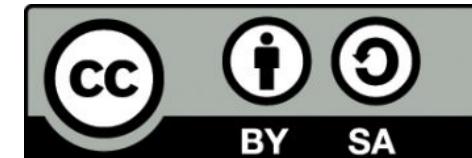


# Introdução à Plataforma Arduino

# Estrutura do Curso

- Pontos Abordados
  - Plataforma Arduino
  - Prototipagem em protoboard
  - Conceitos básicos de eletrônica e programação em C
  - Montagem de projetos

***“Introdução à Plataforma Arduino”***, de Saulo Jacques pode ser copiada, distribuída e utilizada em obras derivadas, desde que utilizada licença idêntica e atribuído créditos devidos ao autor. **Creative Commons - Atribuição-Compartilhamento 4.0 Internacional.**



# Material Disponível

## Materiais do Curso com Acesso Livre

<https://github.com/smjacques/Horta-Automatizada-ESDI>

The screenshot shows the GitHub repository page for 'smjacques / Horta-Automatizada-ESDI'. The repository has 3 commits, 1 branch, 0 releases, and 1 contributor. The latest commit was 13 minutes ago. The repository contains files for Slides, esquemas, imagens, sketches, and README.md.

This repository

smjacques / Horta-Automatizada-ESDI

3 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Commit	Time
Slides	first commit	13 minutes ago
esquemas	first commit	13 minutes ago
imagens	first commit	13 minutes ago
sketches	first commit	13 minutes ago
README.md	Update README.md	21 minutes ago

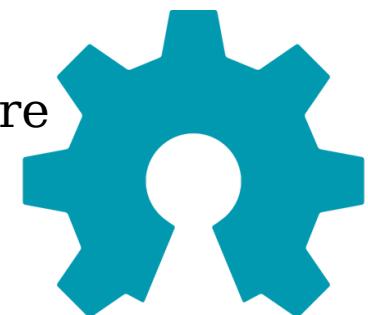
# O Projeto Arduino

**Itália (Ivrea) - 2005**

Plataforma para prototipação de circuitos eletrônicos

Ambiente de programação baseado em Wiring e C++

Hardware e ambiente de programação de código aberto e livre



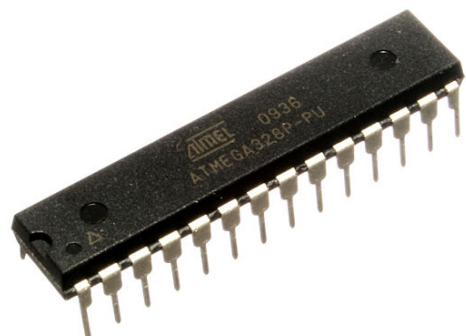
# O Projeto Arduino

Microcontrolador - Circuito integrado (CI) que incorpora várias funcionalidades

“computador de um único chip”

Utilizado com frequência em sistemas embarcados (carros, eletrodomésticos, automação residencial, etc)

Computação Física: Sistema que pode interagir com seu ambiente por meio de circuitos eletrônicos e softwares.



# A Placa Arduino

13 pinos digitais (0 V **ou** 5 V)  
(INPUT/OUTPUT configurável)

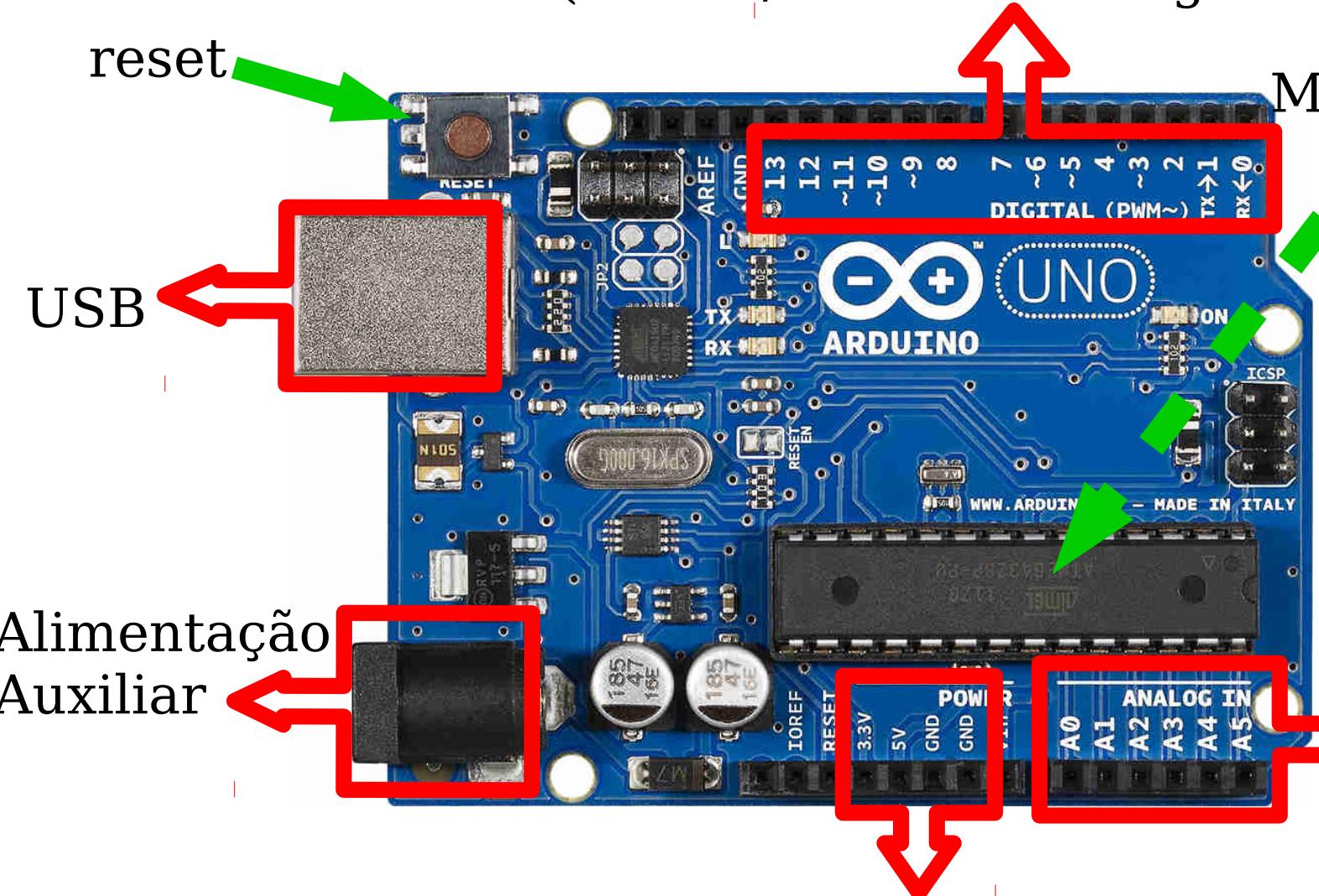
reset

USB

Alimentação  
Auxiliar

Fonte da imagem:

[http://en.wikipedia.org/wiki/Arduino#mediaviewer/File:Arduino\\_Duemilanove\\_2009b.jpg](http://en.wikipedia.org/wiki/Arduino#mediaviewer/File:Arduino_Duemilanove_2009b.jpg)

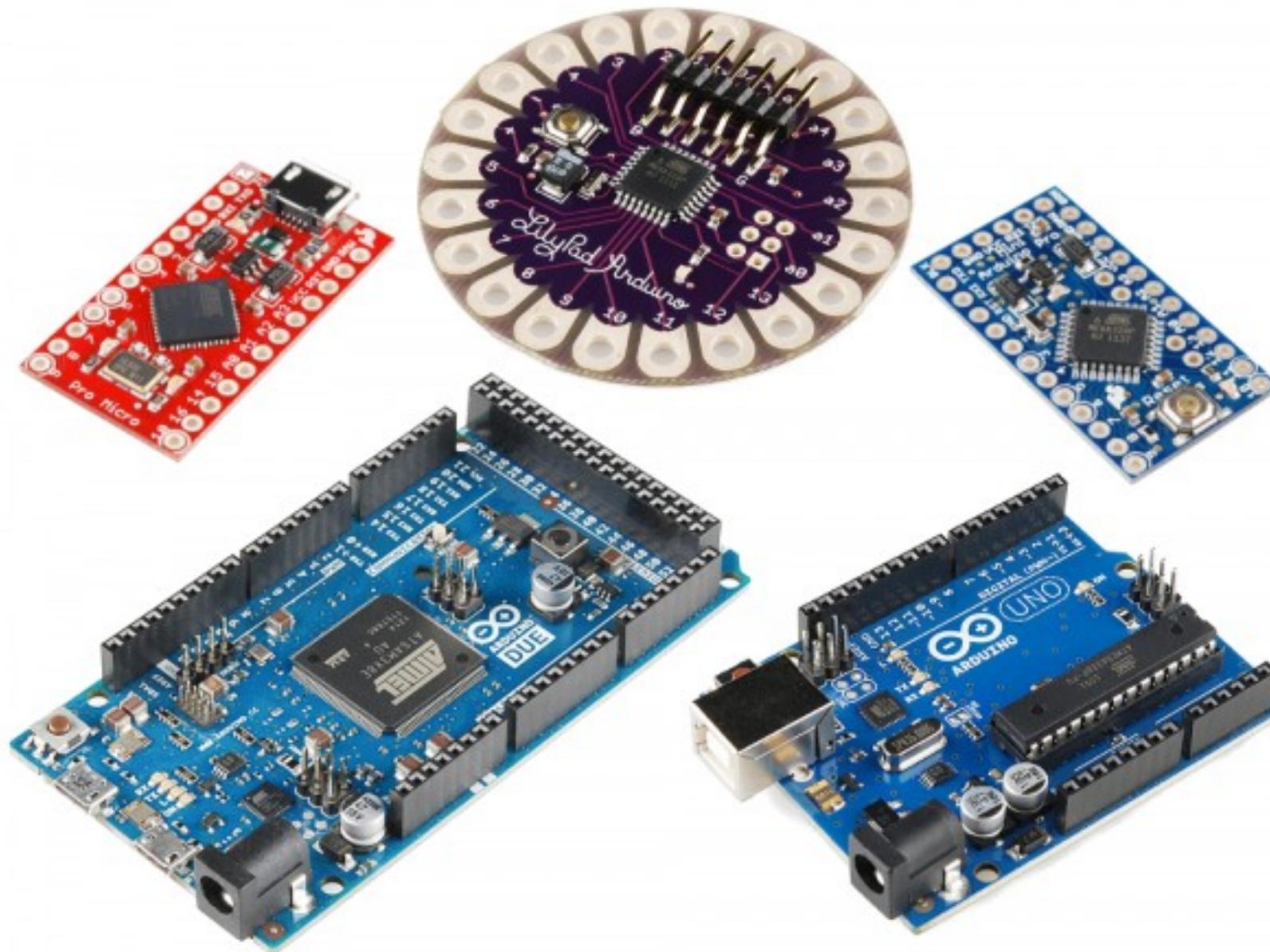


Microcontrolador  
Atmega328

- ✓ Clock: 16 MHz
- ✓ EEPROM: 1 KB (bootloader)
- ✓ SRAM: 2 KB
- ✓ Flash: 32 KB (“pendrive”)
- ✓ 5 V - 50 mA

6 pinos  
de entrada  
análogica  
(lê tensões  
entre 0 V e 5 V)

# Variantes: Uno, Mega, Mini, Micro, etc.



# Ambiente de Desenvolvimento Instalando e Usando a IDE

# Fazendo download do software (IDE)

## Download the Arduino IDE



### ARDUINO 1.8.5

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board. Refer to the [Getting Started](#) page for Installation instructions.

**Windows** Installer

**Windows** ZIP file for non admin install

**Windows app** 

**Mac OS X** 10.7 Lion or newer

**Linux** 32 bits

**Linux** 64 bits

**Linux** ARM

[Release Notes](#)

[Source Code](#)

[Checksums \(sha512\)](#)

Disponível em: <https://www.arduino.cc/en/Main/Software>

# Fazendo download do software (IDE)

## Download the Arduino IDE

- Escrever o código do programa
- Salvar o código do programa
- Compilar um programa
- Transportar o código compilado para a placa do Arduino



ARDUINO 1.8.5

The Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other

open source software. This software can be used with any Arduino Board. Refer to the [Getting Started](#) page for Installation instructions.

[Windows Installer](#)

[Windows ZIP file for non admin install](#)

[Windows app](#)

[Mac OS X 10.7 Lion or newer](#)

[Linux 32 bits](#)

[Linux 64 bits](#)

[Linux ARM](#)

[Release Notes](#)

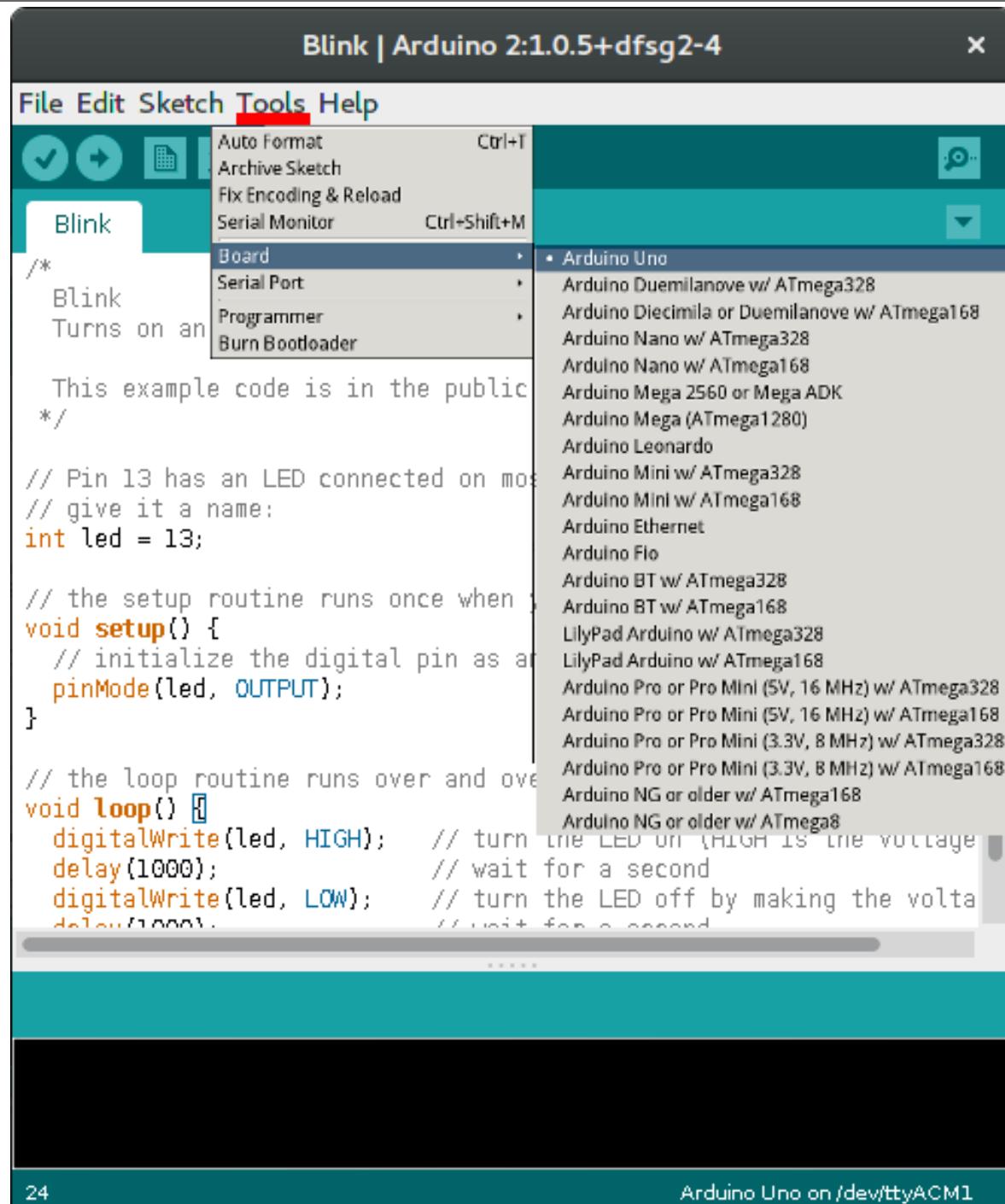
[Source Code](#)

[Checksums \(sha512\)](#)

# Selecionando Modelo da Placa

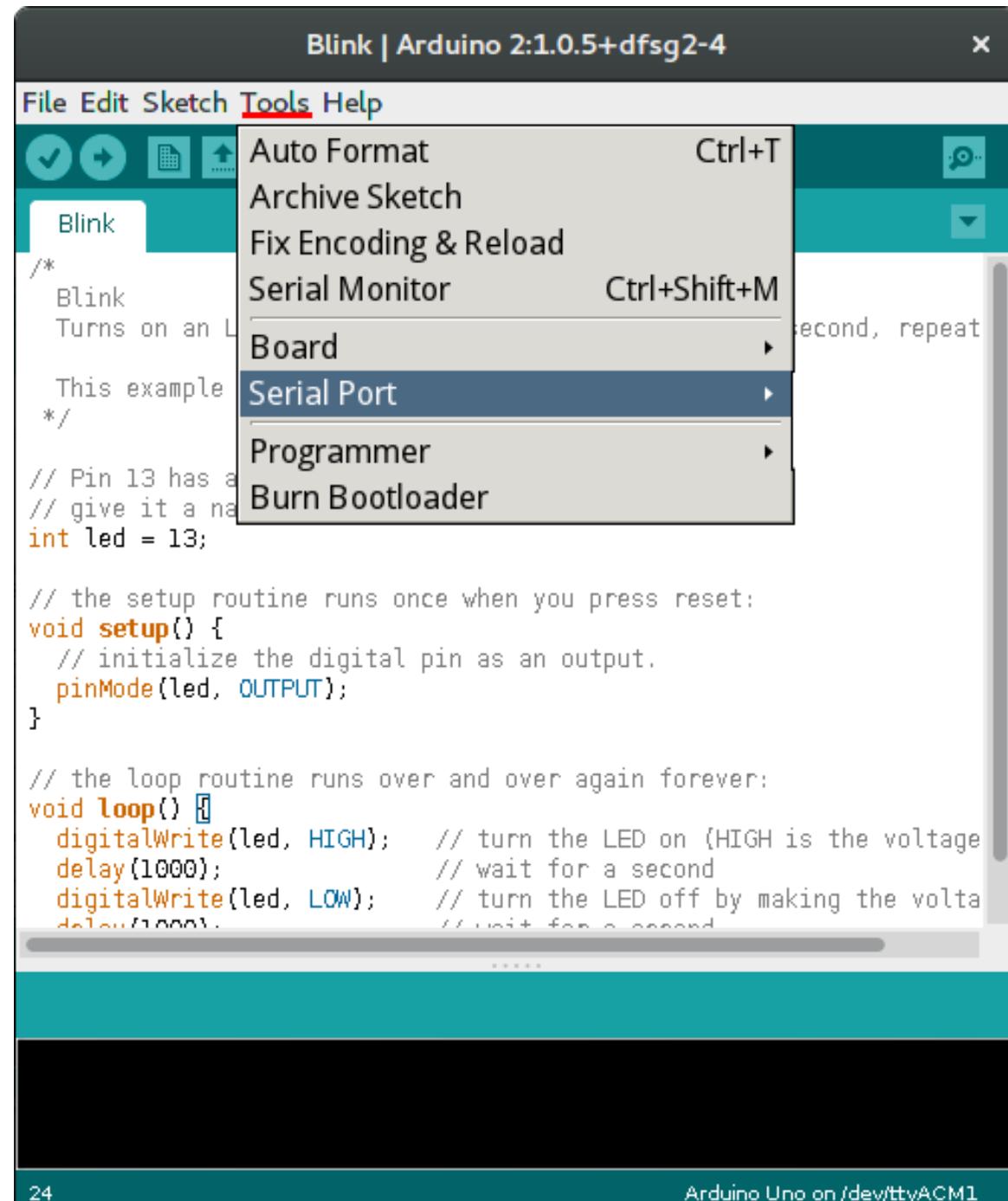
Antes de programar a placa é preciso selecionar o modelo usado usando os seguintes passos:

Tools → Board → Modelo do Arduino



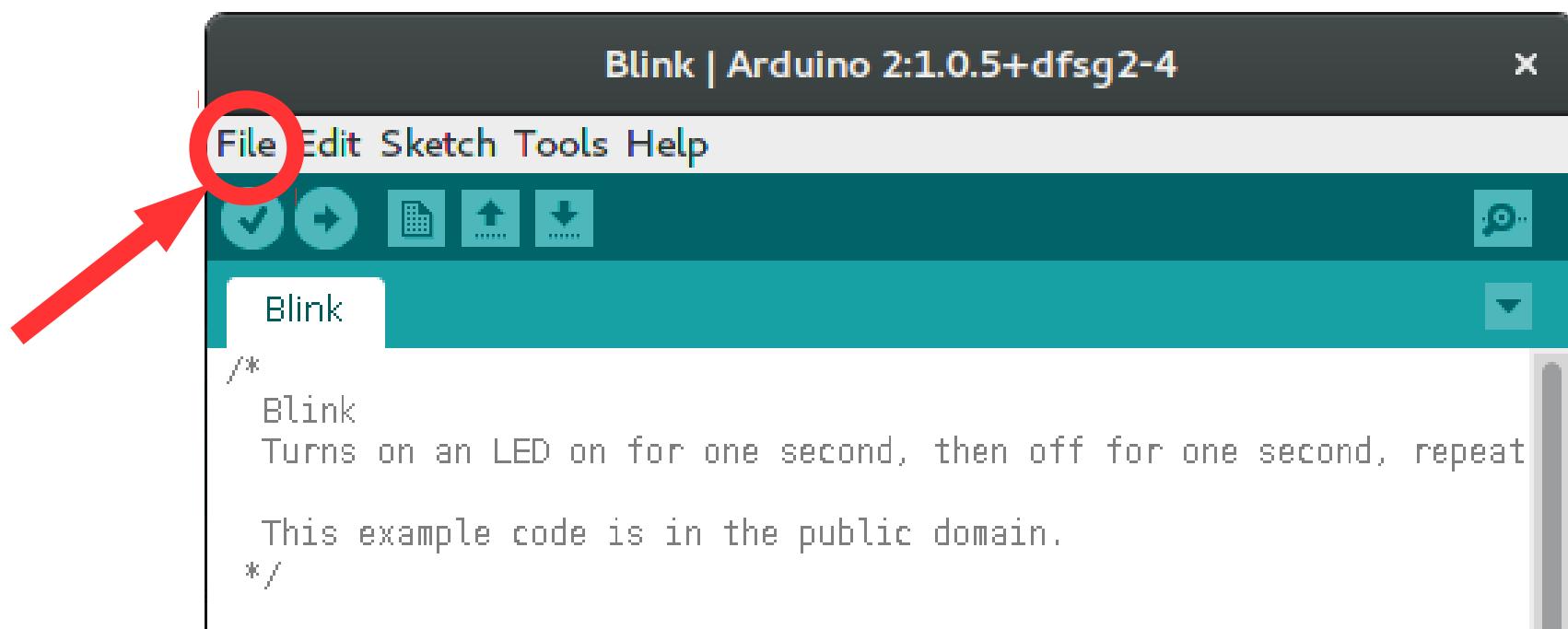
# Selecionando a Porta Serial

Nome e número da Porta Serial pode variar de acordo com o modelo de Arduino e/ou quando o cabo USB é reconectado ao computador

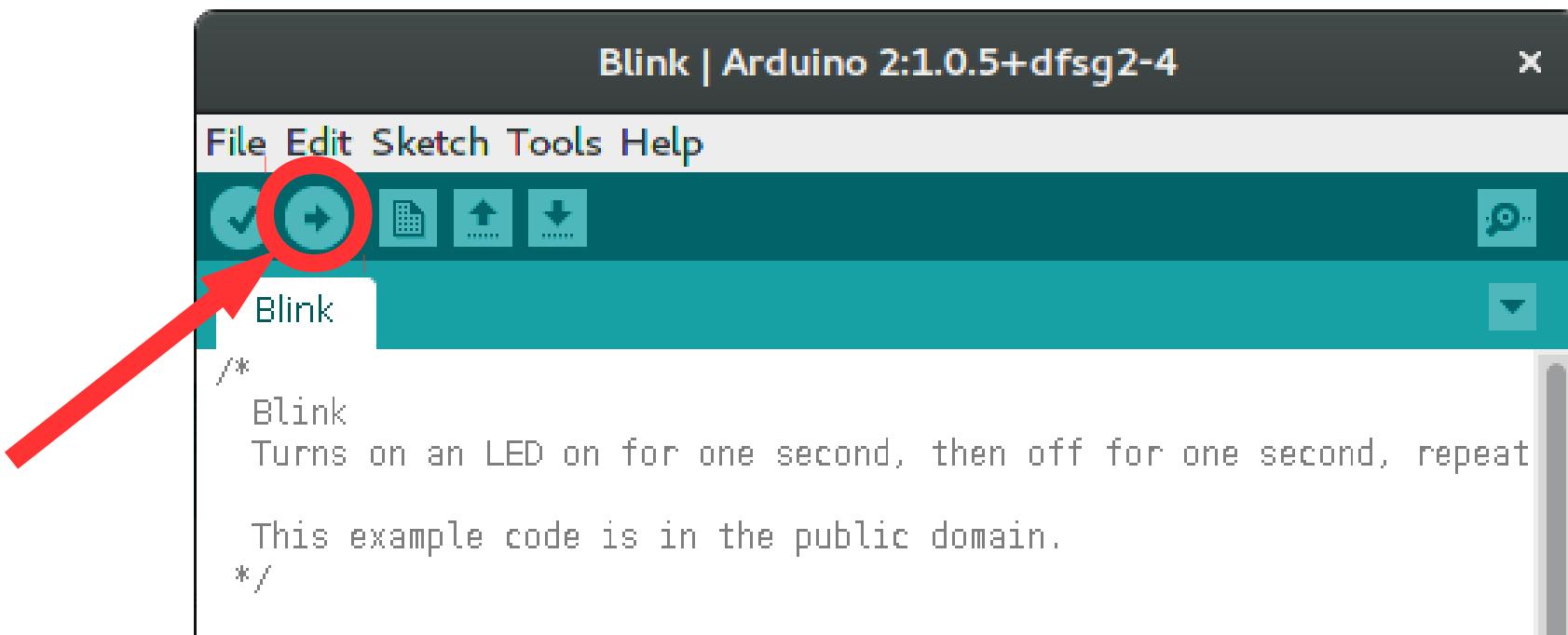


# Testando: Carregue o exemplo Blink

File → Examples → Basics → Blink



# Fazendo upload para placa



# Primeiros Passos:

## O Código

# Estrutura principal

```
void setup()
{
    // executa uma vez ao ligar a placa
}

void loop()
{
    // repete execução enquanto estiver ligada
}
```

- Em alguns SO a IDE informa Sketch Arduino vazio!
- Já está sintaticamente correto (pode compilar).
- Precisa de comandos dizendo o que fazer.

# Estrutura principal

```
void setup()
{
    // executa uma vez ao ligar a placa
}

void loop()
{
    // repete execução enquanto estiver ligada
}
```

**setup()**: onde devem ser definidas algumas configurações iniciais do programa. Executa uma única vez.

**loop()**: função principal do programa. Fica executando indefidamente.

# Variáveis e Funções

Declarando variáveis (sintaxe):

```
int      led    =    13;  
         ↓     ↓     ↓  
<tipo> <nome>;  
<tipo> <nome> = <valor atribuído a pino>;
```

# Tipos de Variáveis

**int** - Armazena números inteiros e ocupa 16 bits de memória (2 bytes). A faixa de valores é de -32.768 a 32.767.

**unsigned int** - O mesmo que *int*, porém a faixa de valores válidos é de 0 a 65.535.

**void** - Usado geralmente para informar que uma função não retorna nenhum valor. Indica tipo indefinido.

**float** - Armazena valores de ponto flutuante (com vírgula) e ocupa 32 bits (4 bytes) de memória. A faixa de valores é de -3.4028235E+38 até 3.4028235E+38

**boolean** - valor 1 (true) ou 0 (false). Ocupa um byte de memória.

**char** - Pode ser uma letra ou um número. A faixa de valores válidos é de -128 a 127. Ocupa 1 byte de memória.

**unsigned char** - O mesmo que o char, porém a faixa de valores válidos é de 0 a 255.

# Tipos de Variáveis

**long** - Armazena números de até 32 bits (4 bytes). Valores de -2.147.483.648 à 2.147.483.647.

**unsigned long** - O mesmo que o long, mas a faixa de valores é de 0 até 4.294.967.295.

**short** - Armazena número de até 16 bits (2 bytes). A faixa de valores é de -32.768 até 32.767.

**byte** - Ocupa 8 bits de memória. A faixa de valores é de 0 a 255.

**word** - O mesmo que um *unsigned int*.

# Tipos de Funções

Chamando/executando funções disponíveis (sintaxe):

pinMode(led, OUTPUT);



nome\_da\_função(arg1, arg2, ..., argN);

# Tipos de Funções

```
// Pin 13 has an LED connected on most Arduino boards.  
// give it a name:  
int led = 13;  
  
// the setup routine runs once when you press reset:  
void setup(){  
    // initialize the digital pin as an output.  
    pinMode(led, OUTPUT);  
}  
  
// the loop routine runs over and over again forever:  
void loop() {  
    digitalWrite(led, HIGH);    // turn the LED on (HIGH is the voltage level)  
    delay(1000);                // wait for a second  
    digitalWrite(led, LOW);     // turn the LED off by making the voltage LOW  
    delay(1000);                // wait for a second  
}
```

# Tipos de Funções

- Funções básicas mais comuns do Arduino:

## **Digitais**

pinMode(**num\_do\_pino**, INPUT | OUTPUT)  
digitalWrite(**num\_do\_pino**, LOW | HIGH)  
digitalRead(**num\_do\_pino**)

## **Analógicas**

analogRead(**num\_do\_pino**)  
analogWrite(**num\_do\_pino**, **valor\_PWM**)

delay(**milliseconds**)

# +info: consulte a documentação!

The screenshot shows the Arduino Reference homepage at [www.arduino.cc/en/Reference/HomePage](http://www.arduino.cc/en/Reference/HomePage). The page is organized into three main columns:

- Structure** (Green header):
  - `setup()`
  - `loop()`
- Variables** (Blue header):
  - Constants**
    - `HIGH | LOW`
    - `INPUT | OUTPUT | INPUT_PULLUP`
    - `LED_BUILTIN`
    - `true | false`
    - `integer constants`
    - `floating point constants`
  - Data Types**
    - `void`
    - `boolean`
    - `char`
    - `unsigned char`
    - `byte`
    - `int`
    - `unsigned int`
    - `word`
    - `long`
    - `unsigned long`
- Functions** (Orange header):
  - Digital I/O**
    - `pinMode()`
    - `digitalWrite()`
    - `digitalRead()`
  - Analog I/O**
    - `analogReference()`
    - `analogRead()`
    - `analogWrite() - PWM`
  - Due only**
    - `analogReadResolution()`
    - `analogWriteResolution()`
  - Advanced I/O**
    - `tone()`
    - `noTone()`
    - `shiftOut()`
    - `shiftIn()`

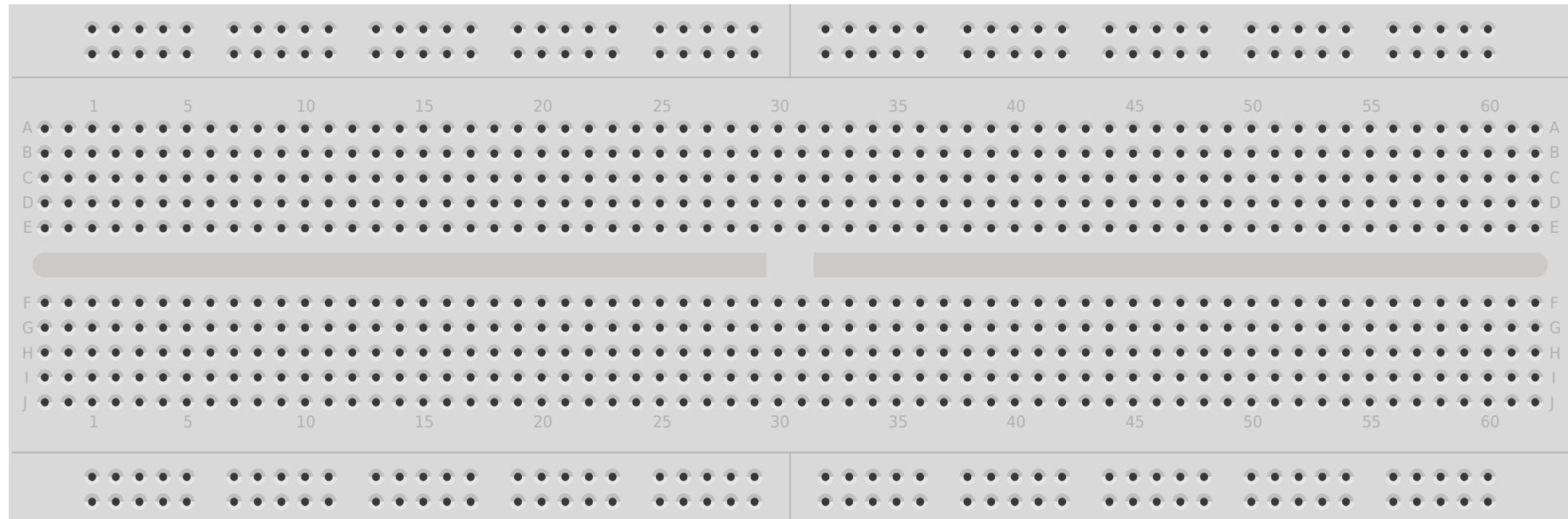
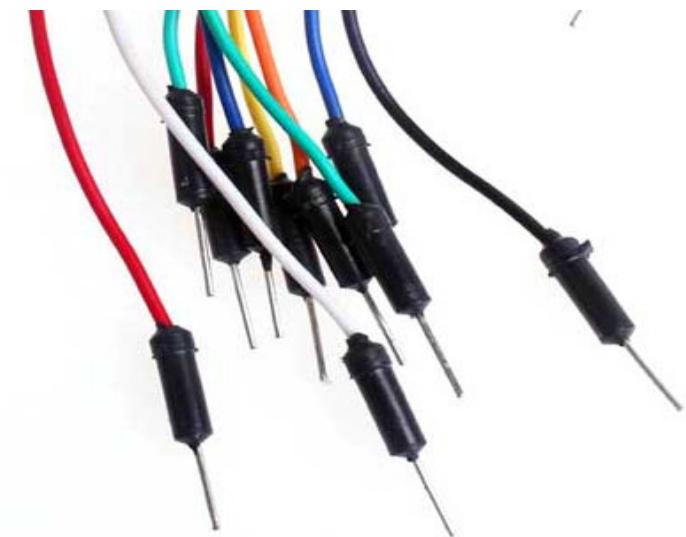
# Primeiros Passos:

## Prototipagem

# Prototipagem: protoboard

- Para que serve?
  - Permite conectar componentes eletrônicos sem soldá-los!
  - Para isso são utilizados fios jumper (imagem ao lado);

Fonte da imagem: <http://www.digibay.in/>

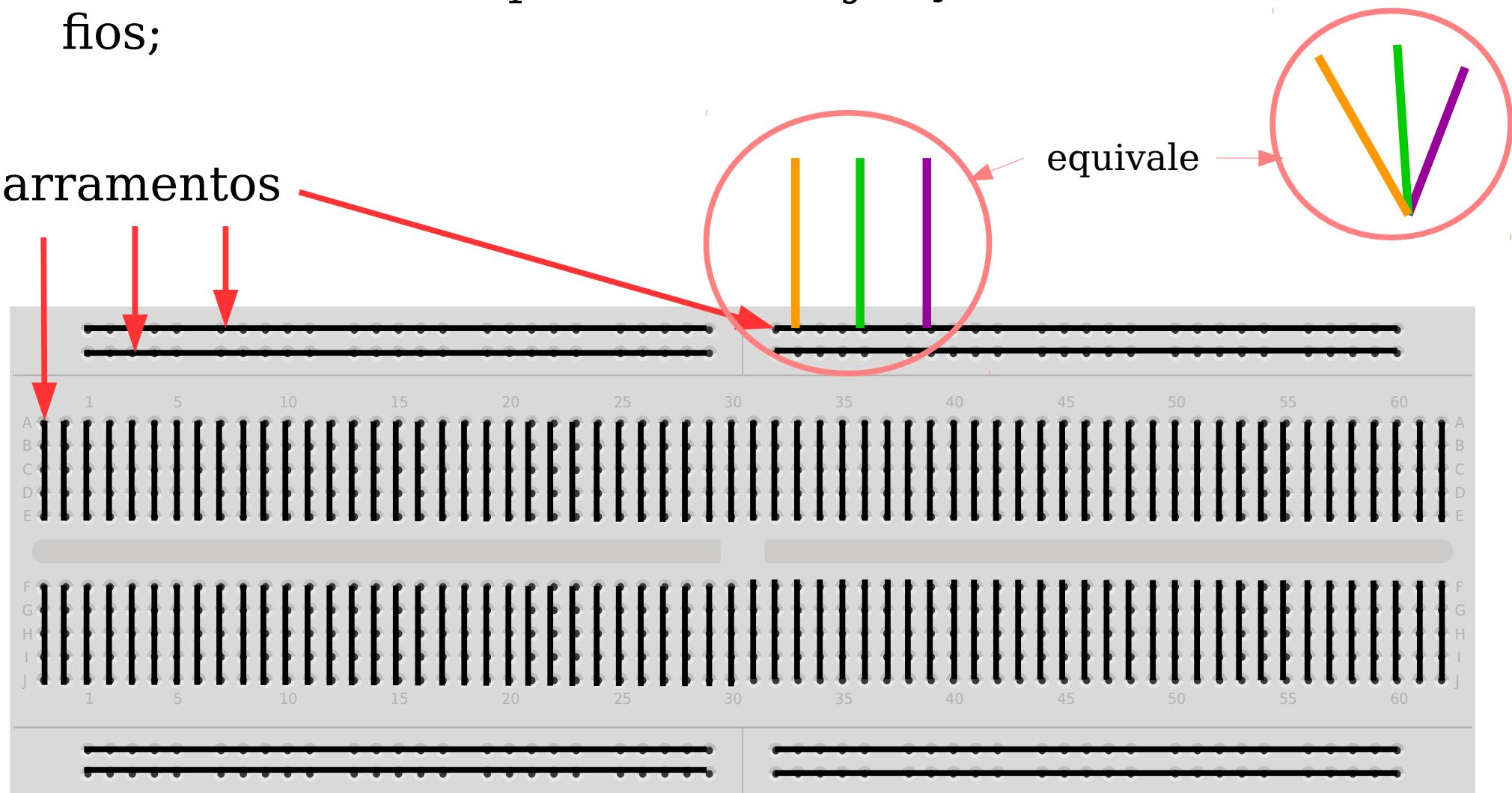


Fonte da imagem:  
<http://fritzing.org/home/>

# Protoboard: Como funciona?

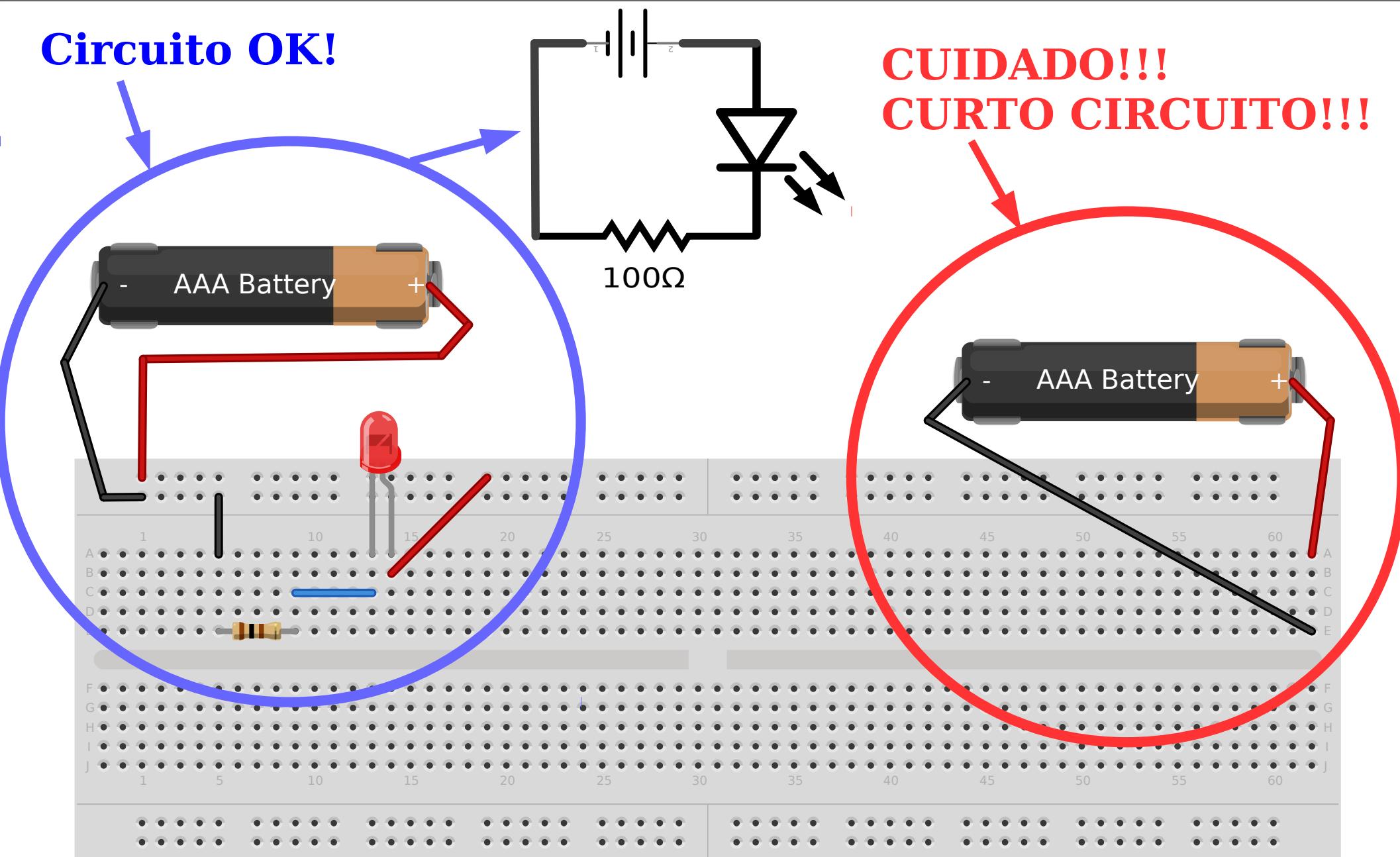
- Consiste num conjunto de barramentos isolados entre si;
- Um barramento equivale à uma junção de dois ou mais fios;

barramentos



# Protoboard: Exemplos

**Circuito OK!**





Enfim!  
Prática

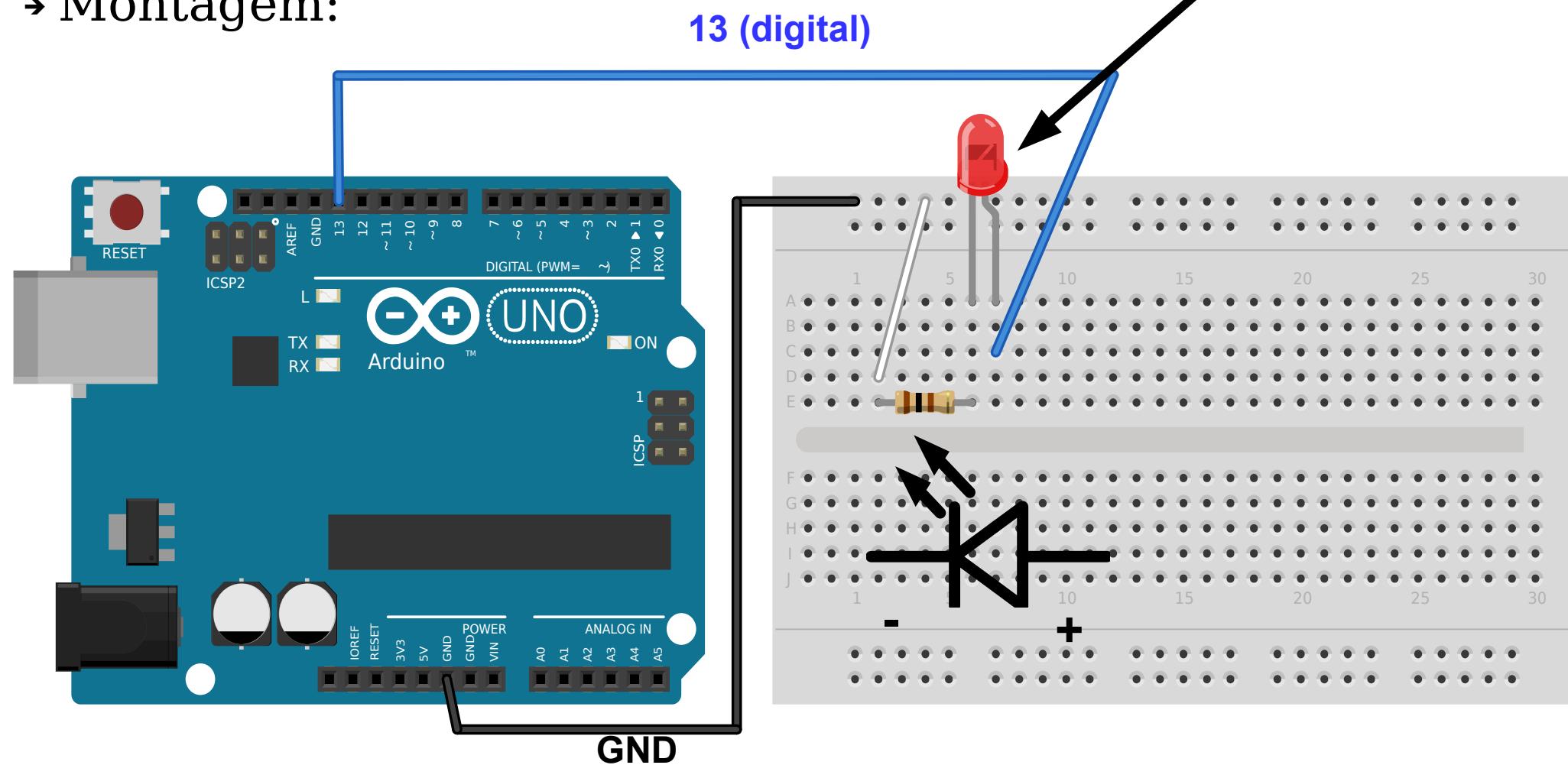
# Projeto 0: Pisca LED externo

## → Materiais:

- ✓ 1 LED
- ✓ 1 resistor de 100 ohms

**Componente polarizado:**  
perna mais longa deve ser ligada no 5 V (positivo)

## → Montagem:



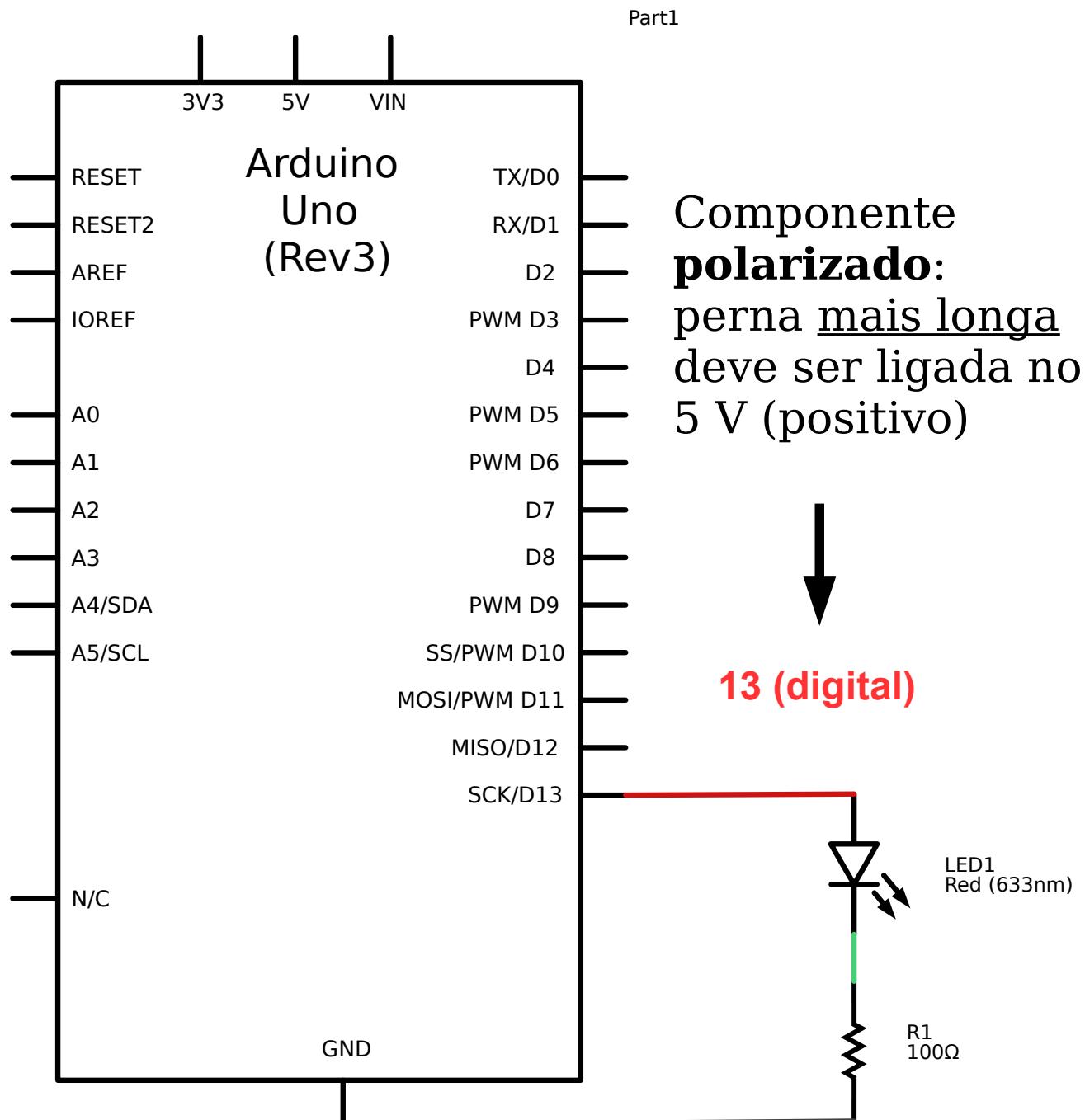
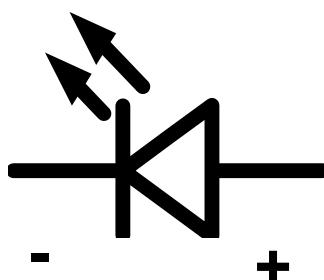
# Projeto 0: Pisca LED externo

## → Materiais:

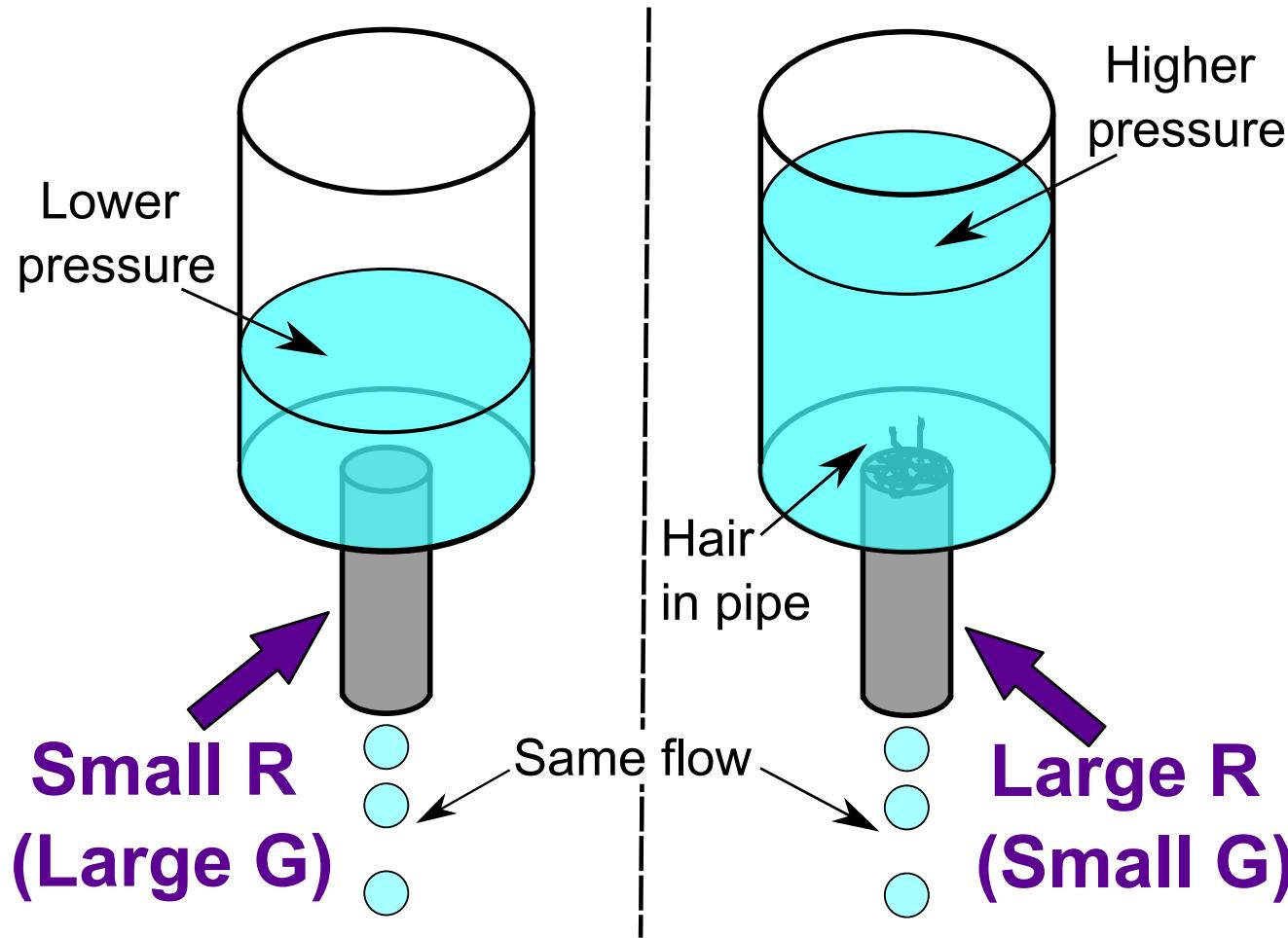
- ✓ 1 LED
- ✓ 1 resistor de 100 ohms
- ✓ fios jumper

## → Montagem:

Light Emitter Diode



# Analogia elétrico-hidráulica



- A pressão é análoga à força eletromotriz
- O escoamento é análogo à corrente elétrica
- A obstrução do cano é análoga à resistência elétrica

Fonte da imagem:

<https://commons.wikimedia.org/wiki/File:ResistanceHydraulicAnalogy.svg>

# Circuitos elétricos

## Lei de Ohm ( $\Omega$ )

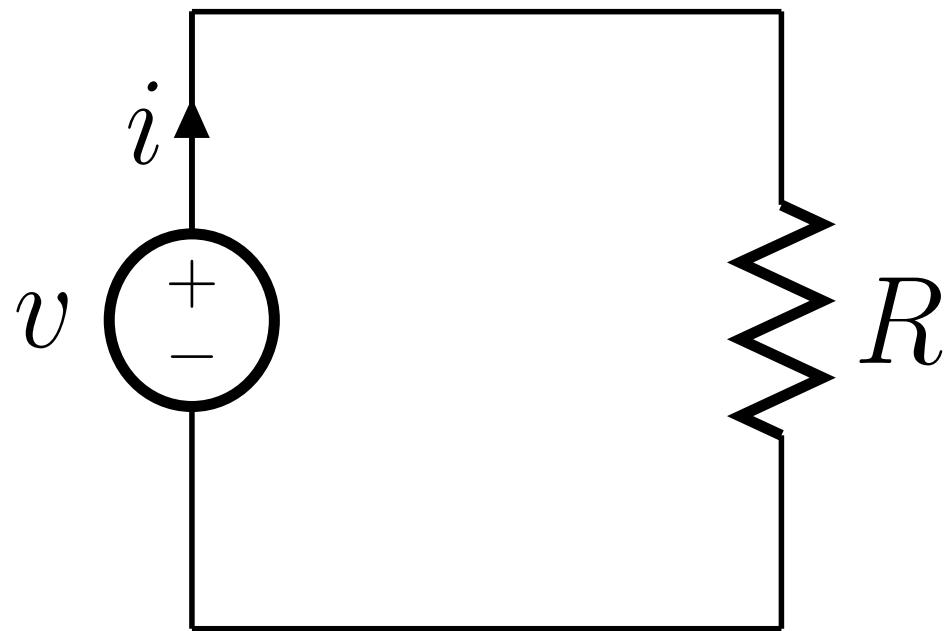
Como medimos a resistência oferecer à passagem da corrente elétrica.

$$R = \frac{V}{I}$$

Resistores causam uma queda de tensão na região do circuito em questão, mas nunca uma queda de corrente.

A corrente elétrica que entra em um terminal do resistor é a mesma corrente que sai pelo outro terminal, porém existe uma queda de tensão.

# Circuitos elétricos



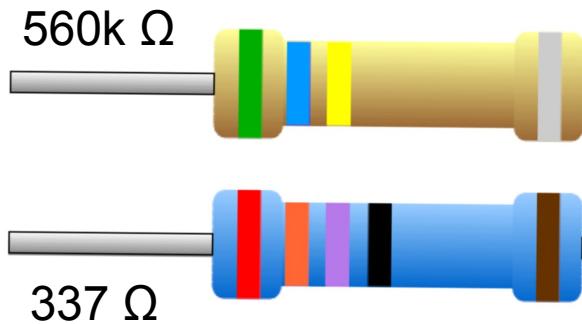
$i$ : Corrente elétrica

$V$ : Tensão elétrica

$R$ : Resistência elétrica

# Circuitos elétricos

## Código de cores dos resistores



Cor	Dígito	Multiplicador	Tolerância
Prata	-	$\times 0,01$	$\pm 10\%$
Dourado	-	$\times 0,1$	$\pm 5\%$
Preto	0	$\times 1$	-
Marrom	1	$\times 10$	$\pm 1\%$
Vermelho	2	$\times 100$	$\pm 2\%$
Laranja	3	$\times 1K$	-
Amarelo	4	$\times 10K$	-
Verde	5	$\times 100K$	$\pm 0,5\%$
Azul	6	$\times 1M$	$\pm 0,25\%$
Violeta	7	$\times 10M$	$\pm 0,1\%$
Cinza	8	-	$\pm 0,05\%$
Branco	9	-	-

Fonte da imagem:

[https://pt.wikipedia.org/wiki/Resistor#/media/File:Tabela\\_de\\_cores\\_de\\_um\\_resistor.jpg](https://pt.wikipedia.org/wiki/Resistor#/media/File:Tabela_de_cores_de_um_resistor.jpg)

# Projeto 0: Pisca LED externo

→ Código:

```
#define PINO_LED    13      // pino digital  
  
int pausa = 1000;           // millisegundos  
  
// executada uma vez ao ligar  
void setup()  
{  
    pinMode(PINO_LED, OUTPUT);  
}  
  
// fica executando para sempre  
void loop()  
{  
    digitalWrite(PINO_LED, HIGH);  
    delay(pausa);  
    digitalWrite(PINO_LED, LOW);  
    delay(pausa);  
}
```

comentários  
importantes

altere o intervalo de  
pausa em um único  
lugar!

serão substituídos  
por “13” (sem aspas)

Vantagens de  
constantes:

- (sintaxe) Definindo constantes:

```
#define <nome> valor
```

- Não ocupam memória;
- Facilita manutenção;

# Referência Arduino: analogRead()

**analogRead:** Lê o valor do pino analógico especificado. A placa Arduino contém um conversor analógico-digital; com isso, ela pode mapear tensões elétricas de entrada de entre 0 e 5 volts para valores inteiros entre 0 e 1023.

## Sintaxe:

`analogRead(pino)`

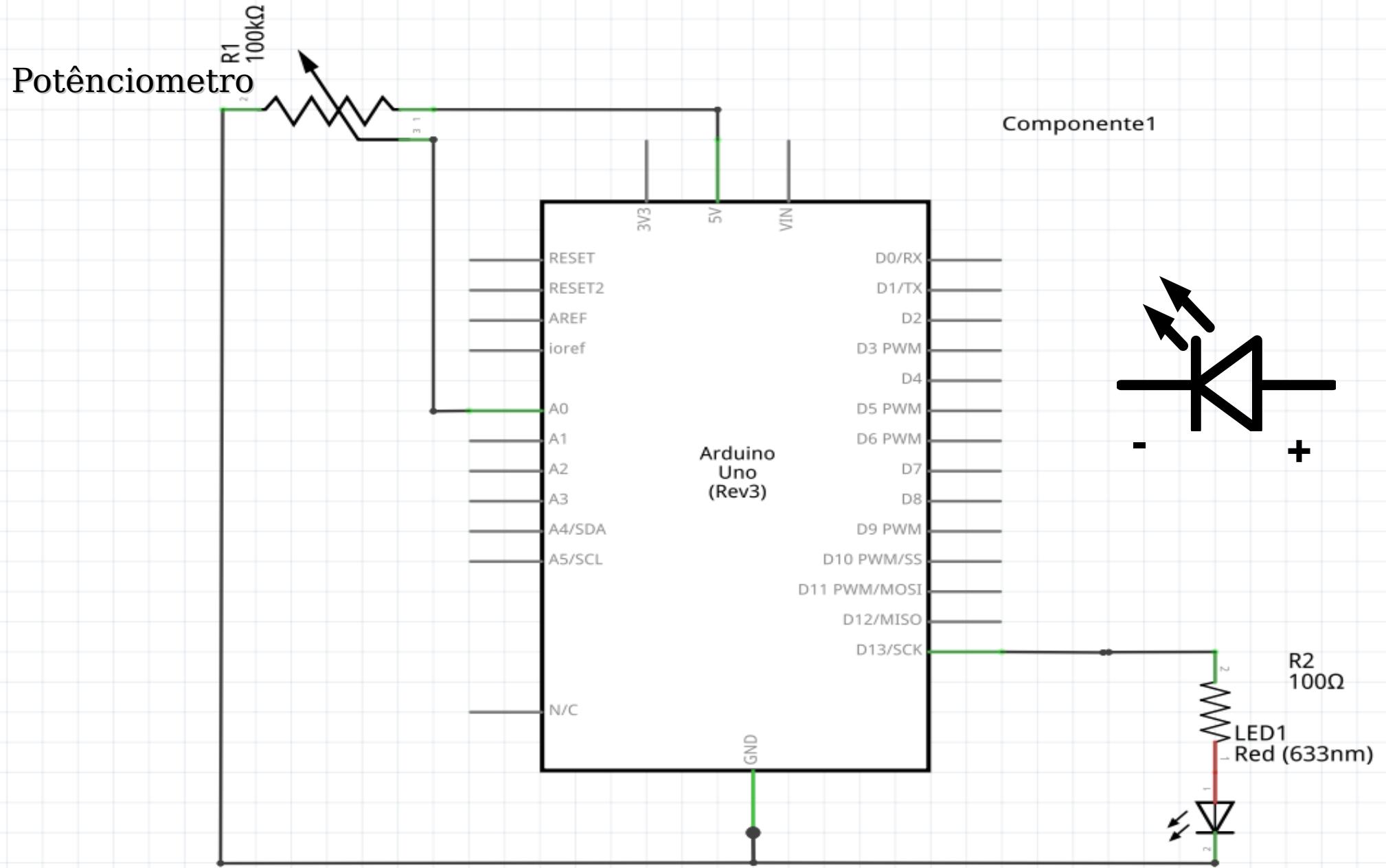
## Parâmetros:

- `pino`: o número do pino.

## Retorno:

- `int (0 a 1023)`

# Projeto 1: Controle Pisca LED

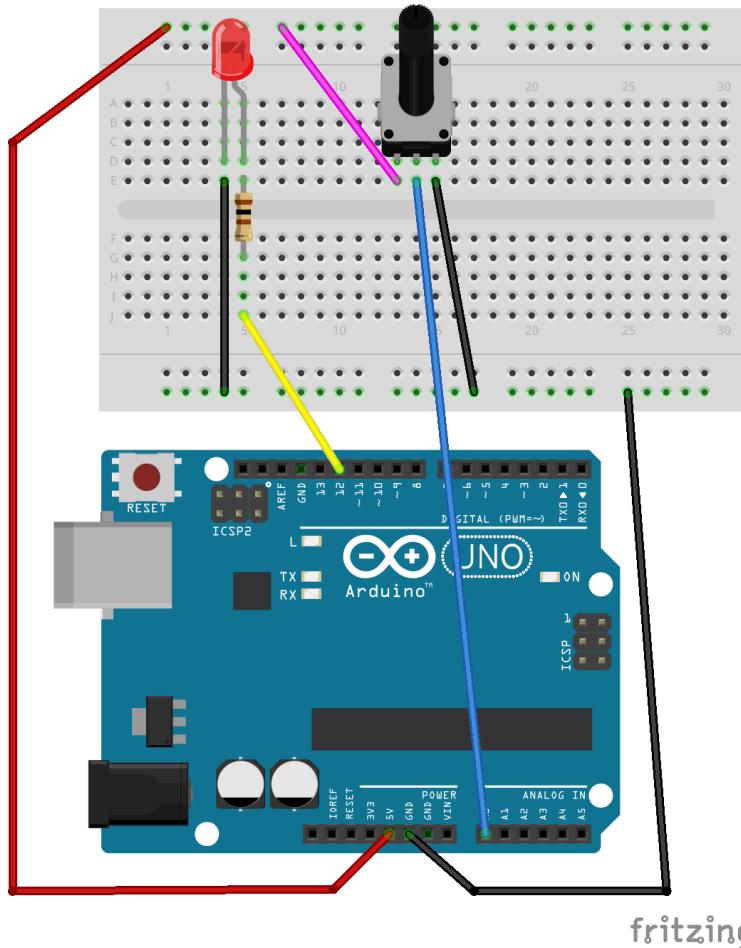


# Projeto 1: Controle Pisca LED

## → Materiais:

- ✓ 1 Potenciômetro (resistor variável)

## → Montagem:



## → Código:

```
#define PINO_LED 12 // pino digital
#define PINO_POT 0 // pino analogico

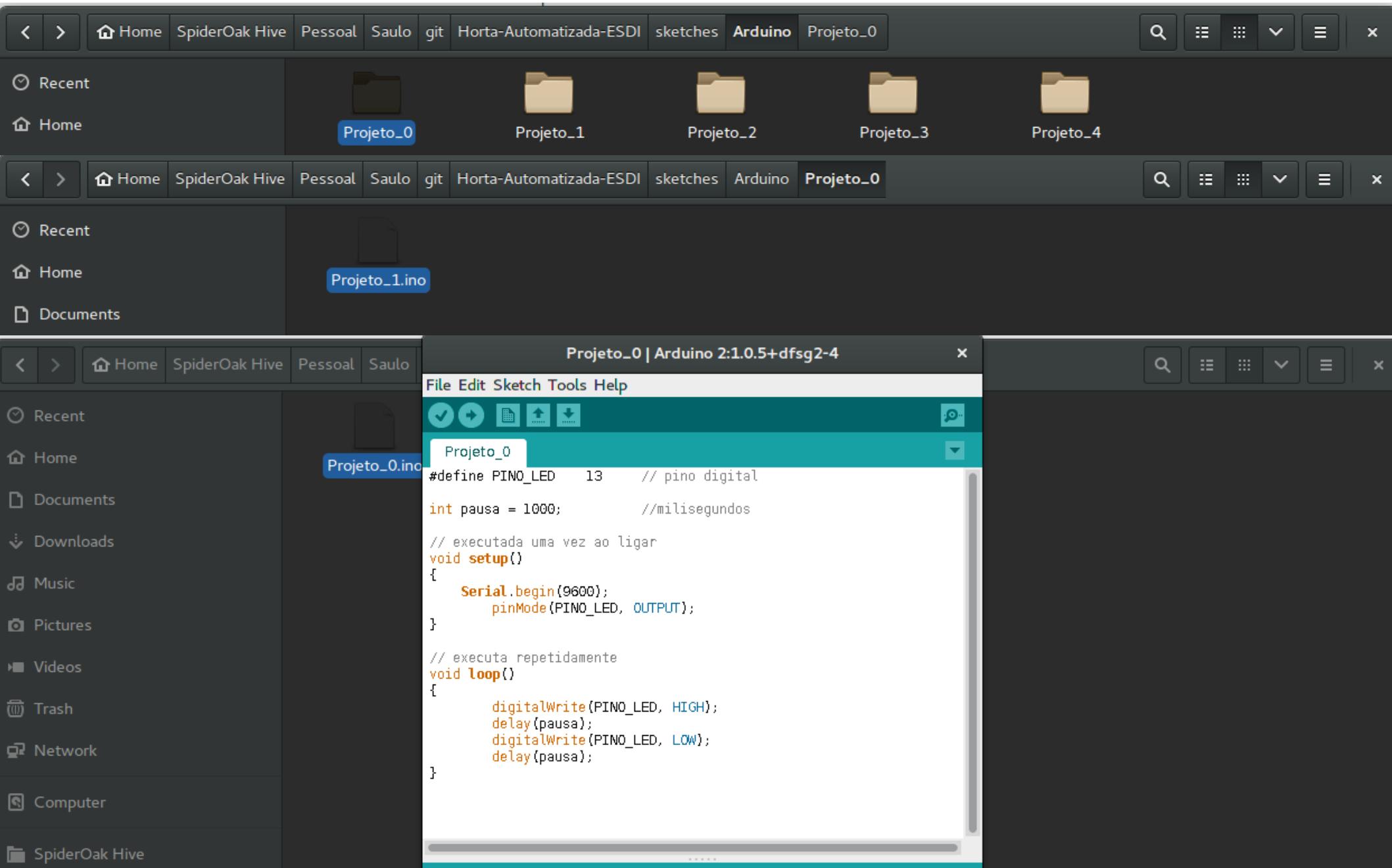
int valor_pot;

void setup()
{
    // prepara uma comunicação serial
    Serial.begin(9600);
    pinMode(PINO_LED, OUTPUT);
}

void loop()
{
    // retorna um valor entre 0 e 1023
    valor_pot = analogRead(PINO_POT);
    // manda p/ USB (ver com Monitor Serial)
    Serial.println(valor_pot);

    digitalWrite(PINO_LED, HIGH);
    delay(valor_pot);
    digitalWrite(PINO_LED, LOW);
    delay(valor_pot);
}
```

# Abrindo o arquivo do projeto 2



# LDR: Resistência Dependente de Luz

- Sua resistência varia conforme a intensidade da luz.
- A medida que a Intensidade Aumenta, a resistência Diminui.

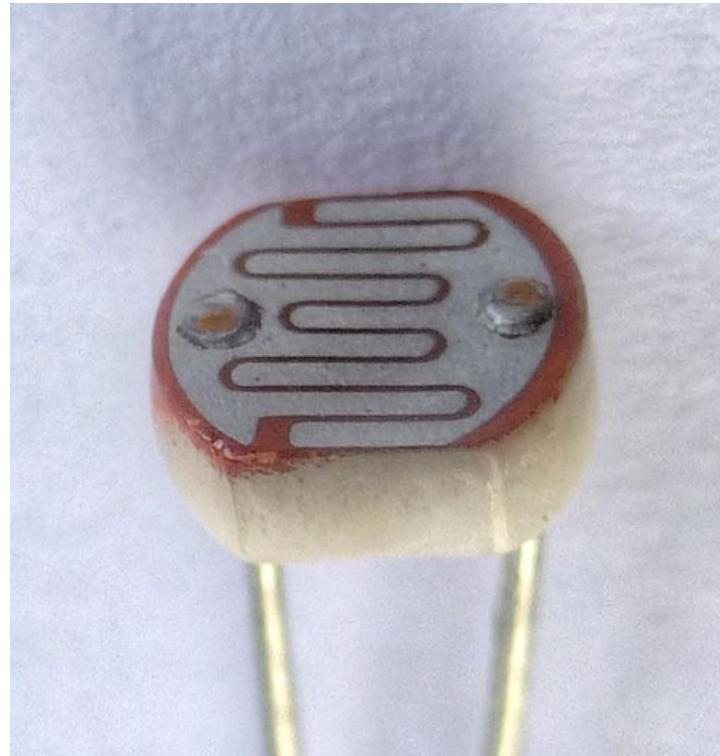
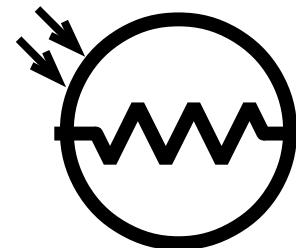


Foto de © Nevit Dilmen [CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>) or GFDL (<http://www.gnu.org/copyleft/fdl.html>)], via Wikimedia Commons  
Retirado de [https://commons.wikimedia.org/wiki/File%3ALDR\\_1480405\\_6\\_7\\_HDR\\_Enhancer\\_1.jpg](https://commons.wikimedia.org/wiki/File%3ALDR_1480405_6_7_HDR_Enhancer_1.jpg)

# Referência Arduino: Serial.begin() e Serial.print()

**Serial.begin:** configura a taxa de troca de dados em bits por segundo entre o computador e o Arduino (transmissão serial).

**Serial.print:** imprime dados na porta serial em um formato de texto que pode ser lido por humanos.

## Sintaxe:

Serial.begin(velocidade)

## Parâmetros:

- Velocidade: 9600 bit/segundo

## Retorno:

- nada

## Sintaxe:

Serial.print(valor)

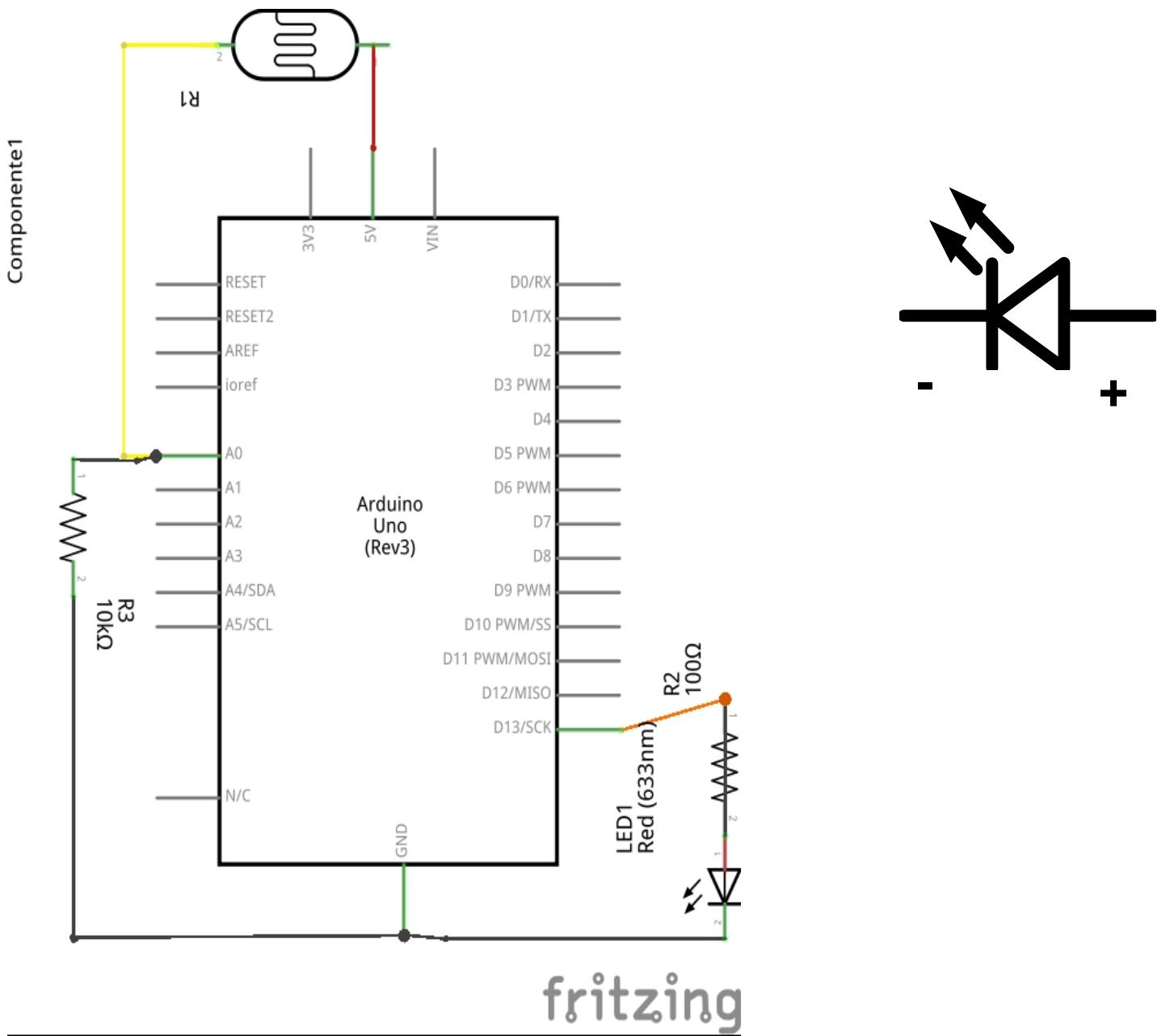
## Parâmetros:

- Valor: qualquer número, texto ou variável.

## Retorno:

- size\_t (long)

# Projeto 2: Controla LED com LDR

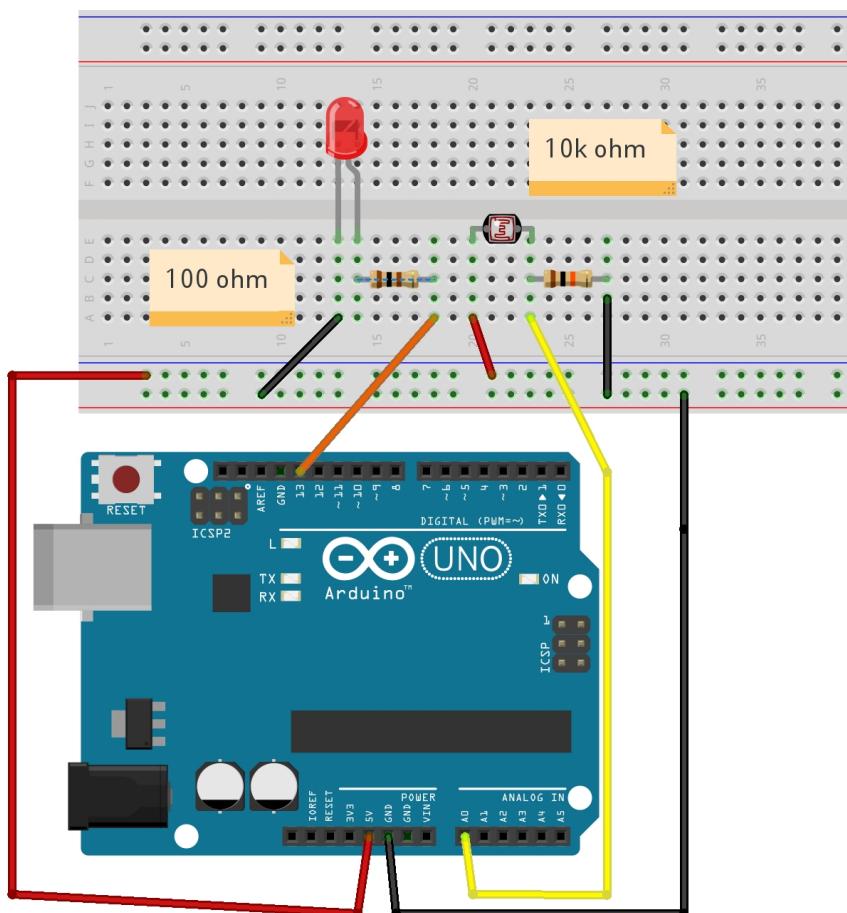


# Projeto 2: Controle LED com LDR

## → Materiais adicionais:

- ✓ 1 LED
- ✓ 1 resistores de 100 ohms
- ✓ 1 resistor de 10k ohm
- ✓ 1 LDR

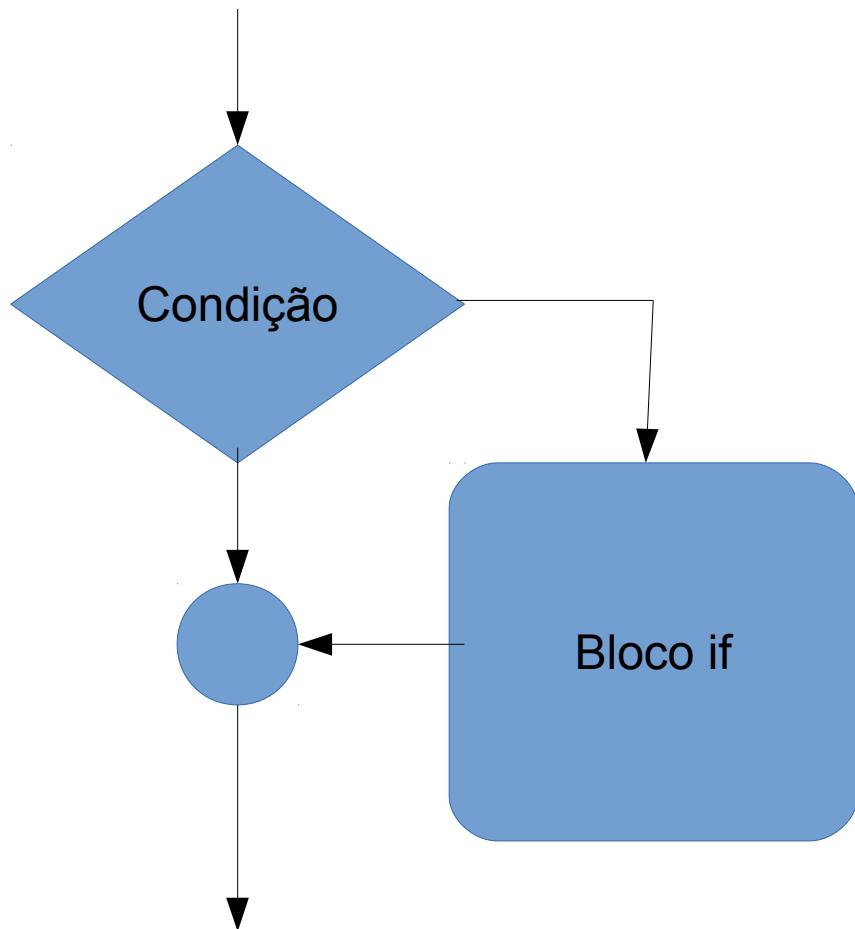
## → Montagem:



## → Código

```
float val_lum;  
int luminosidade;  
byte LDRpin=A0;  
byte LED = 13;  
  
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    val_lum = analogRead(LDRpin);  
    luminosidade = val_lum*0.0977;  
    Serial.write("Luminosidade: ");  
    Serial.println(luminosidade);  
  
    if (luminosidade < 50)  
    {  
        digitalWrite(LED, HIGH);  
    }  
    else  
    {  
        digitalWrite(LED, LOW);  
    }  
    delay(1000);  
}
```

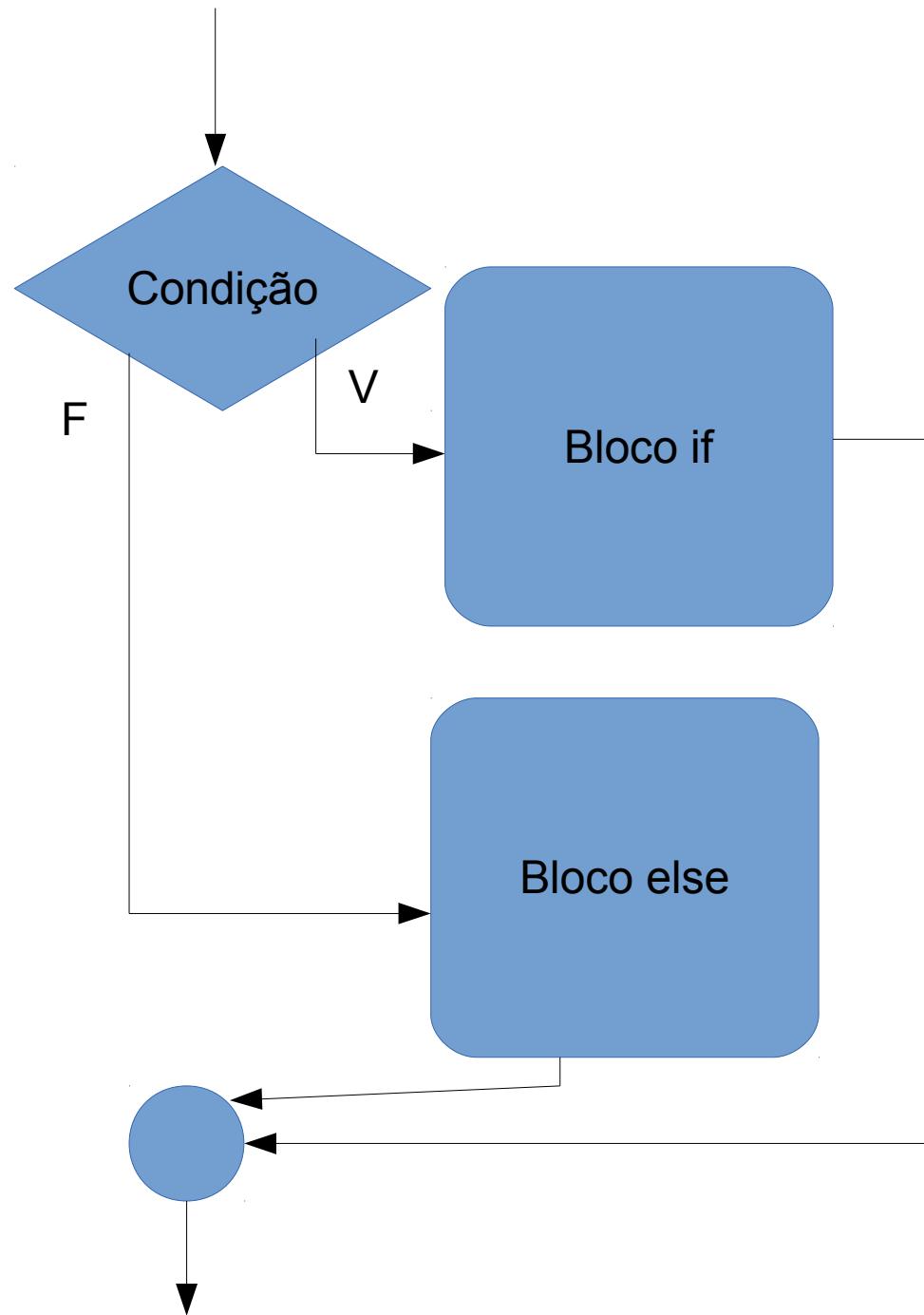
# Desvio condicional: if



```
if (<condição>)
{
    <comando 1>;
    <comando 2>;
    ...
    <comando n>;
}
```

```
se <condição> então:
    <comando 1>;
    <comando 2>;
    ...
    <comando n>;
fim-se
```

# Desvio condicional: if-else



```
if (<condição>)
{
    <comandos V>;
}
else
{
    <comandos F>;
}
```

```
se <condição> então:
    <comandos V>;
senão
    <comandos F>;
fim-se
```

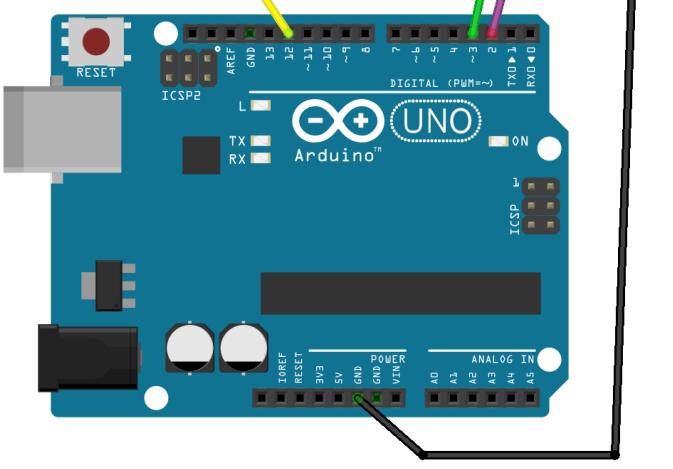
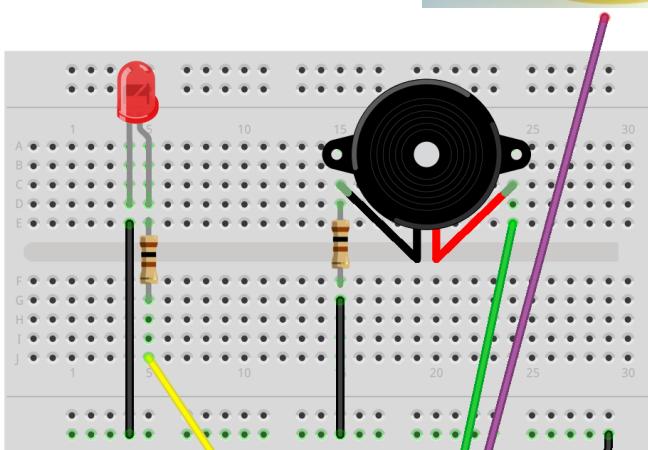
# Projeto 3: Banana CapSense

## → Materiais :

- ✓ 1 LED
- ✓ 1 Buzzer (Piezzo)
- ✓ 2 resistores de 100 ohms
- ✓ 1 banana



## → Montagem:



fritzing

## → Código:

```
#include <pincapsense.h>
#define PINO_LED 12 // pino digital
#define PINO_BUZZER 3 // pino digital PWM
#define PINO_CAP 2 // pino sensor cap.
#define NOTA_E7 2637 // Hz
int valor_cap;

void setup() {
    pinMode(PINO_LED, OUTPUT);
    pinMode(PINO_BUZZER, OUTPUT);
}

void loop() {
    valor_cap = readCapacitivePin(PINO_CAP);
    Serial.println(valor_cap);

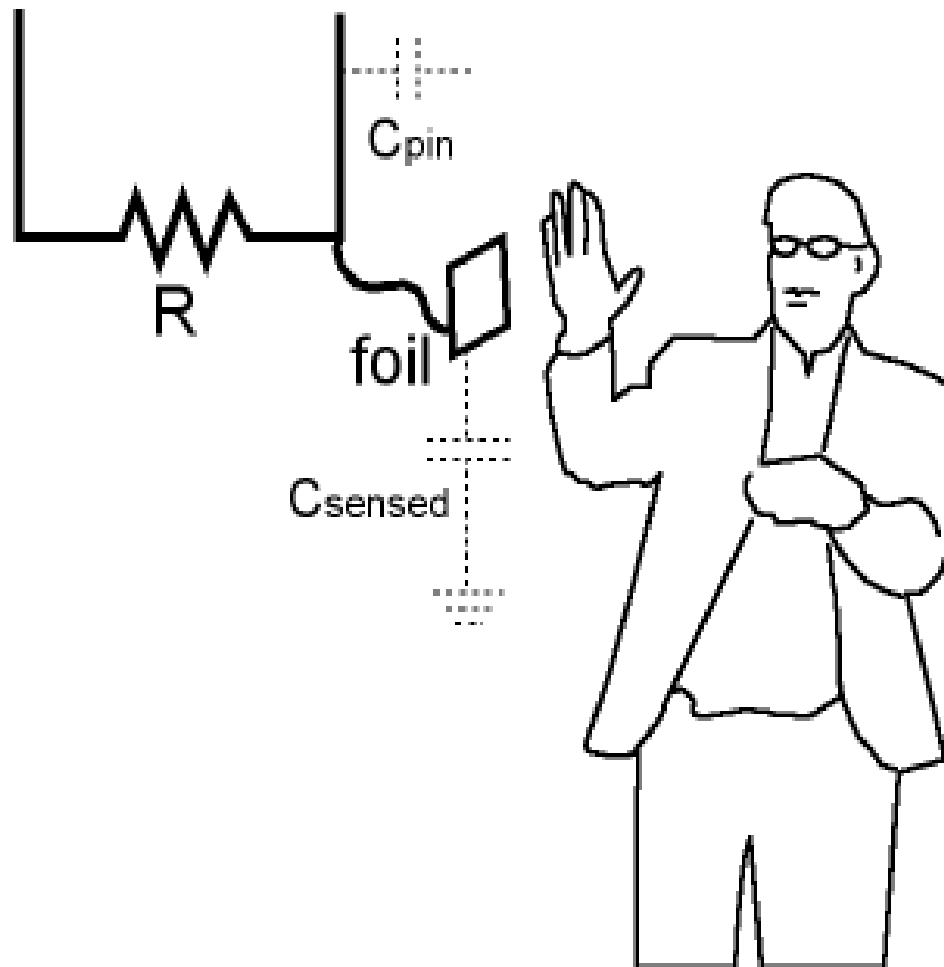
    if (valor_cap > 6) { // tocou na banana?
        digitalWrite(PINO_LED, HIGH);
        tone(PINO_BUZZER, NOTA_E7);
    }
    else {
        digitalWrite(PINO_LED, LOW);
        noTone(PINO_BUZZER);
    }
    delay(200);}
```

Fonte da imagem da banana:

[http://commons.wikimedia.org/wiki/Banana#mediaviewer/File:Bananen\\_Frucht.jpg](http://commons.wikimedia.org/wiki/Banana#mediaviewer/File:Bananen_Frucht.jpg)

# Sensor capacitivo: princípio

Send pin      Receive pin



→ **Capacitor:**  
componente que  
acumula cargas

→ Pode ser  
**descarregado:**  
basta ligar as placas  
por um fio + resistor

→ Pinos do Arduino já  
possuem um **resistor**  
**interno** associado

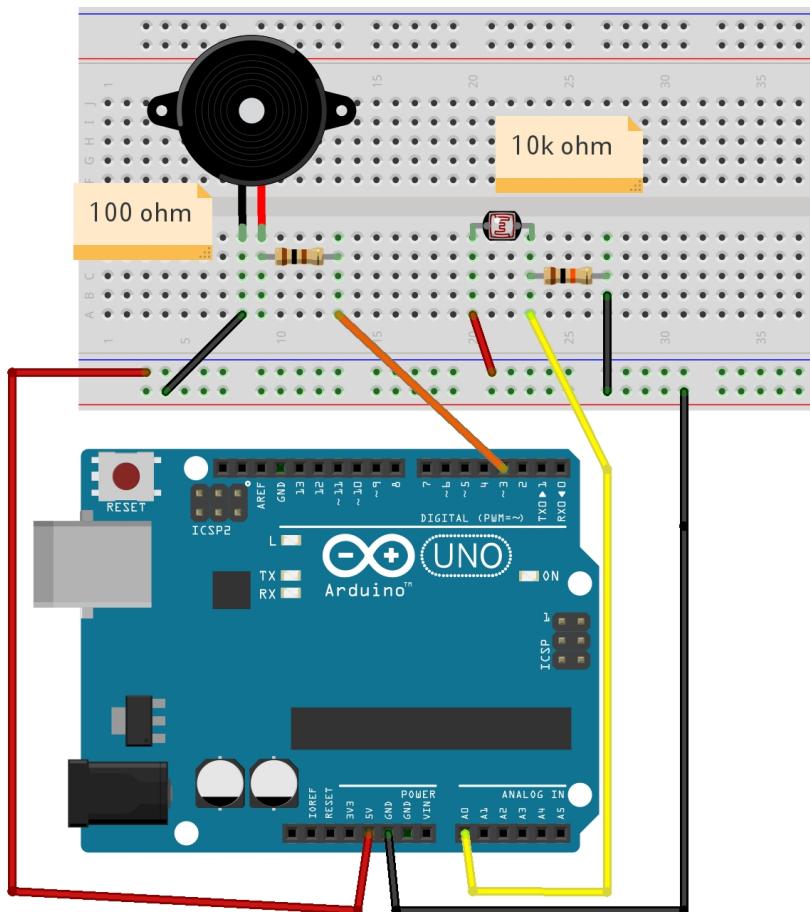
→ Leitura CapSense:  
Estima-se **ciclos de  
leitura** para quedas e  
subidas de tensão

# Projeto 4: Buzzer de Luz

## → Materiais :

- ✓ 1 Buzzer (Piezzo)
- ✓ 1 resistor de 100 ohms
- 1 resistor de 10k ohm
- 1 LDR

## → Montagem:



```
#define PINO_BUZZER 3 // pino digital PWM
#define LDRpin A0

float val_lum, periodo = 200;
int luminosidade;

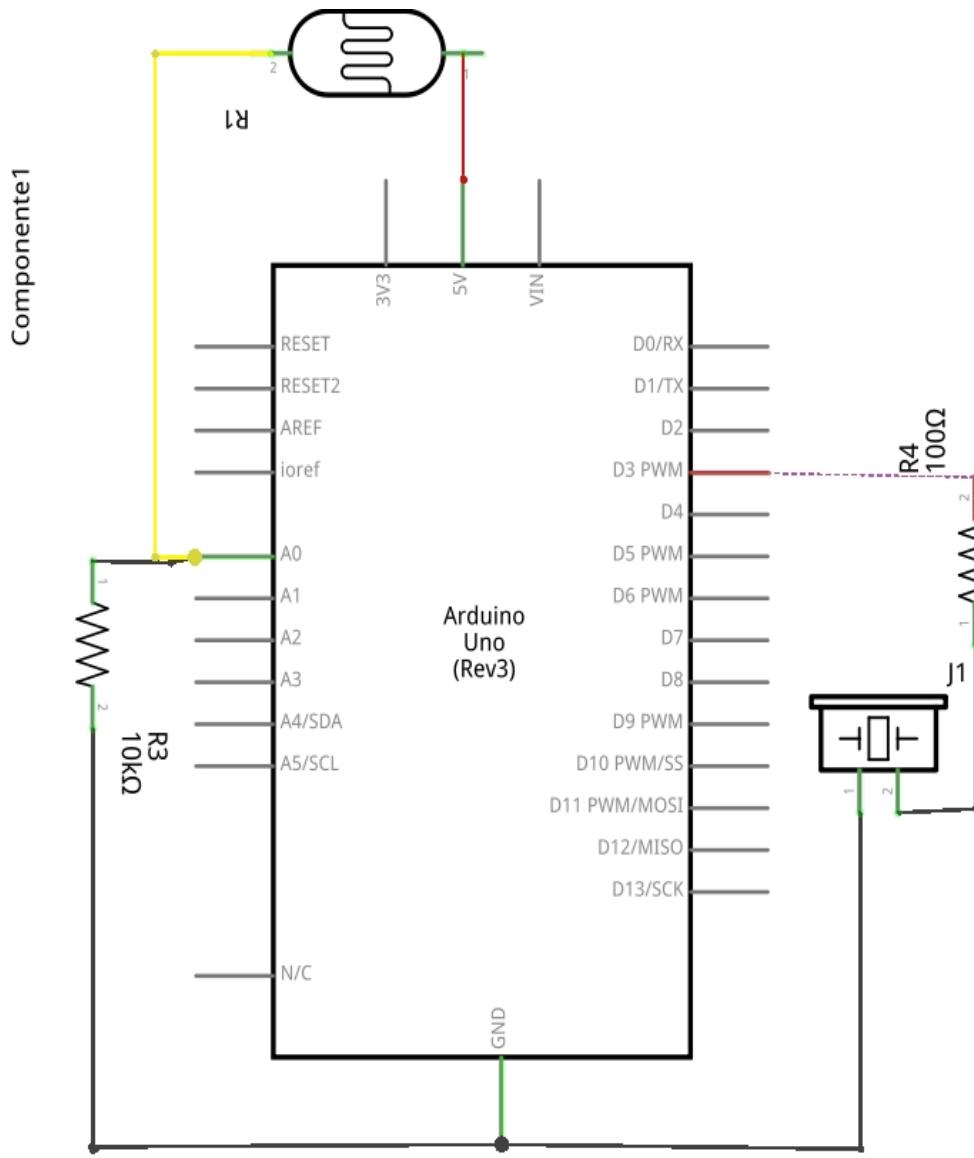
void setup() {
    pinMode(PINO_BUZZER, OUTPUT);
    pinMode(LDRpin, INPUT);
    Serial.begin(9600);
}

void loop() {
    val_lum = analogRead(LDRpin);
    luminosidade = val_lum*9.77;

    Serial.write("Luminosidade: ");
    Serial.println(luminosidade);

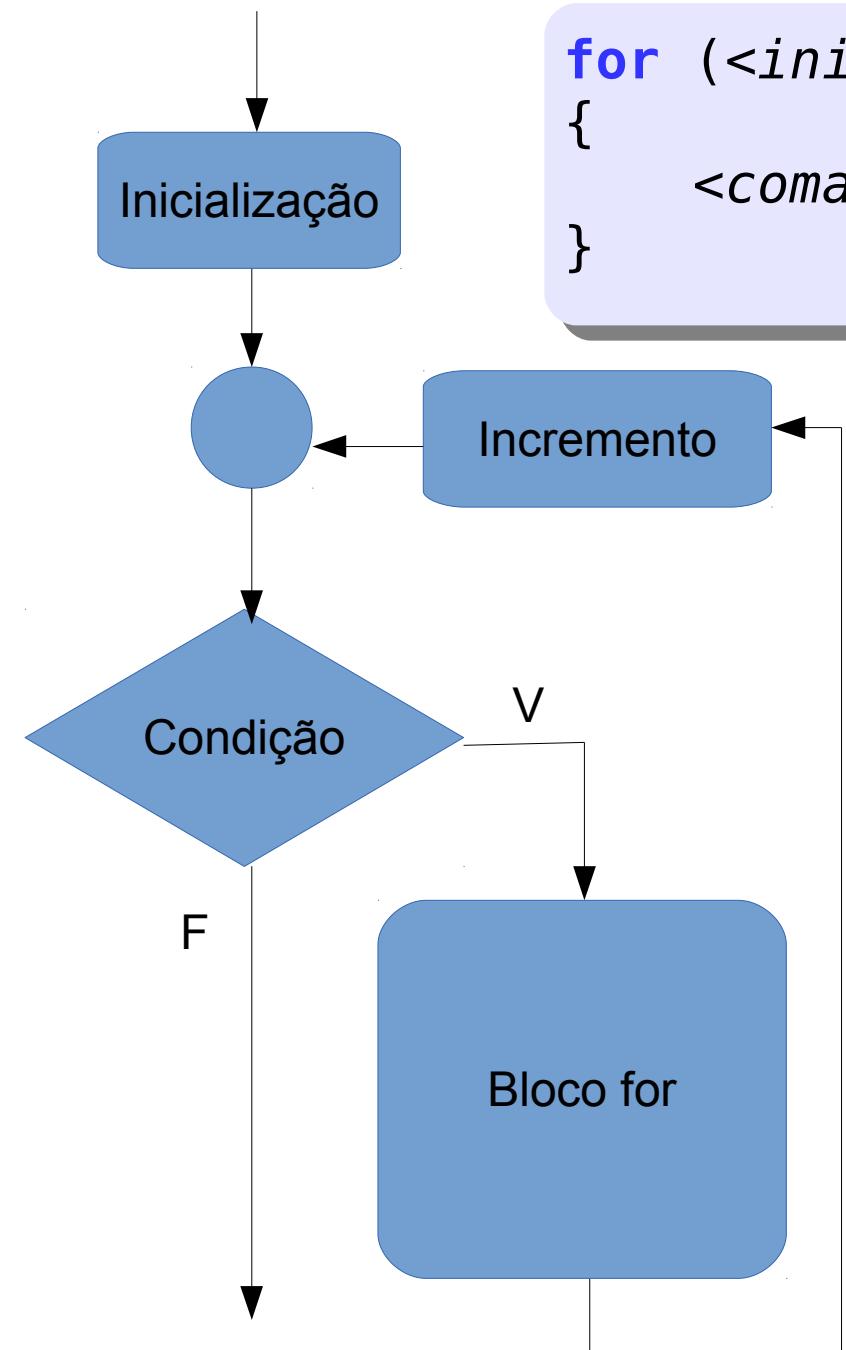
    tone(PINO_BUZZER, luminosidade);
    delay(periodo);
}
```

# Projeto 4: Buzzer de Luz



fritzing

# Laço de repetição: for



```
for (<inicialização>; <condição>; <incremento>)
{
    <comandos>;
}
```

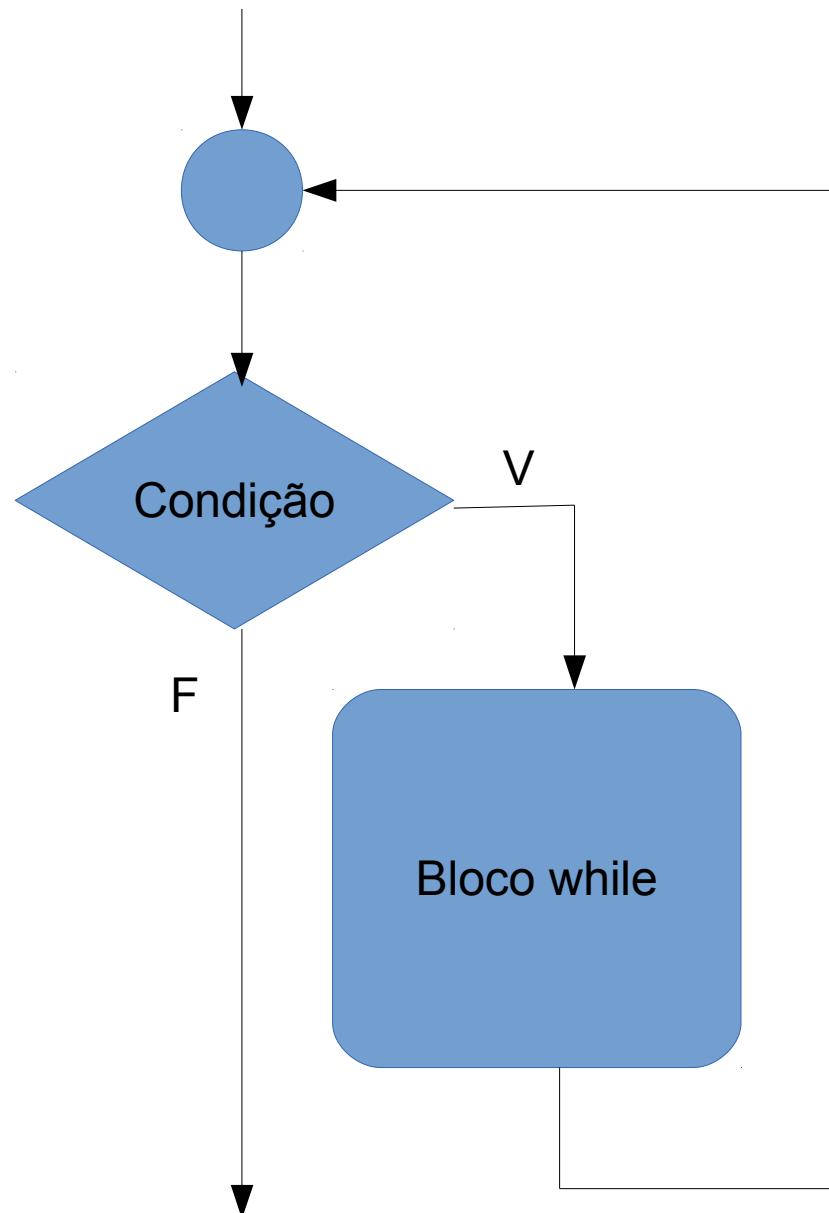
```
para <valor_inicial>
até <valor_final> faça:
    <comandos>;
fim-para
```

→ Melhor p/ contar repetições:

```
para i = 0 até 10 faça:
    <comandos>;
fim-para
```

```
for (i = 0; i < 10; i++)
{
    <comandos>;
}
```

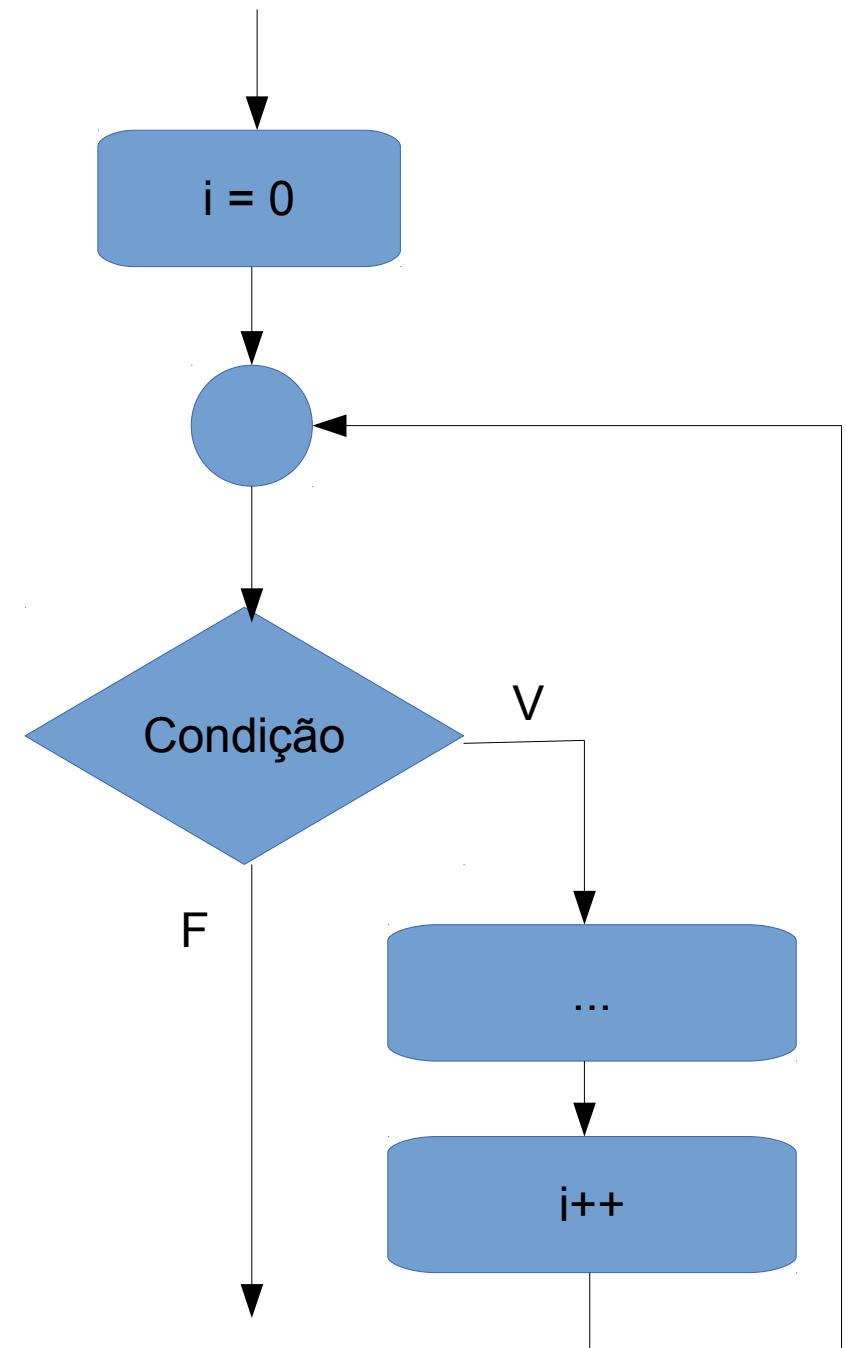
# Laço de repetição: while



```
while (<condição>)
{
    <comando 1>;
    <comando 2>;
    ...
    <comando n>;
}
```

```
equanto <condição>
faça:
    <comando 1>;
    <comando 2>;
    ...
    <comando n>;
fim
```

# loop while: padrão bastante comum



→ Quando usado para contar repetições:

```
i = 0;  
equanto i < 10 faça:  
    ...  
    i ← i + 1;  
fim-enquanto
```

```
i = 0;  
while (i < 10)  
{  
    ...  
    i++;  
}
```