Oficina de Robótica

Eletrônica e Programação em Arduino

OFICINA DE ROBÓTICA
INVENTAR E RECICLAR PARA EDUCAR
oficinaderobotica.ufsc.br

LARM
Laboratório de Automação e Robótica Móvel

UNIVERSIDADE FEDERAL DE SANTA CATARINA

OFICINA DE ROBÓTICA
INVENTAR E RECICLAR PARA EDUCAR
oficinaderobotica.ufsc.br

Apresentação

- Material produzido para o projeto Oficina de Robótica por:
  - Anderson Luiz Fernandes Perez
  - Renan Rocha Darós

- Contatos:
  - Universidade Federal de Santa Catarina – Laboratório de Automação e Robótica Móvel
    - anderson.perez (at) ufsc.br
    - renanrdaros (at) hotmail.com

- http://oficinaderobotica.ufsc.br

UNIVERSIDADE FEDERAL DE SANTA CATARINA

LARM
Laboratório de Automação e Robótica Móvel

Sumário

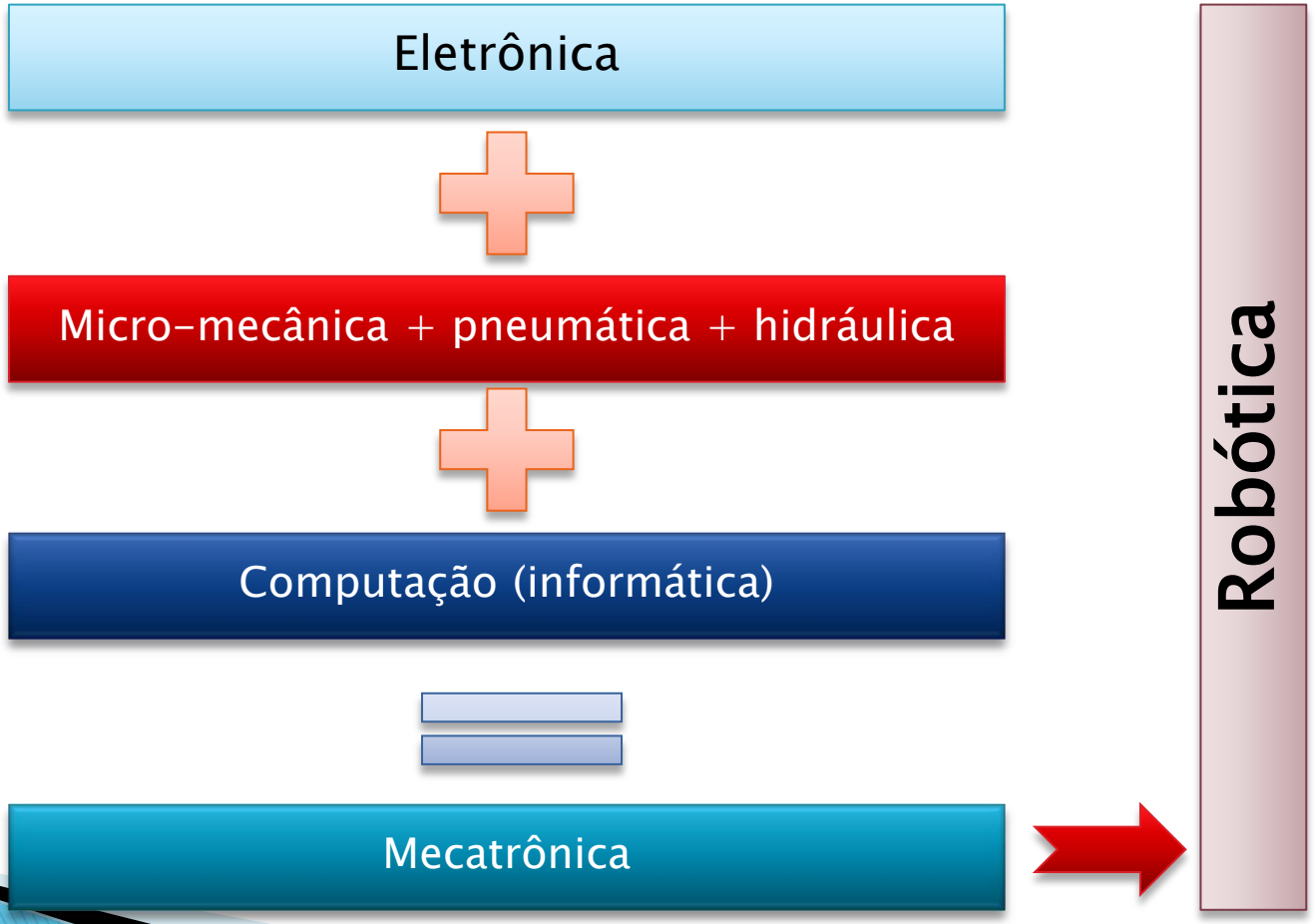
- ELETRÔNICA
  - Introdução
  - Diagramas Esquemáticos
  - Corrente e Tensão
  - Resistência Elétrica
  - Condutores e Isolantes
  - Resistores
  - LED
  - Protoboard
  - Motor DC
  - Ponte H
- ARDUINO
  - Introdução ao Arduino
  - Arduino UNO
  - Programação
  - Ambiente de Desenvolvimento
  - Funções *setup()* e *loop()*
  - Monitor Serial
  - E/S Digital
  - Entrada Analógica
  - PWM

Introdução

- A eletrônica pode ser definida como a ciência que estuda formas de controlar a energia elétrica em circuitos elétricos.
- É um ramo da engenharia que desenvolve soluções aplicando os princípios de eletricidade descobertos pela física.
- Usa circuitos elétricos formados por condutores elétricos e componentes eletrônicos para controlar sinais elétricos.
- A eletrônica divide-se em analógica e digital.

Introdução

- Eletrônica na robótica



Diagramas Esquemáticos

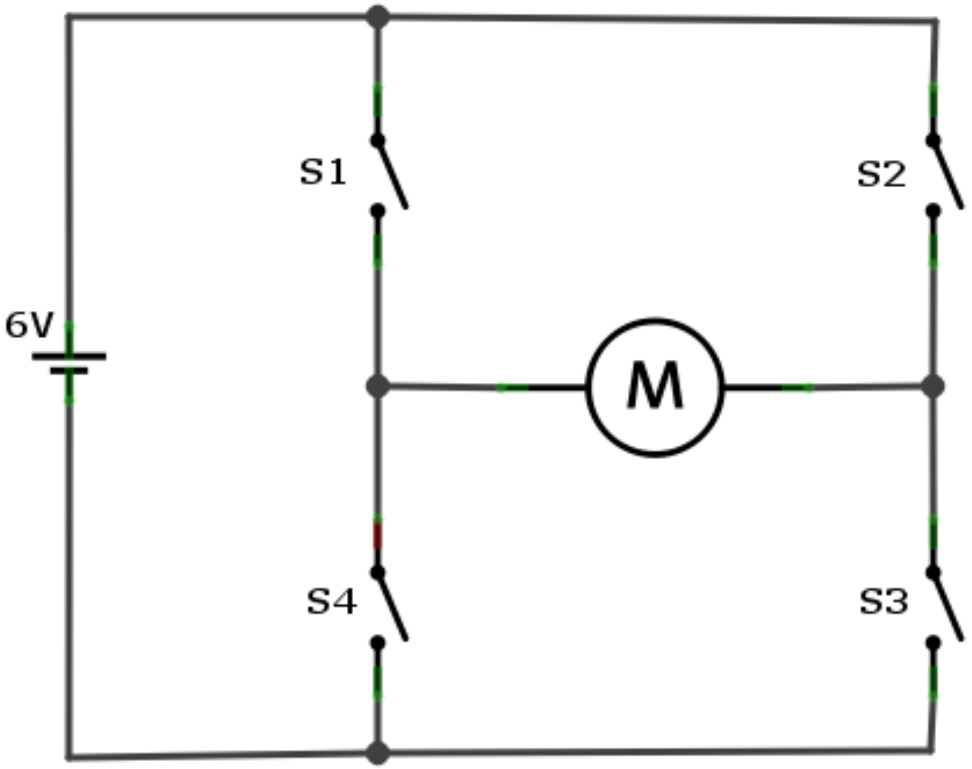
- É comum precisarmos representar graficamente um circuito ou sistema eletrônico.
- A representação gráfica de sistemas e circuitos eletrônicos facilita o desenvolvimento de novos projetos, serve como guia durante o reparo de equipamentos e é uma ótima ferramenta de ensino e aprendizagem de eletrônica.
- Dentre todas as formas de representação gráfica utilizadas na eletrônica, a que mais se destaca são os Diagramas Esquemáticos.

Diagramas Esquemáticos

- Apresentam a forma como todos os componentes de um circuito estão interconectados para formar um sistema ou parte dele.
- Utilizam símbolos, que representam os mais variados componentes eletrônicos, e linhas que representam as conexões entre os terminais desses componentes.

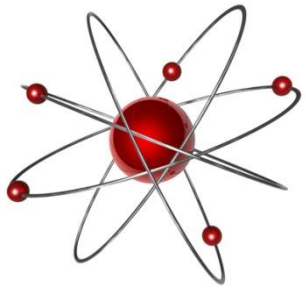
Diagramas Esquemáticos

- Exemplo:



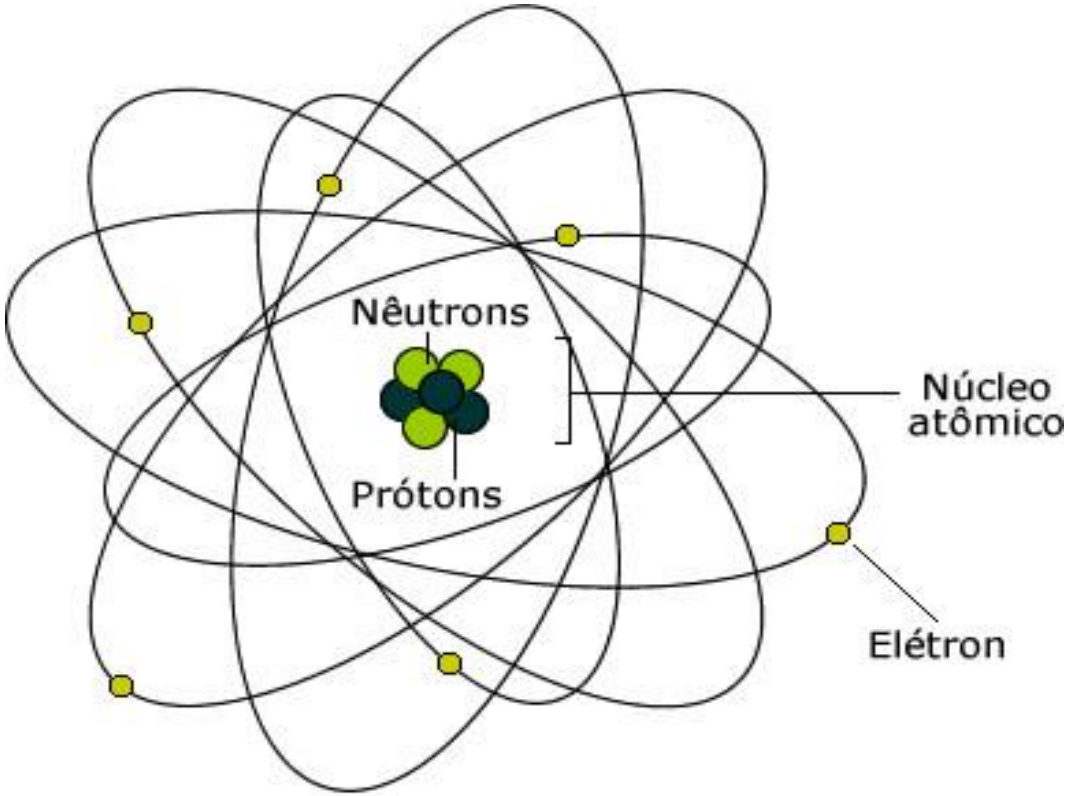
Corrente e Tensão

- Átomo
  - Composto por prótons, nêutrons e elétrons.
  - Os prótons carregam cargas positivas e estão presentes no núcleo do átomo.
  - Os nêutrons não carregam carga e assim como os prótons estão presentes no núcleo do átomo.
  - Os elétrons carregam carga negativa e orbitam o núcleo do átomo.



Corrente e Tensão

- Átomo



Corrente e Tensão

OFICINA DE ROBÓTICA
INVENTAR E RECICLAR PARA EDUCAR
oficinaderobotica.ufsc.br

- Átomo
  - Quando o átomo possui o mesmo número de elétrons e de prótons é considerado neutro.
  - Quando o átomo possui um número maior de prótons do que de elétrons é considerado positivo.
  - Quando o átomo possui um número maior de elétrons do que de prótons é considerado negativo.

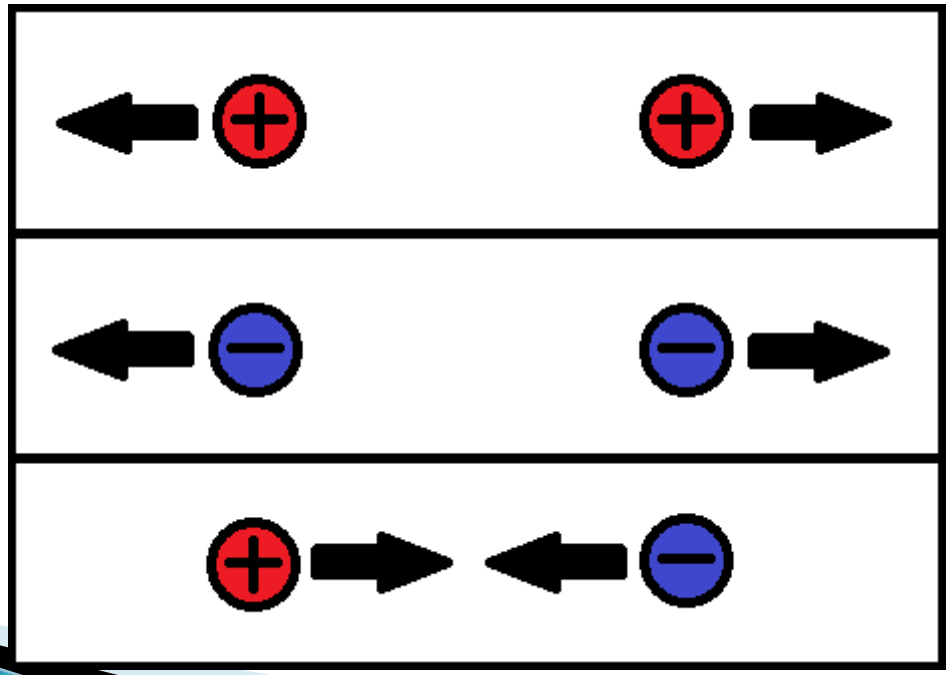
Ionização é o nome dado quando o átomo ganha ou perde elétrons.

UNIVERSIDADE FEDERAL DE SANTA CATARINA

LARM
Laboratório de Automação e Robótica Móvel

Corrente e Tensão

- Atração e Repulsão
  - Corpos com cargas de sinais opostos de atraem e corpos com cargas de mesmo sinal de repelem.



Corrente e Tensão

OFICINA DE ROBÓTICA
INVENTAR E RECICLAR PARA EDUCAR
oficinaderobotica.ufsc.br

- Corrente Elétrica
  - Os elétrons livres movimentam-se de um átomo a outro através de um meio condutor.
  - Corrente elétrica é um fluxo de elétrons que circula em um condutor.
  - A corrente elétrica (I) é medida em Ampère (A).
  - Para os elétrons se moverem de um átomo a outro é necessário haver uma diferença de potencial ou tensão.

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

LARM
Laboratório de Automação
e Robótica Móvel

Corrente e Tensão

OFICINA DE ROBÓTICA
INVENTAR E RECICLAR PARA EDUCAR
oficinaderobotica.ufsc.br

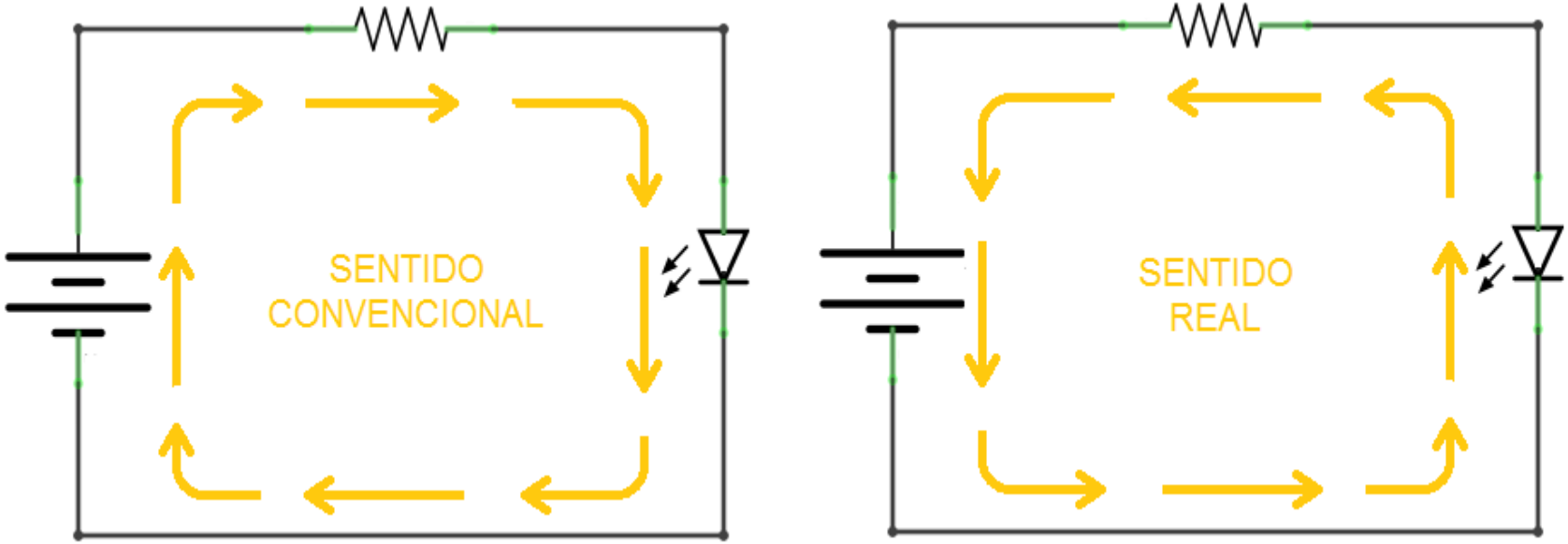
- Sentido Real vs Sentido Convencional da Corrente Elétrica
  - Em um circuito os elétrons livres se deslocam do polo negativo para o polo positivo. Esse é o sentido real da corrente elétrica.
  - Em análise de circuitos, entretanto, costuma-se considerar que os elétrons se deslocam no sentido oposto: do polo positivo para o polo negativo. Esse é o sentido convencional da corrente elétrica.

UNIVERSIDADE FEDERAL DE SANTA CATARINA

LARM
Laboratório de Automação e Robótica Móvel

Corrente e Tensão

- Sentido Real vs Sentido Convencional da Corrente Elétrica

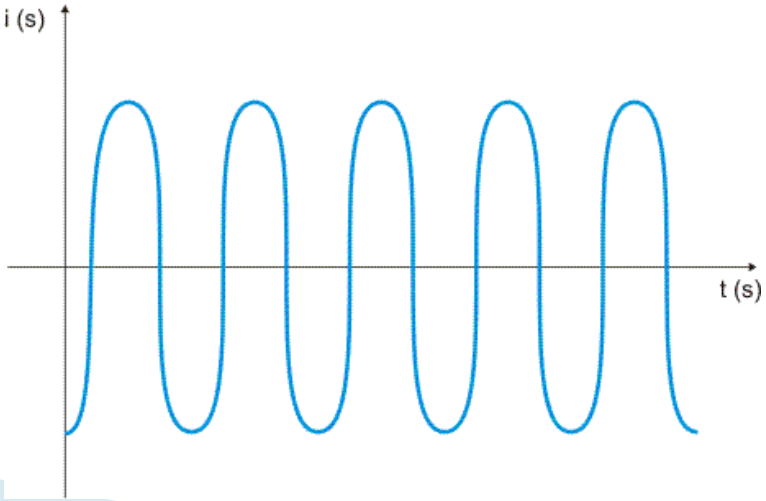


Corrente e Tensão

- Tipos de correntes elétricas
  - Corrente contínua
    - Os elétrons se movem sempre no mesmo sentido.
    - Grande parte dos equipamentos eletrônicos trabalha com corrente contínua.

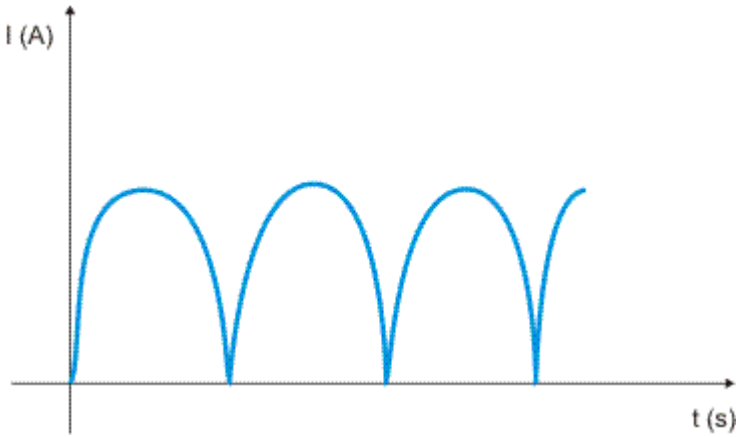
Corrente e Tensão

- Tipos de correntes elétricas
  - Corrente alternada
    - Na corrente alternada o sentido dos elétrons é invertido periodicamente, ou seja, ora a corrente é positiva, ora é negativa.
    - A energia elétrica que chega em nossas casas é do tipo corrente alternada.



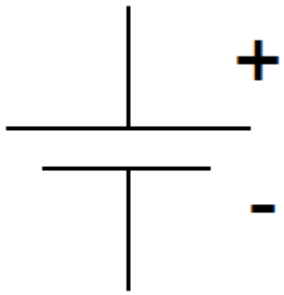
Corrente e Tensão

- Tipos de correntes elétricas
  - Corrente pulsante
    - Somente alterna o valor.
    - Corrente resultante da retificação da corrente alternada.



Corrente e Tensão

- Tensão Elétrica
  - É a força responsável por impulsionar os elétrons em um condutor.
  - A tensão é medida em Volts (V).
  - Exemplos:
    - Bateria/pilha de 9 volts
    - Tomada de 110 ou 220 volts



Resistência Elétrica

- Resistência elétrica é uma grandeza que indica o quanto um determinado condutor se opõe a passagem de corrente elétrica.
- Bons condutores de eletricidade possuem um número maior de elétrons livres, por esse motivo possuem uma baixa resistência elétrica.
- A resistência elétrica é medida em Ohms e o símbolo é a letra grega ômega – Ω.

UNIVERSIDADE FEDERAL DE SANTA CATARINA

LARM
Laboratório de Automação e Robótica Móvel

Condutores e Isolantes

- Condutores
  - São materiais que pouco se opõem à passagem de corrente elétrica.
  - Possuem baixa resistividade.
  - Os elétrons da camada de valência estão fracamente ligados ao núcleo e, assim, quebram facilmente suas ligações com o átomo, tornando-se livres para compor a corrente elétrica.

Condutores e Isolantes

- Condutores

ELEMENTO	RESISTIVIDADE (Ω-m)
Prata	$1,59 \times 10^{-8}$
Cobre	$1,72 \times 10^{-8}$
Ouro	$2,44 \times 10^{-8}$
Alumínio	$2,82 \times 10^{-8}$

Condutores e Isolantes

- Isolantes
  - Fazem muita oposição à passagem de corrente elétrica.
  - Possuem alta resistividade.
  - Os elétrons da camada de valência estão fortemente ligados ao núcleo e, por isso, precisam de uma energia muito maior para desfazer suas ligações com o átomo. Isso resulta em poucos elétrons livres para compor a corrente elétrica.

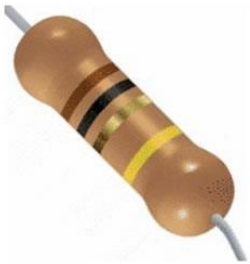
Condutores e Isolantes

- Isolantes

ELEMENTO	RESISTIVIDADE (Ω–m)
Vidro	10^9 a 10^{13}
Borracha	10^{13} a 10^{15}
Porcelana	3×10^{12}

Resistores

- O resistor é um componente eletrônico utilizado para limitar o fluxo de corrente.
- Os resistores podem ser do tipo fixo ou do tipo variável.
- Os resistores mais comuns são os de filme de carbono.



Resistores

- Um resistor fixo de filme de carbono possui em seu corpo faixas coloridas que indicam o seu valor de resistência.

- Onde:
  - A primeira faixa indica o primeiro número.
  - A segunda faixa indica o segundo número.
  - A terceira faixa indica o multiplicador.
  - A quarta faixa indica a tolerância.

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

LARM
Laboratório de Automação
e Robótica Móvel

Resistores

- Tabela de cores para a identificação de resistores

Cores	Faixa 1 e 2	Faixa 3	Faixa 4
Preto	0	1	–
Marrom	1	10	1%
Vermelho	2	100	2%
Laranja	3	1000	–
Amarelo	4	10.000	–
Verde	5	100.000	–
Azul	6	1.000.000	–
Violeta	7	10.000.000	–
Cinza	8	–	–
Branco	9	–	–
Ouro	–	–	5%
Prata	–	–	10%
Sem cor	–	–	20%

Resistores

- Exemplo:

Primeira faixa = laranja -> 3
Segunda faixa = laranja -> 3
Terceira faixa = marrom -> 10

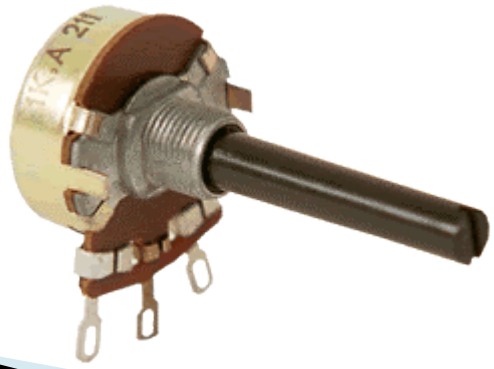
Resistor de: 33 * 10 = 330 Ω

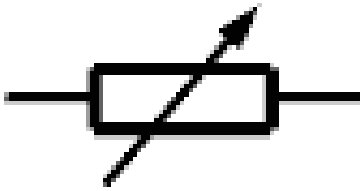
Quarta faixa = ouro -> tolerância de 5%

Resistor de: 313.5 Ω a 346.5 Ω

Resistores Variáveis

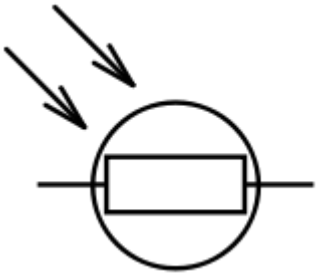
- Potenciômetro
  - É um resistor variável, ou seja, sua resistência pode ser ajustada conforme a necessidade da aplicação (circuito).
  - Um potenciômetro pode ser linear ou logarítmico, dependendo da função do ângulo de giro de seu eixo.





Resistores Variáveis

- LDR (*Light Dependent Resistor* – resistor dependente de luz)
  - O LDR ou foto resistor é um resistor variável que aumenta ou diminui a resistência de acordo com a intensidade da luz que está sendo incidida sobre ele.
  - É um tipo de sensor muito utilizado em robótica.
  - Quanto maior a luminosidade incidida sobre ele menor será a resistência.

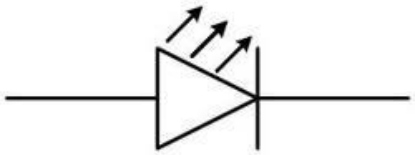


Resistores Variáveis

- Sensor de Temperatura
  - Os sensores de temperatura, termistores, podem ser do tipo NTC – *Negative Temperature Coefficient* ou PTC – *Positive Temperature Coefficient*.
  - Nos sensores do tipo NTC a resistência diminui com o aumento da temperatura.
  - Nos sensores do tipo PTC a resistência aumenta com o aumento da temperatura.

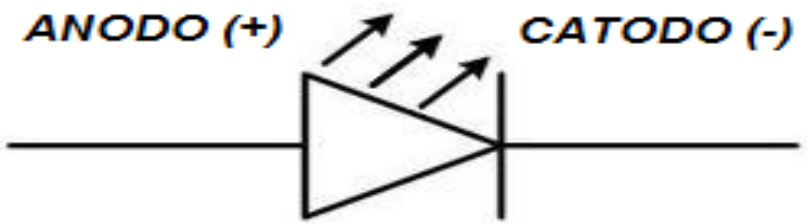
LED

- LED (*Light-Emitting Diode* – Diodo Emissor de Luz)
  - O LED emite luz visível (amarela, verde, vermelha, laranja ou azul) ou luz infravermelha.
  - Deve ser ligado em série com um resistor limitador de corrente.



LED

- LED (*Light-Emitting Diode* – Diodo Emissor de Luz)
  - O LED é um exemplo de componente eletrônico polarizado.
  - O posicionamento desse componente em um circuito precisa levar em conta os polos da fonte de alimentação.



LED

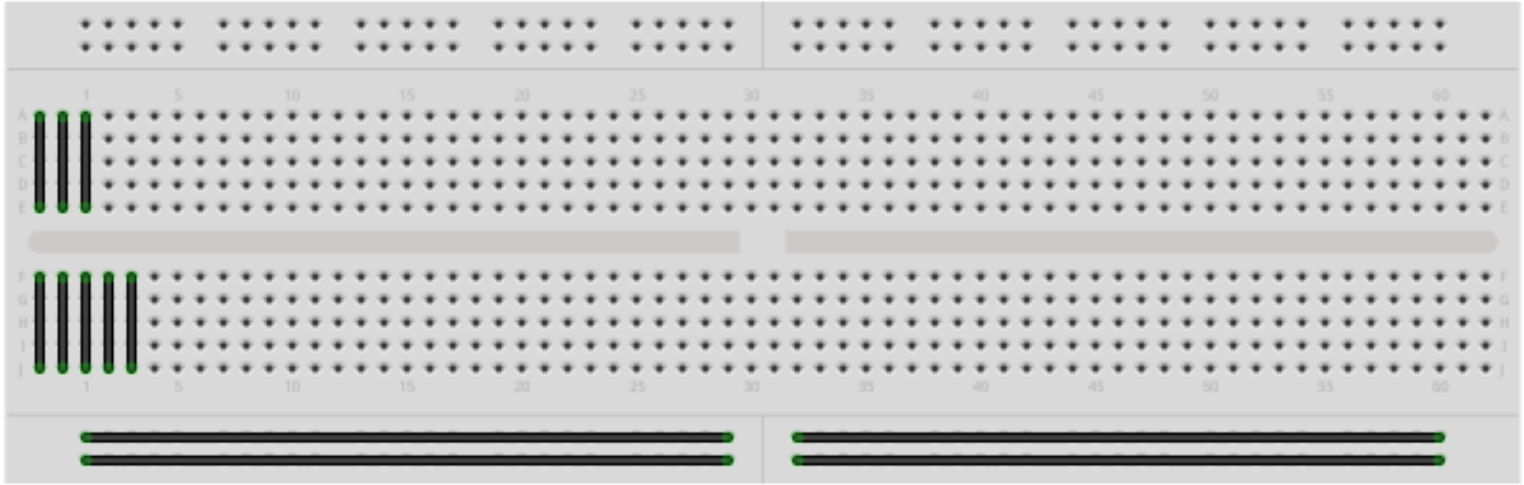
- Identificação dos Terminais

Protoboard (Matriz de Contatos)

- Ferramenta que auxilia no desenvolvimento de protótipos de circuitos eletrônicos.
- Torna desnecessária a soldagem de componentes eletrônicos em uma placa.
- É composta de furos que são interconectados por um material condutor localizado abaixo da camada de plástico.

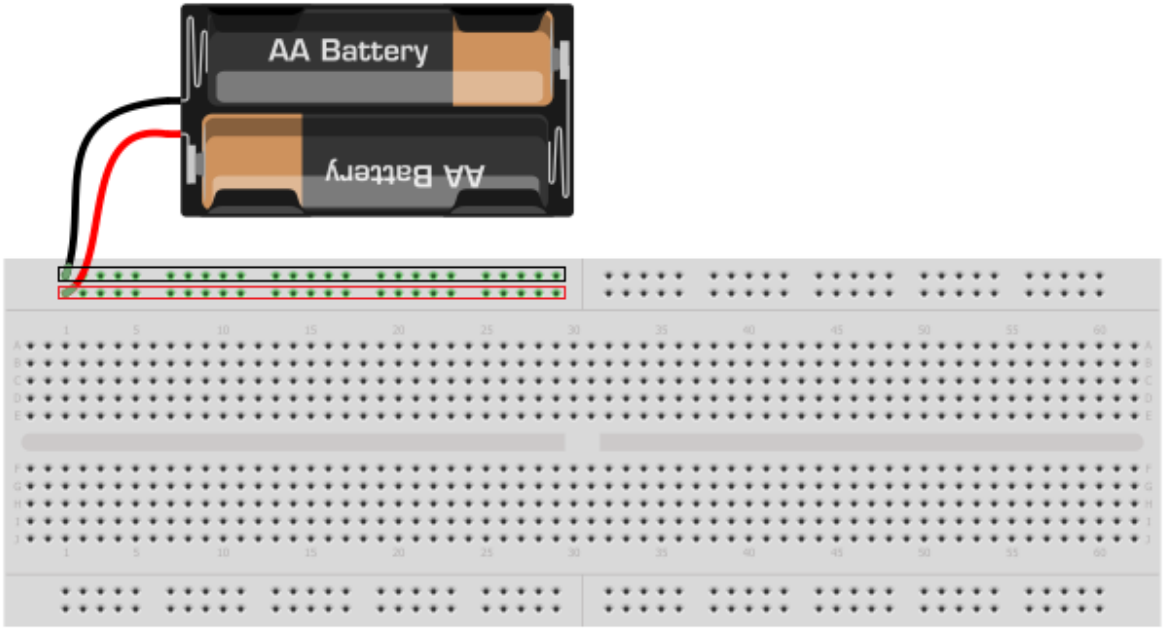
Protoboard (Matriz de Contatos)

- A figura ilustra a forma como os furos estão interconectados.



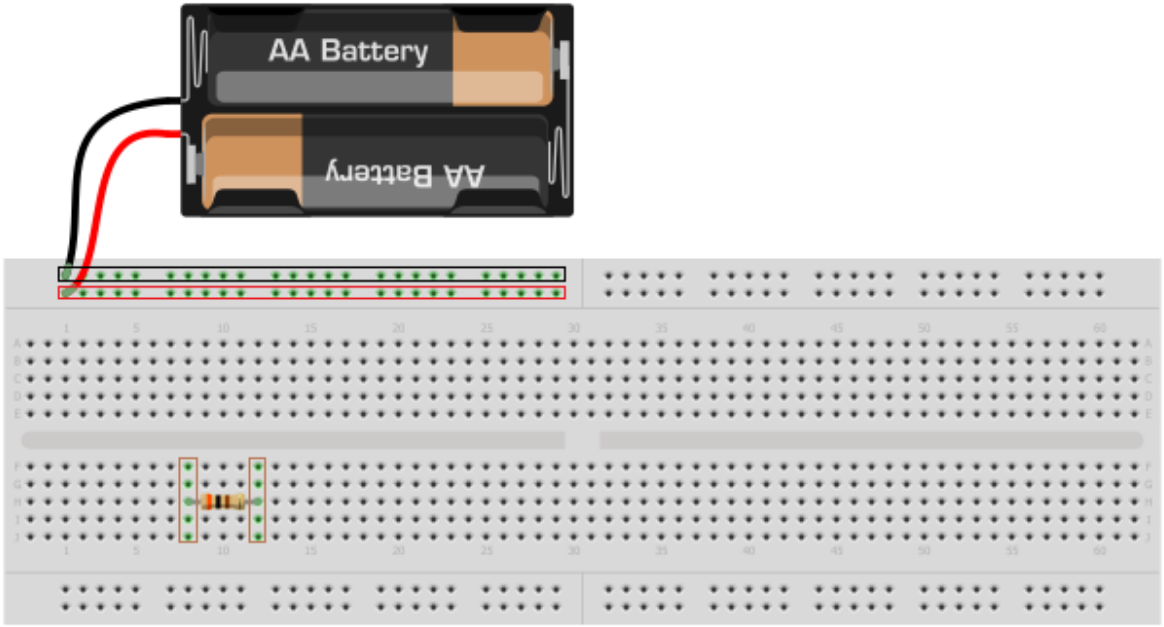
Protoboard (Matriz de Contatos)

- Passo-a-passo para a montagem de um pequeno circuito
  - Passo 1



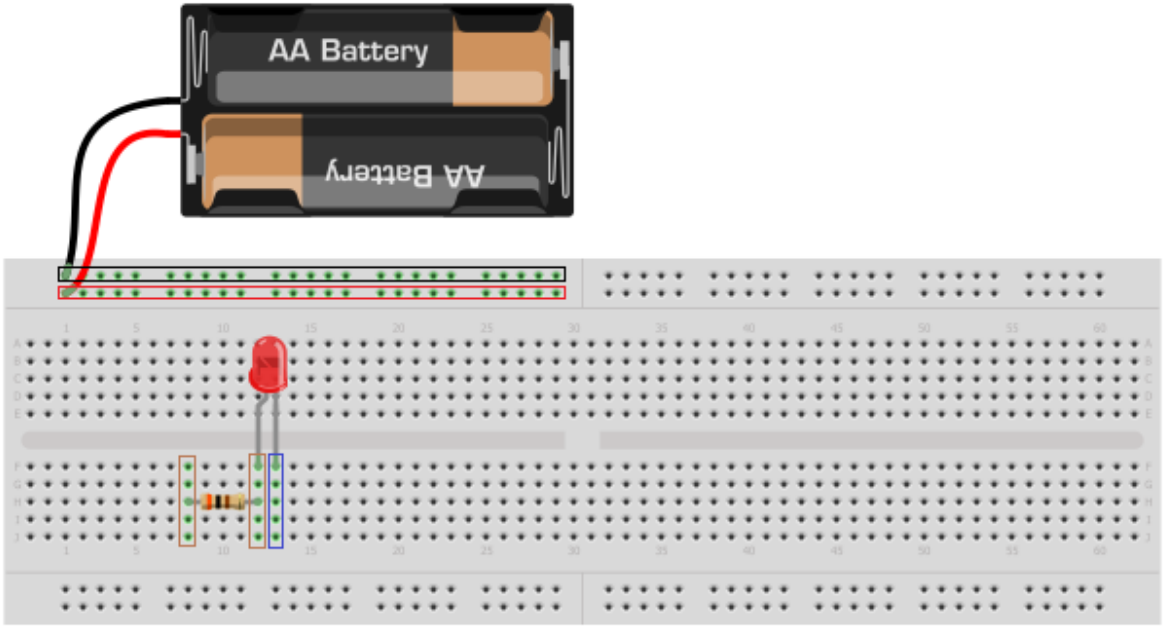
Protoboard (Matriz de Contatos)

- Passo-a-passo para a montagem de um pequeno circuito
  - Passo 2



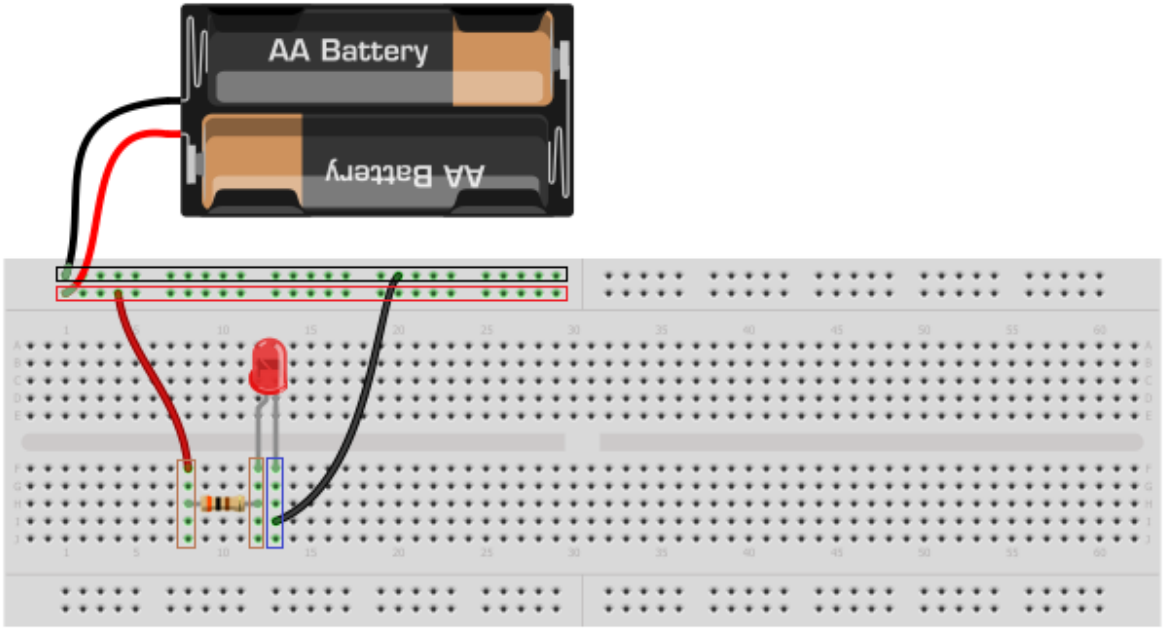
Protoboard (Matriz de Contatos)

- Passo-a-passo para a montagem de um pequeno circuito
  - Passo 3



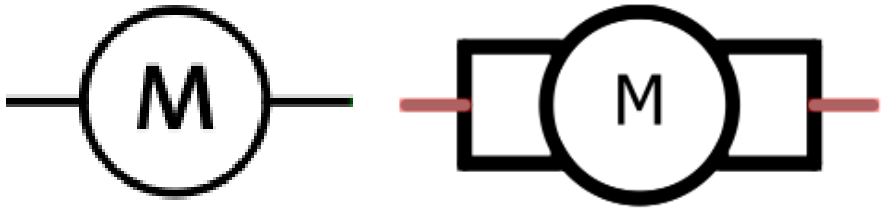
Protoboard (Matriz de Contatos)

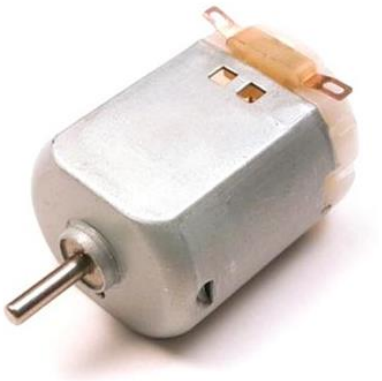
- Passo-a-passo para a montagem de um pequeno circuito
  - Passo 4



Motor DC (Motor de Corrente Contínua)

- Transforma energia elétrica em energia mecânica.
- O sentido de giro do rotor depende do sentido da corrente que percorre as bobinas do motor.
- Invertendo-se os polos da fonte de alimentação, inverte-se o sentido de giro do rotor.





UNIVERSIDADE FEDERAL DE SANTA CATARINA
LARM
Laboratório de Automação e Robótica Móvel

Motor DC (Motor de Corrente Contínua)

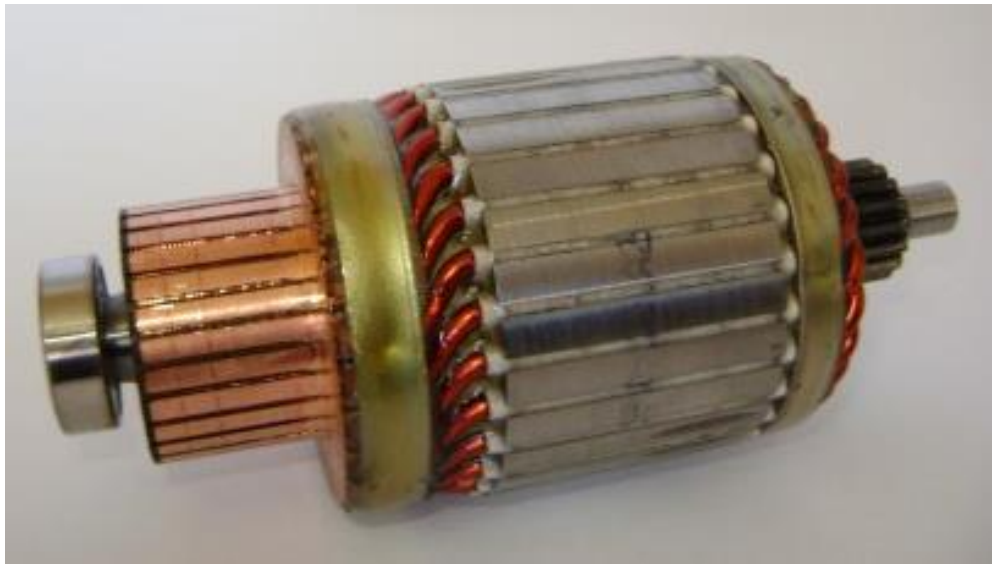
- Partes de um motor DC:
  - Rotor: Parte móvel do motor. Gira quando o motor é alimentado.
  - Estator: Parte estática do motor. É montado ao redor do rotor.
  - Escovas: Conectam os terminais ao comutador.
  - Comutador: Conecta o rotor à alimentação e faz a inversão do sentido da corrente, necessária para o correto funcionamento do motor.

Motor DC (Motor de Corrente Contínua)

- Rotor: Parte móvel do motor. Possui bobinas que geram um campo magnético quando o motor é alimentado.

Motor DC (Motor de Corrente Contínua)

- Rotor: Parte móvel do motor. Possui bobinas que geram um campo magnético quando o motor é alimentado.

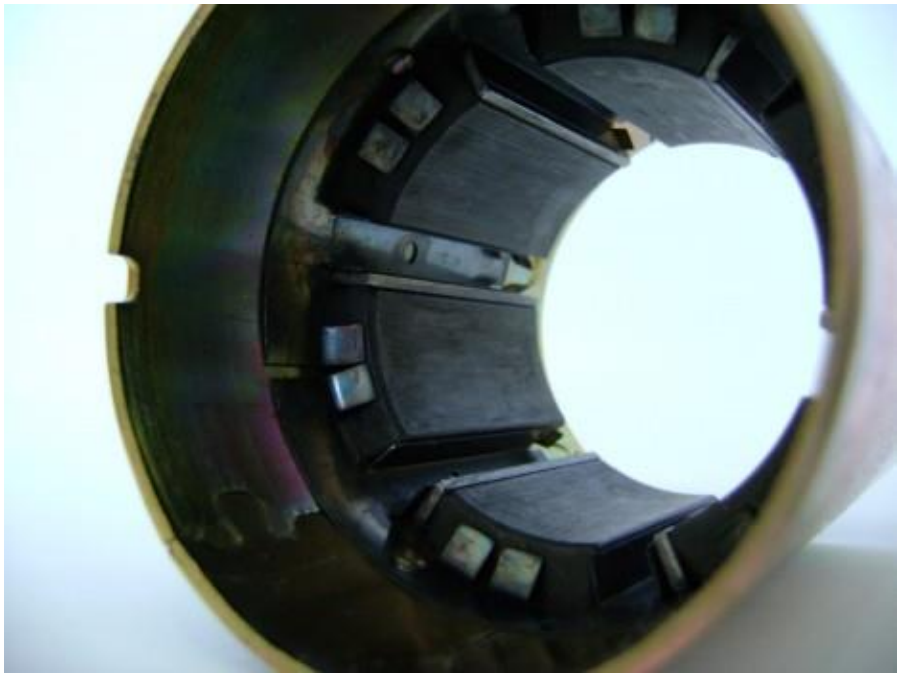


Motor DC (Motor de Corrente Contínua)

- Estator: Parte estática do motor. É montado ao redor do rotor. Composto de ímãs permanentes ou bobinas.

Motor DC (Motor de Corrente Contínua)

- Estator: Parte estática do motor. É montado ao redor do rotor. Composto de ímãs permanentes ou bobinas.

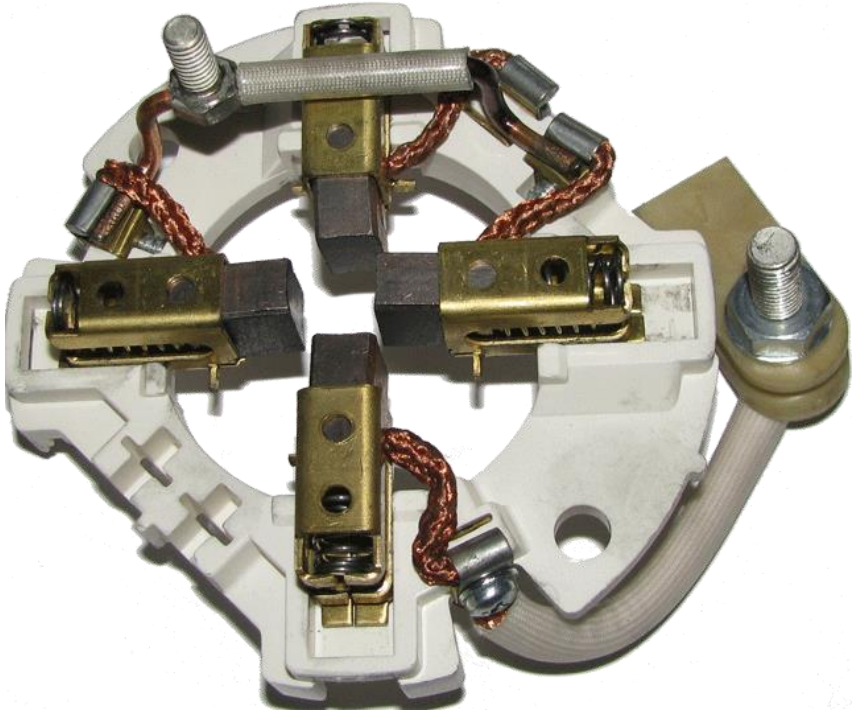


Motor DC (Motor de Corrente Contínua)

- Escovas: Conectam os terminais ao comutador.

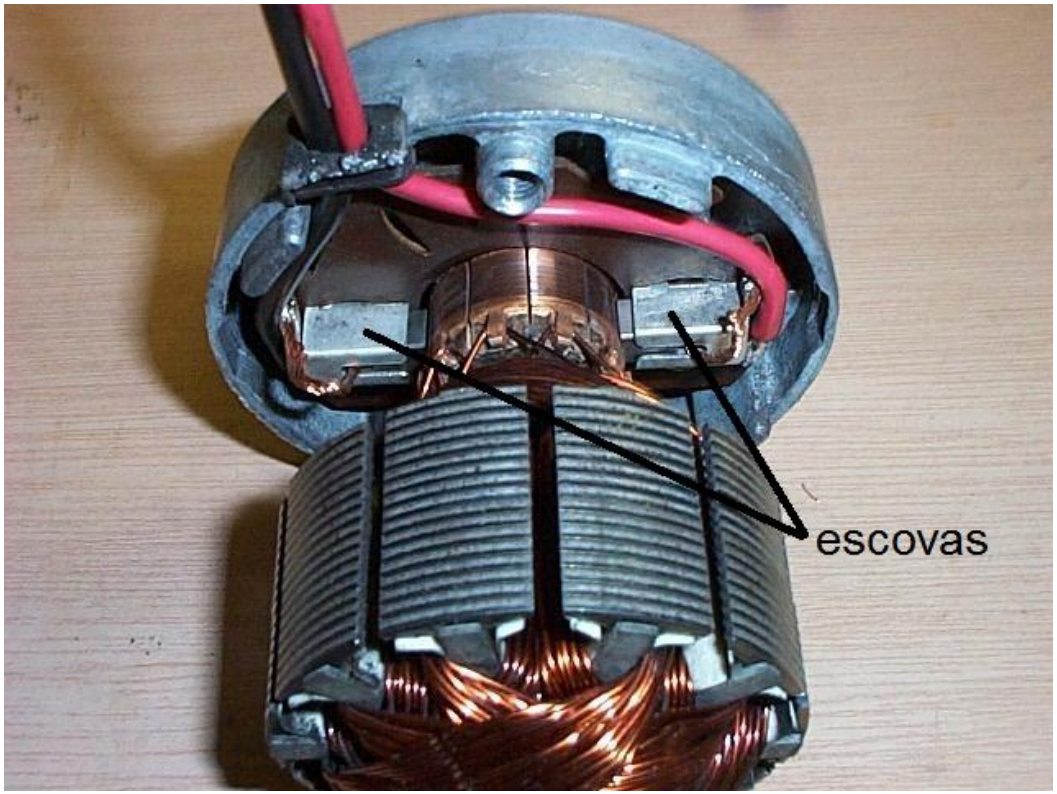
Motor DC (Motor de Corrente Contínua)

- Escovas: Conectam os terminais ao comutador.



Motor DC (Motor de Corrente Contínua)

- Escovas: Conectam os terminais ao comutador.

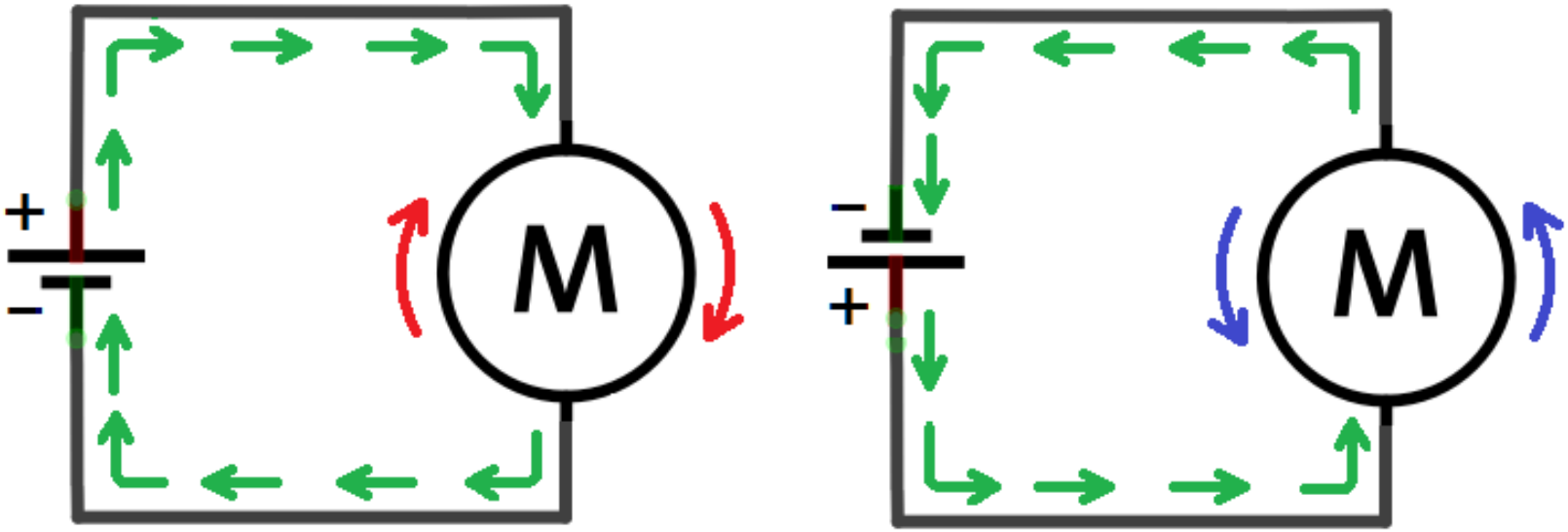


Motor DC (Motor de Corrente Contínua)

- Comutador: Conecta o rotor à alimentação e faz a inversão do sentido da corrente, necessária para o correto funcionamento do motor.

Motor DC (Motor de Corrente Contínua)

- Inversão do sentido de giro:

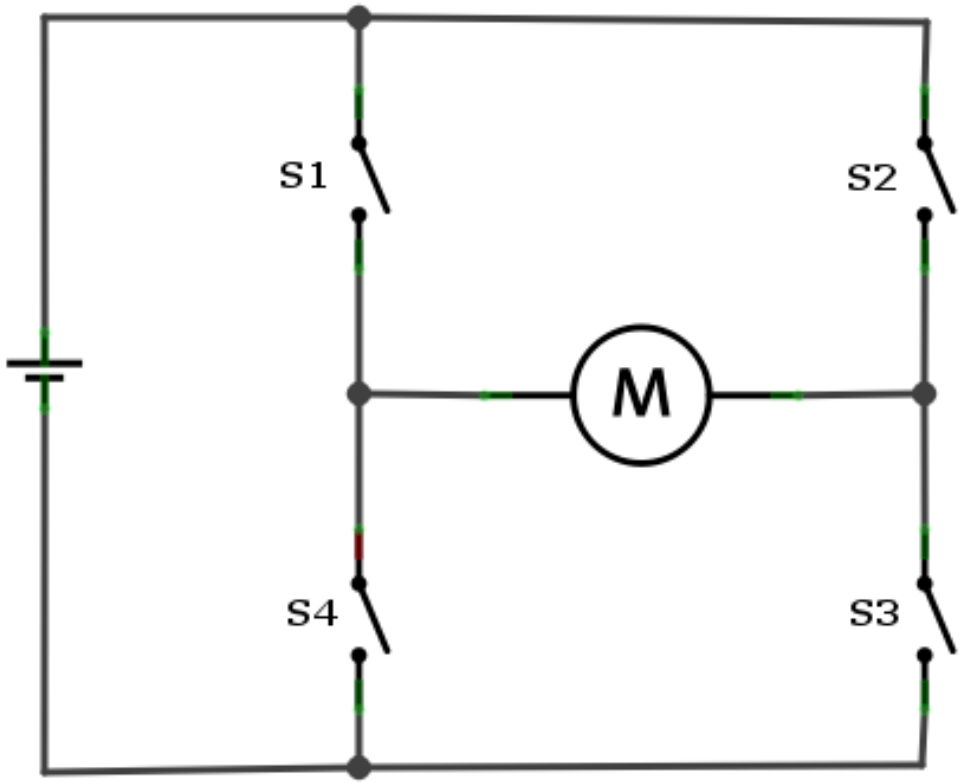


Ponte H

- A Ponte H é um circuito que permite a inversão do sentido de giro de um motor DC através da comutação de chaves eletrônicas.
- Pode ser implementada com chaves de contato, como push-buttons, ou transistores, que permitem o acionamento e inversão do sentido de giro de um motor através de sinais elétricos, sem a intervenção humana.

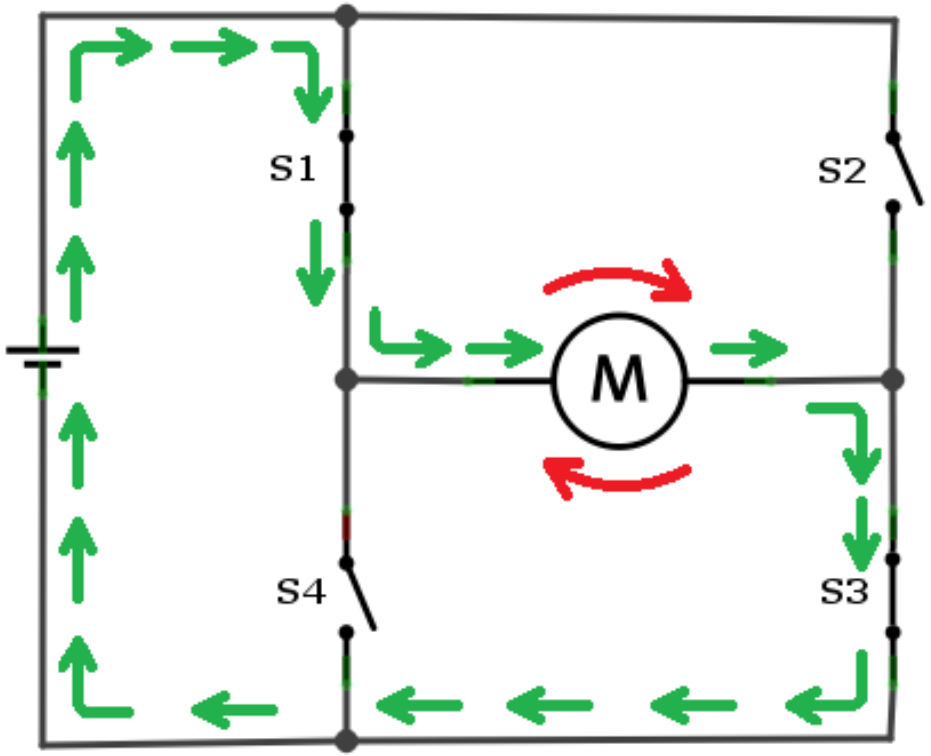
Ponte H

- Todas as chaves abertas – Motor parado.



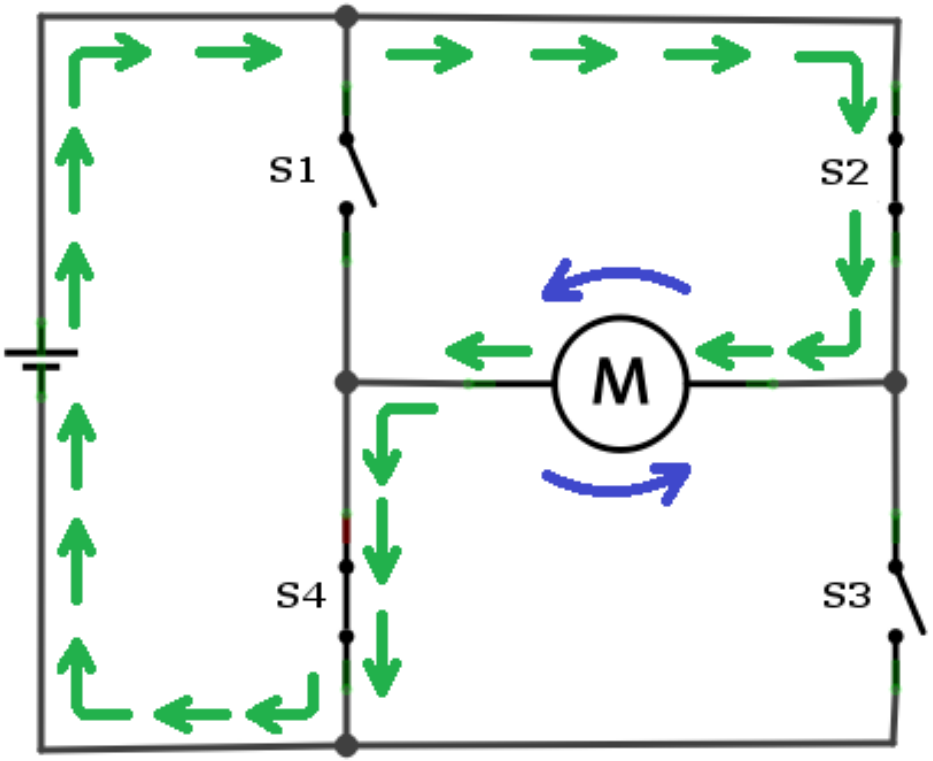
Ponte H

- S1-S3 fechadas e S2-S4 abertas - Rotor gira em um sentido.



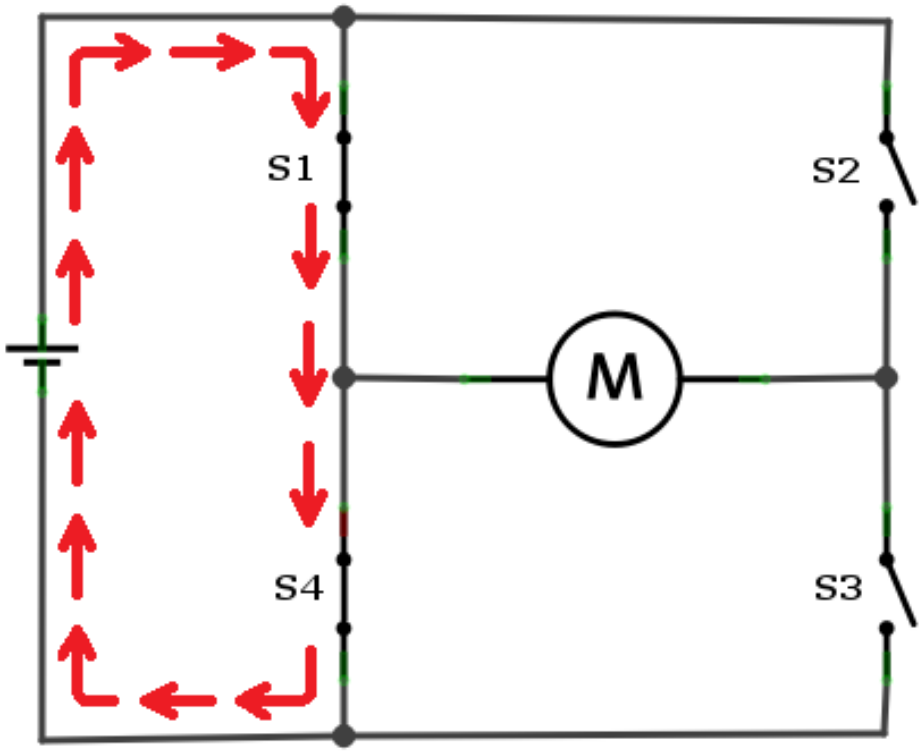
Ponte H

- S2-S4 fechadas e S1-S3 abertas - Rotor gira no sentido oposto ao anterior.



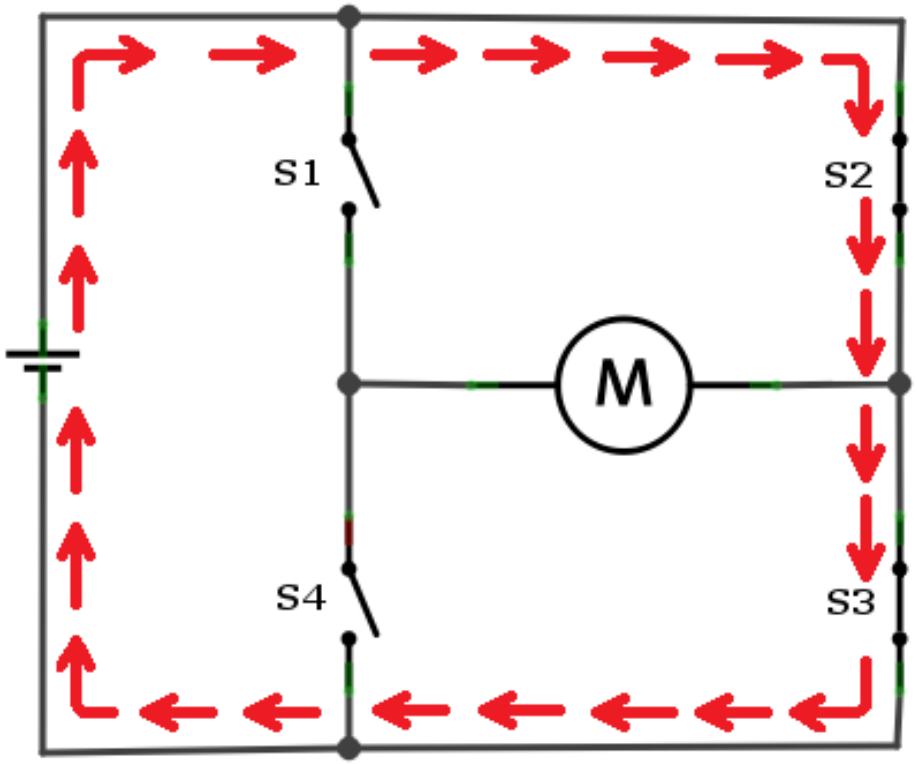
Ponte H

- S1-S4 fechadas - Essa configuração não deve ocorrer. *** CURTO CIRCUITO ***



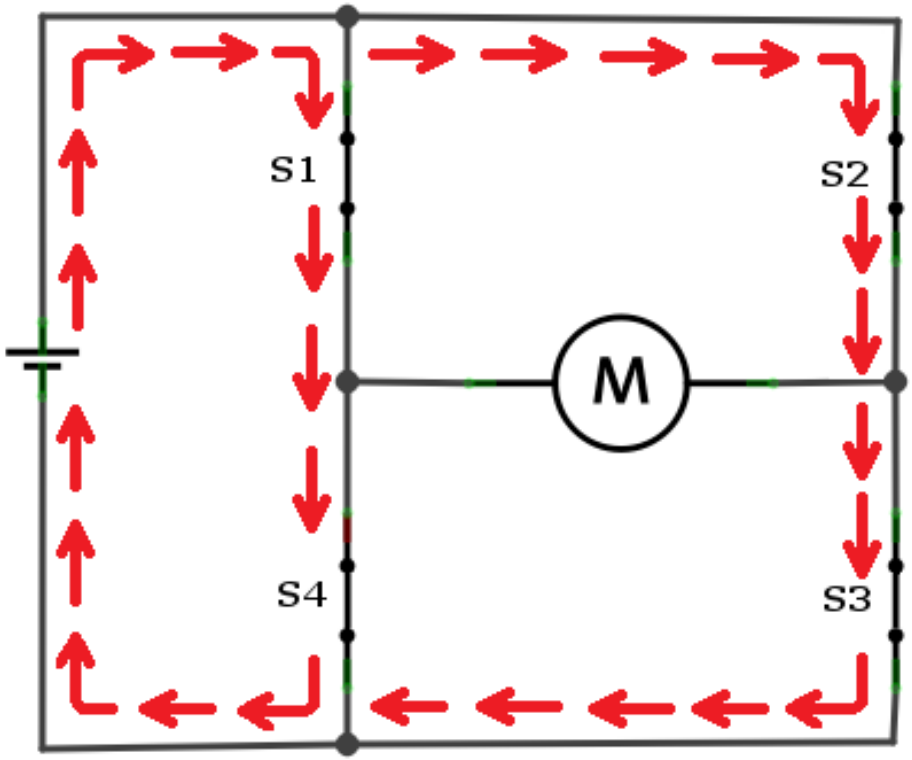
Ponte H

- S2-S3 fechadas - Também não deve ocorrer. *** CURTO CIRCUITO ***



Ponte H

- Todas as chaves fechadas – Também não deve ocorrer. *** CURTO CIRCUITO ***



UNIVERSIDADE FEDERAL
DE SANTA CATARINA

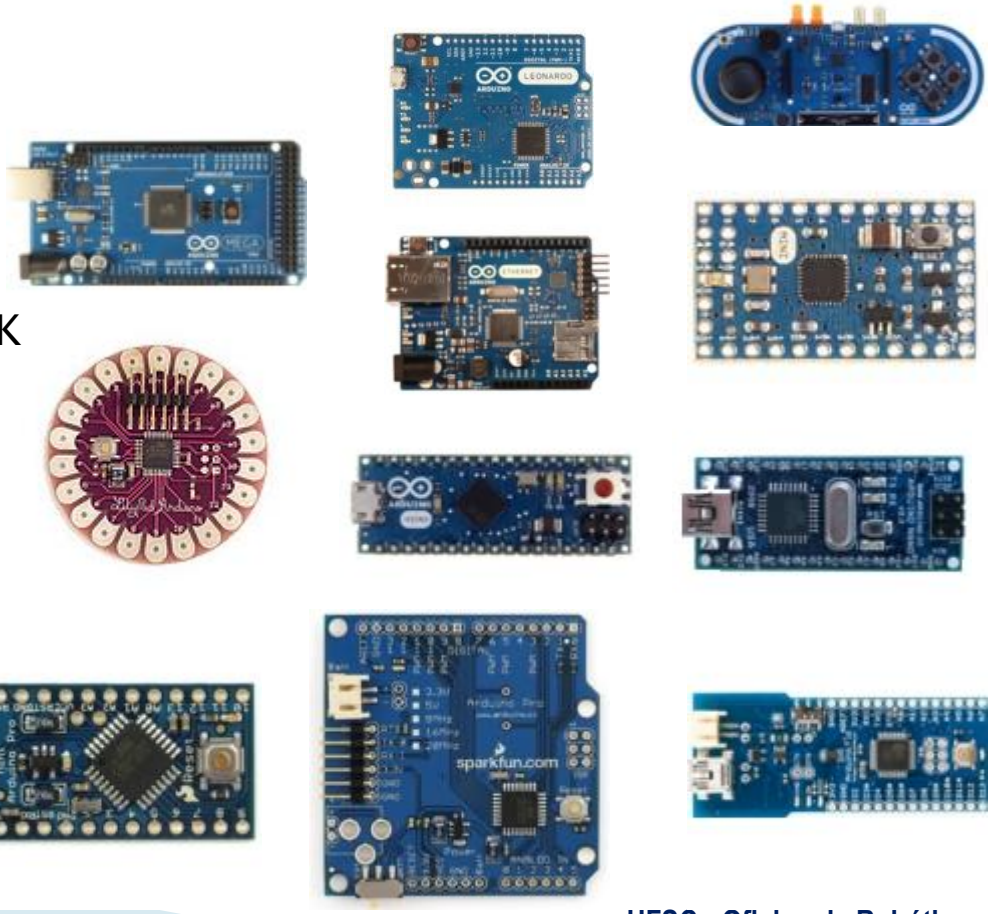
LARM
Laboratório de Automação
e Robótica Móvel

Introdução ao Arduino

- O Arduino é uma plataforma utilizada para prototipação de circuitos eletrônicos.
- O projeto do Arduino teve início em 2005 na cidade de Ivrea, Itália.
- O Arduino é composto por uma placa com microcontrolador Atmel AVR e um ambiente de programação baseado em Wiring e C++.
- Tanto o hardware como o ambiente de programação do Arduino são livres, ou seja, qualquer pessoa pode modificá-los e reproduzi-los.
- O Arduino também é conhecido como plataforma de computação física.

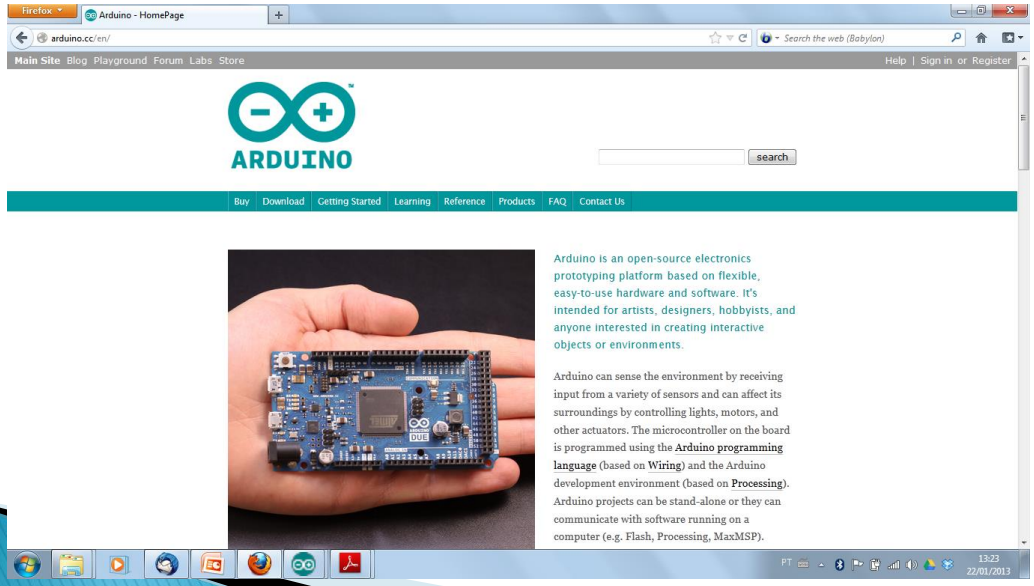
Introdução ao Arduino

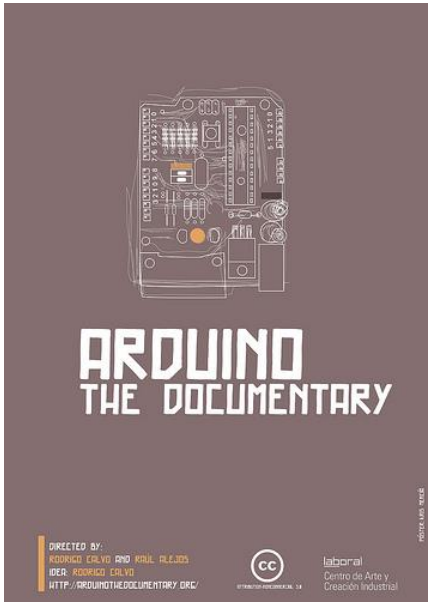
- Tipos de Arduino
  - Existem vários tipos de Arduino com especificidades de hardware. O site oficial do Arduino lista os seguintes tipos:
    - Arduino UNO
    - Arduino Leonardo
    - Arduino Due
    - Arduino Esplora
    - Arduino Mega
    - Arduino Mega ADK
    - Arduino Ethernet
    - Arduino Mini
    - Arduino LilyPad
    - Arduino Micro
    - Arduino Nano
    - Arduino ProMini
    - Arduino Pro
    - Arduino Fio



Introdução ao Arduino

- Referências na WEB:
  - O site oficial do Arduino é http://arduino.cc
  - Um documentário sobre o Arduino pode ser assistido em: http://arduinothedocumentary.org/



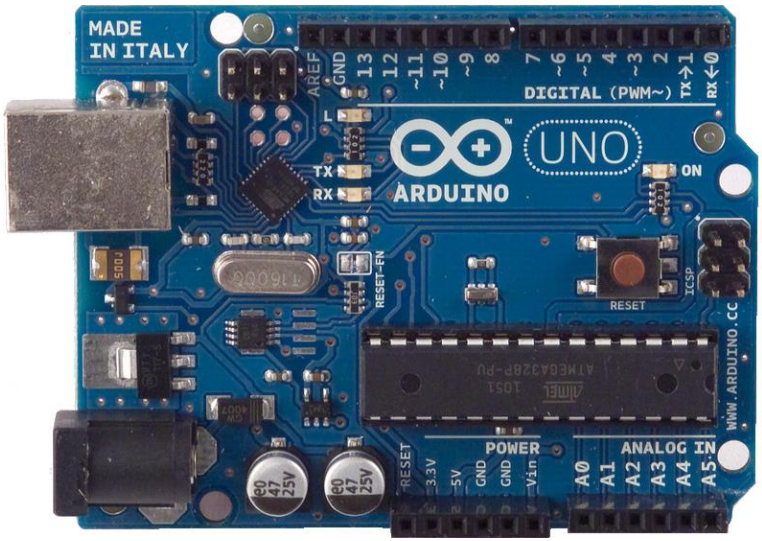


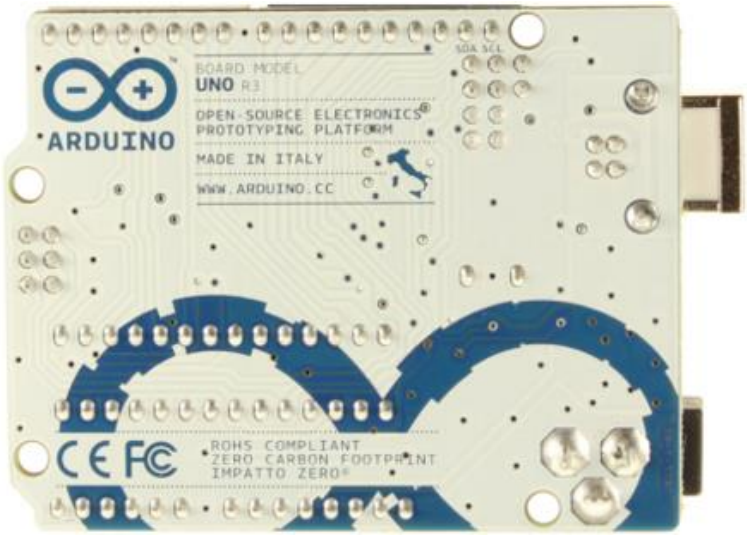
UNIVERSIDADE FEDERAL DE SANTA CATARINA

LARM
Laboratório de Automação e Robótica Móvel

Arduino UNO

- Vista da placa do Arduino UNO Rev 3 (frente e verso)





UNIVERSIDADE FEDERAL
DE SANTA CATARINA

LARM
Laboratório de Automação
e Robótica Móvel

Arduino UNO

- Características
  - Microcontrolador: ATmega328
  - Tensão de operação: 5V
  - Tensão recomendada (entrada): 7-12V
  - Limite da tensão de entrada: 6-20V
  - Pinos digitais: 14 (seis pinos com saída PWM)
  - Entrada analógica: 6 pinos
  - Corrente contínua por pino de entrada e saída: 40 mA
  - Corrente para o pino de 3.3 V: 50 mA
  - Quantidade de memória FLASH: 32 KB (ATmega328) onde 0.5 KB usado para o bootloader
  - Quantidade de memória SRAM: 2 KB (ATmega328)
  - Quantidade de memória EEPROM: 1 KB (ATmega328)
  - Velocidade de clock: 16 MHz

UNIVERSIDADE FEDERAL DE SANTA CATARINA
LARM
Laboratório de Automação e Robótica Móvel

Arduino UNO

- Alimentação
  - O Arduino UNO pode ser alimentado pela porta USB ou por uma fonte externa DC.
  - A recomendação é que a fonte externa seja de 7 V a 12 V e pode ser ligada diretamente no conector de fonte ou nos pinos Vin e Gnd.

Programação

- O microcontrolador do Arduino é um computador que segue instruções detalhadas dadas por seres humanos.
- Para que o Arduino execute determinada tarefa, precisamos "ensiná-lo" a executar essa tarefa, passo a passo.
- Os humanos passam instruções para o Arduino escrevendo programas.
- Um programa é uma sequência de instruções codificadas em uma linguagem de programação.

Programação

- Linguagens de Programação
  - Uma linguagem de programação é um meio utilizado para se comunicar com computadores, inclusive o Arduino, de um modo relativamente simples para os seres humanos.
  - Os computadores só conseguem executar instruções a eles enviadas na forma de sequências de 0's e 1's (linguagem de máquina).
  - Passar instruções a um computador usando linguagem de máquina seria extremamente complexo e tedioso, por isso usamos linguagens de programação mais próximas da linguagem natural.

Programação

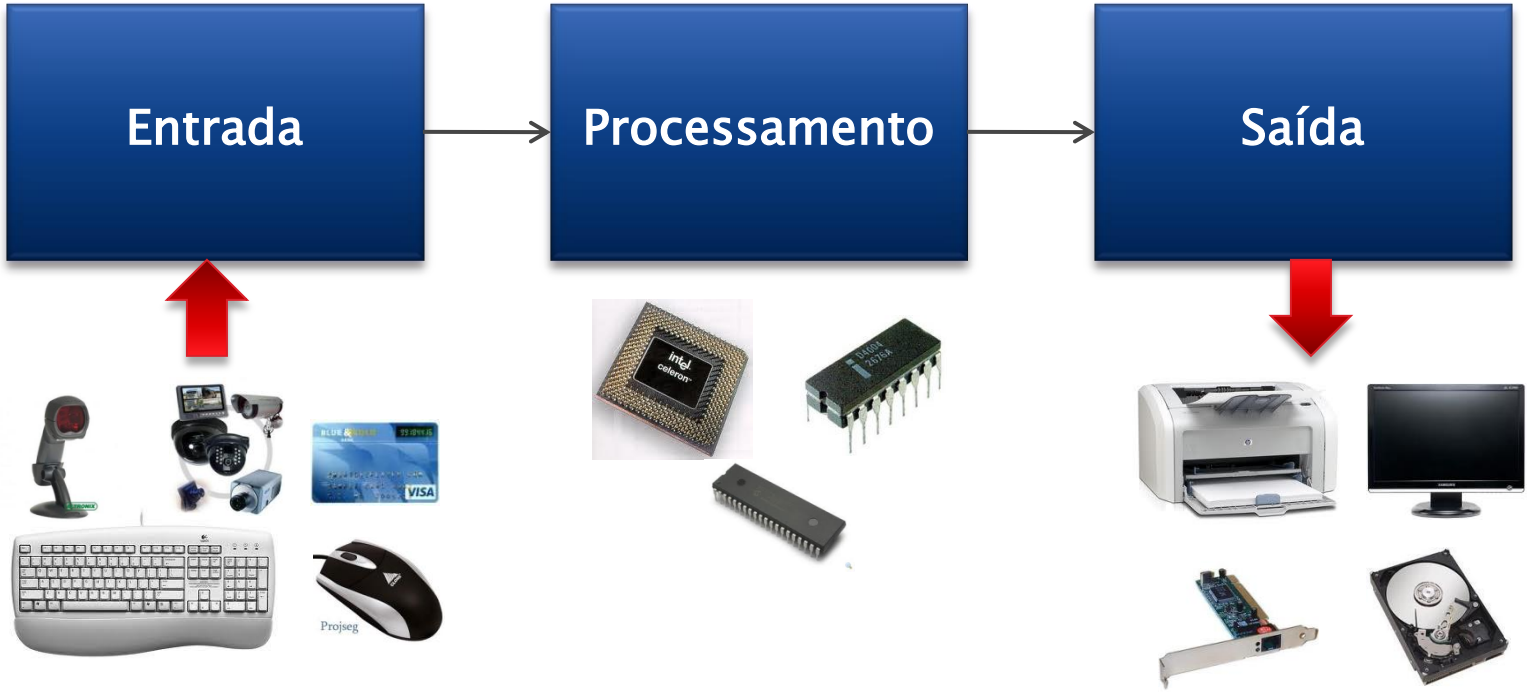
- Algoritmo
  - Sequência de passos que visa atingir um objetivo bem definido.

Ingridienti:
5 den di ái
3 cuié di ói
1 cabêss di repôi
1 cuié di mastumati
Sali a gosto

Mé qui fais?!
Casca u ái, pica u ái e soca o ái cum sali. Quenta o ói; foga o ái no ói quentim.
Pica o repôi bemmm finimm, foga o repôi.
Poim a mastumati mexi ca cuié pra fazê o moi.
Prontim!

Programação

- Processamento de dados

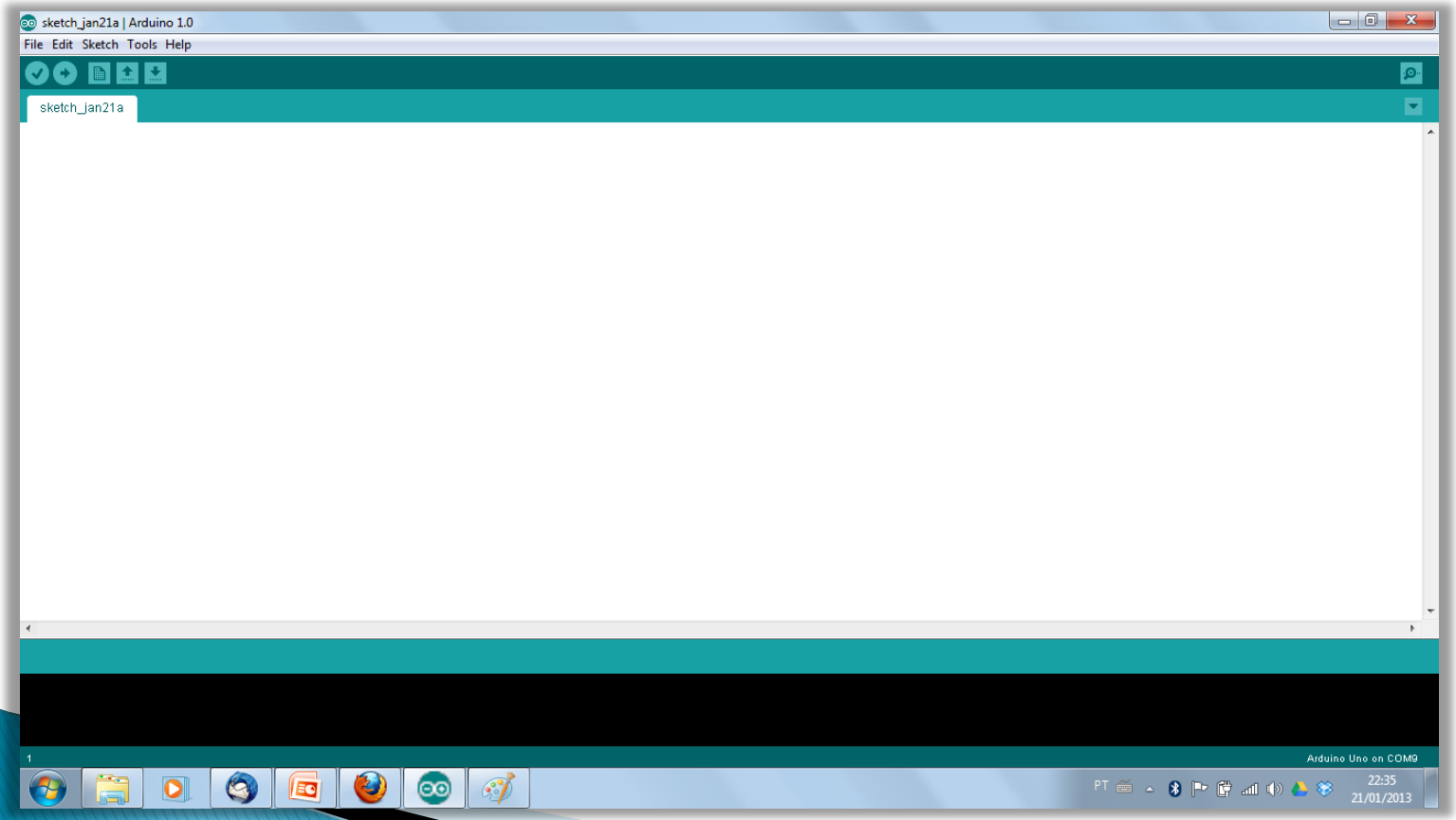


Ambiente de Desenvolvimento

- O ambiente de desenvolvimento do Arduino (IDE) é gratuito e pode ser baixado no seguinte endereço: arduino.cc.
- As principais funcionalidades do IDE do Arduino são:
  - Escrever o código do programa
  - Salvar o código do programa
  - Compilar um programa
  - Transportar o código compilado para a placa do Arduino

Ambiente de Desenvolvimento

- Interface principal do ambiente de desenvolvimento



Funções *setup()* e *loop()*

- As duas principais partes (funções) de um programa desenvolvido para o Arduino são:
  - setup(): onde devem ser definidas algumas configurações iniciais do programa. Executa uma única vez.
  - loop(): função principal do programa. Fica executando indefinidamente.
- Todo programa para o Arduino deve ter estas duas funções.

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

LARM
Laboratório de Automação
e Robótica Móvel

Funções *setup()* e *loop()*

- Formato das funções *setup()* e *loop()*

setuploop

```
void setup()
{

}

void loop()
{

}
```

Funções *setup()* e *loop()*

- Primeiro programa: Blink LED

blink_led

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

LARM
Laboratório de Automação
e Robótica Móvel

Comentários

- Muitas vezes é importante comentar alguma parte do código do programa.
- Existem duas maneiras de adicionar comentários a um programa em Arduino.
  - A primeira é usando //, como no exemplo abaixo:
    - // Este é um comentário de linha
  - A segunda é usando /* */, como no exemplo abaixo:
    - /* Este é um comentário de bloco. Permite acrescentar comentários com mais de uma linha */

- **Nota:**
  - Quando o programa é compilado os comentários são automaticamente suprimidos do arquivo executável, aquele que será gravado na placa do Arduino.

Comentários

- Primeiro programa comentado

comentarios

```
/*****************************************************
*       OFICINA DE ROBÓTICA - LARM - UFSC            *
*                                                    *
* Blink Led: Primeiro programa em Arduino.           *
*            Pisca um led conectado à porta 13.      *
*****************************************************/

// função usada para configurações iniciais
void setup()
{
  pinMode(13, OUTPUT);
}

// principal função do programa - laço infinito
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Portas digitais

- O Arduino possui tanto portas digitais quanto portas analógicas.
- As portas servem para comunicação entre o Arduino e dispositivos externos, por exemplo: ler um botão, acender um led ou uma lâmpada.
- Conforme já mencionado, o Arduino UNO, possui 14 portas digitais e 6 portas analógicas (que também podem ser utilizadas como portas digitais).

Portas digitais

- As portas digitais trabalham com valores bem definidos. No caso do Arduino esses valores são 0V e 5V.
- 0V indica a ausência de um sinal e 5V indica a presença de um sinal.
- Para escrever em uma porta digital basta utilizar a função digitalWrite(*pino, estado*).
- Para ler um valor em uma porta digital basta utilizar a função digitalRead(*pino*).

UNIVERSIDADE FEDERAL DE SANTA CATARINA
LARM
Laboratório de Automação e Robótica Móvel

Portas digitais

- As portas digitais são usadas para entrada e saída de dados.
- Para definir se uma porta será usada para entrada ou para saída de dados, é necessário explicitar essa situação no programa.
- A função pinMode(*pino, estado*) é utilizada para definir se a porta será de entrada ou saída de dados.
- Exemplos:
  - Define que a porta 13 será de saída
    - pinMode(13, OUTPUT)
  - Define que a porta 7 será de entrada
    - pinMode(7, INPUT)

Constantes

- Um dado é constante quando não sofre nenhuma variação no decorrer do tempo.
- Do início ao fim do programa o valor permanece inalterado.
- Exemplos:
  - 10
  - “Bata antes de entrar!”
  - -0,58

Constantes

- A criação de constantes no **Arduino** pode ser feita de duas maneiras:
  - Usando a palavra reservada **const**
    - Exemplo:
      - const int x = 100;
  - Usando a palavra reservada **define**
    - Exemplo:
      - #define X 100

Constantes

- No Arduino existem algumas constantes previamente definidas e são consideradas palavras reservadas.
- As constantes definidas são:
  - true – indica valor lógico verdadeiro
  - false – indica valor lógico falso
  - HIGH – indica que uma porta está ativada, ou seja, está em 5V.
  - LOW – indica que uma porta está desativada, ou seja, está em 0V.
  - INPUT – indica que uma porta será de entrada de dados.
  - OUTPUT – indica que uma porta será de saída de dados.

Variáveis

- Variáveis são lugares (posições) na memória principal que servem para armazenar dados.
- As variáveis são acessadas através de um identificador único.
- O conteúdo de uma variável pode variar ao longo do tempo durante a execução de um programa.
- Uma variável só pode armazenar um valor a cada instante.
- Um identificador para uma variável é formado por um ou mais caracteres, obedecendo a seguinte regra: o primeiro caractere deve, obrigatoriamente, ser uma letra.

Variáveis

- ATENÇÃO!!!
  - Um identificador de uma variável ou constante não pode ser formado por caracteres especiais ou palavras reservadas da linguagem.

Tipos de Dados

- Tipos de dados definem:
  - A quantidade de memória que uma variável ou constante irá ocupar;
  - As operações que podem ser executadas sobre uma variável ou constante de determinado tipo;
  - A faixa de valores que uma variável ou constante pode armazenar;
  - O modo como o valor armazenado será interpretado.

Tipos de Dados

- Tipos de Variáveis no Arduino

Tipo	Definição
void	Indica tipo indefinido. Usado geralmente para informar que uma função não retorna nenhum valor.
boolean	Os valores possíveis são true (1) e false (0). Ocupa um byte de memória.
char	Ocupa um byte de memória. Pode ser uma letra ou um número. A faixa de valores válidos é de -128 a 127.
unsigned char	O mesmo que o **char**, porém a faixa de valores válidos é de 0 a 255.
byte	Ocupa 8 bits de memória. A faixa de valores é de 0 a 255.
int	Armazena números inteiros e ocupa 16 bits de memória (2bytes). A faixa de valores é de -32.768 a 32.767.
unsigned int	O mesmo que o **int**, porém a faixa de valores válidos é de 0 a 65.535.
word	O mesmo que um **unsigned int**.

Tipos de Dados

- Tipos de Variáveis no Arduino

Tipo	Definição
long	Armazena números de até 32 bits (4 bytes). A faixa de valores é de -2.147.483.648 até 2.147.483.647.
unsigned long	O mesmo que o **long**, porém a faixa de valores é de 0 até 4.294.967.295.
short	Armazena número de até 16 bits (2 bytes). A faixa de valores é de -32.768 até 32.767.
float	Armazena valores de ponto flutuante (com vírgula) e ocupa 32 bits (4 bytes) de memória. A faixa de valores é de -3.4028235E+38 até 3.4028235E+38
double	O mesmo que o **float**.

Monitor Serial

- O monitor serial é utilizado para comunicação entre o Arduino e o computador (PC).
- O monitor serial pode ser aberto no menu *tools* opção *serial monitor*, ou pressionando as teclas CTRL+SHIFT+M.
- As principais funções do monitor serial são: *begin()*, *read()*, *write()*, *print()*, *println()* e *available()*.

Monitor Serial

- Algumas funções bastante usadas:
  - *begin()*: inicializa a comunicação entre o Arduino e um computador;
  - *read()*: recebe caracteres inseridos no monitor serial;
  - *print()*: imprime caracteres no monitor serial;
  - *println()*: imprime caracteres no monitor serial, mas causa uma quebra de linha no final;
  - *available()*: retorna o número de bytes disponíveis no buffer de leitura do monitor serial.

Monitor Serial

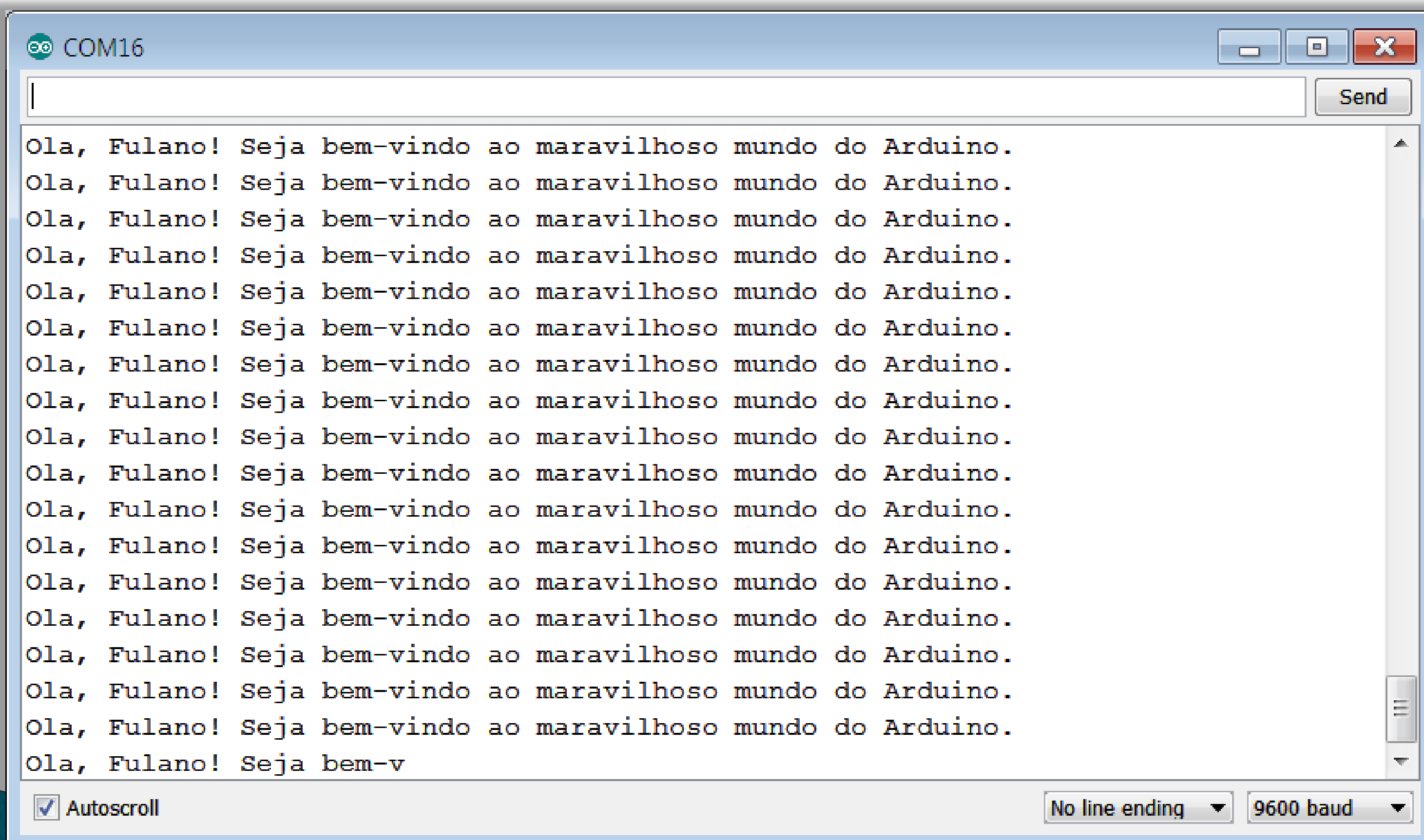
- Imprimindo uma mensagem no monitor serial

monitor_serial §

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

void loop()
{
  Serial.print("Ola, Fulano! Seja bem-vindo ao ");
  Serial.println("maravilhoso mundo do Arduino.");
}
```

Monitor Serial

- Saída no monitor serial



UNIVERSIDADE FEDERAL DE SANTA CATARINA

LARM
Laboratório de Automação e Robótica Móvel

Operadores

- Em uma linguagem de programação existem vários operadores que permitem operações do tipo:
  - Aritmética
  - Relacional
  - Lógica
  - Composta

Operadores

- Operadores aritméticos

Símbolo	Função
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão inteira)

Operadores

- Operadores relacionais

Símbolo	Função
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
==	Igual
!=	Diferente

Operadores

- Operadores lógicos

Símbolo	Função
&&	E (and)
||	OU (or)
!	Não (not)

Operadores

- Operadores compostos

Símbolo	Função
++	Incremento
--	Decremento
+=	Adição com atribuição
-=	Subtração com atribuição
*=	Multiplicação com atribuição
/=	Divisão com atribuição

Operadores

- Operador de Atribuição
  - A atribuição de valores a variáveis e constantes é feita com o uso do operador de atribuição (=).
  - O operador de atribuição coloca o valor situado à sua direita dentro do objeto localizado à sua esquerda.
  - Exemplos:
    - int valor = 100;
    - const float pi = 3.14;
  - **Atenção!!!**
    - O operador de atribuição não vale para o comando ***#define***.

Operadores

- Usando o operador de atribuição

atribuicao

```
int numero = 1; // inicialização

void setup()
{
  Serial.begin(9600);
  Serial.print("A variavel 'numero' vale: ");
  Serial.println(numero);
  delay(2000);
  numero = 5; // atribuição
  Serial.print("Agora a variavel 'numero' vale: ");
  Serial.println(numero);
  delay(2000);
}

void loop()
{
  numero = numero + 1; // atribuição
  Serial.print("Agora a variavel 'numero' vale: ");
  Serial.println(numero);
  delay(2000);
}
```

Comandos de Seleção

- Em vários momentos em um programa precisamos verificar uma determinada condição afim de selecionar uma ação ou ações que serão executadas.
- Um comando de seleção também é conhecido por desvio condicional, ou seja, dada um condição, uma parte do programa é executada.
- Os comandos de seleção podem ser do tipo:
  - Seleção simples
  - Seleção composta
  - Seleção de múltipla escolha

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

LARM
Laboratório de Automação
e Robótica Móvel

Comandos de Seleção

- Seleção simples
  - Um comando de seleção simples avalia uma condição, ou expressão, para executar uma ação ou conjunto de ações.
  - No Arduino o comando de seleção simples é:

```
if (expr) {
   cmd
}
```

  - onde:
    - ***expr*** – representa uma expressão a ser avaliada que pode ser do tipo lógica, relacional ou aritmética. O resultado da avaliação de uma expressão é sempre um valor lógico.
    - ***cmd*** – comando(s) a ser executado.

Comandos de Seleção

- Seleção simples
  - Acendendo leds pelo monitor serial

selecao_simples

```
const int led_verde = 5;
const int led_amarelo = 6;
const int led_vermelho = 7;
char led;

void setup()
{
  pinMode(led_verde, OUTPUT);
  pinMode(led_amarelo, OUTPUT);
  pinMode(led_vermelho, OUTPUT);
  Serial.begin(9600);
}
```

```
void loop()
{
  led = Serial.read();

  if (led == 'G') {
    digitalWrite(led_verde, HIGH);
  }
  if (led == 'Y') {
    digitalWrite(led_amarelo, HIGH);
  }
  if (led == 'R') {
    digitalWrite(led_vermelho, HIGH);
  }
}
```

Comandos de Seleção

- Seleção simples
  - Verificando se há caracteres no buffer de leitura

serial_available

```
char caractere;

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

void loop()
{
   if (Serial.available()) {
      caractere = Serial.read();
      Serial.print("Caractere lido: ");
      Serial.println(caractere);
   }
}
```

Comandos de Seleção

- Seleção composta
  - Um comando de seleção composta é complementar ao comando de seleção simples.
  - O objetivo é executar um comando mesmo que a expressão avaliada pelo comando *if (expr)* retorne um valor falso.
  - No Arduino o comando de seleção composta é:

```
if (expr) {
   cmd;
}
else {
    cmd;
}
```

  - onde:
    - ***expr*** – representa uma expressão a ser avaliada que pode ser do tipo lógica, relacional ou aritmética. O resultado da avaliação de uma expressão é sempre um valor lógico.
    - ***cmd*** – comando(s) a ser executado.

Comandos de Seleção

- Seleção composta
  - Verificando se há caracteres no buffer de leitura

selecao_composta_01

```
char caractere;

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

void loop()
{
   if (Serial.available()) {
      caractere = Serial.read();
      Serial.print("Caractere lido: ");
      Serial.println(caractere);
   }
   else {
      Serial.println("Insira um caractere!");
   }
   delay(400);
}
```

Comandos de Seleção

- Seleção composta (Comandos if aninhados)
  - Acendendo e apagando leds pelo monitor serial

selecao_composta_02

```
const int led_verde = 5;
const int led_amarelo = 6;
const int led_vermelho = 7;
char led;

void setup()
{
  pinMode(led_verde, OUTPUT);
  pinMode(led_amarelo, OUTPUT);
  pinMode(led_vermelho, OUTPUT);
  Serial.begin(9600);
}
```

```
void loop()
{
  if (Serial.available()) {
    led = Serial.read();

    if (led == 'G') {
      digitalWrite(led_verde, HIGH);
    }
    else if (led == 'g') {
      digitalWrite(led_verde, LOW);
    }
    else if (led == 'Y') {
      digitalWrite(led_amarelo, HIGH);
    }
    else if (led == 'y') {
      digitalWrite(led_amarelo, LOW);
    }
    else if (led == 'R') {
      digitalWrite(led_vermelho, HIGH);
    }
    else if (led == 'r') {
      digitalWrite(led_vermelho, LOW);
    }
  }
}
```

Comandos de Seleção

- Seleção de múltipla escolha
  - Na seleção de múltipla escolha é possível comparar vários valores.
  - No Arduino o comando de seleção de múltipla escolha é:

```
switch (valor) {
    case x: cmd1;
            break;
    case y: cmd2;
            break;
    default: cmd3;
}
```

  - onde:
    - *valor* – é um dado a ser avaliado. É representado por uma variável de memória.
    - cmd_x – comando a ser executado.
    - *case*– indica a opção a ser executada.
    - *default* – comando padrão que deverá ser executado se nenhuma outra escolha (*case*) tiver sido selecionada.

Comandos de Seleção

- Seleção de múltipla escolha
  - Acendendo e apagando leds pelo monitor serial

selecao_multipla_escolha

```
const int led_verde = 5;
const int led_amarelo = 6;
const int led_vermelho = 7;
char led;

void setup()
{
  pinMode(led_verde, OUTPUT);
  pinMode(led_amarelo, OUTPUT);
  pinMode(led_vermelho, OUTPUT);
  Serial.begin(9600);
}
```

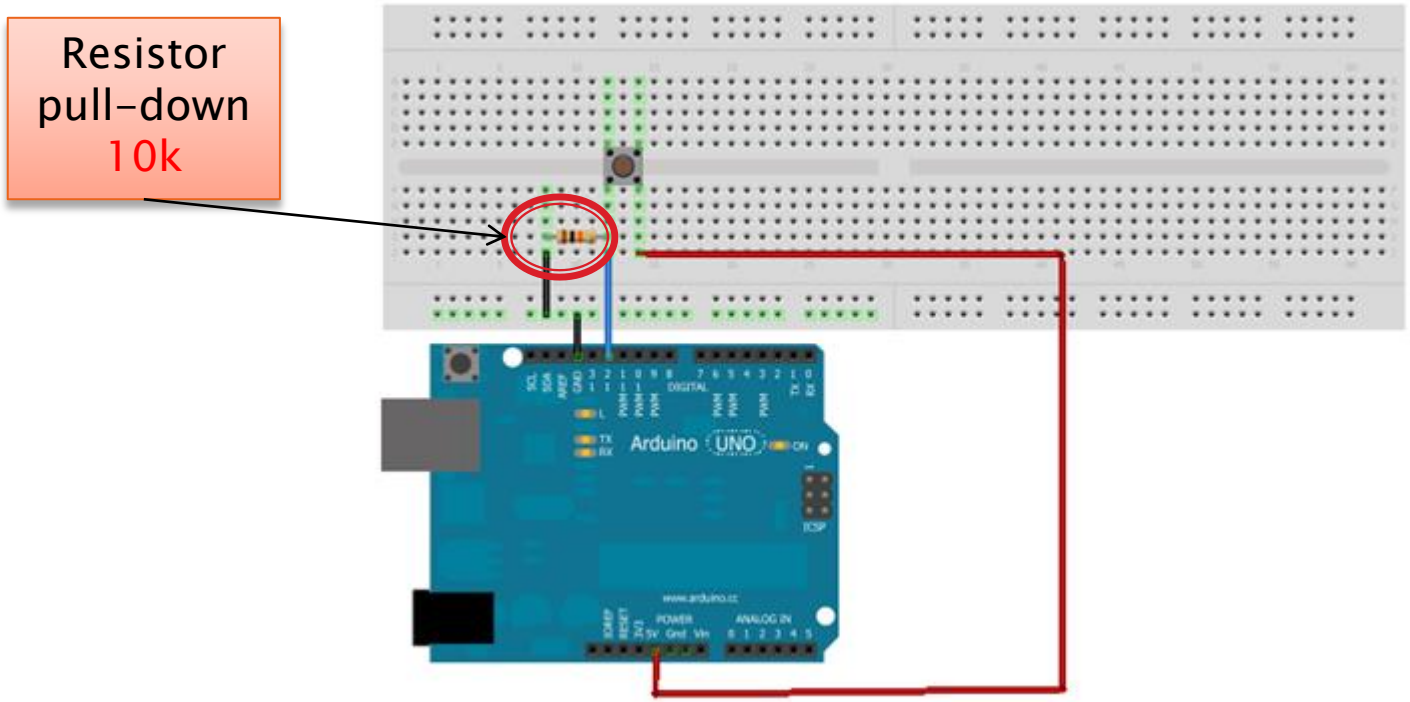
```
void loop()
{
  if (Serial.available()) {
    led = Serial.read();

    switch (led) {
      case 'G': digitalWrite(led_verde, HIGH);
                break;
      case 'g': digitalWrite(led_verde, LOW);
                break;
      case 'Y': digitalWrite(led_amarelo, HIGH);
                break;
      case 'y': digitalWrite(led_amarelo, LOW);
                break;
      case 'R': digitalWrite(led_vermelho, HIGH);
                break;
      case 'r': digitalWrite(led_vermelho, LOW);
                break;
      default: Serial.println("Nenhum led selecionado!");
    }
  }
}
```

Entrada Digital de Dados

- Lendo um botão
  - Para ler um botão basta ligá-lo em uma porta digital.
  - Para que um circuito com botão funcione adequadamente, ou seja, sem ruídos, é necessário o uso de resistores *pull-down* ou *pull-up*.
  - Os resistores *pull-down* e *pull-up* garantem que os níveis lógicos estarão próximos às tensões esperadas.

Entrada Digital de Dados

- Lendo um botão com resistor *pull-down*
  - Ligação na protoboard



Entrada Digital de Dados

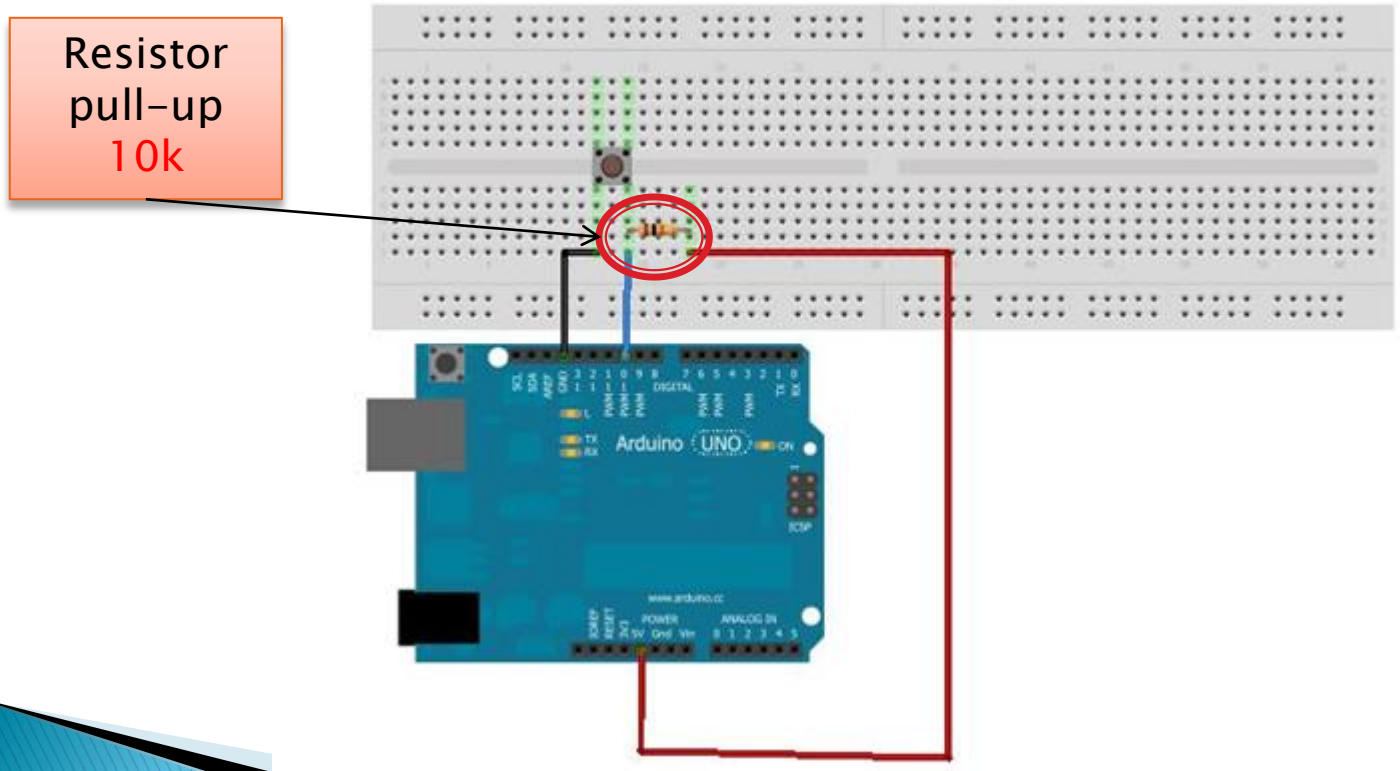
- Lendo um botão com resistor *pull-down*
  - Programa

pulldown

```
const int BOTAO = 8;
boolean estado_botao;

void setup()
{
   pinMode(BOTAO, INPUT);
   Serial.begin(9600);
}

void loop()
{
   estado_botao = digitalRead(BOTAO);
   if (estado_botao) {
     Serial.println("Botao pressionado!!!");
   }
}
```

Entrada Digital de Dados

- Lendo um botão com resistor *pull-up*
  - Ligação na protoboard



Entrada Digital de Dados

- Lendo um botão com resistor *pull-up*
  - Programa

pullup

```
const int BOTAO = 8;
boolean estado_botao;

void setup()
{
   pinMode(BOTAO, INPUT);
   Serial.begin(9600);
}

void loop()
{
   estado_botao = digitalRead(BOTAO);
   if (estado_botao == false) {
      Serial.println("Botao pressionado!!!");
   }
}
```

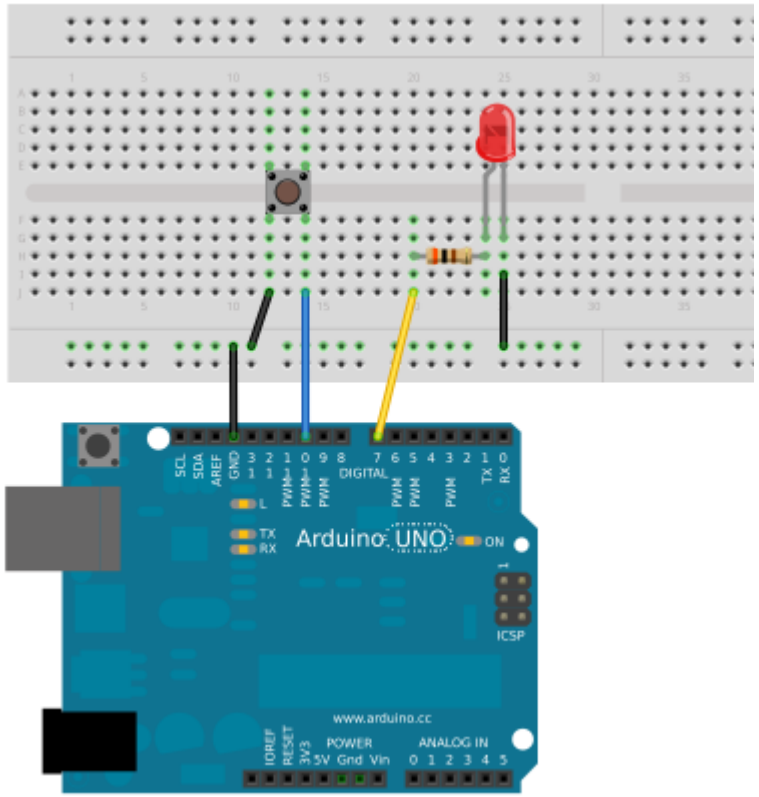
Entrada Digital de Dados

- **Nota**
  - O Arduino possui resistores *pull-up* nas portas digitais.
  - Para ativar os resistores *pull-up* de uma porta digital basta defini-la como entrada e colocá-la em nível alto (HIGH) na função *setup()*.
    - pinMode(*pin, INPUT*)
    - digitalWrite(*pin, HIGH*)
  - Para desativar os resistores *pull-up* de uma porta digital basta colocá-la em nível baixo.
    - digitalWrite(*pin, LOW*)

UNIVERSIDADE FEDERAL DE SANTA CATARINA
LARM
Laboratório de Automação e Robótica Móvel

Entrada Digital de Dados

- Ativando o resistor *pull-up* de uma porta digital
  - Quando o botão for pressionado o led irá apagar



Entrada Digital de Dados

- Ativando o resistor *pull-up* de uma porta digital
  - Quando o botão for pressionado o led irá apagar

pullup_arduino

```
const int LED = 7;
const int BOTAO = 10;
boolean estado_botao;

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(BOTAO, INPUT);
  // ativa o resistor pull-up
  digitalWrite(BOTAO, HIGH);
}
```

```
void loop()
{
  estado_botao = digitalRead(BOTAO);

  if (estado_botao == HIGH) {
    digitalWrite(LED, HIGH);
  }
  else {
    digitalWrite(LED, LOW);
  }
}
```

UNIVERSIDADE FEDERAL DE SANTA CATARINA
LARM
Laboratório de Automação e Robótica Móvel

Entrada Digital de Dados

- **Nota**
  - O Arduino possui uma constante chamada *INPUT_PULLUP* que define que a porta será de entrada e o resistor *pull-up* da mesma será ativado.
  - Exemplo:

```
void setup()
{
  pinMode(10, INPUT_PULLUP);
}
```

Define a porta 10 como entrada de dados e ativa o resistor pull-up.

Entrada Analógica de Dados

- O Arduino UNO possui 6 (seis) portas analógicas.
- Por padrão todas as portas analógicas são definidas como entrada de dados, desta forma não é necessário fazer esta definição na função *setup()*.
- O conversor analógico-digital do Arduino é de 10 (dez) bits, logo a faixa de valores lidos varia de 0 a 1023.
- As portas analógicas no Arduino UNO são identificadas como A0, A1, A2, A3, A4 e A5. Estas portas também podem ser identificadas por 14 (A0), 15 (A1), 16 (A2), 17 (A3), 18 (A4) e 19 (A5).

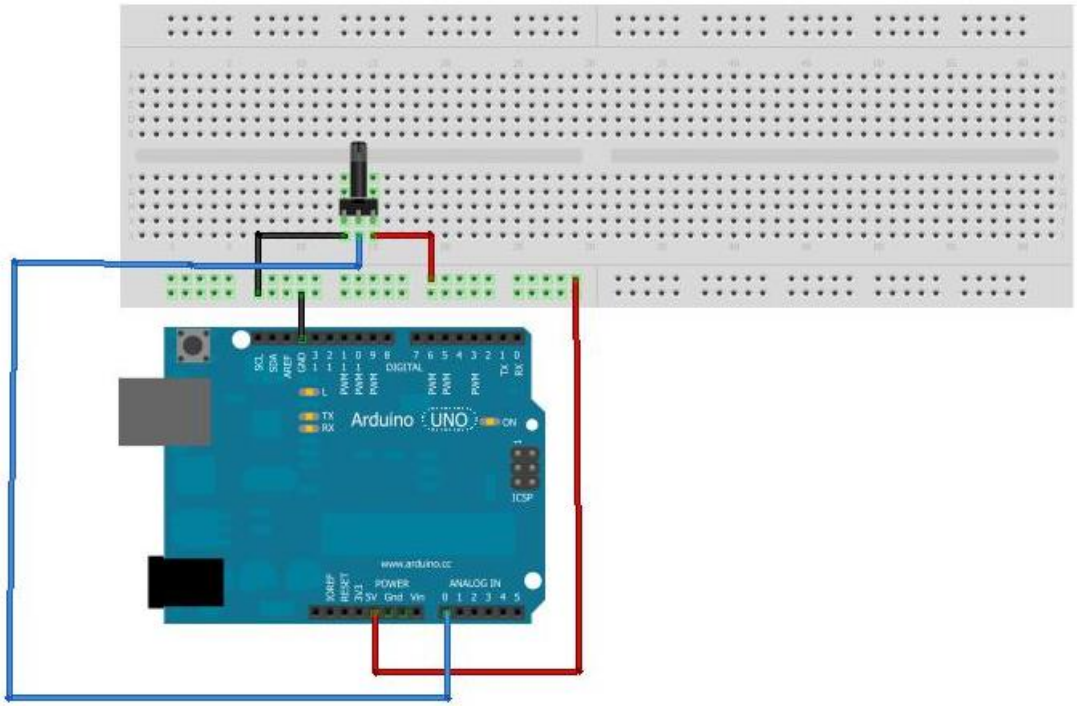
Entrada Analógica de Dados

- Na seção "Portas Digitais" vimos que para ler dados em uma porta digital precisávamos usar uma função chamada digitalRead().
- De forma semelhante, para fazer uma leitura de dados em uma porta analógica usaremos analogRead().

- Exemplo:
  - analogRead(A0)

Entrada Analógica de Dados

- Lendo dados de um potenciômetro



Entrada Analógica de Dados

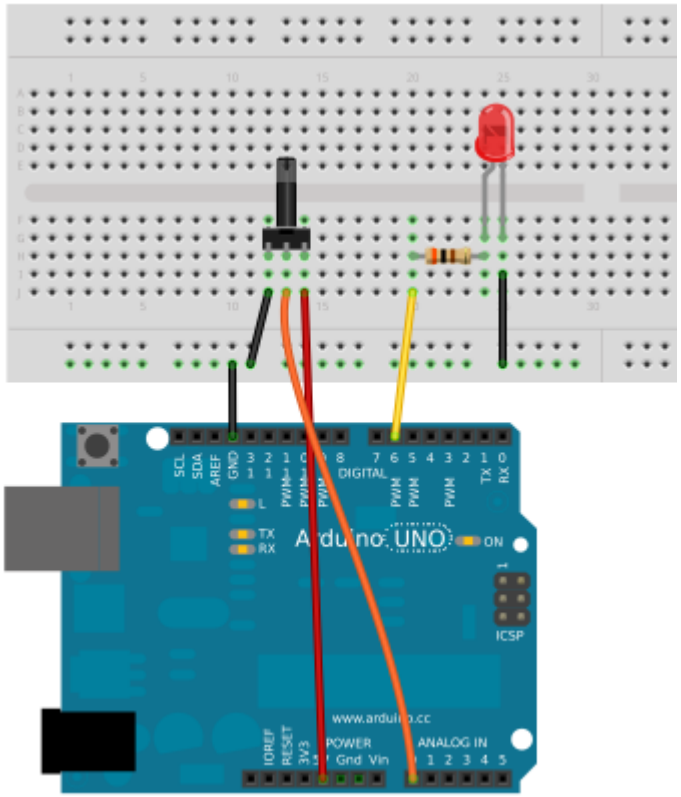
- Lendo dados de um potenciômetro

potenciometro

```
int valor;

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

void loop()
{
  valor = analogRead(A0);
  Serial.println(valor);
}
```

Entrada Analógica de Dados

- Lendo dados de um potenciômetro e acionando um LED



Entrada Analógica de Dados

- Lendo dados de um potenciômetro e acionando um LED

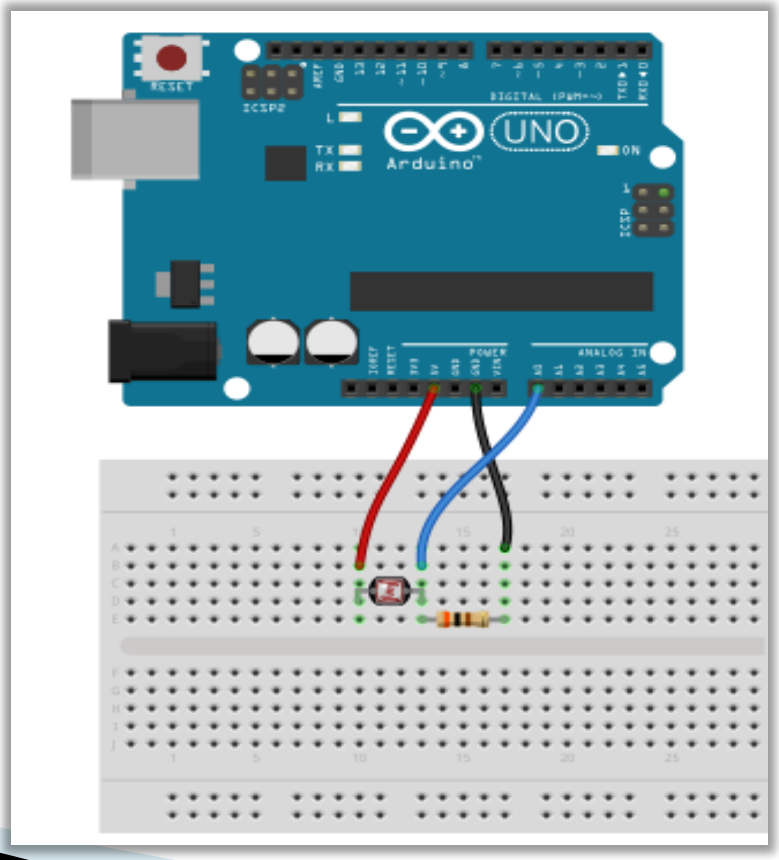
pot_led

```
const int LED = 6;
const int POT = A0;
int valor;

void setup()
{
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
}
```

```
void loop()
{
  valor = analogRead(POT);
  Serial.println(valor);

  digitalWrite(LED, HIGH);
  delay(valor);
  digitalWrite(LED, LOW);
  delay(valor);
}
```

Entrada Analógica de Dados

- Lendo dados de um LDR e imprimindo no monitor serial.



Entrada Analógica de Dados

- Lendo dados de um LDR e imprimindo no monitor serial.

LDR

```
const int LDR = A0;
int valor;

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

void loop()
{
  valor = analogRead(LDR);
  Serial.print("Valor lido: ");
  Serial.println(valor);
  delay(200);
}
```

Entrada Analógica de Dados

- Mapeando valores
  - Algumas vezes precisamos alterar valores que se encontram dentro de uma determinada faixa, de modo a obter um novo valor, proporcional ou inversamente proporcional ao primeiro, e que se enquadre em uma nova faixa de valores.
  - A biblioteca do Arduino possui uma função chamada map(), que realiza essa tarefa.

Entrada Analógica de Dados

- Mapeando valores
  - Exemplo de uso da função map():
    - novo_valor = map(valor, min_in, max_in, min_out, max_out);

    Onde:
    - novo_valor recebe o valor já modificado pela função map();
    - valor é o dado a ser alterado;
    - min_in é o menor valor da faixa de entrada;
    - max_in é o maior valor da faixa de entrada;
    - min_out é o menor valor da faixa de saída;
    - max_out é o maior valor da faixa de saída.

Entrada Analógica de Dados

- Mapeando valores

map

```
const int POT = A0;
int valor;

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

void loop()
{
  valor = analogRead(POT);
  Serial.print("Valor lido:    ");
  Serial.println(valor);

  valor = map(valor, 0, 1023, 0, 100);
  Serial.print("Valor mapeado: ");
  Serial.println(valor);
  Serial.println();
  delay(400);
}
```

Comandos de Repetição

- Muitas vezes é necessário repetir uma determinada instrução ou conjunto de instruções.
- Os comandos de repetição mantêm em um "laço" uma instrução ou conjunto de instruções enquanto uma condição estiver sendo satisfeita.
- Os comandos de repetição do Arduino são:
  - Baseados em um contador
  - Baseados em uma expressão com teste no início
  - Baseados em uma expressão com teste no final

UNIVERSIDADE FEDERAL DE SANTA CATARINA
LARM
Laboratório de Automação e Robótica Móvel

Comandos de Repetição

- Repetição baseada em um contador
  - Este tipo de comando de repetição deve ser utilizado quando se sabe a quantidade de vezes que uma determinada instrução deve ser executada.
  - No Arduino o comando de repetição baseado em um contador é:

```
for (inicialização; condição; incremento) {
  cmd;
}
```

  - onde:
    - *inicialização*: é onde se atribui um valor inicial a um contador;
    - *condição* : é uma expressão relacional ou lógica;
    - *incremento* : onde se atribui um novo valor ao contador.

Comandos de Repetição

- Repetição baseada em um contador
  - Escrevendo uma mensagem *x* vezes no monitor serial

for

```
int vezes = 10; // qtde de vezes que a mensagem será impressa
int contador; // irá contar quantas vezes a mensagem já foi impressa

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

  for (contador = 0; contador < vezes; contador++) {
    Serial.println("Testando o comando de repeticao for()");
  }
}

void loop()
{

}
```

Comandos de Repetição

- **Nota**
  - É possível declarar o contador dentro do cabeçalho do laço for.

```
for (int i = 0; i < 50; i++) {
  Serial.println("Oficina de Robotica");
}
```

Comandos de Repetição

- Repetição com teste no início
  - Este tipo de comando de repetição avalia uma expressão, caso seja verdadeira, a(s) instrução(ções) dentro do "laço" permanecem sendo executadas.
  - No Arduino o comando de repetição com teste no início é:

```
while (expr) {
  cmd;
}
```

  - onde:
    - ***expr*** – é uma expressão que pode ser lógica, relacional ou aritmética. A permanência de execução do "laço" é garantida enquanto a expressão for verdadeira.
  - Nota:
    - Neste tipo de comando de repetição a avaliação da expressão é realizada no início do laço, ou seja, pode ser que o *cmd* não execute nenhuma vez.

Comandos de Repetição

- Repetição com teste no início

while

```
const int BOTAO = 6;

void setup()
{
  Serial.begin(9600);
  pinMode(BOTAO, INPUT_PULLUP);
}

void loop()
{
  while (digitalRead(BOTAO)) {
    Serial.println("Pressione o botao!");
  }
  Serial.println("Obrigado!");
  delay(2000);
}
```

Comandos de Repetição

- Repetição com teste no início

semaforo

```
/**********************************
* Nome do Programa: Semáforo     *
*                                *
* Descrição:                     *
*    Mantém o sinal aberto até   *
*    que um pedestre solicite a  *
*    travessia. Então fecha o    *
*    sinal por alguns segundos.  *
**********************************/

const int BOTAO = 6;
const int VERDE = 8;
const int AMARELO = 9;
const int VERMELHO = 10;

void setup()
{
  pinMode(VERDE, OUTPUT);
  pinMode(AMARELO, OUTPUT);
  pinMode(VERMELHO, OUTPUT);
  pinMode(BOTAO, INPUT_PULLUP);
}

void loop()
{
  // abre o sinal
  digitalWrite(VERDE, HIGH);

  // aguarda pedestre
  while (digitalRead(BOTAO))
    ; // instrução vazia

  // muda para o amarelo
  digitalWrite(VERDE, LOW);
  digitalWrite(AMARELO, HIGH);
  delay(4000);

  // fecha o sinal
  digitalWrite(AMARELO, LOW);
  digitalWrite(VERMELHO, HIGH);
  delay(8000);

  // abre novamente
  digitalWrite(VERMELHO, LOW);
}
```

Comandos de Repetição

- Repetição com teste no final
  - Este tipo de comando de repetição avalia uma expressão, caso seja verdadeira, a(s) intrução(ções) dentro do "laço" permanecem executando.
  - No Arduino o comando de repetição com teste no final é:

```
do {
  cmd;
} while (expr) ;
```

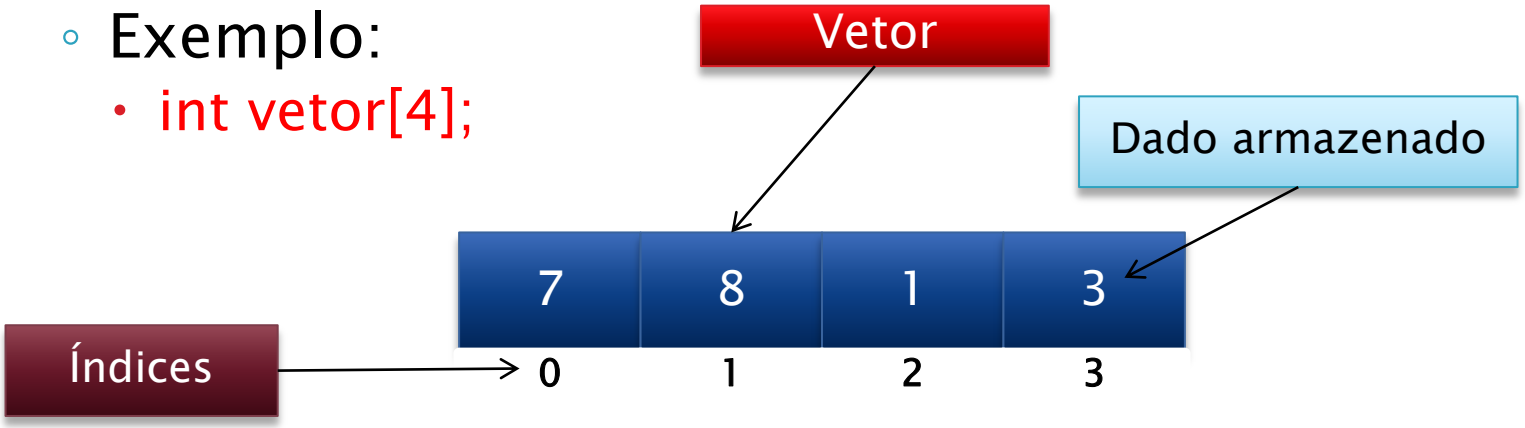
  - onde:
    - *expr* – é uma expressão que pode ser lógica, relacional ou aritmética. A permanência de execução do "laço" é garantida enquanto a expressão for verdadeira.
  - Nota:
    - Neste tipo de comando de repetição a avaliação da expressão é realizada no final do laço, ou seja, é garantido que pelo menos uma vez o *cmd* será executado.

Vetores e Matrizes

- Uma variável escalar pode armazenar muitos valores ao longo da execução do programa, porém não ao mesmo tempo.
- Existem variáveis que podem armazenar mais de um valor ao mesmo tempo. Essas variáveis são conhecidas como "variáveis compostas homogêneas".
- No Arduino é possível trabalhar com dois tipos de variáveis compostas homogêneas, vetores e matrizes.

Vetores e Matrizes

- Vetor
  - A declaração de um vetor é feita da mesma maneira que uma variável escalar, entretanto é necessário definir a quantidade de itens do vetor.
  - Exemplo:
    - int vetor[4];



- Vetor com 4 (quatro) elementos do tipo inteiro.

Vetores e Matrizes

- Vetor

  - Para atribuir um valor a uma determinada posição do vetor, basta usar o índice, ou seja, a posição onde o valor será armazenado no vetor.
  - Exemplo:
    - vetor[0] = 7;
    - Atribui o valor 7 a posição 0 do vetor.

Vetores e Matrizes

- Vetor

  - Para acessar um determinado valor em uma posição do vetor, basta usar o índice, ou seja, a posição onde o valor está armazenado no vetor.
  - Exemplo:
    - digitalWrite(*vetor[0], HIGH*);
    - Ativa a porta cujo número está definido na posição 0 do vetor.

Vetores e Matrizes

- Vetor
  - Acendendo e apagando leds cujas portas estão definidas em um vetor

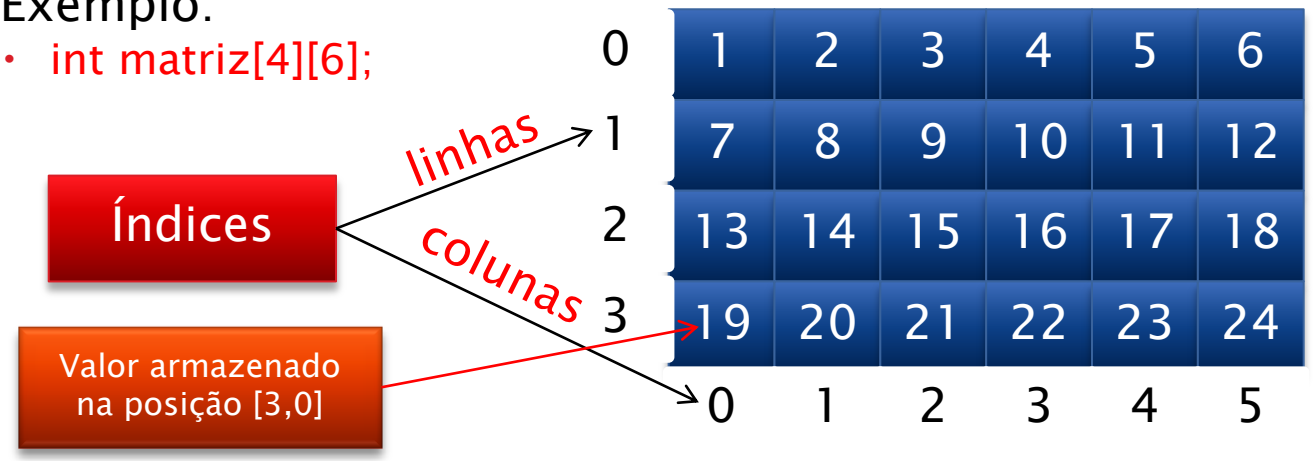
vetores

```
const int leds[5] = {2, 3, 4, 5, 6};
int i;

void setup()
{
  for (i = 0; i < 5; i++) {
    pinMode(leds[i], OUTPUT);
  }
}
```

```
void loop()
{
  for (i = 0; i < 5; i++) {
    digitalWrite(leds[i], HIGH);
    delay(1000);
  }
  for (i = 0; i < 5; i++) {
    digitalWrite(leds[i], LOW);
    delay(1000);
  }
}
```

Vetores e Matrizes

- Matriz
  - Uma matriz é similar a um vetor, entretanto pode ser formada por duas ou mais dimensões.
  - Uma matriz bidimensional possui um determinado número de linhas e de colunas.
  - Exemplo:
    - int matriz[4][6];



  - Matriz com 4 (quatro) linhas e 6 (seis) colunas de elementos do tipo inteiro.

UNIVERSIDADE FEDERAL DE SANTA CATARINA

LARM
Laboratório de Automação e Robótica Móvel

Vetores e Matrizes

- Matriz
  - Para atribuir um valor a uma determinada posição da matriz, basta usar o índice da linha e o índice da coluna, ou seja, a posição onde o valor será armazenado na matriz.
  - Exemplo:
    - matriz[1][2] = 9;
    - Atribui o valor 9 a posição 1 (linha), 2 (coluna) da matriz.

Vetores e Matrizes

- Matriz
  - Para acessar um determinado valor em uma posição da matriz, basta usar o índice da linha e o da coluna, ou seja, a posição onde o valor está armazenado na matriz.
  - Exemplo:
    - digitalWrite(*matriz[0][0], HIGH*);
    - Ativa a porta cujo número está definido na posição 0 (linha), 0 (coluna) da matriz.

Vetores e Matrizes

- Matriz
  - Acendendo e apagando leds aleatoriamente em uma matriz

matrizes

```
const int matriz_leds[2][2] = {{2, 3}, {4, 5}};
int i, j;

void setup()
{
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; j++) {
      pinMode(matriz_leds[i][j], OUTPUT);
    }
  }
}
```

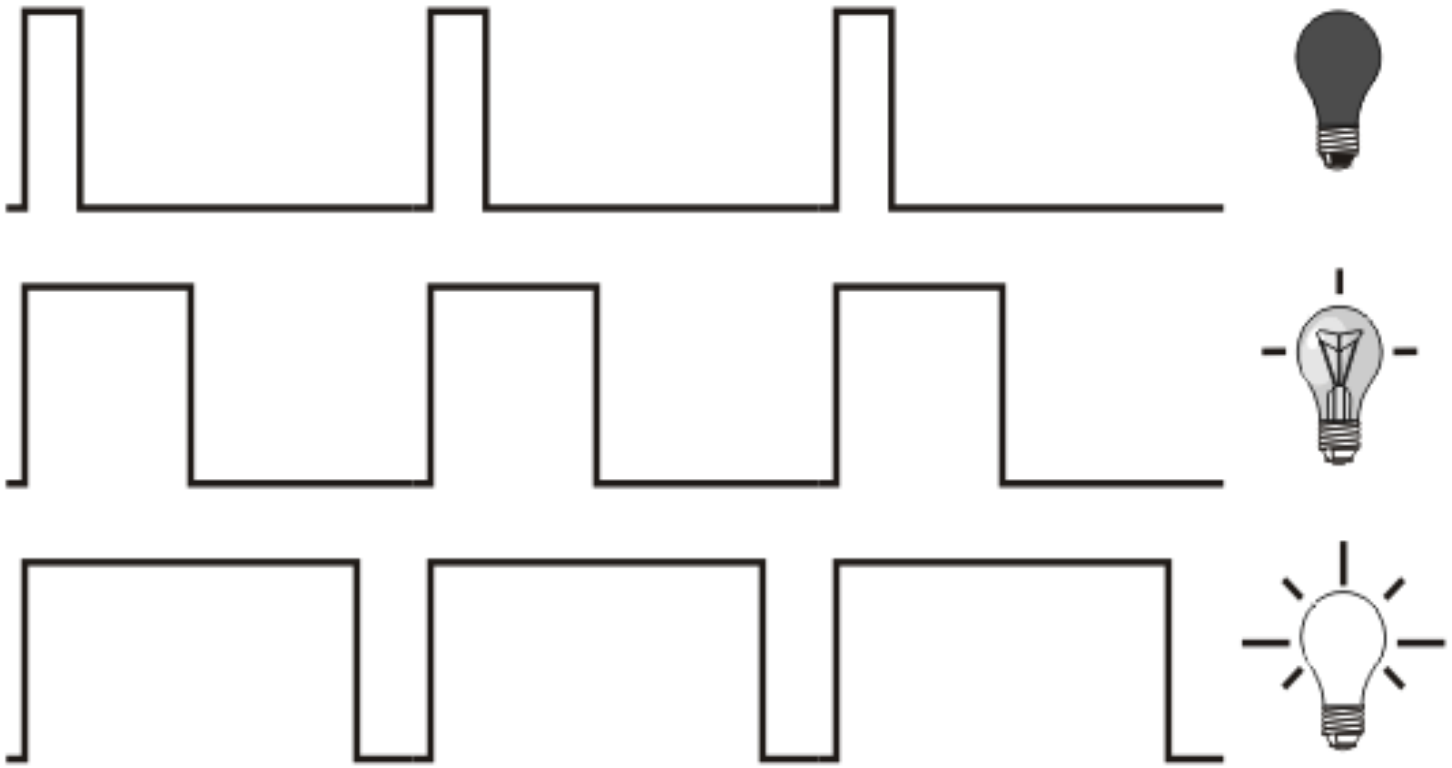
```
void loop()
{
  i = random(2);
  j = random(2);

  digitalWrite(matriz_leds[i][j], HIGH);
  delay(100);
  digitalWrite(matriz_leds[i][j], LOW);
  delay(100);
}
```

Pulse Width Modulation (PWM)

- A Modulação por Largura de Pulso (Pulse Width Modulation – PWM) é uma técnica que nos permite gerenciar a quantidade de energia enviada para uma saída digital.
- Essa modulação é feita definindo-se um ciclo de trabalho que determina com que frequência o sinal muda do nível lógico HIGH para o nível lógico LOW e vice-versa.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
LARM
Laboratório de Automação e Robótica Móvel

Pulse Width Modulation (PWM)



Extraído de *Teach Yourself PIC Microcontrollers for Absolute Beginners – M. Amer Iqbal Qureshi, 2006*

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

LARM
Laboratório de Automação
e Robótica Móvel

Pulse Width Modulation (PWM)

- O Arduino UNO possui 6 (seis) portas PWM – 3, 5, 6, 9, 10 e 11.
- O sinal PWM pode variar de 0 a 255 e para ativá-lo basta usar a seguinte instrução em uma das portas PWM:
  - analogWrite(*pin, valor*);
- Note que as portas PWM são todas digitais, porém o sinal é modulado "como se fosse" um sinal analógico.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
LARM
Laboratório de Automação e Robótica Móvel

Pulse Width Modulation (PWM)

- Exemplo: Usando o PWM para controlar a intensidade de luz emitida por um LED.

PWM

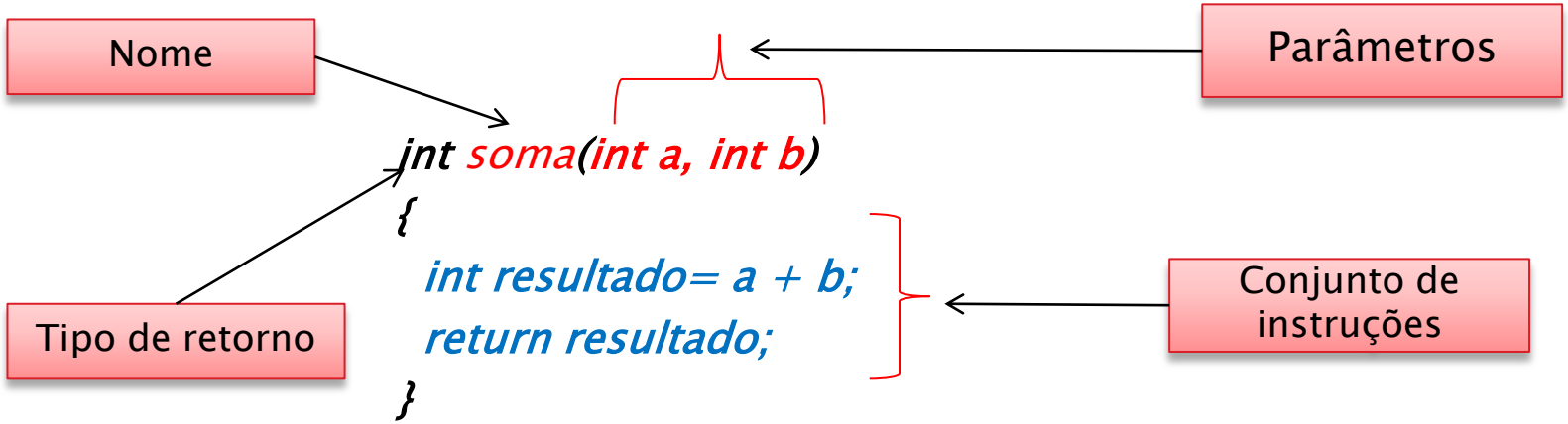
```
const int LED = 3;

void setup()
{
  pinMode(LED, OUTPUT);
}

void loop()
{
  for (int i = 0; i <= 255; i++) {
    analogWrite(LED, i);
    delay(30);
  }
}
```

Modularização

- O objetivo da modularização é separar o programa em módulos funcionais – "dividir para conquistar".
- Um módulo pode ser chamado (acionado) em qualquer ponto do programa.
- Os módulos funcionais de um programa também são chamados de funções.
- Uma função implementa uma ou mais instruções responsáveis por uma parte do programa.
- As funções deixam um programa mais organizado e legível, uma vez que são responsáveis por ações bem específicas.

Modularização

- Uma função tem quatro partes fundamentais:
  - um tipo de dado associado a ela (pode ser *void*);
  - um nome;
  - uma lista de parâmetros (se houver);
  - conjunto de instruções.



Modularização

- Exemplo: Blink Leds Modularizado

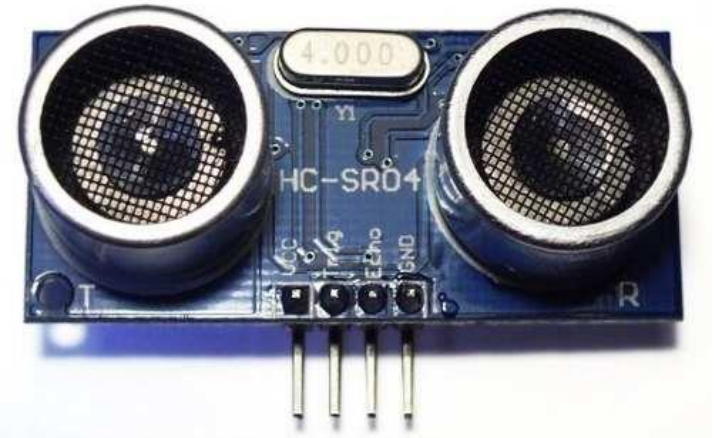
modularizacao_blink_led

```
const int led_verde = 2;
const int led_amarelo = 3;
const int led_vermelho = 4;

void pisca_led(int porta)
{
  digitalWrite(porta, HIGH);
  delay(1000);
  digitalWrite(porta, LOW);
  delay(1000);
}
```

```
void setup()
{
  pinMode(led_verde, OUTPUT);
  pinMode(led_amarelo, OUTPUT);
  pinMode(led_vermelho, OUTPUT);
}

void loop()
{
  pisca_led(led_verde);
  pisca_led(led_amarelo);
  pisca_led(led_vermelho);
}
```

Sensor Ultrassônico

- É um sensor muito utilizado em robótica para detectar a presença de obstáculos.
- É composto por um emissor e um receptor:
  - O emissor (trigger) emite um onda sonora de alta frequência;
  - O receptor (echo) recebe a onda emitida pelo trigger, após esta refletir em algum objeto e retornar para o sensor.



Sensor Ultrassônico

- Para escrever o código que irá controlar o sensor ultrassônico, usaremos a biblioteca Ultrassonico.
- Para ter acesso às funcionalidades dessa biblioteca, precisamos incluir no nosso código o arquivo de cabeçalhos Ultrassonico.h.

```
#include <Ultrassonico.h>
```

Sensor Ultrassônico

- O próximo passo é criar uma variável do tipo Ultrassonico, que irá representar o sensor no nosso programa.

```
// Cria uma objeto Ultrassonico, especificando os pinos
// aos quais o sensor está conectado (trigger e echo).
Ultrassonico sensor(3, 4);
```

- Depois disso, sempre que precisarmos saber qual a distância entre o sensor e qualquer obstáculo, basta chamarmos a função membro distancia().

```
sensor.distancia();
```