

Coleta e Processamento de Dados de Sensores com Arduino e R

Lincoln Sousa de Oliveira
Marcson de Azevedo Araújo

Objetivos

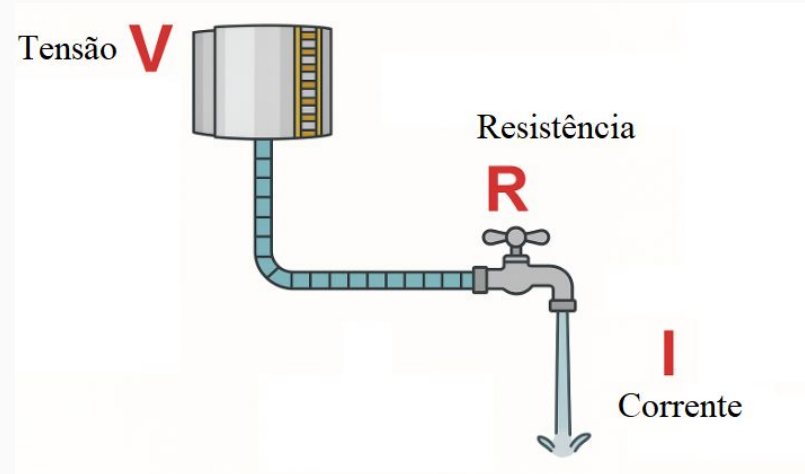
- Conceitos Básicos de Eletrônica
- Introdução a Arduino
- Simulação e Montagem de Circuitos
- Exercício prático com Arduino
- Exercício prático com Arduino e R

Eletrônica

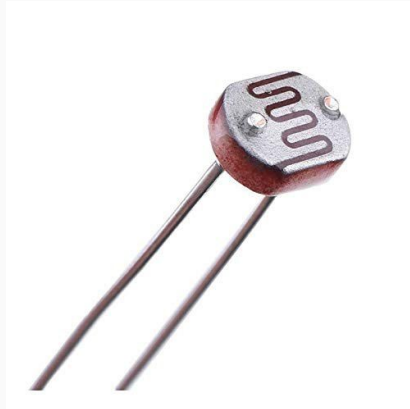
- A eletrônica é a área da engenharia elétrica que se dedica ao estudo e desenvolvimento de dispositivos e circuitos que utilizam componentes eletrônicos
- Permite a interação entre dispositivos físicos e sistemas digitais.

Corrente, Tensão e Resistência

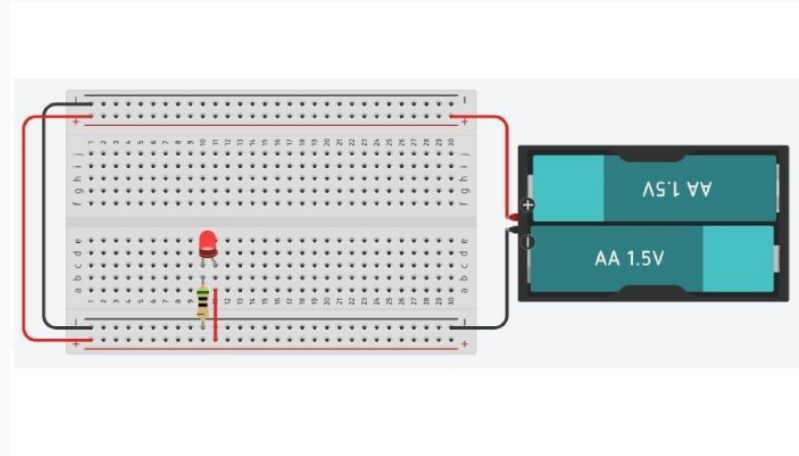
- Corrente: É o fluxo de elétrons que passa por um condutor. Quanto mais elétrons fluem, maior a corrente.
- Tensão: É a “força” que empurra os elétrons através do circuito. Também chamada de diferença de potencial.
- Resistência: É a oposição à passagem da corrente elétrica. Quanto maior a resistência, menor a corrente que passa.
- Lei de Ohm: $V = R * I$



Exemplo de Componentes



Protoboard



O que é Arduino?

- Plataforma open-source de prototipagem eletrônica.
- Combina hardware (placa) e software (IDE de programação).
- Fácil de usar, ideal para iniciantes e projetos educacionais.
- Permite interação com sensores, atuadores e outros dispositivos.
- Muito usado em automação, robótica, IoT e projetos.

Tipos de Arduino



Arduino UNO



Arduino Leonardo



Arduino MiniPro

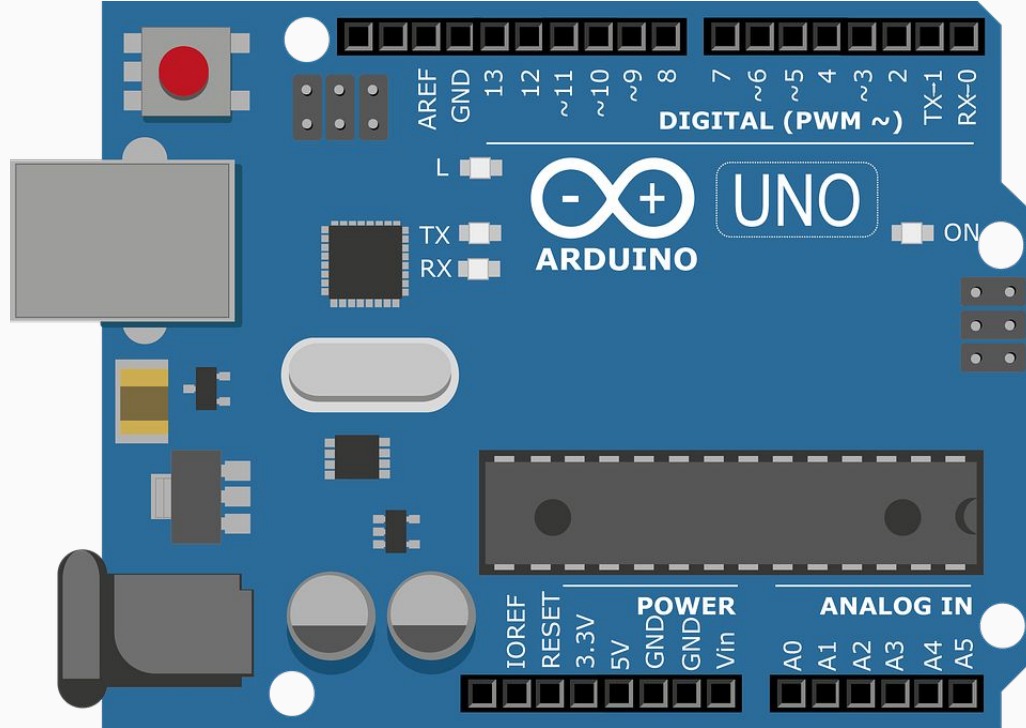


Arduino MEGA

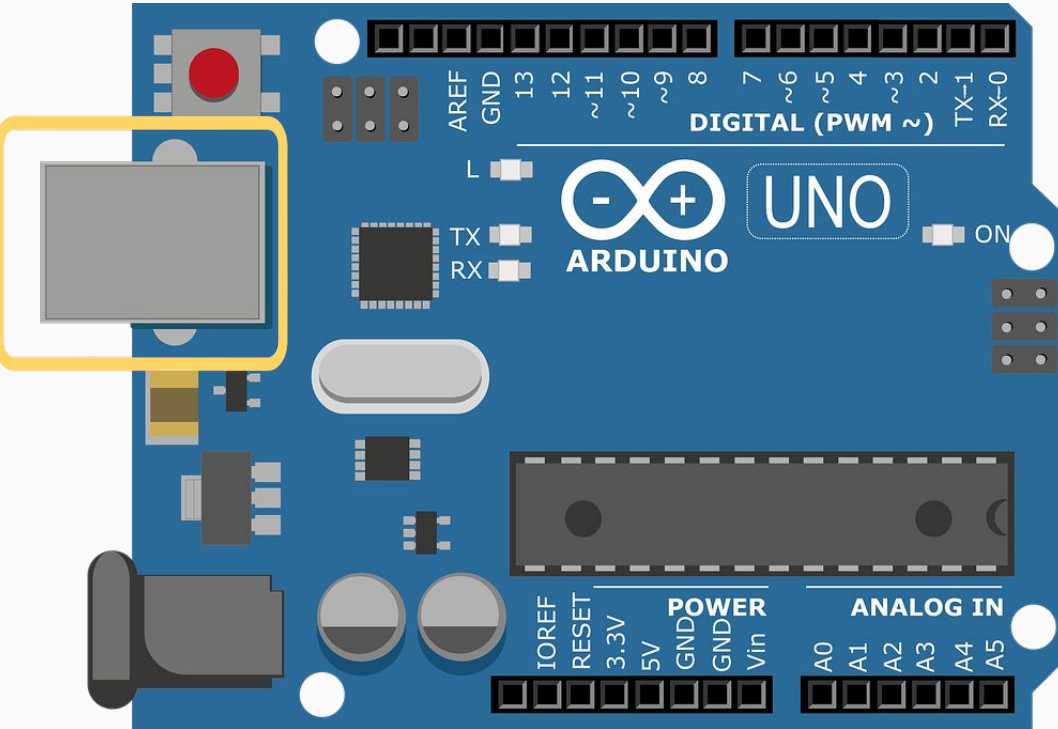


Arduino DUE

Arduino UNO



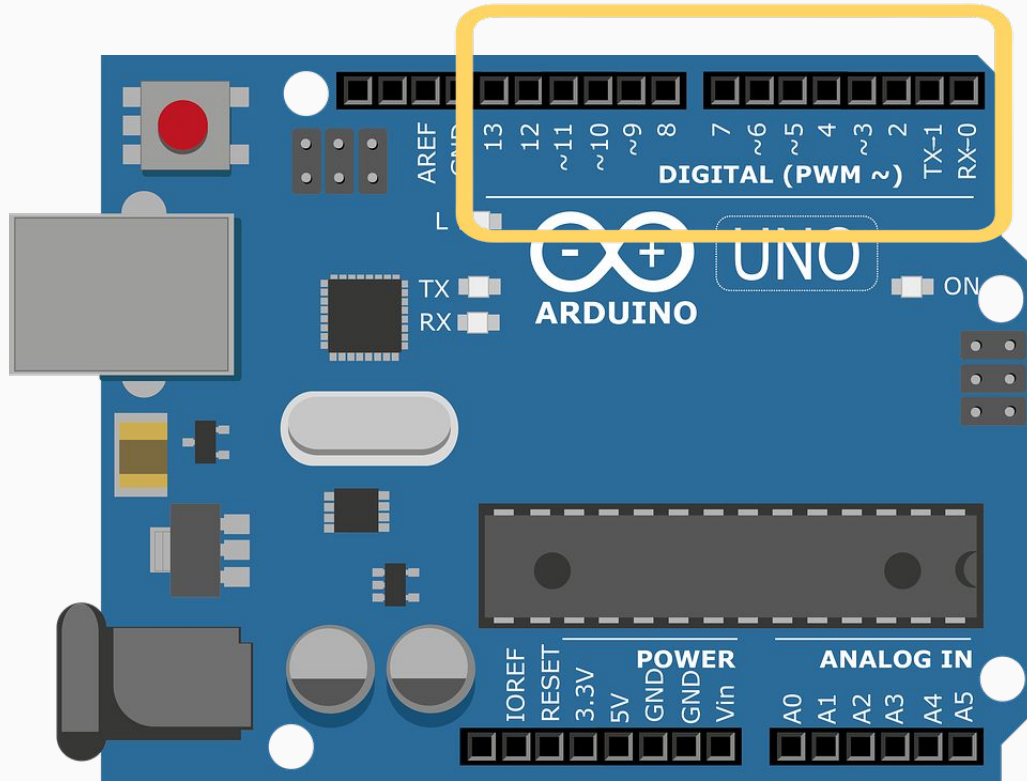
Arduino UNO



Porta USB B (Arduino) -> USB (Computador)

- Faz a conexão do Arduino ao computador.
- Serve para:
 - **Enviar o código** (upload).
 - **Alimentação** da placa (5V).
 - **Comunicação** serial de dados

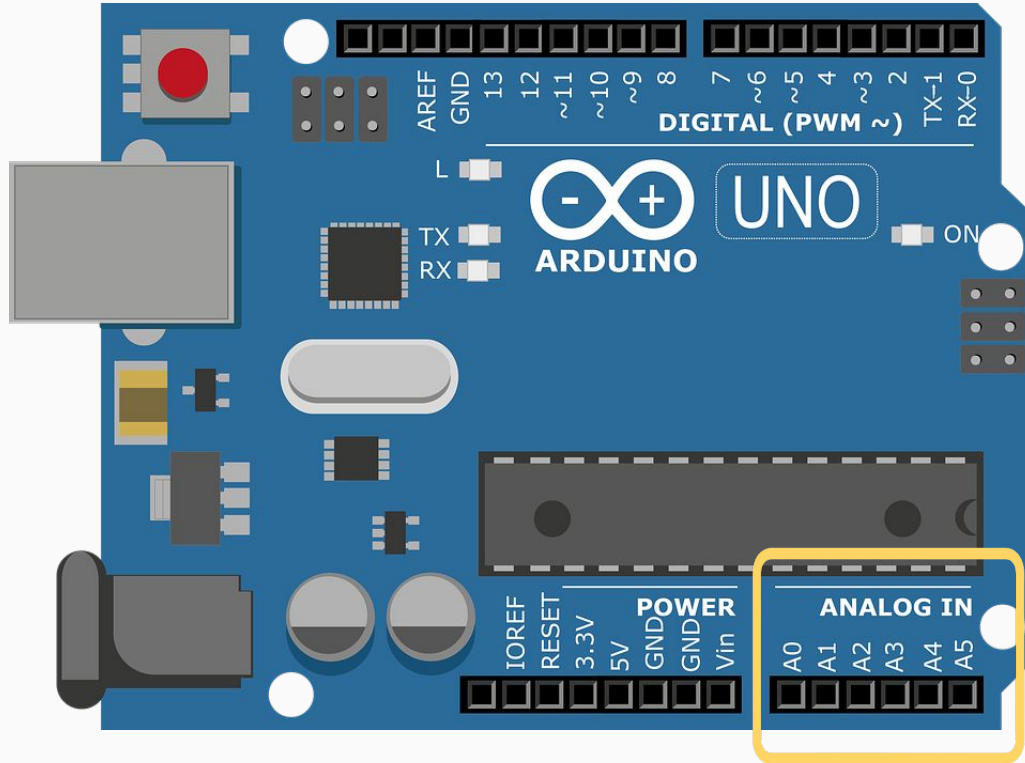
Arduino UNO



Portas Digitais (D0 a D13)

- Funcionam como **entrada ou saída digital** (valores: HIGH ou LOW).
- Ex: acionamento de LED, leitura do botão, sensor digital.
- Algumas têm funções especiais:
 - **D0 (RX) e D1 (TX)**: comunicação serial (evite usar em projetos como porta digital).
 - **D3, D5, D6, D9, D10, D11**: suporte a **PWM** (modulação por largura de pulso)

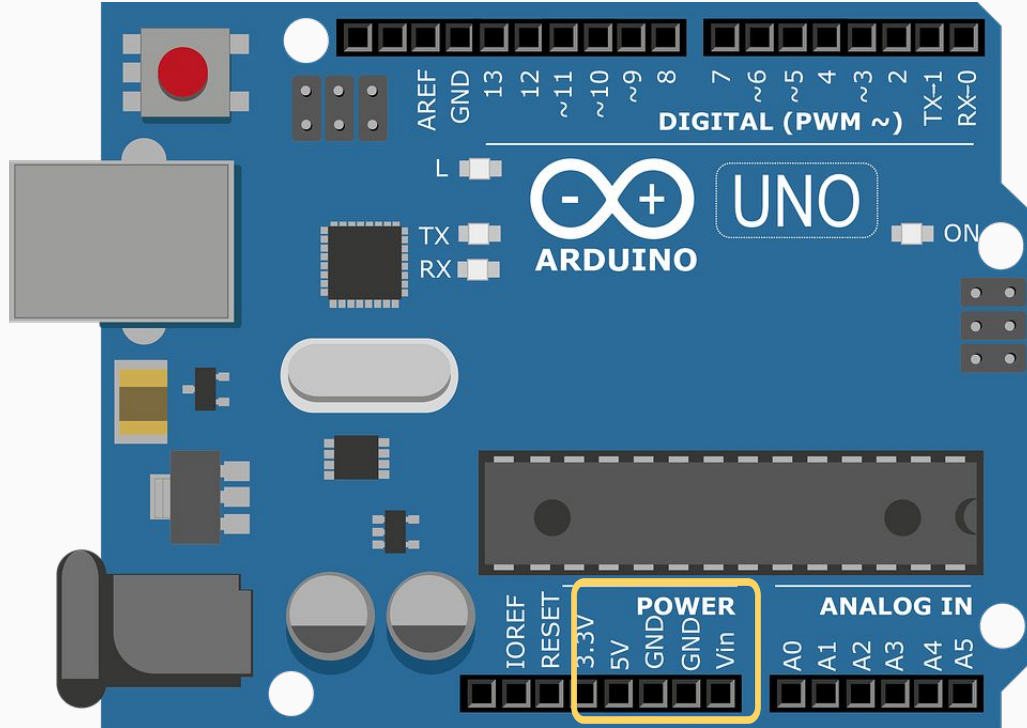
Arduino UNO



Portas Analógicas (A0 a A5)

- Recebem sinais analógicos (0 a 5V) → convertidos em valores digitais (0 a 1023).
- Usadas para sensores que variam continuamente, como LDR (luz), Gás, etc.
- Podem ser usadas como portas digitais também, se necessário.

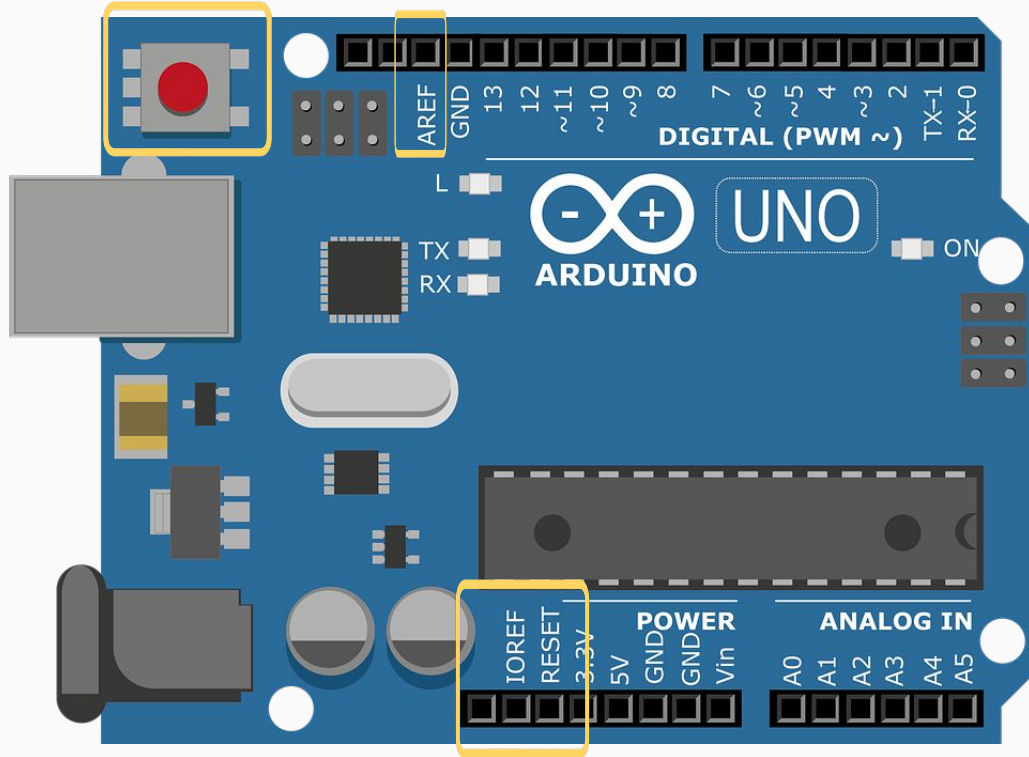
Arduino UNO



Portas de Alimentação

- **GND:** Terra (referência de 0V) — várias disponíveis.
- **5V:** Tensão regulada fornecida pelo Arduino (para sensores e componentes).
- **3.3V:** Alimentação de 3,3V (para dispositivos que não suportam 5V).
- **VIN:** Entrada de tensão (6–12V) se estiver usando fonte externa.

Arduino UNO



Pinos especiais

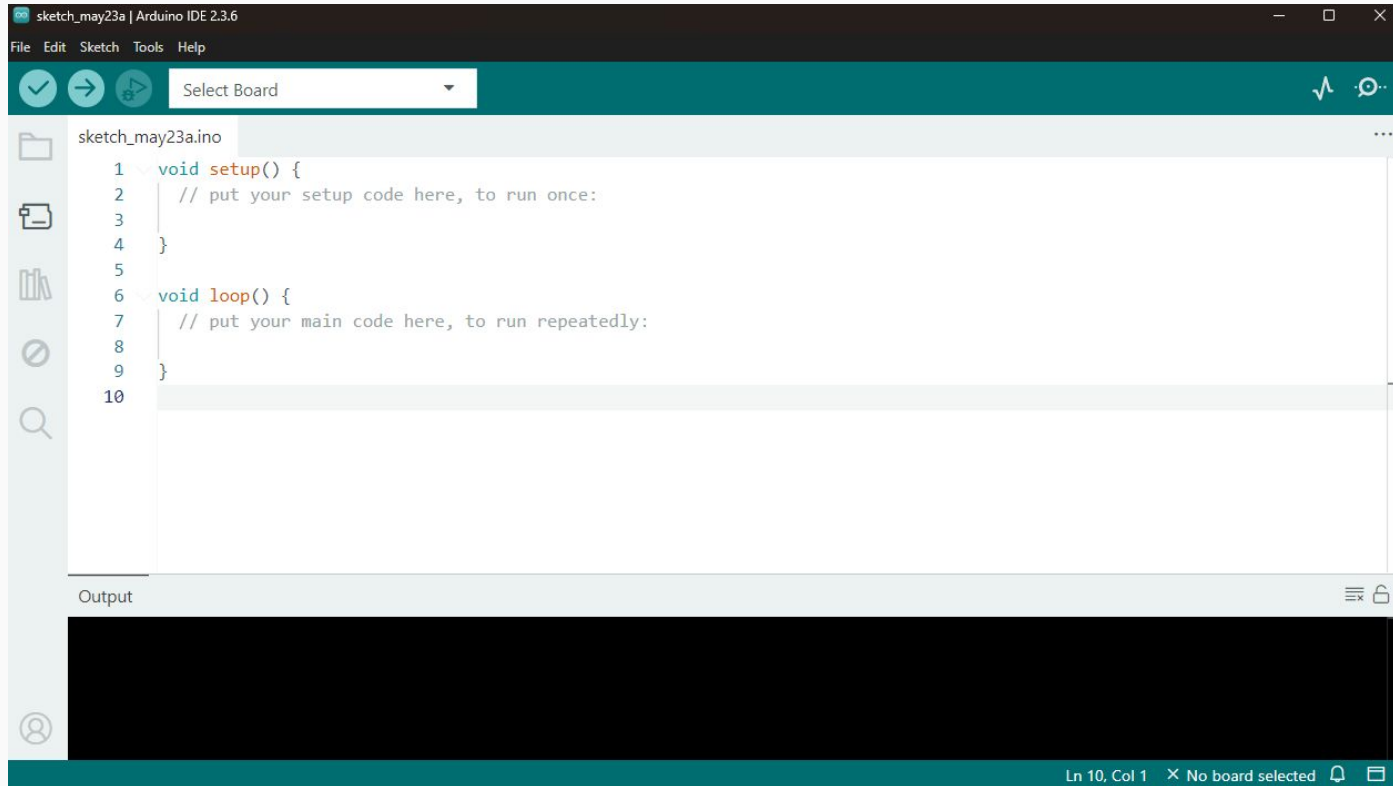
- **AREF:** Referência externa para entrada analógica (pouco usado por iniciantes).
Permite definir uma tensão máxima personalizada para leituras analógicas.
- **RESET:** Reinicia o programa do Arduino (também há botão físico na placa).
Pode ser usado por circuitos externos para reiniciar a placa automaticamente.
- **IOREF:** Fornece a tensão de operação da placa (normalmente 5V).
Usado por shields para saber se devem operar com 5V ou 3.3V.

Linguagem do Arduino

- Baseada em C/C++, com simplificações para facilitar o uso.
- Pode ser programado na Arduino IDE, VS Code, entre outras ferramentas
- Estrutura básica com duas funções principais:
 - `setup()` → Executa uma única vez após iniciar.
 - `loop()` → Executa continuamente enquanto estiver ligado.
- Possui funções integradas como `digitalWrite()`, `analogRead()`, `delay()`, etc.
- Uso de bibliotecas prontas facilita a integração com sensores, atuadores e módulos.
- Programas são chamados de sketches.

Arduino IDE

<https://www.arduino.cc/en/software/>



Funções principais

💡 Controle de Pinos Digitais

- `pinMode(pino, modo)` → Define se o pino será **INPUT**, **OUTPUT**
- `digitalWrite(pino, valor)` → Escreve **HIGH** ou **LOW** em um pino digital
- `digitalRead(pino)` → Lê o valor **HIGH** ou **LOW** de um pino digital

📊 Leitura e Escrita Analógica

- `analogRead(pino)` → Lê um valor analógico (0 a 1023) de um pino
- `analogWrite(pino, valor)` → Escreve um sinal analógico em um pino

🕒 Temporização

- `delay(ms)` → Pausa o programa por milissegundos
- `millis()` → Retorna o tempo (em ms) desde que o programa começou
- `delayMicroseconds(us)` → Pausa por microssegundos

☐ Comunicação Serial

- `Serial.begin(baudrate)` → Inicia a comunicação serial
- `Serial.print(valor)` → Envia dados para o monitor serial
- `Serial.println(valor)` → Envia dados com quebra de linha

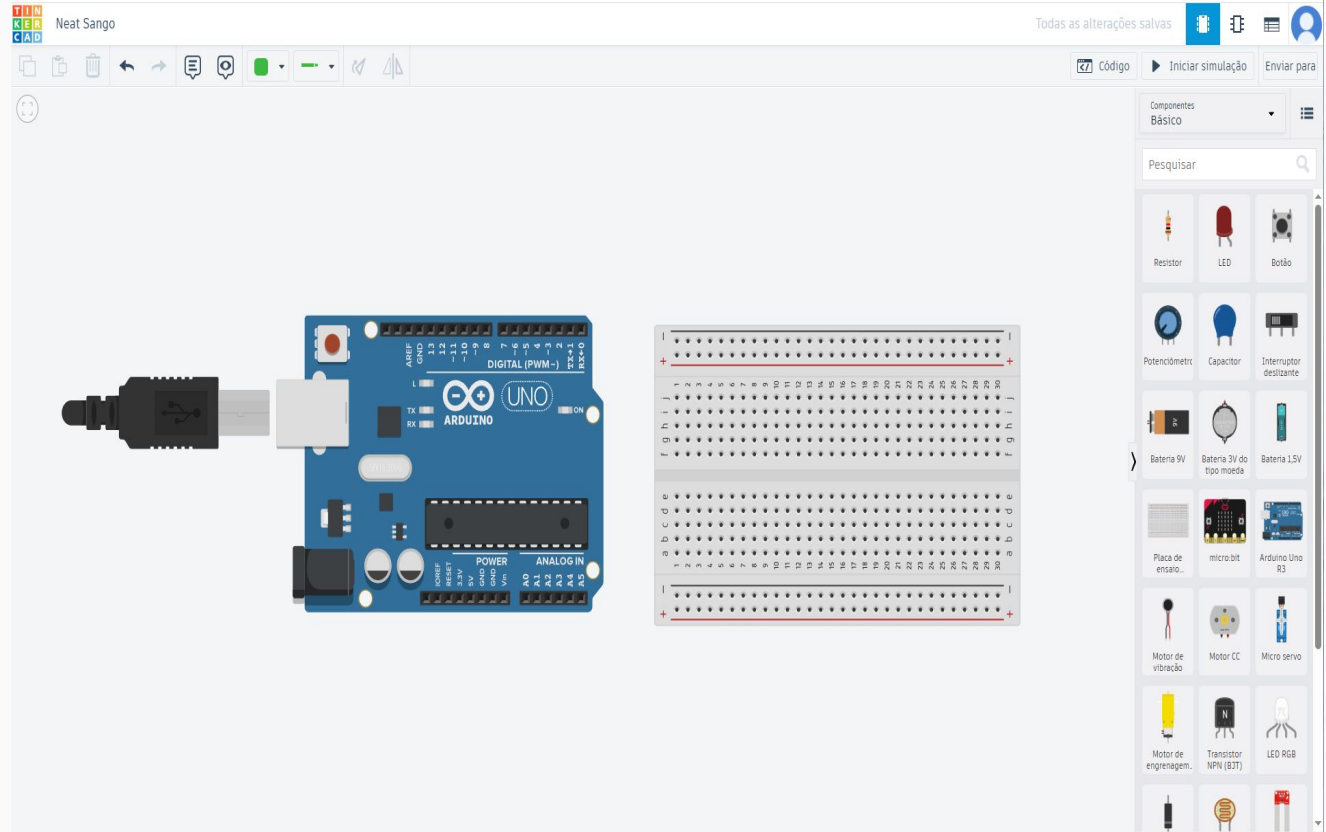
🧠 Definição de Variáveis e Constantes

- `#define NOME valor` → Define uma constante sem tipo
- `const tipo nome = valor;` → Define uma constante com tipo
- `int nome = valor;` → Declara uma variável inteira
- `float nome = valor;` → Declara uma variável de ponto flutuante
- `bool nome = valor;` → Declara uma variável booleana
- `char nome = 'c';` → Declara uma variável do tipo caractere
- `String nome = "texto";` → Declara uma variável do tipo string

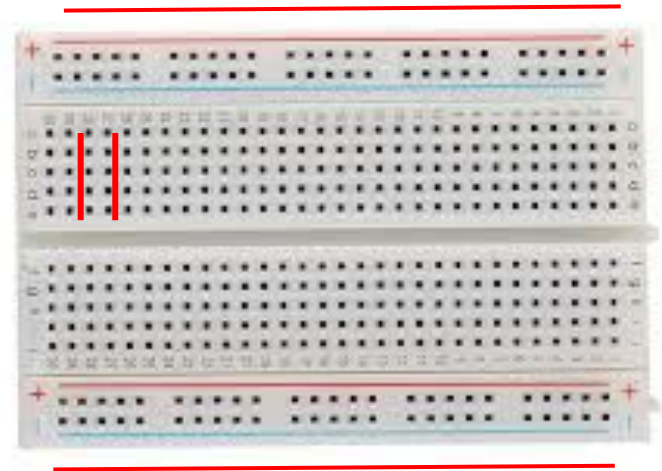
Simulação: Tinkercad

<https://www.tinkercad.com/join>

1. Criar conta pessoal
2. Criar projeto de circuito
3. Acender LED



Funcionamento do protoboard

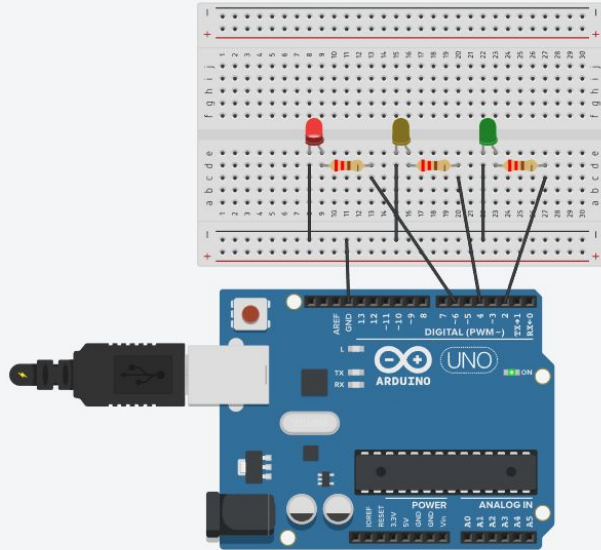


Prática: Semáforo

- Projetar um protótipo de semáforo utilizando LEDs de três cores diferentes, seguindo a seguinte rotina:
 - Verde: 6 segundos
 - Amarelo: 2 segundos
 - Vermelho: 5.5 segundos
 - Retorno ao início



Gabarito: Semáforo



```
1 #define VERDE 2
2 #define AMARELO 4
3 #define VERMELHO 6
4
5 void setup()
6 {
7   Serial.begin(9600);
8   pinMode(VERDE, OUTPUT);
9   pinMode(AMARELO, OUTPUT);
10  pinMode(VERMELHO, OUTPUT);
11 }
12
13 void loop()
14 {
15   digitalWrite(VERDE, HIGH);
16   Serial.println("VERDE");
17   delay(6000);
18   digitalWrite(VERDE, LOW);
19   digitalWrite(AMARELO, HIGH);
20   Serial.println("AMARELO");
21   delay(2000);
22   digitalWrite(AMARELO, LOW);
23   digitalWrite(VERMELHO, HIGH);
24   Serial.println("VERMELHO");
25   delay(5500);
26   digitalWrite(VERMELHO, LOW);
27 }
```

Monitor serial

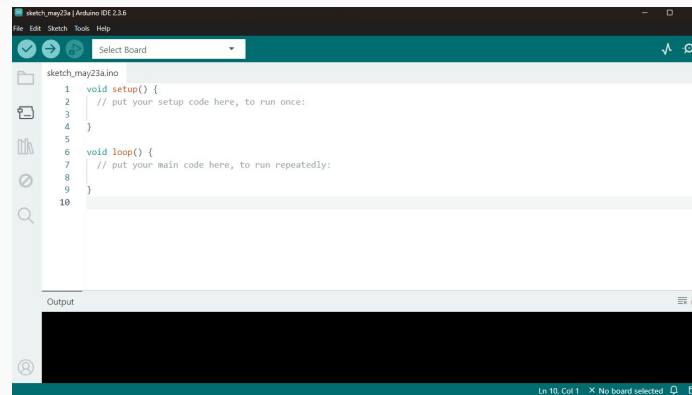
VERDE
AMARELO
VERMELHO

Env. Apag. AA

Preparo do Arduino para conexão

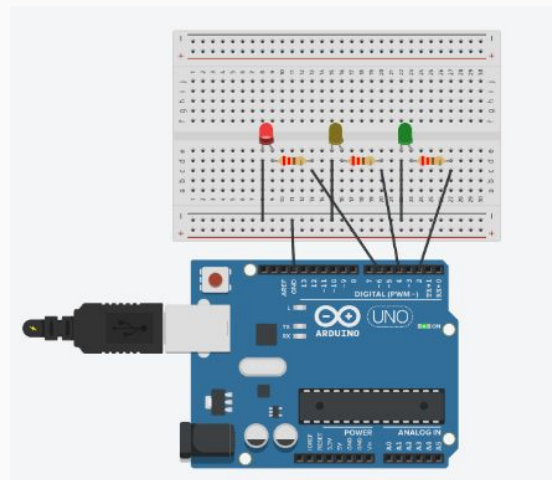
Na IDE do Arduino - preparar código para

- receber informações
- preparar
- enviar pela porta serial



No Arduino - preparar equipamento para

- receber o código preparado
- conectar componentes
- conectar no computador via USB



Conexão serial: Semáforo

Conexão do Arduino ao Computador (via USB)

- A comunicação entre Arduino e computador é feita via porta USB.
- O Arduino envia dados pela porta serial para serem lidos no R.

Escrita e Leitura de Dados

- As informações devem ser escritas no Arduino usando `Serial.print()` ou `Serial.println()`.
- O R faz a leitura dessas informações por meio da conexão serial.

Atenção ao Uso da Porta Serial

- A porta serial **não pode estar aberta em dois lugares ao mesmo tempo.**
 - Se estiver aberta no Monitor Serial do Arduino, o R **não conseguirá acessá-la.**
 - Se estiver aberta no R, é necessário **fechá-la antes de usar em outro programa.**

R, RSTUDIO, ABERTURA DA CONEXÃO SERIAL

Para conexão com o R

- Identificar a porta
- Preencher detalhes da conexão

Processo de escrita e leitura de informações

□ Comunicação Serial NO ARDUINO

- `Serial.begin(baudrate)` → Inicia a comunicação serial
- `Serial.print(valor)` → Envia dados para o monitor serial
- `Serial.println(valor)` → Envia dados com quebra de linha

□ Comunicação Serial NO R

- `conexao <- serial::serialConnection(port = porta, mode = modo)` → parâmetros para conexão
- `open(conexao)` → Inicia a comunicação serial
- `leitura_raw <- serial::read.serialConnection(conexao)` → Recebe dados do `Serial.println()`
- `close(conexao)` → Encerra a comunicação serial

R - Conexão com o Arduino

FUNÇÕES DO R PACOTE PARA LEITURA DO ARDUINO

Pacote Necessário: `serial`: Para comunicação com a porta serial (ler dados do Arduino).

- `serialConnection()`: **Estabelece a Conexão Serial**

É necessário especificar a porta, a taxa de transmissão (baud rate) e outros parâmetros da comunicação.

Ex.

```
con_arduino <- serialConnection(port = 'COM3', mode  
= "9600,n,8,1")
```

9600 *baudrate*, **n** – *Paridade (parity)*, **8** – *Data bits*. **1** – *Stop bit*

Outras funções

`open()`: **Abre a Conexão Serial**

Depois de criar o objeto de conexão com `serialConnection()`, você precisa abri-lo para iniciar a comunicação.

`isOpen()`: **Verifica o Status da Conexão**

Útil para verificar se a conexão está ativa antes de tentar ler ou escrever dados.

Basicamente, `listPorts()` **lista todos os nomes de portas seriais que o R consegue detectar** e acessar no seu computador. Isso elimina a adivinhação sobre qual "COM" (no Windows) ou "/dev/tty" (no Linux/macOS) corresponde ao seu Arduino.

`close()`: **Fecha a Conexão Serial**

É **crucial** fechar a conexão serial quando você terminar de usá-la para liberar a porta e evitar problemas de acesso por outros programas.

`read.serialConnection()`: Lê Dados da Porta Serial

Esta é a função principal para receber dados do Arduino. Ela lê bytes ou caracteres da porta serial.

É a forma como trazemos para o R os valores observado no monitor serial.

`write.serialConnection()`: Envia Dados para a Porta Serial (Arduino)

Permite enviar strings ou bytes do R para o Arduino.

R - Conexão com o Arduino

Exemplo de leitura de informações enviadas do Arduino para o R pela conexão serial

```
1 library(serial)
2
3 listPorts()
4
5 porta <- "COM3"
6 baudrate <- "9600"
7
8 conexao <- serialConnection(port = porta,
9                             mode = paste0(baudrate,"n,8,1"))
10
11 open(conexao)
12
13 while(TRUE){
14   leitura <- read.serialConnection(conexao)
15   print(leitura)
16   Sys.sleep(10)
17 }
18
19 close(conexao)
20
21
```

```
[1] "AMARELOVERMELHO"
```

Preparo do R para conexão

No Rstudio - preparar código para

- identificar a porta
- realizar a conexão
- receber informações
- estruturar as informações recebidas
- preparar para armazenamento
- armazenar

```
1 library(serial)
2
3 listPorts()
4
5 porta <- "COM3"
6 baudrate <- "9600"
7
8 conexao <- serialConnection(port = porta,
9                             mode = paste0(baudrate,"n,8,1"))
10
11 open(conexao)
12
13 while(TRUE){
14   leitura <- read.serialConnection(conexao)
15   leitura <- strsplit(leitura, ",")[[1]]
16   print(leitura)
17   Sys.sleep(10)
18 }
19
20 close(conexao)
```

```
> while(TRUE){
+   leitura <- read.serialConnection(conexao)
+   leitura <- strsplit(leitura, ",")[[1]]
+   print(leitura)
+   Sys.sleep(10)
+ }
character(0)
[1] "AMARELO" "VERMELHO"
[1] "VERDE"   "AMARELO" "VERMELHO"
```

Agora que a informação está em memória no R, como armazenar?

Preparo do R para conexão - Banco de dados SQLITE

Banco de dados **SQLite**

- É um **banco de dados relacional** (SQL). Organiza os dados em tabelas com colunas e linhas, e as relações entre essas tabelas são definidas.
- **Armazenamento de Dados:** Os dados são armazenados em um **arquivo único no disco**. Em vez de usar um servidor separado, todo o banco de dados está contido em um único arquivo **.db**.
- **Natureza "Embedded" (Embutido):** Funciona como uma biblioteca embutida em aplicativos, sem a necessidade de um processo de servidor separado. Isso o torna **leve e fácil de integrar**.
- **Sintaxe SQL Padrão:** Utiliza a linguagem **SQL (Structured Query Language)** para gerenciar e consultar os dados, o que o torna familiar para quem já trabalha com outros bancos de dados relacionais.

Detalhe da conexão do SQLite para armazenar dados da conexão Serial

No contexto da aplicação que estamos trabalhando e do SQLite, é necessário um ajuste para que seja possível gerenciar escrita e leitura das informações do banco de dados. Isso é feito com o WAL.

O Write-Ahead Logging (WAL) surge como uma alternativa robusta e essencial para aprimorar a **concorrência e a resiliência**. Ao desacoplar as operações de leitura e escrita, o WAL permite que leitores acessem versões consistentes do banco de dados enquanto as escritas ocorrem em um arquivo de log separado. Essa abordagem elimina os bloqueios de escrita exclusivos que penalizam o desempenho em cenários de alta concorrência.

A implementação do WAL é uma evolução estratégica no SQLite, mitigando gargalos de performance e fortalecendo a tolerância a falhas. Para sistemas que exigem alta disponibilidade e operações simultâneas de leitura e escrita, sua compreensão e configuração adequada são cruciais para extrair o máximo potencial do banco de dados. <https://www.sqlite.org/wal.html>

Preparo do R para conexão

Conexão do R com arduino, para leitura e conexão com SQLite para armazenamento

Pacotes Necessários:

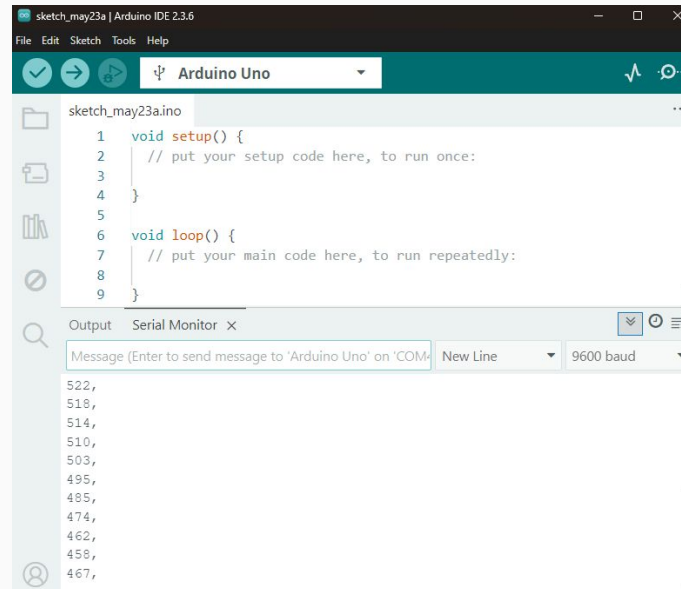
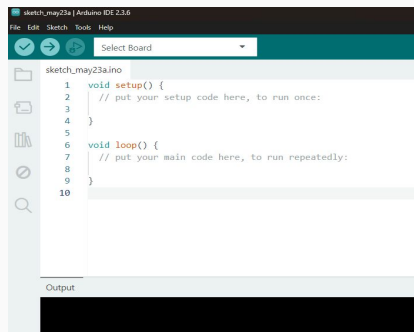
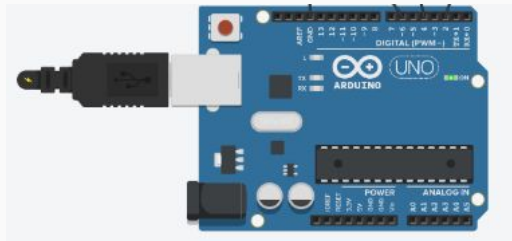
1. **serial**: para comunicação com a porta serial (ler dados do Arduino)
2. **DBI**: Interface padrão do R para bancos de dados. Ele fornece um conjunto comum de funções para interagir com diferentes sistemas de banco de dados
3. **RSQLite**: Driver específico para bancos de dados SQLite, que se integra ao **DBI**

Workflow completo

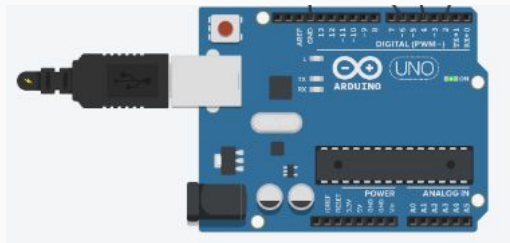
- Programação no Arduino
- Preparo de código para um Arduino printar dados na porta serial
- Montagem dos componentes eletrônicos
- Compilação do código e verificação no monitor serial da IDE do Arduino
- Abertura da conexão serial com o R
- Abertura da conexão do R com o banco de dados
- Armazenamento no Banco de dados

Programação no arduino

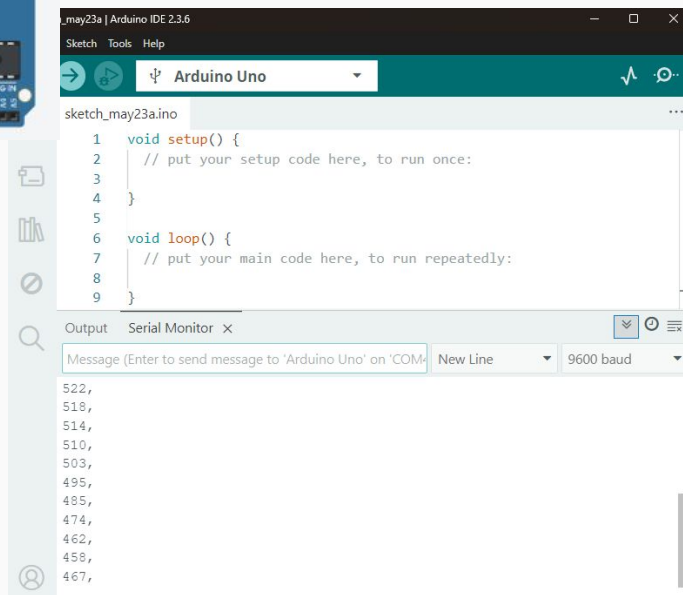
- Conectar o Arduino no computador via USB
- Abrir a IDE do Arduino, encontrar a porta
- Escrever o código
- Compilar e observar no monitor serial



Leitura no R



1. Ligar arduino no PC
2. Compilar código
3. No R, ler e armazenar no banco de dados



```
library(serial)
library(DBI)

listPorts()

porta <- "COM5"
baudrate <- "9600"

conexao <- serialConnection(port=porta,
                             mode = paste0(baudrate,"n,8,1"))
open(conexao)

while(TRUE) {
  leitura <- strsplit(leitura_raw, ",")[1]
  print(leitura)
  Sys.sleep(10)
}

close(conexao)
```

Preparo do R para conexão - Exemplo

```
library(DBI)

# Conectando a um banco SQLite (arquivo local)

con <- dbConnect(RSQLite::SQLite(),
"meubanco.sqlite")

dbExecute(con, "PRAGMA journal_mode = WAL;")

while(TRUE) {
  # PROCESSO DE LEITURA

  dbExecute(con, "BEGIN TRANSACTION")

  # SE LEITURA TEM VALORES VÁLIDOS

  dbExecute(con,
    paste0("INSERT INTO tabela(colunas) VALUES
(valores colunas)")
  )

  dbExecute(con, "COMMIT")

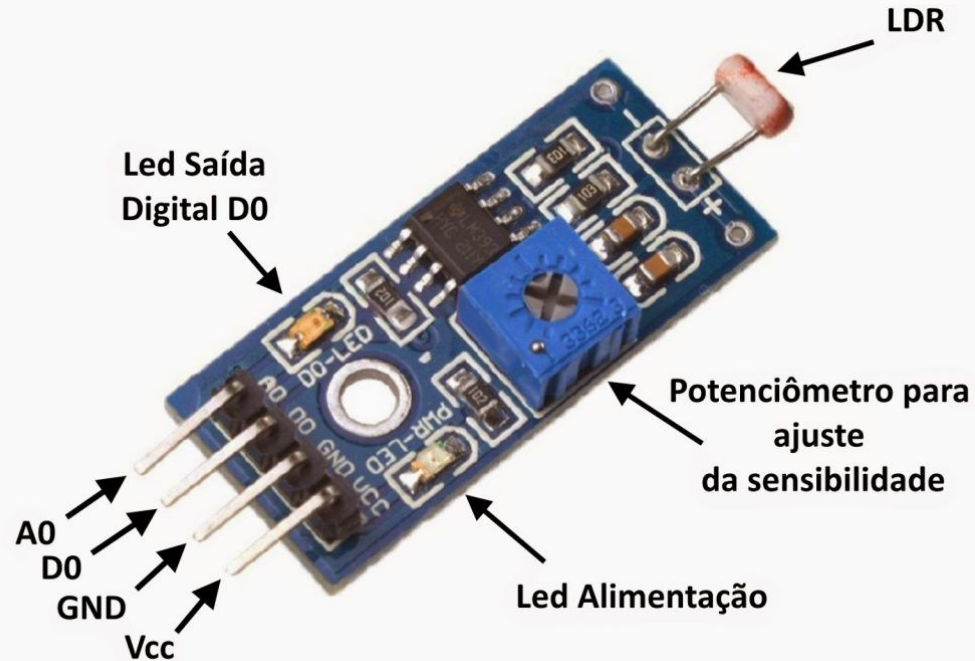
  Sys.sleep(10)
  # Aguarda 10 segundos entre as leituras
}

close(conexao)
```

Prática: Sensor de luz

- Projetar acionamento automático do LED quando a luz ambiente for detectada abaixo de um nível pré-estabelecido no sensor LDR.
- Use `Serial.println()` o valor da leitura analógica do sensor.

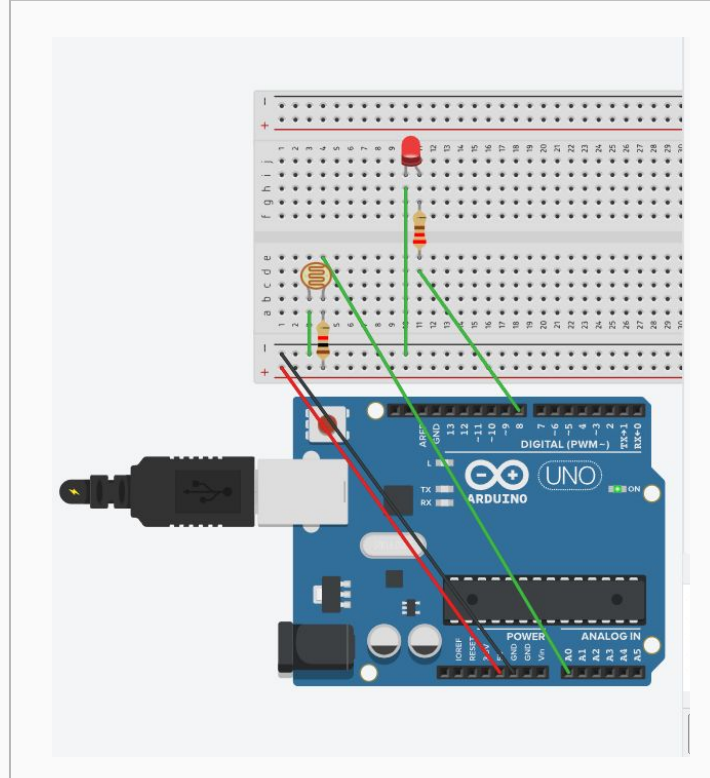
VAMOS A PRÁTICA



Prática final - Montagem

PARTE 1

- Montagem de arduino com Sensor (com um módulo real e no Tinkercad)
- Ligação do Arduino no PC



Prática final - IDE Arduino

PARTE 2

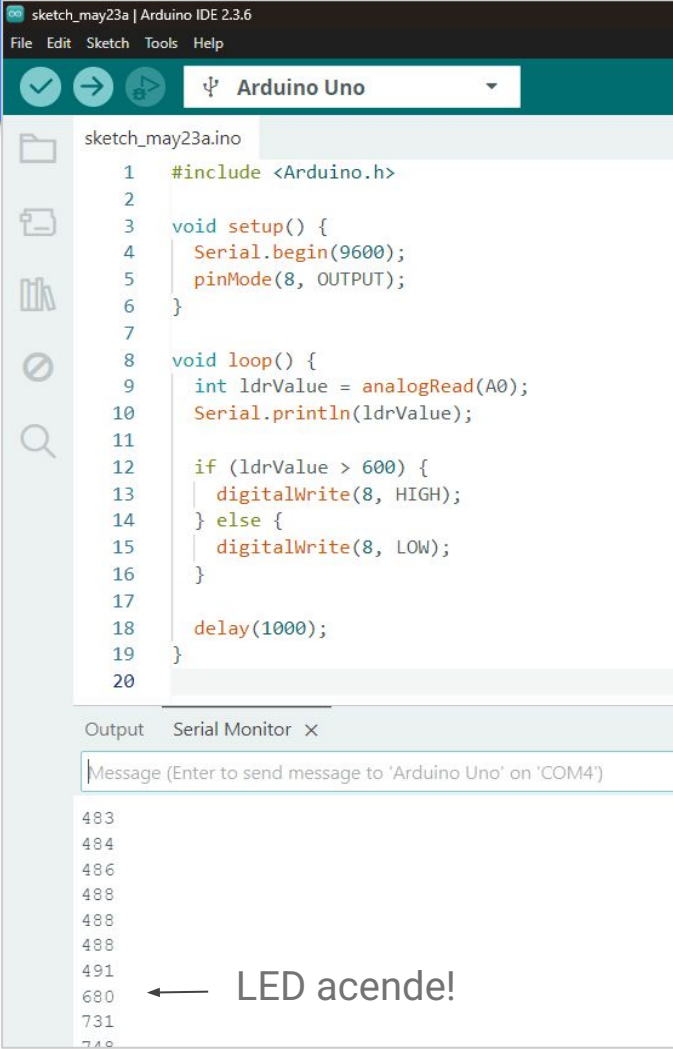
- Preparo de código para o Arduino ler o resultado do sensor
- Compilação do código e verificação no monitor serial da IDE do Arduino

```
void setup() {  
    Serial.begin(9600);  
    pinMode(8, OUTPUT);  
    pinMode(A0, INPUT);  
}  
  
void loop() {  
    int ldrValue = analogRead(A0);  
    Serial.println(ldrValue);  
  
    if (ldrValue > 600) {  
        digitalWrite(8, HIGH);  
    } else {  
        digitalWrite(8, LOW);  
    }  
  
    delay(1000);  
}
```

Prática final - IDE Arduino

PARTE 2

- Preparo de código para o Arduino ler o resultado do sensor
- Compilação do código e verificação no monitor serial da IDE do Arduino



The screenshot shows the Arduino IDE 2.3.6 interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar contains icons for checking, running, and uploading code. The board is set to 'Arduino Uno'. The sketch file 'sketch_may23a.ino' is open, displaying the following code:

```
1  #include <Arduino.h>
2
3  void setup() {
4      Serial.begin(9600);
5      pinMode(8, OUTPUT);
6  }
7
8  void loop() {
9      int ldrValue = analogRead(A0);
10     Serial.println(ldrValue);
11
12     if (ldrValue > 600) {
13         digitalWrite(8, HIGH);
14     } else {
15         digitalWrite(8, LOW);
16     }
17
18     delay(1000);
19 }
20
```

Below the code editor, the 'Serial Monitor' tab is active. It shows a message input field with the placeholder text 'Message (Enter to send message to 'Arduino Uno' on 'COM4')'. The serial output area displays a list of line numbers (483, 484, 486, 488, 488, 488, 491, 680, 731, 748) and an arrow pointing to the line number 680 with the text 'LED acende!'.

Prática final - Rstudio - leitura e armazenamento

PARTE 3

- Abertura da conexão serial com o R
- Abertura da conexão do R com o banco de dados
- Armazenamento no Banco de dados

```
library(serial)

listPorts()

porta <- "COM5"
baudrate <- "9600"

conexao <- serial::serialConnection(
  port=porta,
  mode = paste0(baudrate,"n,8,1"))

open(conexao)

while(TRUE) {
  leitura_raw <- serial::read.serialConnection(conexao)
  leitura <- strsplit(leitura_raw, ",")[[1]]

  print(leitura)
}
Sys.sleep(10) # Aguarda 10 segundos entre as leituras
}

close(conexao)
```

Prática final - Rstudio - visualização

PARTE 4

- Criação do banco de dados

```
library(DBI)
```

```
# Conectando a um banco SQLite (arquivo local)
```

```
con <- dbConnect(RSQLite::SQLite(), "meubanco.sqlite")
```

```
dbExecute(con, "PRAGMA journal_mode = WAL;")
```

```
dbExecute(con,
```

```
"CREATE TABLE IF NOT EXISTS sensor_log (tempo INTEGER, leitura INTEGER)")
```

```
#dbExecute(con, "DROP TABLE sensor_log") Apaga os dados da tabela
```

```
dados <- dbGetQuery(con, "
```

```
  SELECT * FROM sensor_log
```

```
  ORDER BY tempo DESC
```

```
")
```

```
dbExecute(con, paste0(
```

```
  "INSERT INTO sensor_log (tempo, leitura) VALUES (",
```

```
  timestamp, ", ", valor_int, ")")
```

```
))
```

```
dbExecute(con, "COMMIT")
```

```
dados
```

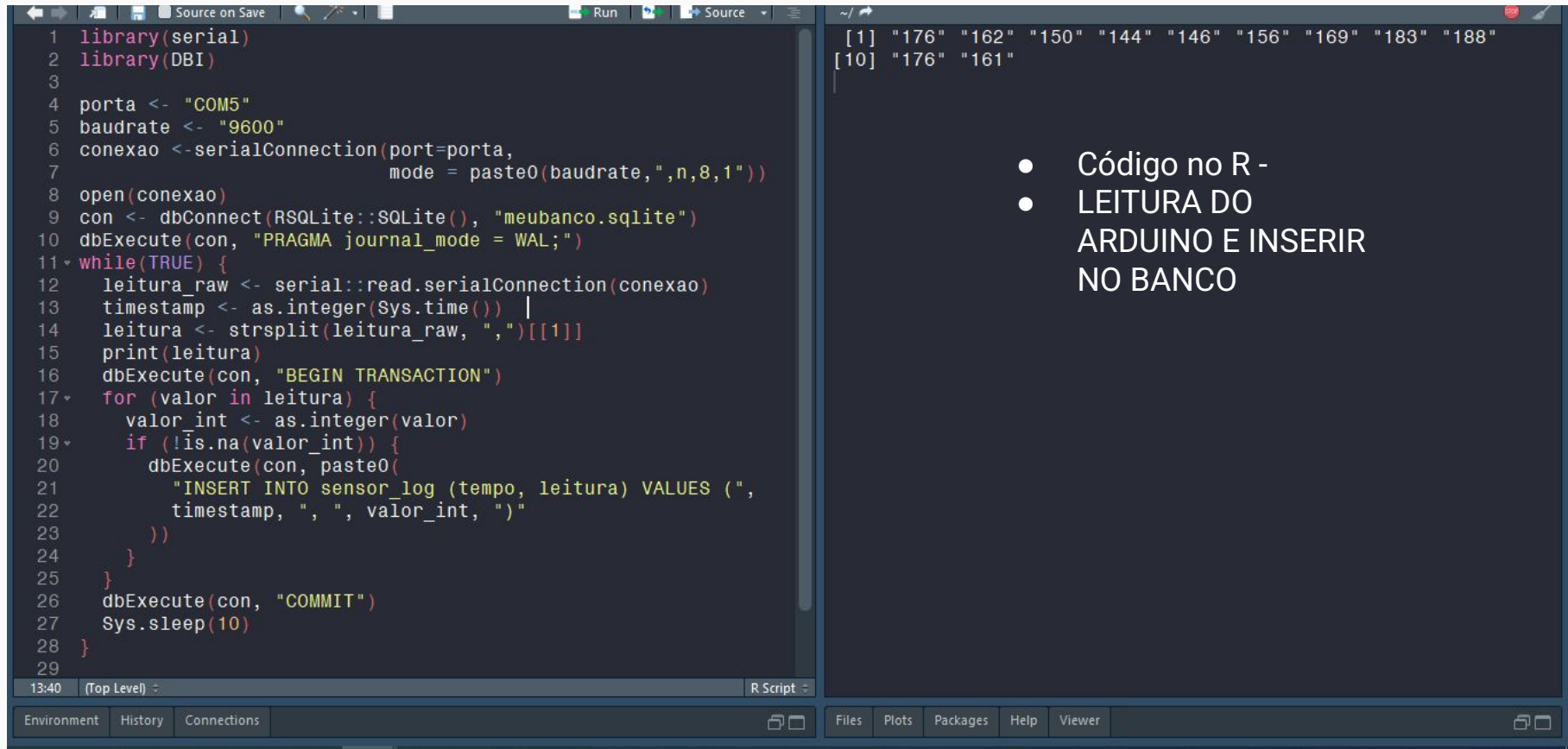

Prática final - Rstudio - leitura e armazenamento

PARTE 4

- Abertura da conexão serial com o R
- Abertura da conexão do R com o banco de dados
- Armazenamento no Banco de dados

```
dbExecute(con, "BEGIN TRANSACTION")
for (valor in leitura) {
  valor_int <- as.integer(valor)
  if (!is.na(valor_int)) {
    dbExecute(con, paste0(
      "INSERT INTO sensor_log (tempo, leitura) VALUES (",
      timestamp, ", ", valor_int, ")"
    ))
  }
}
dbExecute(con, "COMMIT")
```

Prática final - Rstudio - leitura e armazenamento



The screenshot displays the RStudio interface with a script editor on the left and a console on the right. The script editor contains R code that sets up a serial connection to a device (COM5) and connects to a SQLite database named 'meubanco.sqlite'. It then enters a loop to read data from the serial connection and insert it into a table named 'sensor_log' in the database. The console shows the output of the script, displaying two rows of data: [1] "176" "162" "150" "144" "146" "156" "169" "183" "188" and [10] "176" "161".

```
1 library(serial)
2 library(DBI)
3
4 porta <- "COM5"
5 baudrate <- "9600"
6 conexao <- serialConnection(port=porta,
7                             mode = paste0(baudrate,"n,8,1"))
8 open(conexao)
9 con <- dbConnect(RSQLite::SQLite(), "meubanco.sqlite")
10 dbExecute(con, "PRAGMA journal_mode = WAL;")
11 while(TRUE) {
12   leitura_raw <- serial::read.serialConnection(conexao)
13   timestamp <- as.integer(Sys.time()) |
14   leitura <- strsplit(leitura_raw, ",")[[1]]
15   print(leitura)
16   dbExecute(con, "BEGIN TRANSACTION")
17   for (valor in leitura) {
18     valor_int <- as.integer(valor)
19     if (!is.na(valor_int)) {
20       dbExecute(con, paste0(
21         "INSERT INTO sensor_log (tempo, leitura) VALUES (",
22         timestamp, ", ", valor_int, ")")
23     )
24   }
25 }
26 dbExecute(con, "COMMIT")
27 Sys.sleep(10)
28 }
29
```

[1] "176" "162" "150" "144" "146" "156" "169" "183" "188"
[10] "176" "161"

- Código no R -
- LEITURA DO ARDUINO E INSERIR NO BANCO

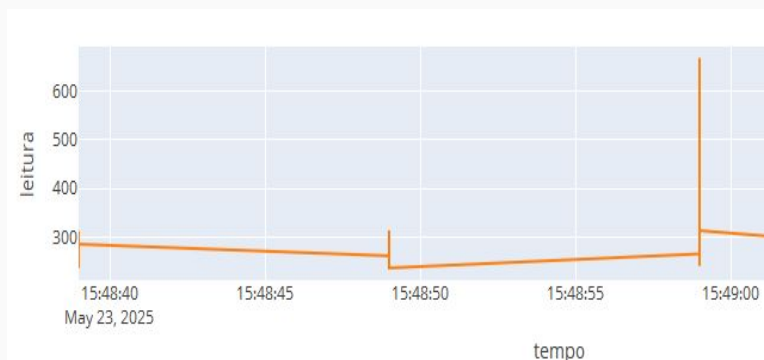
Prática final - Rstudio - visualização - exemplo

PARTE 5

- Leitura para visualização

Variável dados em ambiente como um dataframe para gerar visualizações.

<https://plotly.com/r/time-series/>



```
library(plotly)
library(dplyr)
library(DBI)
library(RSQLite)

con <- dbConnect(SQLite(), "meubanco.sqlite")
dbExecute(con, "PRAGMA journal_mode = WAL;")

df = dbGetQuery(con, "SELECT * FROM sensor_log")

df$tempo <- as.POSIXct(df$tempo)

fig <- plot_ly(df, type = 'scatter', mode = 'lines') %>%
  add_trace(x = ~tempo, y = ~leitura, name = 'GOOG') %>%
  layout(showlegend = F)
options(warn = -1)

fig <- fig %>%
  layout(xaxis = list(zerolinecolor = '#ffff',
    zerolinewidth = 2,
    gridcolor = 'ffff'),
    yaxis = list(zerolinecolor = '#ffff',
    zerolinewidth = 2,
    gridcolor = 'ffff'),
    plot_bgcolor = '#e5ecf6', width = 900)
```

fig

Extras e complicações

- RTC - Módulo para salvar tempos
- Gerência de Buffer
- Wal para garantir conexões múltiplas ao banco no SQLite
- Delay x Millis
- Shiny para visualizar dados



lyncolnsousa@id.uff.br

marcsonazevedo@gmail.com

Programas utilizados

- <https://plotly.com/r/time-series/>
- <https://www.arduino.cc/en/software/>
- <https://www.tinkercad.com/>
- <https://www.sqlite.org/wal.html>
- <https://cran.r-project.org/>
- <https://cran.r-project.org/web/packages/serial/index.html>
- Github do projeto: [GitHub - serial_import_ser_2025: serial import from text sent from arduino to a DB with R language](#)