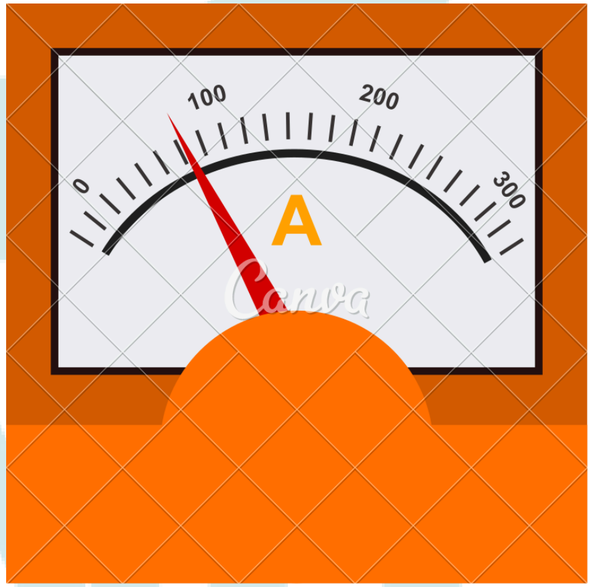
- **Definição**: Diferença de potencial elétrico medida em volts (V).
- **Campo Elétrico**: Cria um campo que impulsiona elétrons, permitindo o fluxo de corrente.
- **Importância**: Determina a energia potencial disponível para trabalho elétrico.
- **Fundamental**: Garante o funcionamento adequado de circuitos elétricos e eletrônicos.



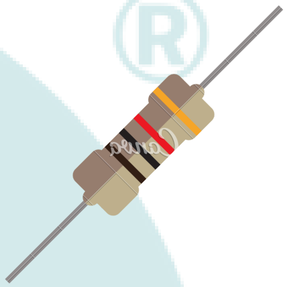


Corrente Elétrica

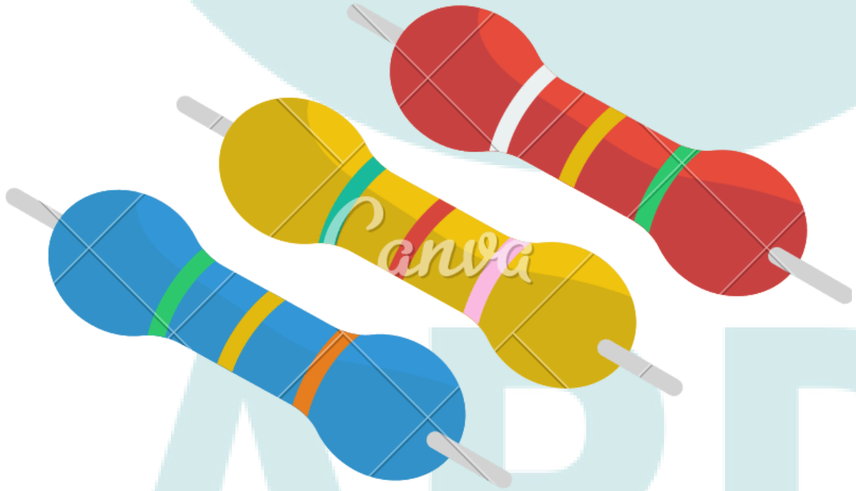
- **Definição**: Fluxo de elétrons em um circuito, medida em amperes (A).
- **Movimento Ordenado**: Ao longo de condutores como fios metálicos.
- **Essencial**: Transmite energia e permite trabalho elétrico.
- **Intensidade**: Determina a quantidade de carga em um ponto em um período de tempo.



Resistência Elétrica

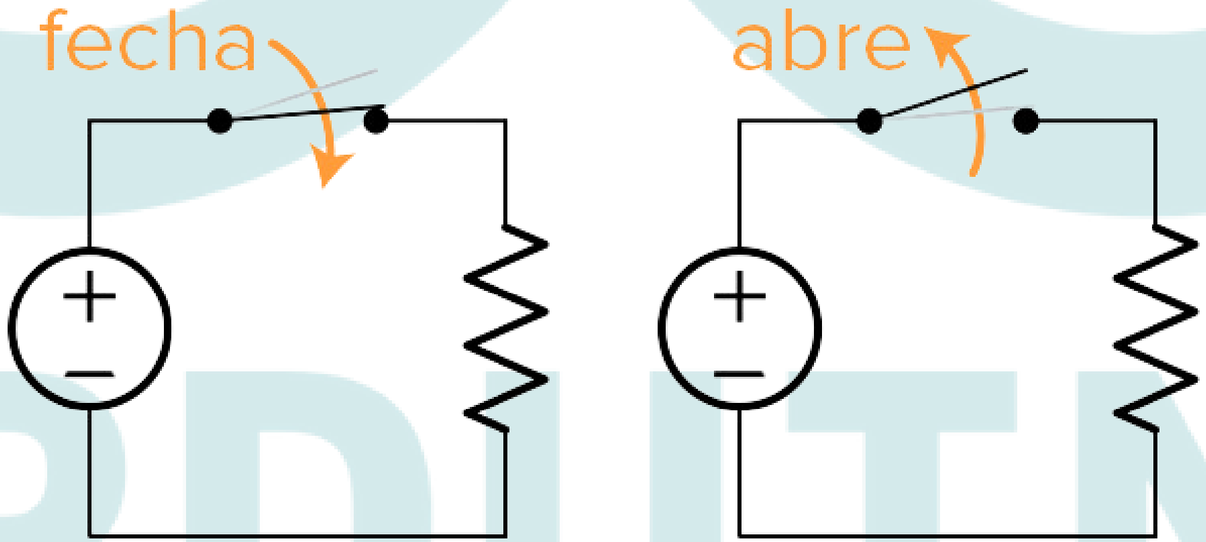


- **Definição**: Oposição ao fluxo de corrente medida em ohms (Ω).
- **Dependência**: Propriedades do material, comprimento e área do condutor.
- **Lei de Ohm**: Relação entre tensão, corrente e resistência em um circuito.
- **Resistores**: Componentes para controlar resistência, regulando a corrente elétrica.



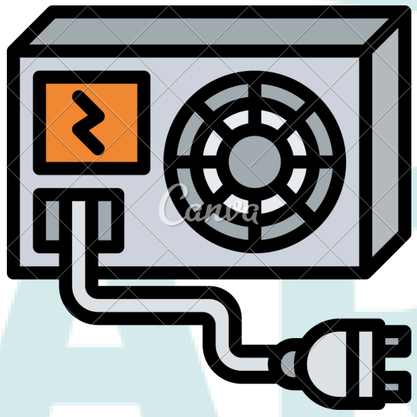
Circuito Elétrico

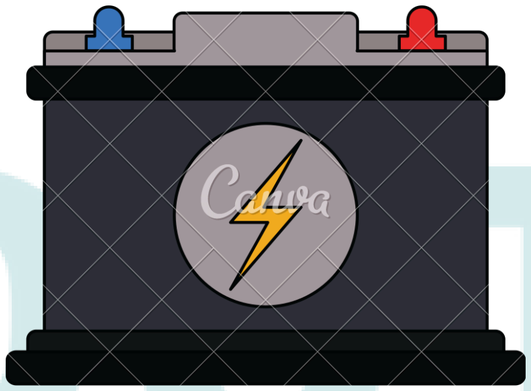
- **Definição**: Estrutura fechada que permite o fluxo de corrente elétrica.
- **Componentes**: Incluem fios condutores e dispositivos interligados.
- **Fluxo de Corrente**: Da fonte de energia, através do circuito, alimentando dispositivos e retornando à fonte.
- **Essencial**: Seguem leis da eletricidade, fundamentais para sistemas elétricos cotidianos.



Fontes de Alimentação

- **Definição**: Dispositivos para fornecer energia elétrica a outros dispositivos eletrônicos.
- **Conversão de Energia**: Transformam energia de entrada (rede elétrica ou bateria) para alimentar componentes eletrônicos.
- **Tipos**: Lineares e chaveadas, com características e aplicações específicas.
- **Importância**: Fundamental em eletrônica, computadores e dispositivos eletrônicos, garantindo fornecimento estável e controlado de energia.





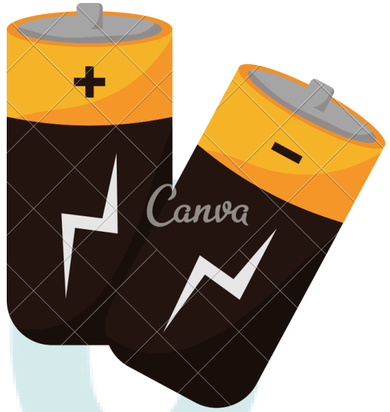


Tabela Resistores

Cor	1ª Faixa	2ª Faixa	Faixa multiplicadora	Tolerância
Preto	0	0	×1Ω	-
Marrom	1	1	×10Ω	±1%
Vermelho	2	2	×100Ω	±2%
Laranja	3	3	×1kΩ	-
Amarelo	4	4	×10kΩ	-
Verde	5	5	×100kΩ	±0,5%
Azul	6	6	×1MΩ	±0,25%
Violeta	7	7	×10MΩ	±0,1%
Cinza	8	8	-	±0,05%
Branco	9	9	-	-
Dourado	-	-	×0,10Ω	±5%
Prateado	-	-	×0,01Ω	±10%
Sem cor	-	-	-	±20%

Lei de Ohm

A Lei de Ohm, formulada por Georg Simon Ohm, estabelece que a corrente elétrica (I) em um circuito é diretamente proporcional à tensão (V) aplicada e inversamente proporcional à resistência (R) do circuito.

- **Definição**: Estabelece relação entre corrente elétrica (I), tensão (V) e resistência (R) em um circuito.
- **Fórmula**: I=V/R.
- **Aplicações Práticas**: Projetar, dimensionar e diagnosticar circuitos elétricos.

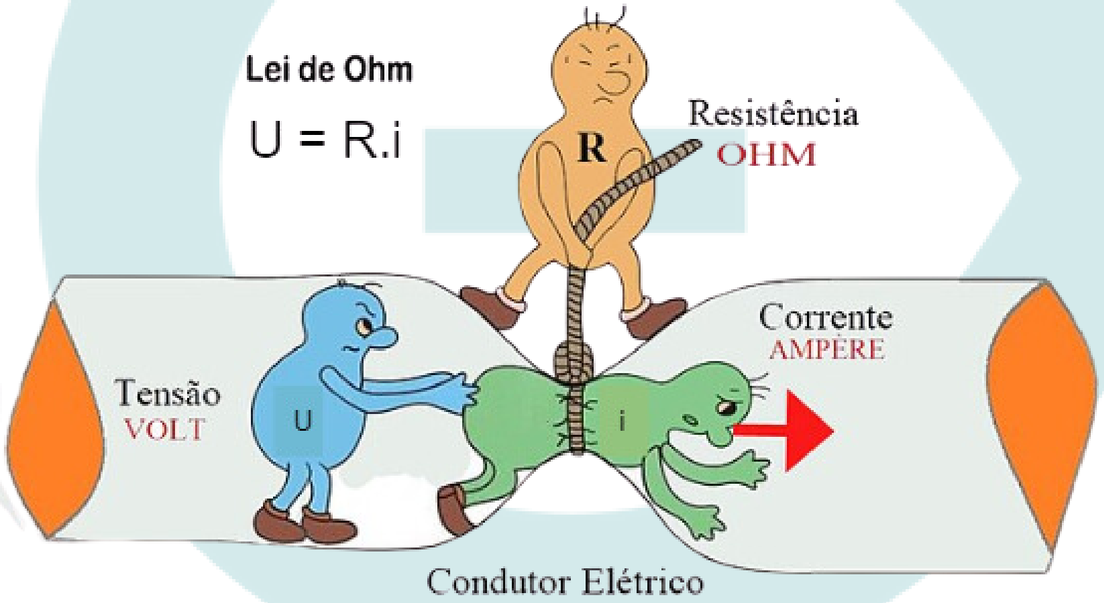
$$V = I.R$$

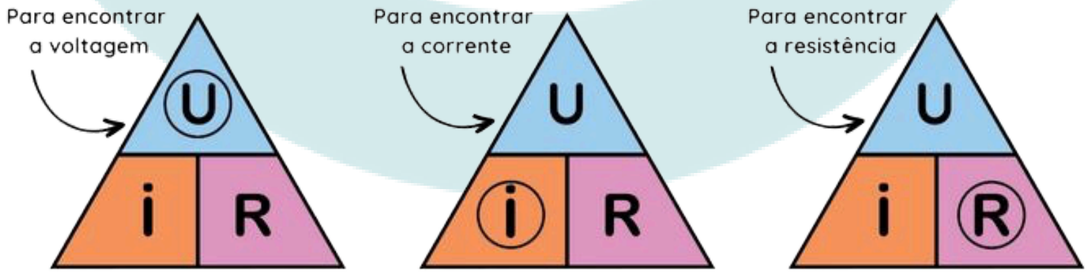
Exemplo Prático

Considere um circuito simples com um resistor de 220 ohms conectado a uma fonte de alimentação de 5 volts. Calcule a corrente que passa pelo resistor, a tensão através dele e a resistência do componente.

- Corrente (I):
  - I = V / R
  - 5V / 220Ω = 0.0227A (ou 22.7mA).
- Tensão (V):
  - V = I × R
  - 0.0227A × 220Ω = 5V.
- Resistência (R):
  - Dados fornecidos indicam 220 ohms.

Auxilio para Lei de Ohm





$$U = i \times R \qquad i = \frac{U}{R} \qquad R = \frac{U}{i}$$

Aplicações da Lei de Ohm

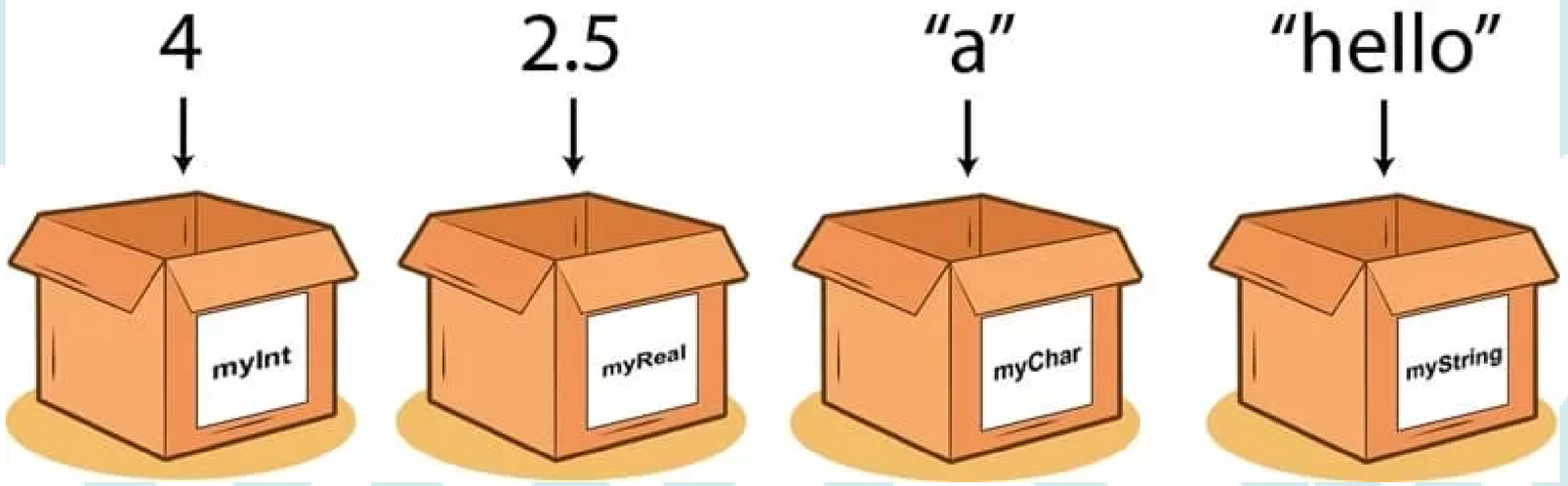
- **Projeto e Dimensionamento**: Garante corrente apropriada para dispositivos.
- **Manutenção e Diagnóstico**: Identificação de problemas de resistência excessiva.
- **Eletrônica, Doméstica e Transmissão de Energia**: Otimização do fluxo elétrico para eficiência e segurança.

Variáveis em Arduino

- Variáveis armazenam e representam dados.
- **Tipos de variáveis**:
  - **Inteiro (int)**: Armazena números inteiros, sem parte decimal.
    - Exemplo: int idade = 30;
  - **Flutuante (float)**: Armazena números com parte decimal.
    - Exemplo: float temperatura = 24.7;
  - **Caractere (char)**: Armazena um único caractere.
    - Exemplo: char letra = 'K';
  - **Lógico (boolean)**: Armazena valores verdadeiro ou falso.
    - Exemplo: boolean ligado = true;
  - **String (string)**: Armazena sequência de caracteres.
    - Exemplo: "Hello, world!";
- Declaração padrão: tipo_de_dado nome_da_variavel = valor_inicial.

Manipulação de Variáveis

- Manipulação permite armazenar, alterar e utilizar dados durante a execução do programa.
- **Exemplo**: armazenar valores de sensores, controlar dispositivos, realizar cálculos.



Constantes em Arduino

- Constantes são valores fixos que não podem ser alterados durante a execução do programa.
- **Utilização da palavra-chave 'const'**:
  - **Exemplo**: const int LED_VERMELHO = 12;
- **Vantagens**:
  - Tornam o código mais legível.
  - Facilitam a manutenção, permitindo atualizar valores em um único lugar.
- **Limitações**:
  - Ao contrário das variáveis, as constantes não podem ser alteradas após a definição o valor permanece constante durante a execução do programa.

Estrutura Básica - setup() e loop()

- **setup()**:
  - Executado uma vez no início do programa.
  - Configurações iniciais e inicialização de variáveis.
- **loop()**:
  - Executado repetidamente após a função setup().
  - Contém o código principal do programa, executado em um loop infinito.

Elementos Adicionais

- **Comentários**:
  - Linhas começando com "//" ou "/* */" são comentários.
  - Documentação do código para explicar funcionalidades e facilitar a compreensão.
- **Inclusão de Bibliotecas e Variáveis Globais**:
  - #include <NomeDaBiblioteca.h> para importar bibliotecas.
  - Variáveis acessíveis em todo o programa para armazenar informações importantes.

Controle de Fluxo, Funções e Delay

- **Controle de Fluxo**:
  - Estruturas condicionais (if, else, switch) e loops (for, while).
  - Tomada de decisões e repetição de blocos de código com base em condições.
- **Chamadas de Funções**:
  - Execução de tarefas encapsuladas em funções.
  - Modularização do código para facilitar a compreensão e manutenção.
- **Delay**:
  - delay(milissegundos) para pausar a execução por um período especificado.
  - Esses blocos formam a estrutura básica para desenvolver códigos organizados, modulares e eficientes em programação Arduino.

Demonstração Prática

Nesse código apresentado, temos a definição da porta 12 do Arduino com um LED, após isso na função setup esse LED foi definido como um dispositivo de saída, ou seja, OUTPUT. Na funcao loop estamos fazendo com que esse LED acenda por 1 segundo e desligue por 10. Com essa funcao loop isso acontecera repetidamente até que o Arduino seja desligado ou carregado com outro programa.

```
1   void setup() {
2
3   }
4
5   void loop() {
6
7   }
8
```

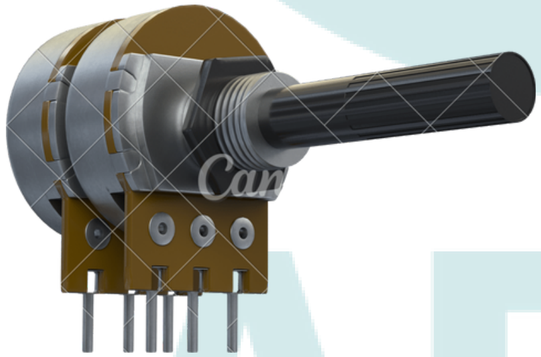
```
1   int LED = 12;
2
3   void setup() {
4     pinMode(LED, OUTPUT); //Definindo o LED que esta na porta 12 como OUTPUT (Saida)
5   }
6
7   void loop() {
8     digitalWrite(LED, HIGH); //Liga
9     delay(1000); //1 segundo
10    digitalWrite(LED, LOW);//Desliga
11    delay(10000);// 10 segundos
12  }
13
```

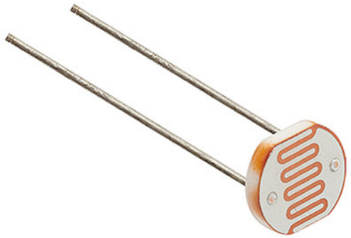
Conceito de Estado “HIGH” e “LOW”

- **Estado High**:
  - Representa nível de tensão "alto" ou "1" em sistemas digitais.
  - Voltagem próxima à alimentação (geralmente 5V em Arduino padrão).
  - Indica sinal ativo, como LED aceso.
- **Estado Low**:
  - Representa nível de tensão "baixo" ou "0" em sistemas digitais.
  - Voltagem próxima a zero volts.
  - Indica sinal inativo, como LED apagado.
- **Aplicações**:
  - **Controle de Dispositivos**: Usado para controlar LEDs, motores, relés, etc., conectados a pinos Arduino.
  - **Lógica Binária**: High associado a "1", Low a "0", base da lógica binária em sistemas digitais.

Entrada no Arduino

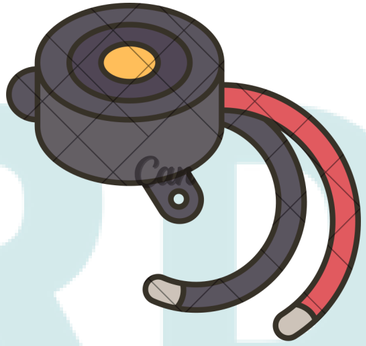
- **Entrada (INPUT)**: Configuração de pinos para receber informações do ambiente externo.
  - **Exemplos**: Sensores, botões, potenciômetros.
  - **Leitura de Dados**: Capaz de ler informações, como estado de botões ou leitura de sensores.





Saída e Controle Bidirecional

- **Saída (OUTPUT)**: Configuração de pinos para enviar informações ao ambiente externo.
  - **Exemplos**: LEDs, motores, relés.
  - **Envio de Dados**: Capaz de enviar sinais, como acender LEDs ou ativar motores.
- **Controle Bidirecional**:
  - Muitos pinos no Arduino têm capacidade bidirecional.
  - Podem ser configurados tanto como INPUT quanto OUTPUT.



Configuração e Exemplo Prático

- **Configuração no Arduino IDE**: Uso da função pinMode() para configurar pinos como INPUT ou OUTPUT.
  - **Exemplo**: pinMode(pin, INPUT) configura um pino como entrada.
- **Interação com Sensores e Atuadores**:
  - **Sensores**: Leitura de sensores de temperatura, por exemplo, usando pinos configurados como INPUT.
  - **Atuadores**: Controle de LEDs ou motores usando pinos configurados como OUTPUT.
- **Manipular entradas e saídas é crucial para a interação efetiva do Arduino com o ambiente, permitindo projetos interativos e controlados por programação.**

Laboratório 01

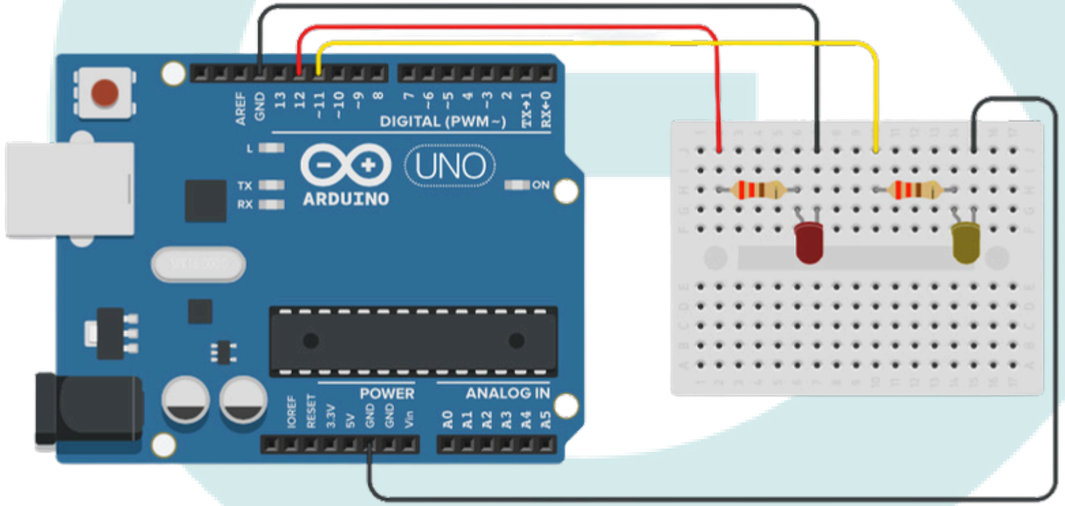
Piscando Dois LEDs Alternadamente

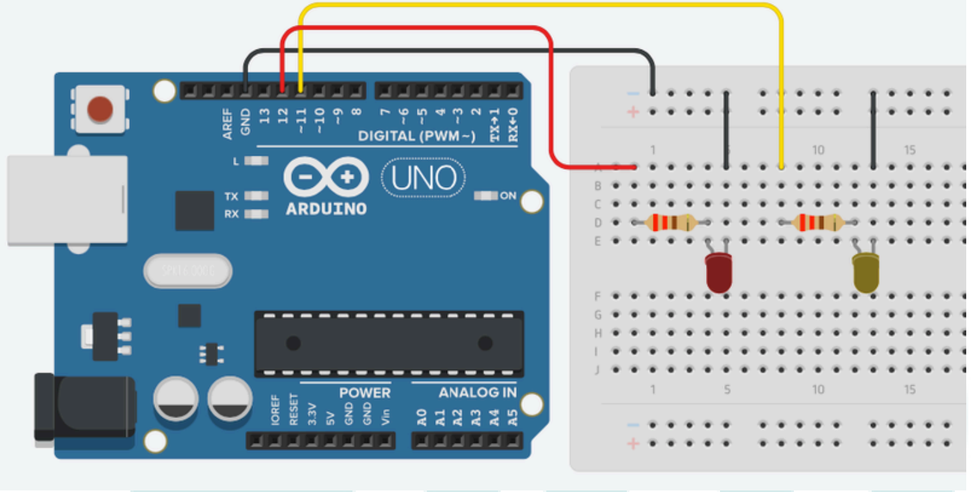
Neste guia, você aprenderá a montar um circuito usando um Arduino para fazer dois LEDs piscarem alternadamente.

Materiais Necessários:

- Arduino Uno.
- 2 LEDs (de cores diferentes, se possível).
- 2 resistores de 220 ohms.
- Cabos jumper (fios de conexão).
- Protoboard.

Solução Laboratório 01






```arduino
int LED_VERMELHO = 12;
int LED_AMARELO = 11;

void setup() {
  pinMode(LED_VERMELHO, OUTPUT); //Definindo o LED que esta na porta 12 como OUTPUT (Saida)
  pinMode(LED_AMARELO, OUTPUT);
}

void loop() {
  digitalWrite(LED_VERMELHO, HIGH); //Liga o LED vermelho por 2 segundos
  delay(2000); //2 segundos
  digitalWrite(LED_VERMELHO, LOW);//Desliga o LED Vermelho
  digitalWrite(LED_AMARELO, HIGH);//Liga o LED amarelo por 2 segundos e apaga
  delay(2000); //2 segundos
  digitalWrite(LED_AMARELO, LOW);
}
```

Uma boa prática em eletrônica é colocar todos os jumpers que vão para o terra no barramento horizontal da protoboard que terá somente uma ligação para o GND do Arduino. Enquanto estamos trabalhando com projetos simples como esse não vemos sentido, mas a medida que o número de componentes aumentam é possível notar a necessidade dessa abordagem para solução de nossos circuitos.

Laboratório 02

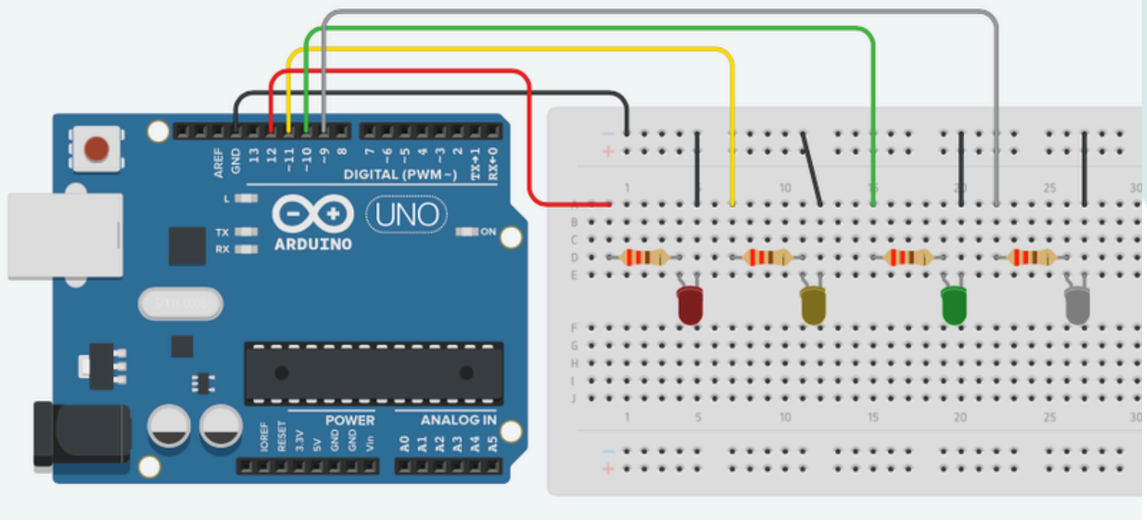
Simulação Natal com circuito de pisca-pisca com 4 LEDs.

Neste guia, você aprenderá a montar um circuito usando um Arduino para simular luzes de natal com LEDs.

Materiais Necessários:

- Arduino Uno
- 4 LEDs (diferentes cores, se possível)
- 4 Resistores de 220 ohms (ou adequados para os LEDs utilizados)
- Protoboard
- Jumpers (fios de conexão)

Solução Laboratório 02




```arduino
int LED_VERMELHO = 12;
int LED_AMARELO = 11;
int LED_VERDE = 10;
int LED_BRANCO = 9;

void setup() {
  pinMode(LED_VERMELHO, OUTPUT); //Definindo o LED que esta na porta 12 como OUTPUT (Saida)
  pinMode(LED_AMARELO, OUTPUT);
  pinMode(LED_VERDE, OUTPUT);
  pinMode(LED_BRANCO, OUTPUT);
}

void loop() {
  digitalWrite(LED_VERMELHO, HIGH); //Liga o LED vermelho
  digitalWrite(LED_AMARELO, HIGH); //Lida o LED amarelo
  delay(500);
  digitalWrite(LED_VERMELHO, LOW); //Desliga o LED Vermelho
  digitalWrite(LED_VERDE, HIGH); //Liga o LED verde
  delay(500);
  digitalWrite(LED_AMARELO, LOW); //Desliga o LED amarelo
  digitalWrite(LED_BRANCO, HIGH); //Liga o LED branco
  delay(500);
  digitalWrite(LED_BRANCO, LOW); //Desliga o LED branco
  digitalWrite(LED_VERDE, LOW); //Desliga o LED verde
}
```