FIAP GRADUAÇÃO

FIAP

SISTEMAS DE INFORMAÇÃO

Cognitive and Semantics Computing & IoT

PROF. ANTONIO SELVATICI

Versão 8 – Setembro/2022

3. DISPOSITIVOS DE IoT

SENSORES

- Sensores são dispositivos ou componentes eletrônicos capazes de detectarem eventos ou medir grandezas físicas
  - Sensor de temperatura
  - Sensor de colisão
  - Sensor de distância
- São o equivalente aos órgãos de sentidos nos seres vivos, como visão e audição

SENSORES DIGITAIS

- São aqueles que apresentam sua saída na forma digital, ou seja através de um ou mais bits
  - No Arduino e em outras placas controladoras, esses bits correspondem a uma ou mais portas ou pinos recebendo um nível de tensão alto ou baixo
  - No caso do Arduino o nível alto corresponde a ler algo próximo de 5V, enquanto o nível baixo equivale a ler algo próximo a 0V

SENSORES DIGITAIS

- Transmitem sua informação através de sequências de bits, podendo ser:
  - Paralelo: um conjunto de bits é informado através da leitura de várias portas ao mesmo tempo. Por exemplo: porta paralela de impressão de computadores antigos
  - Serial: a sequências de bits são recebidas em sequência através pela mesma portado microcontrolador
- A forma mais simples de sensores digitais são aqueles que informam apenas o valor de 1 bit, correspondendo à existência ou não de um evento
  - Por exemplo: apertar um botão
  - Leitura pelo comando `digitalRead(porta)`

SENSORES DE CONTATO

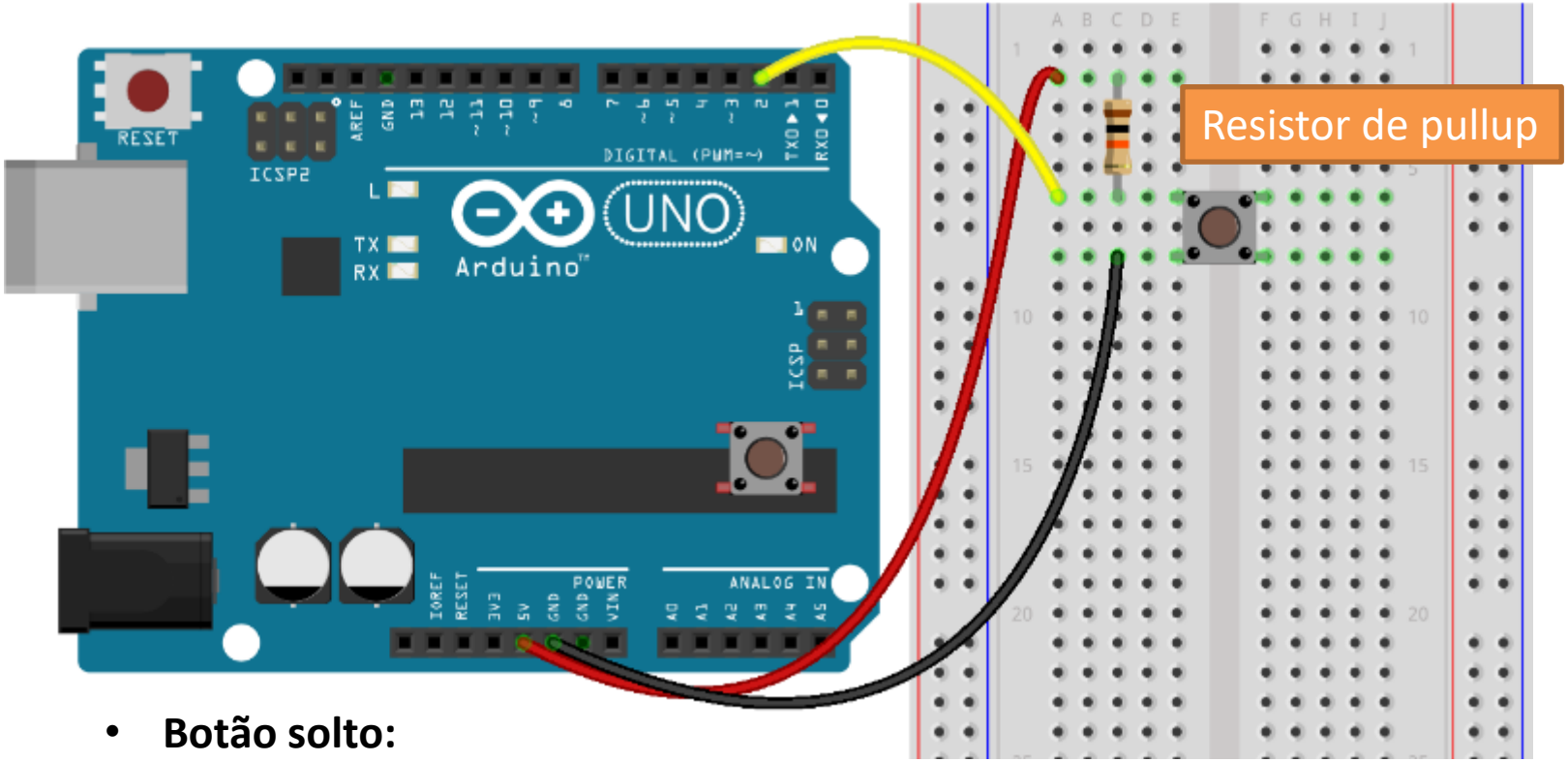
- Funcionam da mesma forma de um botão: quando há o contato com algum objeto, fecha-se o contato elétrico e o nível lógico de leitura é modificado

Botão

Colisão

Contato magnético

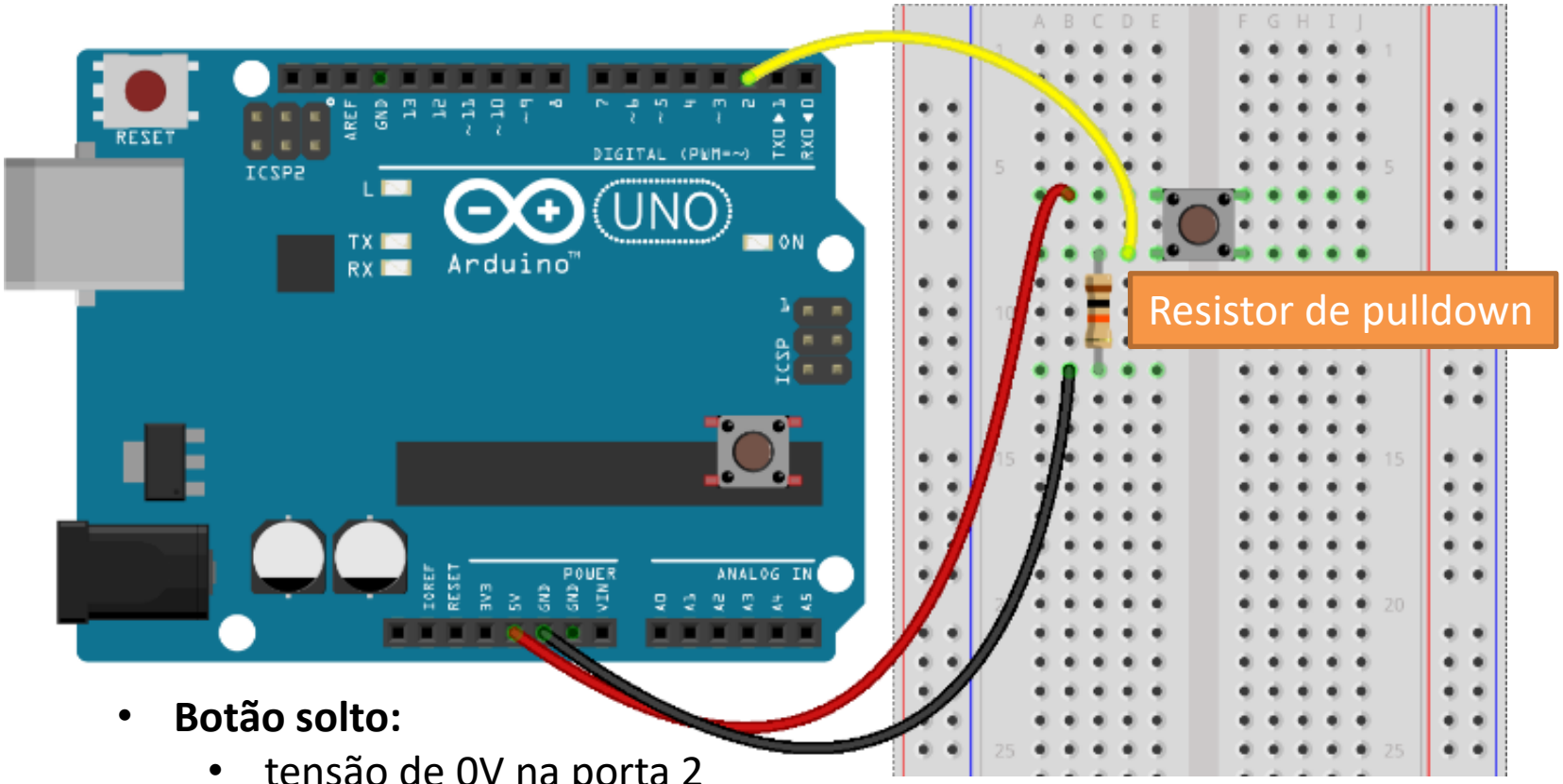
CONFIGURAÇÃO PULLUP DO BOTÃO



- **Botão solto:**
  - tensão de 5V na porta 2
  - `digitalRead(2) -> 1`

- **Botão pressionado:**
  - tensão de 0V na porta 2
  - `digitalRead(2) -> 0`

CONFIGURAÇÃO PULLDOWN DO BOTÃO



- **Botão solto:**
  - tensão de 0V na porta 2
  - `digitalRead(2) -> 0`

- **Botão pressionado:**
  - tensão de 5V na porta 2
  - `digitalRead(2) -> 1`

PORTAS DE ENERGIA (POWER) DO ARDUINO UNO

- 3V3 – fornece uma tensão de 3,3V com relação ao GND
- 5V – fornece uma tensão de 5V com relação ao GND
- GND – conectada ao potencial 0V do Arduino
- VIN – fornece a tensão de entrada do Arduino, principalmente se estiver conectado a uma fonte externa de tensão

LEITURA DO BOTÃO

- Construa uma das duas configurações (Pullup ou Pulldown)
- Execute o programa a seguir e acompanhe pelo monitor serial:
  - Experimente apertar o botão e ver a saída

```
int botao = 2; // entrada digital na porta 2
int val = 0;
void setup() {
  Serial.begin(9600);
  pinMode(botao, INPUT); // porta 2 vira entrada
}
void loop(){
  val = digitalRead(botao); // lê a porta do botão
  Serial.println(val);
  delay(2000);
}
```

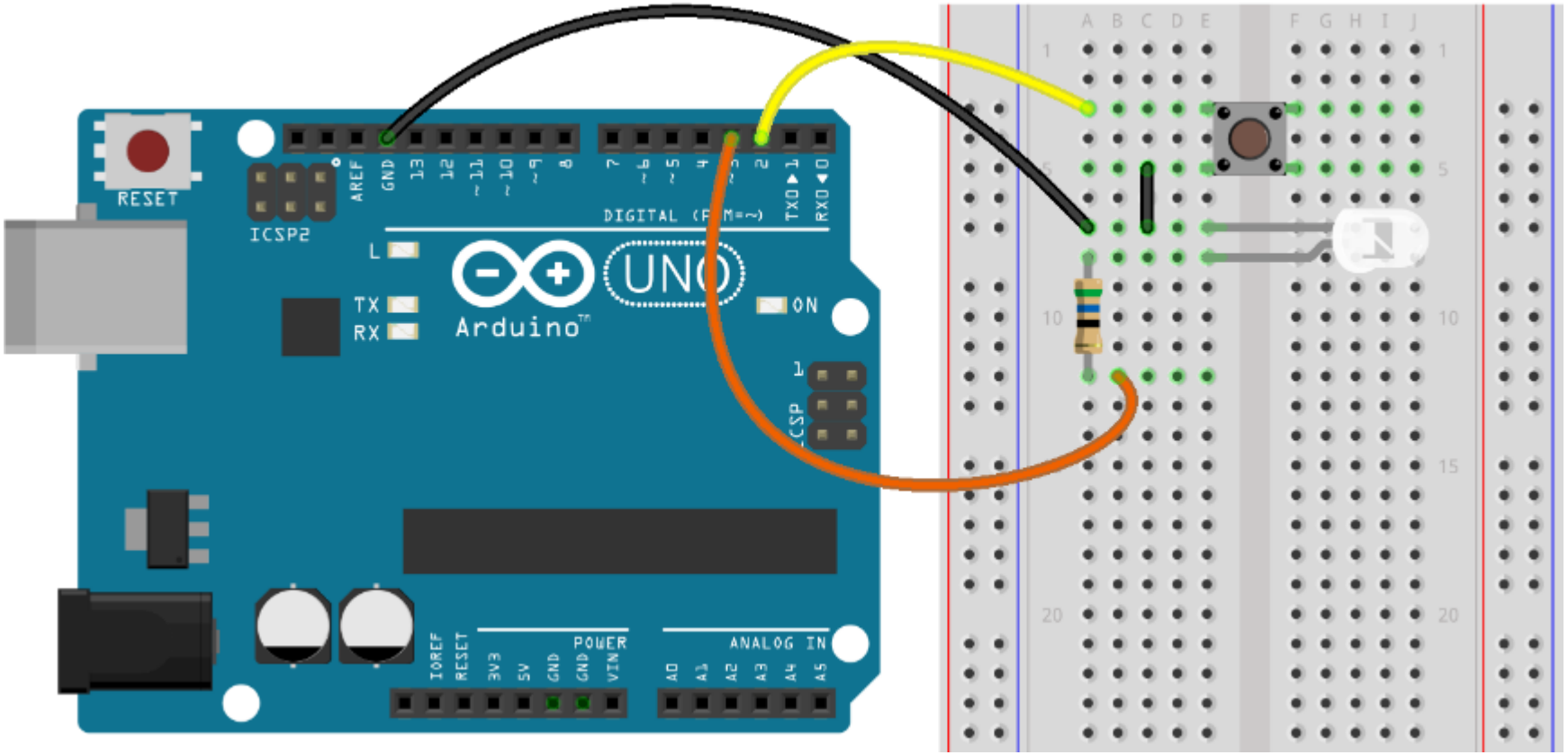
CONFIGURAÇÃO PULLUP DO ARDUINO

- Sem o resistor de *pullup* ou de *pulldown*, a tensão elétrica no pino de leitura ficaria "flutuando" quando o botão estivesse solto
  - É equivalente a não ter nada conectado à porta, então a tensão ali fica indefinida
- Em vez de termos que montar o resistor sempre que formos usar o botão, podemos usar o modo de entrada INPUT_PULLUP do Arduino
  - O Arduino conecta o resistor internamente
  - Ao pressionar o botão, deve conectar o GND à porta de leitura

EXPERIMENTO 4: ACENDER O LED AO PRESSIONAR O BOTÃO

- Materiais:
  - 1 Arduino Uno com cabo usb;
  - 1 Protoboard;
  - 1 LED alto brilho;
  - 1 Resistor de 56 ohm (ver cores na tabela)
  - 1 push button
  - cabinhos

EXPERIMENTO 4: CIRCUITO



EXPERIMENTO 4: PROGRAMAÇÃO

```
int botao = 2; // entrada digital na porta 2
int led = 3; // LED na porta 3
int val = 0;
void setup() {
  pinMode(botao, INPUT_PULLUP); // porta 2 vira entrada
  pinMode(led, OUTPUT); // porta 3 vira saída
}
void loop(){
  val = digitalRead(botao); // lê a porta do botão
  // o botão possui lógica inversa
  digitalWrite(led, !val);
  delay(50);
}
```

INTERRUPÇÕES NO ARDUINO

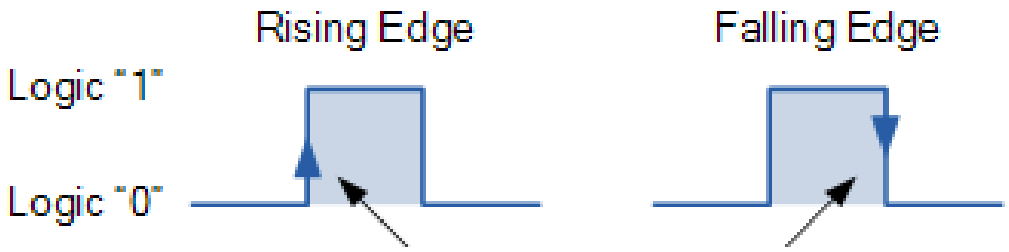
Realizando operações críticas em tempo real

- Imagine o seguinte cenário:
  - Uma fábrica precisa monitorar a pressão de 100 tanques de fabricação de amônia
  - Para ler a pressão de cada tanque o controlador demora 250ms
  - Se algum tanque estiver com sobre-pressão, há um tempo hábil de um segundo para abrir a válvula de escape
- Se o controlador tivesse que ler a pressão de cada tanque para eventualmente tomar a decisão de abrir a válvula de algum deles, poderíamos ter uma explosão na fábrica, já que ele demoraria 25 segundos para ler todos os tanques
- Para essas situações, microcontroladores e microprocessadores possibilitam as chamadas **interrupções**, que são eventos que servem de gatilhos para ações especiais a serem executadas
- Assim, um sensor especial de sobre-pressão poderia estar ligado a uma porta do controlador, que lançaria uma interrupção quando um tanque estivesse nessa situação

INTERRUPÇÕES NO ARDUINO

Realizando operações críticas em tempo real

- A ação a ser executada numa interrupção é executada na forma de uma ISR (*Interrupt Service Routine*), uma função com certas limitações que é invocada assim que ocorre a interrupção, interrompendo o processamento sendo executado no momento
- O gatilho pode ser gerado internamente, através de um temporizador chegando a zero, por exemplo, ou...
- Pode ser um gatilho externo, como o valor de uma porta de entrada sendo escrito
- Mais comumente o gatilho externo é de dois tipos
  - *Rising Edge*: interrupção com valor indo de LOW para HIGH
  - *Falling Edge*: interrupção com valor indo de HIGH para LOW



EXEMPLO DE INTERRUPÇÃO - ARDUINO

```
int led = 3; //Porta do LED
//Porta da interrupção:
//O Arduino Uno só aceita as portas 2 e 3
int interruptPort = 2;

//Variáveis modificadas por interrupções devem ser volatile
volatile int state = LOW;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(interruptPort, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPort), toggle, CHANGE);
}
void loop() {
   // Qualquer processamento mais longo...
}
void toggle() {
  state = !state;
  digitalWrite(led, state);
}
```

INTERRUPÇÕES NO ARDUINO

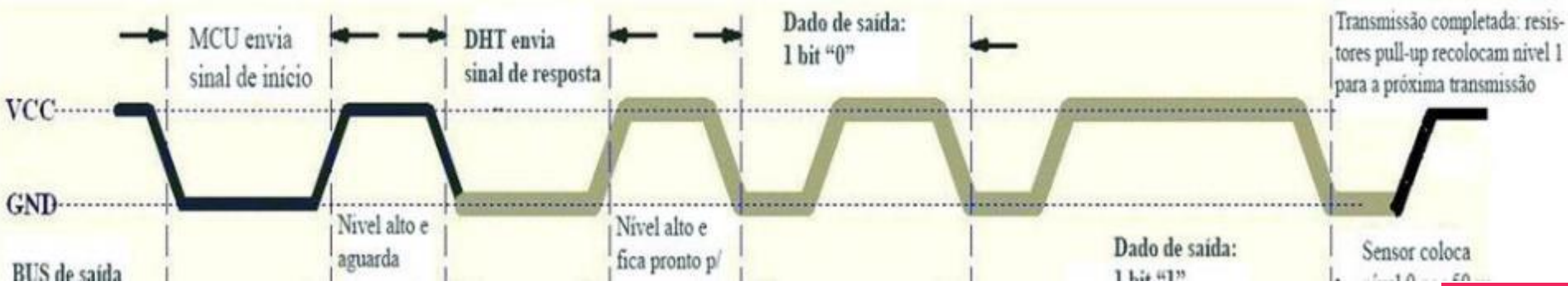
- Durante a execução de uma ISR, as interrupções estão desabilitadas
  - Por isso devem ser de rápida execução
  - Funções de E/S podem não funcionar durante sua execução
- Para ativar uma interrupção, invocamos a função:
  - `attachInterrupt(NUMERO, ISR, MODO);`
  - `NUMERO`: identificação da interrupção, a depender da porta usada (ver em http://arduino.cc/en/Reference/attachInterrupt)
    - Para traduzir a porta usada para o número da interrupção, usar a função `digitalPinToInterrupt(pin)`
  - `ISR`: função a ser invocada durante a interrupção
  - `MODO`: o tipo do gatilho, podendo ser
    - LOW: dispara a interrupção quando a porta estiver em LOW
    - CHANGE: dispara quando há mudança de valor
    - RISING: dispara quando ocorre uma Rising Edge na porta
    - FALLING: dispara quando ocorre uma Falling Edge na porta

EXPERIMENTO 5:

- Use a mesma lista de materiais e o mesmo circuito do experimento 4
- Usando como base a programação do slide 18, modifique a definição da interrupção e, para cada tipo de interrupção usada, teste-a no Arduino
- Os tipos de interrupção são:
  - LOW
  - CHANGE
  - RISING
  - FALLING

SENSORES DIGITAIS SERIAIS

- Os sensores digitais mais complexos necessitam de mais de um bit para comunicar sua informação a cada instante. Assim podem enviá-la através de uma sequência de bits, que devem ser lidos pelo Arduino segundo um protocolo específico.



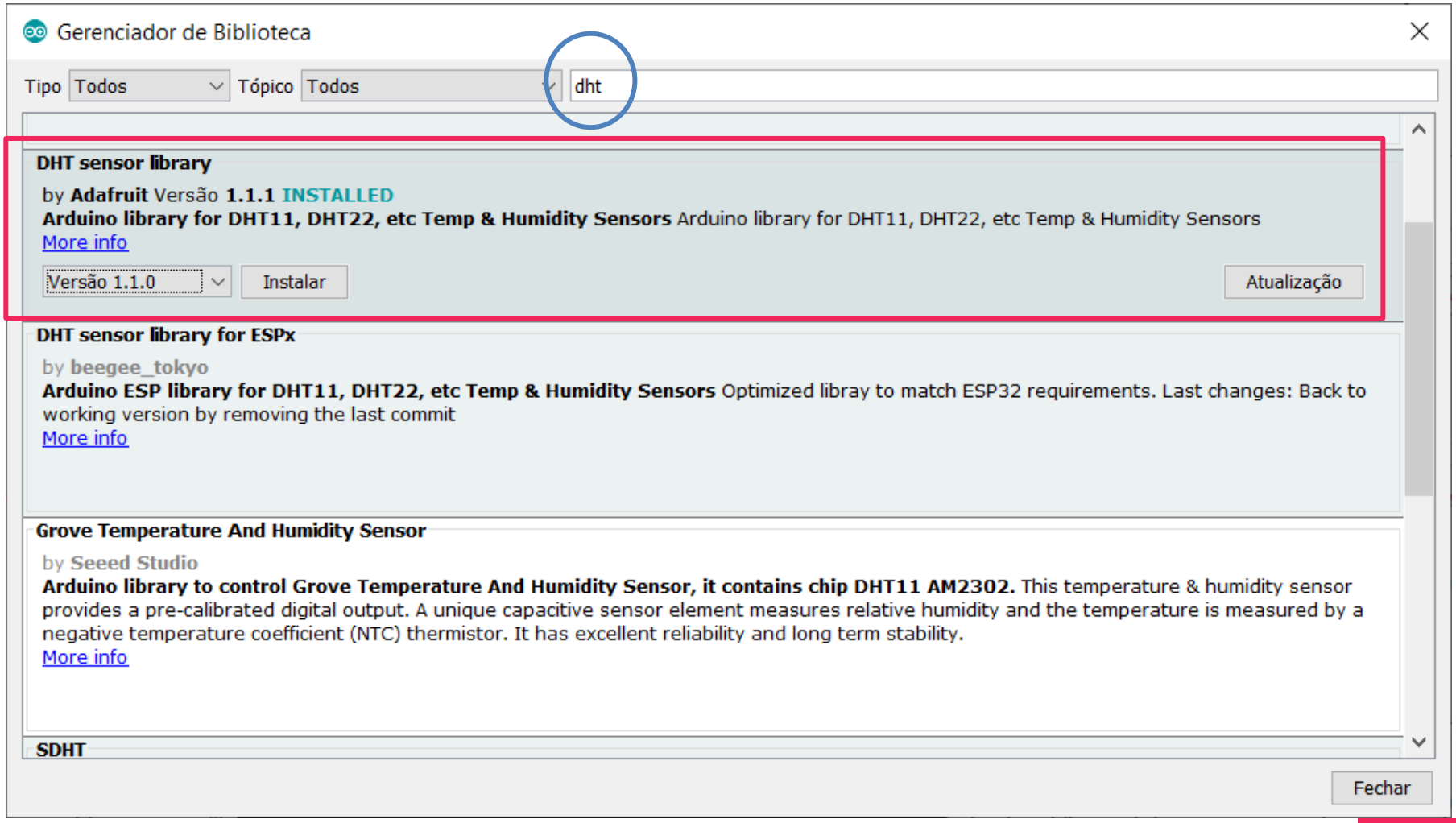
UTILIZANDO BIBLIOTECAS

- Para prover a implementação dos protocolos específicos de cada sensor ou atuador digital, bem como outras funcionalidades para além da API padrão, empregamos bibliotecas do Arduino.
- Há bibliotecas que já vêm instaladas no Arduino IDE, e basta, ser invocadas através da diretiva `#include`, mas outras precisam ser instaladas antes.
- Há duas formas de instalar novas bibliotecas:
  - Através do gerenciador de bibliotecas:
  - `Sketch → Import Libraries → Library Manager...`
  - Através da importação de arquivos compactados
  - `Sketch → Import Libraries → Add Library...`
    - Obs: é o modo como deve ser feito no dia da prova caso ainda não esteja instalada
- As bibliotecas possuem definição de classes e podem conter exemplos de utilização

O SENSOR DHT-11

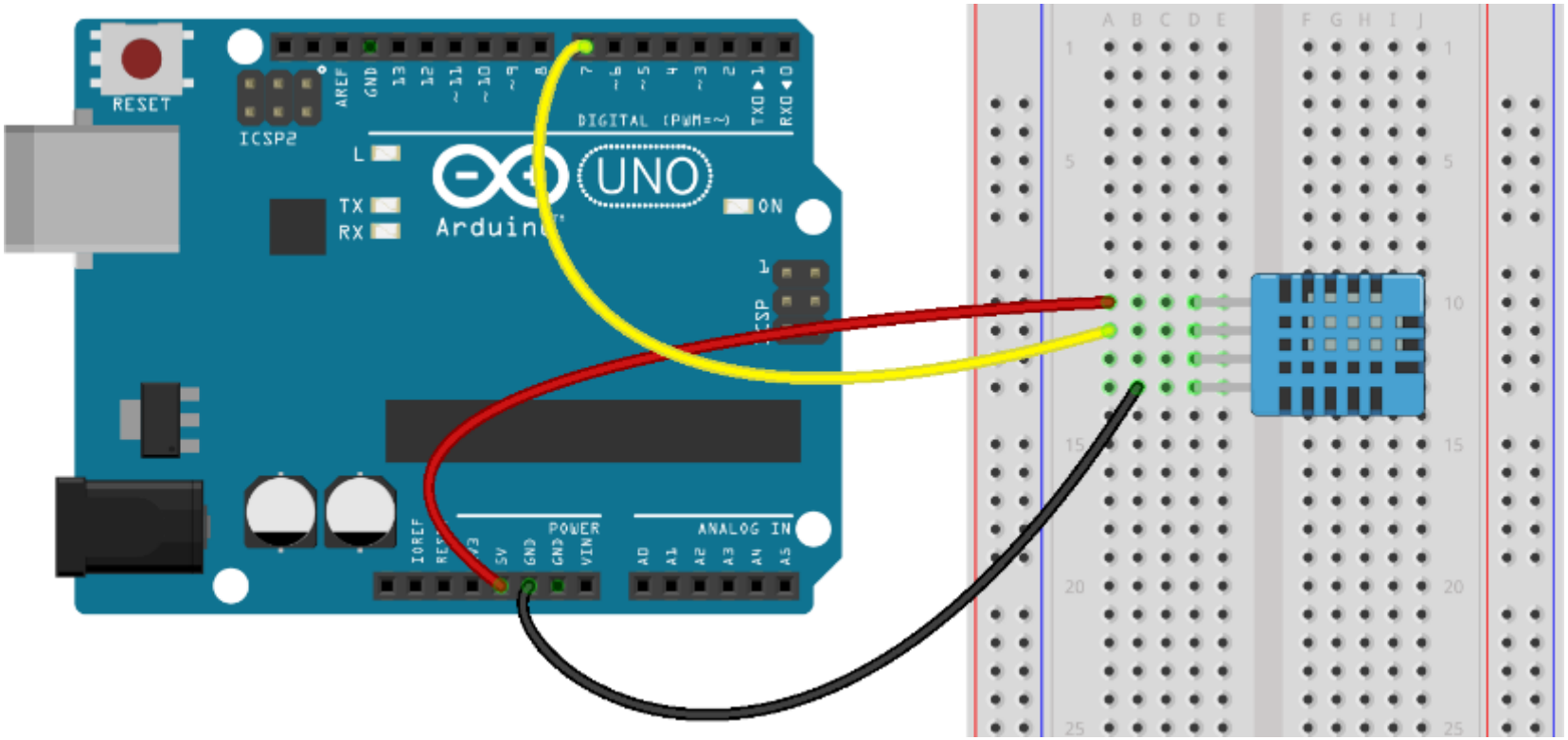
- O sensor DHT11 fornece tanto temperatura quanto umidade do ar instantaneamente e de forma muito fácil.
- A comunicação com o Arduino é feita através de um protocolo serial próprio
- Verificar no datasheet:
  - http://robocraft.ru/files/datasheet/DHT11.pdf
- A biblioteca DHT cuida de todas as funções necessárias, restando para nós apenas acessar aos dados.
- Instalando a biblioteca
  - O arquivo DHT.zip da biblioteca está fornecida no portal de apostilas
  - No gerenciador de bibliotecas, instalar a versão 1.1.0 da biblioteca da Adafruit
  - Versões mais novas, a partir da 1.3.0, exigem o uso da API **Adafruit Unified Sensors**

BIBLIOTECA A SER INSTALADA



EXPERIMENTO 6: LENDO O SENSOR DHT 11

Lendo os dados através da porta 7



EXPERIMENTO 6: PROGRAMAÇÃO

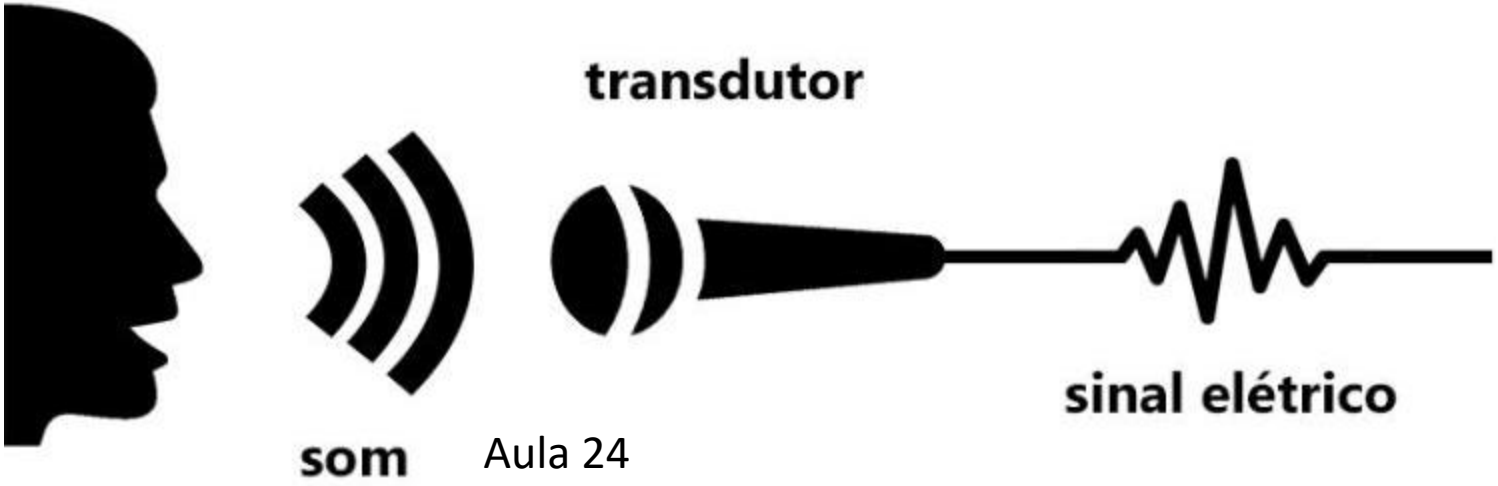
```
#include "DHT.h"
const int DHTPIN = 7; // pino que estamos conectado
const int DHTTYPE = DHT11; // DHT11 ou DHT22
DHT dht(DHTPIN, DHTTYPE); //Instanciação do objeto do sensor
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  // A leitura da temperatura e umidade pode levar até 250ms!
  float h = dht.readHumidity();//Valor da umidade
  float t = dht.readTemperature(); //Valor da temperatura
  if (isnan(t) || isnan(h)) {
    Serial.println("Erro ao ler do DHT");
  } else {
    Serial.print("Umidade: ");
    Serial.print(h); Serial.print(" %\t");
    Serial.print("Temperatura: ");
    Serial.print(t); Serial.println(" °C");
  }
}
```

API DHT

- A diretiva `#include "arquivo"` importa o cabeçalho da biblioteca com a prototipação das funções
  - Necessário quando não faz parte da API padrão do Arduino
  - A biblioteca tem que estar instalada na IDE
- O modificador `const` indica que a variável não poderá ter seu valor modificado
- A classe `DHT` é definida no cabeçalho `DHT.h`
- O construtor `DHT(int porta, int tipo)` é invocado para instanciar o objeto `dht` no código
  - Esse tipo de objeto é chamado de automático, pois é destruído assim que o seu escopo é finalizado
  - Objetos dinâmicos no C++ têm a sintaxe semelhante à do Java, com o uso da palavra chave `new`, mas devem ser explicitamente destruídos com `delete`
- Os métodos `readTemperature()` e `readHumidity()` fazem a leitura da temperatura e da umidade, respectivamente.

SENSORES ANALÓGICOS

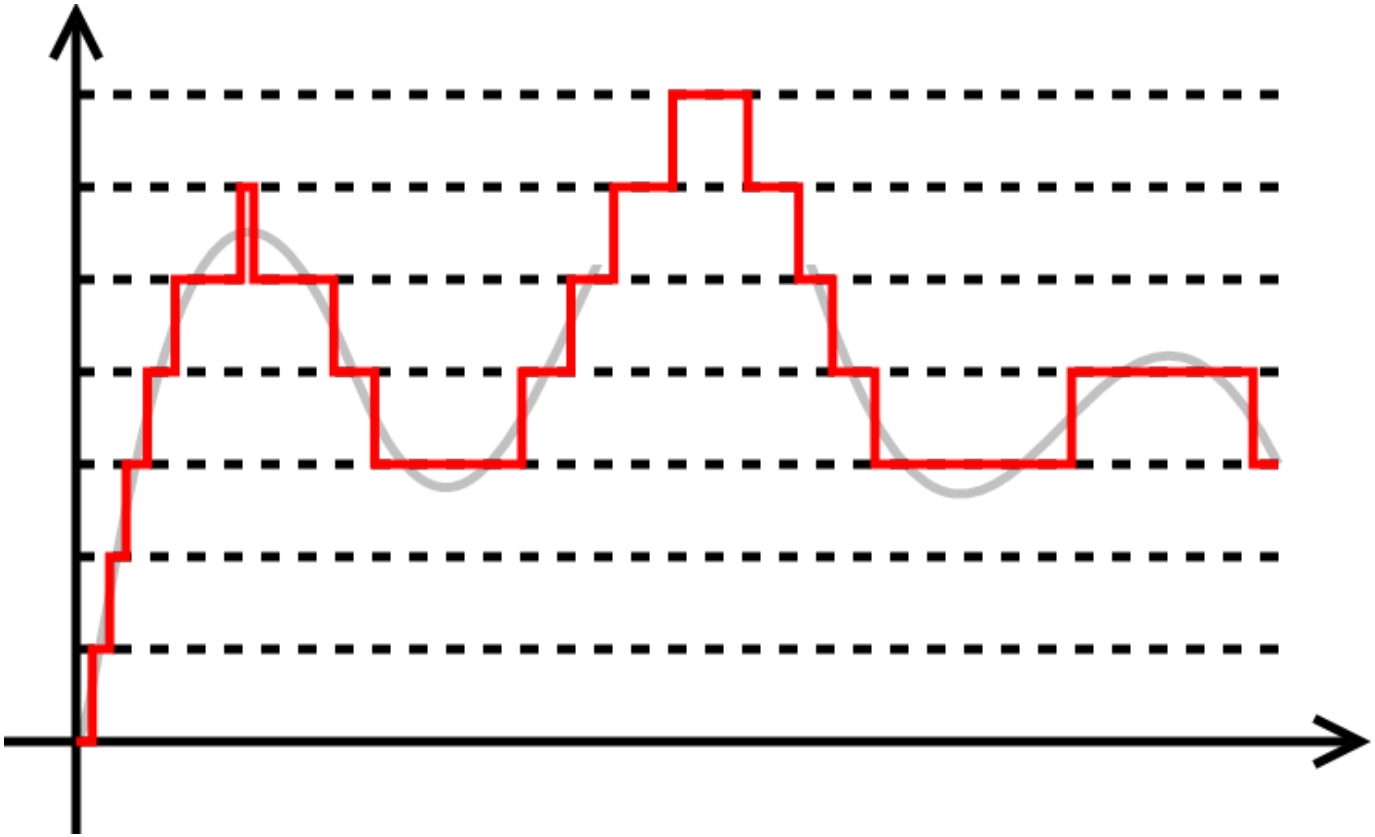
- Sensores analógicos são aqueles que apresentam a sua informação na forma analógica, ou seja, através de um nível de tensão que varia de forma análoga ao fenômeno que está medindo
- Este tipo de sensor também é conhecido como **transdutor**



ADC – ANALOG TO DIGITAL CONVERTER

- O Arduino precisa transformar a informação analógica em digital para poder aproveitá-la
- Essa transformação é realizada por um circuito chamado ADC, que está presente em praticamente todos os micro-controladores
- Assim, a cada leitura do sensor, o ADC transforma o valor da tensão elétrica em um número inteiro, representado por uma certa quantidade de bits

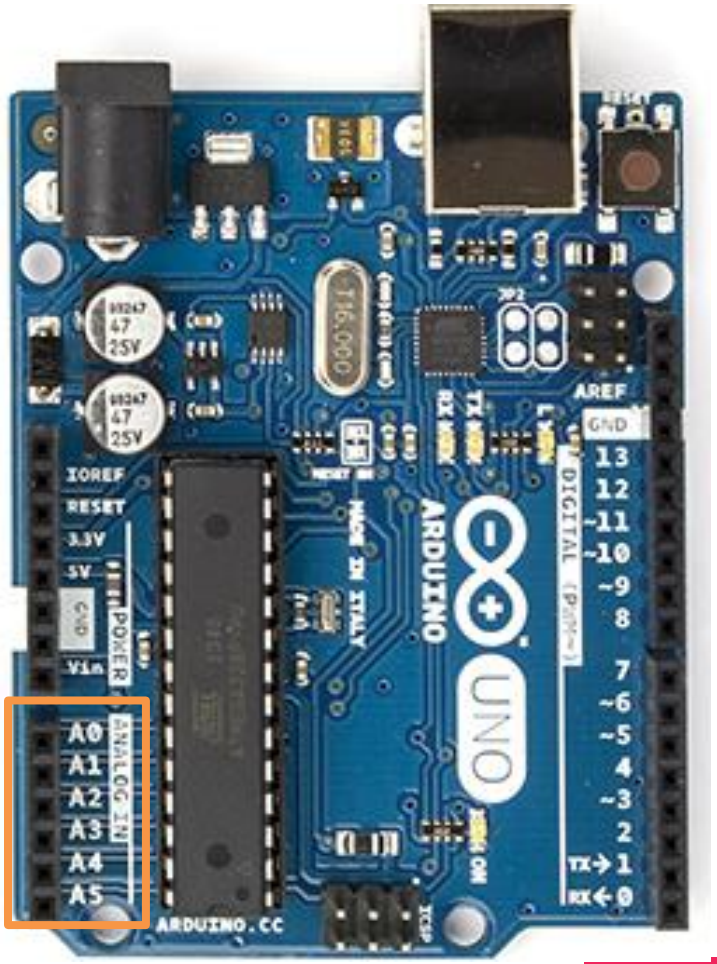
ADC - EXEMPLO



- Quantos níveis são usados para representar o sinal?
- De quantos bits precisamos para representar esses níveis?

ADCs DO ARDUINO – PORTAS ANALÓGICAS

- Os ADCs do Arduino estão nas chamadas portas digitais
  - Arduino Uno: A0 a A5
- Medem a tensão entre a porta de entrada e o GND
- Fazem a leitura de valores entre 0 e 1023
- Cada leitura é realizada através da função: `analogRead(int porta)`

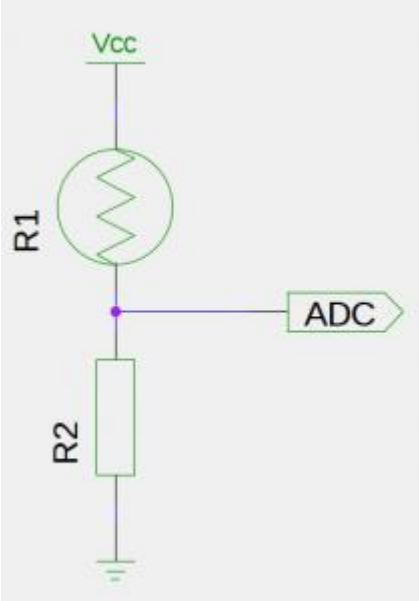


Sensor de luminosidade (LDR)

- Nosso primeiro sensor analógico será o LDR (Light Dependent Resistor), um sensor de luminosidade ambiente
- Ele possui dois contatos elétricos planos separados por uma trilha de sulfito de cádmio.
  - Quando há incidência de luz, essa substância permite a condução de corrente, porém apresentando uma certa resistência elétrica.
  - Quanto maior essa incidência de luz, menor será essa resistência, e portanto, caso haja uma tensão aplicada, maior será a corrente elétrica ali passando.
- O LDR funciona então como um resistor cuja resistência varia de acordo com a incidência de luz.
  - Para medir a luz, precisamos medir essa resistência
  - Para tanto, aplicamos uma diferença de potencial e medimos a corrente que passa

Medindo a luminosidade com o LDR

- Usamos um resistor (R2) ligado em série com o LDR (R1), aplicando uma tensão constante nas extremidades livres
- A resistência equivalente dos dois componentes é a soma das resistências
de ambos, e a corrente que passa nos dois componentes é a mesma
- Como a resistência do resistor é constante, a tensão elétrica entre as suas extremidades será proporcional à corrente, que por sua vez é proporcional à luminosidade ambiente

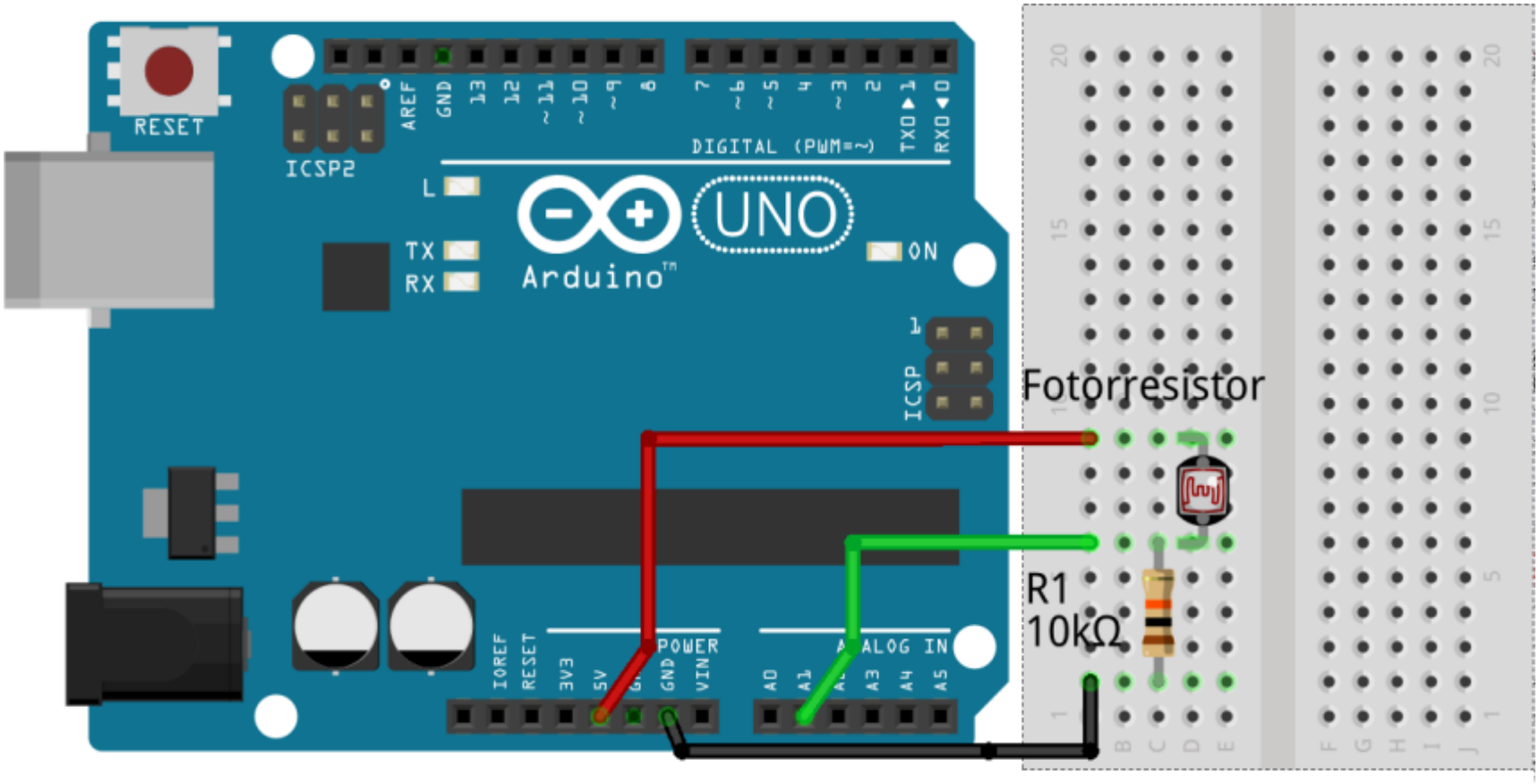


EXPERIMENTO 7: LENDO SENSOR DE LUMINOSIDADE

- Materiais:
  - 1 Arduino Uno
  - 1 Protoboard
  - 1 LDR (fotorresistor)
  - 1 Resistor de 10 kilo-ohm
  - Cabinhos tipo jumper

EXPERIMENTO 7: CIRCUITO

Lendo um sensor de luminosidade



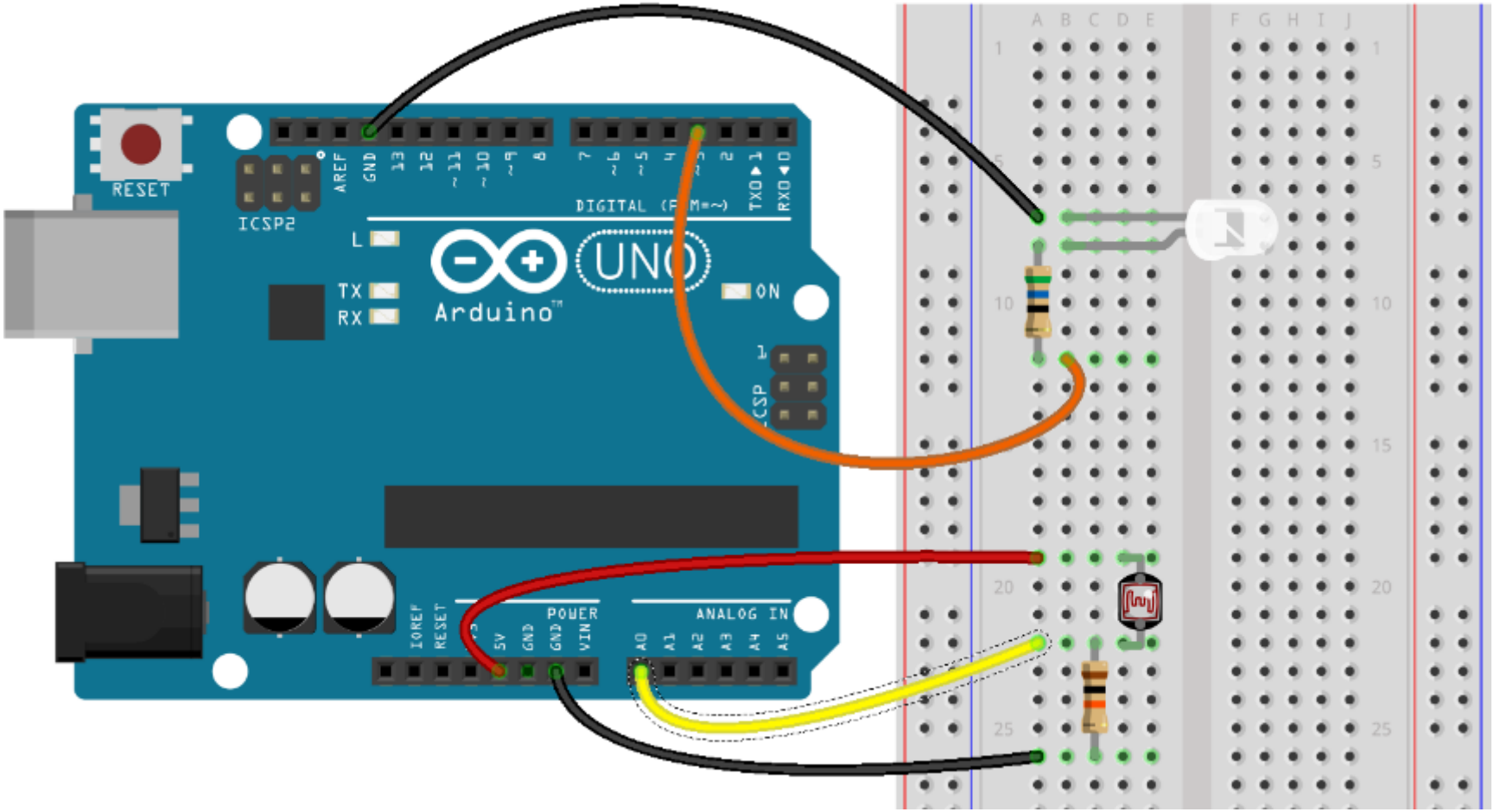
EXPERIMENTO 7: PROGRAMAÇÃO

```
//Pino analógico em que o sensor está conectado
const int sensor = A1;

void setup(){
  Serial.begin(9600);
}

void loop(){
  //Lendo o valor do sensor.
  int valorSensor = analogRead(sensor);

  //Exibindo o valor do sensor no serial monitor.
  Serial.println(valorSensor);

  delay(500);
}
```

EXPERIMENTO 8: CONTROLE DA LUMINOSIDADE PELA ILUMINAÇÃO

- O objetivo é fazer com que a luz do LED seja controlada pela luminosidade ambiente. Assim, quanto mais claro o ambiente, menor a luz do LED
- Materiais:
  - 1 Arduino Uno
  - 1 Protoboard
  - 1 LDR (fotorresistor)
  - 1 LED de alto brilho
  - 1 Resistor de 10 kilo-ohm
  - 1 Resistor de 56 ohm
  - Cabinhos tipo jumper

EXPERIMENTO 8: CIRCUITO



EXPERIMENTO 8: PROGRAMAÇÃO

- Neste experimento, o grupo deverá construir o próprio programa através dos exemplos anteriores
- A cada loop, primeiro o sensor de luminosidade deve ser lido, então o valor do Duty Cycle PWM do brilho deve ser calculado e por fim este deve ser enviado ao LED
- Lembre-se de que a luminosidade é um valor entre 0 e 1023, enquanto o Duty Cycle é expresso com valores entre 0 e 255. Qual conta deverá ser realizada?
  - Para luminosidade = 0, o controle PWM deverá ser 255
  - Para luminosidade = 1023, o controle PWM deverá ser 0
- Para ajudar no cálculo, pesquise a função do Arduino
  - `map(valor, x_min, x_max, y_min, y_max)`
  - No nosso exemplo, `x_min=0`, `x_max=1023`, `y_min=255` e `y_max=0`
  - https://www.arduino.cc/reference/pt/language/functions/math/map/

REFERÊNCIAS

1. Arduino Team. **Arduino Reference: Digital Read.** url: https://www.arduino.cc/reference/pt/language/functions/digital-io/digitalread/
2. Adafruit Team. **DHT Sensor library.** url: https://github.com/adafruit/DHT-sensor-library
3. Arduino Team. **Arduino Reference: Map.** url: https://www.arduino.cc/reference/pt/language/functions/math/map/