

Interfaces físicas para dispositivos móveis

Felipe Navarro Balbino Alves

Github do curso

https://github.com/fnbalves/curso_interfaces_fisicas_android/

Interface homem máquina

Como nós enxergamos um computador?



O conjunto

Monitor Mouse Teclado

Nos permite identificar facilmente um computador

Interface homem máquina

Como o computador nos enxergaria?



Dedos

[teclado/mouse]

Olho

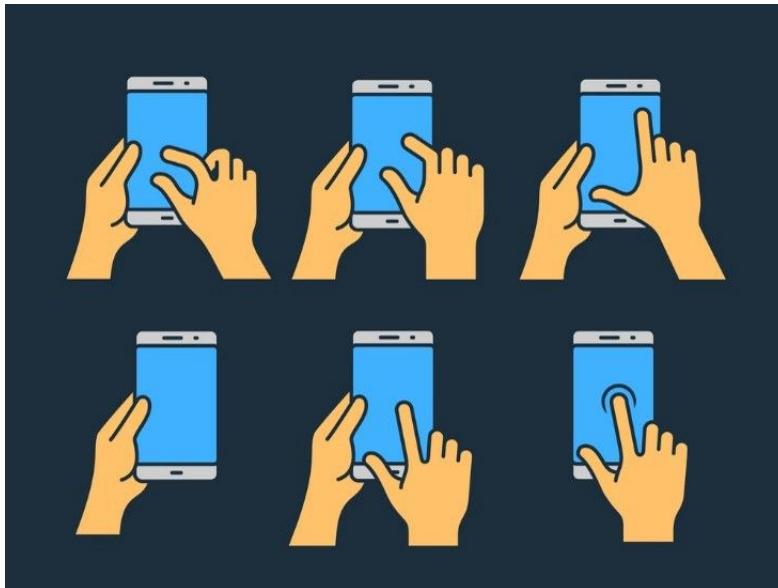
[monitor]

Duas orelhas

[caixa de som]

Interface homem máquina

No entanto, os computadores começaram a mudar, e, consequentemente, a forma como interagimos com os mesmos



Interface homem máquina

O surgimento de telas touch screen não se restringiu aos smartphones....



Ainda assim, a forma como interagimos com nossos dispositivos ainda pode evoluir bastante...

Computação física

“Mudar a forma como os computadores nos veem mudará como eles interagem conosco”

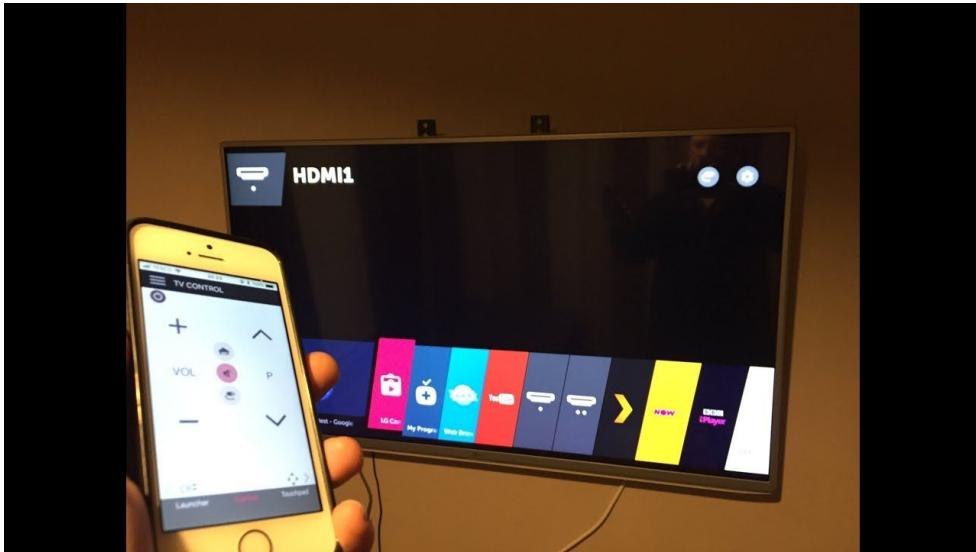
Tom Igoe - Physical Computing

Xbox accessible controller



Computação física

A computação física também pode mudar a forma como interagimos com o próprio ambiente



A abordagem deste curso

Ao longo deste curso, empregaremos conhecimentos de computação e eletrônica para desenvolvermos aplicações de computação física

Computação: Comportamento definido por software

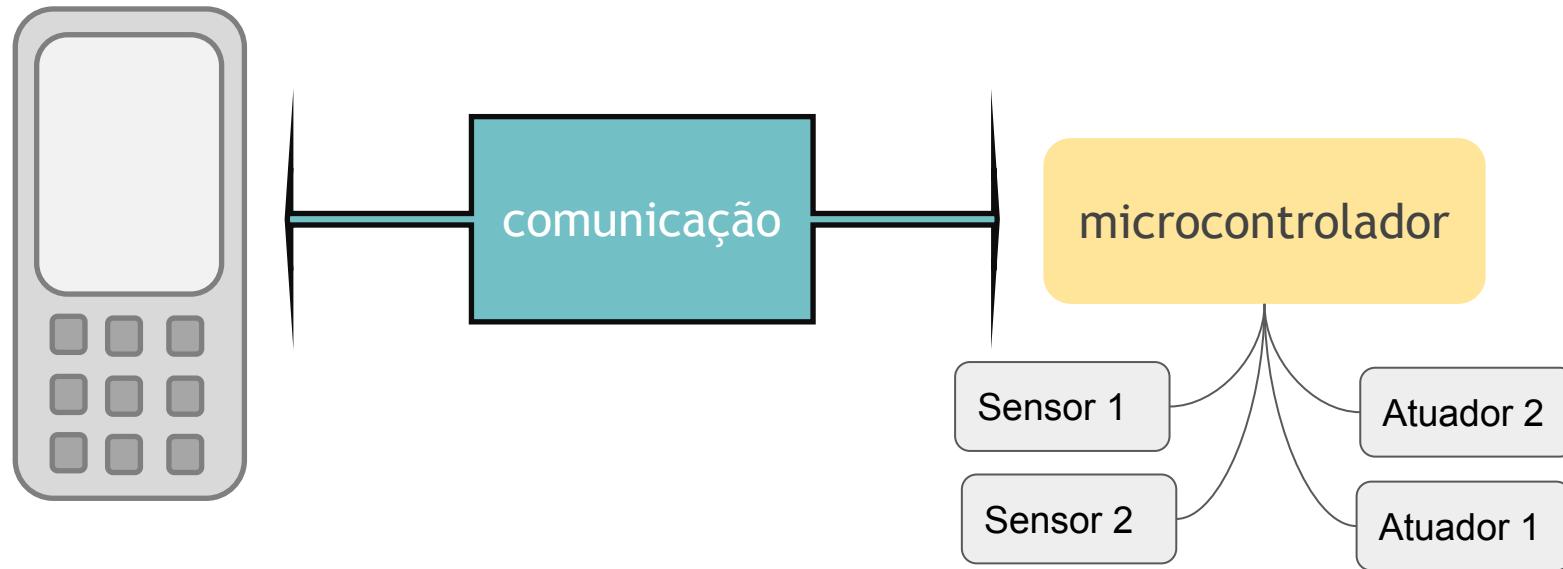
Eletrônica: Uso de microcontroladores e circuitos em geral

O que são microcontroladores (MCU) ?

Microcontroladores são pequenos computadores em circuito integrado com um **número reduzido de instruções** além de um conjunto de **periféricos** (ex: Memória, interfaces de comunicação, etc...). São utilizados em sistemas computacionais com **uso específico**.



Arquitetura das soluções desenvolvidas



Arquitetura das soluções desenvolvidas

Conectados ao microcontrolador estão

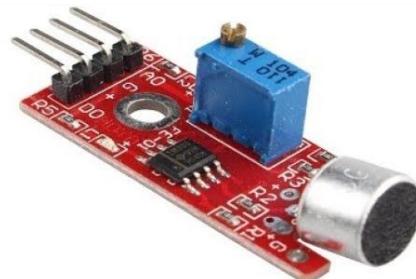
Sensores: Responsáveis por perceber o ambiente e alterações no mesmo



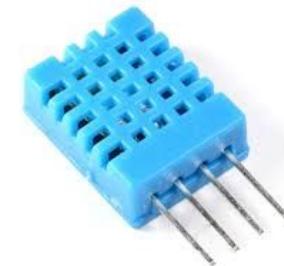
Presença



Luz



Som



Temperatura /
Umidade

Arquitetura das soluções desenvolvidas

Conectados ao microcontrolador estão

Atuadores: Responsáveis por criar alterações no ambiente



Luz



Movimento



Som

Arquitetura das soluções desenvolvidas

Smartphone

Será utilizado o sistema Android



Arquitetura das soluções desenvolvidas

Microcontrolador

Utilizaremos neste curso a plataforma **Arduino**



A plataforma arduino

É uma das plataformas de prototipagem mais conhecidas da atualidade. Possui **software e hardware livres** e **baixo custo**. O principal objetivo destas plataformas é facilitar o desenvolvimento de microcontroladores através do uso de uma **única placa** e um pacote de software para desenvolvimento



A plataforma arduino

Microcontrolador: Atmel

Linguagem de programação: Wiring (subconjunto do processing, baseado em C/C++)

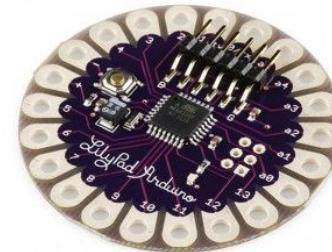
Diversos tipos:



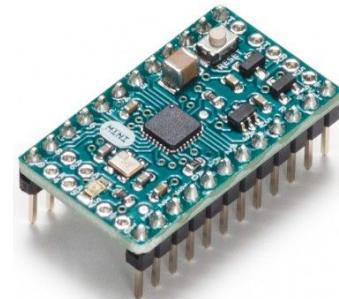
Arduino Uno



Arduino Mega



Lilypad



Arduino mini

A plataforma arduino

Portas:

14 entradas / saídas digitais

6 entradas analógicas

Memória:

RAM (1k)

Flash (programa): 16k - 2k (bootloader)

Clock (velocidade de processamento) : 16 Mhz

A plataforma arduino

Utilizando o IDE do Arduino:

Instale o software correspondente ao seu sistema

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

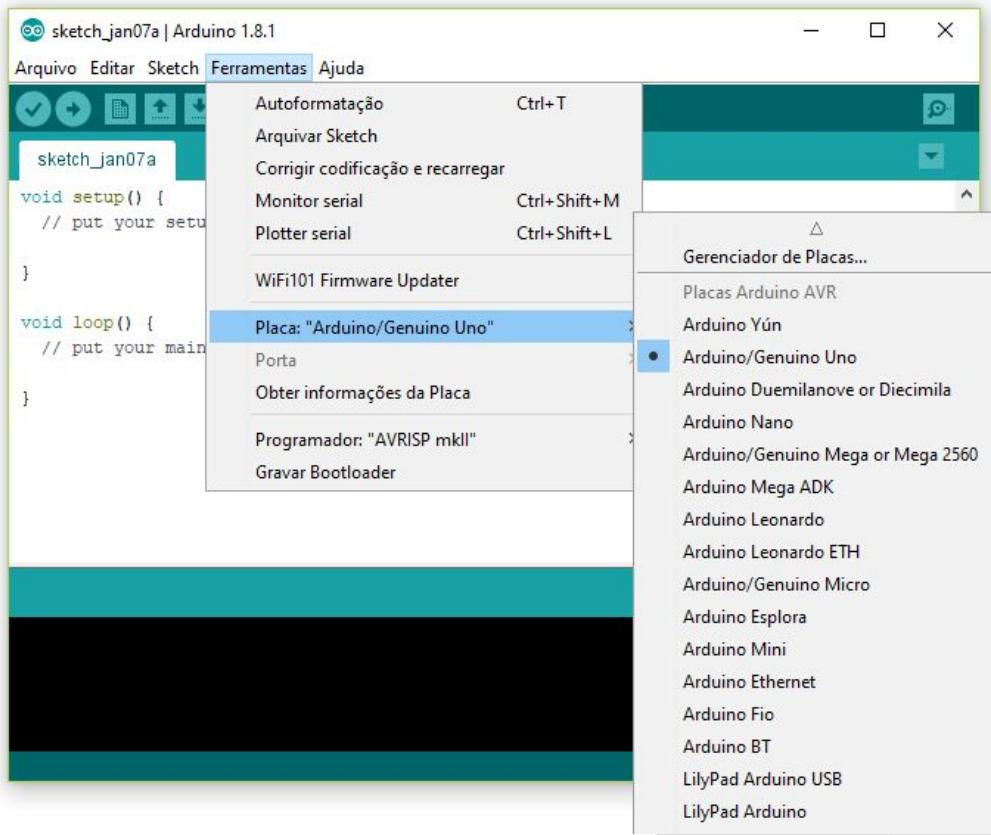
No caso do Windows, utilize o **Instalador** para que os drivers sejam instalados automaticamente

Conecte o Arduino no computador

A plataforma arduino

Utilizando o IDE do Arduino:

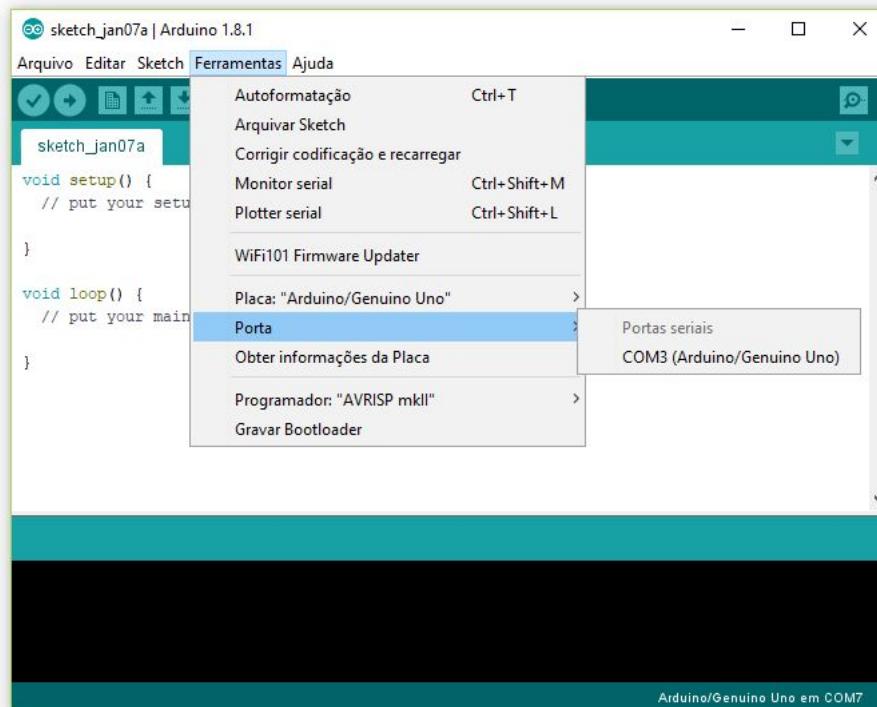
Selecionando a placa e
a porta serial



A plataforma arduino

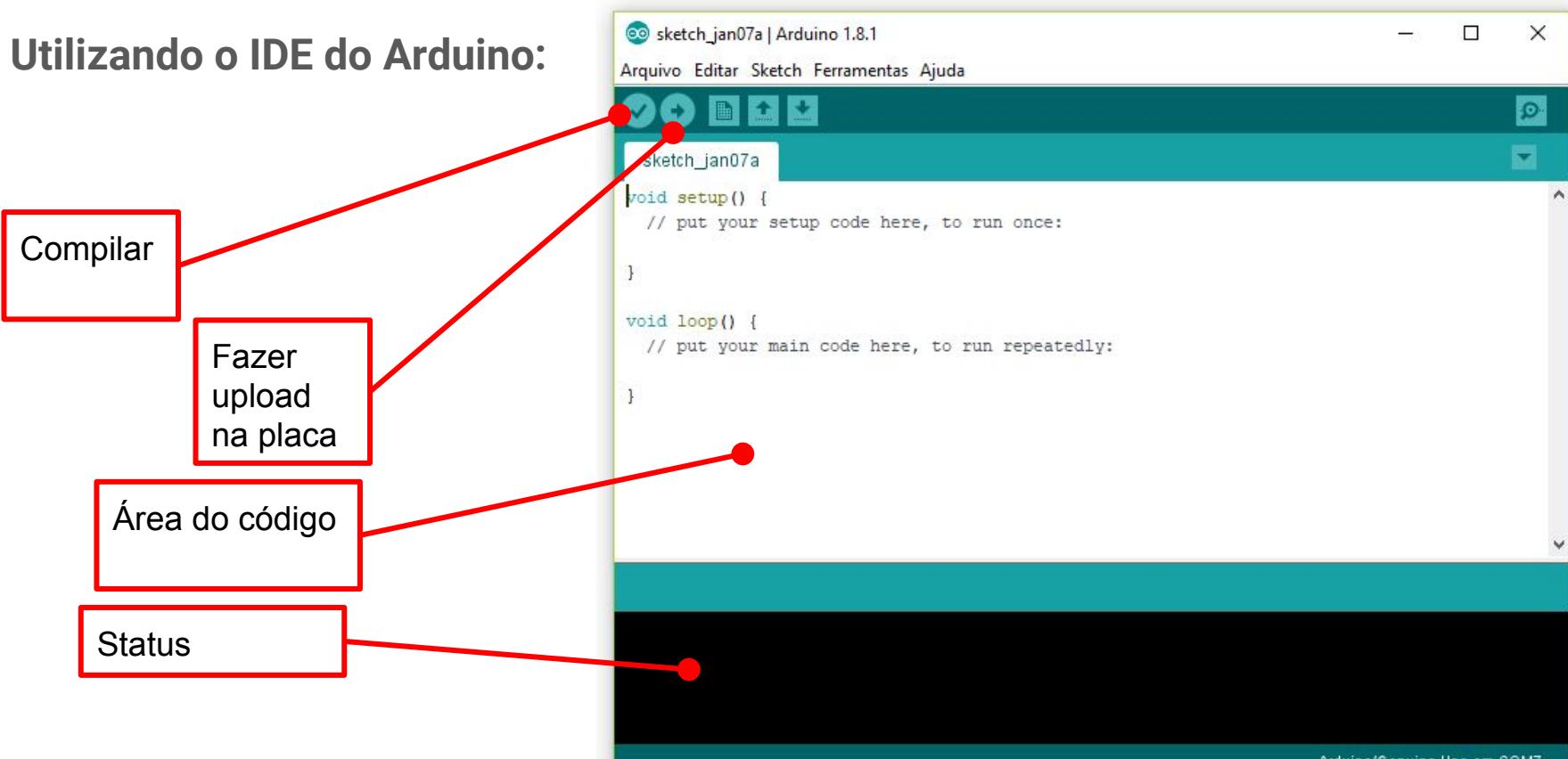
Utilizando o IDE do Arduino:

Selecionando a placa e
a porta serial



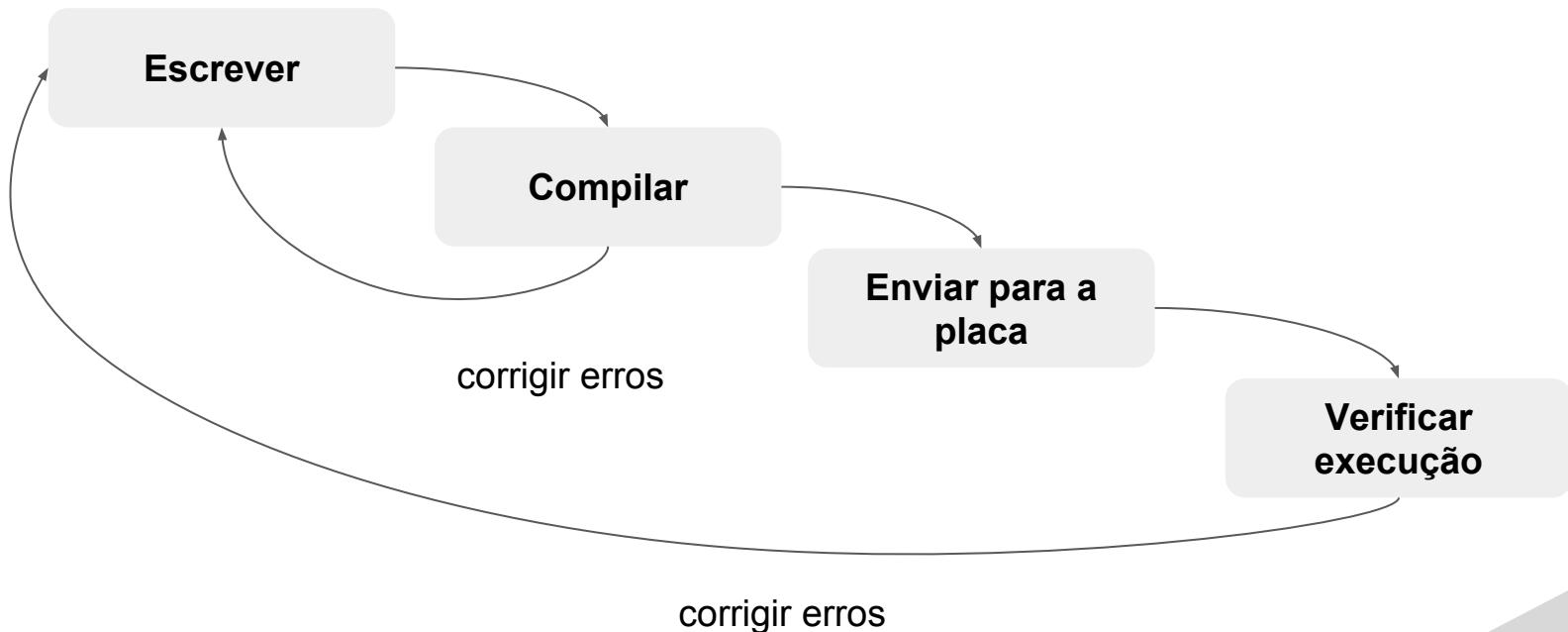
A plataforma arduino

Utilizando o IDE do Arduino:



A plataforma arduino

Workflow arduino

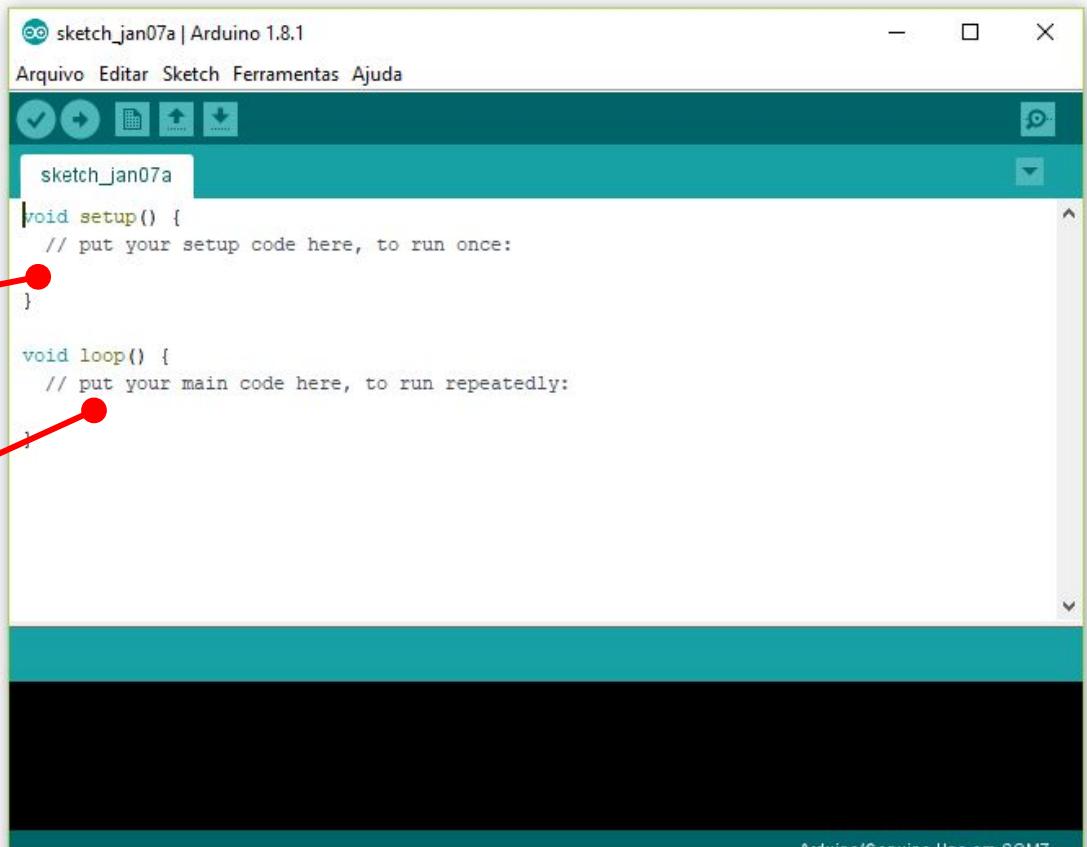


A plataforma arduino

Estrutura do código

Código executado
apenas uma vez
para configurar os
periféricos

Loop principal -
Leitura de sensores,
comunicação de
dados, etc...



The screenshot shows the Arduino IDE interface with the title bar "sketch_jan07a | Arduino 1.8.1". The menu bar includes "Arquivo", "Editar", "Sketch", "Ferramentas", and "Ajuda". Below the menu is a toolbar with icons for file operations. The main code area contains:

```
sketch_jan07a
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Two red arrows point from the explanatory text boxes to the corresponding code blocks in the IDE. One arrow points from the "Código executado apenas uma vez para configurar os periféricos" box to the `setup()` function. Another arrow points from the "Loop principal - Leitura de sensores, comunicação de dados, etc..." box to the `loop()` function.

A plataforma arduino

Alguns comandos introdutórios

pinMode(*numero_pino, modo*) - configura um pino como entrada (INPUT) ou saída (OUTPUT);

digitalWrite(*numero_pino, valor*) - desliga (LOW) ou liga (HIGH) uma saída digital;

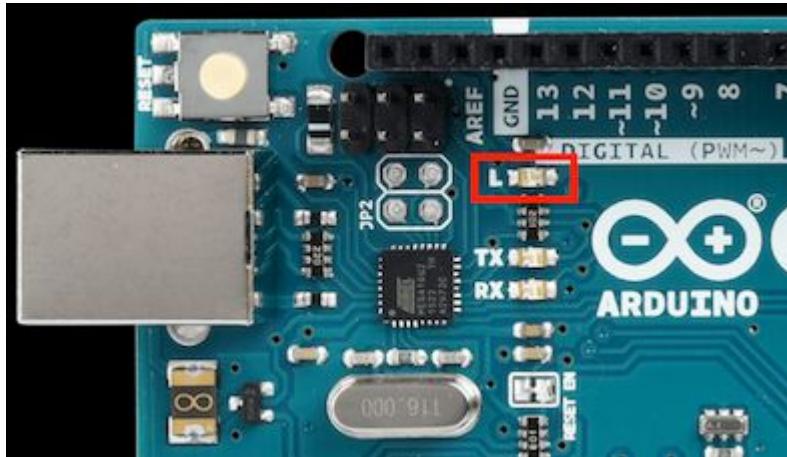
digitalRead(*numero_pino*) - lê uma entrada digital;

delay(*milissegundos*) - espera um determinado tempo em milissegundos;

A plataforma arduino

Atividade prática 1- Acender e apagar um LED

Vamos utilizar o LED built-in na placa - ligado ao pino 13 no Arduino Uno



Apresentação
de código

A plataforma arduino

Atividade prática 2 - Faça o Led piscar periodicamente

Atividade prática 3 - Faça o Led acender em meia potência

**NOW IT'S
YOUR TURN.**

A plataforma arduino

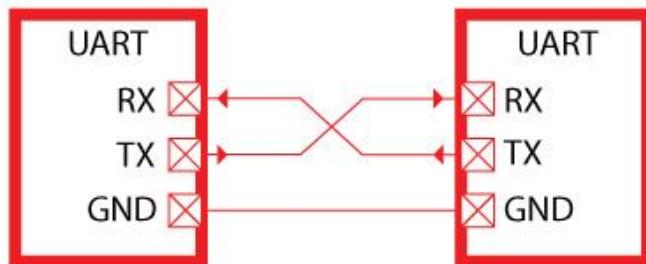
Pacote serial

O microcontrolador Atmega não possui comunicação USB. Em vez disso, ele possui uma interface serial UART (Universal Asynchronous Receiver / Transmitter):

A plataforma arduino

Comunicação serial

O microcontrolador Atmega não possui comunicação USB. Em vez disso, ele possui uma interface serial UART (Universal Asynchronous Receiver / Transmitter):



A plataforma arduino

Comunicação serial

Uma frame de informação serial possui um start bit (baixo lógico), um ou nenhum bit de paridade e um ou mais stop bits (alto lógico)

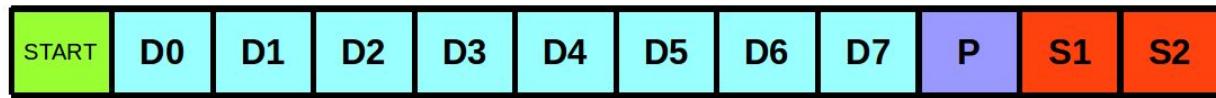
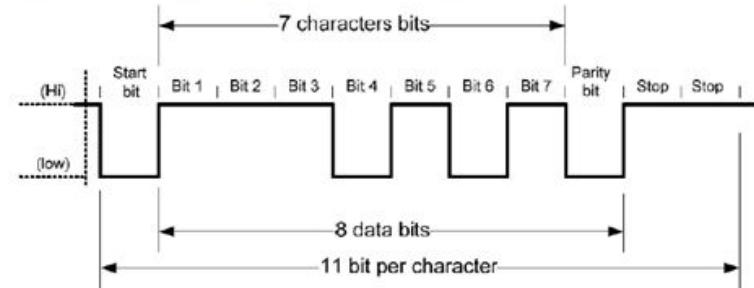


Figure 14-3. Character ASCII "W" serial transmission.



A plataforma arduino

Comunicação serial

É importante definir o **Baudrate** da comunicação, que é a taxa de bits por segundo.

Valores comumente usados:

9600

115200

etc...

A plataforma arduino

Comunicação USB no Arduino

Para que o Arduino se comunique via USB com o computador, existe um chip que converte a comunicação serial para uma comunicação USB (chip FTDI). Esta comunicação é usada para a gravação do código do Arduino e comunicação em geral com o computador



A plataforma arduino

Comunicação USB no Arduino

Para utilizar o módulo serial, utilizaremos as seguintes funções:

Serial.begin(*baudrate*) - inicia o módulo serial com um baudrate específico

Serial.print(*string*) - Envia dados pela serial

Serial.println(*string*) - Envia dados pela serial com uma quebra de linha no final

Serial.read() - Lê um caractere

Serial.available() - Verifica se existem dados a serem lidos

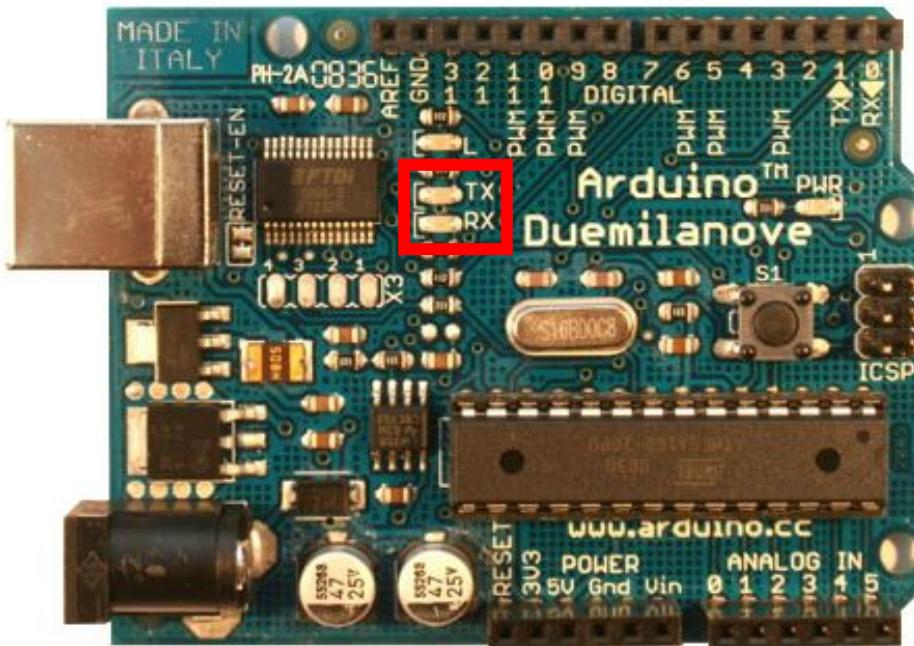
A plataforma arduino

Comunicação USB no Arduino

Leds

Tx: Dados enviados para o
PC

Rx: Dados enviados pelo
PC



A plataforma arduino

Atividades práticas

Atividade prática 4 - Enviar dados para o PC e observar no monitor serial

Atividade prática 5 - Enviar dados do PC para o arduino

A plataforma arduino

Atividade prática 6 - Faça o Led acender ou apagar a partir de um comando da serial

**NOW IT'S
YOUR TURN.**

A plataforma arduino

Lendo Strings completas

O Arduino fornece algumas opções para ler Strings completas

Serial.readString() - Lê uma sequência de caracteres até um certo *timeout*

Serial.setTimeout(time)- Configura o timeout para a função readString

Serial.readStringUntil(char)- Lê a string até um determinado caractere

A plataforma arduino

Manipulação de Strings completas

Criando uma nova String

`String s = String(char_seq); ou String s = "algo";`

string.substring(init, end) - Retorna um pedaço da string original

string.toInt() - Converte a String para um número inteiro

string.toFloat()- Converte a String para um número em ponto flutuante

strlen - Tamanho de um char array

A plataforma arduino

Conversão de dados em cadeias de caracteres

Podemos utilizar arrays de chars em vez de Strings

atoi() - Converte a String para um número inteiro

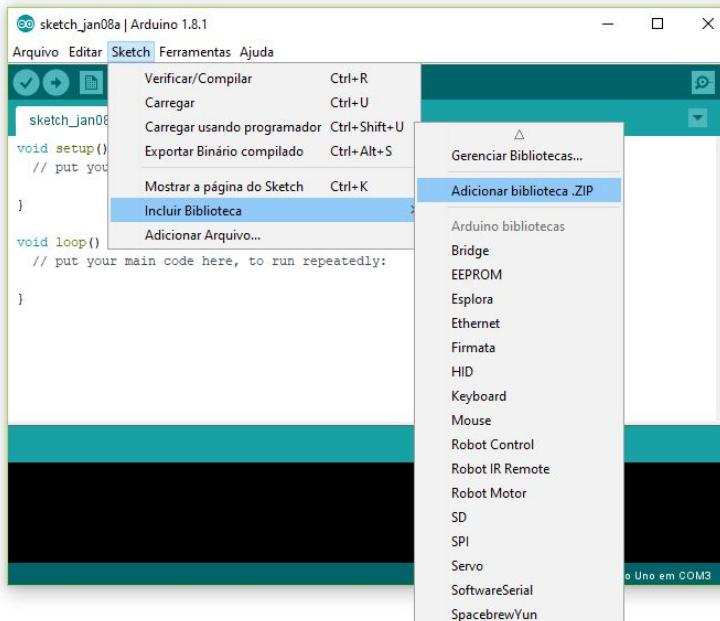
atof()- Converte a String para um número em ponto flutuante

A plataforma arduino

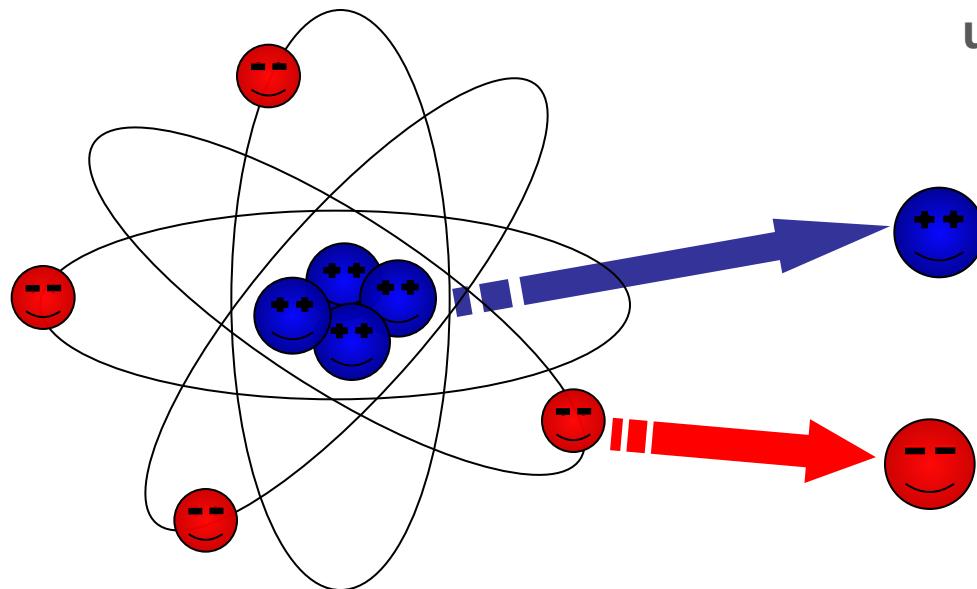
Bibliotecas do Arduino

Algumas libs podem ser encontradas em:

<https://www.arduino.cc/en/Reference/Libraries>



Fundamentos de eletrônica



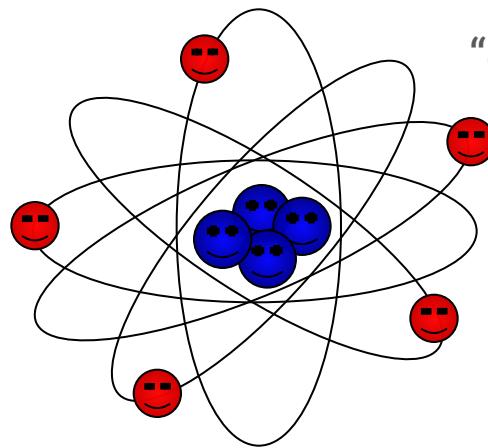
universo formado de átomos

prótons: cargas positivas

elétrons: cargas negativas

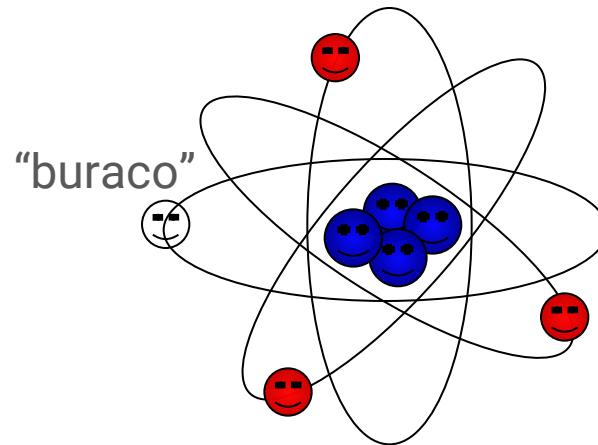
eletroscopia - interação entre partículas atômicas

Fundamentos de eletrônica



“elétron extra”

Átomos com mais elétrons que prótons estão carregados negativamente (íon negativo)



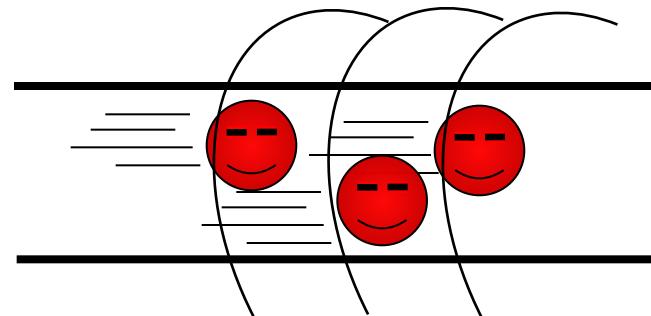
“buraco”

Átomos com menos elétrons que prótons estão carregados positivamente (íon positivo)

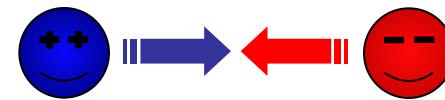
Fundamentos de eletrônica



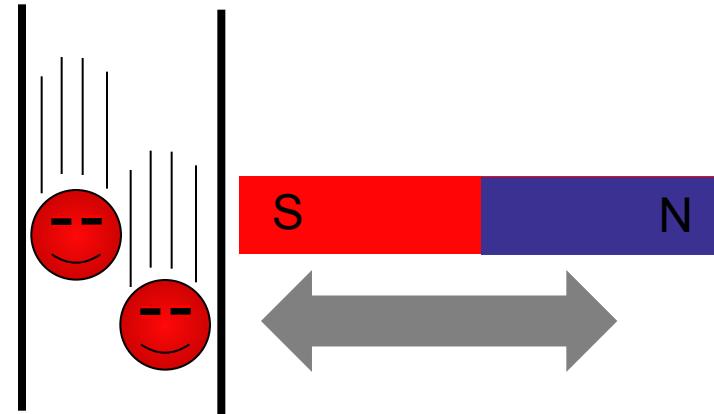
cargas iguais se repelem



cargas em movimento
geram campo magnético

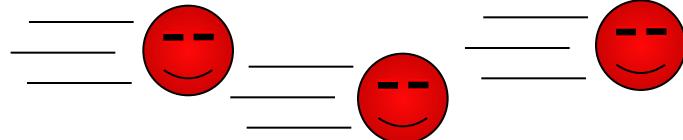


cargas opostas se atraem



campo magnético em movimento
gera corrente elétrica

Fundamentos de eletrônica



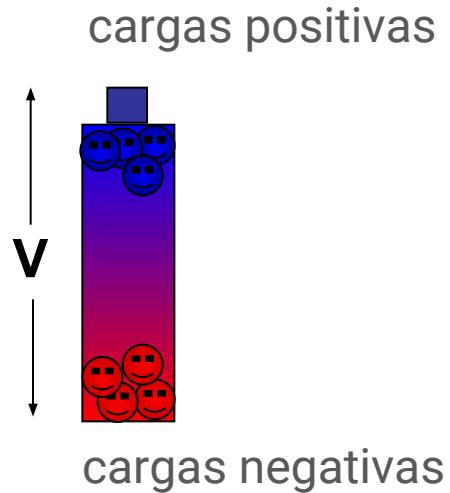
condutor – permite o fluxo de elétrons



isolante – evita a passagem de elétrons

Fundamentos de eletrônica

diferença de potencial
ou tensão.

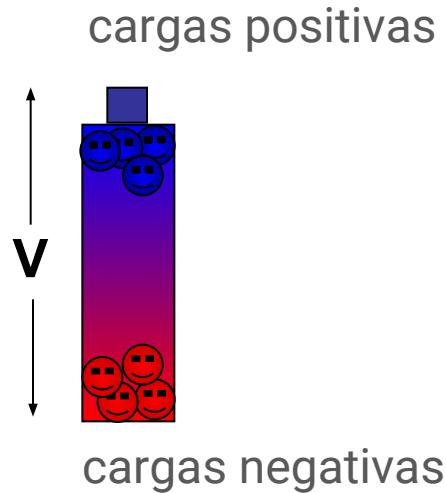


quanto maior a tensão, mais “força” tem os elétrons

Tensão elétrica é medida em **Volts**. Quando lidamos com o Arduino, utilizaremos principalmente fontes de 5 volts

Fundamentos de eletrônica

diferença de potencial
ou tensão.

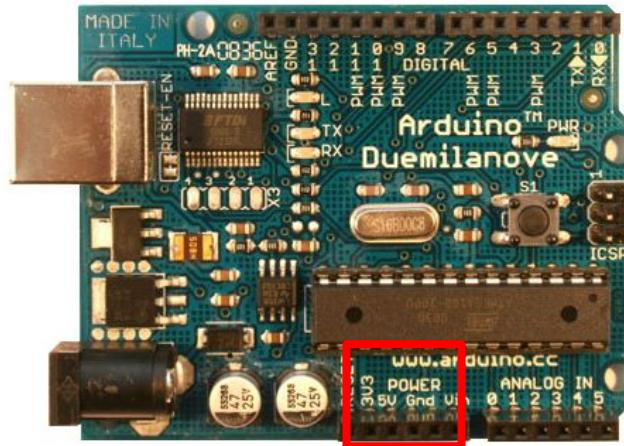


Os pinos do Arduino também
não suportam tensões
maiores que 5V

quanto maior a tensão, mais “força” tem os elétrons

Fundamentos de eletrônica

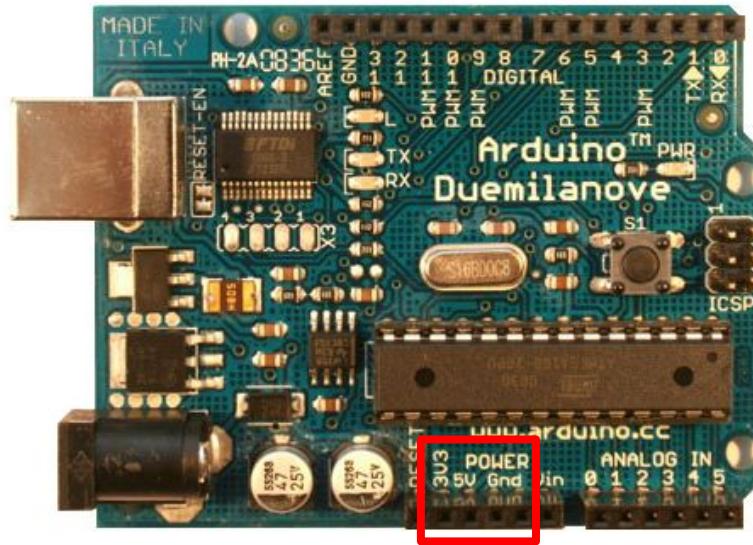
Exemplo de fontes de tensão



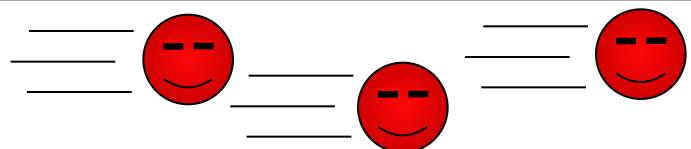
Fundamentos de eletrônica

Terminologia

Muitas vezes em eletrônica, o terminal que recebe a tensão positiva é referido como VCC e o terminal que recebe (Ground)

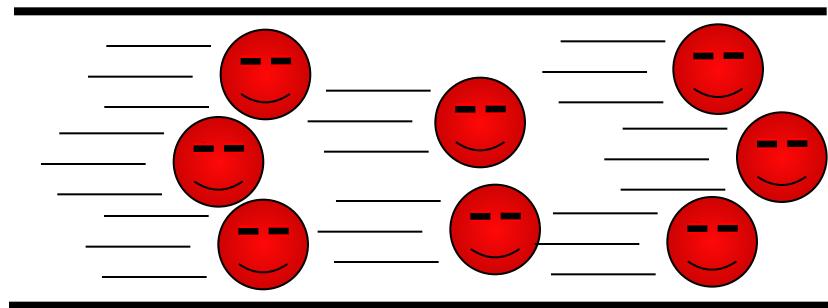


Fundamentos de eletrônica - corrente



fluxo de elétrons em um condutor

Corrente elétrica é medida em Amperes. Ao lidarmos com arduino, utilizamos em geral, correntes pequenas de até 1 A



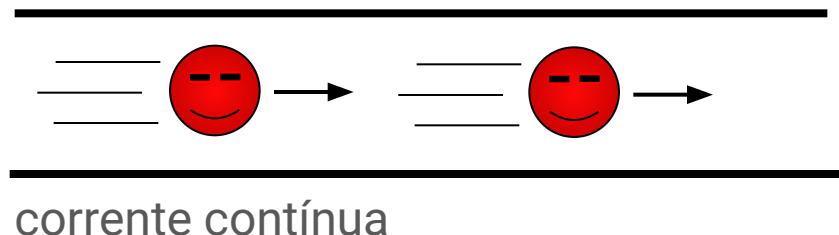
quanto maior a corrente, maior a “quantidade” de elétrons

Fundamentos de eletrônica - corrente

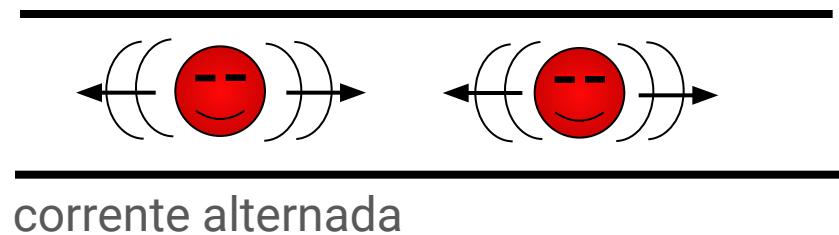
Exemplo: USB

Uma porta USB 2.0 é capaz de fornecer uma tensão de 5V para a alimentação de circuitos e uma corrente de até 500 mA. O USB 3.0 pode chegar a até 900 mA. Tendo em vista tais fatos, ao usar o Arduino na porta do computador, evite circuitos que consumam mais de 500 mA.

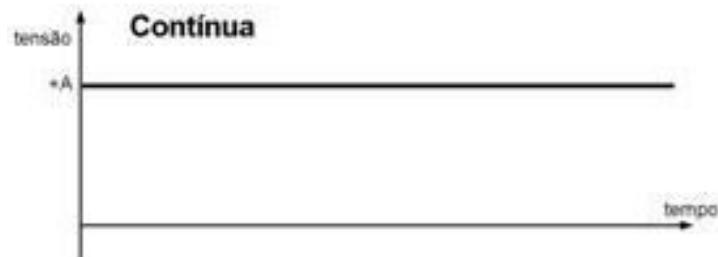
Fundamentos de eletrônica - corrente



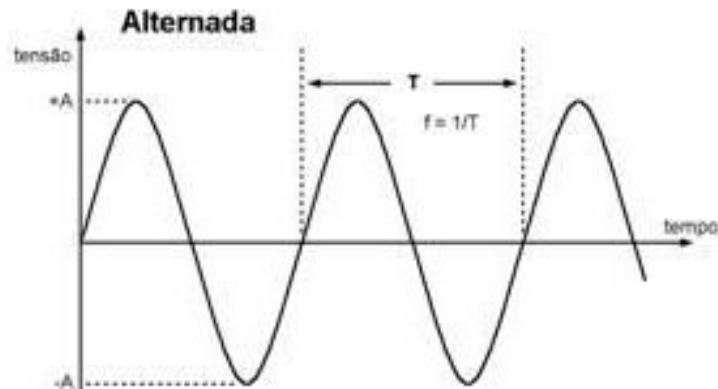
Placas de Arduino são circuitos de **corrente contínua**. Tensões alternadas são bastante úteis para a transmissão de energia pois pode-se converter níveis de tensão facilmente através de **transformadores**.



Fundamentos de eletrônica - corrente



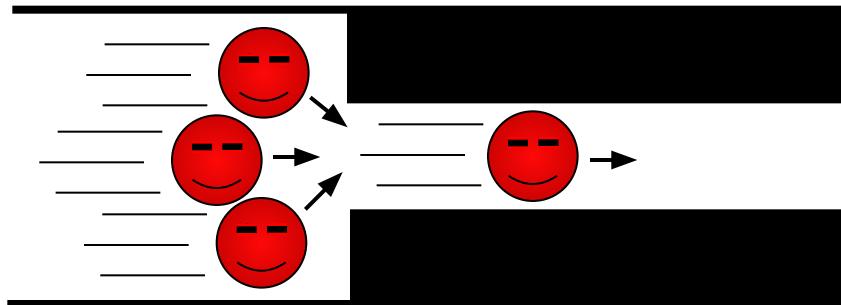
mesma polaridade no tempo (sentido contínuo)



inversão de polaridade no tempo

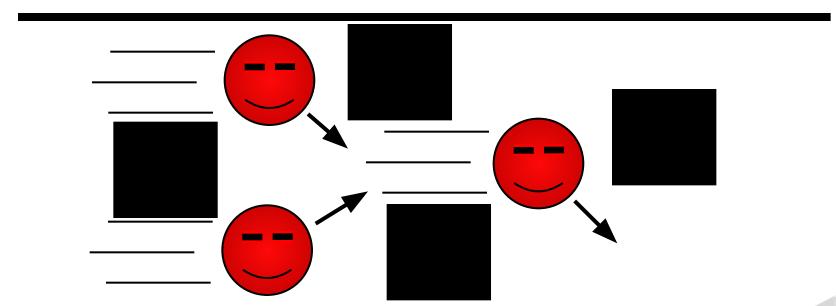


Fundamentos de eletrônica - resistência elétrica



Resistências elétricas são medidas em ohms

propriedade do material condutor em reduzir a passagem dos elétrons



elétrons “se acumulam e batem” no condutor, “dissipando” sua energia (gerando calor)

Fundamentos de eletrônica - tipos de resistores

tipos:

carvão
[carbono]
filme
fio



resistência:

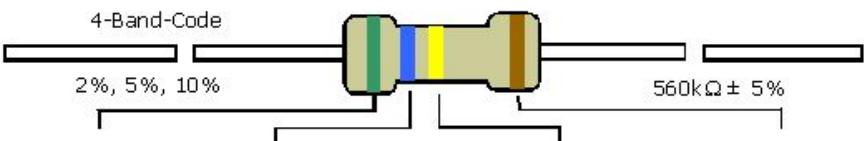
fixo
variável

transformam energia elétrica em energia térmica
[pode ser usado como atuador]

Fundamentos de eletrônica - tipos de resistores

valores expressos em ohms

o corpo dos resistores possui um código de cores para identificar o valor



4-Band-Code

2%, 5%, 10%

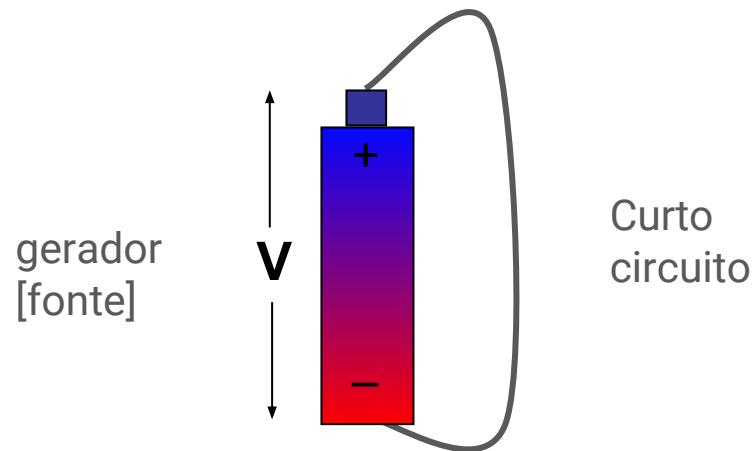
$560\text{k}\Omega \pm 5\%$

COLOR	1st BAND	2nd BAND	3rd BAND	MULTIPLIER	TOLERANCE	
Black	0	0	0	1Ω		
Brown	1	1	1	10Ω	$\pm 1\%$	(F)
Red	2	2	2	100Ω	$\pm 2\%$	(G)
Orange	3	3	3	$1\text{K}\Omega$		
Yellow	4	4	4	$10\text{K}\Omega$		
Green	5	5	5	$100\text{K}\Omega$	$\pm 0.5\%$	(D)
Blue	6	6	6	$1\text{M}\Omega$	$\pm 0.25\%$	(C)
Violet	7	7	7	$10\text{M}\Omega$	$\pm 0.10\%$	(B)
Grey	8	8	8		$\pm 0.05\%$	
White	9	9	9			
Gold				0.1	$\pm 5\%$	(J)
Silver				0.01	$\pm 10\%$	(K)

Fundamentos de eletrônica - resistência elétrica

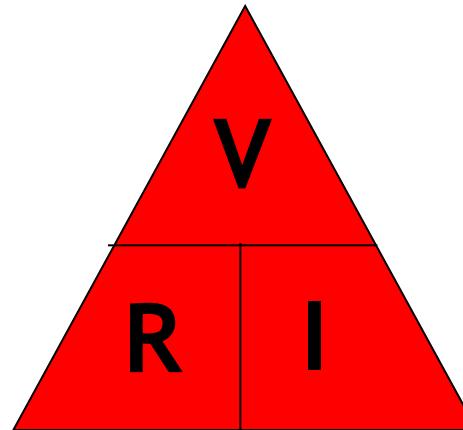
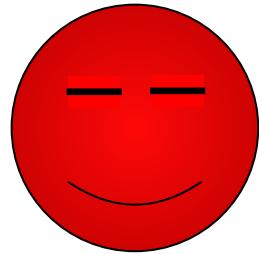
Importância das resistências elétricas

Resistências elétricas funcionam para limitar a corrente elétrica fornecida por uma fonte de tensão. Ligar os terminais de uma bateria ou fonte sem nenhuma resistência causa um **curto circuito**



Fundamentos de eletrônica - lei de ohm

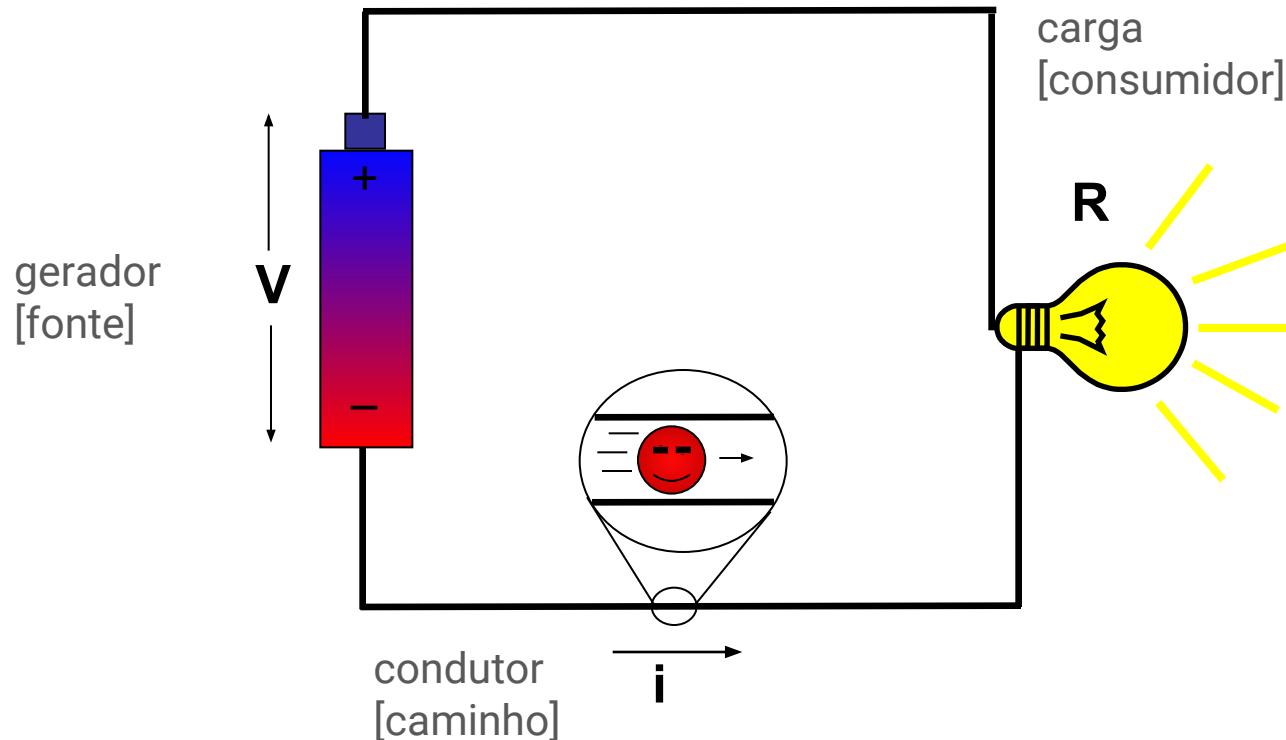
$$V = R \times I$$



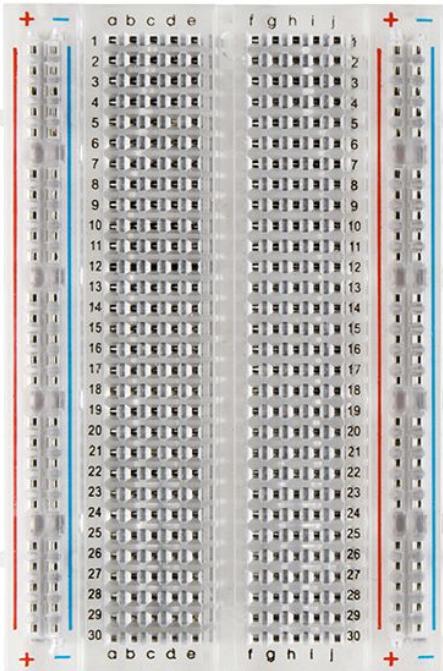
$$R = V/I$$
$$I = V/R$$

a diferença de potencial (V) entre dois pontos de um condutor é proporcional à corrente elétrica (I) que o percorre e à sua resistência (R)

Fundamentos de eletrônica - circuito elétrico



Fundamentos de eletrônica - montagem de

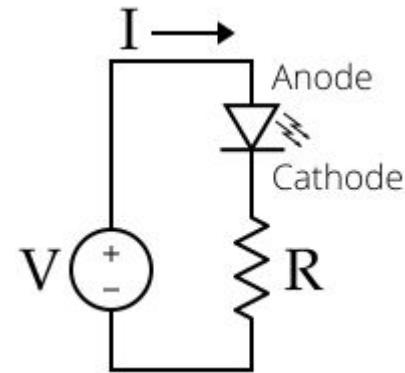


As trilhas laterais são para alimentação e estão conectadas verticalmente, já as trilhas do centro são para construir o circuito e estão ligadas horizontalmente

Fundamentos de eletrônica - montagem de circuitos

Protopboard

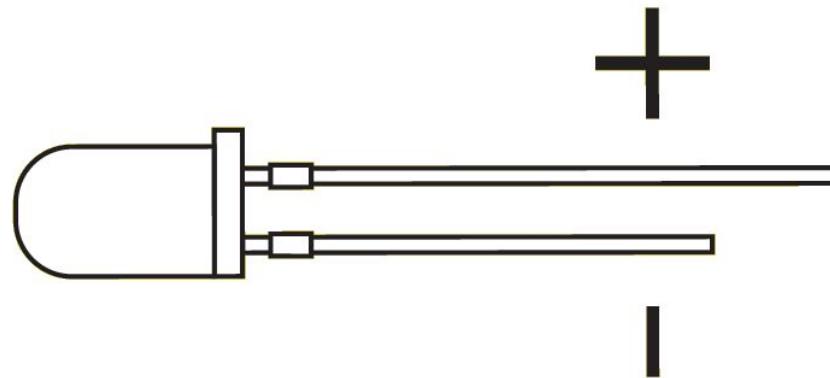
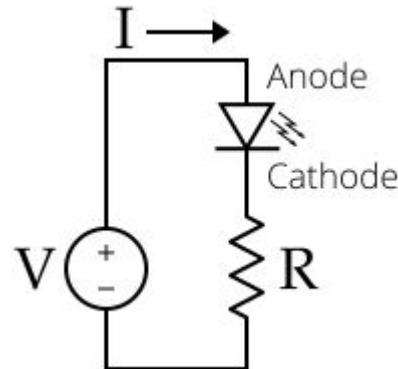
Exemplo: Acendendo um LED



Fundamentos de eletrônica - montagem de circuitos

Protopboard

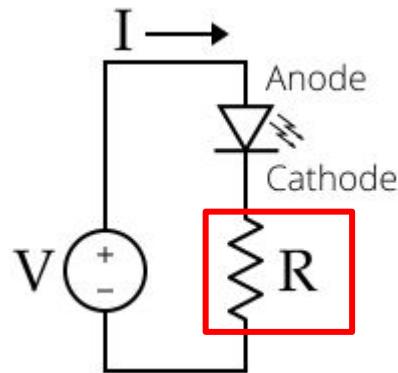
LEDs são diodos, e só deixam a corrente passar em um sentido



Fundamentos de eletrônica - montagem de circuitos

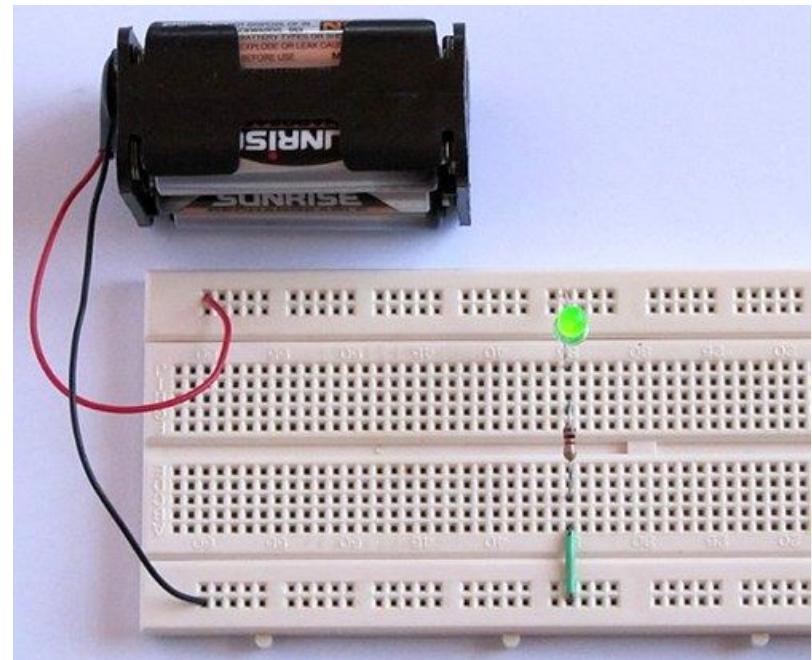
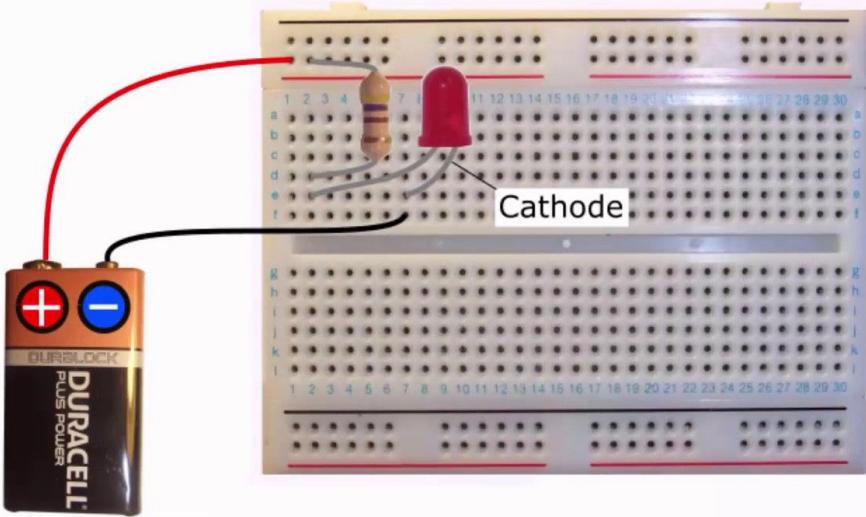
Protopboard

LEDs possuem um limite de corrente que pode passar eles sem causar danos. Para garantir isto, precisamos inserir um resistor (da ordem de 100 ohms é suficiente para 5V)



Fundamentos de eletrônica - montagem de circuitos

Protopboard



Fundamentos de eletrônica - montagem de circuitos

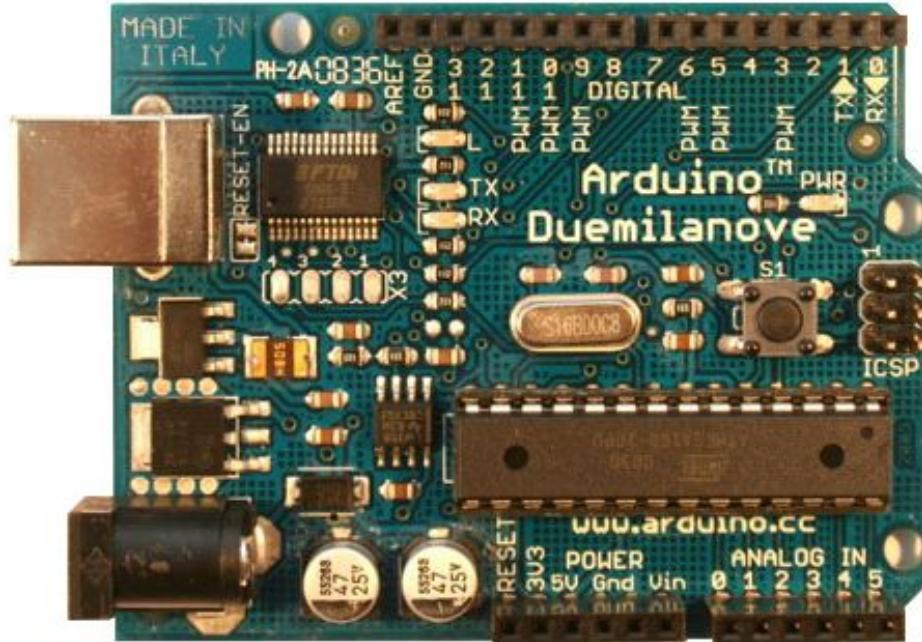
Protopboard

Atividade prática 8 - Ligue um LED com a alimentação do Arduino

Atividade prática 9 - Refaça a atividade de piscar o LED com um LED externo

**NOW IT'S
YOUR TURN.**

Voltando ao Arduino....



Vamos aprender agora
mais funções da placa
Arduino e da linguagem
Wiring

Sensores

Existem dois tipos de sensores:

Digitais: Tem um número limitado de estados, geralmente dois
(Ex: botões, sensores de barreira, sensores de presença)

Analógicos: Possuem um número grande (ou infinito) de possíveis estados (Ex: sensor de temperatura, pressão, etc...)

Sensores

Leitura digital

É usada quando o sensor tem apenas dois estados.

`digitalRead()` - lê um pino de entrada

The screenshot shows the Arduino IDE interface. The title bar reads "sketch_jan07a | Arduino 1.8.1". The menu bar includes "Arquivo", "Editar", "Sketch", "Ferramentas", and "Ajuda". The toolbar contains icons for file operations like Open, Save, and Print. The code editor window displays the following sketch:

```
int pin_number = 8;
void setup() {
    // Configuramos o pino como leitura digital
    pinMode(pin_number, INPUT);

    //Configuração da serial
    Serial.begin(9600);
}

void loop() {
    int digital_reading = digitalRead(pin_number);
    Serial.println(digital_reading);

    delay(100);
}
```

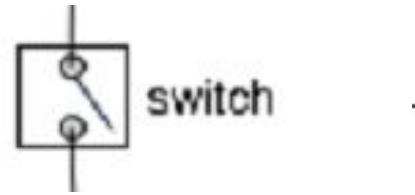
Sensores

Botões

Interrompe a passagem da corrente elétrica

Liga/desliga o circuito

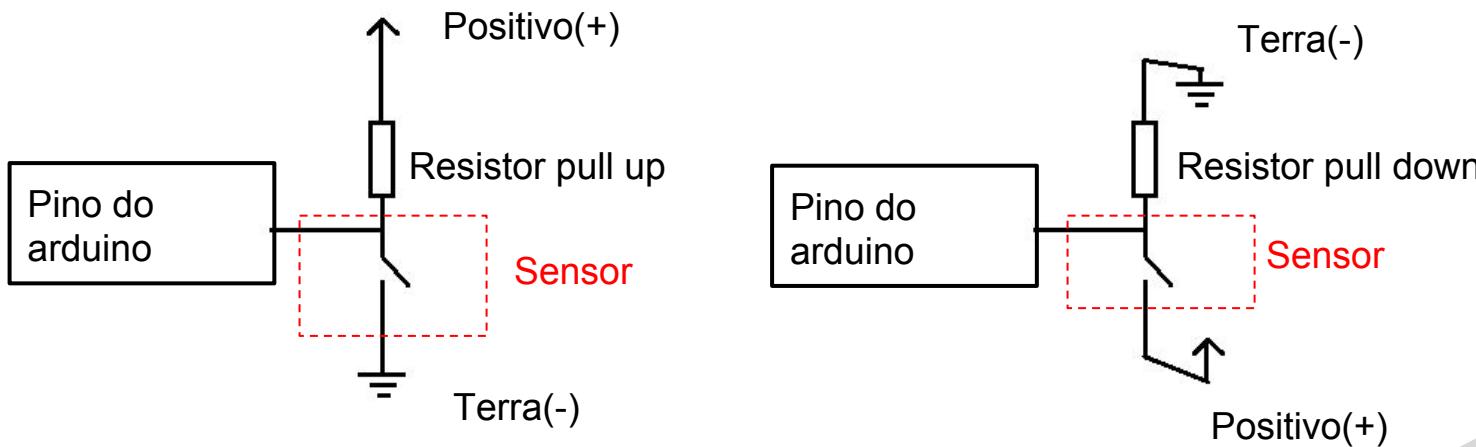
Sensor de toque



Sensores

Como ler o sinal dos sensores

Ligaçāo: Em geral, sensores desse tipo funcionam por contato:



Sensores

Como ler o sinal dos sensores

Ligaçāo: Em geral, sensores desse tipo funcionam por contato:

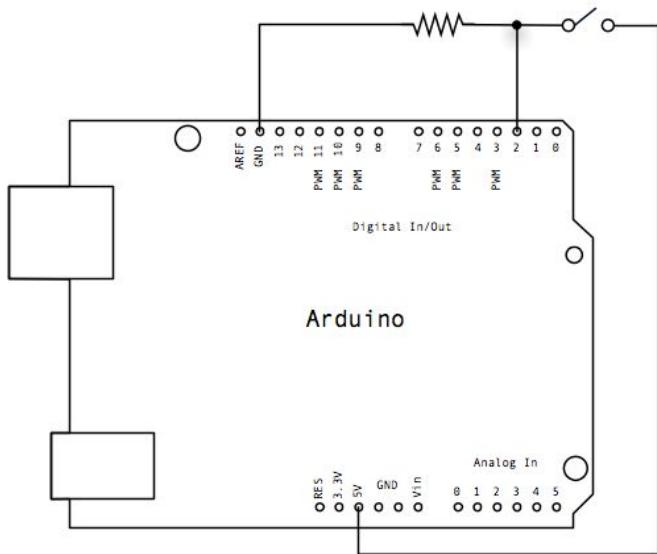


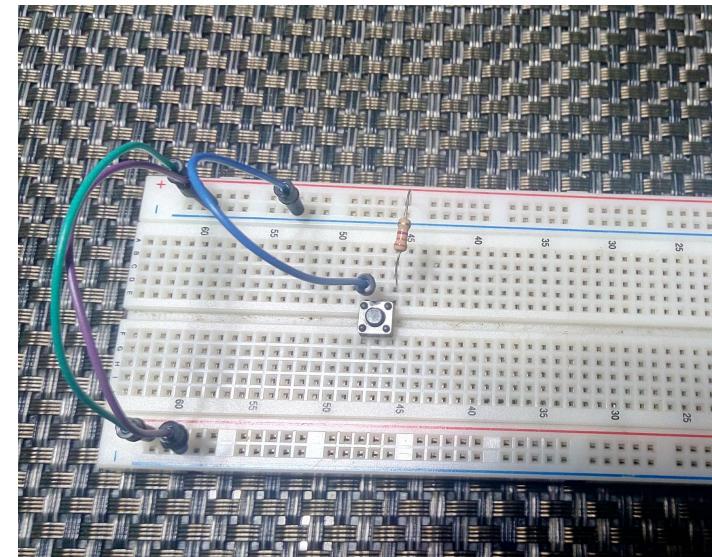
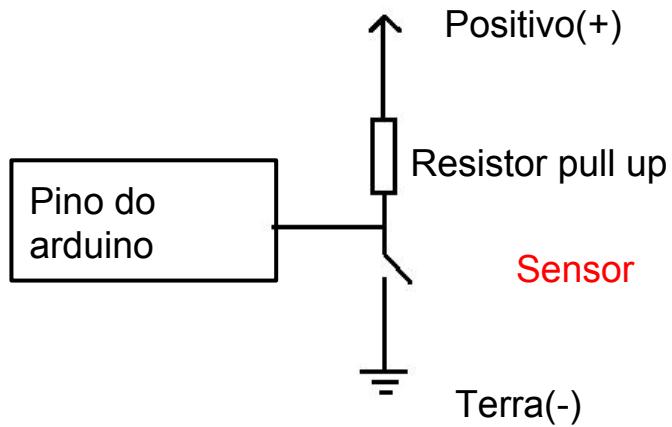
Figura retirada de <http://arduino.cc/>

Sensores

Atividade prática 10 - Fazer um circuito para ler 1 botão.

Quando o botão for apertado. O Arduino deve enviar via serial a palavra APERTOU.

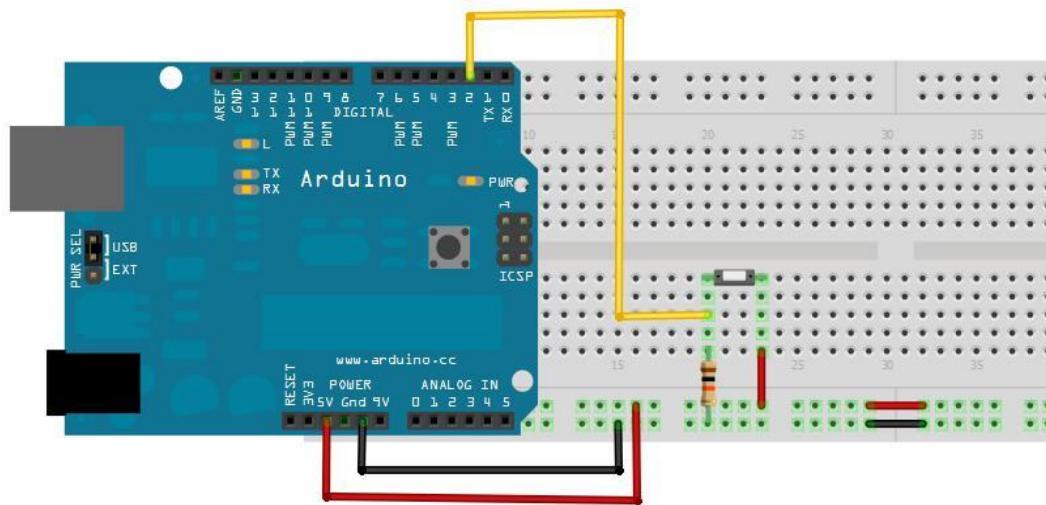
**NOW IT'S
YOUR TURN.**



Sensores

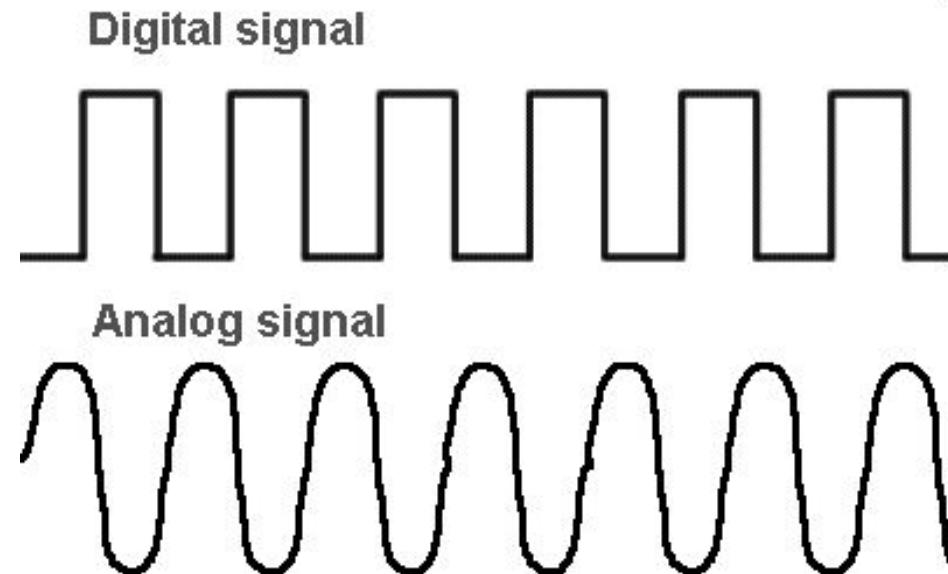
Atividade prática 11 - Fazer um circuito para ler 1 botão.

Quando o botão for apertado. O Arduino deve enviar via serial a palavra APERTOU.



Sensores

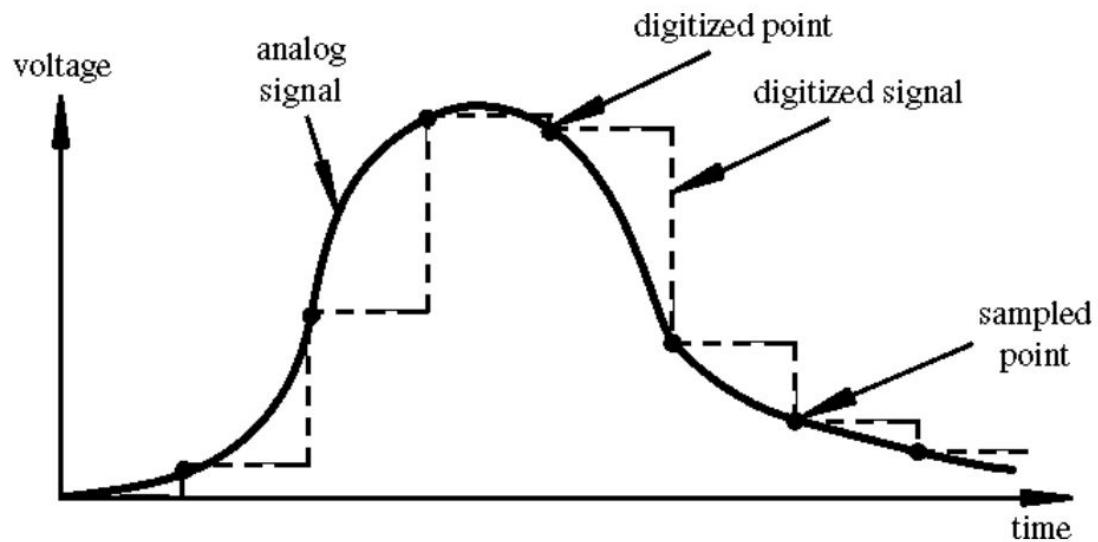
Leitura analógica



sinal com variação
discreta (valores
pré-definidos)

Sensores

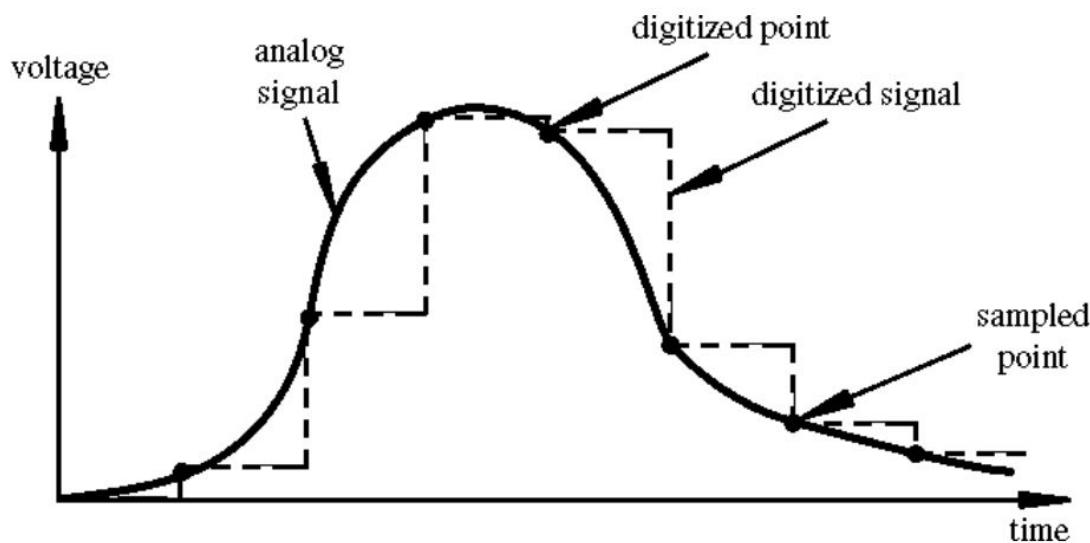
Conversão analógica - digital



valor é lido em intervalos regulares de tempo e transformado em um número digital

Sensores

Conversão analógica - digital



O microcontrolador converte um sinal de tensão em um número inteiro de x bits. No arduino, temos 10 bits

0V é codificado como 0
(00000...0)

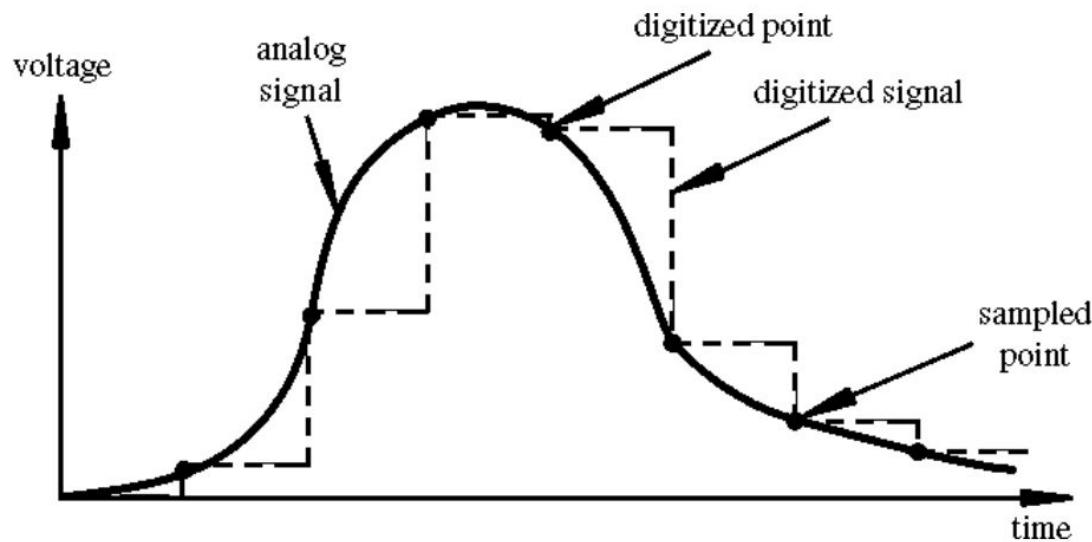
5V é codificado como
 $2^{10} - 1 = (111\dots1)$

1.3V será codificado como qual valor?

$$(1.3 / 5) * 1023 = 266$$

Sensores

Conversão analógica - digital



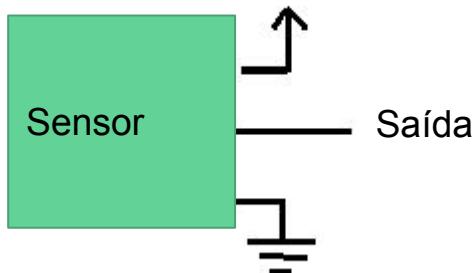
A menor diferença de tensão percebida é $5/1024 = 0,0048$. Logo, 1V e 1,003V serão codificados da mesma forma! Quanto mais bits tiver um ADC, mais preciso ele é

Sensores

Conversão analógica - como ler o sinal dos sensores?

Algumas situações comuns

Sensores que apenas necessitam de alimentação

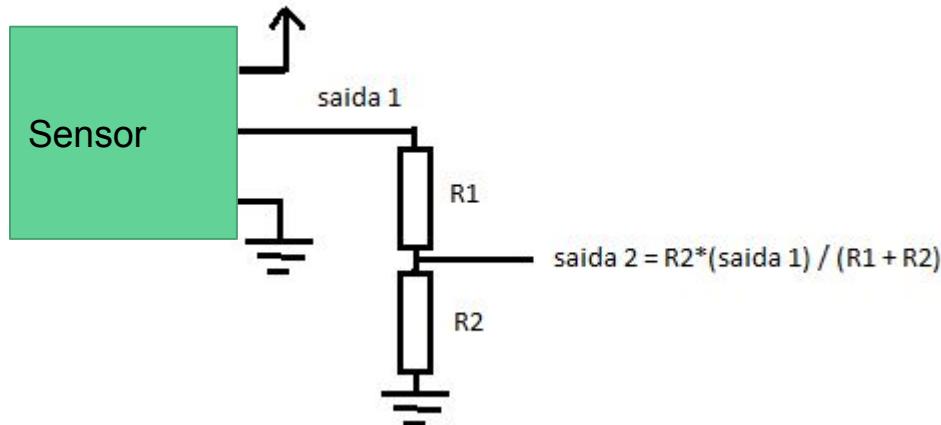


E se a saída do sensor for maior do que o máximo que o meu micro-controlador aguenta?

Sensores

Conversão analógica - como ler o sinal dos sensores?

Divisor de tensão



Se o sensor for para 5V e o microcontrolador for 3.3V, queremos que o termo $R2/(R1 + R2) = 3.3/5$

Logo, podemos tomar $R2 = 3.3k$
 $R1 = 1.7k$.

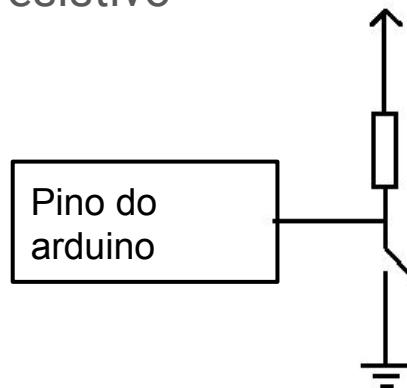
Assim, quando a saída 1 for 5V, a saída 2 vai ser 3.3V

Sensores

Conversão analógica - como ler o sinal dos sensores?

Algumas situações comuns:

Sensor resistivo



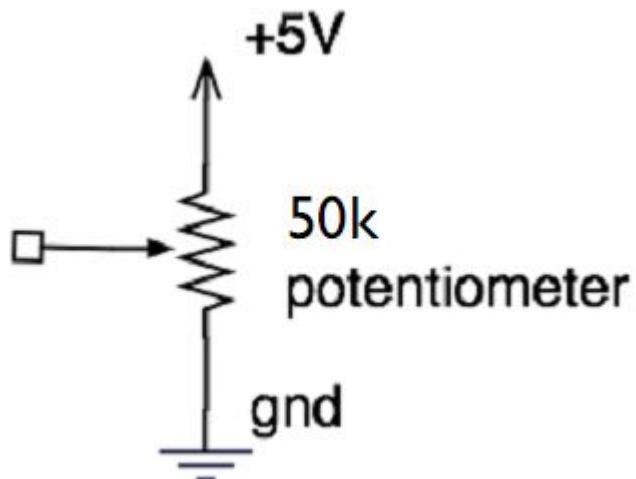
Mesmo funcionamento do divisor de tensão!

Sensores

Conversão analógica - como ler o sinal dos sensores?

Algumas situações comuns:

Resistência variável



Sensores

Lendo valores analógicos no Arduino

`analogRead()` - lê um pino de entrada



sketch_jan07a | Arduino 1.8.1

Arquivo Editar Sketch Ferramentas Ajuda

```
sketch_jan07a §

int analog_number = 0;
void setup() {

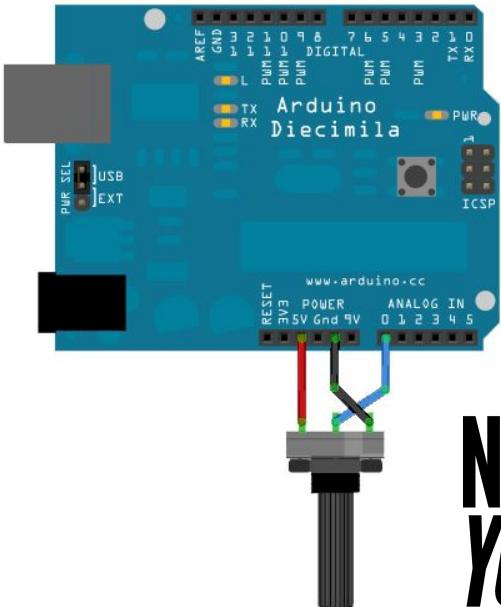
    //Configuração da serial
    Serial.begin(9600);
}

void loop() {
    int analog_reading = analogRead(analog_number);
    Serial.println(analog_reaging);

    delay(100);
}
```

Sensores

Atividade prática 12 - Criar um circuito que lê a tensão de um potenciômetro e



NOW IT'S
YOUR TURN.

sketch_jan07a | Arduino 1.8.1

Arquivo Editar Sketch Ferramentas Ajuda

```
sketch_jan07a §

int analog_number = 0;
void setup() {

    //Configuração da serial
    Serial.begin(9600);
}

void loop() {
    int analog_reading = analogRead(analog_number);
    Serial.println(analog_reaging);

    delay(100);
}
```

Sensores

Calibração de sensores

Caso da linha reta.

É necessário criar uma reta de conversão entre a saída do ADC e a grandeza que se deseja medir.

$$\text{saída} = a \cdot \text{adc} + b.$$

$$\begin{aligned}\text{valor_1} &= a \cdot \text{adc_1} + b \\ \text{valor_2} &= a \cdot \text{adc_2} + b\end{aligned}$$

$$a = (\text{valor_2} - \text{valor_1}) / (\text{adc_2} - \text{adc_1}) \text{ e } b = \text{valor_1} - a \cdot \text{adc_1}$$

Método simplificado!
O “cientificamente” correto
seria fazer regressão linear
+ modelamento de ruído

Sensores

Calibração de sensores

Calibração - caso da linha reta.

Ex: um sensor de temperatura, a 20 C dá como saída 234 e a 50 C dá como saída 764.

Qual é a temperatura quando a saída é 300?

valor_1 = 20, adc_1 = 234, valor_2 = 50, adc_2 = 764.

Logo: $a = (\text{valor}_2 - \text{valor}_1)/(\text{adc}_2 - \text{adc}_1) = 0.0566$ e
 $b = \text{valor}_1 - a * \text{adc}_1 = 6.755$. Logo

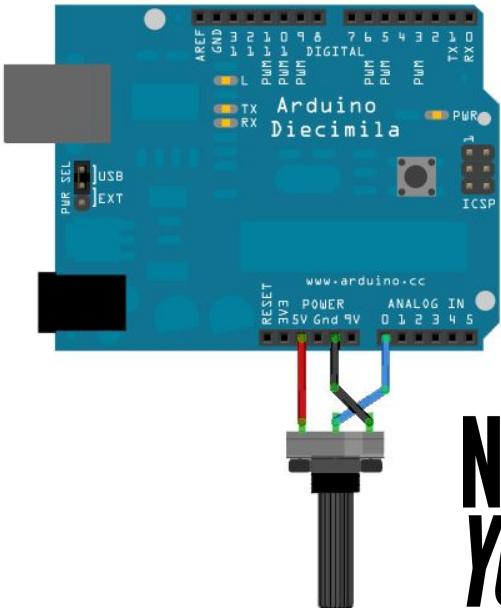
$\text{saída} = 0.0566 * \text{adc} + 6.755$.

Quando o $\text{adc} = 300$, $\text{saída} = 0.0566 * 300 + 6.755 = 23.735$ C

Método simplificado!
O “cientificamente” correto
seria fazer regressão linear
+ modelamento de ruído

Sensores

Atividade prática 13 - Criar um circuito que informa o ângulo do potenciômetro



sketch_jan07a | Arduino 1.8.1

Arquivo Editar Sketch Ferramentas Ajuda

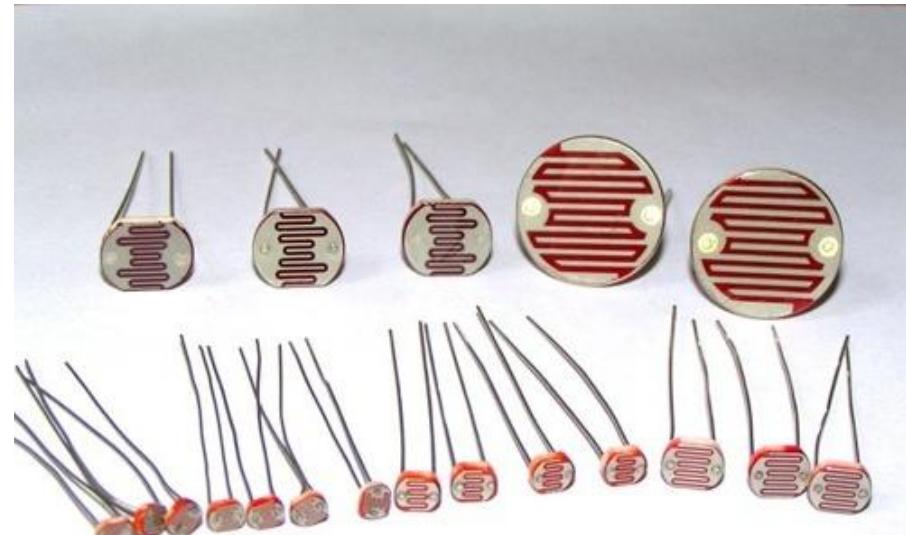
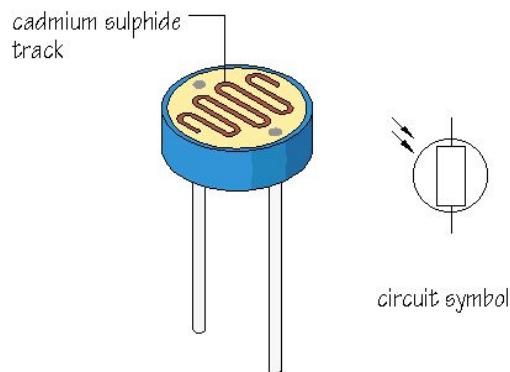


sketch_jan07a §

```
int analog_number = 0;  
void setup() {  
  
    //Configuração da serial  
    Serial.begin(9600);  
}  
  
void loop() {  
    int analog_reading = analogRead(analog_number);  
    Serial.println(analog_reading);  
  
    delay(100);  
}
```

Sensores

Exemplo de sensor resistivo - LDR

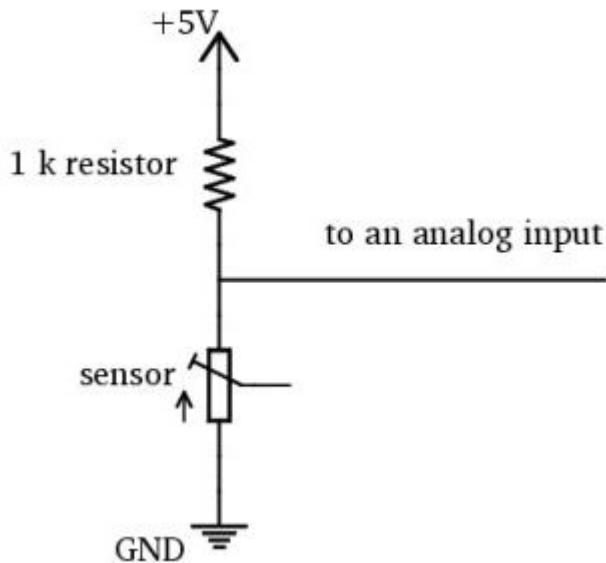


Sensores

Resistor variável sensível à luz

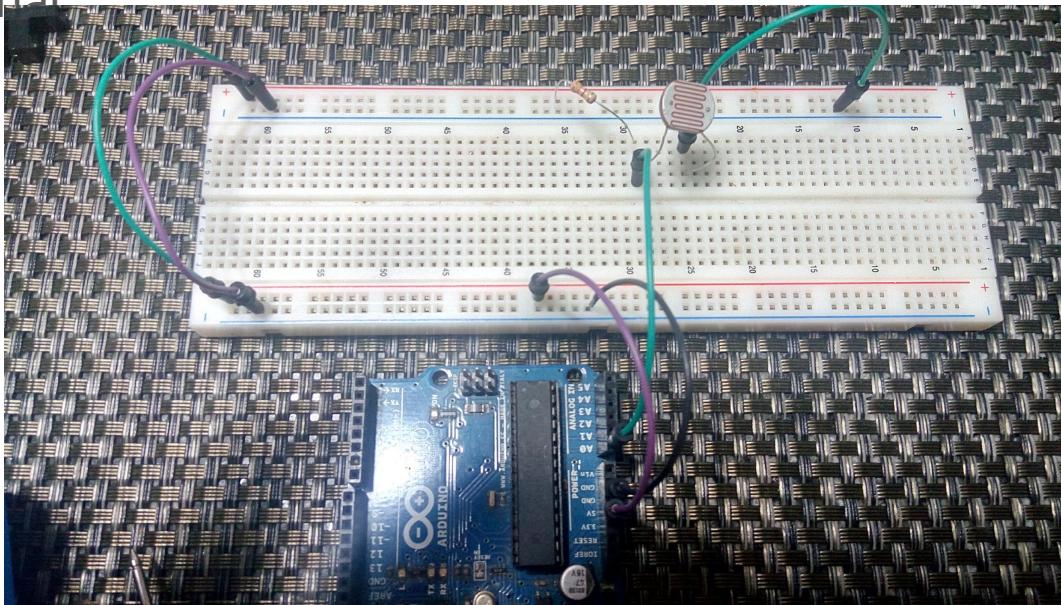
circuito para arduino

porque o resistor de 1k?
para limitar a corrente
se o LDR assumir valores
muito baixos e para formar o
divisor de tensão



Sensores

Atividade prática 14 - Criar um circuito que lê a tensão de um LDR e envia para uma serial.



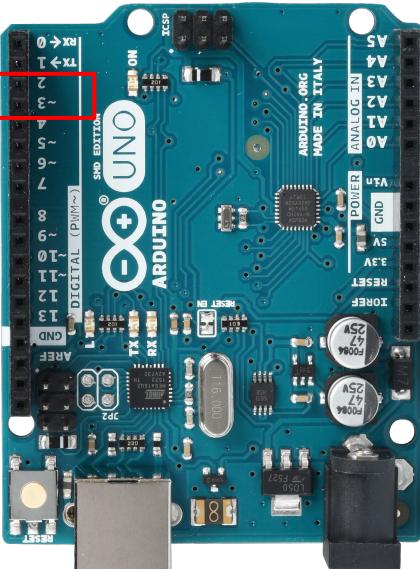
Sensores

Sensores que enviam pulsos

Caso especial dos sensores digitais, mas aqui estamos interessados em pulsos

Ex: Sensor de fluxo

Aqui, utilizamos interrupções para não perder os pulsos



sketch_jan07a | Arduino 1.8.1

Arquivo Editar Sketch Ferramentas Ajuda

sketch_jan07a §

```
//Sempre declare variaveis alteradas por interrupcoes
//Como volatile
volatile int num_pulses = 0;
int interrupt_pin = 2;
void pulse_routine(){
    num_pulses++;
}

void setup() {
    pinMode(interrupt_pin, INPUT);
    attachInterrupt(0, pulse_routine, RISING);
    //Configuração da serial
    Serial.begin(9600);
}

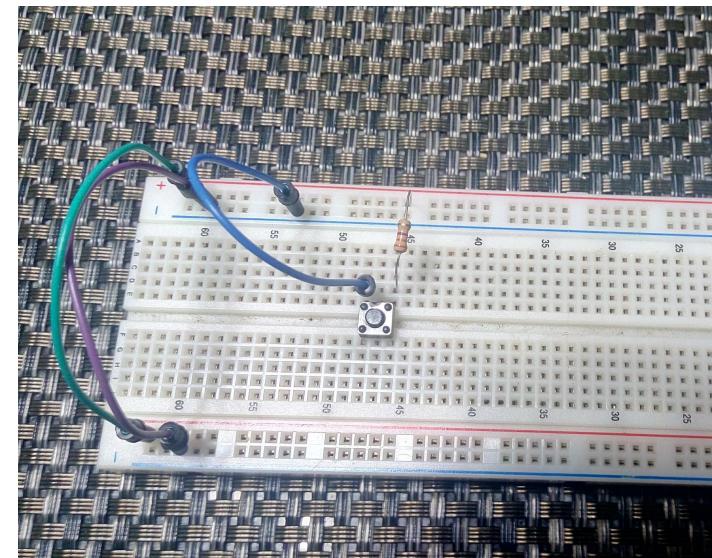
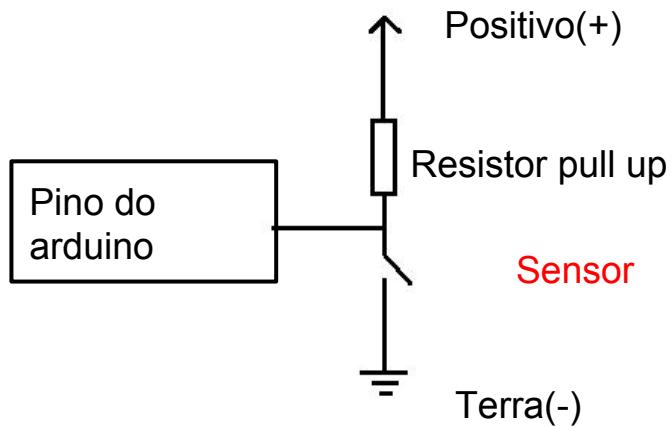
void loop() {
    Serial.println(num_pulses);
    delay(1000);
}
```

10 Arduino/Genuino Uno em COM1

Sensores

Atividade prática 15 - Refazer a atividade dos botões utilizando interrupções

NOW IT'S
YOUR TURN.



Atuadores

Como lidar com atuadores

Atuadores são elementos do sistema que atuam no ambiente

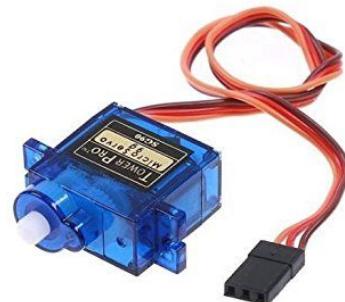
Motor DC;

Motor de passo;

Servo mecanismo;

Luz LED;

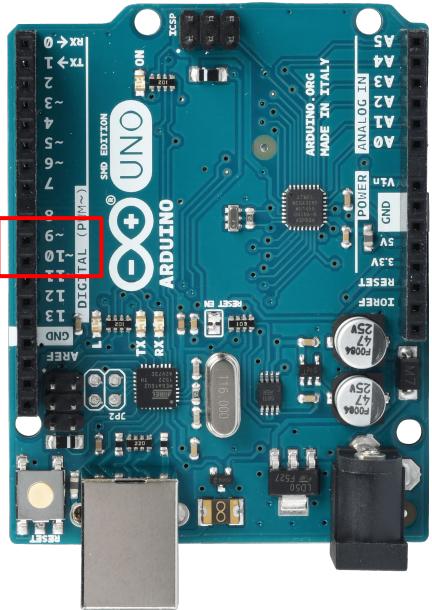
Buzzer;



Atuadores

Controle de potência e PWM

`analogWrite()` – escreve um valor analógico no pino



experiment_pwm | Arduino 1.8.1

Arquivo Editar Sketch Ferramentas Ajuda

experiment_pwm

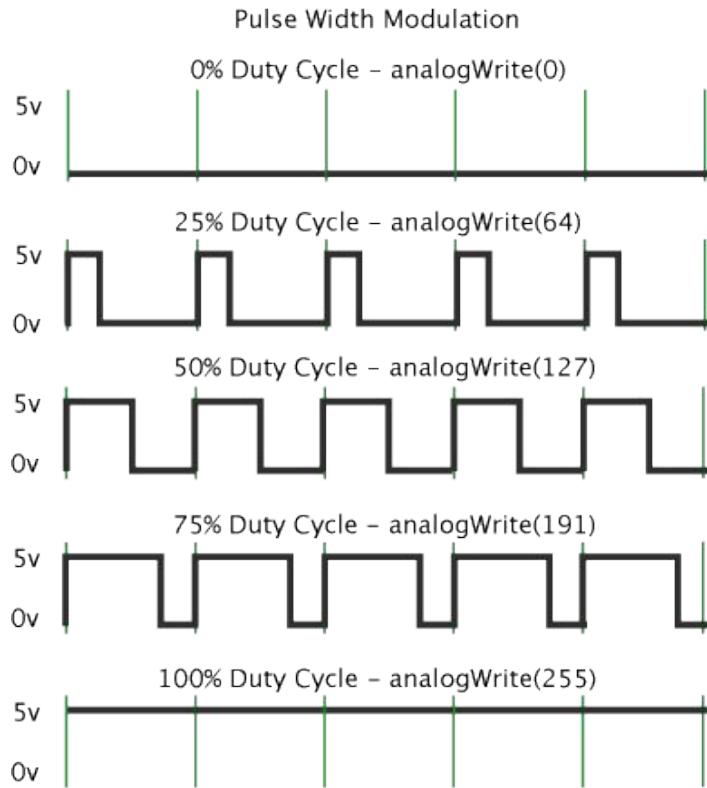
```
int pin_pwm = 9;
void setup() {
  pinMode(pin_pwm, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  analogWrite(pin_pwm, 200);
}
```

Arduino/Genuino Uno em COM3

Atuadores

PWM



a função `analogWrite()` escreve “pulsos” muito rápidos no pino digital (só funciona nos pinos marcados com PWM).

o valor a ser escrito representa o tempo que o pulso fica em nível alto e varia de 0 a 255.

quanto mais tempo o pulso permanecer em nível alto, maior é a “tensão média” da saída

Atuadores

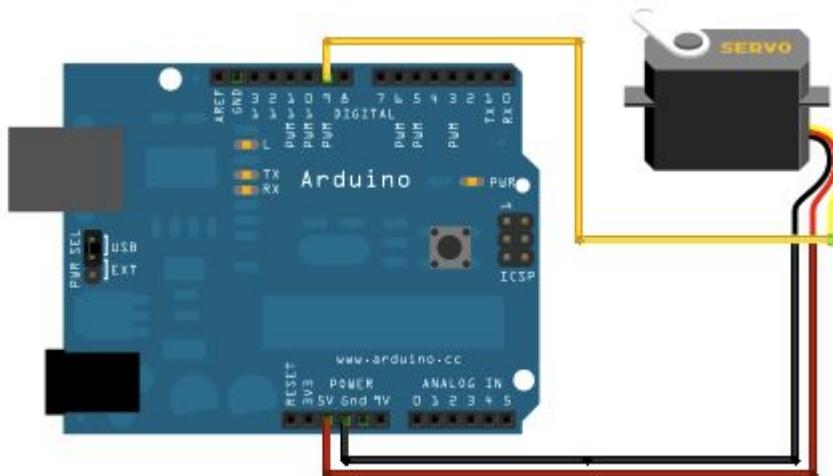
**Atividade prática 16 - Controlar o brilho de um LED através da
técnica PWM**

**NOW IT'S
YOUR TURN.**

Atuadores

Servo

Já possui um controlador de corrente e posição



Sweep | Arduino 1.8.1

Arquivo Editar Sketch Ferramentas Ajuda

Sweep

by Scott Fitzgerald
<http://www.arduino.cc/en/Tutorial/Sweep>

```
#include <Servo.h>

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

void setup() {
  myservo.attach(9); // attaches the servo on pin 9 to the servo
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to
    // in steps of 1 degree
    myservo.write(pos); // tell servo to go to position
    delay(15); // waits 15ms for the servo to get there
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to
    myservo.write(pos); // tell servo to go to position
    delay(15); // waits 15ms for the servo to get there
  }
}
```

Arduino/Genuino Uno em COM1

Atuadores

Atividade prática 17 - Controle a posição do servo com um potenciômetro

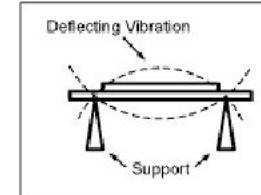
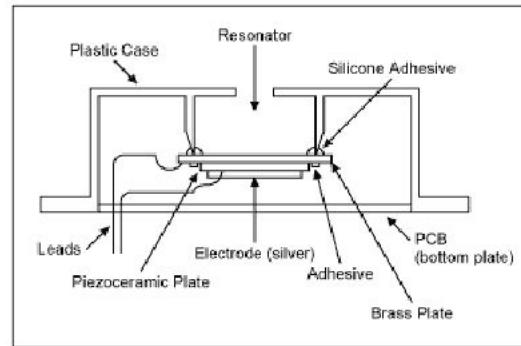
**NOW IT'S
YOUR TURN.**

Atuadores sonoros

Buzzer piezoelétrico

Formado por cerâmica piezoelétrica e disco metálico.

Ao receber uma tensão, a cerâmica se expande. Quando removemos a tensão ela volta.

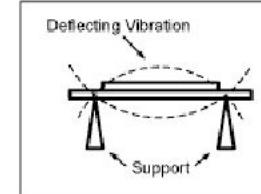
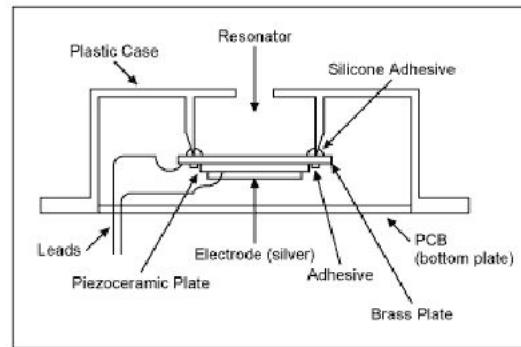


Atuadores sonoros

Buzzer piezoelétrico

2 polos: um é ligado no GND e outro no pino de saída desejado

aplicando uma tensão variável produz vibração que é traduzida em som



Atuadores sonoros

Como programar o arduino para tocar uma nota musical?

uma nota musical é um som em uma determinada frequência

a frequência de uma nota significa quantas vezes o atuador sonoro vibra em 1 segundo

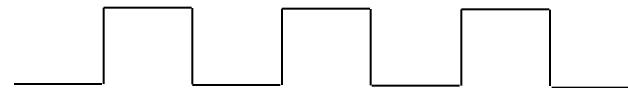
Atuadores sonoros

para fazer o atuador vibrar, escrevemos no pino uma sequência de valores HIGH e LOW, tantas vezes por segundo quanto for a frequência da nota

o tempo de cada variação HIGH e LOW é chamada de período e é o inverso da frequência

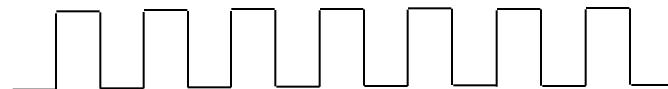
1 segundo

baixa frequência



período

alta frequência



período

Atuadores sonoros

Programar o arduino para tocar uma nota musical

```
void playTone(int period, int duration)
{
    for (long i = 0; i < duration * 1000L; i += period* 2)
    {
        digitalWrite(speakerPin, HIGH);
        delayMicroseconds(period);
        digitalWrite(speakerPin, LOW);
        delayMicroseconds(period);
    }
}
```

Atuadores sonoros

Programar o arduino para tocar uma nota musical

```
timeHigh = periodo / 2 = 1 / (2 * frequência)
```

* nota	frequência	periodo	tempo em nível alto
* c (dó)	261 Hz	3830	1915
* d (ré)	294 Hz	3400	1700
* e (mi)	329 Hz	3038	1519
* f (fá)	349 Hz	2864	1432
* g (sol)	392 Hz	2550	1275
* a (lá)	440 Hz	2272	1136
* b (si)	493 Hz	2028	1014
* C (dó)	523 Hz	1912	956

Não é necessário escrever essas frequências, podemos incluir o arquivo **pitches.h**

Atuadores sonoros

Como tocar uma nota musical?

pitches.h

```
#define NOTE_B0 31
#define NOTE_C1 33
#define NOTE_CS1 35
#define NOTE_D1 37
#define NOTE_DS1 39
#define NOTE_E1 41
#define NOTE_F1 44
#define NOTE_FS1 46
#define NOTE_G1 49
#define NOTE_GS1 52
...
```

Atuadores sonoros

Arduino já possui uma função para tocar notas

```
tone(pin, frequency);
// emite uma determinada nota (representada pela
// frequência) no pino correspondente

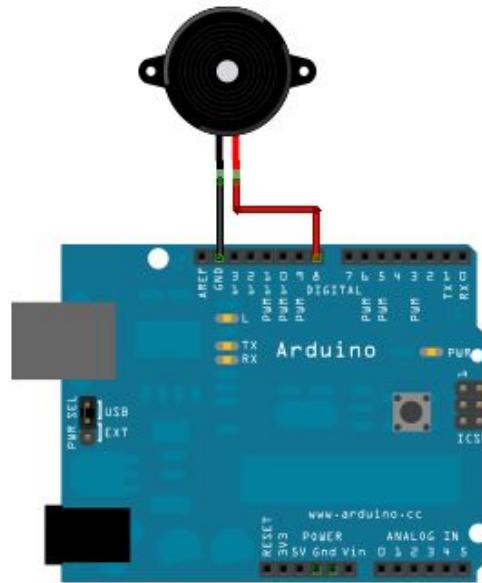
noTone(pin);
// para de emitir a frequência definida por tone()
// no pino correspondente

tone(pin, frequency, duration);
// emite uma determinada nota (representada pela
// frequência) no pino correspondente durante uma
// determinada duração
```

Atuadores

Atividade prática 17 - Testar a execução de notas musicais no Arduino (produza uma escala maior)

**NOW IT'S
YOUR TURN.**



Arduino - Timers

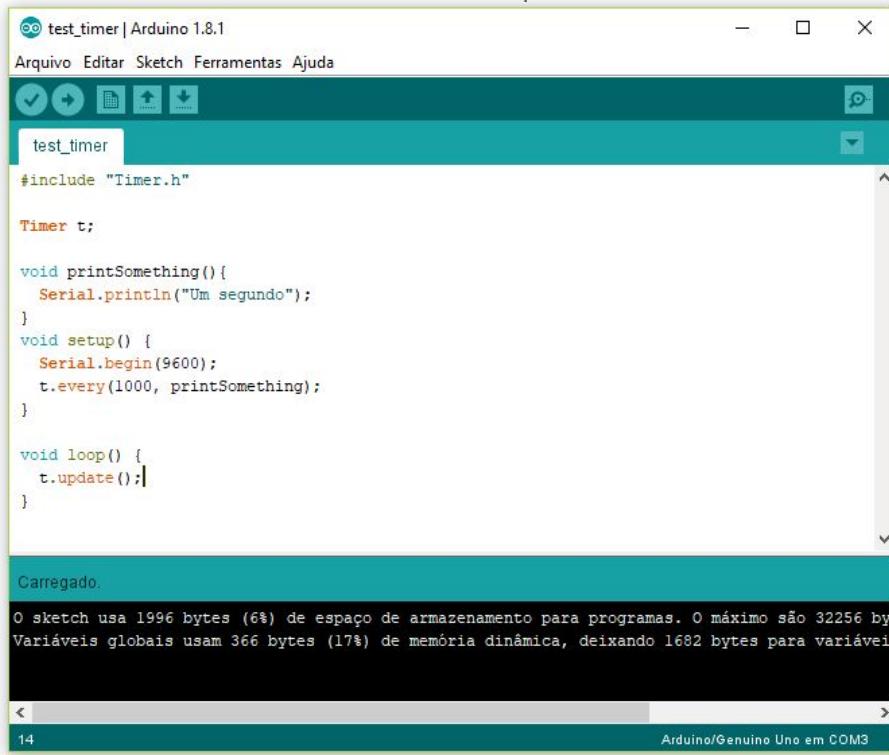
O Atmega, assim como muitos controladores, possuem hardwares específicos chamados **Timers** para permitir a contagem de tempo de forma precisa e assim, permite ao desenvolvedor criar aplicações que envolvam intervalos de tempo bem precisos.

Uma aplicação bastante útil é a recuperação periódica de uma função em intervalos de tempo específicos.

Vamos utilizar esta lib para o Timer:

<https://github.com/JChristensen/Timer>

Arduino - Timers



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** test_timer | Arduino 1.8.1
- Menu Bar:** Arquivo Editar Sketch Ferramentas Ajuda
- Toolbar:** Includes icons for Open, Save, Upload, and others.
- Code Editor:** The code for the sketch is displayed:

```
#include "Timer.h"

Timer t;

void printSomething(){
    Serial.println("Um segundo");
}

void setup() {
    Serial.begin(9600);
    t.every(1000, printSomething);
}

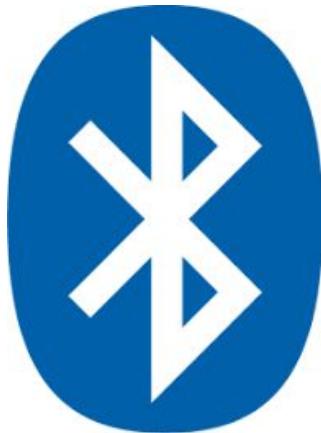
void loop() {
    t.update();
}
```
- Status Bar:** Carregado.
O sketch usa 1996 bytes (6%) de espaço de armazenamento para programas. O máximo são 32256 bytes.
Variáveis globais usam 366 bytes (17%) de memória dinâmica, deixando 1682 bytes para variáveis.
- Bottom Status:** Arduino/Genuino Uno em COM3

Projeto da disciplina - parte 1

Neste curso, cada grupo deverá desenvolver uma aplicação que envolva uma **aplicação android** e um **dispositivo físico**. Neste momento, planeje e prototipe a parte física do projeto.

Comunicação Arduino x Android

Neste momento, utilizaremos a comunicação **Bluetooth** para conectar a placa Arduino com um dispositivo Android. Isto nos permitirá uma comunicação sem fio simples (à qual o usuário comum está acostumado) e de baixo custo. O Bluetooth opera em **2.4GHz**



Comunicação Arduino x Android

Outras tecnologias que também operam em 2.4GHz

WiFi



ZigBee

Telefones sem fio

Microfones Wireless

Até fornos microondas emitem radiação em 2.4GHz

Bluetooth

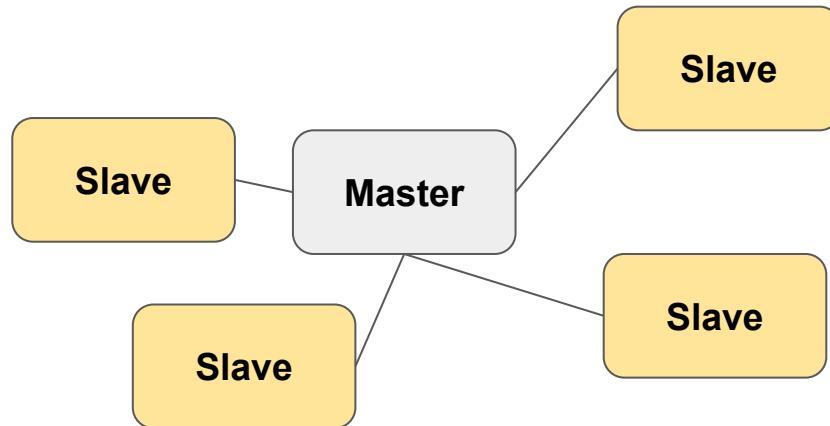
Tipos de Bluetooth

Classe	Potência máxima permitida	Alcance (Aproximado)
Classe 1	100 mW (20 dbm)	até 100 metros
Classe 2	2.5 mW (4 dbm)	até 10 metros
Classe 3	1 mW (0 dbm)	~ 1 metro

Bluetooth

Redes Bluetooth (Piconets)

Um dispositivo pode ser **Master** ou **Slave**. Cada Master pode se conectar com até 7 Slaves. Um Slave não pode se comunicar diretamente com outro Slave



Bluetooth

Redes Bluetooth (Piconets) - Endereçamento na rede

Cada dispositivo tem um endereço (**BD_ADDR**) com 48-bits que o identifica na rede. Geralmente representado por um valor hexadecimal de 12 dígitos. Os últimos 12 bits são a parte mais única do endereço. Dispositivos também podem possuir um **Nome** amigável para facilitar o uso.

Bluetooth

Processo de conexão Bluetooth

O processo de conexão Bluetooth tem os seguintes estados:

Inquiry: Processo no qual um dispositivo requisita o endereço dos dispositivos próximos

Paging: Processo de início de conexão

Connection: A conexão está estabelecida

Bluetooth

Bonding e Pairing

Dois dispositivos podem ser configurados para automaticamente iniciar uma conexão quando estiverem próximos. Isto é chamado de **Bonding**.

Este tipo de configuração é criado no processo chamado de pareamento ou **Pairing**, que exige a interação do usuário.

Bluetooth

Perfis de Bluetooth

A tecnologia Bluetooth é aplicada em diversas situações, que exigem diferentes requisitos.

Ex: Transferência de arquivos, Streaming de áudio, Distribuição de vídeos, etc....

Cada uma dessas aplicações determina um **profile Bluetooth**

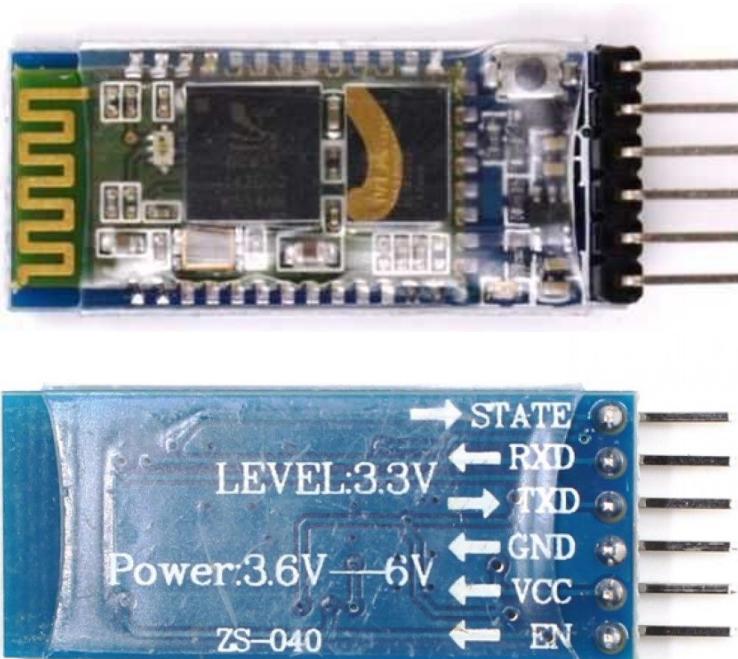
Lista completa em:

https://pt.wikipedia.org/wiki/Bluetooth#Requisitos_de_sistema

Bluetooth

Perfil utilizado

Dentre os perfis Bluetooth, utilizaremos o perfil SPP (Serial Port Profile), portanto, adequado para a comunicação Bluetooth. Para tal, utilizaremos o módulo HC-05 ou o módulo HC-06. Ambos **classe 2**



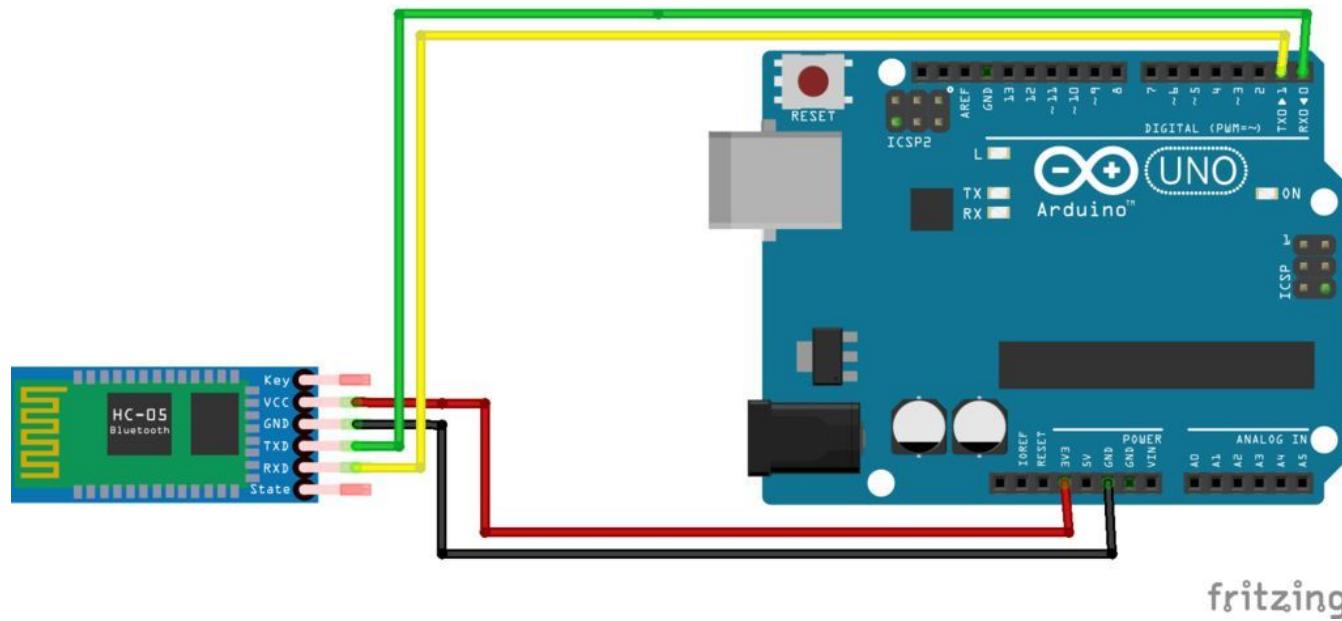
Bluetooth

Diferença entre os módulos

O módulo **HC-05** pode funcionar como Master ou Slave. Já o módulo HC-06 é mais simples e só pode funcionar como **Slave**

Bluetooth

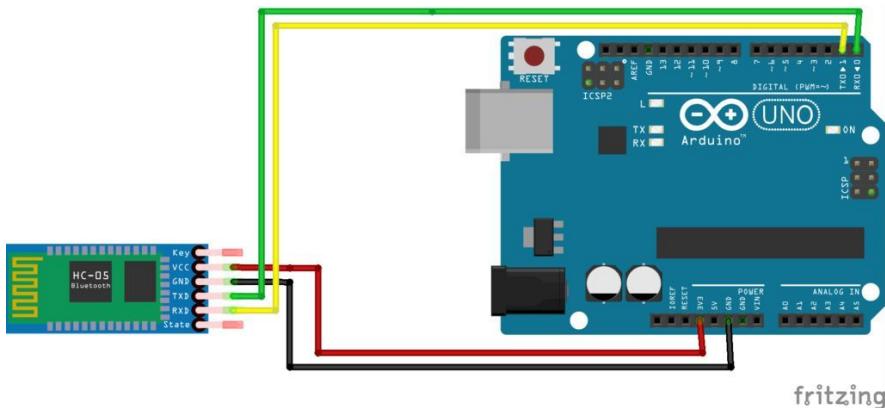
Ligaçāo do módulo HC-05 ao Arduino



Bluetooth

Atividade prática 18 - Teste a comunicação arduino x android com uma aplicação de Bluetooth serial e o Serial monitor do IDE do Arduino

Passo 1: Pareie o módulo Bluetooth com o celular. A senha é 1234 ou 0000



Bluetooth

Atividade prática 19 - Faça um programa de loopback (o arduino deve devolver via bluetooth tudo o que ele recebe)

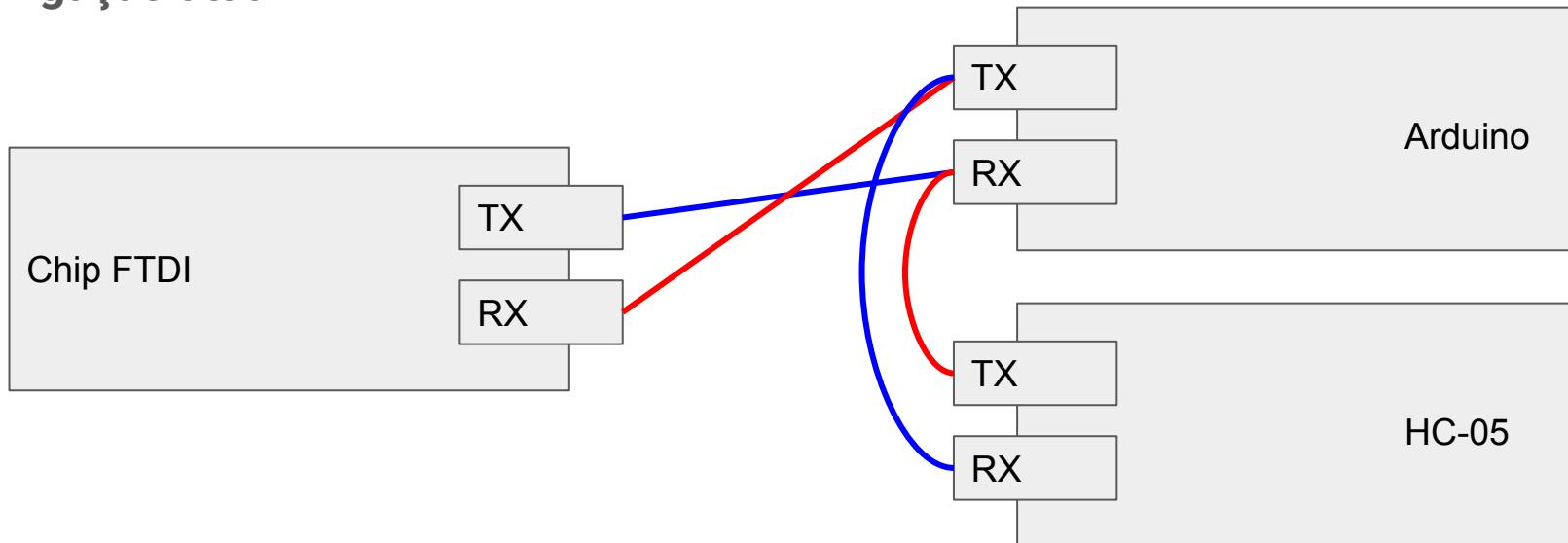
**NOW IT'S
YOUR TURN.**

Configurando o módulo Bluetooth

Como pudemos observar, a configuração padrão do módulo HC-05 é um dispositivo com o nome “HC-05” e senha 1234. No entanto, podemos customizar estas configurações. Para tal, precisaremos fazer uma ligação um pouco diferente com o módulo HC-05, a fim de ser possível mandar comandos seriais ao módulo através do **monitor serial**

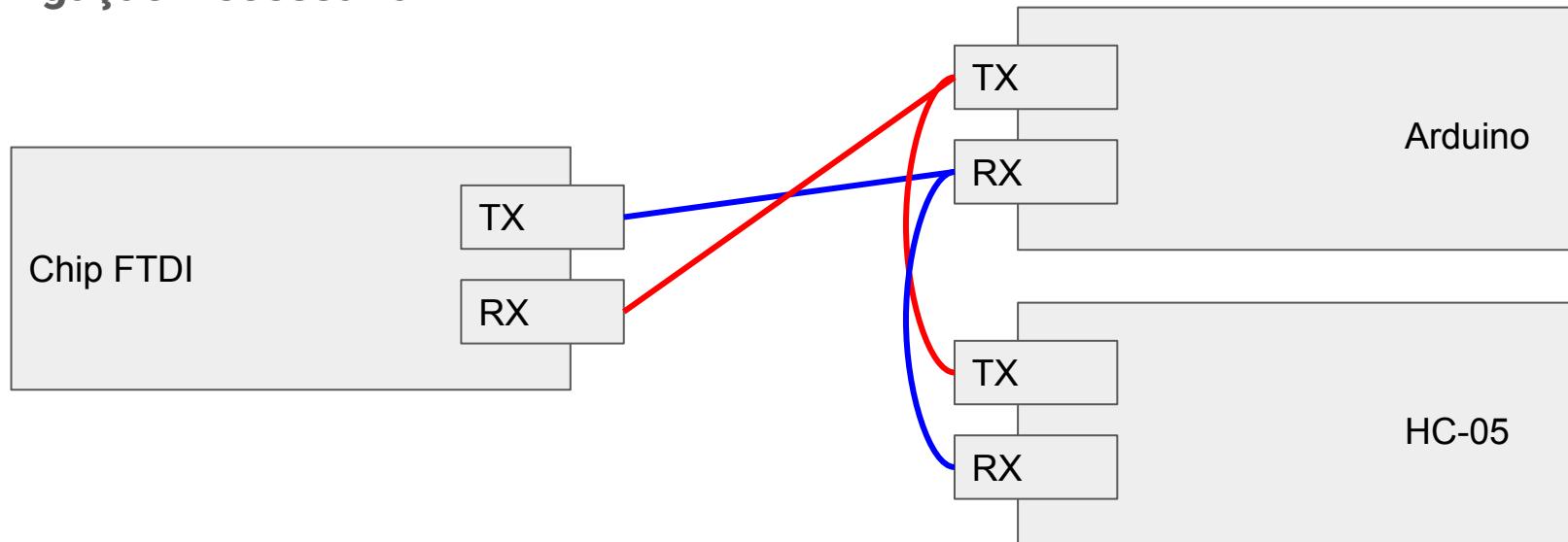
Configurando o módulo Bluetooth

Ligação atual



Configurando o módulo Bluetooth

Ligação Necessária

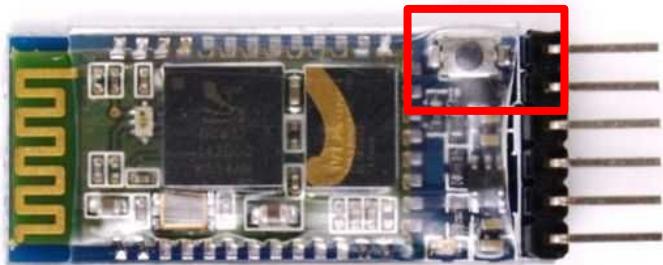


Configurando o módulo Bluetooth

Entrando no modo “SLAVE”

Para o HC-05, mantenha o botão presente na placa pressionado antes de conectar o pino VCC. O LED deve piscar devagar.

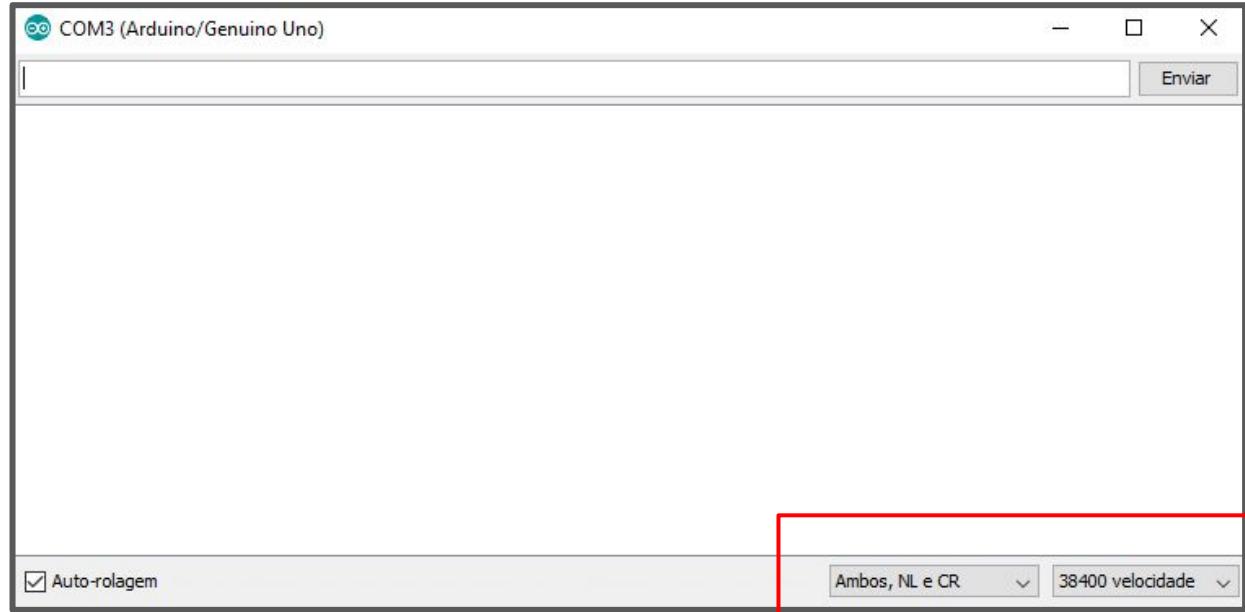
Para o HC-06, nenhuma ação é necessária, somente conecte o módulo.



Configurando o módulo Bluetooth

Configurando o monitor Serial (HC-05)

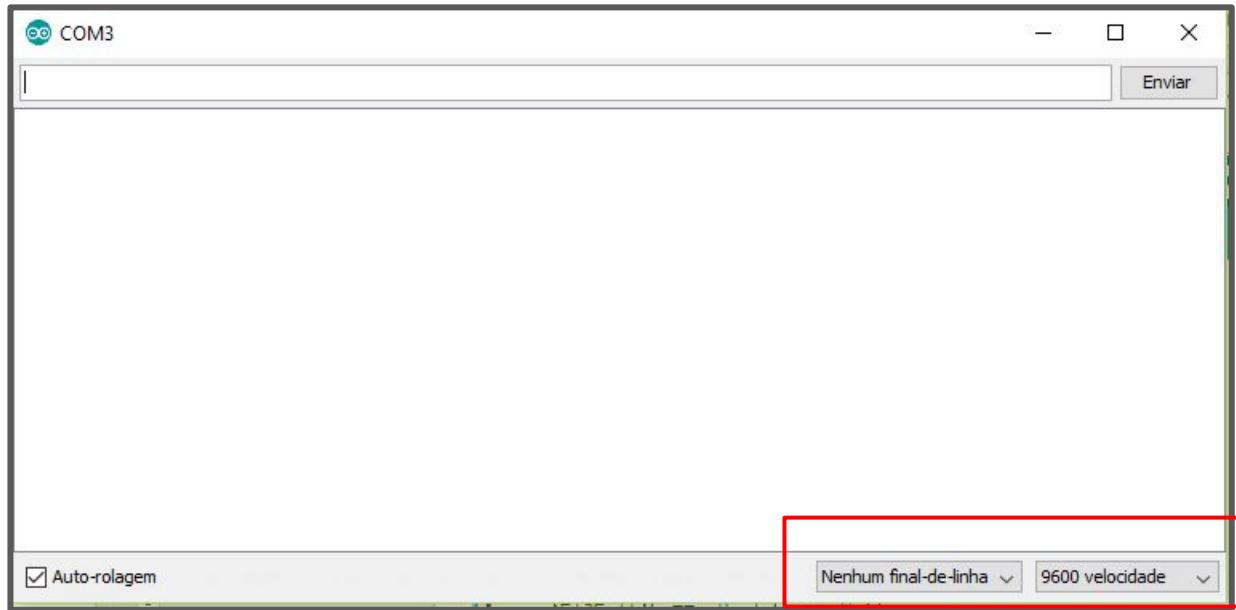
Insira o Baudrate **38400** e a opção NL + CR para terminador de linha



Configurando o módulo Bluetooth

Configurando o monitor Serial (HC-06)

Insira o Baudrate **9600** (ou o Baudrate de operação) e a opção Nenhum final de linha



Configurando o módulo Bluetooth

Comandos mais úteis

HC-05:

AT+NAME=x: Setar o nome

AT+ADDR: Ver o endereço Bluetooth

AT+ORGL: Resetar as configurações de fábrica

AT+PSWD: Ver a senha default/ AT+PSWD=x: Setar a senha

Configurando o módulo Bluetooth

Comandos mais úteis

HC-06:

AT+NAMEx: Setar o nome (Sem espaço ou "=")

AT+ADDR: Não é implementado

AT+ORGL: Não é implementado

AT+PINx: Setar a senha (Sem espaço ou "=")

Configurando o módulo Bluetooth

Setando o Baudrate

HC-05:

AT+UART=9600

HC-06:

AT+BAUDx

x	baudrate(bps)
1	1200
2	2400
3	4800
4	9600
5	19200
6	38400
7	19200
6	57600
8	115200

Comunicação Bluetooth no Android

Para realizar a comunicação entre um módulo Bluetooth e o Android, utilizaremos a própria API nativa do Android. Estas são as principais classes que utilizaremos ao longo das nossas práticas

BluetoothAdapter - Responsável por criar conexões com dispositivos bluetooth

BluetoothDevice - Abstração de um dispositivo pareado com o Smartphone

BluetoothSocket - Uma conexão específica com um dispositivo Bluetooth

Requisitos para a comunicação bluetooth

Em primeiro lugar, precisamos ativar as permissões necessárias para a utilização do Bluetooth

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

```
<uses-permission android:name="android.permission.BLUETOOTH" />
```

Requisitos para a comunicação bluetooth

Em alto nível, podemos identificar alguns requisitos para uma boa comunicação entre o smartphone e o device bluetooth

Código 1

Checa a permissão do Bluetooth e eventualmente pede para o usuário ativar

Código 2

Código que lista os dispositivos pareados e encontra os seus **nomes** e **endereços**

Requisitos para a comunicação bluetooth

Em alto nível, podemos identificar alguns requisitos para uma boa comunicação entre o smartphone e o device bluetooth

Código 3

Inicia uma conexão com o Bluetooth.
Deve ser um
AsyncTask

Código 4

Envia informações via Bluetooth

Código 5

Recebe informações via Bluetooth. Deve ser uma Thread

Código 1 - Iniciando o Bluetooth

A inicialização do Bluetooth e a listagem dos dispositivos pareados é feita através da classe **BluetoothAdapter**

```
public void initBluetoothAdapter(){
    myBluetooth = BluetoothAdapter.getDefaultAdapter();
    if (myBluetooth == null) {
        Toast.makeText(this.parent.getApplicationContext(),
                      "Funcionalidade Bluetooth não disponível",
                      Toast.LENGTH_LONG).show();
        this.parent.finish();
    }else if(!myBluetooth.isEnabled()){
        Intent bluetoothPerm = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        this.parent.startActivityForResult(bluetoothPerm, 1);
    }
}
```

Código 2 - Listando os dispositivos pareados

A listagem dos dispositivos pareados é igualmente feita pela classe **BluetoothAdapter**

```
public ArrayList<String> getPairedDevices(){
    ArrayList<String> deviceInfo = new ArrayList<String>();

    Set pairedDevices = myBluetooth.getBondedDevices();

    for (Object obj : pairedDevices){
        BluetoothDevice bt = (BluetoothDevice)obj;
        deviceInfo.add(bt.getName() + '_' + bt.getAddress());
    }

    return deviceInfo;
}
```

Códigos 1 e 2 - Parte prática

Vamos encapsular o código visto até agora em uma classe chamada **BluetoothController**.



Códigos 1 e 2 - Parte prática

Atividade prática 20 - Crie uma aplicação que liste dos dispositivos pareados no Logcat



Antes do código 3

Para criar as conexões Bluetooth, precisamos de um objeto **BluetoothSocket** . Vamos adicioná-lo ao BluetoothController. Além disso, vamos inserir uma variável booleana que indica se o Bluetooth está conectado ou não. Vamos também criar os getters and setters dessas variáveis.



Código 3 - Conectando

A criação das conexões também é responsabilidade do **BluetoothAdapter**. No entanto, estas operações devem acontecer dentro de uma **AsyncTask** para não prejudicar as demais funções do app.

```
public class BluetoothConnection extends AsyncTask<Void, Void, Void>{  
  
    BluetoothController myController;  
    String addressToConnect;  
  
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");  
  
    boolean connectionSuccess = true;  
  
    BluetoothConnection(BluetoothController controller, String addressToConnect){  
        this.myController = controller;  
        this.addressToConnect = addressToConnect;  
    }  
  
    @Override  
    protected void onPreExecute(){  
        Toast.makeText(this.myController.getActivity().getApplicationContext(),  
                    "Conectando ao Bluetooth",  
                    Toast.LENGTH_LONG).show();  
    }  
}
```

Código 3 - Conectando

A criação das conexões também é responsabilidade do **BluetoothAdapter**. No entanto, estas operações devem acontecer dentro de uma **AsyncTask** para não prejudicar as demais funções do app.

```
@Override  
protected Void doInBackground(Void... devices){  
    try{  
        if(this.myController.getSocket() == null){  
            BluetoothDevice btDevice = this.myController.getAdapter().getRemoteDevice(this.addressToConnect);  
            BluetoothSocket btSocket = btDevice.createInsecureRfcommSocketToServiceRecord(myUUID);  
            btSocket.connect();  
            this.myController.setSocket(btSocket);  
        }  
    }catch (IOException ex){  
        Log.e("Erro ao conectar", ex.getMessage());  
        this.connectionSuccess = false;  
    }  
  
    return null;  
}
```

Código 3 - Conectando

A criação das conexões também é responsabilidade do **BluetoothAdapter**. No entanto, estas operações devem acontecer dentro de uma **AsyncTask** para não prejudicar as demais funções do app.

```
@Override  
protected void onPostExecute(Void result){  
    if(!this.connectionSuccess){  
        Toast.makeText(this.myController.getActivity().getApplicationContext(),  
                    "Erro na conexao bluetooth",  
                    Toast.LENGTH_LONG).show();  
        this.myController.setConnected(false);  
    }else{  
        Toast.makeText(this.myController.getActivity().getApplicationContext(),  
                    "Conexao estabelecida com sucesso",  
                    Toast.LENGTH_LONG).show();  
        this.myController.setConnected(true);  
    }  
}
```

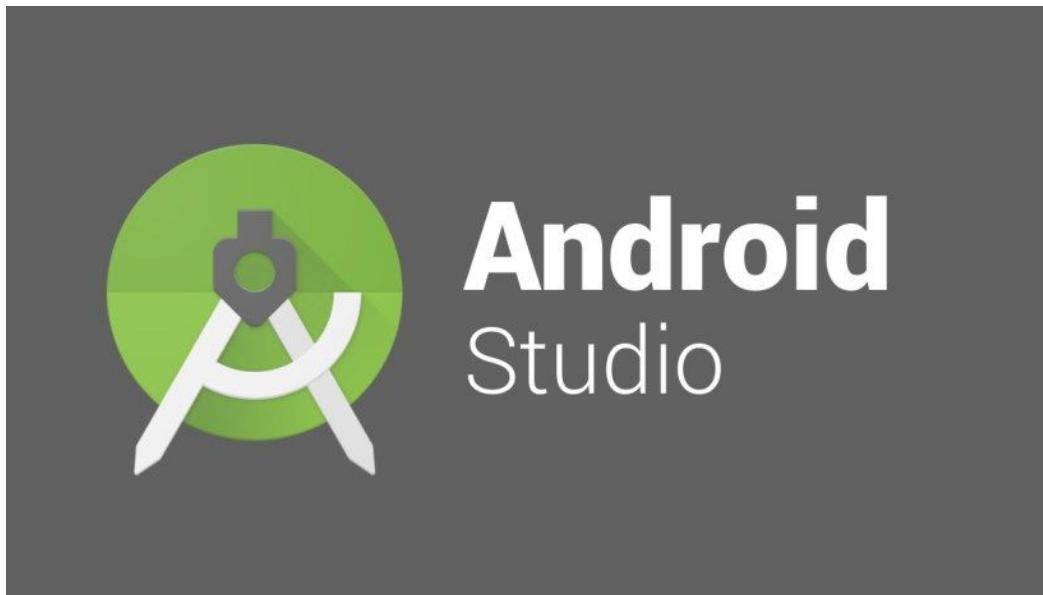
Códigos 3 - parte prática

Vamos criar uma classe que controlará a geração de novas conexões. Ela se chamará **BluetoothConnection**



Códigos 3 - parte prática

Vamos adicionar as chamadas à classe **BluetoothConnection** à classe **BluetoothController**



Códigos 4 - enviando informações

Utilizaremos o objeto **BluetoothSocket** criado para enviar informações pelo Bluetooth. Podemos criar uma nova função na classe **BluetoothController**

```
public boolean sendByBluetooth(String s){
    if (!isConnected()){
        Toast.makeText(parent.getApplicationContext(),
            "Conecte com o Bluetooth primeiro",
            Toast.LENGTH_SHORT).show();

        return false;
    }else{
        try{
            myBtSocket.getOutputStream().write(s.getBytes());
            Log.i("Bluetooth", "Escreveu no Bluetooth " + s);
            return true;
        }catch(IOException ex){
            return false;
        }
    }
}
```

Códigos 4 - parte prática

Implemente a função em questão



Códigos 5 - Lendo informações do Bluetooth

Para fazer isto, criaremos uma Thread

```
import android.bluetooth.BluetoothSocket;
import android.util.Log;
import android.widget.Toast;

import java.io.InputStream;
import java.io.IOException;

public class BluetoothListener extends Thread {
    BluetoothController controller;
    InputStream bluetoothIn = null;
    boolean isRunning = false;

    BluetoothListener(BluetoothController controller){
        this.controller = controller;
        try{
            this.bluetoothIn = controller.getSocket().getInputStream();
        }catch(IOException ex){
        }
        this.isRunning = true;
    }

    @Override
    public void run() {
        byte[] buffer = new byte[1024];
        int bytes;
        while(isRunning) {
            try{
                bytes = bluetoothIn.read(buffer);
                String readMessage = new String(buffer, 0, bytes);
                Log.d("Bluetooth Listener", "Received: " + readMessage);
            }catch(IOException e){
            }
        }
    }
}
```

Códigos 5 - Lendo informações do Bluetooth

Para fazer isto, criaremos uma Thread

```
public void run(){
    byte[] buffer = new byte[256];
    int bytes = 0;

    while(this.isRunning){
        try{
            int bytesAvailable = this.bluetoothIn.available();

            if(bytesAvailable > 0){
                bytes = this.bluetoothIn.read(buffer, off: 0, bytesAvailable);
                String result = new String(buffer);

                this.controller.getActivity().runOnUiThread(
                    new actionToExecute(controller, result));

                Log.i( tag: "Bytes lidos", result);
            }

        }catch(IOException ex){
            Log.e( tag: "Erro no bluetooth", ex.getMessage());
        }
    }

    Log.i( tag: "Bluetooth", msg: "Fechando os streams do Bluetooth");

    this.bluetoothIn = null;
}
```

Códigos 5 - Lendo informações do Bluetooth

Para fazer isto, criaremos uma Thread

```
public void closeListener(){
    this.isRunning = false;
}

}

class actionToExecute implements Runnable{
    String receivedString;
    BluetoothController controller;

    actionToExecute(BluetoothController controller, String receivedString){
        this.controller = controller;
        this.receivedString = receivedString;
    }

    public void run(){
        Toast.makeText(this.controller.getActivity().getApplicationContext(),
            text: "Recebeu " + receivedString,
            Toast.LENGTH_SHORT).show();
    }
}
```

Códigos 5 - Lendo informações do Bluetooth

Para receber mensagens mais complexas, pode ser necessário um protocolo

```
public void run(){
    byte[] buffer = new byte[256];
    int bytes = 0;
    int latestPosition = 0;

    while(this.isRunning){
        try{
            int bytesAvailable = this.blInput.available();

            if(bytesAvailable > 0){
                bytes = this.blInput.read(buffer, latestPosition, bytesAvailable);
                latestPosition += bytes;

                String result = new String(buffer, offset: 0, latestPosition);
                Log.d( tag: "TEST", String.valueOf(result.contains("\n")));

                if(result.contains("\n")){
                    latestPosition = 0;
                    this.myController.getParent().runOnUiThread(new actionToExecute(this.myController.parent, result));
                }
            }
        }catch(Exception ex){}

    }
}
```

Bluetooth

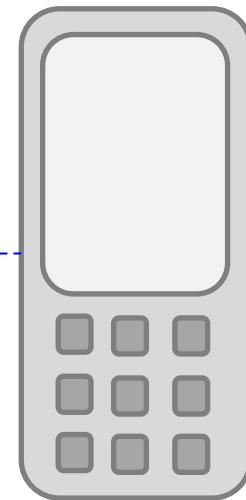
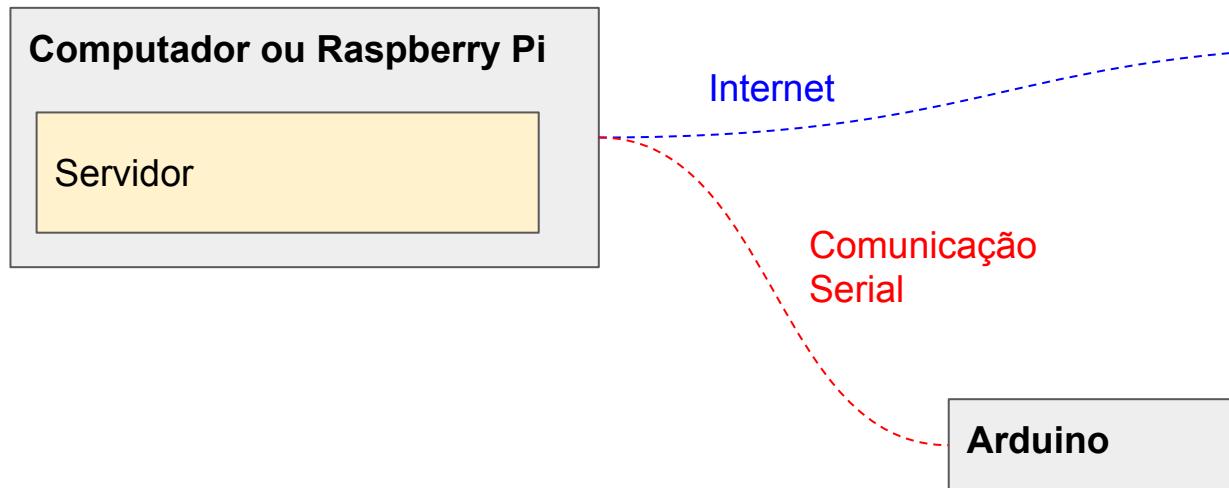
Atividade prática 21 - Faça um programa que controla a potência de um LED com um Slider no Android

Atividade prática 22 - Faça um programa que acenda a luz do celular quando a luz da sala é apagada

NOW IT'S
YOUR TURN.

Alternativa ao Bluetooth - Comunicação via Web Server

Podemos realizar a comunicação com o Arduino através da Internet a partir da seguinte arquitetura



Servidor NodeJS

Para o desenvolvimento da aplicação servidora, utilizaremos a tecnologia Node Js, que permite a construção de servidores completamente em **JavaScript**

Instale o NodeJs agora!

<https://www.ostechnix.com/install-node-js-linux/>

<https://nodejs.org/en/download/>



Servidor NodeJS

Iniciando um projeto NodeJS

O NodeJS possui um gerenciador de dependências chamado **npm**. Para iniciar um projeto, rode o comando **npm init** na pasta e insira **server.js** como entry point. Observe que foi criada a pasta **node_modules** para guardar os módulos do projeto e o arquivo **packages.json** que guarda as configurações do projeto



Servidor NodeJS

Instalando uma lib

Vamos instalar a lib de porta serial no nosso projeto.

Execute o comando:

```
npm install --save serialport
```



Servidor NodeJS

Código inicial do servidor

```
const http = require('http');
const port = 3000;

const requestHandler = (request, response) => {
  if (!request.url.includes('favicon')){
    console.log(request.url);

    if(request.url.includes(' acende')){
      console.log('Mandando acender');
    }else if(request.url.includes('apaga')){
      console.log('Mandando apagar');
    }

    response.end('OK');
  }
}

const server = http.createServer(requestHandler);

server.listen(port, (err) => {
  if (err){
    return console.log('Erro:', err);
  }

  console.log(`Servidor em execucao na porta ${port}`);
});
```

Servidor NodeJS

Adição do código Serial

```
var SerialPort = require('serialport');
const Delimiter = require('@serialport/parser-delimiter');

var serialPort = new SerialPort('COM3', {
  baudRate: 9600,
});

const parser = serialPort.pipe(new Delimiter({ delimiter: '\n' }));

parser.on('data', function(buffer){
  console.log('Received data', buffer.toString('utf8'));
});

function sendBySerial(data){
  serialPort.write(data, function (err, result) {
    if (err) {
      console.log('Erro ao enviar dados seriais : ' + err);
    }
    if (result) {
      console.log('Resposta apos o envio : ' + result);
    }
  });
}
```

Servidor NodeJS

Adição do código Serial

```
const requestHandler = (request, response) => {
  if (!request.url.includes('favicon')){
    console.log(request.url);

    if(request.url.includes('acende')){
      console.log('Mandando acender');
      sendBySerial('A');
    }else if(request.url.includes('apaga')){
      console.log('Mandando apagar');
      sendBySerial('B');
    }

    response.end('OK');
  }
}
```

Comunicação com Android

O Lado do Android

Para realizarmos requisições Http, podemos utilizar a biblioteca Volley do Android



Android Volley Library

Comunicação com Android

O Lado do Android

Comece inserindo a permissão de internet

```
<uses-permission android:name="android.permission.INTERNET"/>
```

E posteriormente, insira a dependência no seu arquivo gradle



The screenshot shows the Android Studio interface with the build.gradle file open. The left sidebar shows project files like mipmap, values, Gradle Scripts, build.gradle (Project), build.gradle (Module), gradle-wrapper.properties, proguard-rules.pro, gradle.properties, settings.gradle, and local.properties. The build.gradle (Module) file is selected and shown in the main editor area. The code in the editor is:

```
21
22 dependencies {
23     compile fileTree(dir: 'libs', include: ['*.jar'])
24     androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25         exclude group: 'com.android.support', module: 'support-annotations'
26     })
27     compile 'com.android.support:appcompat-v7:25.2.0'
28     compile 'com.android.support.constraint:constraint-layout:1.0.2'
29     compile 'com.android.volley:volley:1.0.0'
30     testCompile 'junit:junit:4.12'
31 }
```

A red box highlights the last four lines of the dependencies block, which define the Volley library dependency.

Comunicação com Android

O Lado do Android

```
import com.android.volley.RequestQueue;
import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.Volley;
import android.util.Log;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    queue = Volley.newRequestQueue(this);
```

```
RequestQueue queue;

public void sendRequest(String url){
    // Request a string response from the provided URL.
    StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                Log.d("FUNCIONOU", response);
            }
        }, new Response.ErrorListener() {

            @Override
            public void onErrorResponse(VolleyError error) {
                Log.e("ERRO", "Erro na resposta!");
            }
    );
    queue.add(stringRequest);
}
```

Comunicação com Android

Atividade prática 22 - Crie um aplicativo para controlar um LED via arquitetura cliente servidor

**NOW IT'S
YOUR TURN.**

Projeto da disciplina - parte 2

Neste curso, cada grupo deverá desenvolver uma aplicação que envolva uma **aplicação android** e um **dispositivo físico**. Neste momento, planeje e prototipe tal projeto.