



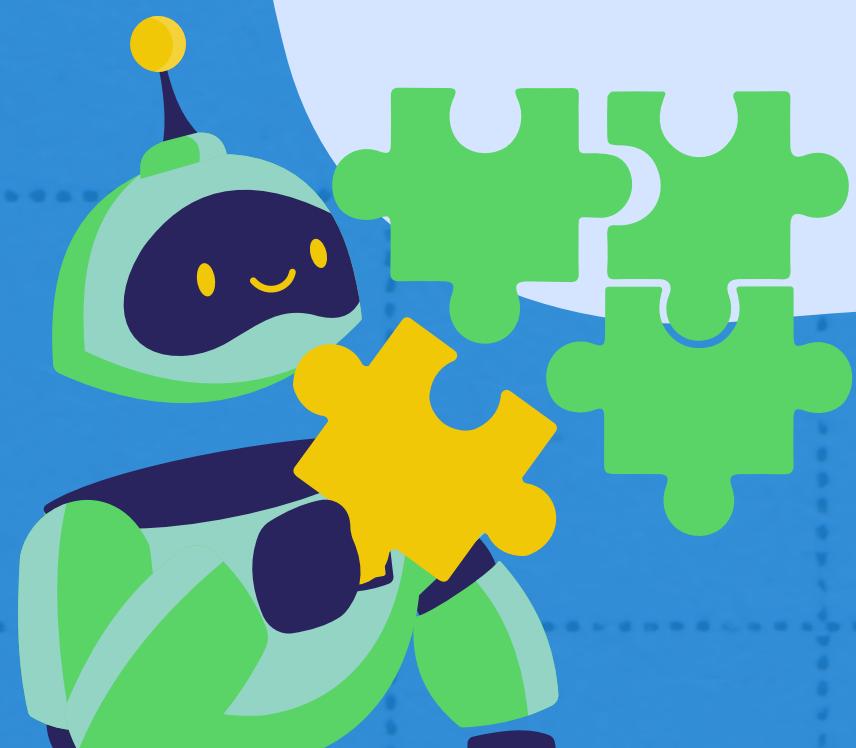
# NIVELAMIENTO DE ROBOTICA

DIA - 2

COMBU  
MOKER



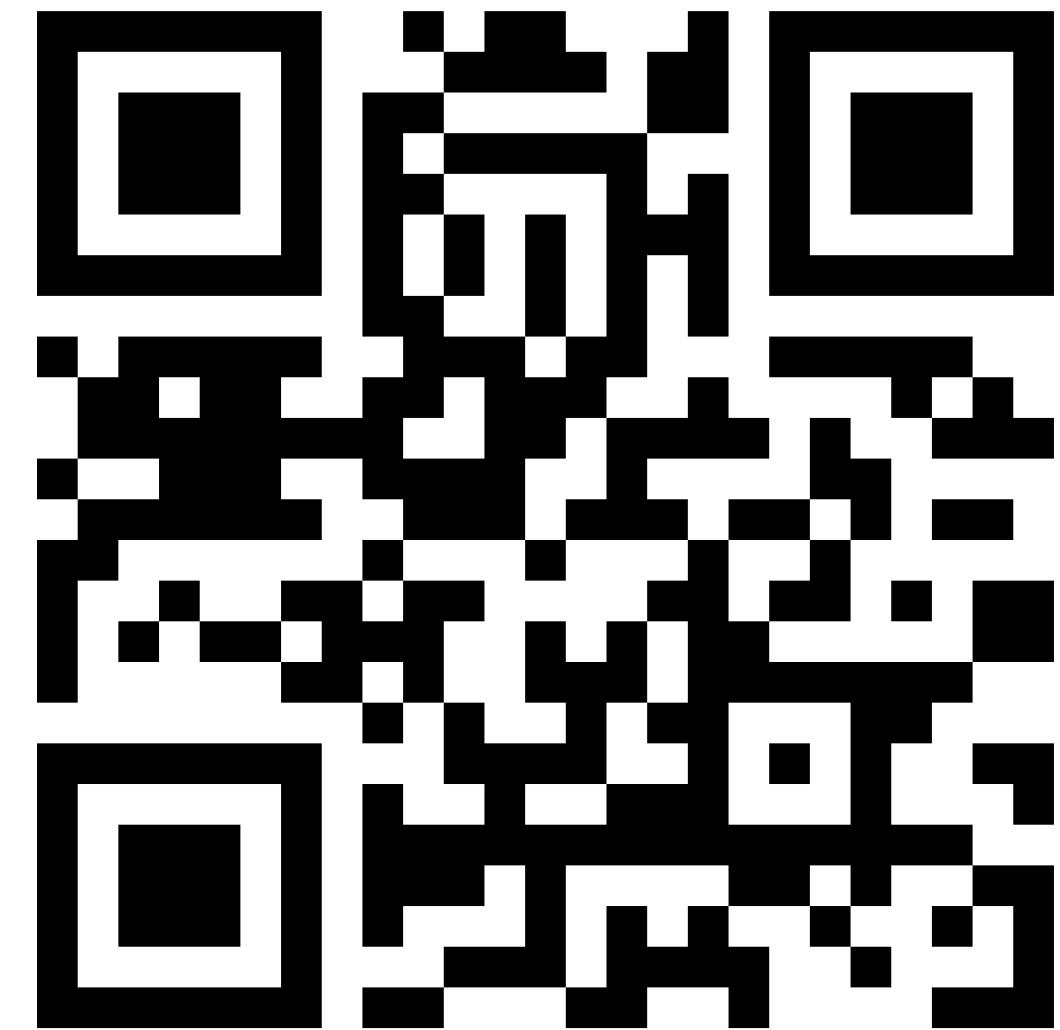
# REVISÃO



COMBU  
MOKER

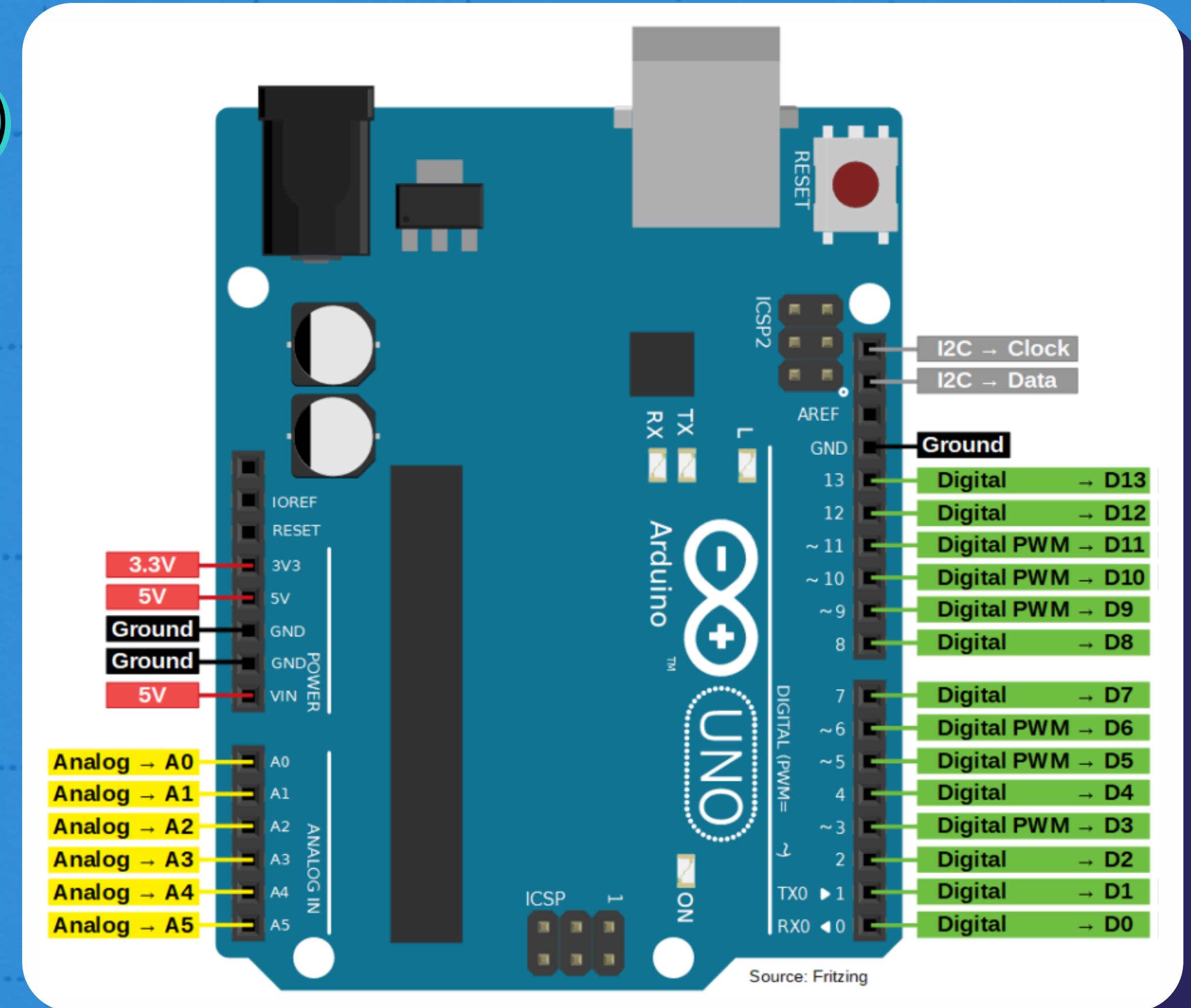
Siga nosso  
Instagram

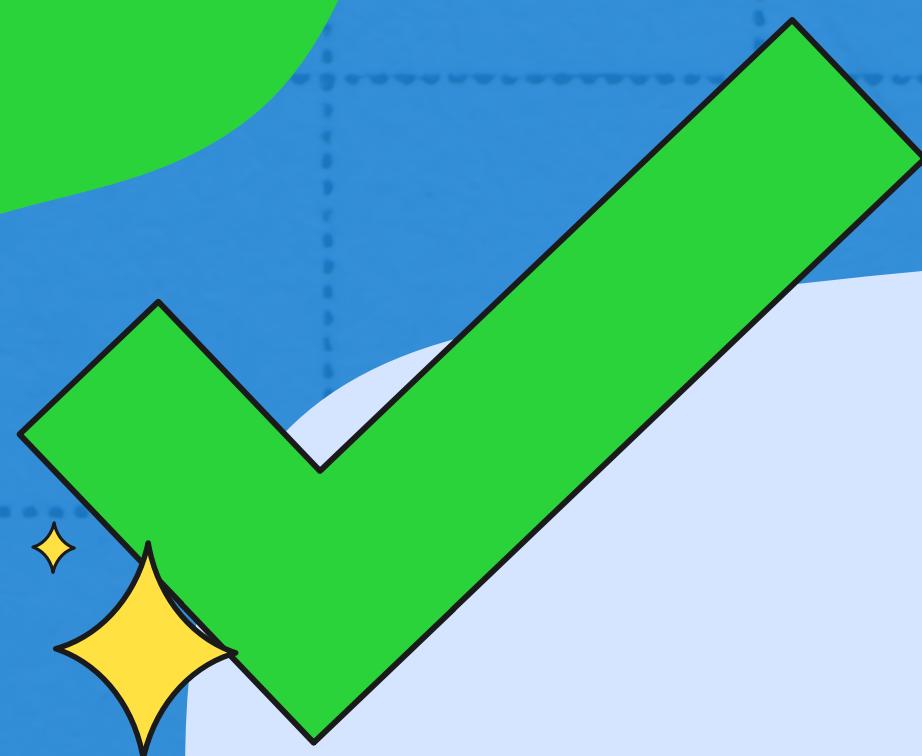
**@combumaker**



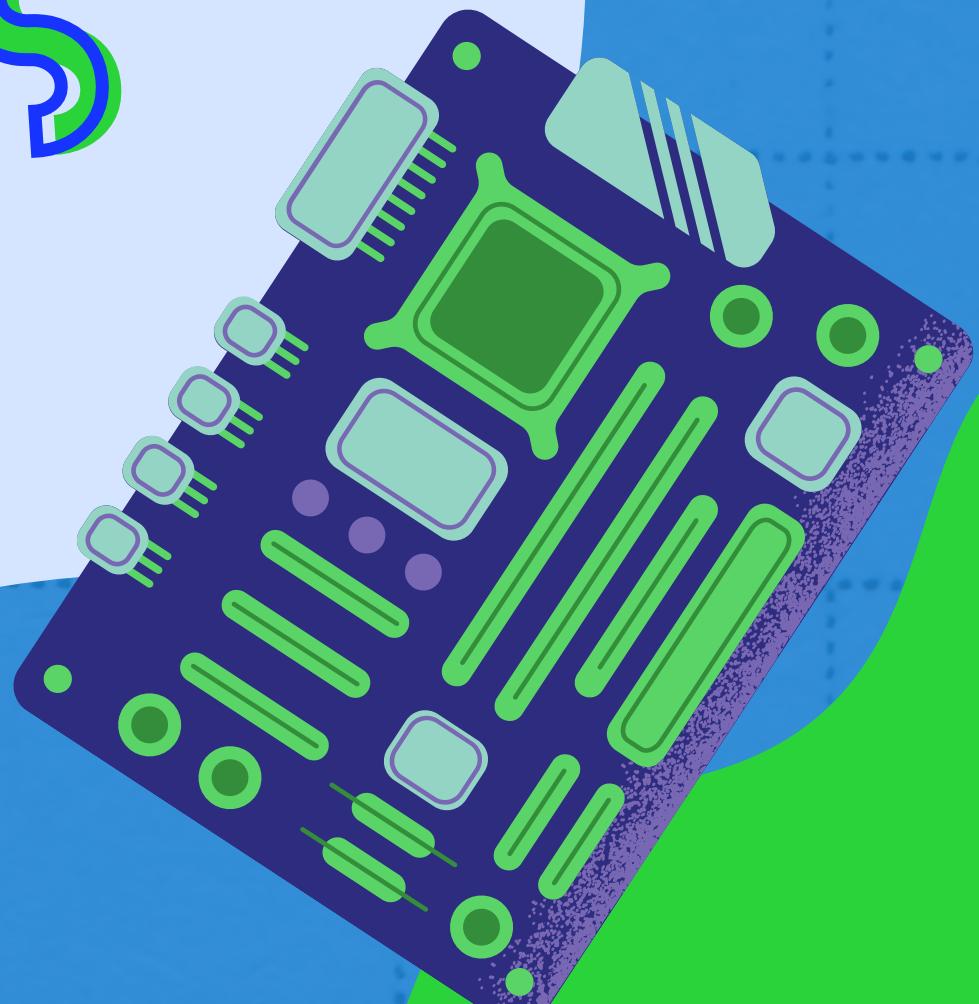
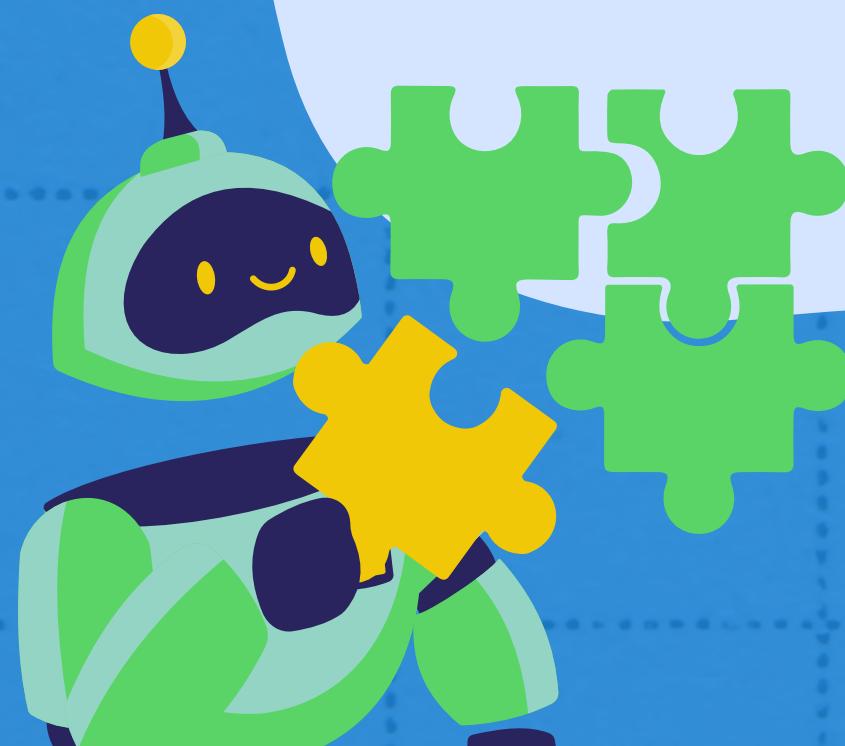
# CONHEÇA O ARDUINO:

COMBU  
MOKER

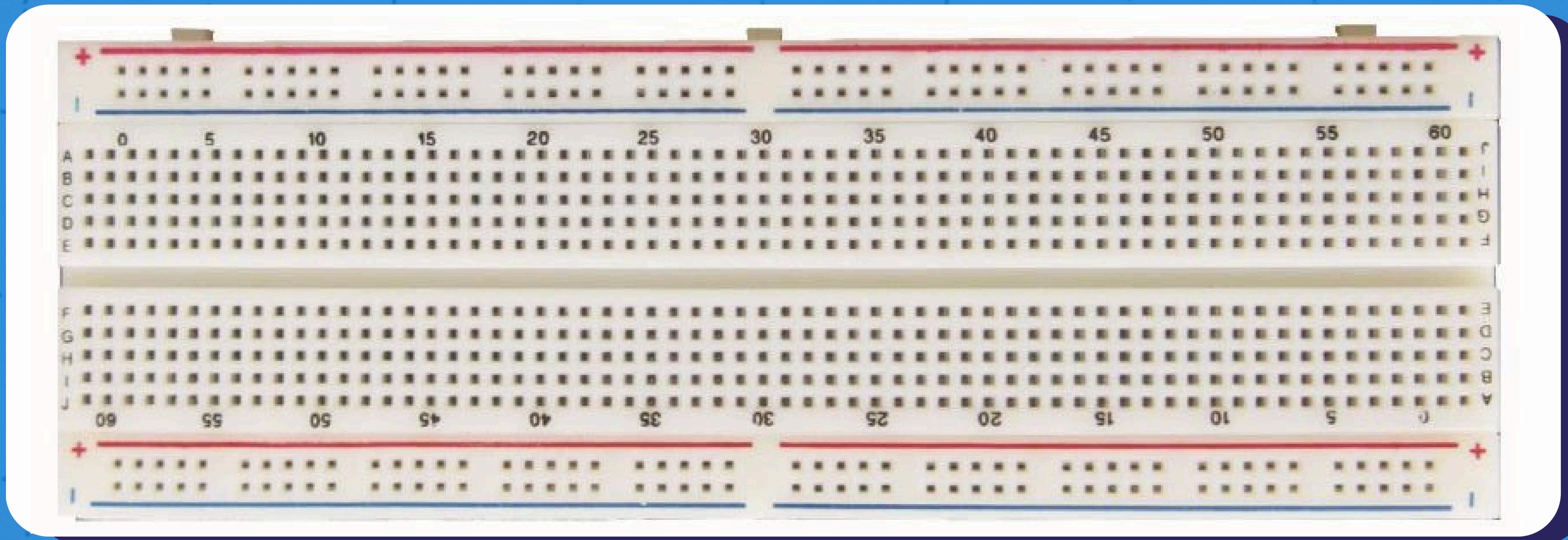




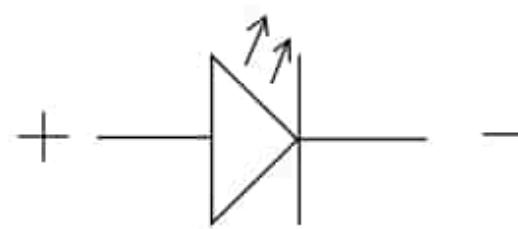
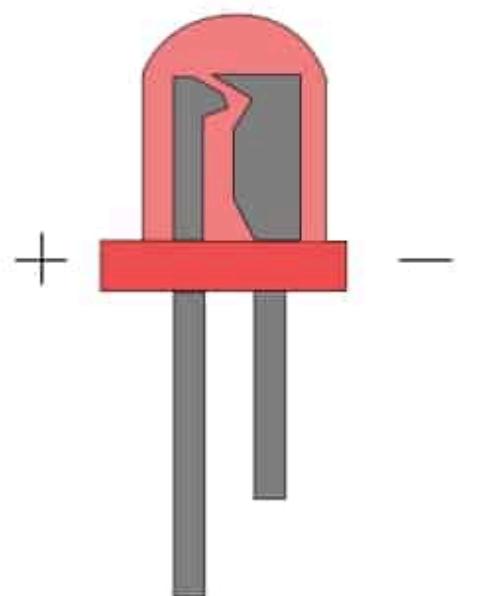
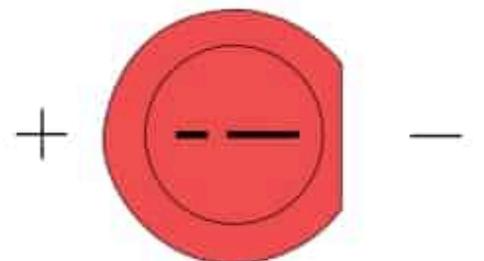
# COMPONENTES



# PROTOBOARD

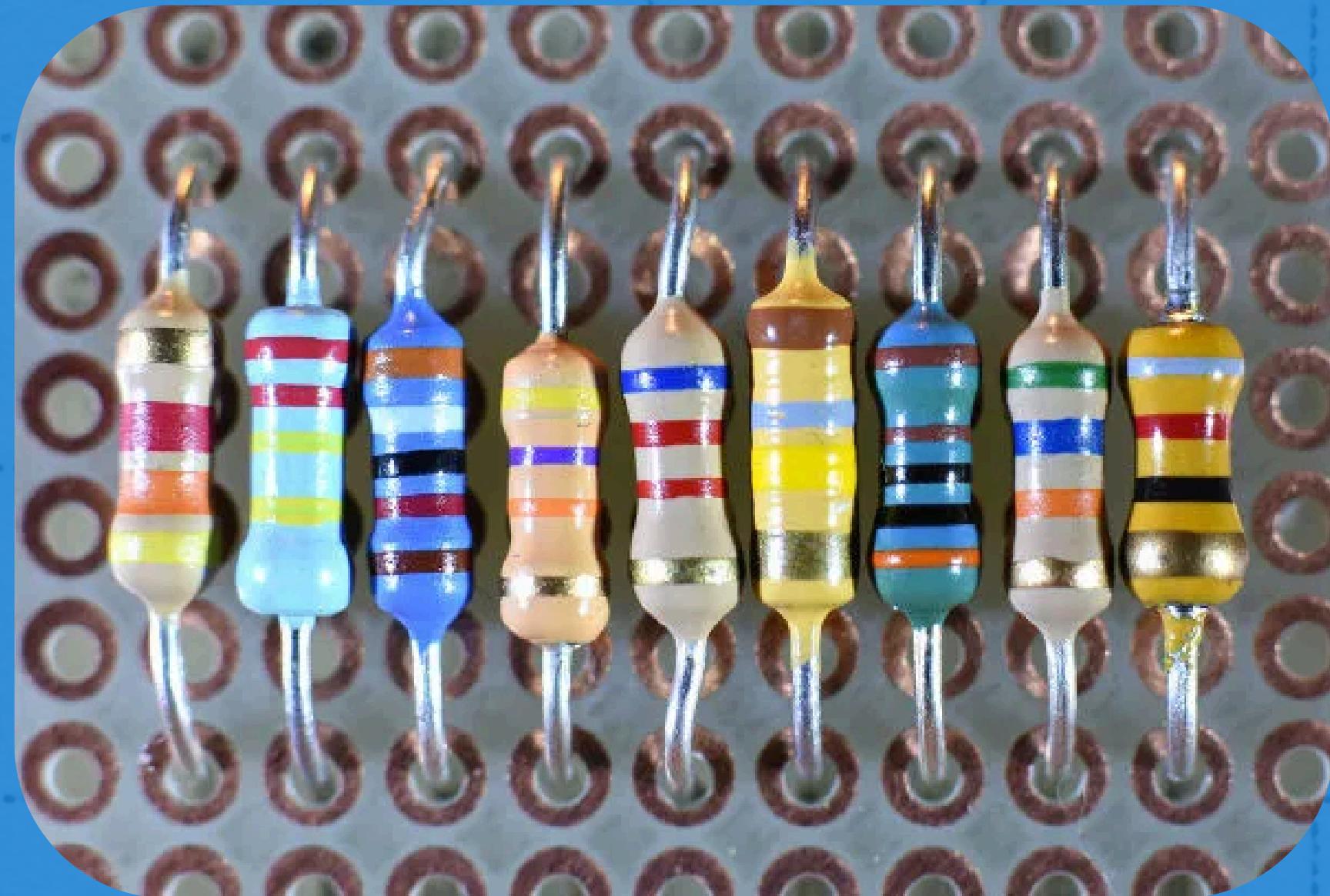


# LED



COMBU  
MOKER

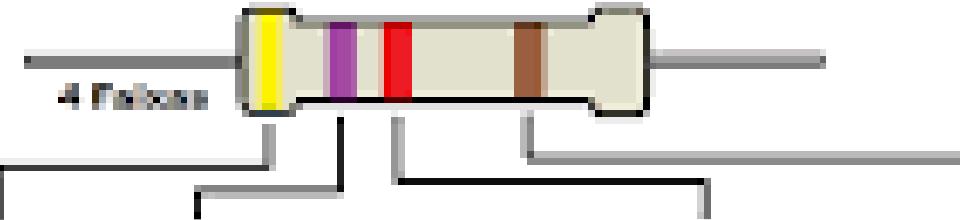
# RESISTORES



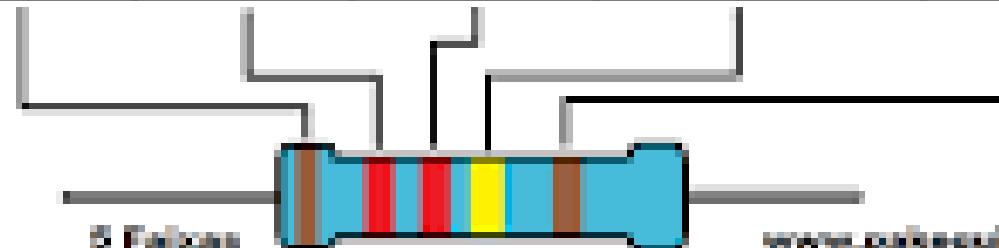
COMBU  
MOKER

# RESISTORES

Tabela: Código de Cores de Resistores

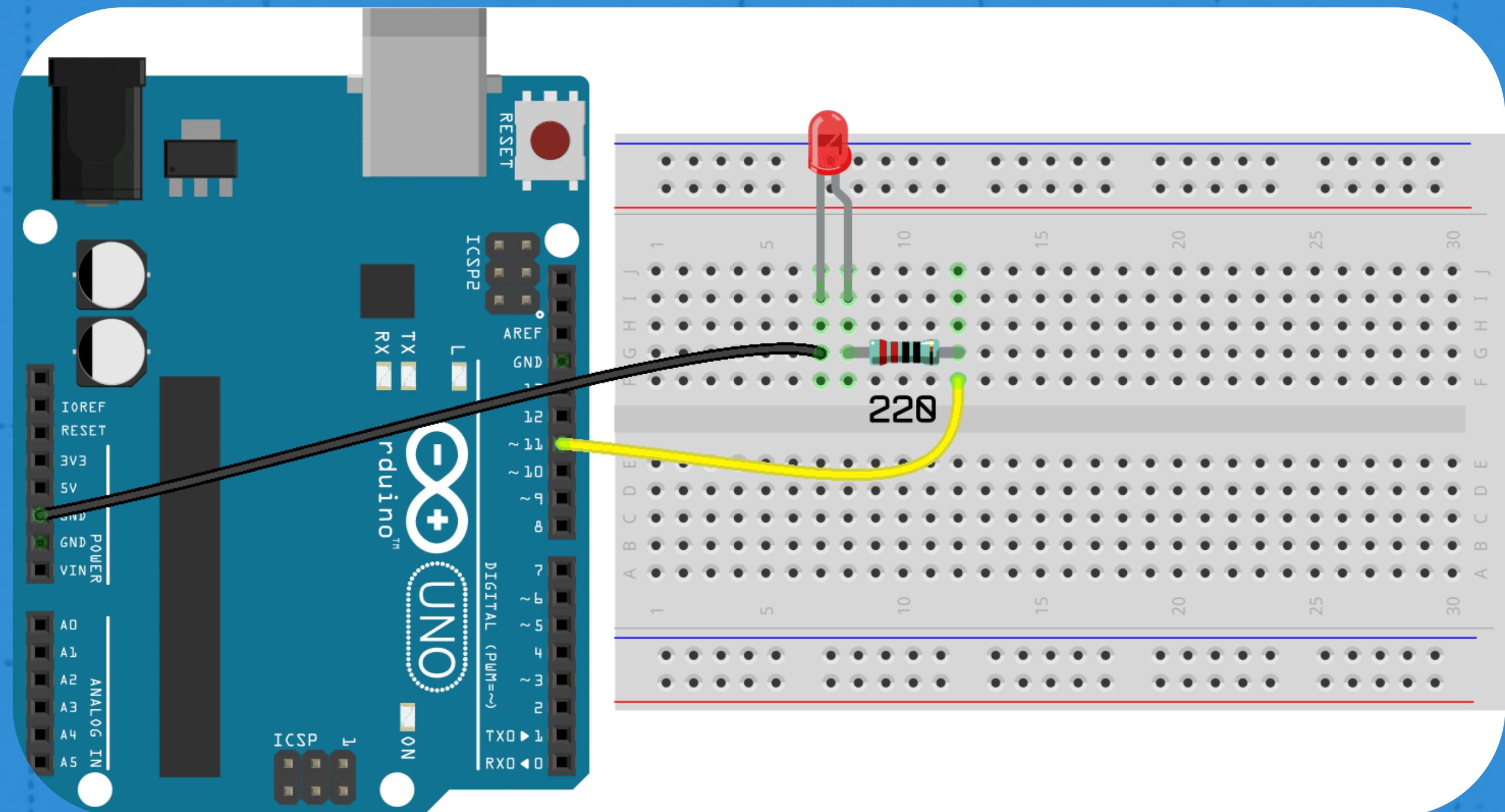


Cor:	1º Faixa:	2º Faixa:	3º Faixa:	Multiplicador:	Tolerância:
Preto	0	0	0	x10	-
Marron	1	1	1	x100	± 1%
Vermelho	2	2	2	x1000	± 2%
Laranja	3	3	3	x1kΩ	-
Amarelo	4	4	4	x10kΩ	-
Verde	5	5	5	x100kΩ	± 0,5%
Azul	6	6	6	x1MΩ	± 0,25%
Violeta	7	7	7	x10MΩ	± 0,1%
Cinza	8	8	8	-	± 0,05%
Branco	9	9	9	-	-
Dourado	-	-	-	x0,1Ω	± 5%
Prateado	-	-	-	x0,01Ω	± 10%



[www.pakequio.com.br](http://www.pakequio.com.br)

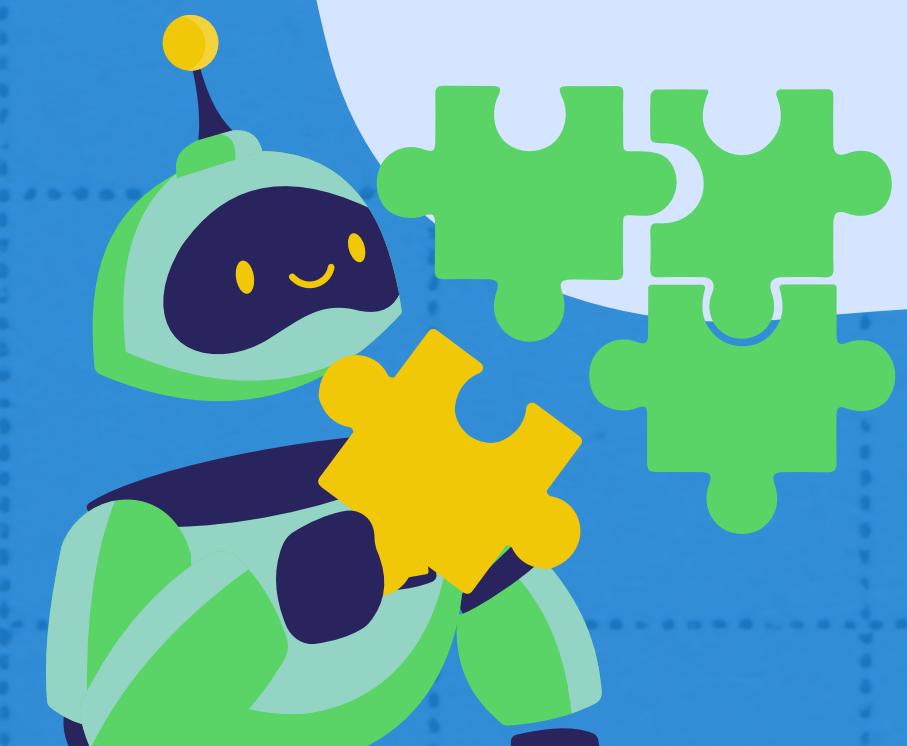
# EXEMPLO



COMBU  
MOKER



# LINGUAGEM ARDUINO



COMBU  
MOKER

# TIPOS DADOS

**INT:** Um tipo de dado inteiro que armazena valores numéricos inteiros;

**FLOAT:** Um tipo de dado de ponto flutuante usado para representar números reais (números com casas decimais);

**Boolean:** Um tipo de dado que armazena valores lógicos, ou seja, true (verdadeiro) ou false (falso);

# TIPOS DADOS

**CARACTERES:** É a unidade básica de texto em um sistema de codificação. Por exemplo, 'a', 'B', '1', '!'.  
\*\*\*

**STRING:** É uma sequência de caracteres. Uma coleção ordenada de caracteres que formam uma unidade de texto. Por exemplo, "Olá, mundo!".



# OPERADORES ARITMÉTICOS

- % (resto);
- \* (multiplicação);
- + (adição);
- (subtração);
- / (divisão);
- = (operador de atribuição).



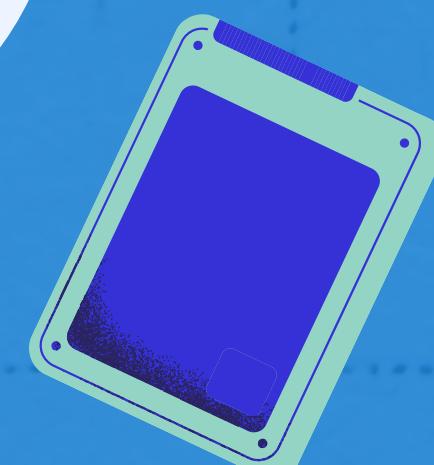
# CONDICIONALES

# CONDICIONAIS

## IF

A estrutura if é usada para executar um bloco de código se uma condição especificada for verdadeira. Se a condição não for verdadeira, o bloco de código dentro do if não será executado.

Pode ser seguido opcionalmente por else ou else if para definir o que fazer quando a condição não for verdadeira.



# CONDICIONAIS

## ELSE IF:

A estrutura else if permite verificar múltiplas condições em sequência. Se a condição dentro do if for falsa, a condição dentro do else if será verificada.



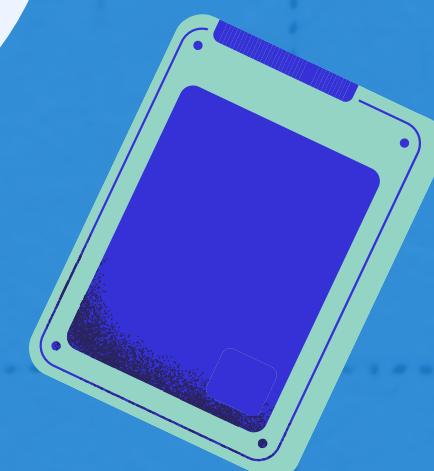
# CONDITIONALS

```
● ● ●  
1 void loop() {  
2     int buttonState = digitalRead(buttonPin);  
3  
4     if (buttonState == LOW) {  
5         digitalWrite(ledPin, HIGH);  
6     } else {  
7         digitalWrite(ledPin, LOW);  
8     }  
9 }
```

# CONDICIONAIS

## SWITCH:

A estrutura switch é usada quando há várias condições possíveis para testar. Ela permite selecionar um dos muitos blocos de código para serem executados.



# CONDICIONAIS

```
● ● ●  
1 switch (buttonState) {  
2   case HIGH:  
3     digitalWrite(ledPin, HIGH);  
4     Serial.println("LED ligado");  
5     break;  
6  
7   case LOW:  
8     digitalWrite(ledPin, LOW);  
9     Serial.println("LED desligado");  
10    break;  
11 }
```



# LACOS DE REPETIÇÃO

COMBU  
MOKER

# LAÇOS

## FOR:

O laço for é utilizado quando se sabe exatamente quantas vezes o código precisa ser repetido. Ele consiste em três partes: uma inicialização, uma condição de continuação e uma expressão de incremento (ou decremento). O bloco de código dentro do for é executado repetidamente enquanto a condição for verdadeira.



# LACOS



```
1  for (int i = 0; i <= rodada; i++) {  
2      digitalWrite(sequencia[i], HIGH);  
3      delay(500);  
4      digitalWrite(sequencia[i], LOW);  
5      delay(500);  
6  }  
7
```

# LAÇOS

## WHILE:

O laço while é usado quando você precisa repetir um bloco de código enquanto uma condição específica for verdadeira. O bloco de código é executado repetidamente enquanto a condição for verdadeira.



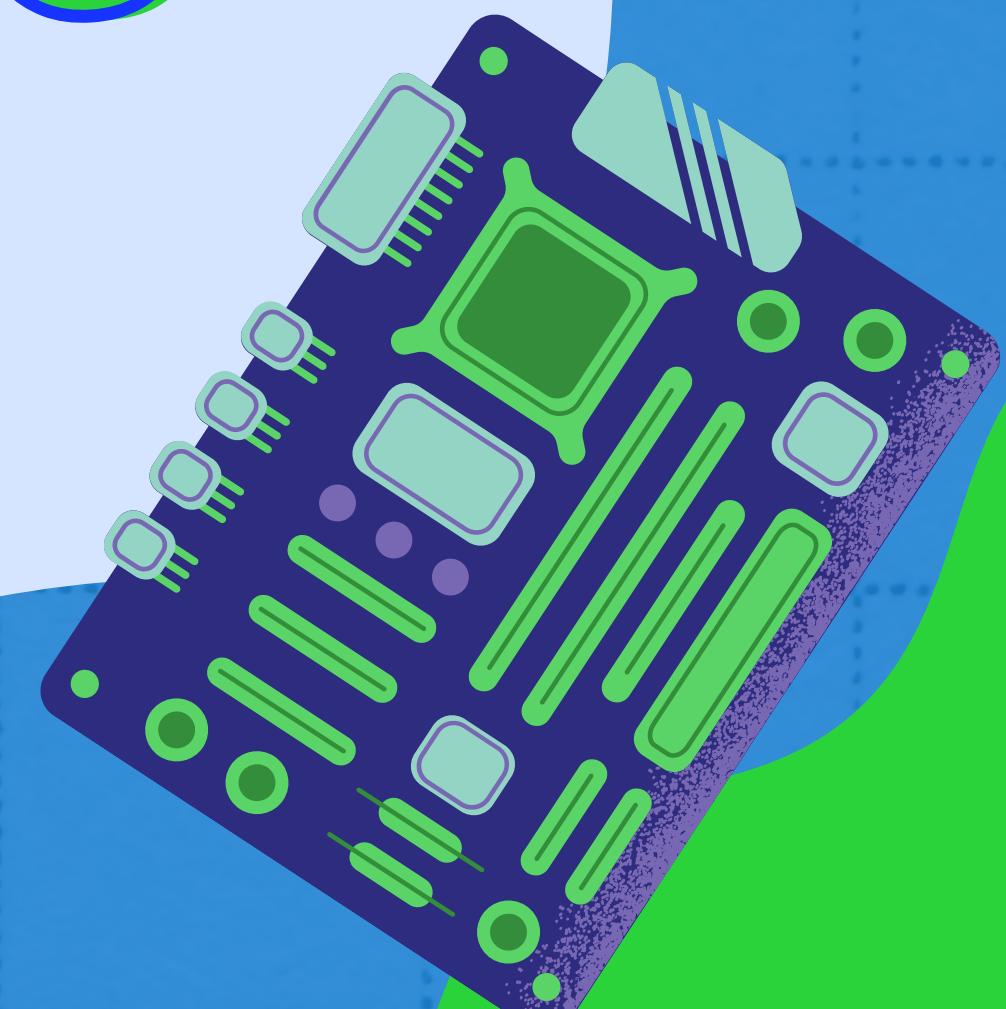
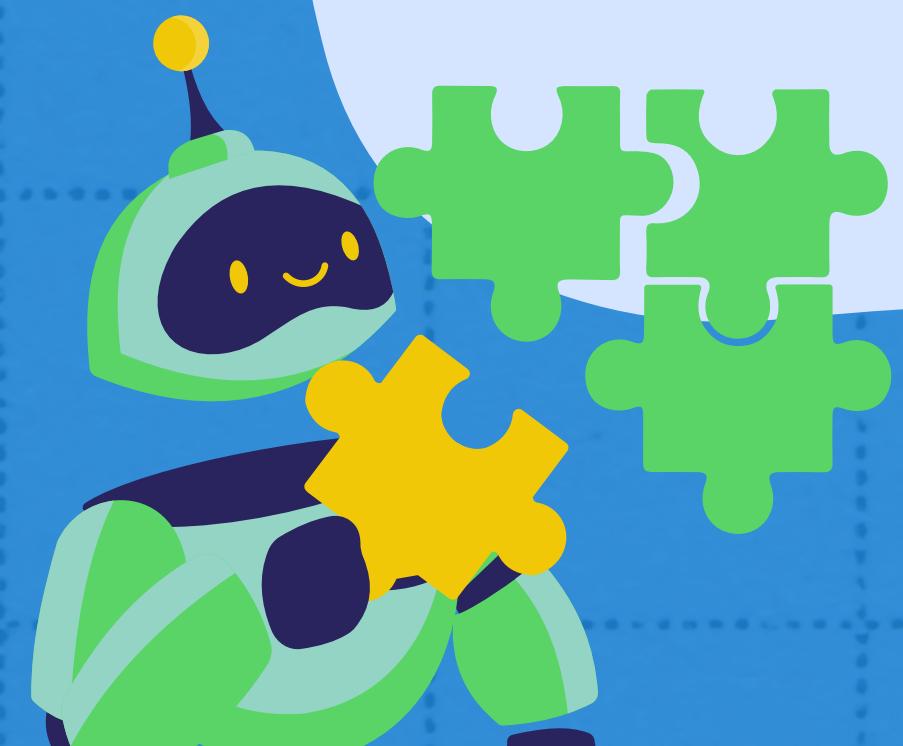
# LACOS

5

```
● ● ●  
1 while (true) {  
2     // Liga o LED  
3     digitalWrite(LED_PIN, HIGH);  
4  
5     // Aguarda 1 segundo  
6     delay(1000);  
7  
8     // Desliga o LED  
9     digitalWrite(LED_PIN, LOW);  
10  
11    // Aguarda 1 segundo  
12    delay(1000);  
13}
```



# PROGRAMAR NO ARDUINO



COMBU  
MOKER

# LINGUAGEM ARDUINO

## Setup:

- **Execução:** É executada apenas uma vez, quando o Arduino liga ou reinicia.
- **Objetivo:** Configurar o hardware do Arduino, como:
  - Definir pinos como entrada ou saída
  - Inicializar bibliotecas
  - Configurar comunicação serial
  - Definir valores iniciais para variáveis



# LINGUAGEM ARDUINO

## Loop:

- **Execução:** É executada continuamente, em um loop infinito, enquanto o Arduino estiver ligado.
- **Objetivo:** Conter o código que define o comportamento do seu programa, como:
  - Ler dados de sensores
  - Controlar LEDs, motores e outros dispositivos
  - Fazer cálculos
  - Realizar comunicação serial

# LINGUAGEM ARDUINO



```
1 int pare = 8;  
2 int atencao = 9;  
3 int siga = 10;
```

**Pinagem:** Para definir as portas, basta declarar o tipo de variável em seguida o nome e em seguida atribuir a variável ao pino que será usado

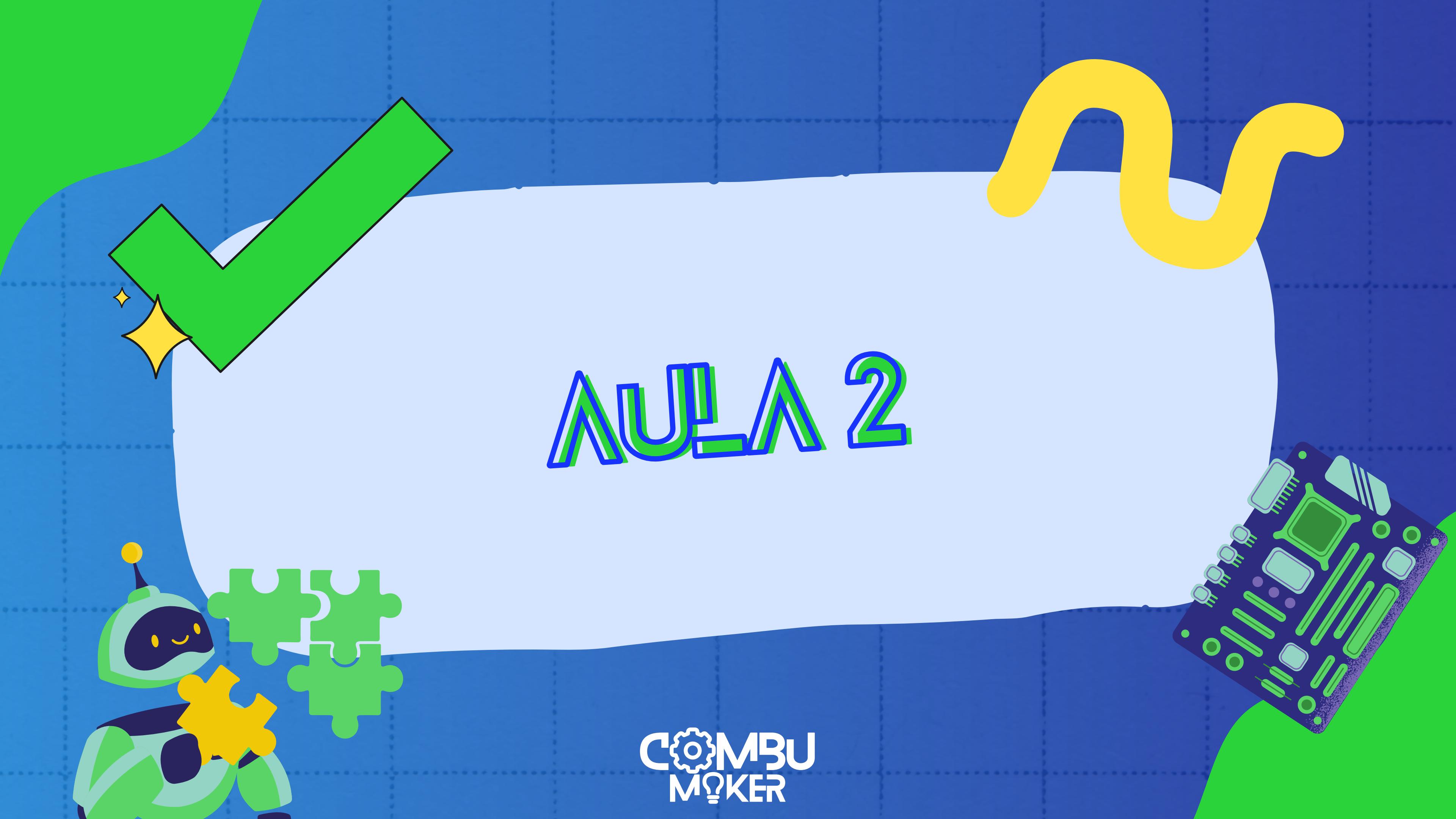
# LINGUAGEM ARDUINO



```
1 pinMode(pare, OUTPUT);  
2 pinMode(atencao, OUTPUT);  
3 pinMode(siga, OUTPUT);
```

## Configuração dos pinos

**(pinMode):** Para começar a usar um pino específico, você precisa configurá-lo como entrada (input) ou saída (output) usando a função **pinMode();**



# AULA 2

COMBU  
MOKER



# COMPARAÇÃO: PYTHON E ARDUINO

# PYTHON

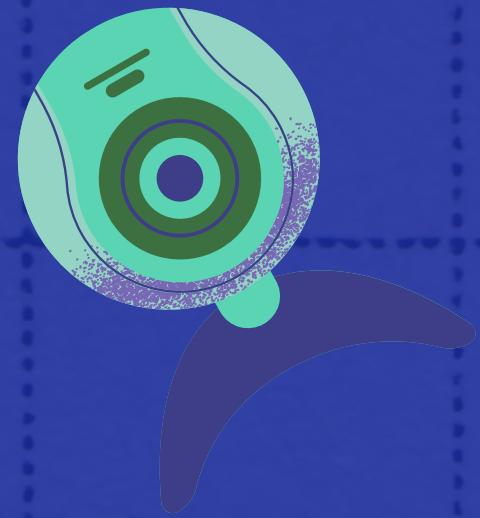
- É uma linguagem de programação conhecida por sua sintaxe simples e legibilidade;
- É uma **linguagem de tipagem dinâmica**, o que significa que a sintaxe não exige que o tipo de dado seja informado de forma explícita;
- Possui uma maior flexibilidade e é mais rápida para desenvolver um algoritmo, porém é mais demorada para a execução do código.
- As variáveis podem ser reatribuídas para armazenar diferentes tipos de dados.

# PYTHON

```
● ● ●  
1 a = 2  
2 b = 6  
3  
4 c = a + b  
5 print(c)  
6  
7 c = 1  
8 print(c)
```

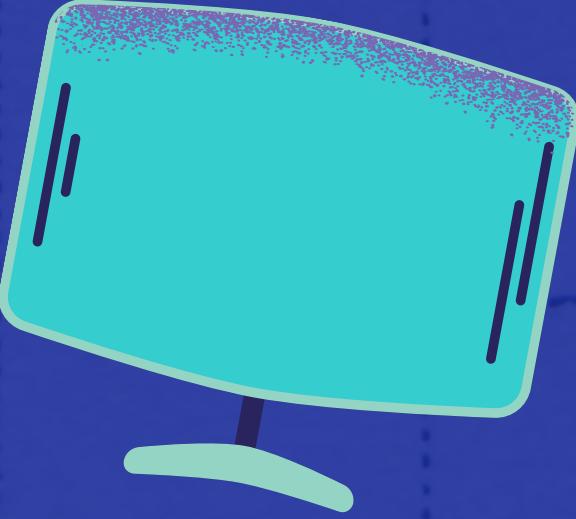
# ARDUINO

- Utiliza uma sintaxe semelhante à linguagem C/C++, com estruturas de controle de fluxo familiares, como loops, condicionais e funções;
- É uma **linguagem de tipagem estática**, ou seja o programador precisa explicitar o tipo de dado usado no código;
- Possui um melhor performance durante a execução do código, mas pode ser mais demorada e complexa para desenvolver um algorítimo.



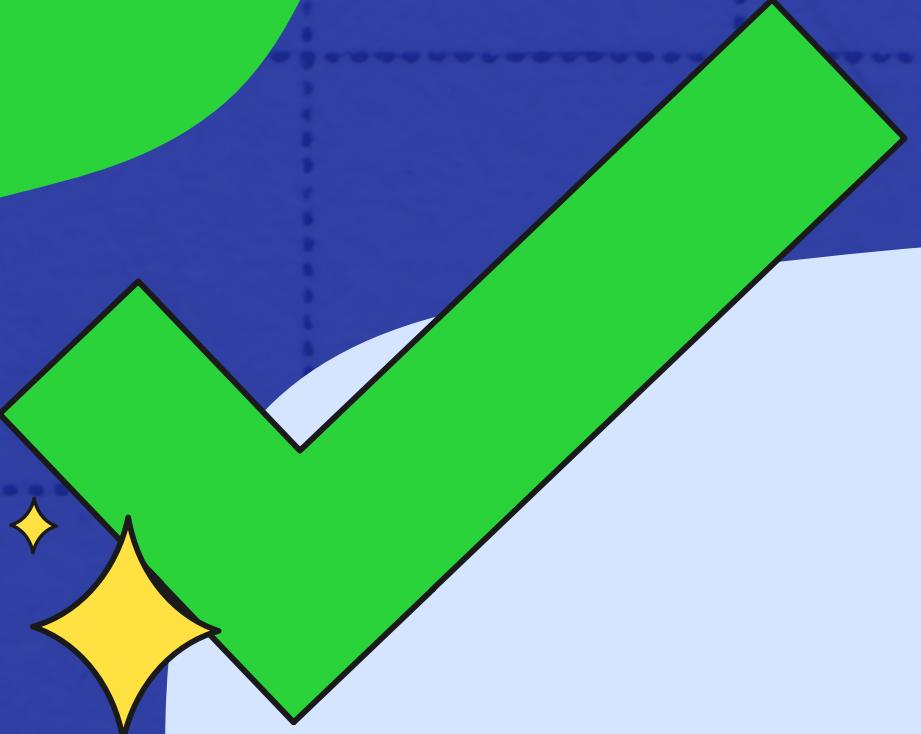
# ARDUINO

eee



```
1 int ledResistor = 8;
2 int ledSemResistor = 7;
3
4 void setup0 {
5   pinMode(ledResistor,OUTPUT);
6   pinMode(ledSemResistor,OUTPUT);
7 }
8 void ligaCR(){
9   digitalWrite(ledResistor, HIGH);
10}
11 void ligaSR(){
12   digitalWrite(ledSemResistor, HIGH);
13}
14 void loop0 {
15   ligaCR();
16   delay(5000);
17   ligaSR();
18   delay(5000);
19 }
```

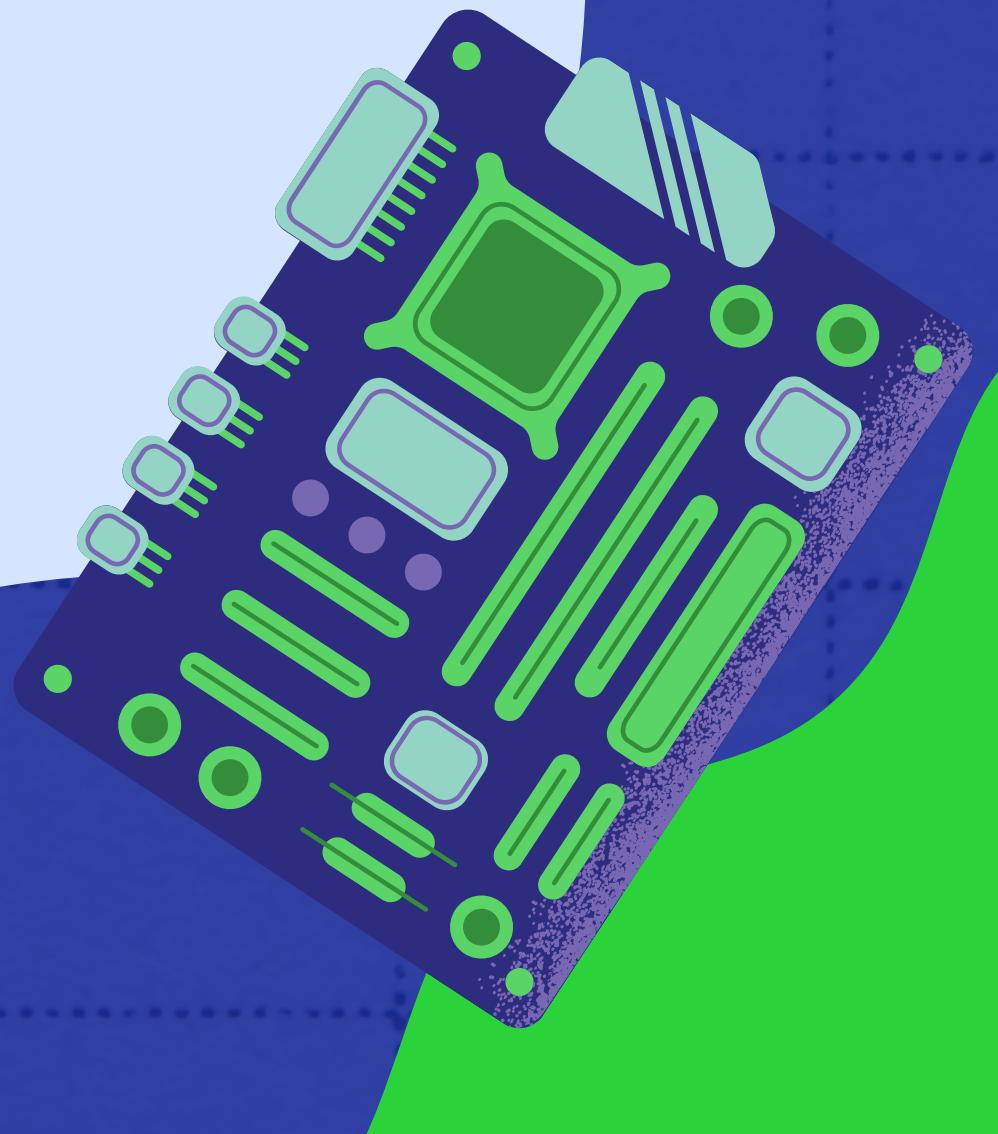




SINAL PWM



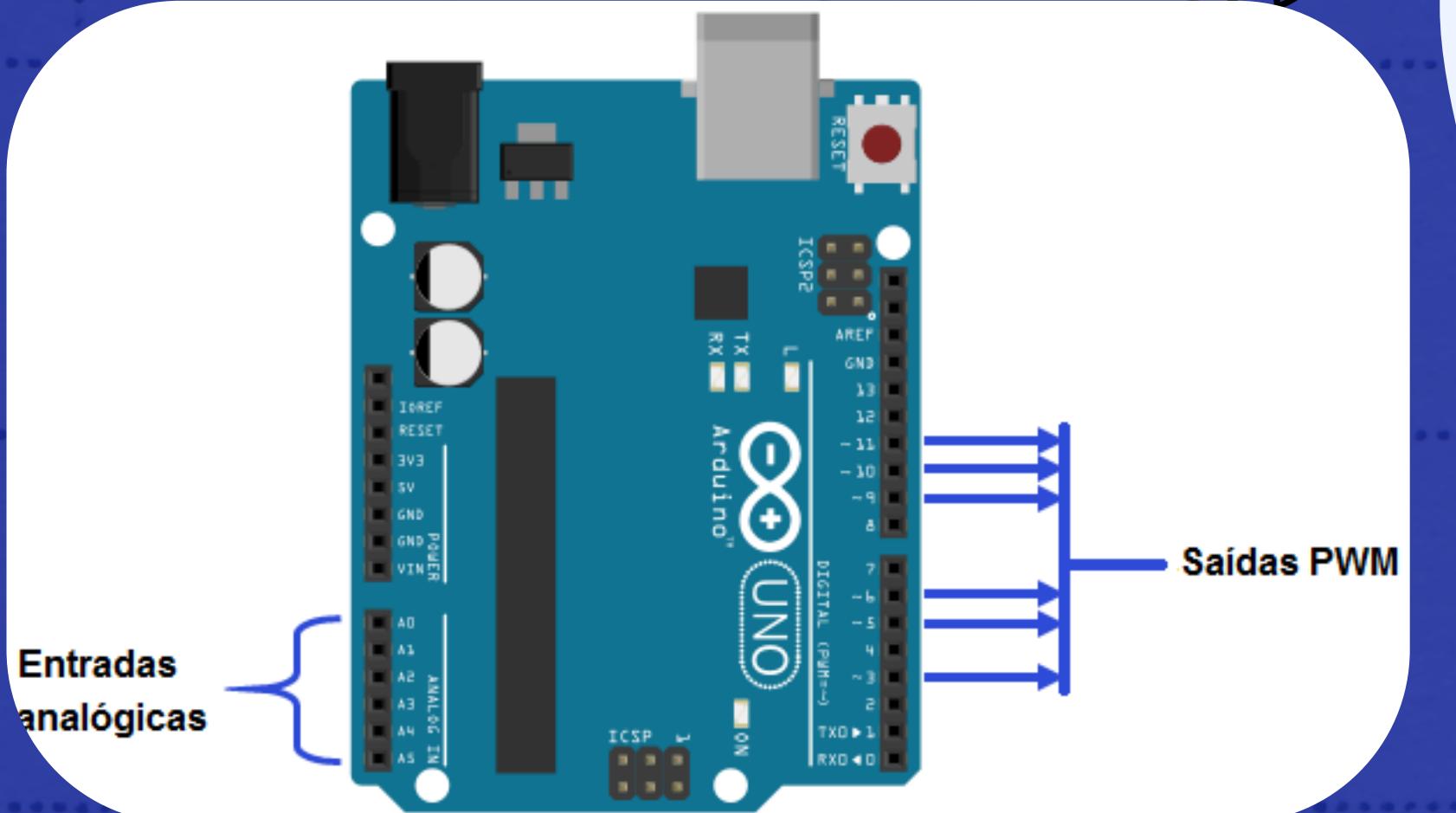
COMBU  
MOKER



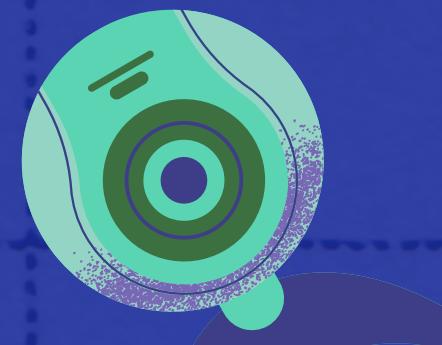
# SINAL PWM

## O que é?

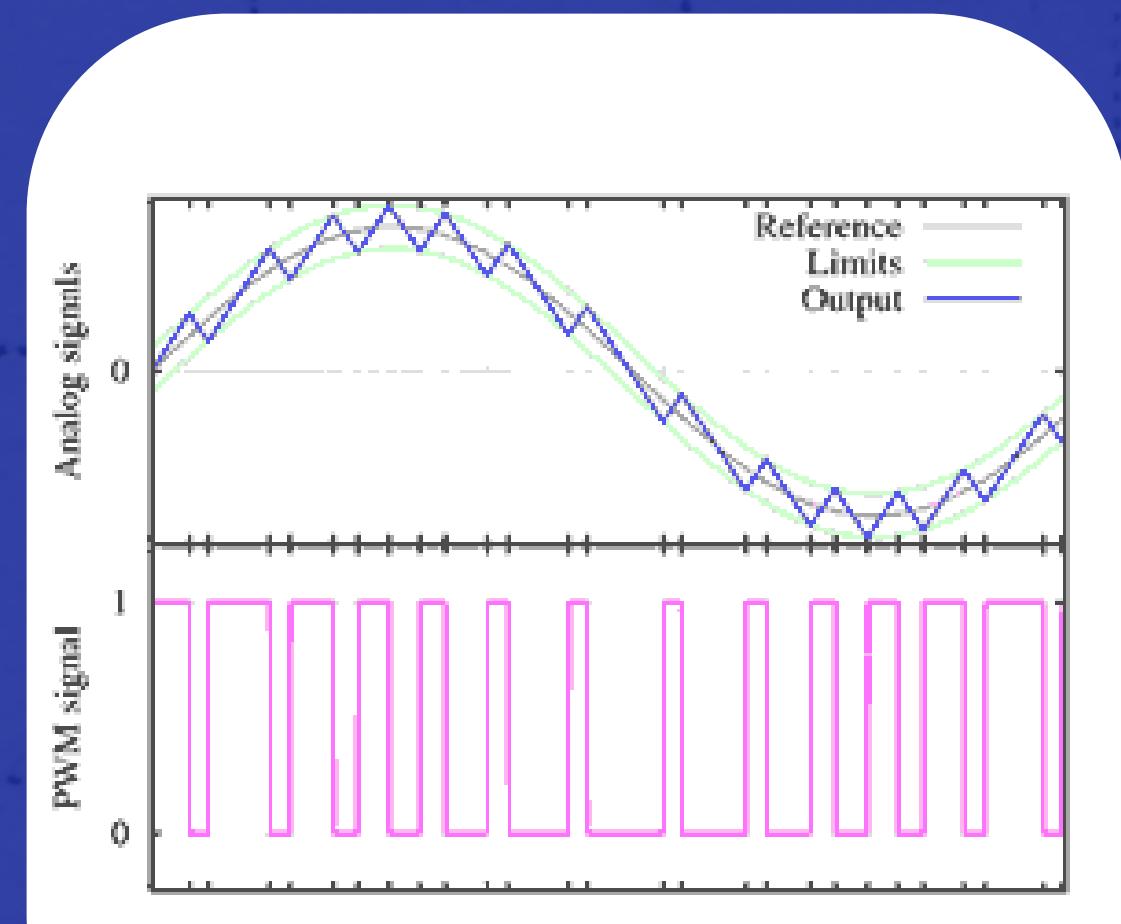
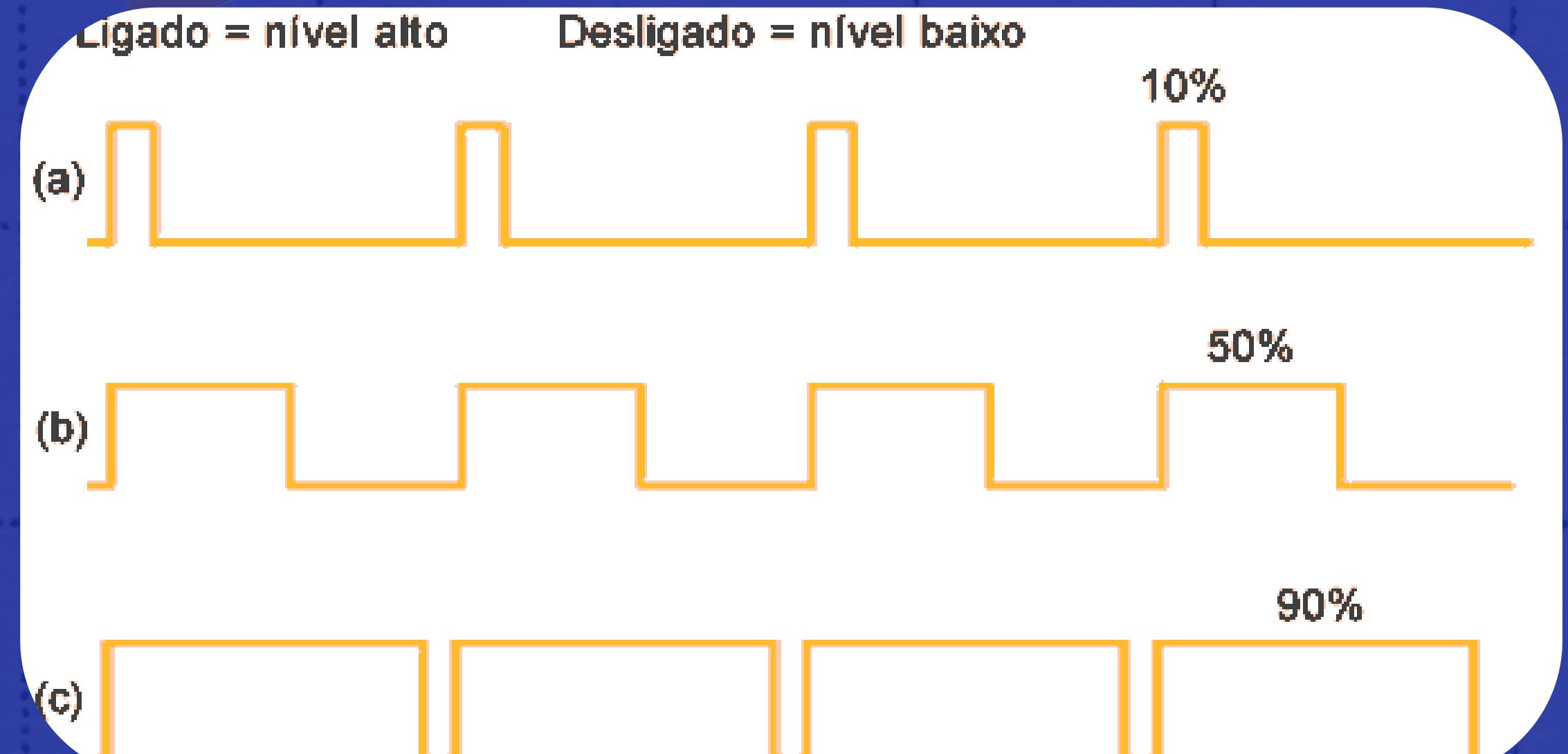
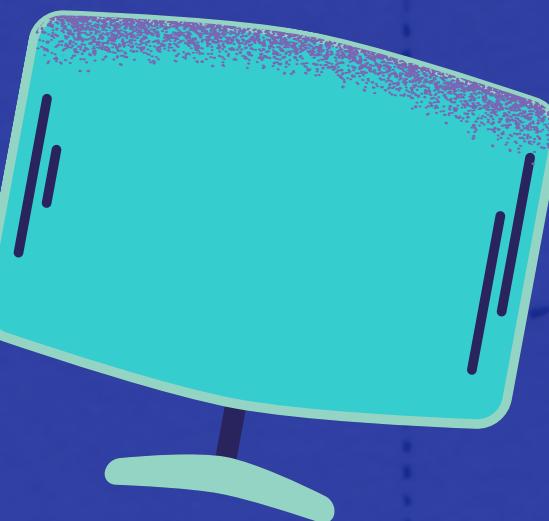
- Pode ser traduzida como "Modulação por Largura de Pulso".
- No Arduino, você pode usar saídas PWM em alguns dos pinos digitais para gerar sinais PWM.
- Permite um controle preciso da potência aplicada a uma carga, minimizando o desperdício de energia.



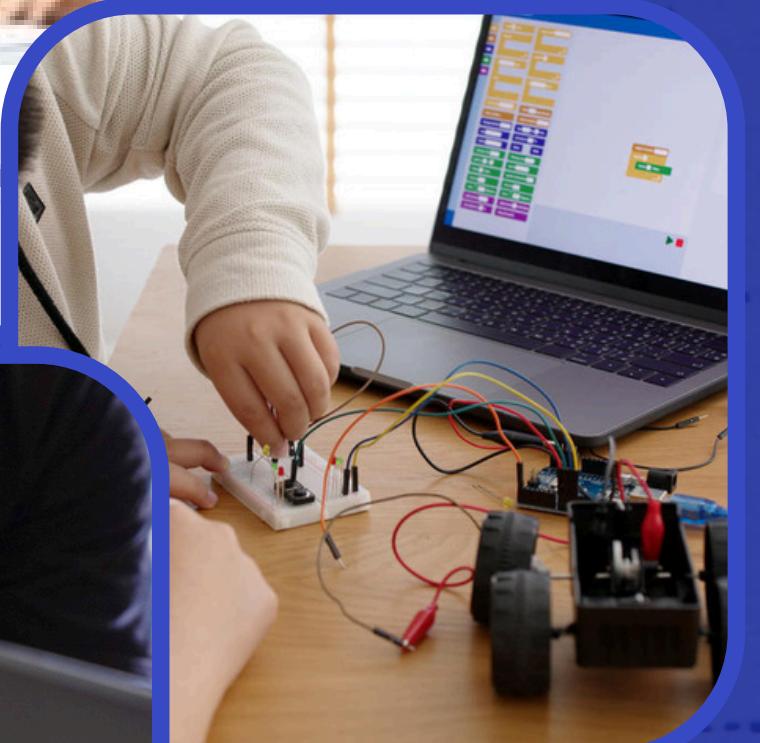
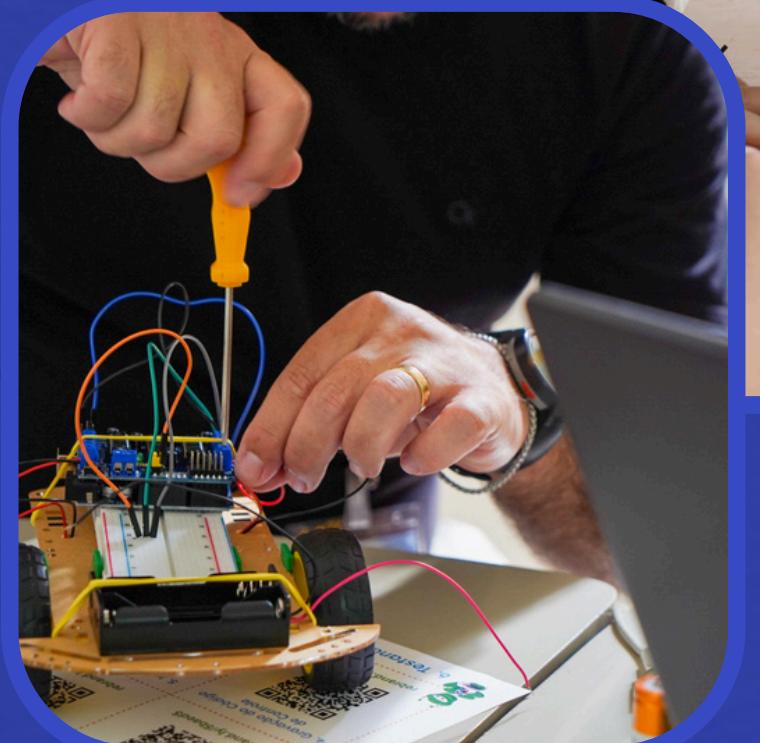
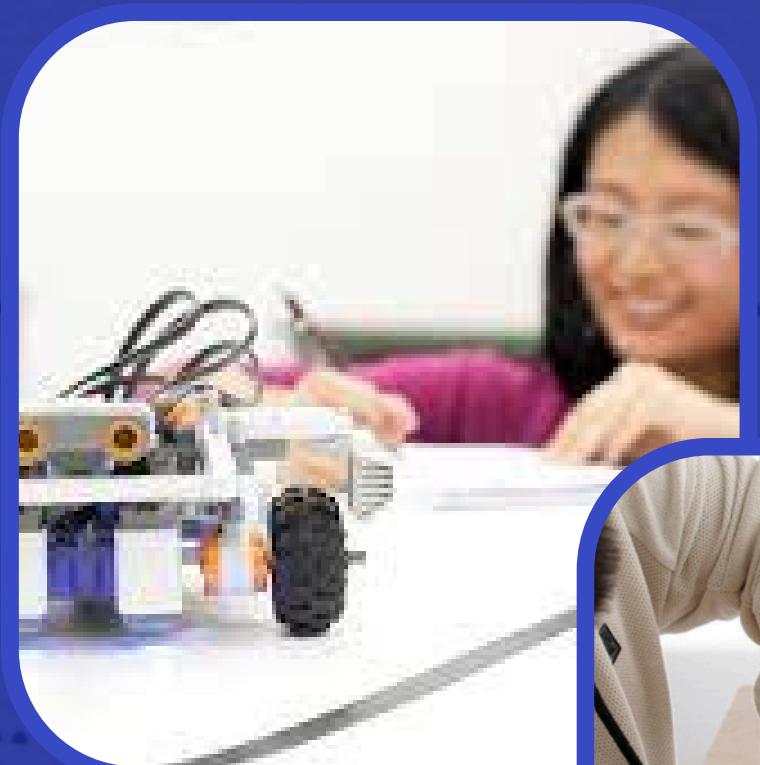
# ARDUINO



eee



# SINAL PWM



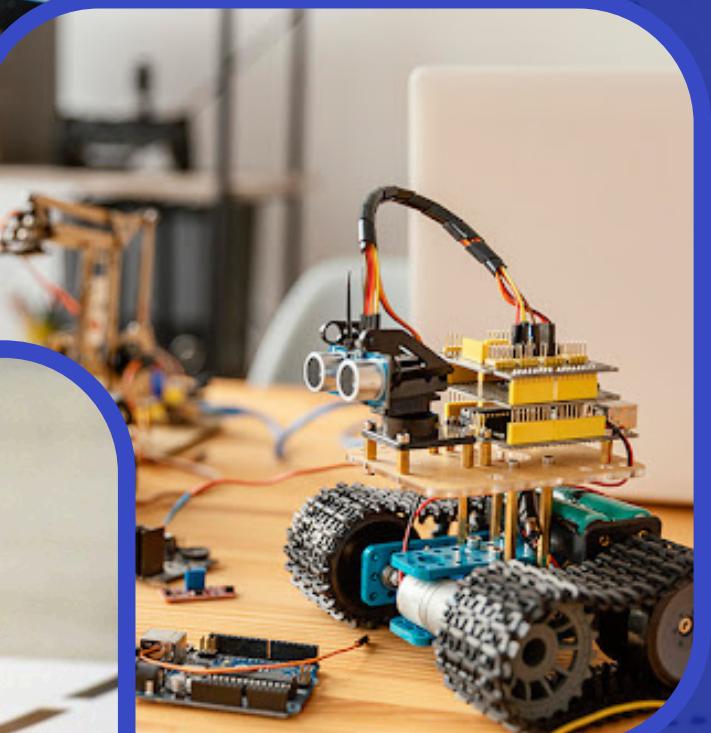
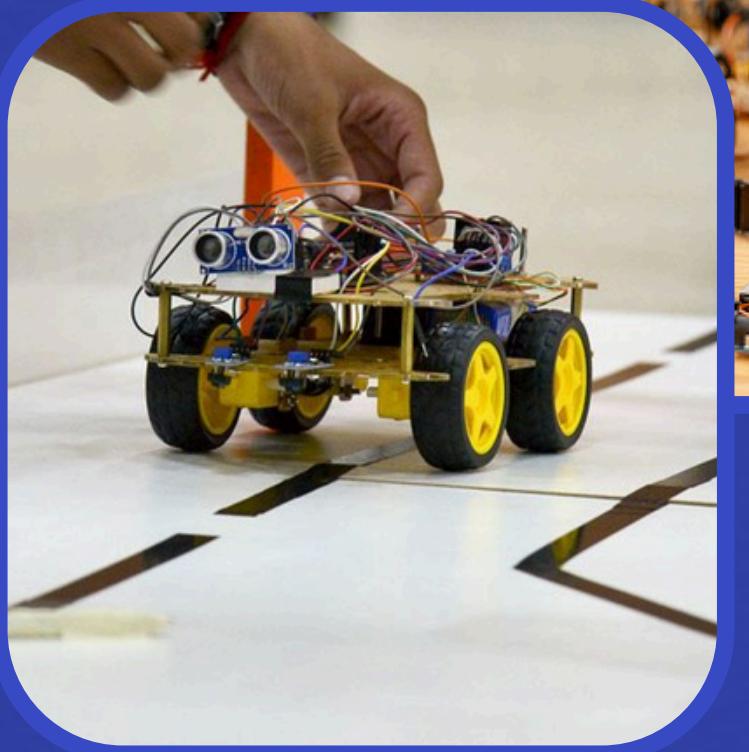
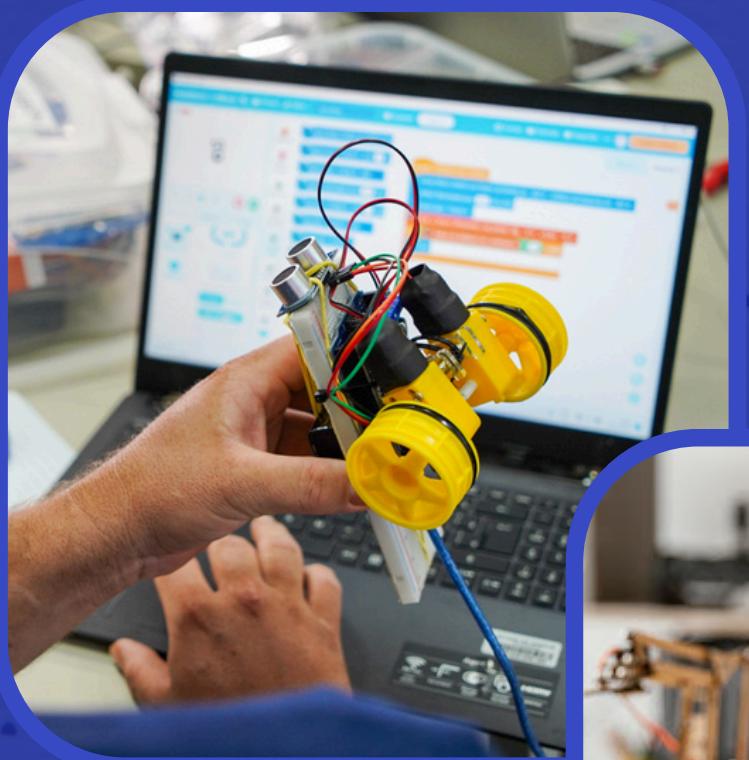
## Eficiência energética:

- O PWM permite um controle preciso da potência aplicada a uma carga, minimizando o desperdício de energia.

## Precisão e controle:

- A modulação por largura de pulso oferece um controle mais preciso da velocidade de motores e outros dispositivos em comparação com métodos analógicos tradicionais.

# SINAL PWM



## Versatilidade:

- O sinal PWM pode ser utilizado em uma ampla gama de aplicações, desde o controle de LEDs e displays até a geração de sons e comunicação digital.

## Simplicidade de implementação:

- Diversos microcontroladores e outros dispositivos digitais possuem pinos específicos para geração de sinais PWM, facilitando a integração em projetos.



# SINAL PWM

- Para “ativar” o sinal PWM no arduino é necessário usar a função **analogWrite()**, no código, o qual recebe dois argumentos, que são, a porta que será utilizada pelo componente e em seguida o ciclo de trabalho que será utilizado pela porta.
- As portas PWM aceitam ciclos de trabalho de 0 até 256, 0 sendo o desempenho mínimo do componente de 256 sendo o desempenho máximo.
- É importante notar que, apesar do nome, a saída de **analogWrite()** é digital e não produz uma saída analógica real.

# ANALOGWRITE()

- A função **analogWrite()** é usada para controlar a frequência do pulso de energia que vai para um pino digital;
- A função **analogWrite()** funciona criando um sinal PWM no pino digital especificado;
- O ciclo de trabalho é especificado como um valor entre 0 e 255, sendo 0 o sinal está sempre desligado, enquanto um valor de 255 significa que o sinal está sempre ligado na frequência máxima;
- Valores intermediários definem o ciclo de trabalho proporcionalmente.

# EXEMPLO

```
1 analogWrite(ledPin, 132);  
2 delay(25);
```

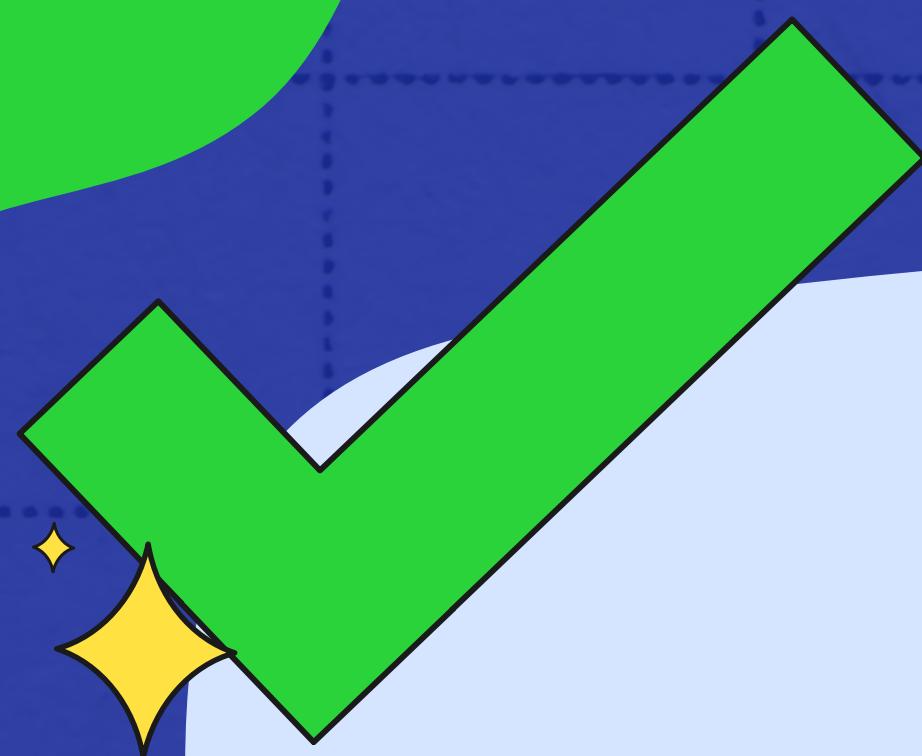


# DESAFIOS

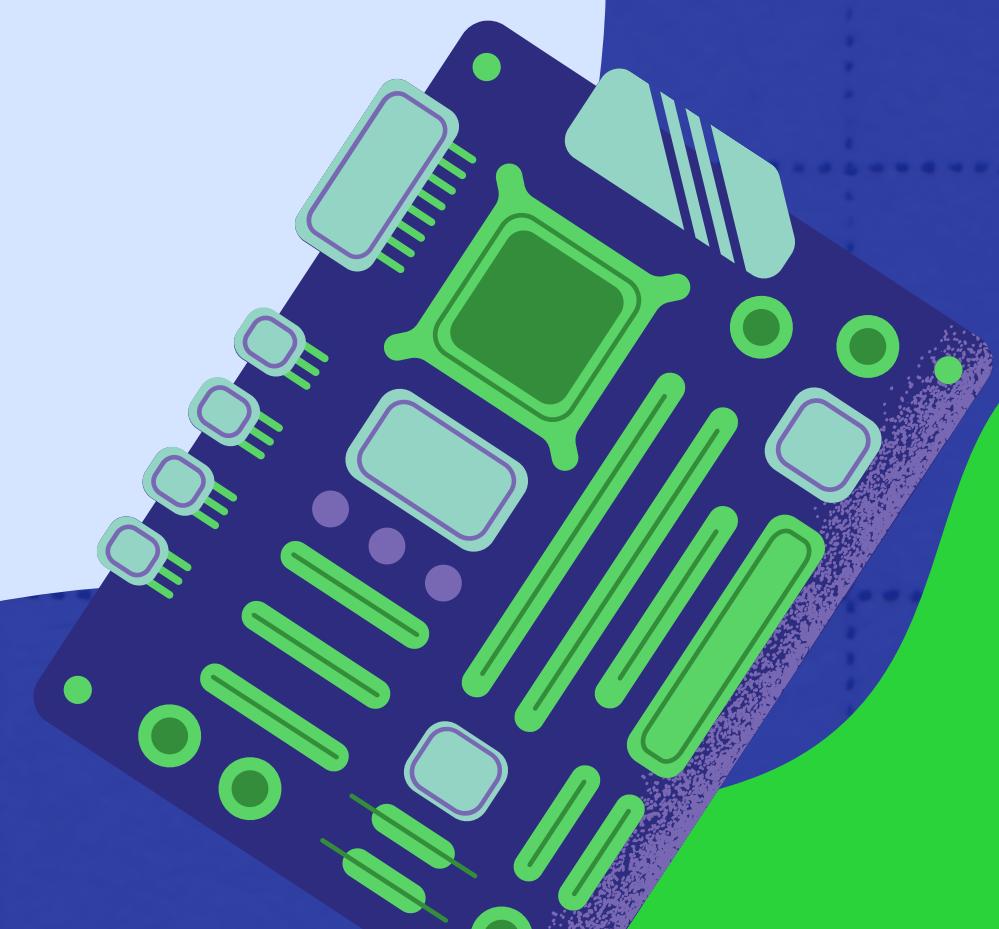
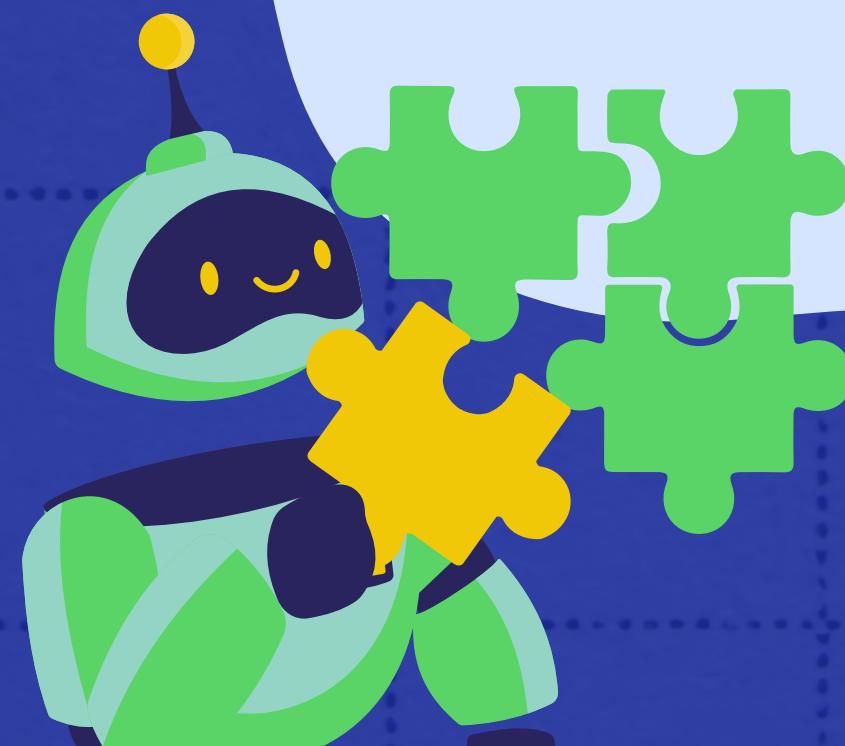
COMBU  
MOKER

# DESAFIO

1. Faça um led aumentar e decrescer sua intensidade

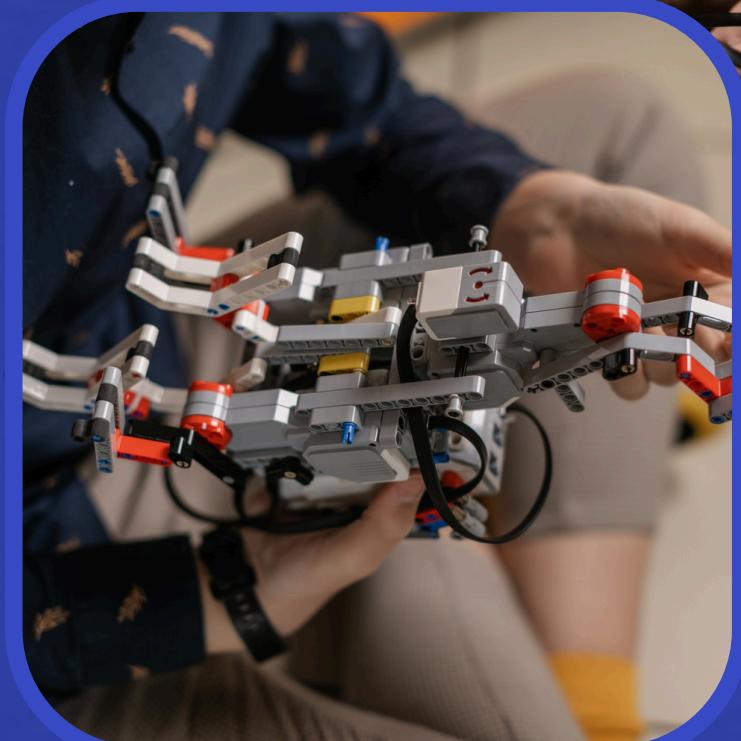


# BUZZER



COMBU  
MOKER

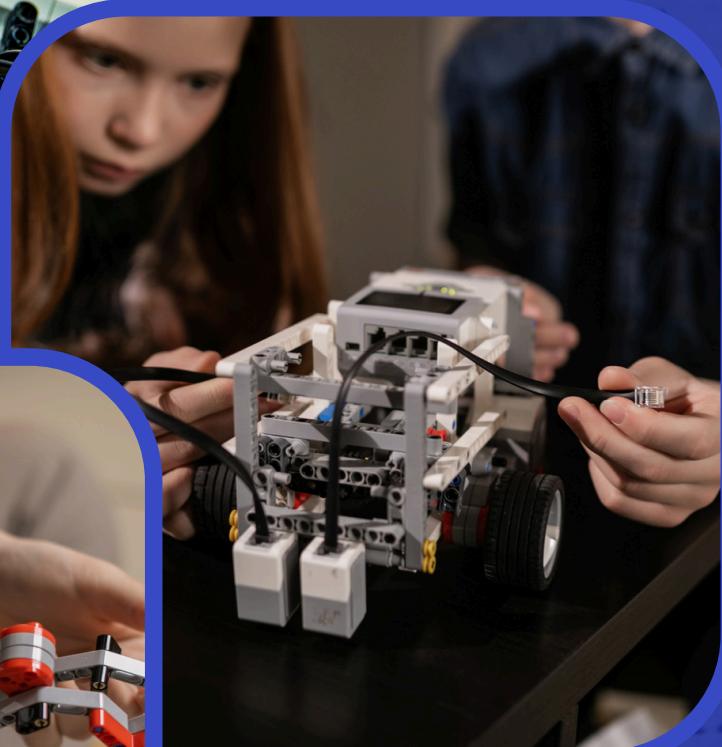
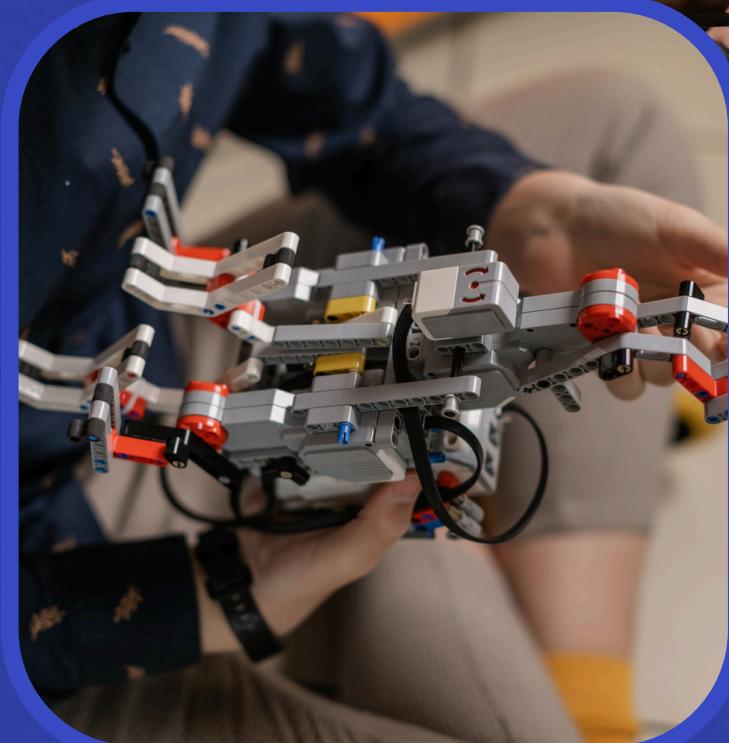
# BUZZER



## O que é?

- Buzzer é um dispositivo utilizado em eletrônica para produzir som ou emitir um sinal sonoro.
- Para usar um buzzer basta atribuí-lo à uma porta PWM (as que possuem sinal “~”), junto com um sinal GND para o aterramento.

# BUZZER

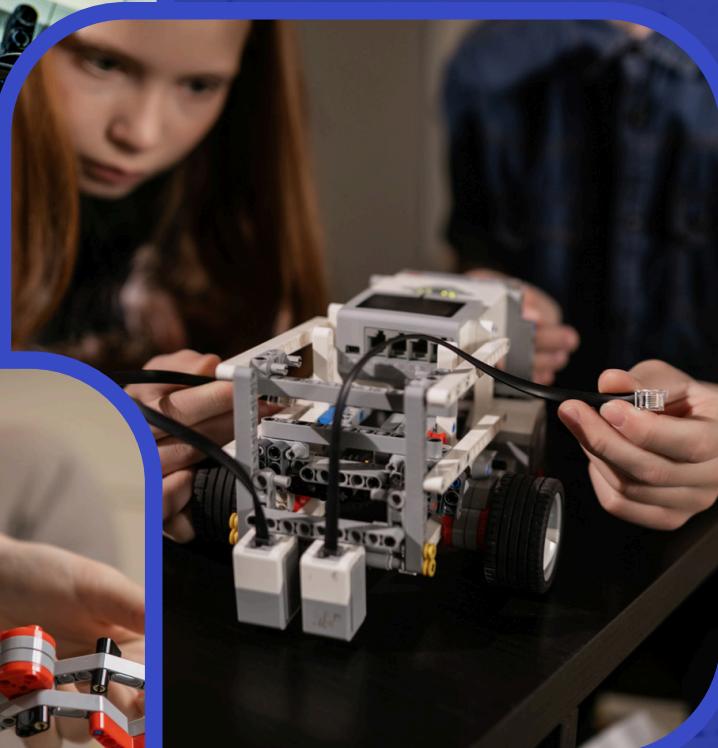


eeee

## Programação:

- Diferentemente dos LEDs, que possuem maneiras diferentes de funcionamento diferentes dependendo da porta que é colocado, os buzzers que usaremos não precisam fazer o uso da função **analogWrite()**, pois só funcionam com sinais PWM, e possui duas funções principais.

# BUZZER



eeee

## **tone():**

- É a função responsável por atribuir o pin que se contra o buzzer e a frequência do som que ele irá emitir.

## **noTone():**

Essa função recebe somente um argumento que é o pin do buzzer e é responsável por parar o som que ele está emitindo .

# EXEMPLO

```
1 tone(buzzerPin, 1000);  
2 delay(1000);
```

# EXEMPLO



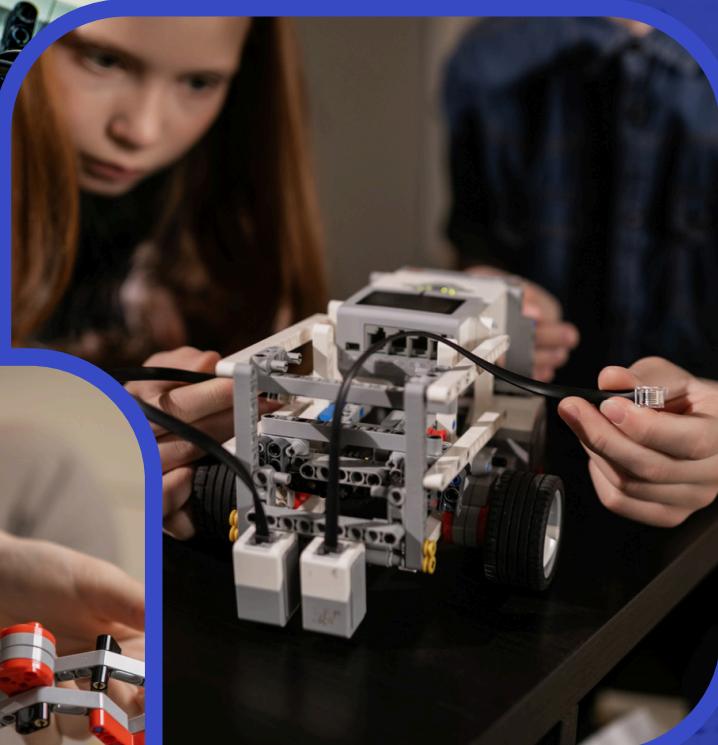
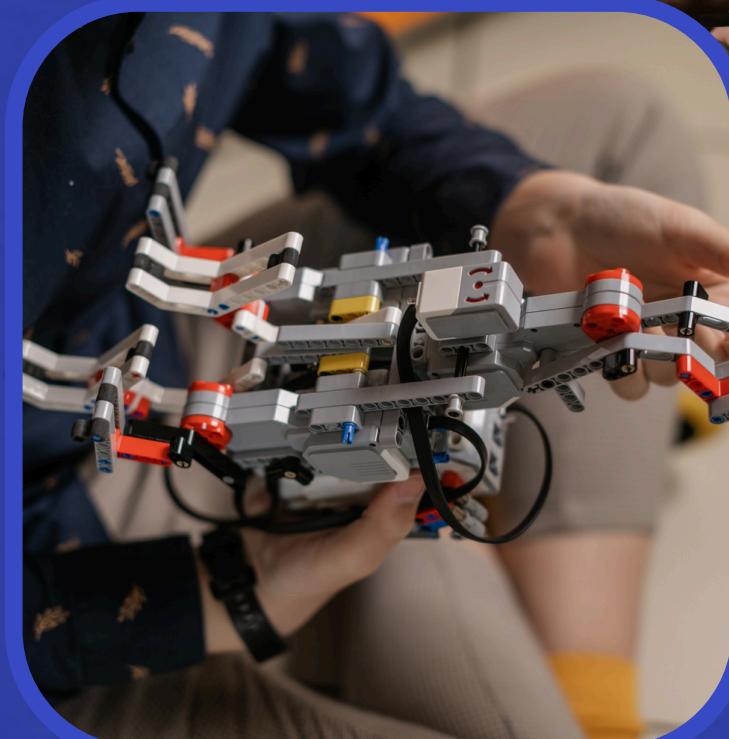
```
1 noTone(buzzerPin);  
2 delay(1000);
```



# NOTAS MUSICAIS

COMBU  
MOKER

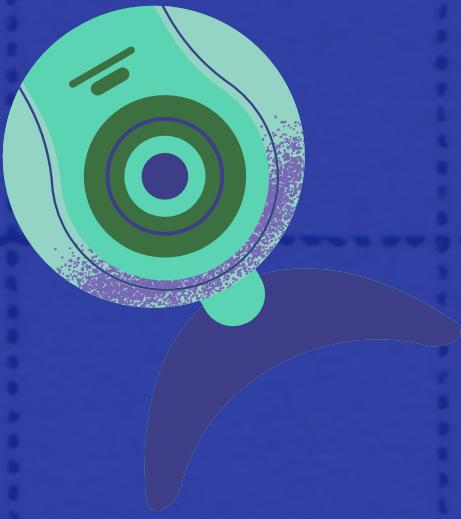
# BUZZER



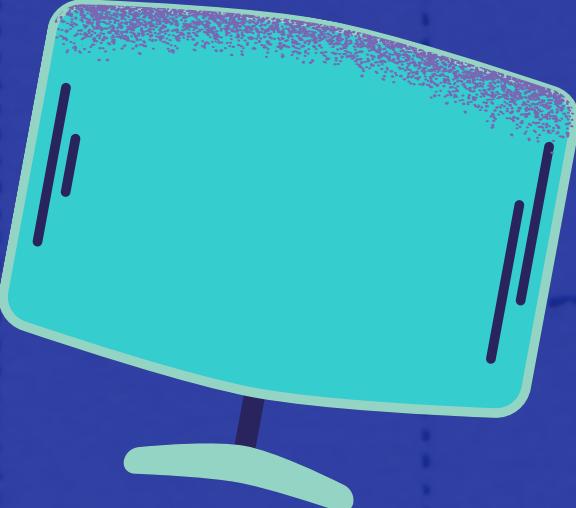
eeee

Com a entrada PWM, é possível realizar a composição de músicas pelo buzzer por meio das notas musicais

# NOTAS



eee



Frequência em Hz

Nota musical		1º Oitava	2º Oitava	3º Oitava	4º Oitava	5º Oitava	6º Oitava	7º Oitava	8º Oitava	9º Oitava	
Dó		33	66	132	264	528	1056	2112	4224	8448	16896
Dó #		34,947	69,894	139,79	279,6	559,15	1118,3	2236,6	4473,2	8946,4	17893
Ré		37,026	74,052	148,1	296,2	592,42	1184,8	2369,7	4739,3	9478,7	18957
Ré #		39,237	78,474	156,95	313,9	627,79	1255,6	2511,2	5022,3	10045	20089
Mi	20,79	41,58	83,16	166,32	332,6	665,28	1330,6	2661,1	5322,2	10644	
Fá	22,03	44,055	88,11	176,22	352,4	704,88	1409,8	2819,5	5639	11278	
Fá #	23,33	46,662	93,324	186,65	373,3	746,59	1493,2	2986,4	5972,7	11945	
Sol	24,72	49,434	98,868	197,74	395,5	790,94	1581,9	3163,8	6327,6	12655	
Sol #	26,19	52,371	104,74	209,48	419	837,94	1675,9	3351,7	6703,5	13407	
Lá	27,75	55,506	111,01	222,02	444	888,1	1776,2	3552,4	7104,8	14210	
Lá #	29,4	58,806	117,61	235,22	470,4	940,9	1881,8	3763,6	7527,2	15054	
Si	31,15	62,304	124,61	249,22	498,4	996,86	1993,7	3987,5	7974,9	15950	
Dó		33	66	132	264	528	1056	2112	4224	8448	16896





# DESAFIOS

COMBU  
MOKER

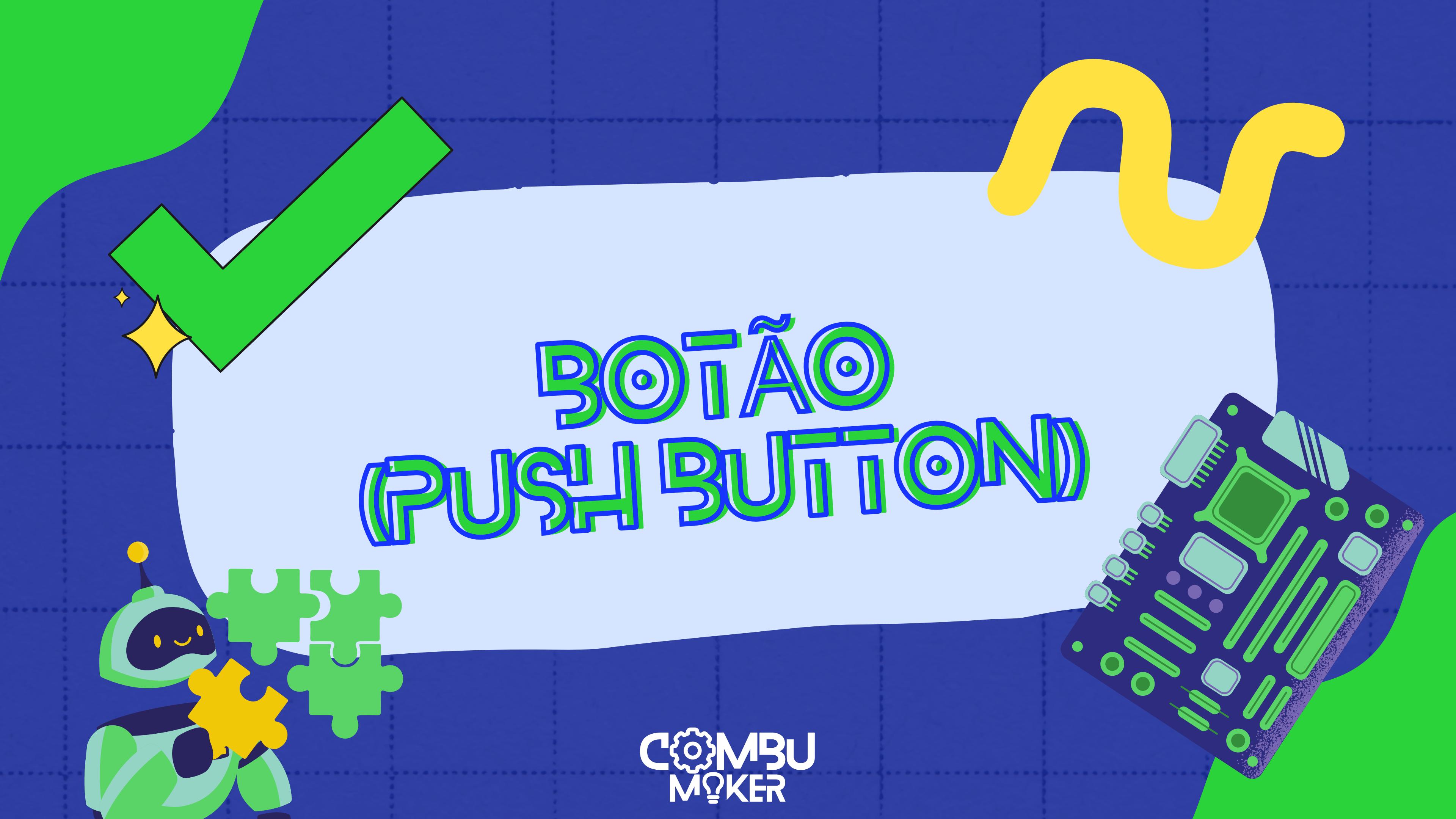
# DESAFIO

1. Toquem uma sequência de sons no buzzer;
2. Acenda um led e altere sua intensidade de acordo com as notas musicais;
3. Faça uma sirene usando o buzzer.

Confira algumas  
músicas na  
linguagem arduino:

[https://github.com/robs  
oncouto/arduino-songs](https://github.com/robsoncouto/arduino-songs)





# BOTÃO PUSH BUTTON

COMBU  
MOKER

# BOTÃO (PUSH BUTTON)



## O que é:

- É um dos componentes eletrônicos mais utilizados para prototipagem de projetos.
- Esta chave é um tipo de interruptor pulsador (conduz somente quando está pressionado).

# BOTÃO (PUSH BUTTON)



eeee

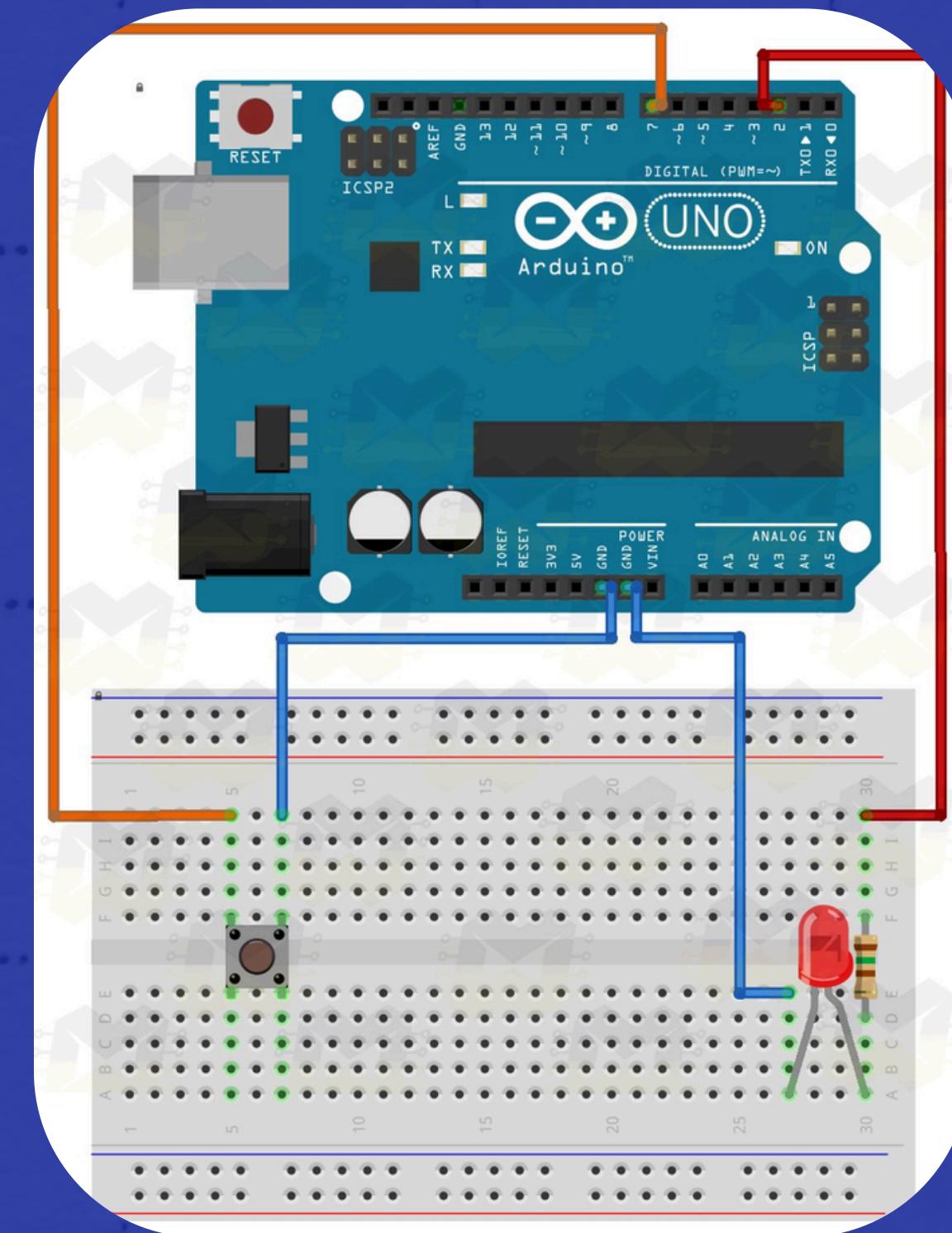


## Pull UP e Pull DOWN:

- São usados em eletrônica para garantir que um circuito funcione corretamente e evite problemas indesejados.

- Ajudam a eliminar “ruídos elétricos” que podem causar leituras erradas no sistema.

# BOTÃO PUSH BUTTON

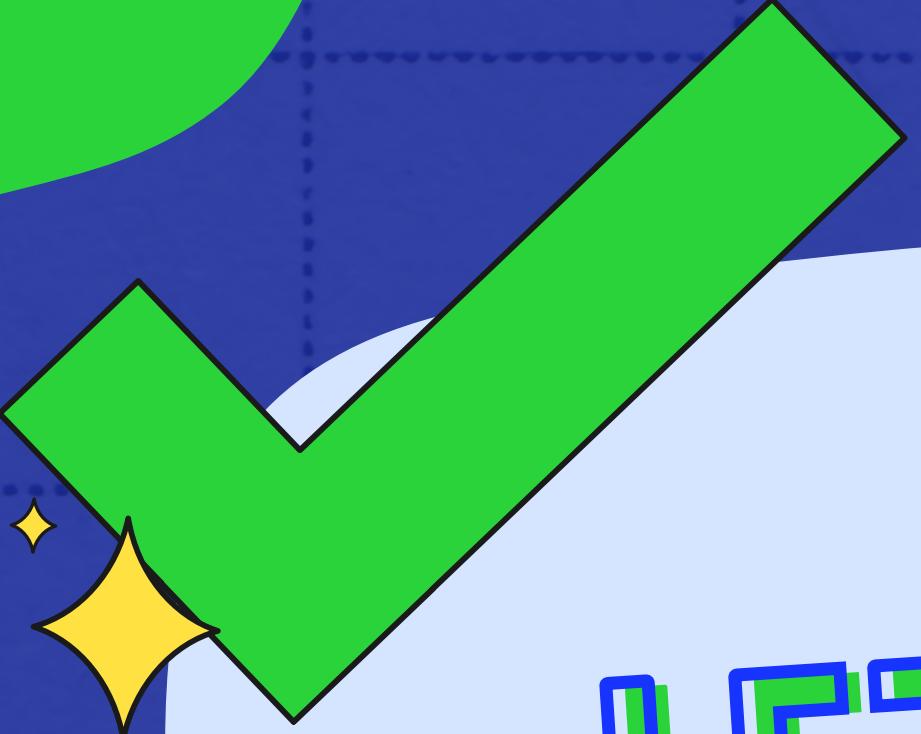


COMBU  
MOKER

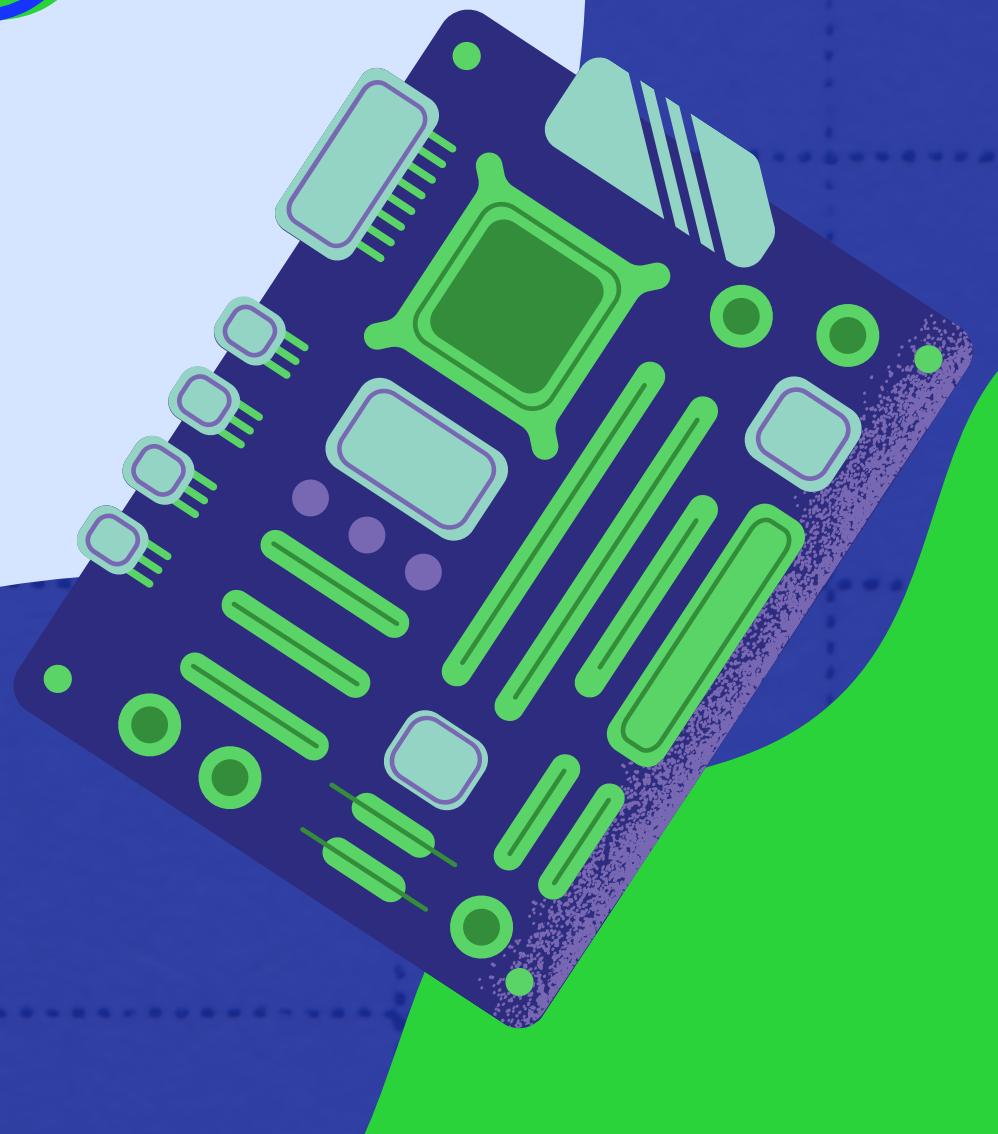
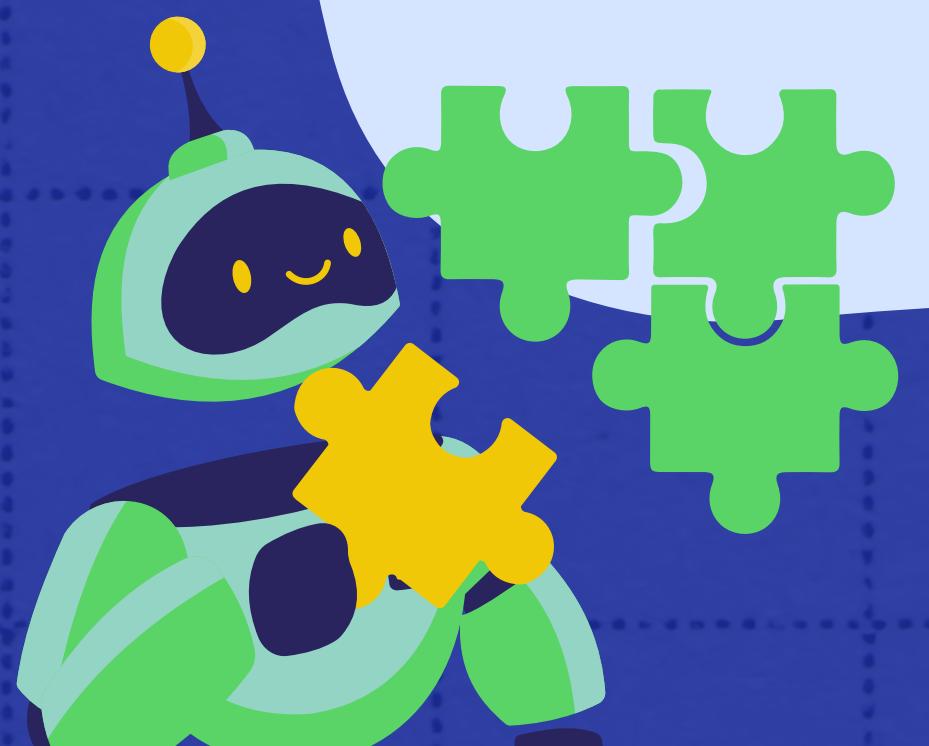
# BOTÃO (PUSH BUTTON)



```
1 void setup() {  
2     pinMode(buttonPin, INPUT_PULLUP);  
3     pinMode(ledPin, OUTPUT);  
4 }
```



**LER ESTADO DO  
BOTÃO**



**COMBU  
MOKER**

# DIGITAL READ



eeee

É necessário fazer o uso da instrução digital read para ler o estado do botão e assim programar o botão de acordo com os seus estados, HIGH ou LOW

# EXEMPLO

```
● ● ●  
1  
2 void loop() {  
3     int buttonState = digitalRead(buttonPin);  
4 }
```



# DESAFIOS

COMBU  
MOKER

# DESAFIO

1. Façam o led acender com o botão;
2. Toque um som quando apertar o botão;
3. Use 3 botões e LEDs. Para cada conjunto, toque um som diferente quando pressionar o botão.