FIAP GRADUAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Arquiteturas Disruptivas e Big Data

PROF. ANTONIO SELVATICI

Versão 3– Fevereiro/2017

SHORT BIO

É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (ITA), com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos. Desenvolveu projetos para Avibrás, IPT e Systax.

PROF. ANTONIO SELVATICI
profantonio.selvatici@fiap.com.br

INTERNET DAS COISAS

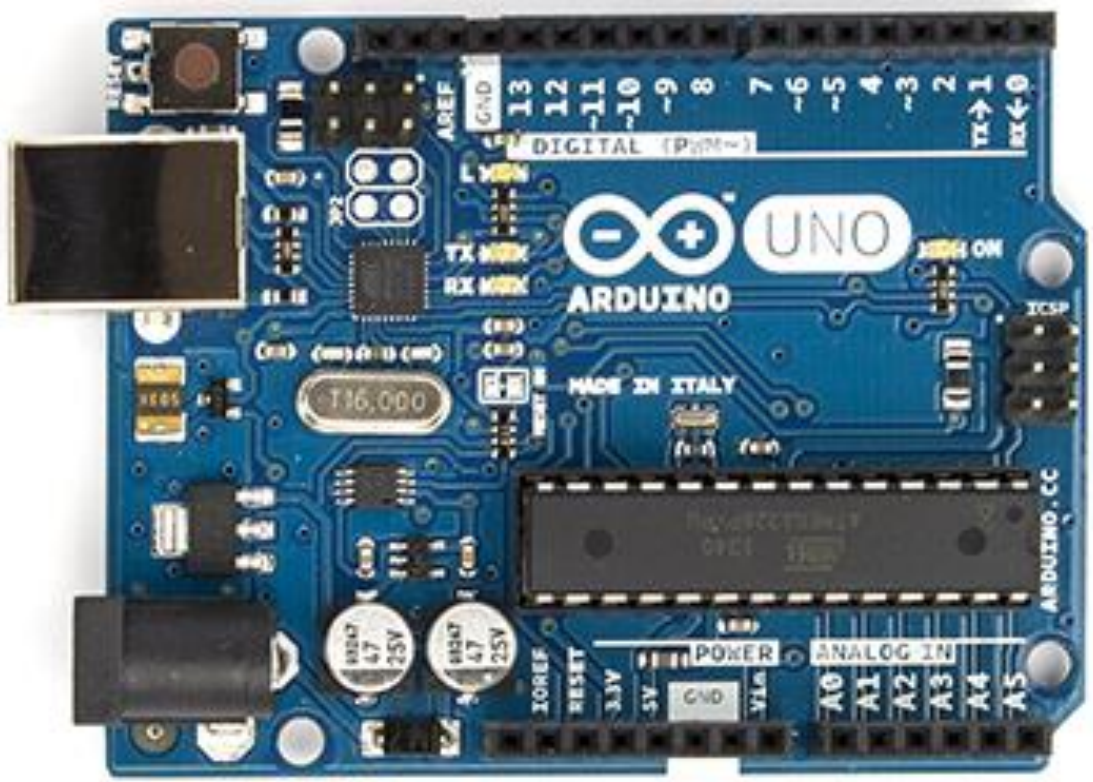
ARDUÍNO

Introdução

- Arduino é uma plataforma de hardware para a rápida execução de projetos eletrônicos, possuindo um microcontrolador Atmel AVR com suporte de entrada/saída embutido
  - É um projeto *open source*, tanto no que tange ao hardware quanto ao software (ou seja, pode ser copiado)
  - Utiliza componentes de baixo custo
  - Emprega uma IDE de programação simplificada baseada em *Wiring*, que simplifica o processo de criação de projetos de C++
- Origem: criado por professores da Ivrea Interaction Design Institute para facilitar e baratear a criação de projetos pelos alunos
- Pode ser usado como um computador independente, ou estar conectado via USB no modo FTDI, sendo mapeado em uma porta serial.

ARDUINO UNO

Arduino Uno



PRIMEIRO PROGRAMA

Hello World

- Formar grupos
- Conectar o Arduino ao computador pelo cabo USB
- Abrir o programa “Arduino IDE”
- Digitar o programa abaixo na janela que se abriu

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println("Hello World!");
  delay(3000);
}
```

- Verificar e enviar o programa
- Abrir a janela Tools → Serial Monitor

COMO FUNCIONA?

Hello World

- O Arduino possui uma tensão de operação de 5V, fornecida pela porta USB do computador
  - No caso de execução stand alone, deve ser fornecida alimentação de 7V a 12V na entrada de tensão DC (valores recomendados)
- Função `void setup();`
  - Chamada no início da execução do programa
  - Deve configurar os dispositivos a serem usados
- Função `void loop();`
  - Executada dentro do laço principal de execução
  - Executa enquanto a placa estiver ligada
- Classe Serial: representa a classe que se comunica com o computador hospedeiro através da porta USB/serial
- Função `delay(milissegundos)`: pausa a execução.

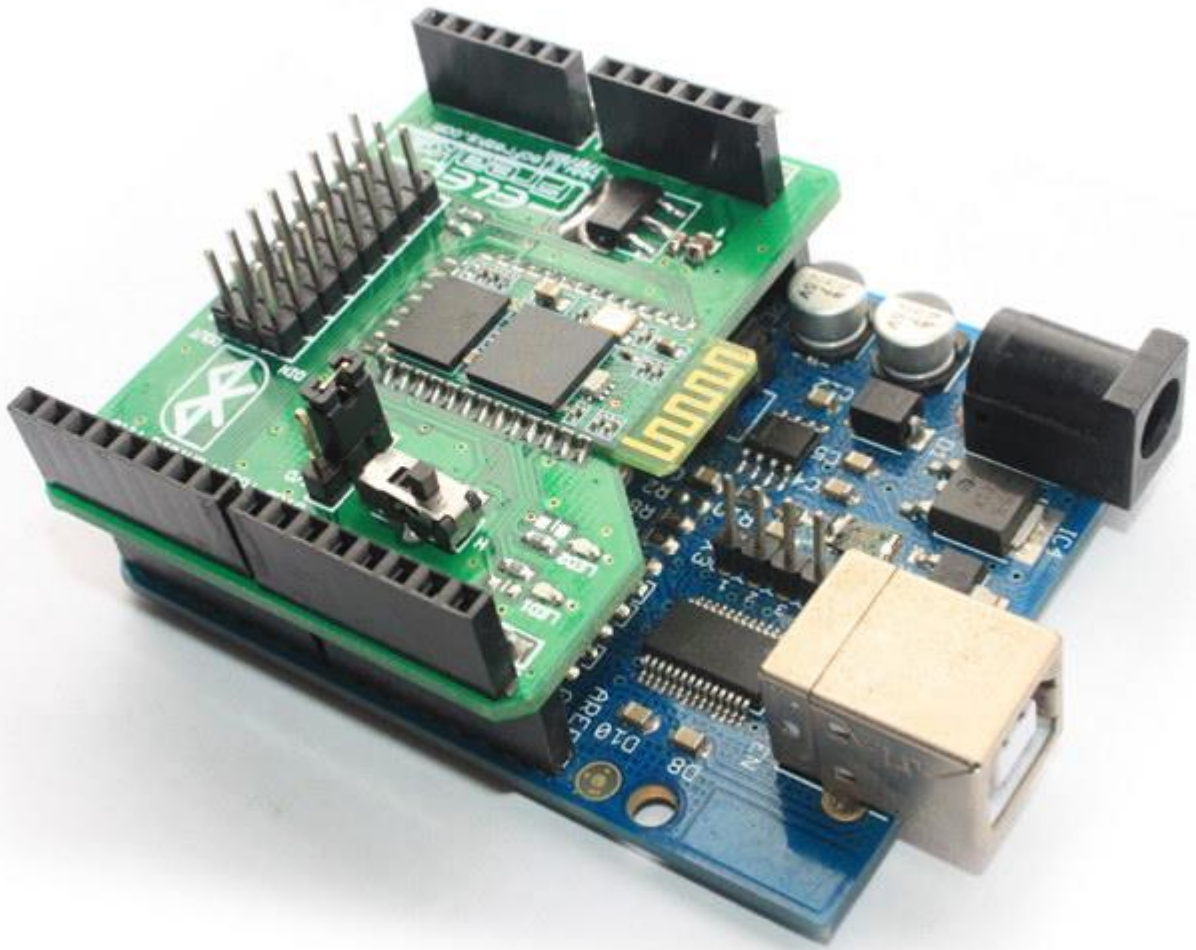
INTERAÇÃO COM DISPOSITIVOS

Sensores e atuadores

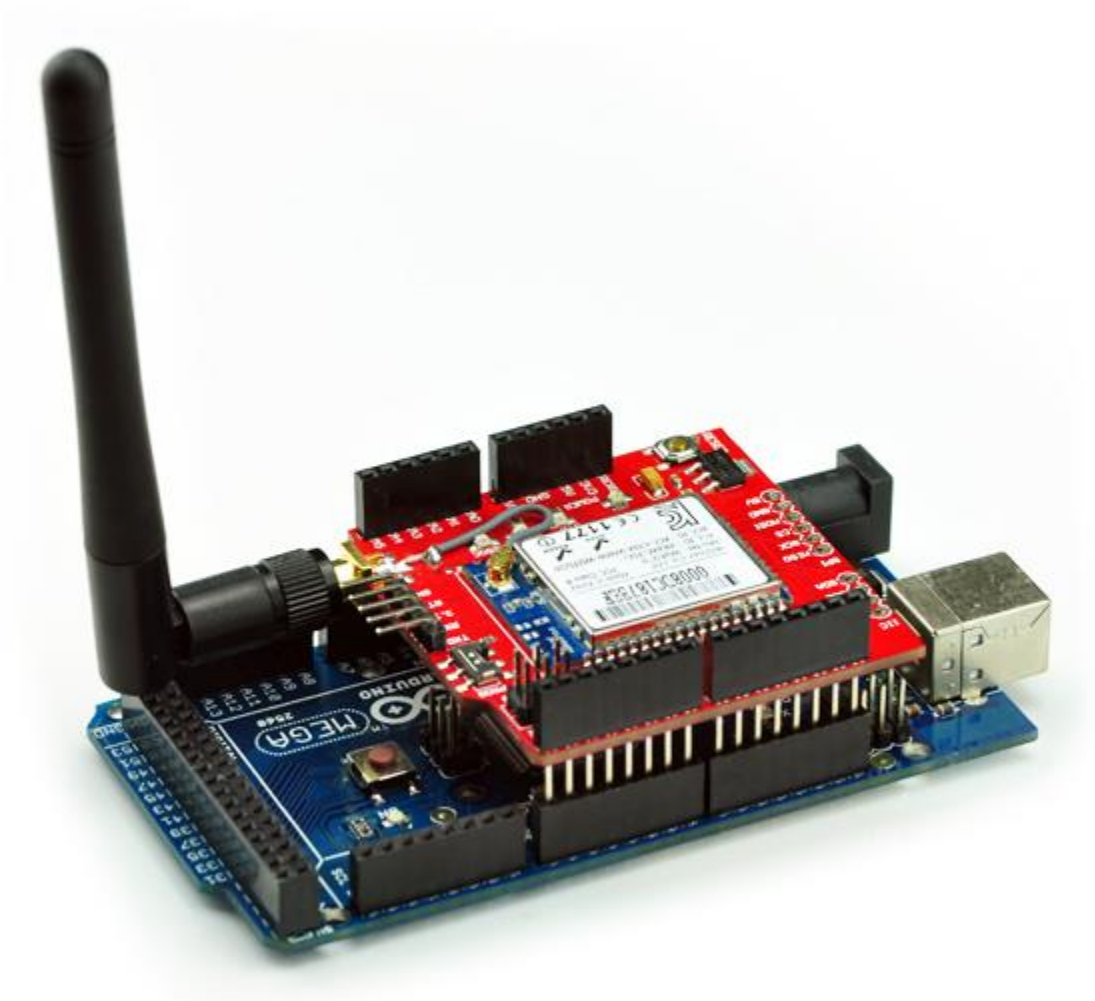
- A placa do Arduino tem como principal objetivo interagir com sensores e atuadores, servindo como controlador primário do sistema de automação, e, eventualmente, comunicando esses dados a um servidor.
  - Para que o Arduino leia dados de sensores, a placa dispõe de diversas *portas de entrada* de dados
  - Para que o Arduino envie comandos a atuadores, a placa dispõe de diversas *portas de saída* de dados
  - Para que o Arduino comunique-se numa rede de dispositivos de forma diferente do cabo USB, o projeto comporta a incorporação de *shields* à placa principal, estendendo suas funcionalidades.
  - Cada *shield* padrão tem uma biblioteca específica para sua utilização

ARDUINO COM SHIELD ETHERNET

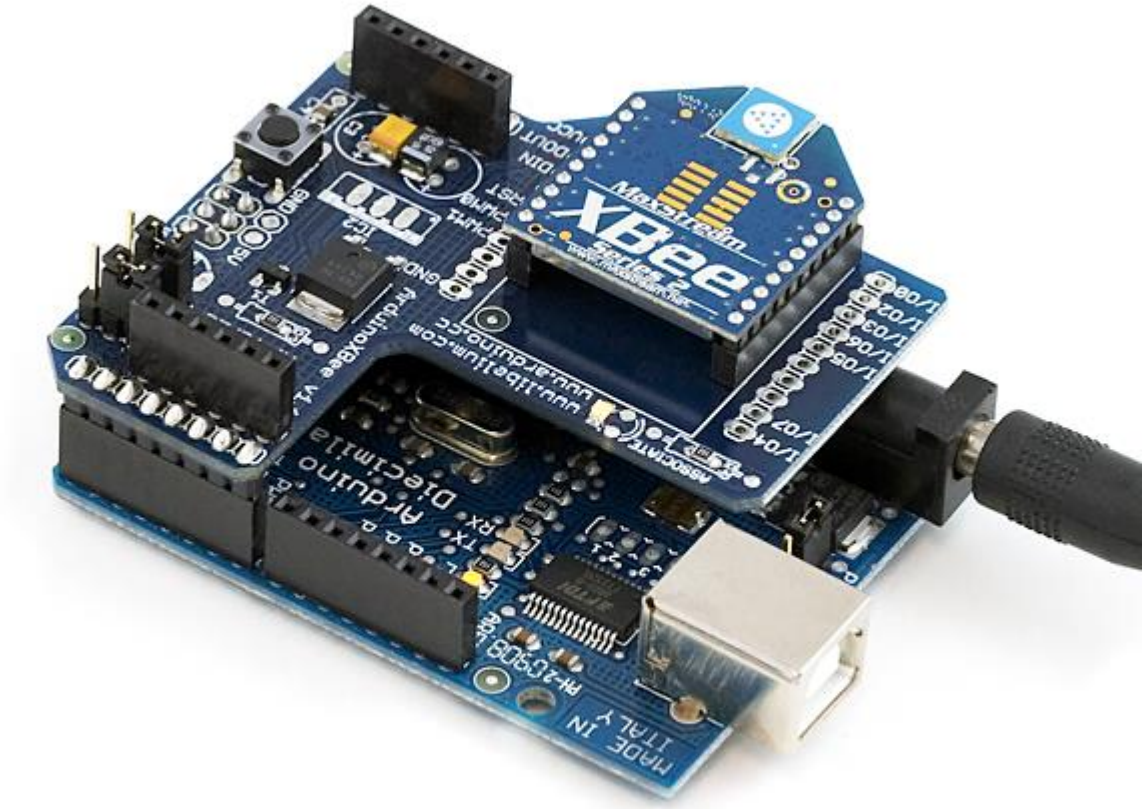
ARDUINO COM SHIELD BLUETOOTH



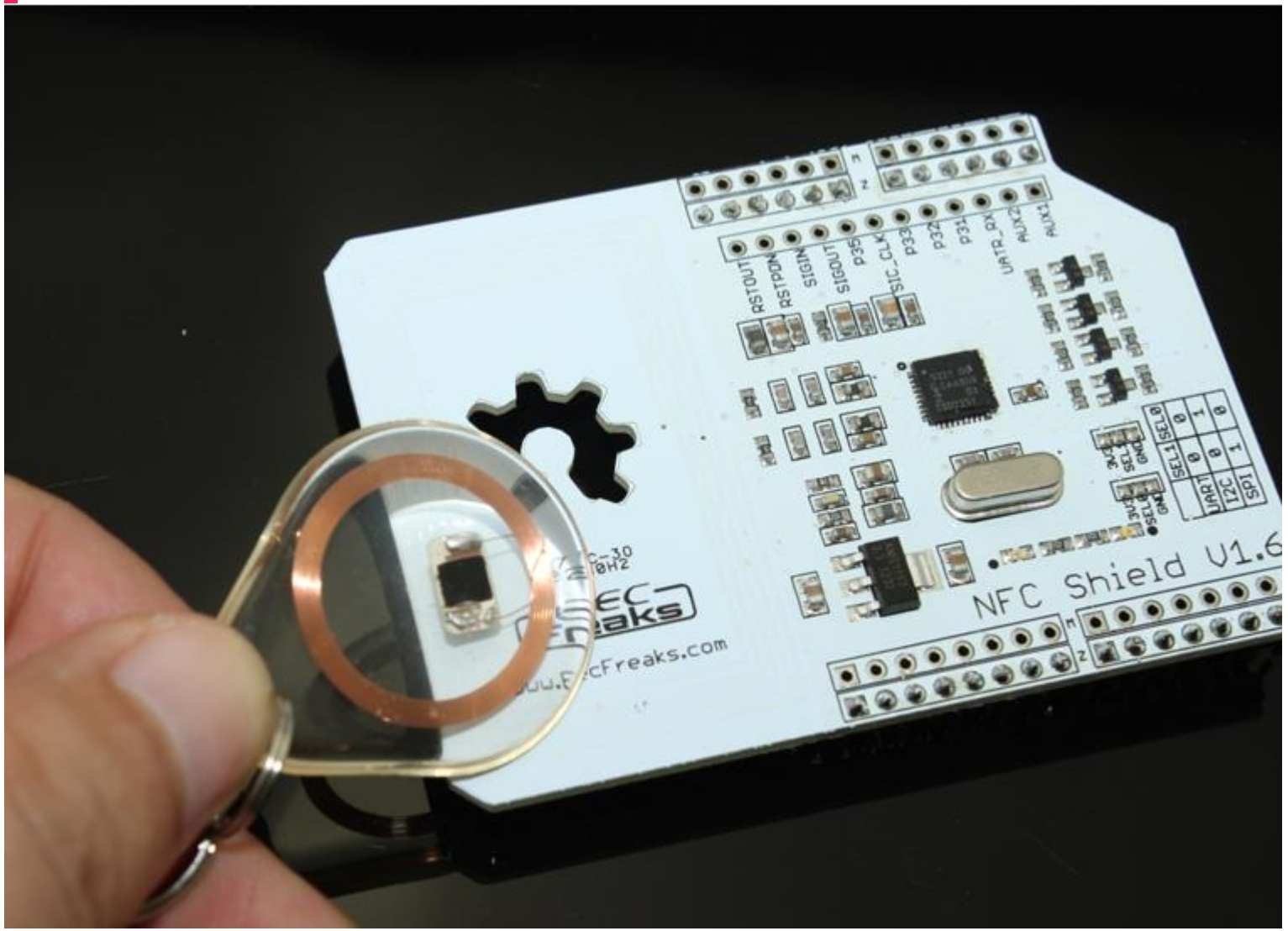
ARDUINO COM SHIELD WIFI



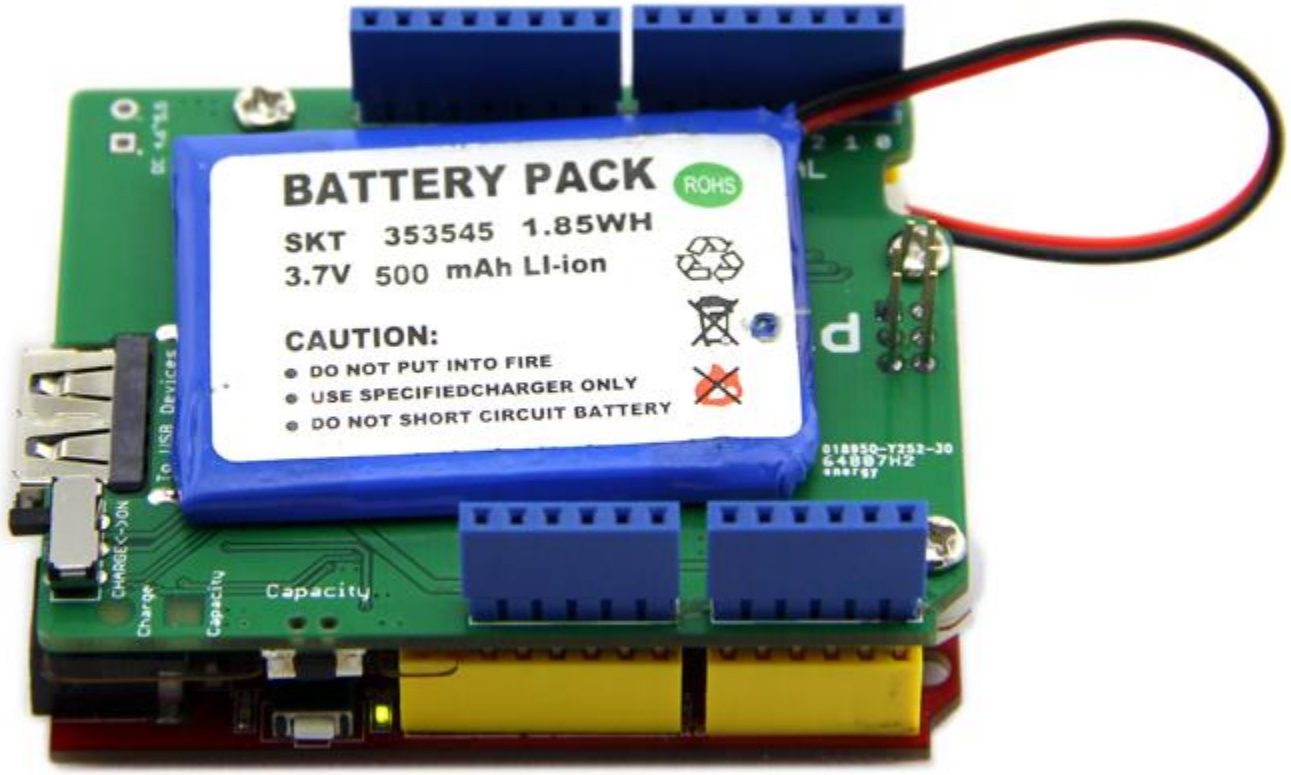
ARDUINO COM SHIELD ZIGBEE



SHIELD NFC


RSTOUT
RSTPDN
SIGIN
SIGOUT
P35
SIC_CLK
P33
P32
P31
UART_RX
AUX2
AUX1
SEL0
SEL1
UART
I2C
SPI
NFC Shield V1.6

ARDUINO COM SHIELD DE BATERIA



Sensores

Fazem a leitura de dados

- A leitura dos sensores é realizada pelas portas de entrada
  - São 6 portas de entrada analógicas...
  - Mais 14 portas de entrada ou saída digitais (GPIO)
- Sensor analógico: a saída do sensor é um nível de tensão, que deve ser capturado em uma das entradas analógicas e tem seu valor comparado com o valor de referência
  - No Arduino Uno, o padrão é 5V para essa referência
  - O valor retorna do varia de 0 (indicando 0V) a 1023 (indicando 5V)
  - Exemplos: sensor de temperatura, umidade, pressão do ar, luminosidade, etc.
- Sensor digital: a saída do sensor é um sinal ligado ou desligado, ou seja, um valor 0 ou 1 lógico
- A porta digital deve ser configurada para a leitura (modo input)
- Exemplo: apertar um botão

Tensão e corrente

- Trabalhar com o Arduíno requer o conhecimento de conceitos como tensão e corrente elétrica
  - Arduíno trabalha com lógica de 5V.
  - Cada porta de saída pode fornecer até 40 mA.
  - O que isso significa?
- A energia elétrica flui através do movimento das cargas elétricas livres, presentes nos materiais condutores elétricos, provocada por uma elevação do potencial elétrico em um ponto desse condutor.
  - Da mesma forma que uma pedra rola do alto da montanha para o vale , a carga elétrica livre tende a mover-se do ponto com maior potencial elétrico para o ponto de menor potencial
  - À diferença no potencial elétrico de um ponto a outro chamamos **diferença de potencial** ou **tensão elétrica**
  - À vazão de cargas elétricas que se movimentam dentro do condutor em um certo ponto chamamos **corrente elétrica**

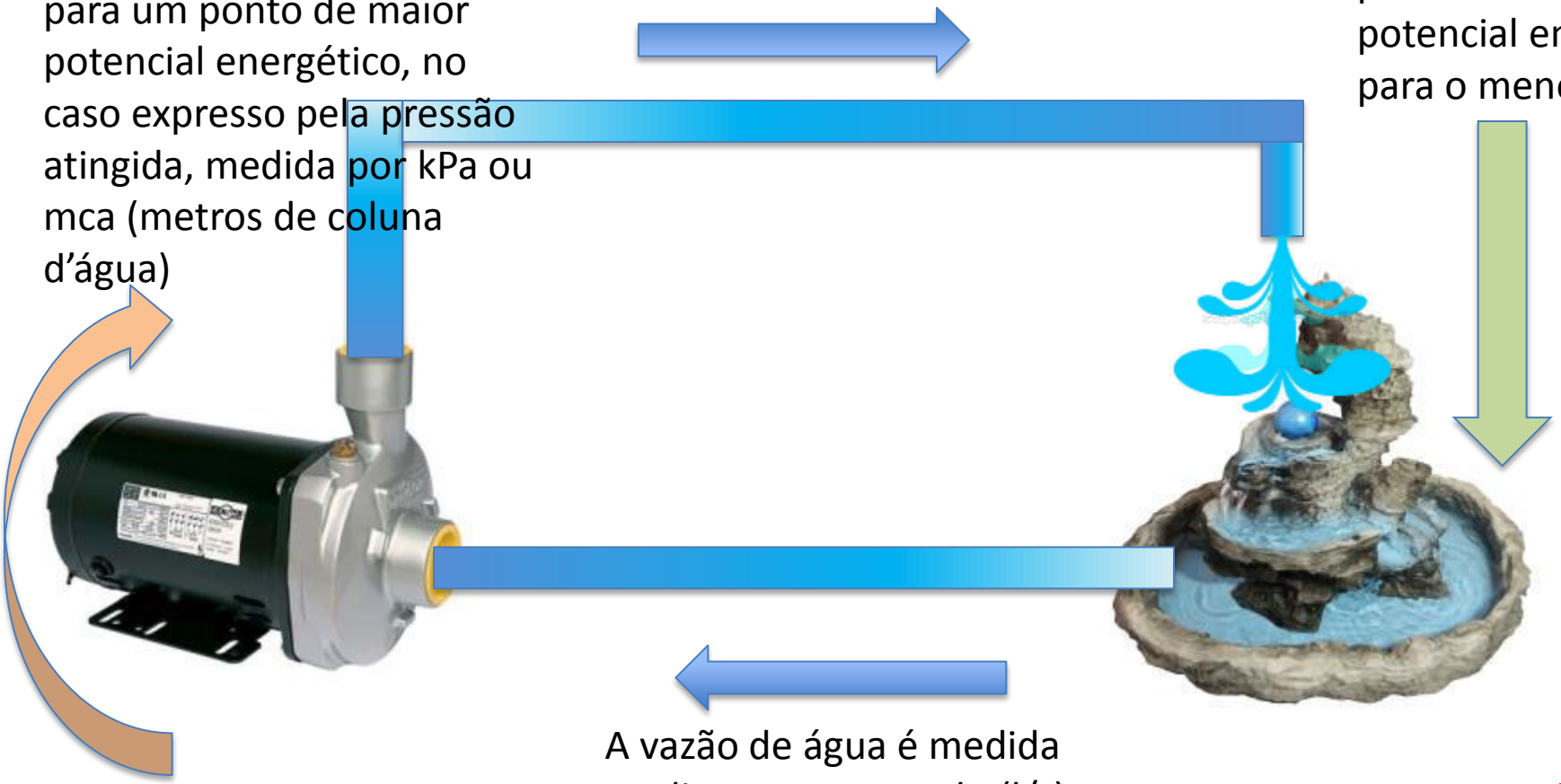
Circuitos elétricos

- Para usarmos a energia elétrica, precisamos criar uma diferença de potencial, que por sua vez irá impor uma corrente elétrica a um material condutor
- A maneira que usamos a energia elétrica é na forma de um circuito elétrico, onde a corrente elétrica está sempre circulando devido a um fornecimento ininterrupto de tensão elétrica.
- Vamos comparar o circuito elétrico a uma fonte que está sempre jorrando água

Circuito elétrico: analogia com uma fonte

A bomba d´água usa energia externa para levar a água para um ponto de maior potencial energético, no caso expresso pela pressão atingida, medida por kPa ou mca (metros de coluna d'água)

A água flui naturalmente do ponto de maior potencial enérgico para o menor

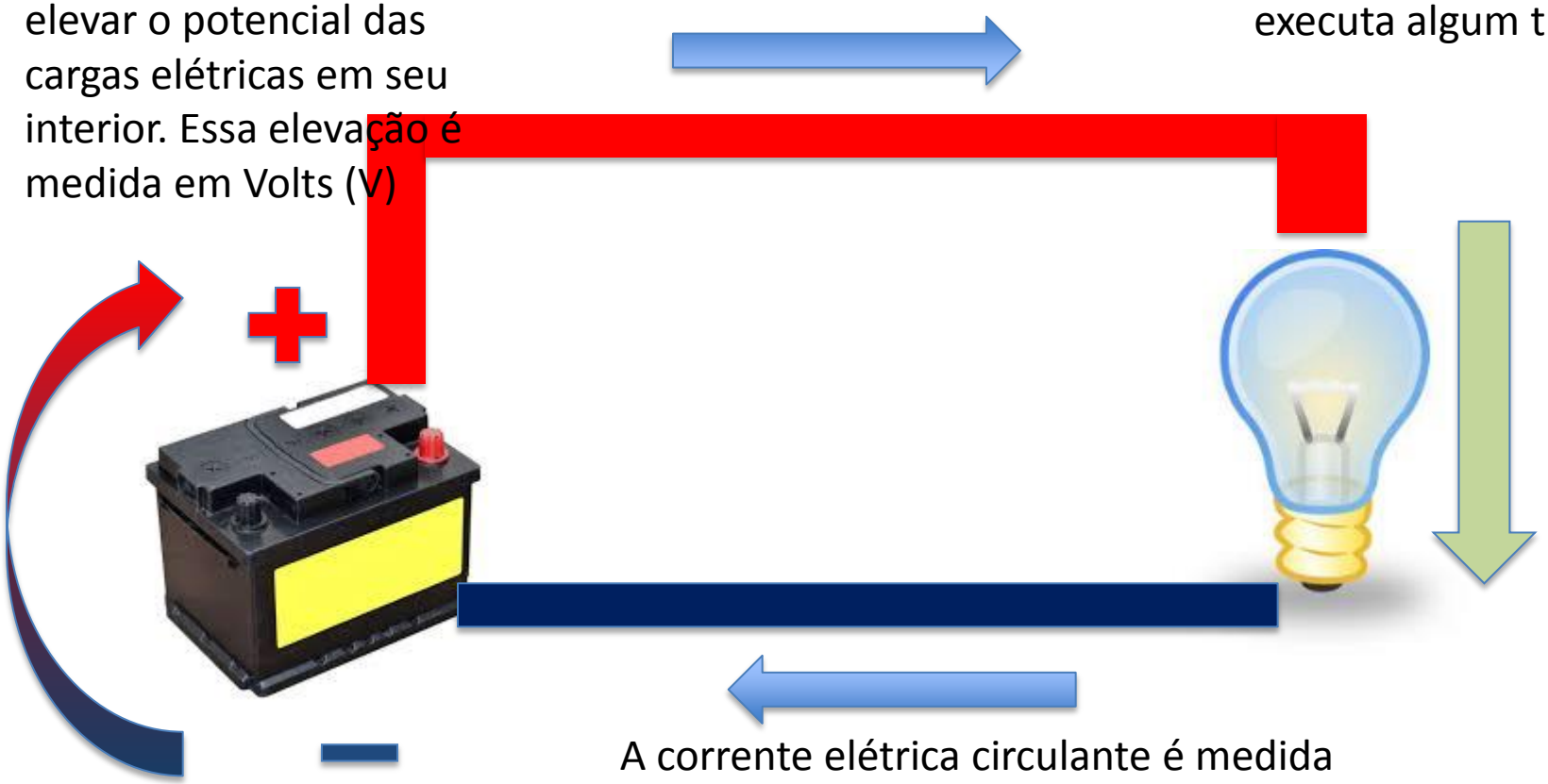


A vazão de água é medida em litros por segundo (l/s)

Circuito elétrico

Uma bateria elétrica usa algum tipo de energia para elevar o potencial das cargas elétricas em seu interior. Essa elevação é medida em Volts (V)

Uma **carga elétrica** usa a diferença de potencial fornecida para gerar uma corrente elétrica que executa algum trabalho



A corrente elétrica circulante é medida em ampères (A), ou em coulombs por segundo (C/s), onde C é a unidade de medida da carga elétrica

Sensores analógico

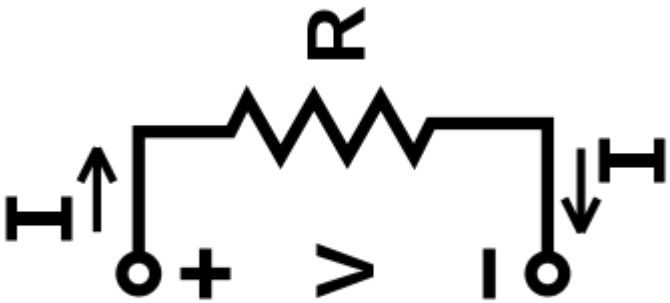
- Os sensores analógicos ou **transdutores elétricos** conseguem capturar um fenômeno físico e transformá-lo em um sinal elétrico (tensão ou corrente) que varia de forma análoga a esse fenômeno.
  - Assim, um microfone é um transdutor eletroacústico, pois transforma a variação da pressão atmosférica em uma corrente elétrica que varia da mesma forma
- Em geral esses sensores não geram sua própria diferença de potencial, por isso dependem do fornecimento externo de tensão para gerar uma corrente que varia de acordo com o fenômeno sendo medido
- Para capturar a medida do sensor, os circuitos digitais como o Arduíno precisam medir o valor dessa corrente elétrica

Medindo tensão e corrente elétrica

- A medida da tensão e da corrente elétrica estão intimamente relacionadas
- Circuitos digitais são muito bons em medir a tensão elétrica, embora, internamente, essa tensão seja transformada numa pequeníssima corrente elétrica.
- Os circuitos digitais que medem a tensão elétrica são chamados de circuitos conversores analógico-digitais (AD), pois convertem um valor de tensão elétrica em um número no formato digital (0s e 1s), composto por uma certa quantidade de bits
- Os conversores AD geram números que variam de 0 a $2^n - 1$, onde n é o número de bits da representação
  - 0 é usado para medir 0V, ou um nível muito pequeno de tensão
  - O valor máximo é usado para medir tensões acima da tensão de referência do conversor
  - No caso do Arduíno, que usa um conversor de 10 bits e tensão de referência de 5V, os valores digitais medidos pelo conversor AD ficarão entre 0 (pata 0V) e 1023 (para 5V)
- Se quisermos medir a corrente elétrica em vez da tensão, precisamos "transformá-la" em tensão elétrica através do uso da Lei de Ohm.

Lei de Ohm

- Um dos dispositivos elétricos mais simples é aquele que usa a corrente elétrica para gerar calor, os chamados **resistores elétricos**.
- Ao receberem uma diferença de potencial em seus terminais, as resistências elétricas permitem a passagem de um valor específico de corrente elétrica.
- À relação entre o valor da tensão elétrica aplicada e da corrente resultante chamamos de resistência elétrica, medida em **ohms** (Ω)
- Assim: $I = V/R$, onde
  - I é a corrente elétrica
  - V é a diferença de potencial aplicada
  - R é o valor da resistência elétrica



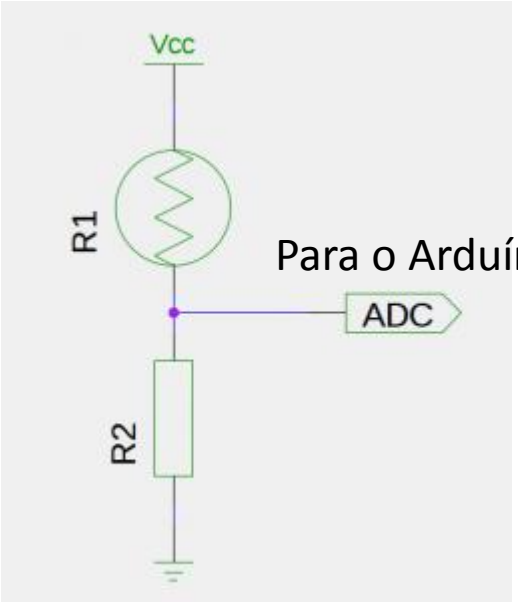
Sensor de luminosidade (LDR)

- 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

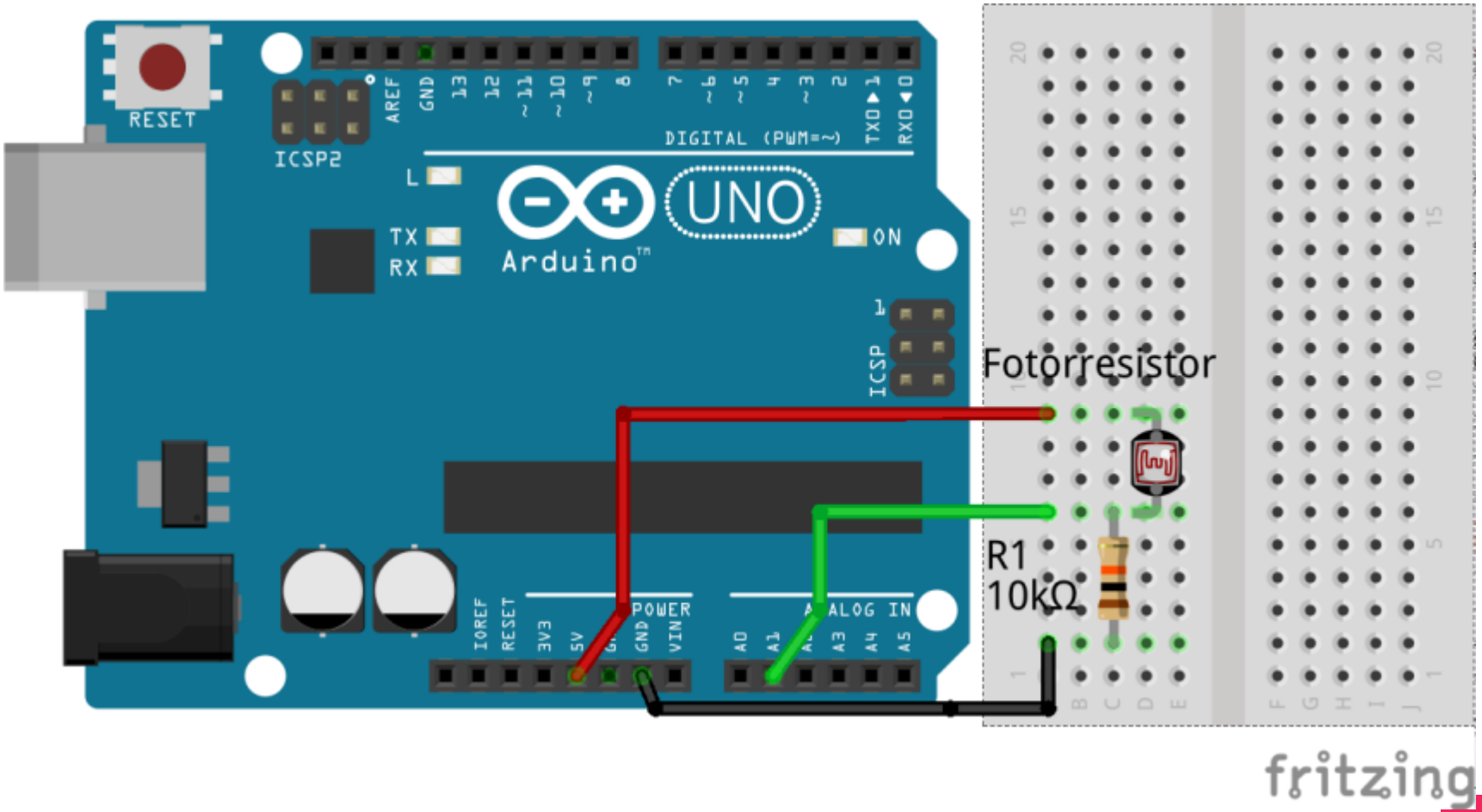
Medindo a luminosidade com o LDR

- 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

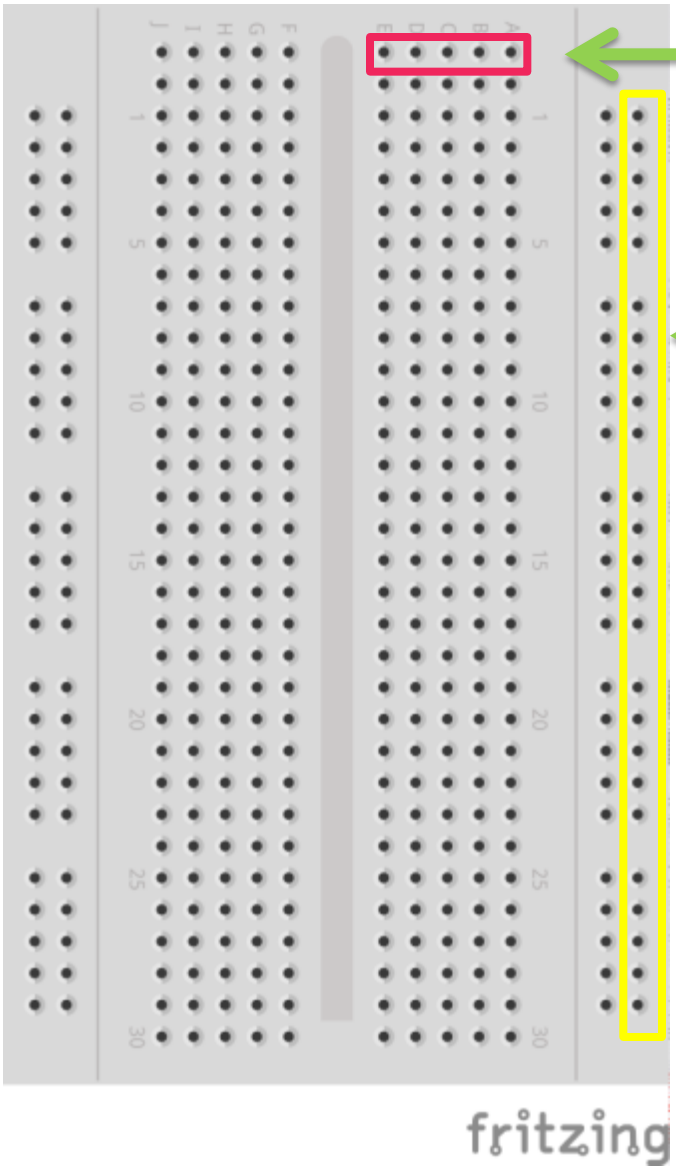


SENSORES ANALÓGICOS

Lendo um sensor de luminosidade



PROTOBOARD



- Faz a conexão elétrica entre componentes
- Cada fileira de 5 orifícios na coluna central forma uma trilha conectada, e isolada das demais trilhas
- Cada coluna de orifícios nas trilhas laterais forma uma trilha de alimentação, e está toda conectada.
  - Geralmente elas são ligadas à tensão de alimentação (5V) ou ao terra (GND ou 0V)
- Dois terminais de componentes plugados à mesma trilha estão eletricamente conectados
  - Nunca podemos conectar dois terminais de um mesmo dispositivo na mesma trilha, pois assim eles estarão em curto-circuito!

SENSORES ANALÓGICOS

Lendo um sensor de luminosidade

```
int sensor = 1;//Pino analógico em que o sensor está conectado

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);
}
```

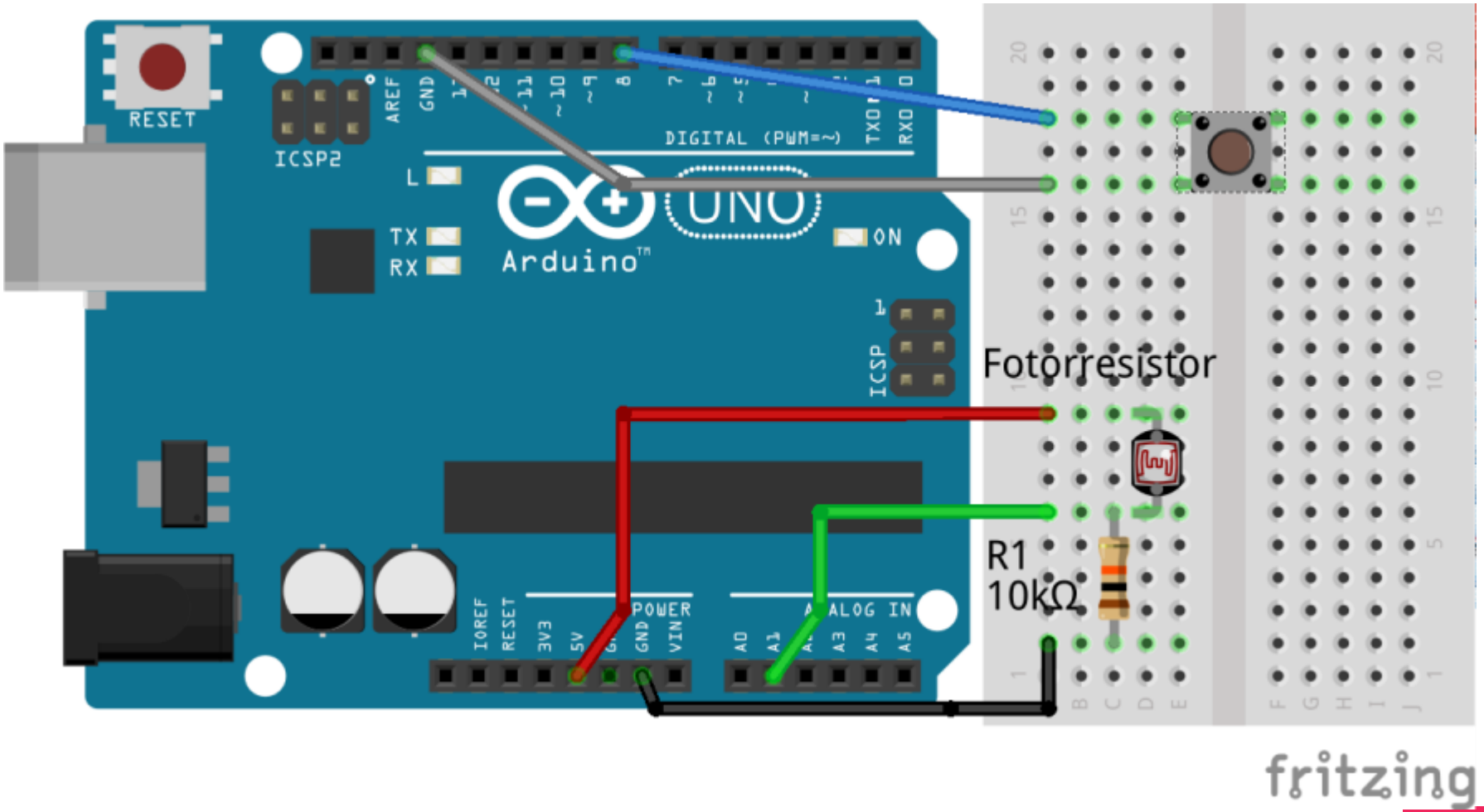
SENSORES DIGITAIS

Fazendo leitura de um botão

- Ligue um terminal do botão na porta 8, e a outra no GND (ver figura logo mais)
- Execute o programa a seguir e acompanhe pelo monitor serial:
  - Experimente apertar o botão e ver a saída

```
int inPin = 8; //entrada digital na porta 8
int val = 0;
void setup() {
  Serial.begin(9600);
  pinMode(inPin, INPUT_PULLUP); //porta 8 vira entrada
}
void loop(){
  val = digitalRead(inPin);   // read the input pin
  Serial.println(val);
  delay(2000);
}
```

SENSORES DIGITAIS



Saídas digitais

Fazem o acionamento de dispositivos do tipo liga/desliga

- As saídas D0 a D13 permitem a leitura ou escrita de valores lógicos. Por isso são chamadas de GPIO (General-Purpose Input and Output)
- As portas D0 (RX0) e D1 (TX0) são destinadas para a comunicação serial – via cabo USB ou comunicação com shields, por exemplo
- São configuradas como saídas digitais na função setup() através da chamada:
  - `pinMode(numero, OUTPUT);`
- Podem acionar dispositivos que necessitem de uma informação digital (0 ou 1) para seu acionamento, por exemplo:
  - Luz LED
  - Relê para acionamento de equipamentos elétricos
  - Dispositivo de acionamento de motores (drives)
- Na saída porta D13 há um LED conectado, que acende assim que a porta esta acionada
- Vamos executar o exemplo: `File` → `Examples` → `01.Basic` → `Blink`
  - `digitalWrite(HIGH | LOW);` aciona ou desliga a saída digital

Exemplo: piscar o LED da porta D13

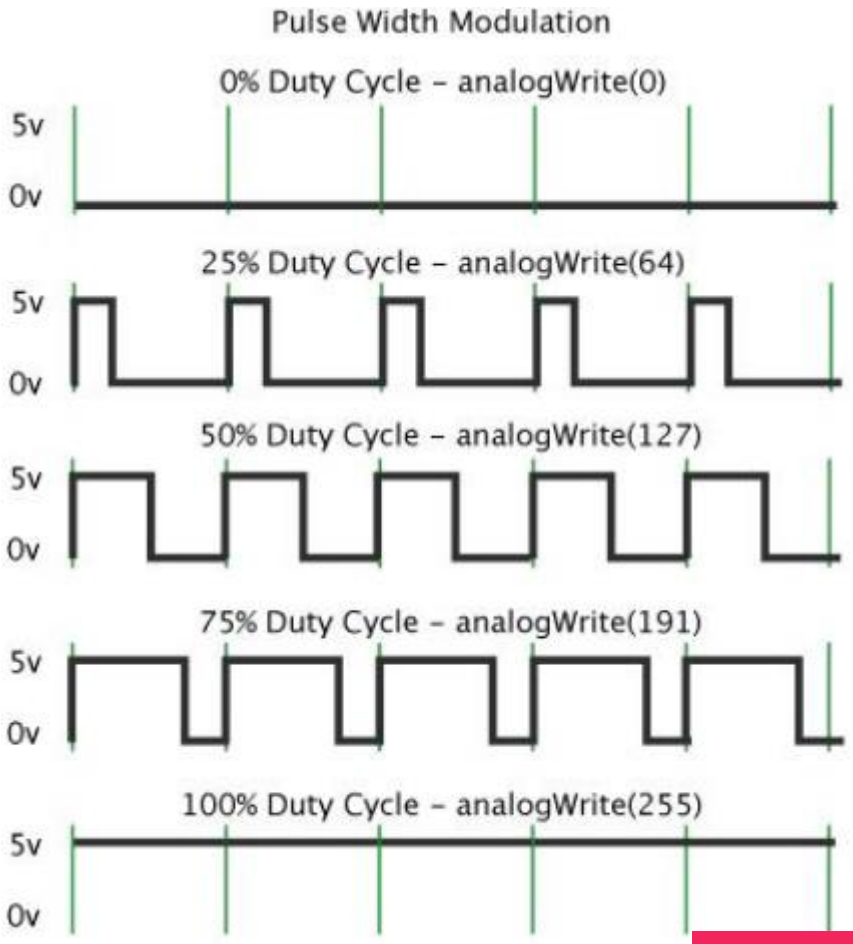
```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
// the loop routine runs over and over again forever:
void loop() {
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

SAÍDAS ANALÓGICAS (PWM)

Fazem o acionamento de dispositivos que possam ser controlados por PWM

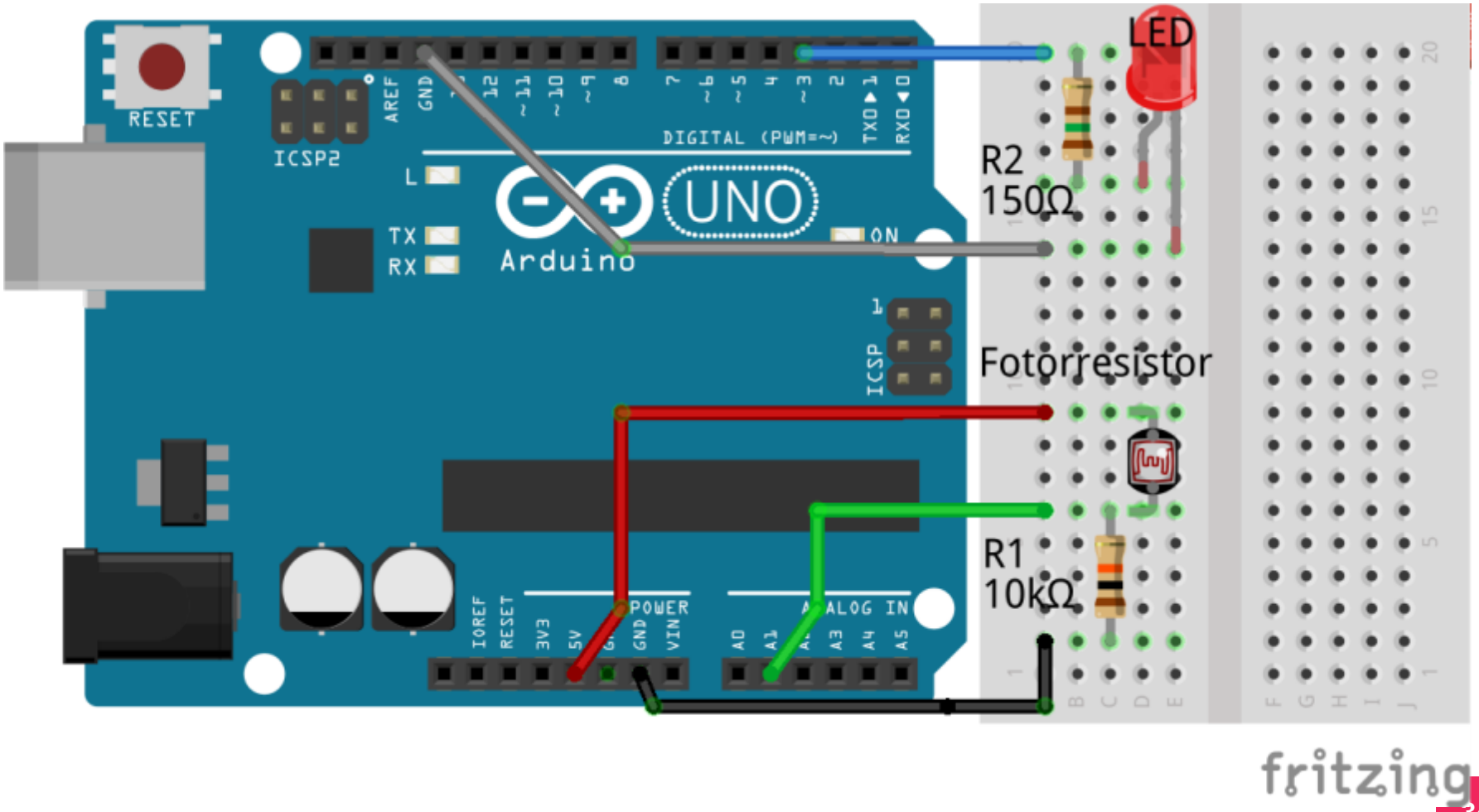
- As portas digitais (GPIO) marcadas com um ~ (til) possuem a capacidade de servirem de saída do tipo PWM – Pulse-Width Modulation
- Uma saída PWM automaticamente alterna períodos em que ela esta ligada (HIGH) e em que está desligada (LOW). A proporção do tempo em que a saída está ligada em relação ao tempo total do ciclo é chamad de Duty Cycle, e varia de 0 a 100%
- Exemplos: motor de passo, iluminação LED ou acionador dimerizável de lâmpada, LED RGB (seleciona a cor do LED)
- Para escrever um valor de 0 a 255 na saída PWM, esta deve estar configurada como uma saída digital, sendo que a escrita se dá da forma:

```
analogWrite(numero, valor);
```



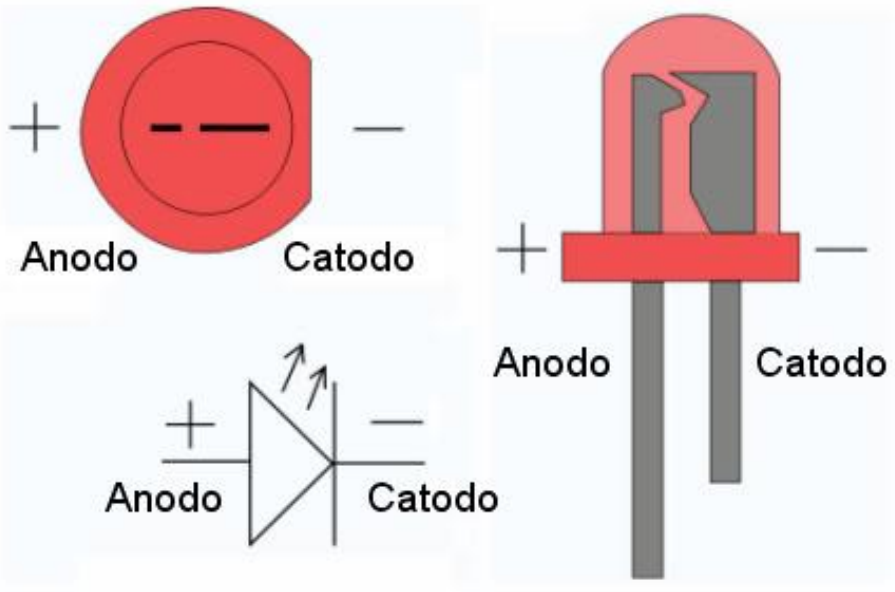
SAÍDA ANALÓGICA

Lendo um sensor de luminosidade



Instalação do LED

- Deve-se sempre interpor um resistor de 100 a 150 ohms em série ao LED, para evitar uma sobretensão no mesmo
  - O resistor pode estar interposto tanto ao catodo quanto ao anodo, indistintamente
- O LED possui polaridade. Seus terminais são:
  - Anodo: correspondente ao polo positivo, deve ser ligado à porta controladora
  - Catodo: correspondente ao polo negativo, deve ser ligado ao GND (0V)



SAÍDA ANALÓGICA

Controlando a intensidade do LED a partir do sensor de luminosidade

```
int sensor = 1;//Pino analógico em que o sensor está conectado
int led = 3; //Pino em que o led está conectado

void setup(){
  Serial.begin(9600);
  pinMode(led,OUTPUT);
}
void loop(){
  //Lendo o valor do sensor.
  int valorSensor = analogRead(sensor);
  //Exibindo o valor do sensor no serial monitor.
  Serial.println(valorSensor);
  //Acionando o LED: quanto menos luz externa, mais forte o LED
  //map(valor,deEscalaAnt,ateEscalaAnt,deNovaEscala,ateNovaEscal)
  analogWrite(led, map(valorSensor,0,1023,255,0));
  delay(50);
}
```

SENSOR DHT-11

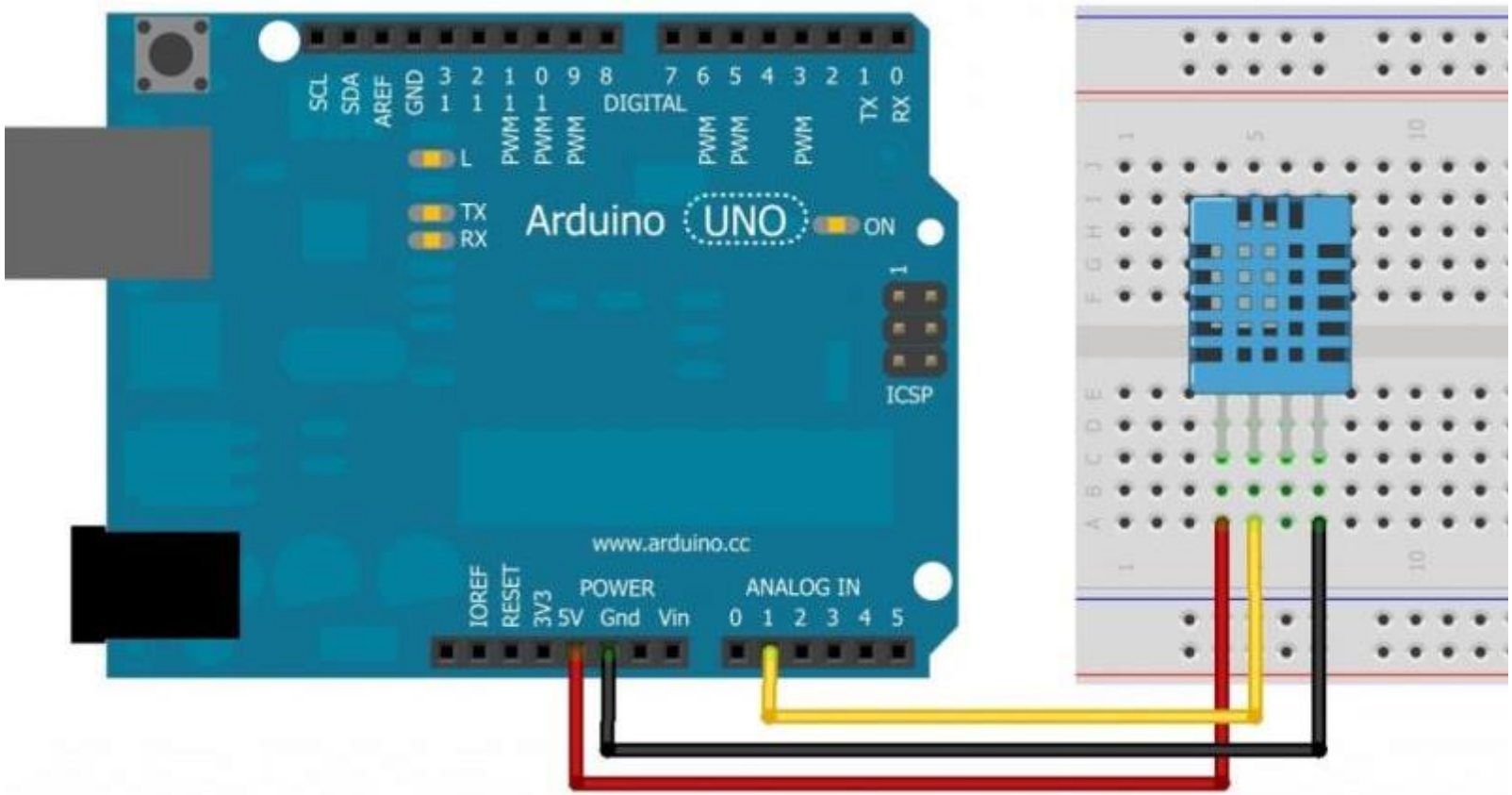
Ou como funcionam as bibliotecas

- O modo mais simples de se trabalhar com bibliotecas no Arduino é através de arquivos compactados
- Elas possuem definição de classes e podem conter exemplos de utilização
- Instalar o arquivo DHT.zip através de
  - `Sketch → Import Libraries → Add Library...`

- O sensor DHT11 fornece tanto temperatura quanto umidade do ar instantaneamente e de forma muito fácil.
  - Isso se deve à biblioteca que cuida de todas as funções necessárias, restando para nós apenas acessar aos dados.

SENSOR DHT-11

Lendo os dados através da porta A1



LENDO O SENSOR DHT-11

```
#include "DHT.h"
#define DHTPIN A1 // pino que estamos conectado
#define DHTTYPE DHT11 // DHT 11
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 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");
  }
}
```

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 sobrepressão, 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 sobrepressão 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

Ligar a porta D2 em GND usando o botão

```
int led = 13; //Porta do LED
int interruptPort = 2; //porta da interrupção
int interruptNumber = 0; //ID da interrupção
//Variáveis modificadas por interrupções devem ser volatile
volatile int state = LOW;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(interruptPort, INPUT_PULLUP);
  attachInterrupt(interruptNumber, toggle, CHANGE);
}

void loop() {
   // Qualquer processamento mais longo...
}

void toggle() {
  state = !state;
  digitalWrite(led,state);
}
```

INTERRUPÇÕES NO ARDUINO

Como funcionam

- A definição de uma **ISR** no Arduino é dada por uma função sem argumentos e sem valor de retorno (void).
- 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)
  - `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

ARDUÍNO – PORTA SERIAL

Como ler a porta serial do Arduino

- Da mesma forma como podemos escrever dados na porta serial do Arduino, enviando dados para o computador, podemos também ler os dados que a placa recebe pela mesma porta
- Podemos receber comandos do computador para executar alguma ação
- No exemplo a seguir, vamos ler o valor da potência do LED a partir da porta serial (de 0 a 255)

- Protocolo **Firmata**: controla o Arduino via porta serial através de um programa de computador externo. Assim, podemos mudar a programação sem ter que reprogramar o Arduino (não usaremos agora...)

- Para auxiliar na recuperação dos dados da porta serial, usamos a classe String do Arduino

ARDUÍNO – STRING

Representando strings no Arduino
Mais métodos de String em https://www.arduino.cc/en/Reference/StringObject

- A forma tradicional de representar strings em C é através de arrays de caracteres
  - `char string_do_c[256] = "Ola, isto eh uma string";`
- No entanto, a API do Arduino fornece a classe String, que é bem mais flexível:
  - `String string_do_Arduino = "Isto eh uma string do Arduino";`
- Criando uma String a partir da concatenação de valores
  - `String outraString = String("Valor: ") + 128;`
- Anexando valores:
  - `outraString += ", outro valor:"; outraString += 256;`
- Interpretando valores numéricos, retornando zero no caso de erro
  - `int numero = minhaString.toInt();`
  - `float numFloat = minhaString.toFloat();`

ARDUINO - LENDO A PORTA SERIAL

```
const int LED = 3;
char nextChar = 0, lendo = 0;
String valor;
void setup() {
  Serial.begin(9600);
  pinMode(LED,OUTPUT);
}
void loop() {
  if (Serial.available() > 0) {
    // lê o byte disponível na porta serial:
    nextChar = Serial.read();
    if(nextChar == 'B') {
      lendo = 1; //lendo <- true
      valor = "";
    } else if(nextChar == 'E') {
      lendo = 0; //lendo <- false
      analogWrite(LED,valor.toInt());
      Serial.println(String("Potencia do LED: ") + valor);
    } else if(lendo && nextChar >= '0' && nextChar <= '9') {
      valor += nextChar;
    }
  }
}
```

Decodificando números e recebendo linhas de texto

- A API do Arduino possui funções que consomem os caracteres disponíveis na porta Serial de acordo com alguma regra
- As funções **Serial.parseInt()** e **Serial.parseFloat()** leem a porta serial em busca de um número inteiro/ponto flutuante
  - Caso caracteres não numéricos sejam encontrados, eles são pulados
  - Essas funções aguardam por um tempo pré-especificado por mais caracteres até retornarem o valor definitivo. Esse tempo é 1000 ms por padrão, e pode ser configurado através de **Serial.setTimeout(int milisseg)**
  - Caso não seja possível decodificar um número, o resultado é 0 (zero)! Isso pode gerar confusões no programa, então fique atento!
- A **função Serial.readBytesUntil()** lê caracteres vindo da porta serial, escrevendo em um vetor de caracteres.
  - A função termina quando o caractere de terminação for detectado, o tamanho máximo for lido ou o tempo de espera por mais texto terminou
  - **Serial.readBytesUntil(char terminador, char buffer[ ] , int tamanho)**

Arduino – Lendo a porta Serial (outro modo)

```
const int LED = 3;
char nextChar = 0;
void setup() {
  Serial.begin(9600);
  pinMode(LED,OUTPUT);
}
void loop() {
  if (Serial.available() > 0) {
    // lê o byte disponível na porta serial:
    nextChar = Serial.read();
    if(nextChar == 'B') {
      //Lê o próximo inteiro vindo da serial
      int valor = Serial.parseInt();
      //Atenção: em caso de erro o valor lido será 0
      analogWrite(LED,valor);
      Serial.println(String("Potencia do LED: ") + valor);
    }
  }
}
```

LENDO A PORTA SERIAL

Escrevendo na porta serial

- Podemos escrever na porta serial do Windows de uma forma bem simples através do Serial Monitor do Arduino (barra de texto superior)

```
B127E
```

- O ideal é termos um programa no computador que se comunicasse com o Arduino via porta serial, e que expusesse um serviço para que possa ser controlado remotamente, através de algum protocolo para troca de dados de sensores e comandos para os atuadores.

- Para formatar as mensagens e garantir que uma ampla gama de programas consigam se comunicar com o Arduino, vamos usar o formato JSON.

JSON – JavaScript Object Notation

- Do próprio site **json.org**:
  - JSON é um formato leve de troca de dados (serialização), de fácil leitura e escrita por humanos e máquinas
  - É parte da especificação de 1999 do JavaScript, que codifica e decodifica JSON nativamente
  - JSON é um formato de texto completamente independente de linguagem
- Exemplo de estrutura JSON:
  - {"nome": "João", "idade": 23, "mulher": false,
    "filhos": ["Pedro", "Artur"] }
  - A quebra de linha é opcional, porém facilita a visualização humana
- Podemos validar um JSON através do site
  `http://jsonlint.com/`

Valores em JSON

- Um valor escrito em JSON pode assumir um dos seguintes formatos:
  - **String**:
    - texto unicode não formatado, escrito sempre entre aspas (como em Java)
    - Exemplos: `"José"`, `"Marçäl"`, `"opa123"`, etc.
  - **Número**:
    - sequência de dígitos com separador decimal (ponto) e notação científica, como em Java, mas aceita apenas números decimais
    - Exemplos: `12`, `15.01`, `1.35e-24`
  - **Objeto** (object):
    - Pares do tipo chave:valor contidos entre chaves ({ }) e separados por vírgula
    - Exemplo: `{"idade":23,"peso":53.5}`
  - **Vetor** (array):
    - Conjunto ordenado de valores contidos entre colchetes ([ ]) e separados por vírgula
    - Exemplo: `[1, 2.5, "três", [4], {"próx": 5}]`
  - **`true`**, **`false`**: constantes lógicas representando verdadeiro e falso, respectivamente
  - **`null`**: constante indicando um valor nulo

Objeto do JSON

- É a estrutura de dados mais emblemática do JSON
- É composto por um conjunto de pares do tipo `chave:valor` separados por vírgula
  - `Chave` deve ser uma **string**, que serve de rótulo para o valor
  - `Valor` é qualquer valor válido do JSON, incluindo um array ou ainda outro objeto
- Alguns objetos válidos
  - { } : objeto vazio
  - { "chave " : "valor " }
  - { "nome" : "Alberto", "idade" : 54, " pais" : [" José", "Maria"] }

Exemplo de um valor JSON válido

```
[
  {
    "id": 100,
    "nome": "Astolfo",
    "sobrenome": "Silva",
    "endereco": {
      "rua": "Rua das Orquideas",
      "no": 23
    }
  },
  {
    "id": 101,
    "nome": "Maria",
    "sobrenome": "Teresa",
    "idade": 49
  }
]
```

JSON no Arduino

- Há várias bibliotecas em C++ para a codificação e a decodificação de JSON, porém nem todas são otimizadas para rodar no Arduino
- A biblioteca que vamos adotar aqui é a **ArduinoJson** (https://github.com/bblanchon/ArduinoJson), que relaciona objetos JSON com a estrutura de dados de dicionário do C++
- Os dicionários do C++ também relacionam uma chave (ou índice) a um rótulo, acrescentando dinamicamente elementos
  - meuDic["nome"] = "Pedro Henrique"; //Acrescenta um elemento
  - long valor = meuDic["idade"]; // Lê o elemento idade
- Para usar a API, a primeira providência é importar o seu cabeçalho no código, trazendo na primeira linha do programa:
  - `#include <ArduinoJson.h>`

Criando um objeto JSON

- Primeiramente, devemos reservar memória para a criação do objeto (aqui é Arduino, não esqueçam)
  - `StaticJsonBuffer<200> jsonBuffer;  //Reserva 200 bytes`
- Então devemos alocar a árvore JSON na memória
  - `JsonObject& raiz = jsonBuffer.createObject();`
  - Podemos pensar em um objeto ou array JSON como sendo uma árvore porque cada elemento é considerado um filho, que por sua vez podem ser estruturas contendo outros objetos ou arrays, e assim por diante
- Agora podemos criar os elementos, com formatos identificados automaticamente
  - `raiz["sensor"] = "gps";`
  - `raiz["time"] = 1351824120;`
- Valores decimais devem ser especificados com o número de casas decimais desejadas, caso contrário são usadas apenas 2 casas. O exemplo abaixo usa 4 casas:
  - `raiz["pi"] = double_with_n_digits(3.1415, 4);`
  - Para acrescentar um array ou outro objeto, é necessário usar um método especial
  - `JsonArray& vetor = raiz.createNestedArray("vetor");`
  - `vetor.add("José"); vetor.add(48.756080, 6); //Usa 6 casas`
  - `JsonObject& obj = raiz.createNestedObject("obj");`
- Finalmente, imprimimos a string JSON resultante na porta serial ou em uma string do C
  - `raiz.printTo(Serial); //manda o resultado pela porta serial`

Lendo um JSON

- Para decodificar um JSON é ainda mais fácil. Basta reservar a memória, checar possíveis erros e resgatar os valores desejados
  - ```
    char json[] = "{
    \"sensor\":\"gps\",\"time\":1351824120,
    \"data\":[48.756080,2.302038]}";
    ```
  - ```
    StaticJsonBuffer<200> jsonBuffer;
    ```
  - ```
    JsonObject& raiz =
    jsonBuffer.parseObject(json);
    ```
  - ```
    if (!raiz.success()) {/* Tratar o erro*/}
    ```
- Capturando os valores:
  - ```
    const char* sensor = raiz["sensor"];
    ```
  - ```
    long time = raiz["time"];
    ```
  - ```
    double latitude = raiz["data"][0];
    ```
  - ```
    double longitude = raiz["data"][1];
    ```
- Exercício: criar um programa que manda a luminosidade, temperatura e umidade no formato JSON pela porta serial, enquanto aceita comandos para o LED usando JSON. Escolha o nome dos campos a serem preenchidos.

Lendo o JSON da porta Serial e mandando a luminosidade

```
#include <ArduinoJson.h>
const int LED = 3;
const int LUZ = A1;
const int TAMANHO = 200;
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(10); //1000ms é muito tempo
  pinMode(LED,OUTPUT);
}
void loop() {
  if (Serial.available() > 0) {
    //Lê o texto disponível na porta serial:
    char texto[TAMANHO];
    Serial.readBytesUntil('\n', texto, TAMANHO);
    //Grava o texto recebido como JSON
    StaticJsonBuffer<TAMANHO> jsonBuffer;
    JsonObject& json = jsonBuffer.parseObject(texto);
    if(json.success() && json.containsKey("led")) {
      analogWrite(LED, json["led"]);
    }
  }
  StaticJsonBuffer<TAMANHO> jsonBuffer;
  JsonObject& json = jsonBuffer.createObject();
  json["luz"] = analogRead(LUZ);
  json.printTo(Serial); Serial.println();
  delay(1000);
}
```

REFERÊNCIAS

- http://www.telecom.uff.br/pet/petws/downloads/tutoriais/arduino/Tut_Arduino.pdf
- http://arduino.cc/en/Reference/HomePage
- https://www.arduino.cc/en/Reference/StringObject
- http://json.org

FIAP