

Oficina de Robótica

Fundamentos de Eletrônica e Programação em Arduino



LARM
Laboratório de Automação e Robótica Móvel

Apresentação

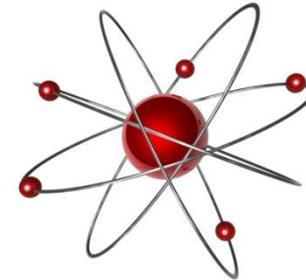
- Material produzido para o projeto **Oficina de Robótica** por:
 - Anderson Luiz Fernandes Perez
 - Miguel Garcia Silvestre
- Contatos:
 - Universidade Federal de Santa Catarina -Laboratório de Automação e Robótica Móvel
 - [anderson.perez \(at\) ufsc.br](mailto:anderson.perez@ufsc.br)
 - [renanrdaros \(at\) hotmail.com](mailto:renanrdaros@hotmail.com)
- <http://oficinaderobotica.ufsc.br>

Sumário

- Corrente e Tensão Elétrica
- Resistência Elétrica
- Introdução ao Arduino
- Microcontroladores
- Arduino UNO
- Ambiente de desenvolvimento
- Funções *setup()* e *loop()*
- Monitor Serial
- Portas digitais
- Programando em Arduino

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
 - 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.

Corrente e Tensão

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

Corrente e Tensão

- Corrente
 - A corrente pode ser medida em:
 - Ampere (A)
 - Miliampere (mA - 10^{-3})
 - Microampere (μA - 10^{-6})

Observação:

- Na eletrônica o sentido da corrente é do polo positivo em direção ao polo negativo.
- Na física o sentido da corrente é do polo negativo para o polo positivo.

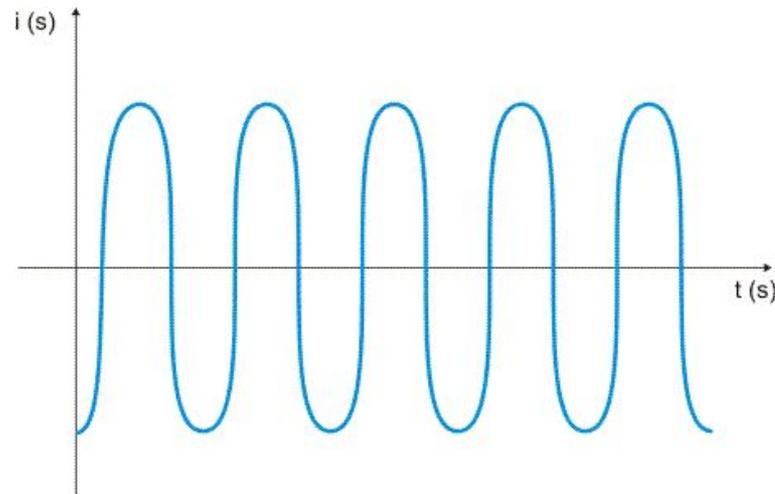
Corrente e Tensão

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



Corrente e Tensão

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



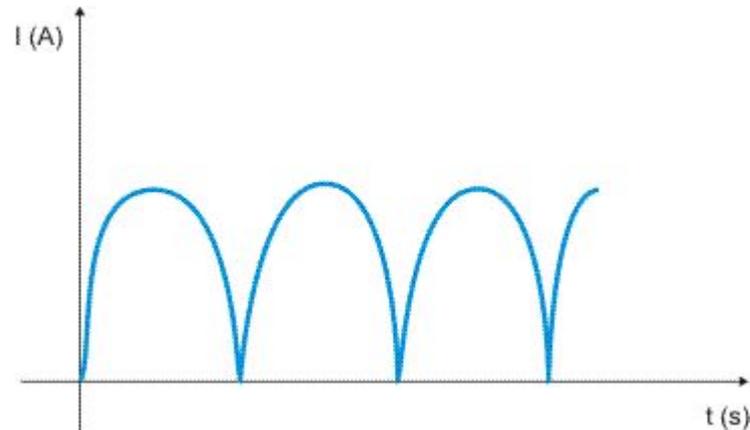
Corrente e Tensão

- Tipos de correntes elétricas
 - Corrente contínua
 - Não altera o seu sentido, ou seja, ou é sempre positiva ou é sempre negativa.
 - Grande parte dos equipamentos eletrônicos trabalha com corrente contínua.



Corrente e Tensão

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



Resistência Elétrica

- Resistência 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, ou seja, **possuem uma baixa resistência**.
- A **resistência é medida em Ohms** e o **símbolo** é a letra grega ômega Ω .

Introdução ao Arduino

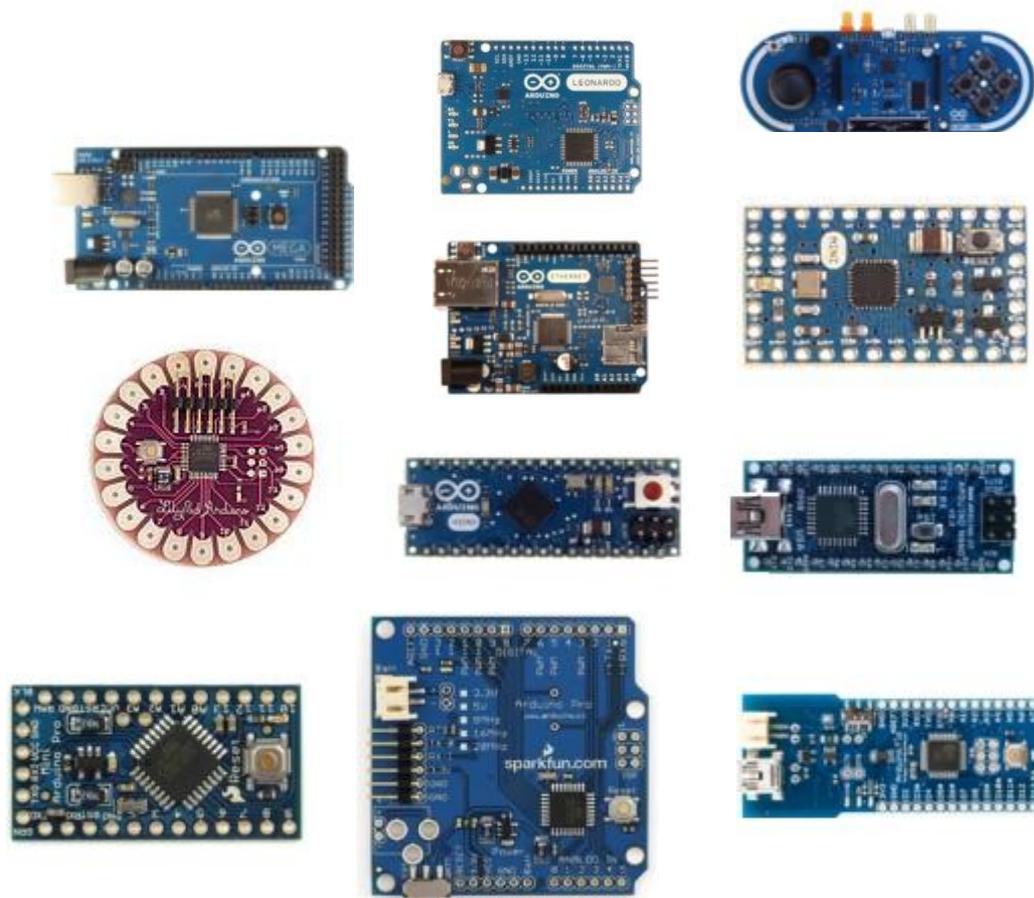
- O Arduino é uma plataforma utilizada para **prototipação de circuitos eletrônicos**.
- O projeto do Arduino teve início em 2005 na cidade de Ivrea, Itália.
- O **Arduino é composto** por **uma placa** com **microcontrolador Atmel AVR** e um **ambiente de programação** baseado em Wiring e C++.
- Tanto 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 de **plataforma de computação física**.

Introdução ao Arduino

- Tipos de Arduino

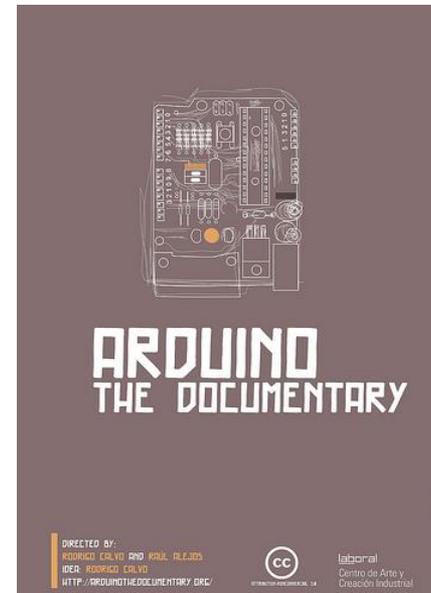
- Existem vários tipos de Arduino com especificidades de hardware. [O site oficial do Arduino lista os seguintes tipos:](#)

- 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



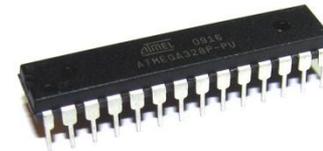
Introdução ao Arduino

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



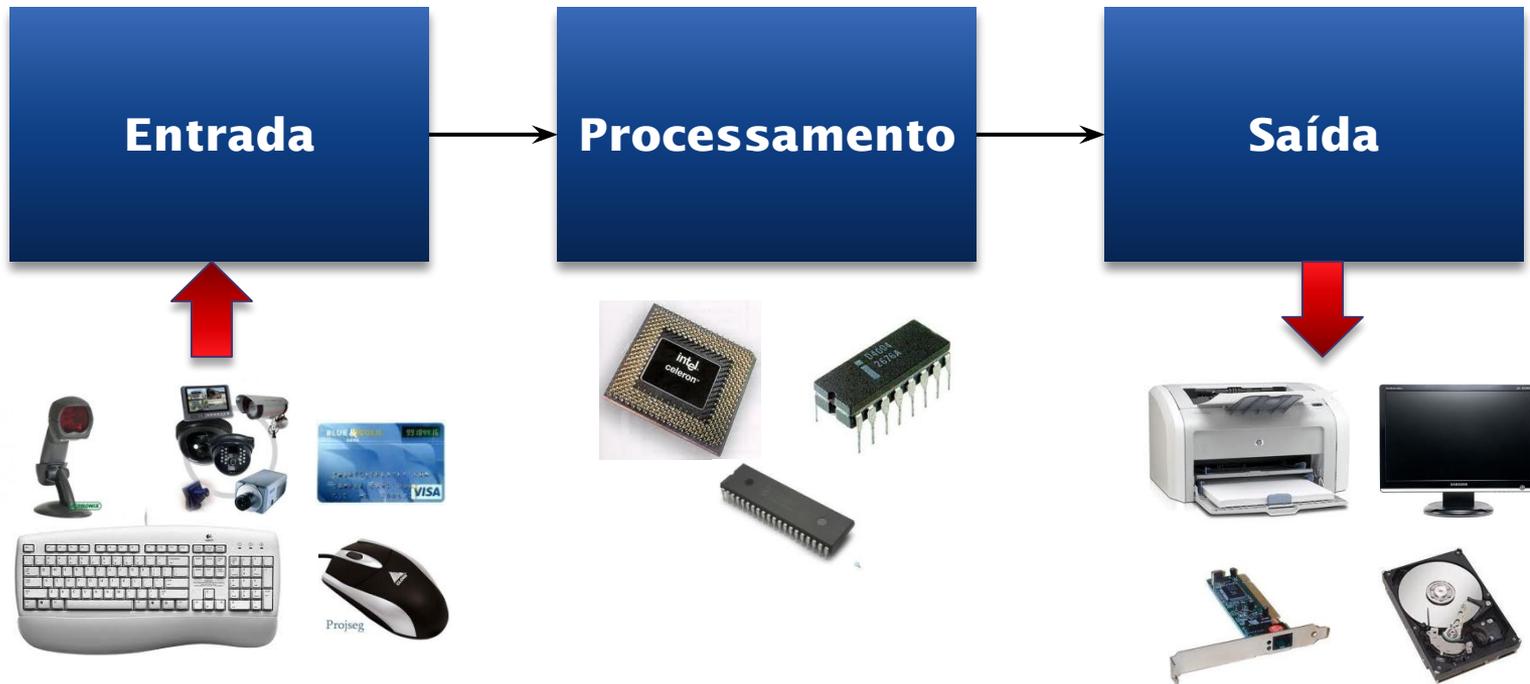
Microcontroladores

- Um **microcontrolador** é um **CI** que incorpora **várias funcionalidades**.
- Alguns vezes os microcontroladores são chamados de “**computador de um único chip**”.
- São **utilizados** em **diversas aplicações** de sistemas embarcados, tais como: **carros, eletrodomésticos, aviões, automação residencial, etc.**



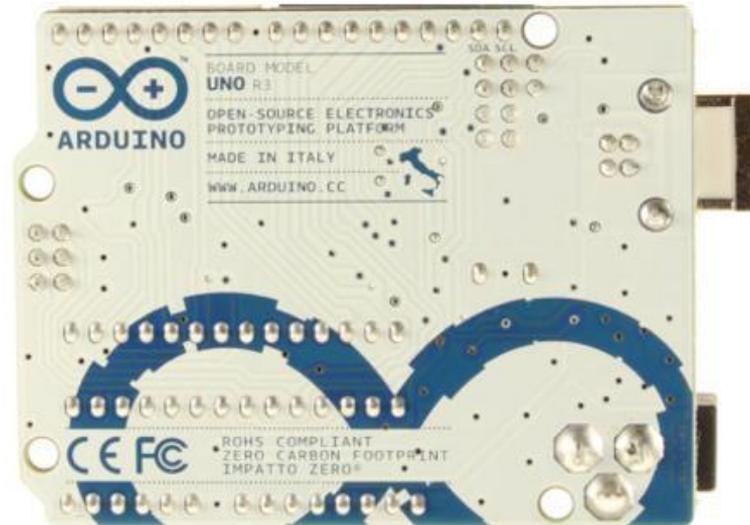
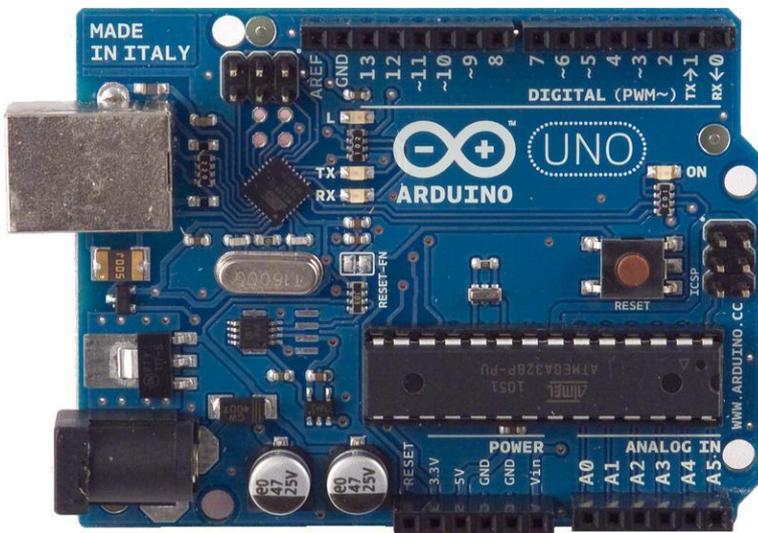
Microcontroladores

- Processamento de dados



Arduino UNO

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



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**

Arduino UNO

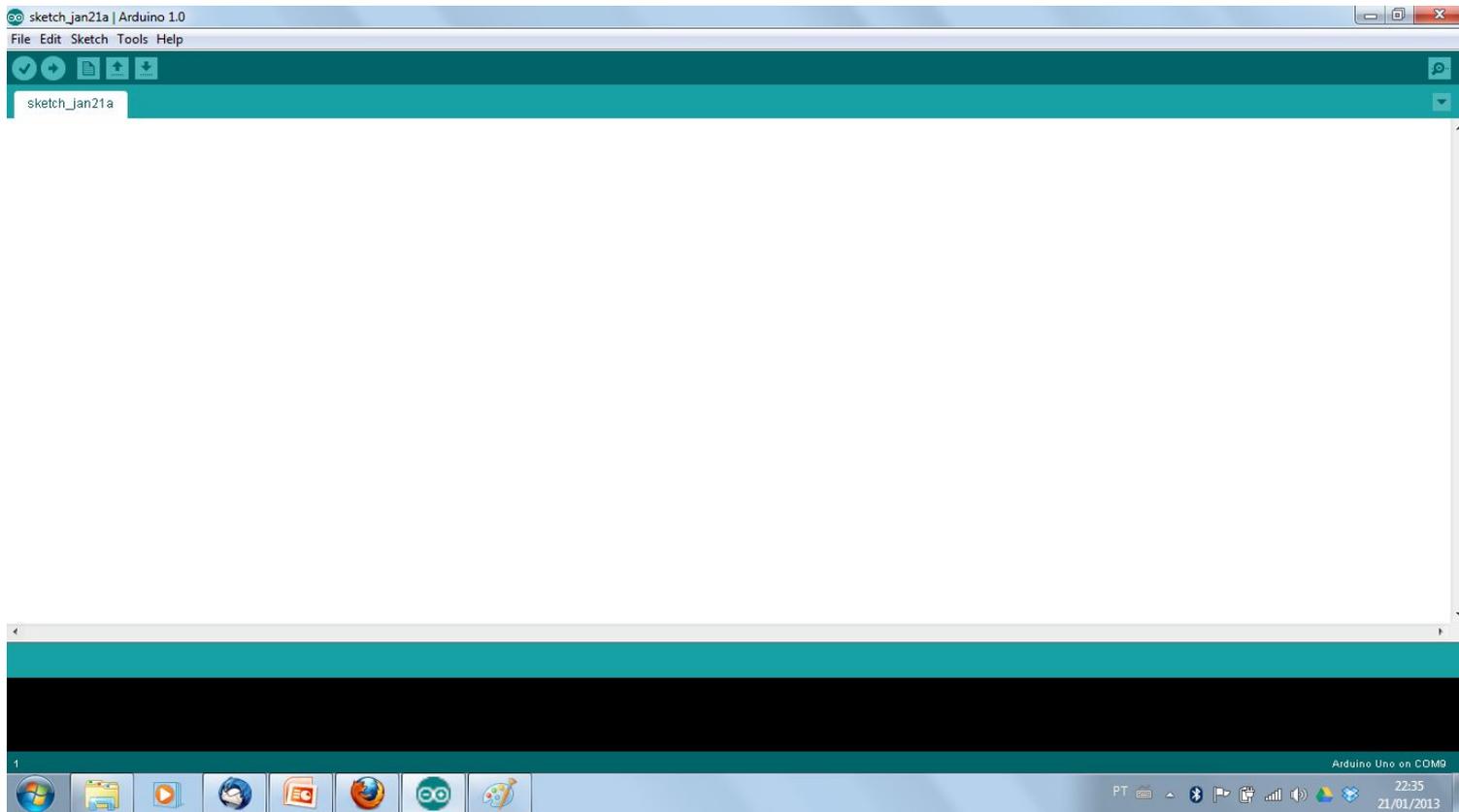
- Alimentação
 - O **Arduino UNO** pode ser alimentado pela porta **USB** ou por uma **fonte externa DC**.
 - A recomendação é que a **fonte externa seja de 7 V a 12 V** e pode ser ligada diretamente no conector de fonte ou nos pinos **Vin** e **Gnd**.

Ambiente de desenvolvimento

- O ambiente de desenvolvimento do Arduino (IDE) é gratuito e pode ser baixado no seguinte endereço: [arduino.cc](https://www.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.

Funções *setup()* e *loop()*

- **Exemplo 1:** formato das funções *setup()* e *loop()*

```
1  void setup()  
2  {  
3  
4  
5  }  
6  
7  void loop()  
8  {  
9  
10  
11 }  
12
```

Funções *setup()* e *loop()*

- **Exemplo 2:** exemplo funções *setup()* e *loop()*

```
1  void setup()  
2  {  
3  |  pinMode(13, OUTPUT);  
4  |  
5  |  
6  void loop()  
7  {  
8  |  digitalWrite(13, HIGH);  
9  |  delay(1000);  
10 |  digitalWrite(13, LOW);  
11 |  delay(1000);  
12 |  
13 }
```

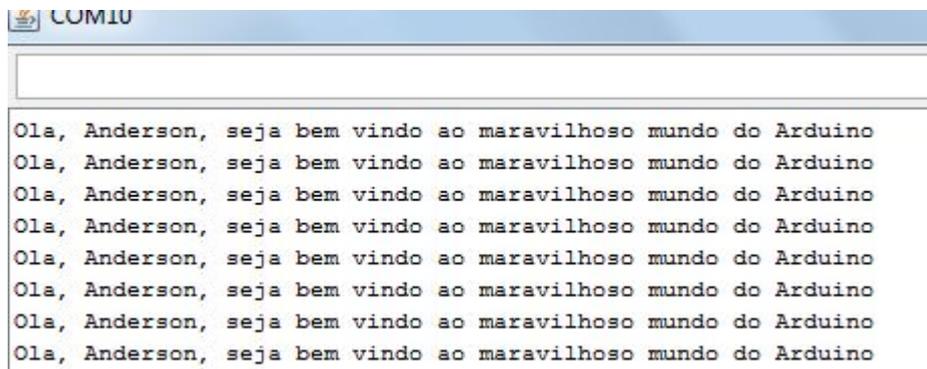
Monitor Serial

- 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

- **Exemplo:** imprimindo uma mensagem de boas vindas no monitor serial

```
1 void setup()
2 {
3   Serial.begin(9600); // Definição da velocidade de transmissão
4 }
5
6 void loop()
7 {
8   Serial.println("Ola, seu nome, seja bem vindo ao maravilhoso mundo do Arduino");
9 }
```



The screenshot shows the Serial Monitor window in the Arduino IDE. The window title is "COM10". The output area displays the message "Ola, Anderson, seja bem vindo ao maravilhoso mundo do Arduino" repeated ten times, one on each line.

Portas digitais

- O Arduino possui tanto portas digitais como 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

- Para definir uma porta como entrada ou saída é necessário explicitar essa situação no programa.
- A função `pinMode(pin, estado)` é utilizada para definir se a porta será de entrada ou saída de dados.
- **Exemplo:**
 - Define que a porta 13 será de saída
 - `pinMode(13, OUTPUT)`
 - Define que a porta 7 será de entrada
 - `pinMode(7, INPUT)`

Portas digitais

- Portas Digitais
 - As portas digitais trabalham com valores bem definidos, ou seja, 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(pin, estado)`.
 - Para ler um valor em uma porta digital basta utilizar a função `digitalRead(pin)`.

Programando em Arduino

- Constantes e Variáveis
 - 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

Programando em Arduino

- Constantes e Variáveis
 - 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`

Programando em Arduino

- Constantes e Variáveis
 - 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.

Programando em Arduino

- Constantes e 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**.

Programando em Arduino

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

Programando em Arduino

- 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 .

Programando em Arduino

- 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 .

Programando em Arduino

- Declaração de Variáveis e Constantes
 - **Exemplo:** declaração de duas constantes e uma variável

```
1  #define BOTAO 10 // constante
2
3  const int pin_botao = 13; // constante
4
5  void setup()
6  {
7
8  }
9
10 void loop()
11 {
12 |   int valor_x; // variável
13 | }
14
```

Programando em Arduino

- Atribuição de valores a variáveis e constantes
 - A atribuição de valores a variáveis e constantes é feito com o uso do **operador de atribuição =**.
 - Exemplos:
 - `int valor = 100;`
 - `const float pi = 3.14;`
 - **Atenção!!!**
 - O operador de atribuição não vale para o comando ***#define***.

Programando em Arduino

- Atribuição de valores a variáveis e constantes
 - **Exemplo:** lendo dados do monitor serial

```
1  int valor = 0;
2
3  void setup()
4  {
5      Serial.begin(9600); // Definição da velocidade de transmissão
6  }
7
8  void loop()
9  {
10     Serial.println("Digite um numero ");
11     valor = Serial.read(); // leitura de dados do monitor serial
12     Serial.print("O numero digitado foi ");
13     Serial.write(valor);
14     Serial.println();
15     delay(2000); // Aguarda por 2 segundos
16 }
```

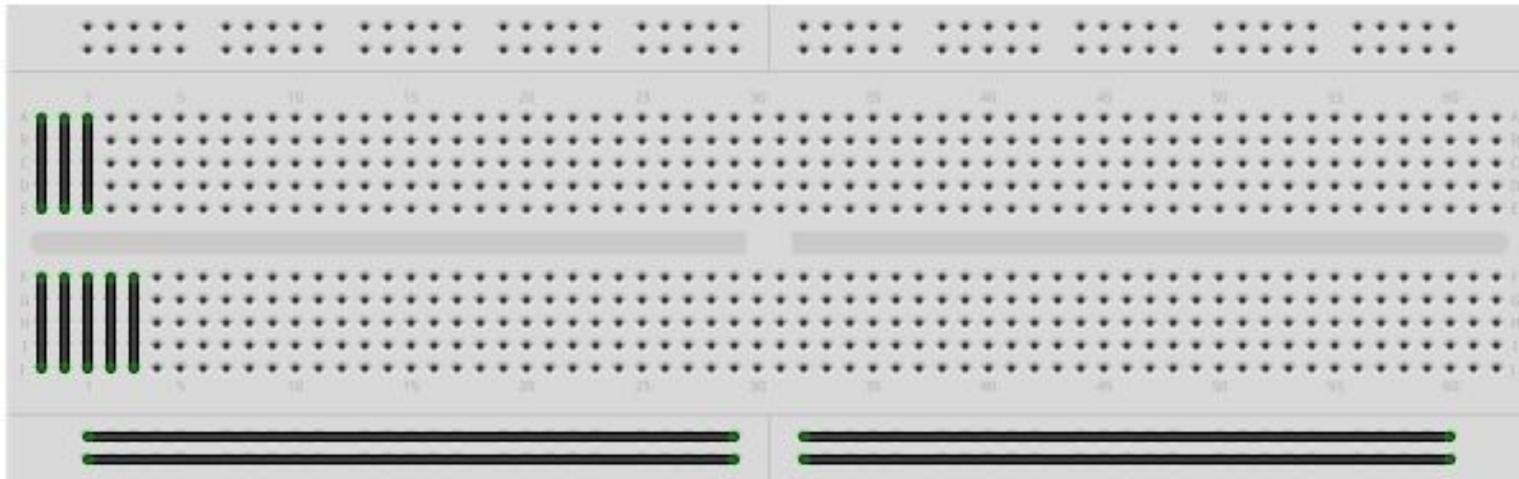
Uso do Protoboard (Matriz de Contatos)

- 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.



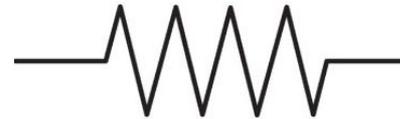
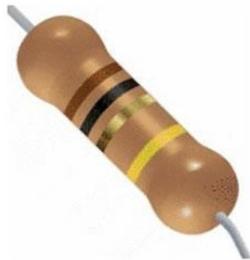
Uso do Protoboard (Matriz de Contatos)

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



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 carbono.



Resistores

- Um resistor fixo de filme carbono possui em seu corpo faixas coloridas.



- 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.

Resistores

- Tabela de cores para a identificação de resistores

Cores	Faixa 1 e 2	Faixa 3	Faixa 4
Preto	0	x1	-
Marrom	1	x10	1%
Vermelho	2	x100	2%
Laranja	3	x1000	-
Amarelo	4	x10.000	-
Verde	5	x100.000	-
Azul	6	x1.000.000	-
Violeta	7	x10.000.000	-
Cinza	8	-	-
Branco	9	-	-
Ouro	-	-	5%
Prata	-	-	10%
Sem cor	-	-	20%

Exercício 1:

Faça a leitura de cada resistor presente no kit e verifique se o padrão de cores corresponde ao valor nominal indicado. Os resistores devem ser de $1\text{ k}\Omega$, $10\text{ k}\Omega$ e $300\ \Omega$. Use a tabela de padrões de cores para resistores para confirmar os valores.

Resistores

- Exemplo:



Primeira faixa = laranja -> 3

Segunda faixa = laranja -> 3

Terceira faixa = marrom -> 10

Resistor de: $33 * 10 = 330 \text{ Ohm}$

Quarta faixa = ouro -> tolerância de 5%

Resistor de: 313.5 Ohm a 346.5 Ohm

Exercício 2:

Construa um circuito com um LED da cor de sua preferência. Varie os valores das resistências presentes no kit e observe o que acontece quando esses valores são alterados. Descreva como a mudança na resistência afeta o funcionamento dos LEDs, como o brilho ou o comportamento geral do circuito.

Resistores

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



Resistores

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

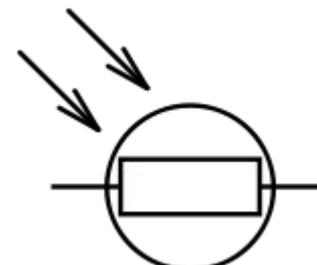
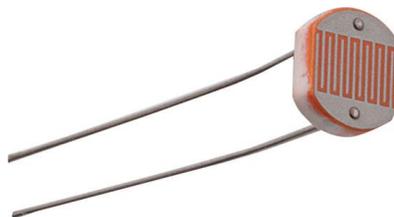


Exercício 3:

Construa um circuito com um LED da cor de sua preferência, um resistor de 300 ohms e um potenciômetro. Varie a posição do manípulo do potenciômetro e observe o que acontece. Descreva como posição do manípulo do potenciômetro afeta o funcionamento dos LEDs, como o brilho ou o comportamento geral do circuito.

Resistores

- 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.
 - O LDR converte a luz do espectro visível em resistência.
 - É um tipo de sensor muito utilizado em robótica.
 - Quanto maior a luminosidade incidida sobre ele menor será a resistência.



Programando em Arduino

- 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, um parte do programa é executada.
 - Os comandos de seleção podem ser do tipo:
 - **Seleção simples**
 - **Seleção composta**
 - **Seleção de múltipla escolha**

Programando em Arduino

- Comando de 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.

Programando em Arduino

- Comando de seleção simples
 - **Exemplo:** acendendo leds pelo monitor serial

```
1  const int led_vermelho = 5;
2  const int led_verde   = 6;
3  const int led_amarelo = 7;
4
5  char led;
6
7  void setup()
8  {
9      pinMode(led_vermelho, OUTPUT);
10     pinMode(led_verde, OUTPUT);
11     pinMode(led_amarelo, OUTPUT);
12     Serial.begin(9600);
13 }
14
15 void loop()
16 {
17     if (Serial.available()) {
18         led = Serial.read();
19
20         if (led == 'R') { // Led vermelho - red
21             digitalWrite(led_vermelho, HIGH); // Acende led
22         }
23         if (led == 'G') { // Led verde - green
24             digitalWrite(led_verde, HIGH); // Acende led
25         }
26         if (led == 'Y') { // Led amarelo - yellow
27             digitalWrite(led_amarelo, HIGH); // Acende led
28         }
29     }
30 }
```

Programando em Arduino

- Comando de 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.

Programando em Arduino

- Comando de seleção composta

- **Exemplo:** acendendo e apagando leds pelo monitor serial

```
1  const int led_vermelho = 5;
2  const int led_verde    = 6;
3  const int led_amarelo  = 7;
4
5  char led;
6
7  void setup()
8  {
9      pinMode(led_vermelho, OUTPUT);
10     pinMode(led_verde, OUTPUT);
11     pinMode(led_amarelo, OUTPUT);
12     Serial.begin(9600);
13 }
```

Programando em Arduino

- Comando de seleção composta
 - **Exemplo:** acendendo e apagando leds pelo monitor serial

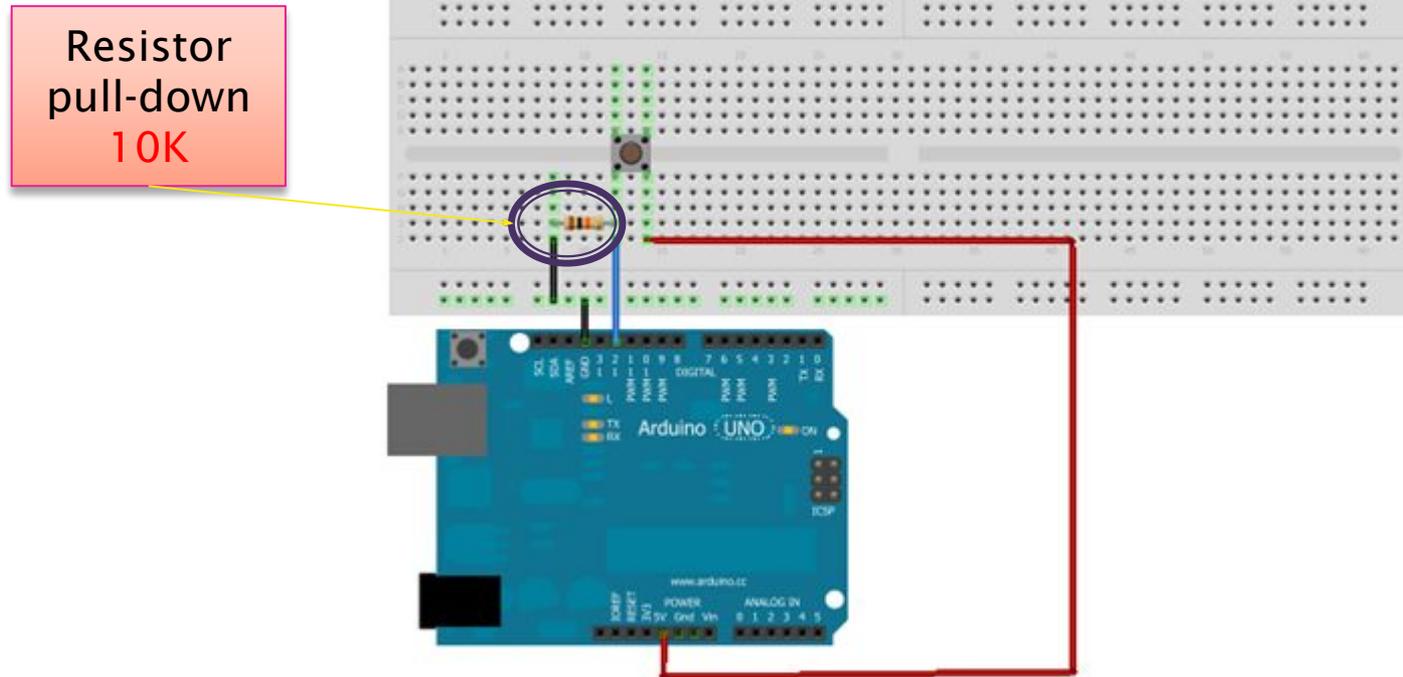
```
1 void loop()
2 {
3   if (Serial.available()) {
4     led = Serial.read();
5
6     if (led == 'R') { // Led vermelho - red
7       digitalWrite(led_vermelho, HIGH); // Acende led
8     }
9     else {
10      if (led == 'r') {
11        digitalWrite(led_vermelho, LOW); // Apaga led
12      }
13    }
14    if (led == 'G') { // Led verde - green
15      digitalWrite(led_verde, HIGH); // Acende led
16    }
17    else {
18      if (led == 'g') {
19        digitalWrite(led_verde, LOW); // Apaga led
20      }
21    }
22    if (led == 'Y') { // Led amarelo - yellow
23      digitalWrite(led_amarelo, HIGH); // Acende led
24    }
25    else {
26      if (led == 'y') {
27        digitalWrite(led_amarelo, LOW); // Apaga led
28      }
29    }
30  }
31 }
```

Programando em Arduino

- 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.

Programando em Arduino

- Lendo um botão com resistor *pull-down*
 - Ligação no protoboard



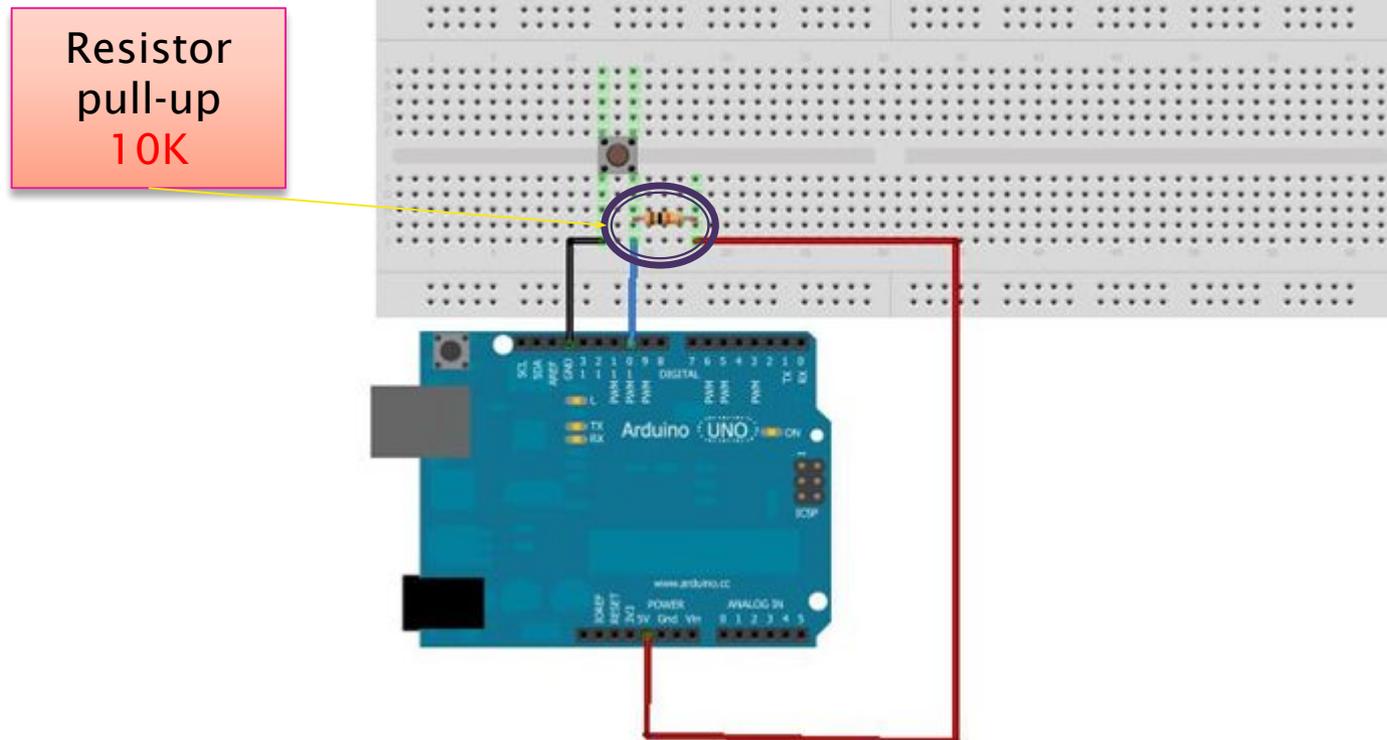
Programando em Arduino

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

```
1  const int botao = 8;
2
3  boolean vlr_btn = false;
4
5  void setup()
6  {
7      pinMode(botao, INPUT);
8      Serial.begin(9600);
9  }
10
11 void loop()
12 {
13     vlr_btn = digitalRead(botao);
14     if (vlr_btn == true) {
15         Serial.println("Botao pressionado!!!");
16     }
17 }
```

Programando em Arduino

- Lendo um botão com resistor *pull-up*
 - Ligação no protoboard



Programando em Arduino

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

```
const int botao = 8;

boolean vlr_btn = false;

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

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

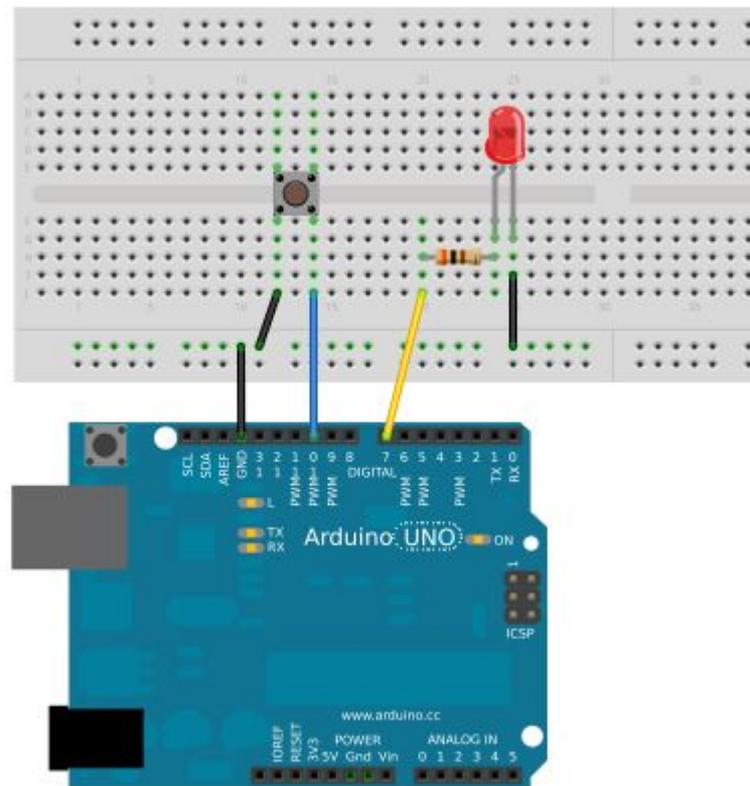
Programando em Arduino

- **Nota**

- O Arduino possui resistores *pull-up* nas portas digitais, e estes **variam de 20K a 50K**.
- 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)`

Programando em Arduino

- **Exemplo:** ativando o resistor *pull-up* de uma porta digital
 - Quanto o botão for pressionado o led irá apagar



Programando em Arduino

- **Exemplo:** ativando o resistor *pull-up* de uma porta digital
 - Quanto o botão for pressionado o led irá apagar

```
1  const int led = 7;
2  const int botao = 10;
3
4  void setup()
5  {
6      pinMode(led, OUTPUT);
7      pinMode(botao, INPUT);
8      digitalWrite(botao, HIGH); // Ativa resistor pull-up
9  }
10
11 void loop()
12 {
13     int valor = digitalRead(botao);
14
15     if (valor == HIGH) {
16         digitalWrite(led, HIGH); // Acende o led
17     }
18     else {
19         digitalWrite(led, LOW); // Apaga o led
20     }
21 }
```

Programando em Arduino

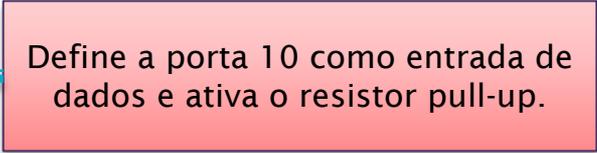
- **Exemplo:** ativando o resistor *pull-up* de uma porta digital

- 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);
}
```



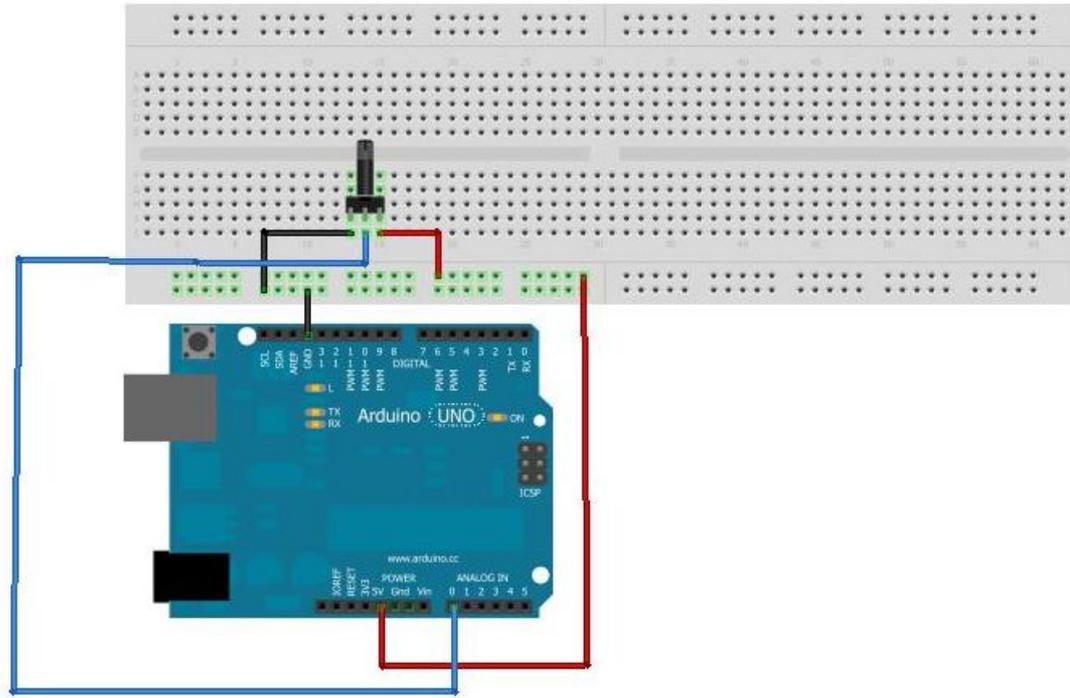
Define a porta 10 como entrada de dados e ativa o resistor pull-up.

Programando em Arduino

- Lendo Portas Analógicas
 - 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)**.

Programando em Arduino

- Lendo Portas Analógicas
 - Exemplo: lendo dados de um potenciômetro



Programando em Arduino

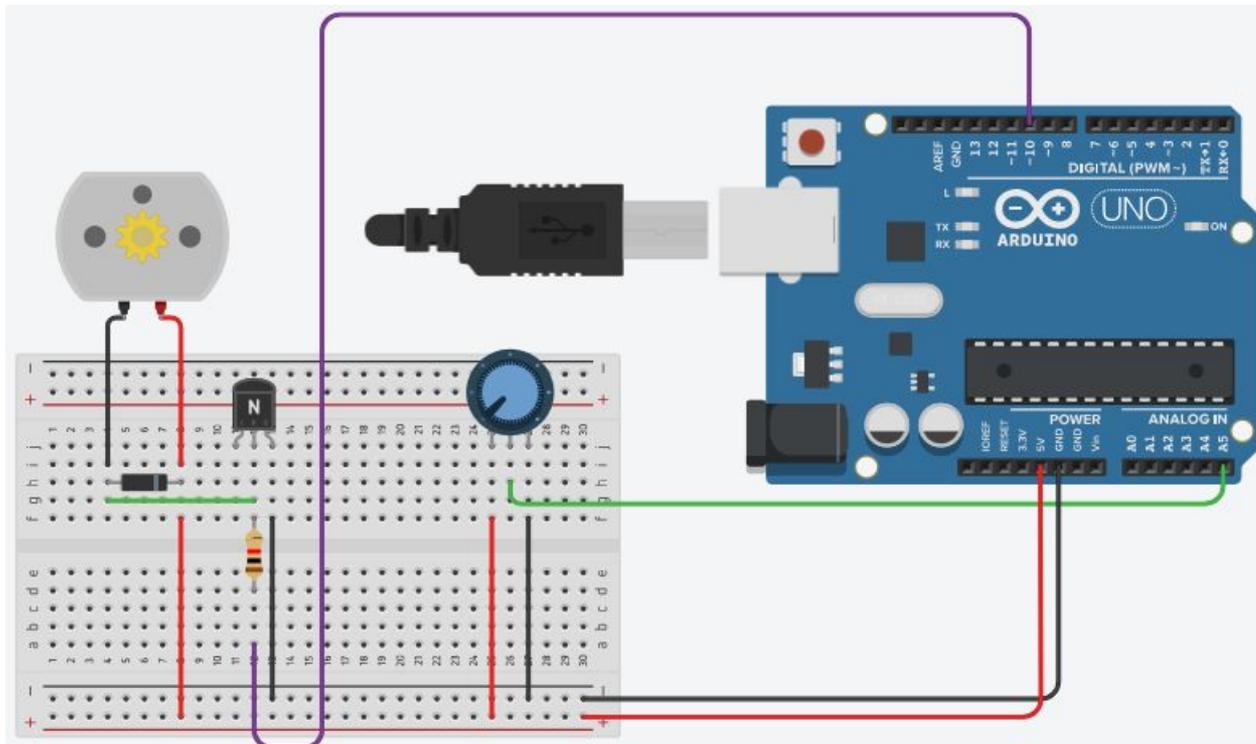
- Lendo Portas Analógicas
 - **Exemplo:** lendo dados de um potenciômetro

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

void loop()
{
  int val = analogRead(0);
  Serial.println(val);
}
```

Programando em Arduino

- Lendo Portas Analógicas
 - **Exemplo:** controlando a velocidade do motor através de um potenciômetro



Programando em Arduino

- Lendo Portas Analógicas
 - **Exemplo:** controlando a velocidade do motor através de um potenciômetro

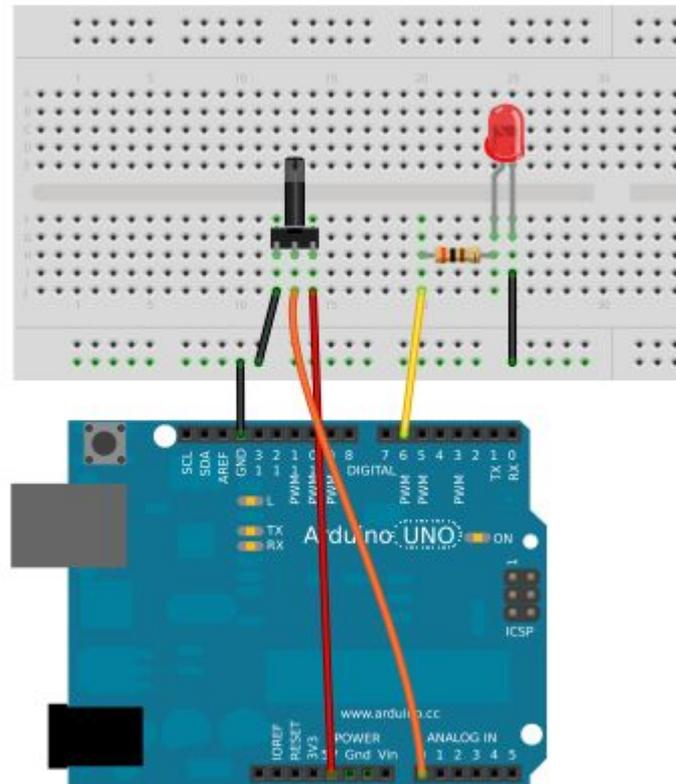
```
int valor = 0;
int pwm = 10;

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

void loop(){
  valor = map(analogRead(A0), 0, 1023, 0, 255);
  Serial.print("Valor lido=");
  Serial.println(valor);
  delay(1000);
  analogWrite(pwm, valor);
}
```

Programando em Arduino

- Lendo Portas Analógicas
 - **Exemplo:** lendo dados de um potenciômetro e acionando um LED



Programando em Arduino

- Lendo Portas Analógicas
 - **Exemplo:** lendo dados de um potenciômetro e acionando um LED

```
const int led = 6;

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

void loop()
{
  int val = analogRead(0);
  Serial.println(val);
  digitalWrite(led, HIGH);
  delay(val);
  digitalWrite(led, LOW);
  delay(val);
}
```

Programando em Arduino

- Comando de seleção de múltipla escolha
 - Na seleção de múltipla escolha é possível avaliar mais de um valor.
 - No Arduino o comando de seleção de múltipla escolha é:

```
switch (valor) {  
    case x: cmd;  
        break;  
    case y: cmd;  
        break;  
    default: cmd;  
}
```

- 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.

Programando em Arduino

- Comando de seleção de múltipla escolha
 - **Exemplo:** acendendo e apagando leds pelo monitor serial

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

char led;

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

Programando em Arduino

- Comando de seleção de múltipla escolha
 - **Exemplo:** acendendo e apagando leds pelo monitor serial

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

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

Programando em Arduino

- Comandos de Repetição
 - Muitas vezes é necessário **repetir uma determinada instrução mais de uma vez**.
 - Os **comandos de repetição** mantêm em um “laço” uma instrução ou conjunto de instruções.
 - Os **comandos de repetição** do Arduino **são**:
 - Baseado em um contador
 - Baseado em uma expressão com teste no início
 - Baseado em uma expressão com teste no final

Programando em Arduino

- Comandos de Repetição
 - Baseado 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 (contador início; expr; incremento do contador) {  
    cmd;  
}
```
 - onde:
 - *contador* = é uma variável do tipo inteiro (int)
 - *expr* = é uma expressão relacional
 - *incremento do contador* = passo de incremento do contador

Programando em Arduino

- Comandos de Repetição

- Baseado em um Contador

- **Exemplo:** escrevendo uma mensagem x vezes no monitor serial

```
int vezes = 10; // Quantidade de vezes que a mensagem será impressa no monitor serial
int executado = 0; // Quantidade de mensagens já impressas

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

void loop()
{
  for (executado; executado < vezes; executado++) {
    Serial.println("Testando o comando de repeticao for()");
  }
}
```

Programando em Arduino

- Comandos de Repetição
 - Baseado em uma expressão com teste no início
 - 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 baseado em uma expressão 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.

Programando em Arduino

- Comandos de Repetição
 - Baseado em uma expressão com teste no início

■ Exemplo:

```
const int botao = 6;
const int led   = 10;

void setup()
{
  pinMode(botao, INPUT);
  pinMode(led, OUTPUT);
  digitalWrite(botao, HIGH); // Ativa resistor pull-up
}

void loop()
{
  // Teste do comando while()
  while (digitalRead(botao)); // Espera até que o botão seja pressionado
  digitalWrite(led, HIGH);
  delay(2000);
  digitalWrite(led, LOW);
}
```

Programando em Arduino

- 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.

Programando em Arduino

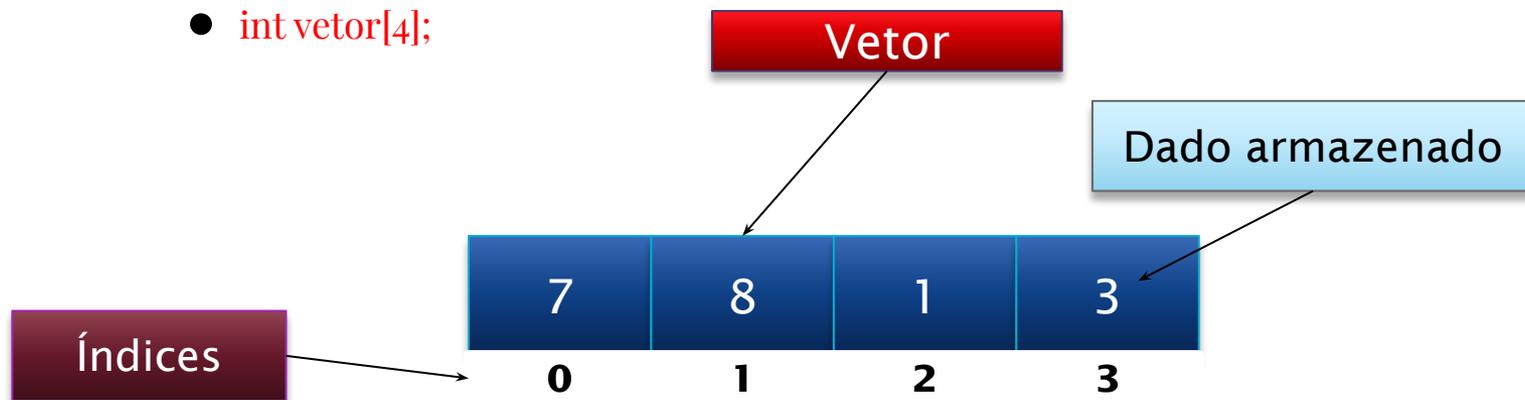
- Vetores e matrizes

- 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.

Programando em Arduino

- 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.

Programando em Arduino

- 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[o], HIGH);`
 - Ativa a porta cujo número está definido na posição `o` do vetor.

Programando em Arduino

- Vetores e matrizes

- Vetor

- Exemplo: acendendo e apagando leds cujas portas estão definidas em um vetor

```
int leds[5] = {2, 3, 4, 5, 6}; // Define as portas onde estão os leds

void setup()
{
  int i;
  for (i = 0; i < 5; i++) {
    pinMode(leds[i], OUTPUT); // Define as portas como saída
  }
}

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

Programando em Arduino

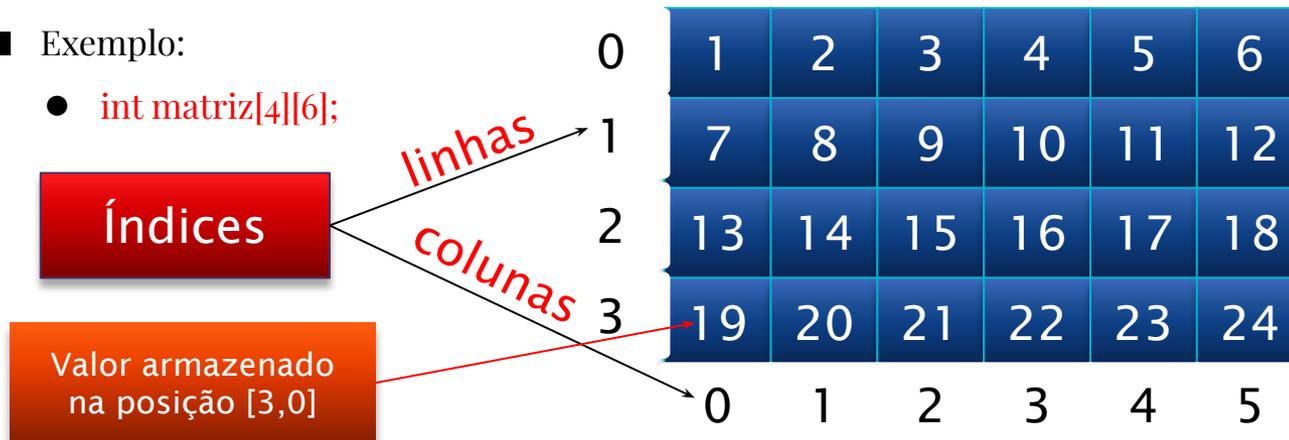
- Vetores e matrizes

- 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.

Programando em Arduino

- 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.

Programando em Arduino

- 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[o][o], HIGH);`
 - Ativa a porta cujo número está definido na posição o (linha), o (coluna) da matriz.

Programando em Arduino

- Vetores e matrizes
 - **Matriz**
 - **Exemplo: acendendo e apagando leds aleatoriamente em uma matriz**

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

void setup()
{
  int i, j;
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 2; j++) {
      // Inicializa portas
      pinMode(matriz_leds[i][j], OUTPUT);
    }
  }
  randomSeed(analogRead(0)); // Define uma semente a partir da porta analógica 0
}

void loop()
{
  int linha, coluna;

  linha = random(2); // Gera um número aleatório entre 0 e 1
  coluna = random(2); // Gera um número aleatório entre 0 e 1

  // Acende led
  digitalWrite(matriz_leds[linha][coluna], HIGH);
  delay(500);
  // Apaga led
  digitalWrite(matriz_leds[linha][coluna], LOW);
  delay(500);
}
```

Transistores

- Os transistores (*TRANSfer resISTOR*) foram criados por Bardeen, Brattain e Schockley, nos EUA em 1947, quando trabalhavam na *Bell Telephone*.
- Um transistor é um componente eletrônico formado por três materiais semicondutores.



Transistores

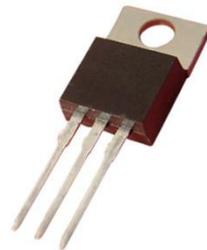
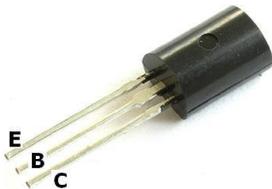
- O princípio básico de funcionamento dos transistores é o uso de uma tensão entre dois terminais para controlar o fluxo de corrente no terceiro terminal.
- Os transistores podem ser usados como chave, amplificadores de sinais e amplificadores de corrente.
- Podem ser ligados em cascata para aumentar o ganho de corrente.

Transistores

- Os terminais de um transistor são:
 - **Base (B)**: comum aos outros dois terminais. Quando está energizada a corrente flui do emissor para o coletor.
 - **Coletor (C)**: responsável por receber os portadores de carga. É a onde entra a corrente a ser controlada.
 - **Emissor (E)**: responsável por emitir portadores de carga. Saída da corrente que foi controlada pelo coletor.
- Um transistor se assemelha a dois diodos, um a esquerda e outro a direita.

Transistores

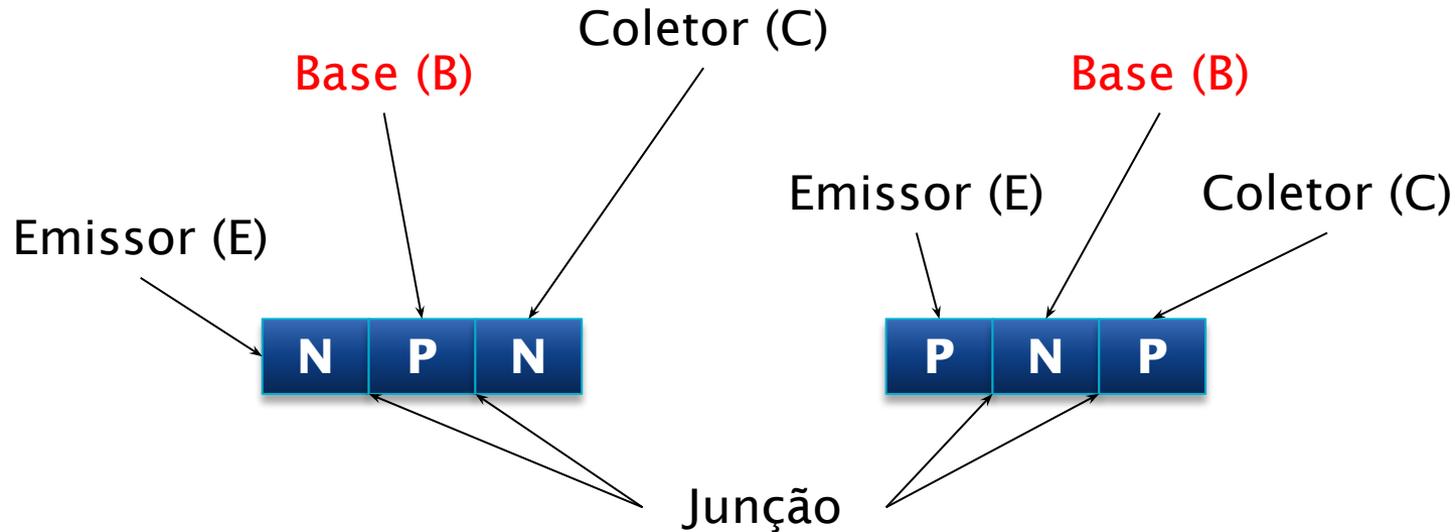
- Os transistores podem ser de:
 - **Baixa potência:** trabalham com correntes menores.
 - **Média potência:** maiores que os de baixa potência. Normalmente são acoplados a dissipadores de calor. Trabalham com correntes maiores que os de baixa potência.
 - **Alta potência:** são maiores que os de média potência e já incluem em sua estrutura um dissipador de calor. Trabalham com altas correntes.



Transistores

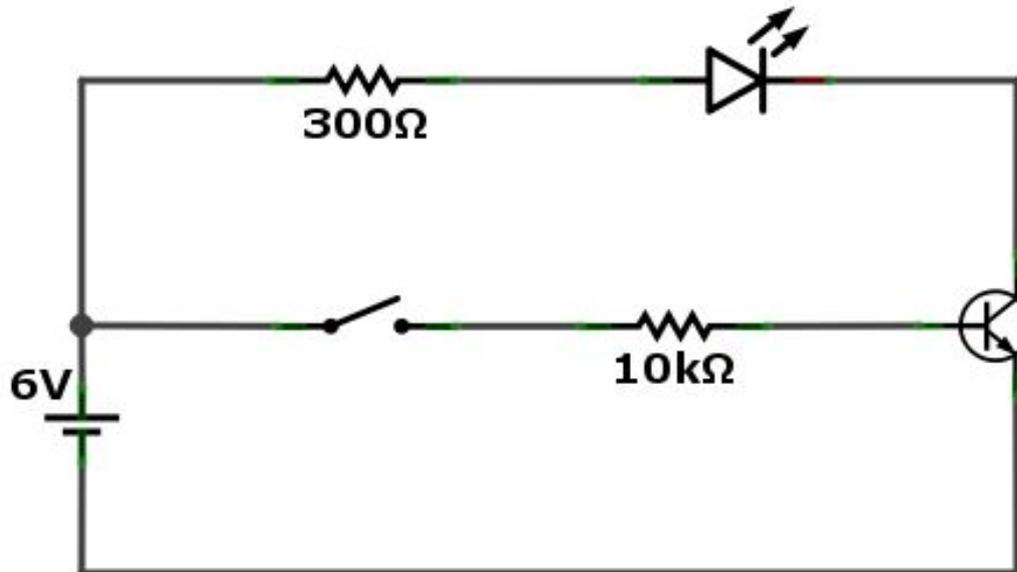
- Transistor Bipolar

- Os transistores **bipolares** podem ser do tipo **NPN** ou **PNP**.



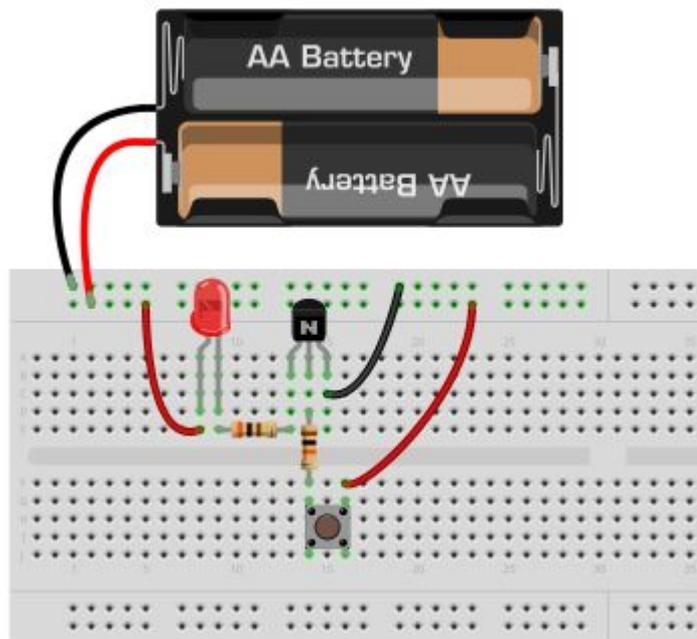
Transistores

- Exemplo
 - Uso do transistor bipolar NPN



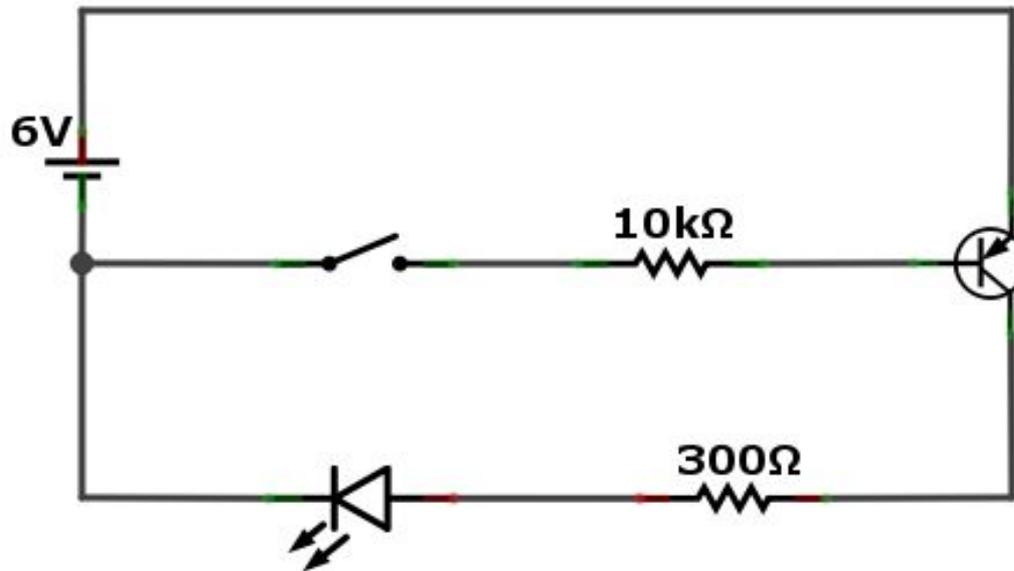
Transistores

- Exemplo
 - Uso do transistor bipolar NPN



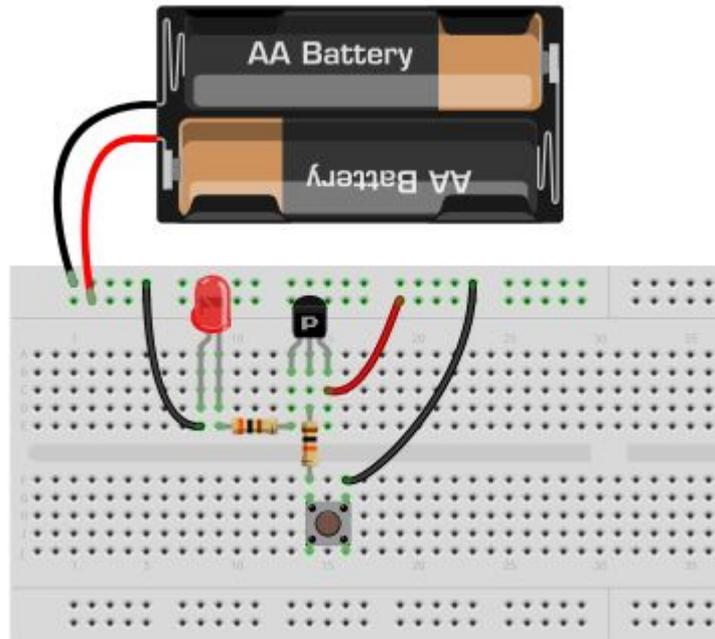
Transistores

- Exemplo
 - Uso do transistor bipolar PNP



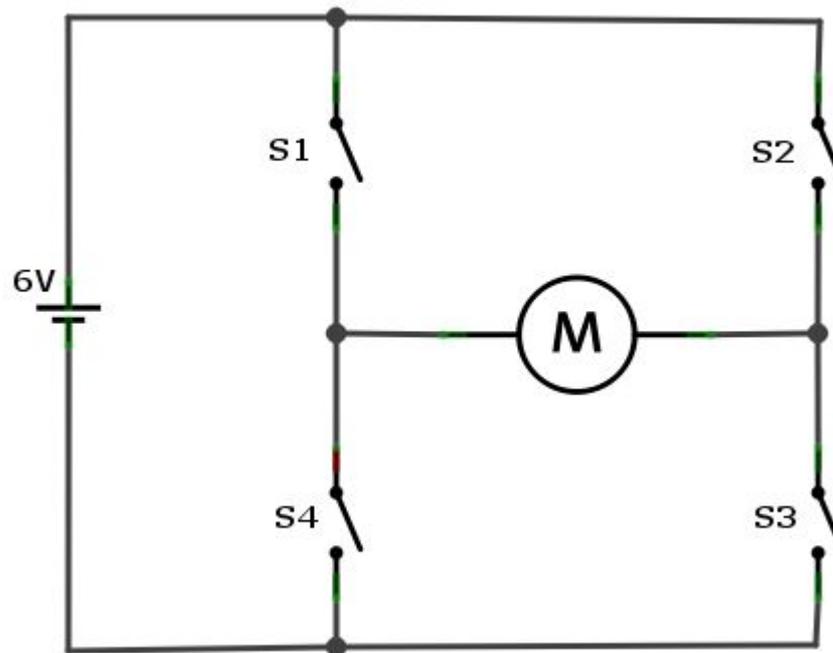
Transistores

- Exemplo
 - Uso do transistor bipolar PNP



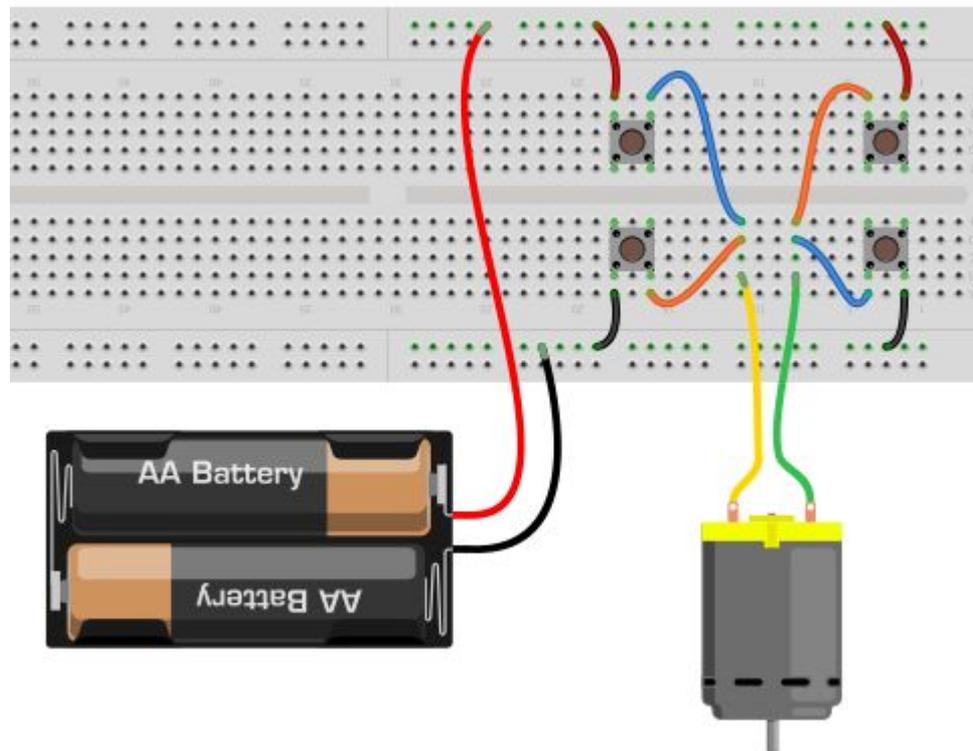
Transistores

- Exemplo
 - Circuito de controle de sentido de giro de um motor DC com botões – **circuito Ponte H**



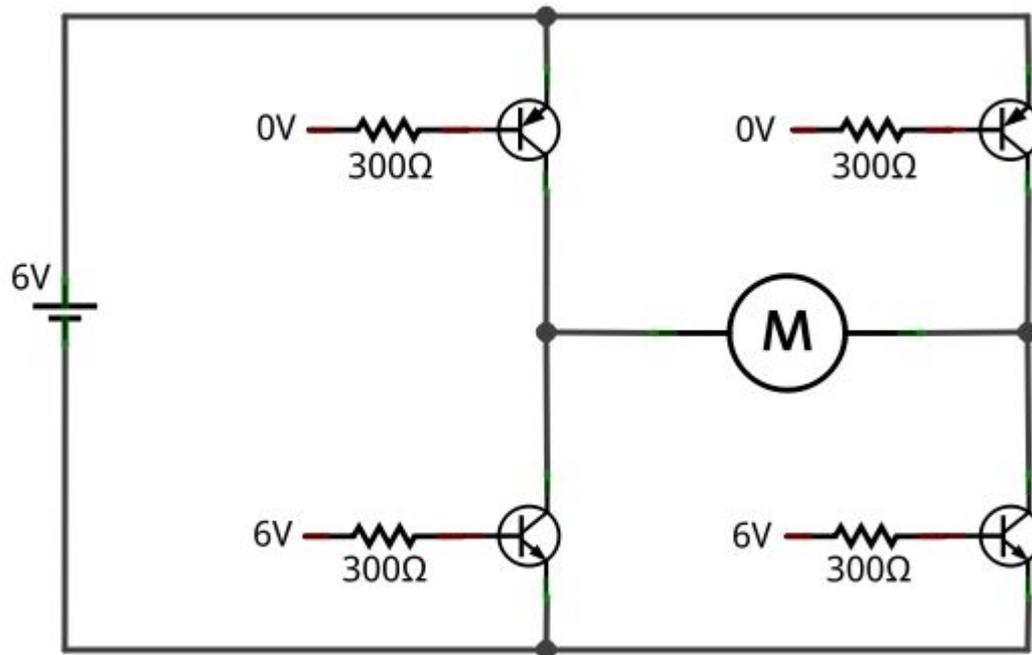
Transistores

- Exemplo
 - Circuito de controle de sentido de giro de um motor DC com botões – **circuito Ponte H**



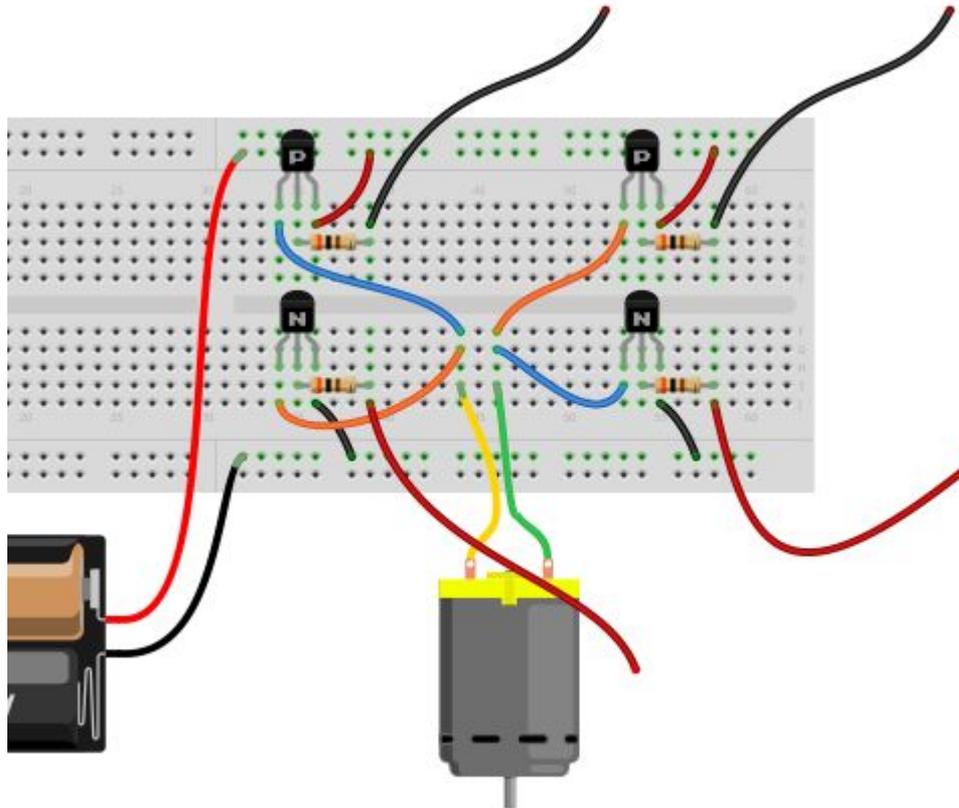
Transistores

- Exemplo
 - Circuito de controle de sentido de giro de um motor DC com transistores NPN e PNP – **circuito Ponte H**



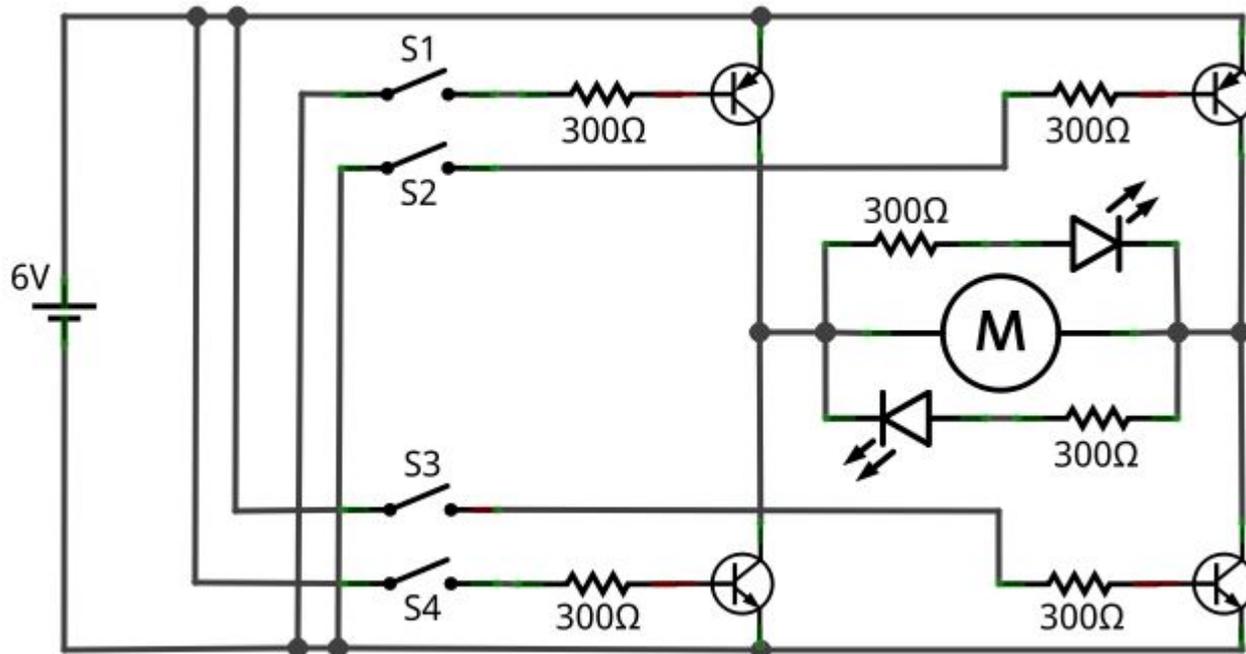
Transistores

- Exemplo
 - Circuito de controle de sentido de giro de um motor DC com transistores NPN e PNP – **circuito Ponte H**



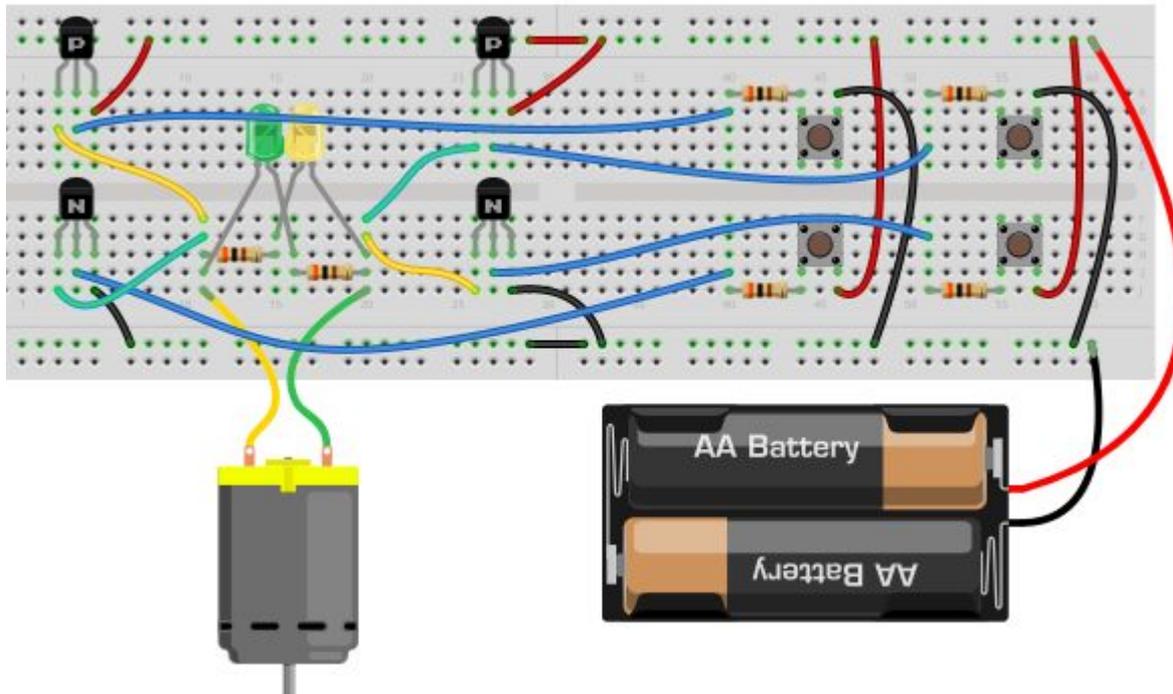
Transistores

- Exemplo
 - Circuito de controle de sentido de giro de um motor DC com transistores NPN e PNP, botões e LEDs – **circuito Ponte H**



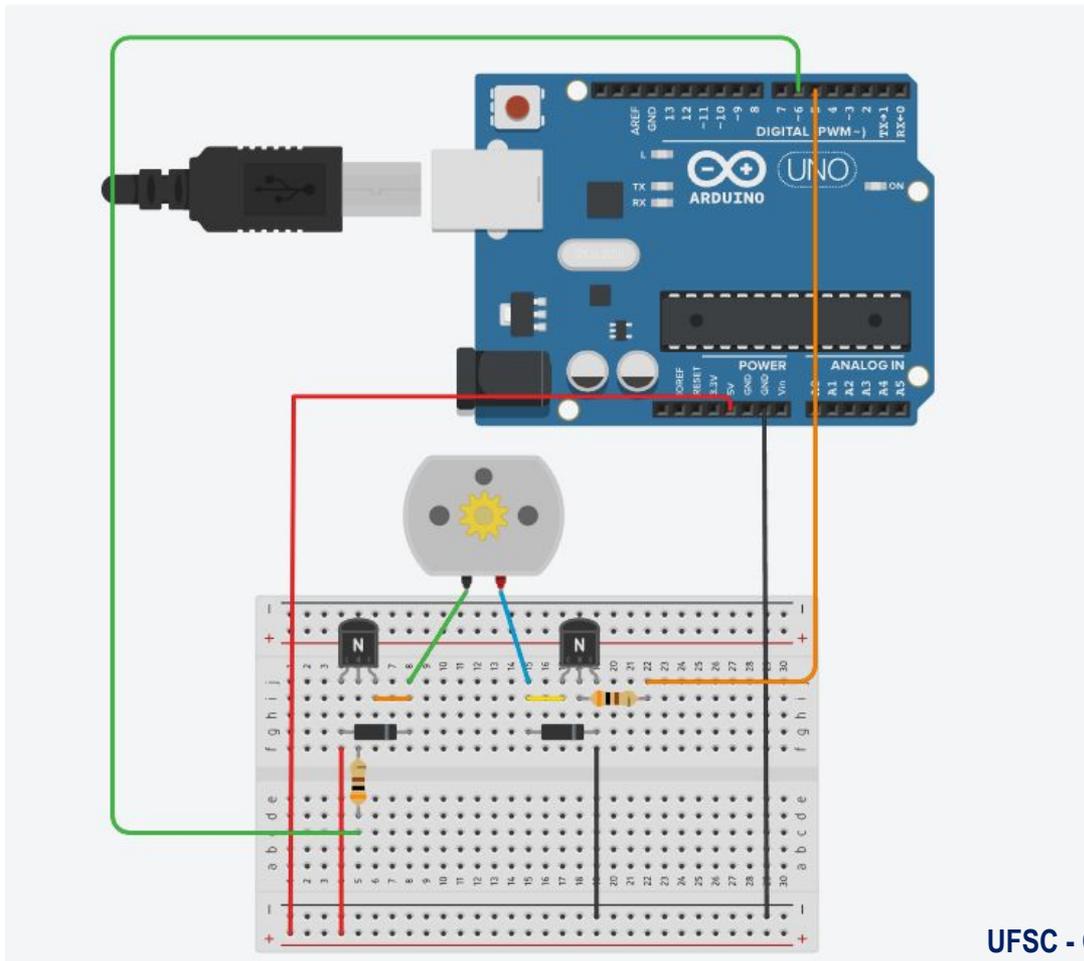
Transistores

- Exemplo
 - Circuito de controle de sentido de giro de um motor DC com transistores NPN e PNP, botões e LEDs – **circuito Ponte H**



Transistores

- Circuito de controle de sentido de giro
 - **Exemplo:** Meia Ponte H



Transistores

- Circuito de controle de sentido de giro
 - Exemplo: Meia Ponte H

```
#define PINOP1 5
#define PINOP2 6

void setup()
{
  pinMode(PINOP1, OUTPUT);
  pinMode(PINOP2, OUTPUT);
}

void loop()
{
  digitalWrite(PINOP1, LOW);
  digitalWrite(PINOP2, HIGH);
}
```

Exercício 4:

Construa um circuito controlador de led com um transistor um led da cor de sua preferência um sensor de luz(LDR). Varie a exposição do sensor LDR. Descreva os efeitos causados, como o brilho ou o comportamento geral do circuito.

Programando em Arduino

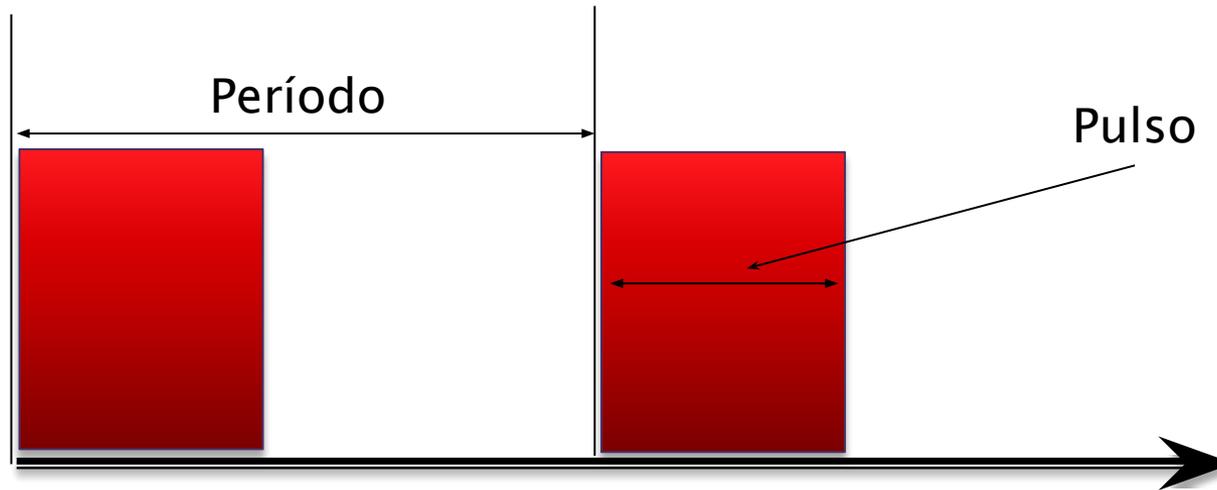
- Sinal PWM – *Pulse Width Modulation* (Modulação por Largura de Pulso)
 - O Arduino UNO possui 6 (seis) portas PWM, 3, 5, 6, 9, 10 e 11.
 - O sinal PWM pode variar de 0 a 255 e para ativá-lo basta usar a seguinte instrução em uma das portas PWM:
 - `analogWrite(pin, sinal_pwm);`
 - Note que as portas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

Programando em Arduino

- Sinal PWM – *Pulse Width Modulation* (Modulação por Largura de Pulso)
 - *Ciclo de Trabalho – Duty-Cycle*
 - O sinal PWM possui um *ciclo de trabalho* que *determina com que frequência o sinal muda do nível lógico HIGH para o nível lógico LOW* e vice versa.
 - No Arduino a frequência do PWM pode ser definida entre 32Hz até 62kHz.

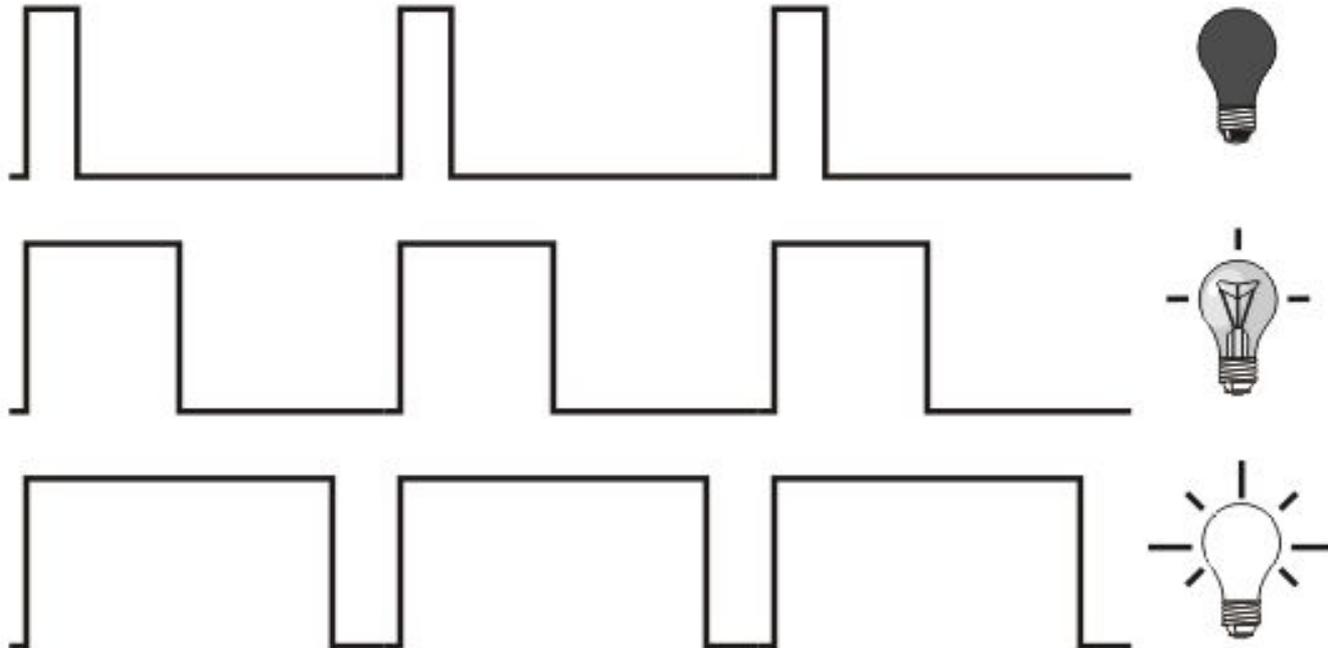
Programando em Arduino

- Sinal PWM – *Pulse Width Modulation* (Modulação por Largura de Pulso)
 - *Ciclo de Trabalho* – *Duty-Cycle*
 - $Duty\ cycle = (100\% * largura\ do\ pulso) / período$



Programando em Arduino

- Sinal PWM – *Pulse Width Modulation* (Modulação por Largura de Pulso)
 - **Exemplo PWM** – extraído de *Teach Yourself PIC Microconrollers for Absolute Beginners* – M. Amer Iqbal Qureshi 2006



Programando em Arduino

- Sinal PWM – *Pulse Width Modulation* (Modulação por Largura de Pulso)

Frequência	Tempo por troca de ciclo	Pinos
30Hz	32 milissegundos	9 e10, 11e 3
61Hz	16 milissegundos	5 e 6
122Hz	8 milissegundos	9 e10, 11e 3
244Hz	4 milissegundos	5 e 6, 11e 3
488Hz	2 milissegundos	9 e10, 11e 3
976Hz (1kHz)	1 milissegundos	5 e 6, 11e 3
3.906Hz (4kHz)	256 microssegundos	9 e10, 11e 3
7.812Hz (8kHz)	128 microssegundos	5 e 6
31.250Hz (32kHz)	32 microssegundos	9 e10, 11e 3
62.500Hz (62kHz)	16 microssegundos	5 e 6

Programando em Arduino

- Sinal PWM – *Pulse Width Modulation* (Modulação por Largura de Pulso)
 - **Exemplo:** mudando a intensidade de um led de alto brilho com sinal PWM

```
const int led_alto_brilho = 3;

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

void loop()
{
  int i;
  for (i = 10; i <= 255; i+=10) {
    analogWrite(led_alto_brilho, i); // Aumenta a intensidade do brilho
    Serial.println(i);
    delay(300);
  }
  for (i = 255; i >= 5; i-=10) {
    analogWrite(led_alto_brilho, i); // Diminui a intensidade do brilho
    Serial.println(i);
    delay(300);
  }
  delay(3000);
}
```

Programando em Arduino

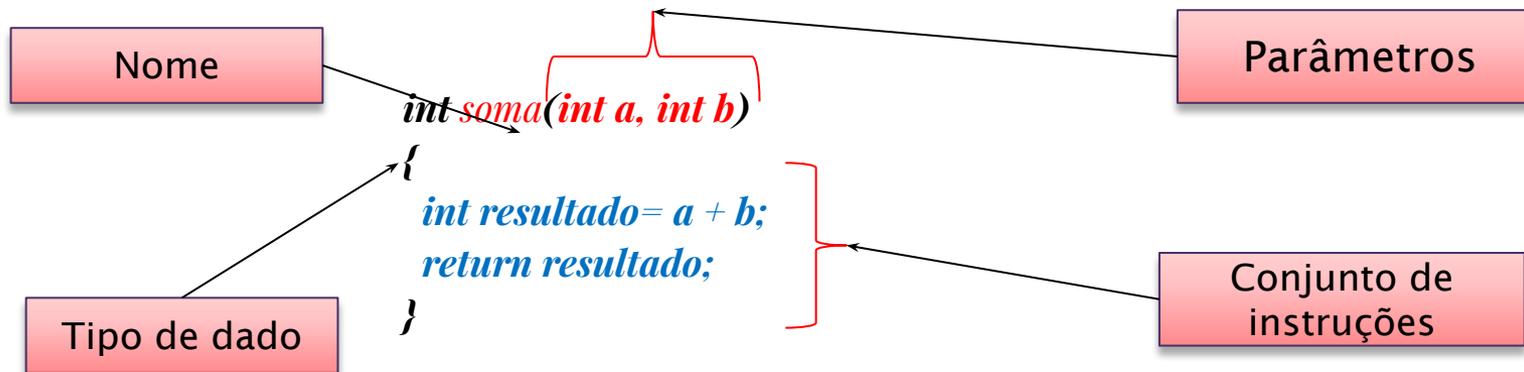
- Modularizando um Programa – **funções**
 - O **objetivo da modularização é separar o programa em módulos funcionais** – “dividir para conquistar”.
 - Um **módulo pode ser chamado** (acionado) **em qualquer ponto do programa**.
 - Os módulos funcionais de um programa também são chamados de funções.
 - **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.

Programando em Arduino

- Modularizando um Programa – **funções**

- Uma função tem quatro partes fundamentais, que são:

- um tipo de dado associado a ela (pode ser *void*);
- um nome;
- uma lista de parâmetros (se houver);
- conjunto de instruções.
- **Exemplo:**



Programando em Arduino

- Modularizando um Programa – **funções**
 - **Exemplo:** programa para acionar 4 (quatro) leds usando funções (**dispostos em matriz**)

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

void pisca_diagonal_principal() // função para controlar os leds da diagonal principal
{
    digitalWrite(matriz_leds[0][0], HIGH);
    digitalWrite(matriz_leds[1][1], HIGH);
    delay(1000);
    digitalWrite(matriz_leds[0][0], LOW);
    digitalWrite(matriz_leds[1][1], LOW);
    delay(1000);
}

void pisca_diagonal_secundaria() // função para controlar os leds da diagonal secundária
{
    digitalWrite(matriz_leds[0][1], HIGH);
    digitalWrite(matriz_leds[1][0], HIGH);
    delay(1000);
    digitalWrite(matriz_leds[0][1], LOW);
    digitalWrite(matriz_leds[1][0], LOW);
    delay(1000);
}
```

Programando em Arduino

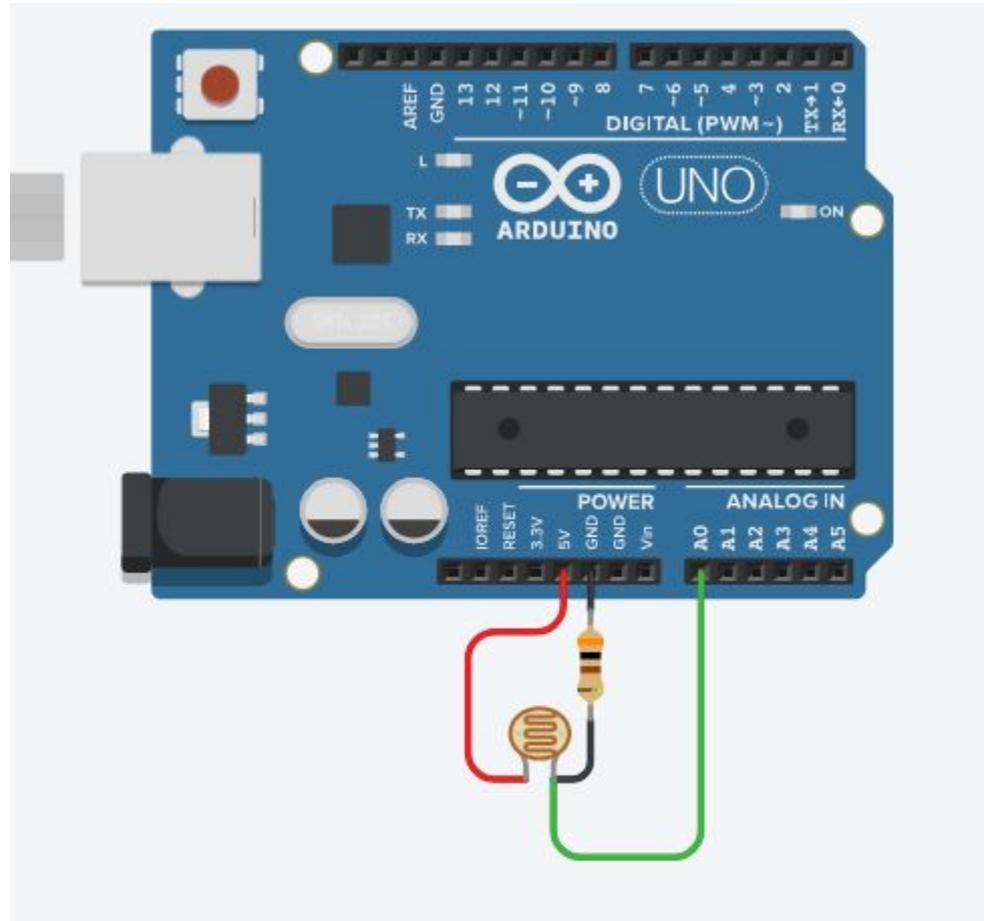
- Modularizando um Programa – **funções**
 - **Exemplo:** programa para acionar 4 (quatro) leds usando funções (**dispostos em matriz**)

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

void loop()
{
  pisca_diagonal_principal();
  pisca_diagonal_secundaria();
}
```

Programando em Arduino

- LDR (*Light Dependent Resistor* – resistor dependente de luz) com arduino



Programando em Arduino

```
int ldr = A0;//Atribui A0 a variável ldr
int valorldr = 0;//Declara a variável valorldr como inteiro

void setup() {
  pinMode(ldr, INPUT);//Define ldr (pino analógico A0) como saída
  Serial.begin(9600);//Inicialização da comunicação serial, com taxa de transferência em bits por segundo de 9600
}
void loop() {
  valorldr=analogRead(ldr);//Lê o valor do sensor ldr e armazena na variável valorldr
  Serial.print("Valor lido pelo LDR = ");//Imprime na serial a mensagem Valor lido pelo LDR
  Serial.println(valorldr);//Imprime na serial os dados de valorldr
}
```

Exercício 5:

Criar um circuito simples que utiliza um led um Arduino Uno, um sensor de luminosidade LDR (Light Dependent Resistor) e resistores de 10k para controlar automaticamente um LED. O circuito é baseado no princípio de que o LDR varia sua resistência conforme a luminosidade incidente, permitindo que o Arduino ajuste o estado do LED com base nessa variação.

Programando em Arduino

- 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.



Programando em Arduino

- Sensor de Temperatura

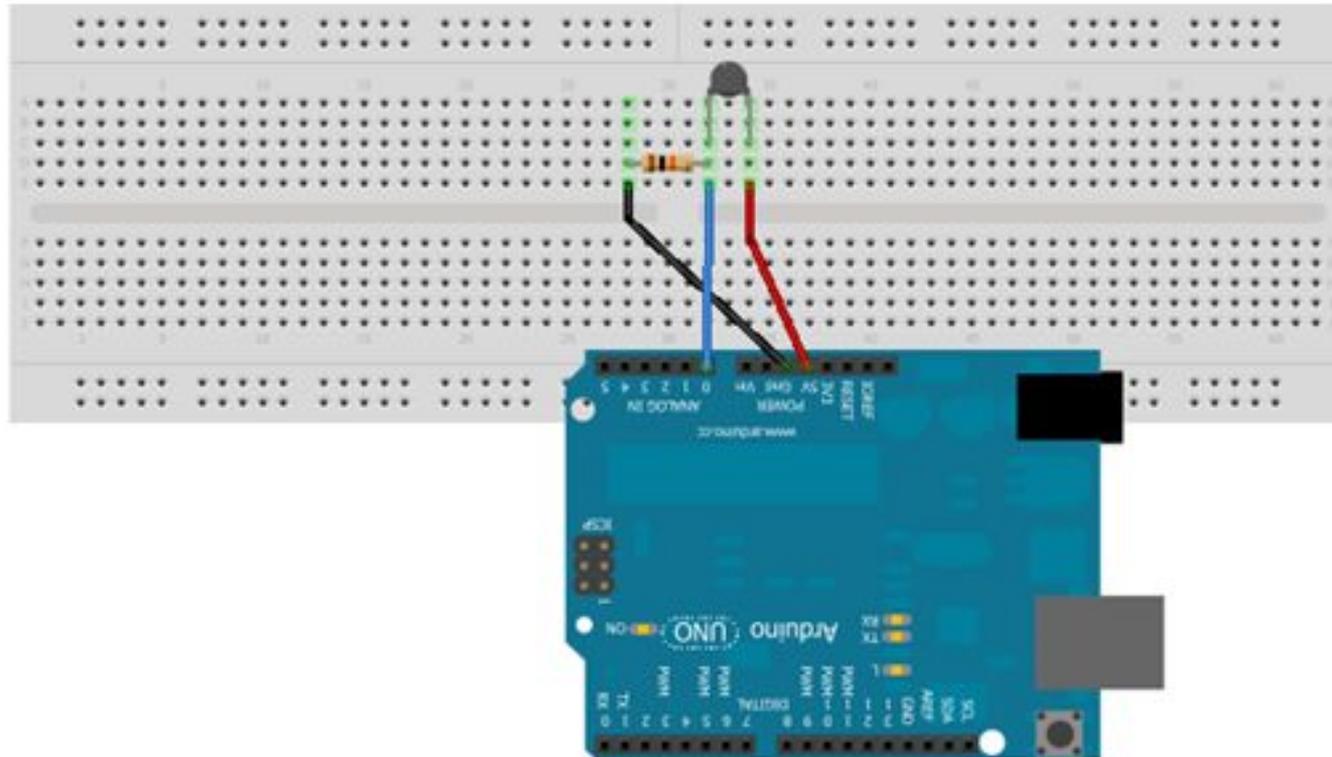
- **Equação de Steinhart-Hart**

$$1/T = a + b * \ln(R) + c * (\ln(R))^3$$

- onde:
 - T = temperatura em Kelvin
 - R = resistência em ohms
 - a, b, c: constantes definidas pelo fabricante do sensor
- Esta equação é utilizada para transformar os valores lidos pelo sensor em temperatura na escala Kelvin.
- Para encontrar a temperatura em Celsius basta subtrair o valor **273.15** da temperatura em Kelvin.

Programando em Arduino

- Sensor de Temperatura



Programando em Arduino

- Sensor de Temperatura

```
/*
Programa que utiliza a equação de Steinhart-Hart

 $1/T = a + b * \ln(R) + c * (\ln(R))^3$ 

*/
#include <math.h>

const int sensor = A0;

double tempCelsius(int valorNTC)
{
    double temp;

    temp = log(((10240000 / valorNTC) - 10000)); // Considerando resistência de 10K
    temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * temp * temp ))* temp );

    temp = temp - 273.15; // Converte Kelvin para Celsius

    return temp;
}
```

Programando em Arduino

- Sensor de Temperatura

```
void setup()  
{  
  Serial.begin(9600);  
}  
  
void loop()  
{  
  int valor = analogRead(sensor);  
  double c = tempCelsius(valor);  
  Serial.println(valor);  
  Serial.println(c);  
  delay(100);  
}
```

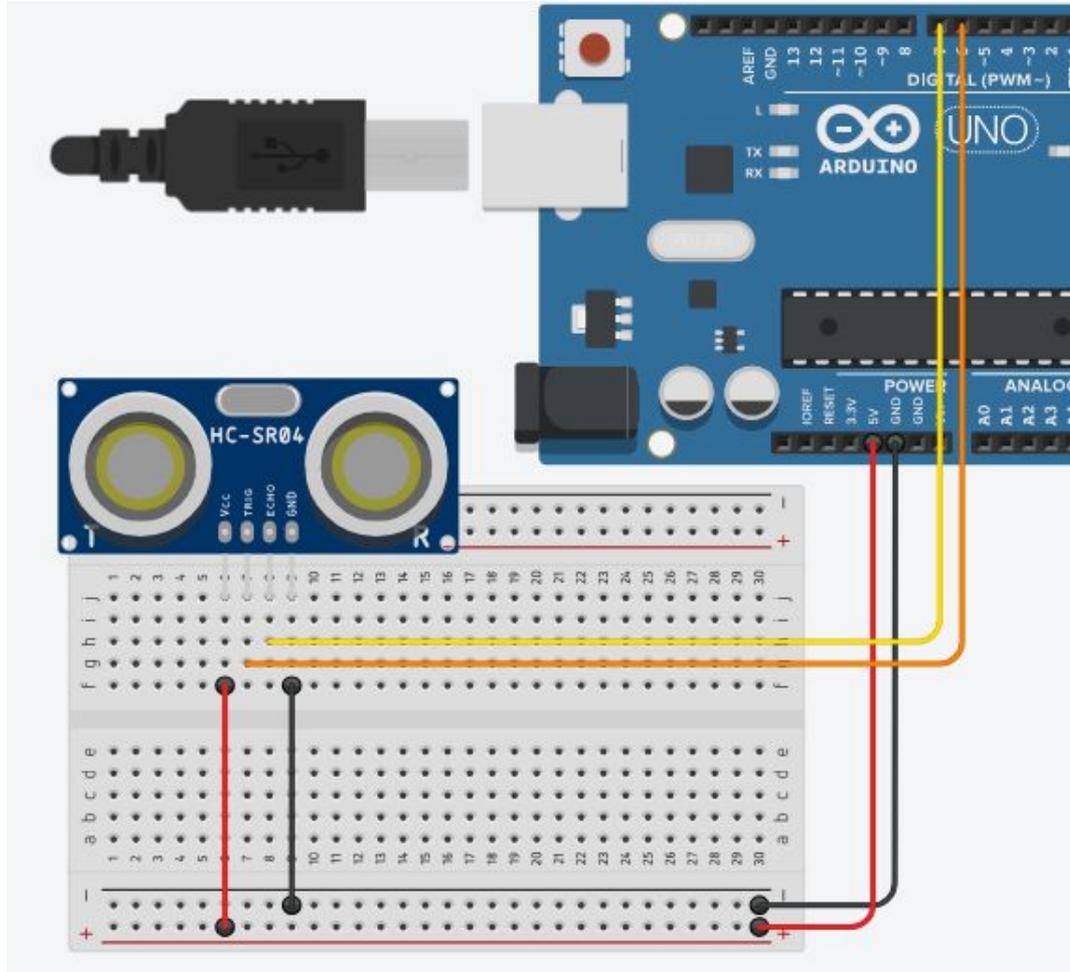
Sensor Ultrassônico

- Sensor de distância Ultrassônico
 - O sensor ultrassônico é um dispositivo eletrônico que usa ondas ultrassônicas para medir a distância de um objeto.
 - As ondas ultrassônicas são emitidas pelo sensor e refletidas pelo objeto, assim é possível medir o tempo que leva para o som refletido retornar ao sensor.



Programando em Arduino

- Sensor de distância Ultrassônico



Programando em Arduino

```
// Define os pinos do sensor
const int trigPin = 7;
const int echoPin = 6;

// Define variáveis para armazenar o tempo de duração do pulso
long duration;
int distance;

void setup() {
  // Inicializa os pinos
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Inicializa a porta serial
  Serial.begin(9600);
}
```

Programando em Arduino

```
void loop() {  
  // Gera um pulso de 10µs no pino de trigger  
  digitalWrite(trigPin, LOW);  
  delayMicroseconds(2);  
  digitalWrite(trigPin, HIGH);  
  delayMicroseconds(10);  
  digitalWrite(trigPin, LOW);  
  
  // Mede o tempo de duração do pulso de eco  
  duration = pulseIn(echoPin, HIGH);  
  
  // Calcula a distância em centímetros  
  distance = duration * 0.034 / 2;  
  
  // Exibe a distância no monitor serial  
  Serial.print("Distancia: ");  
  Serial.print(distance);  
  Serial.println(" cm");  
  
  // Aguarda um curto intervalo antes da próxima leitura  
  delay(1000);  
}
```