Oficina de Robótica

Programação em Arduino

Módulo Básico

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
    - renanrdaros (at) hotmail.com

- http://oficinaderobotica.ufsc.br

Sumário

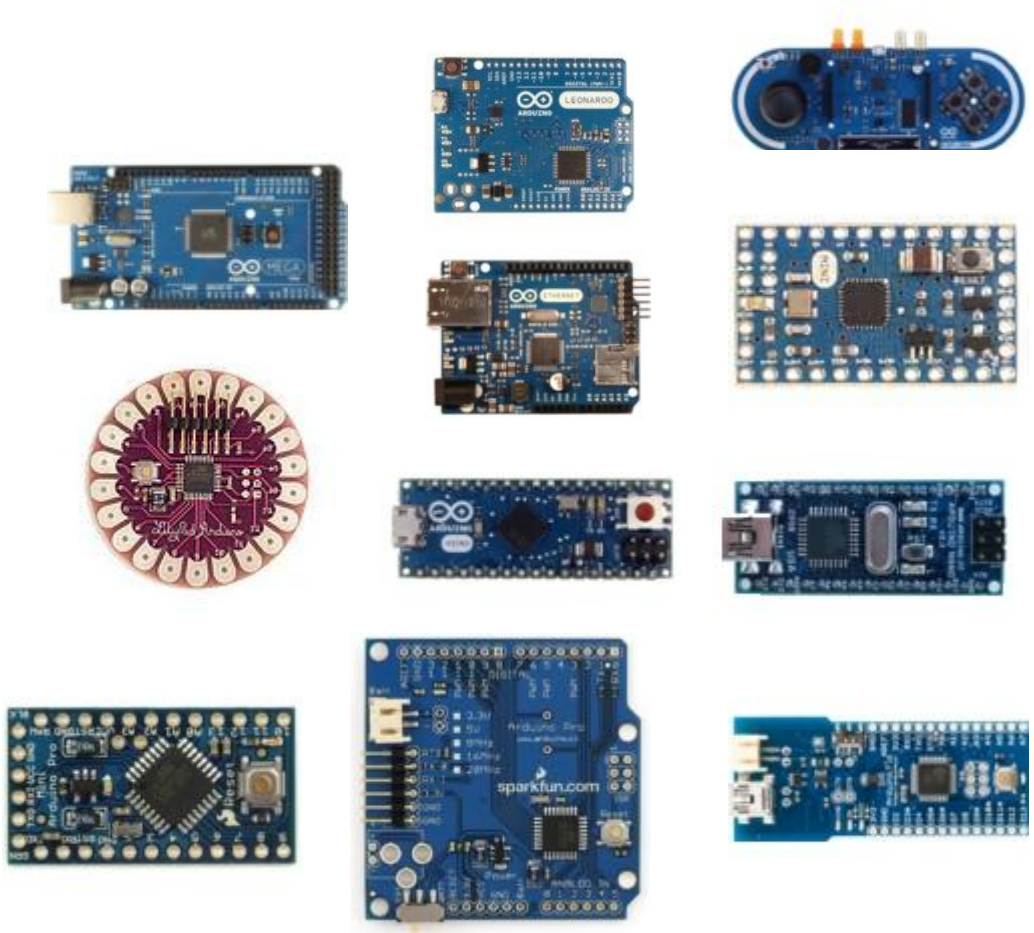
- Introdução
- Microcontroladores
- Arduino UNO
- Ambiente de desenvolvimento
- Funções *setup()* e *loop()*
- Monitor Serial
- Portas digitais e analógicas
- Programando em Arduino
- Expandindo as funcionalidades do Arduino

Introdução

- 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.

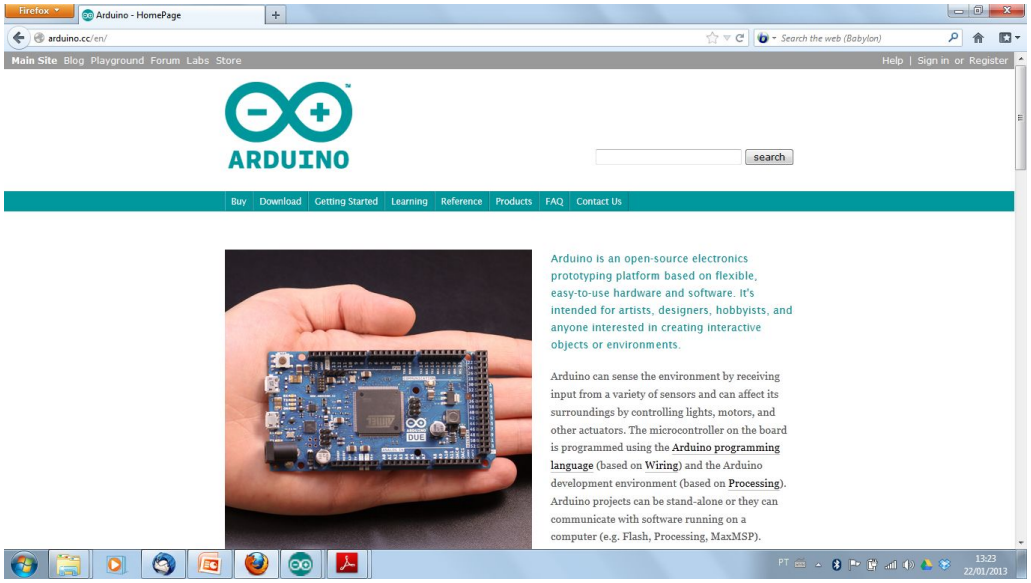
Introdução

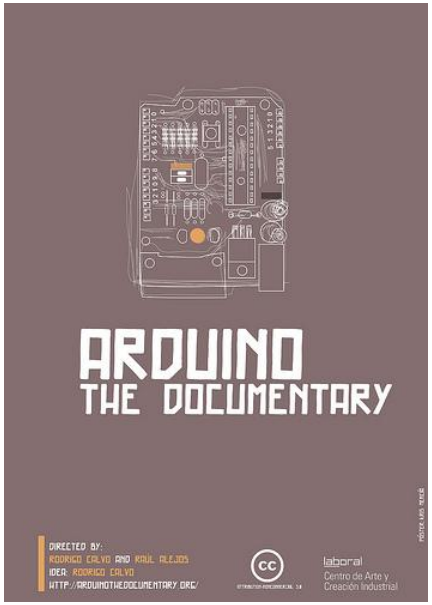
- Tipos de Arduino
  - Existem vários tipos de Arduino com especificidades de hardware. O site oficial do Arduino lista os seguintes tipos:
    - Arduino UNO
    - Arduino Leonardo
    - Arduino Due
    - Arduino Esplora
    - Arduino Mega
    - Arduino Mega ADK
    - Arduino Ethernet
    - Arduino Mini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



Introdução

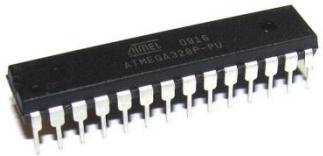
- Referências na WEB:
  - O site oficial do Arduino é http://arduino.cc
  - Um documentário sobre o Arduino pode ser assistido em: http://arduinothedocumentary.org/





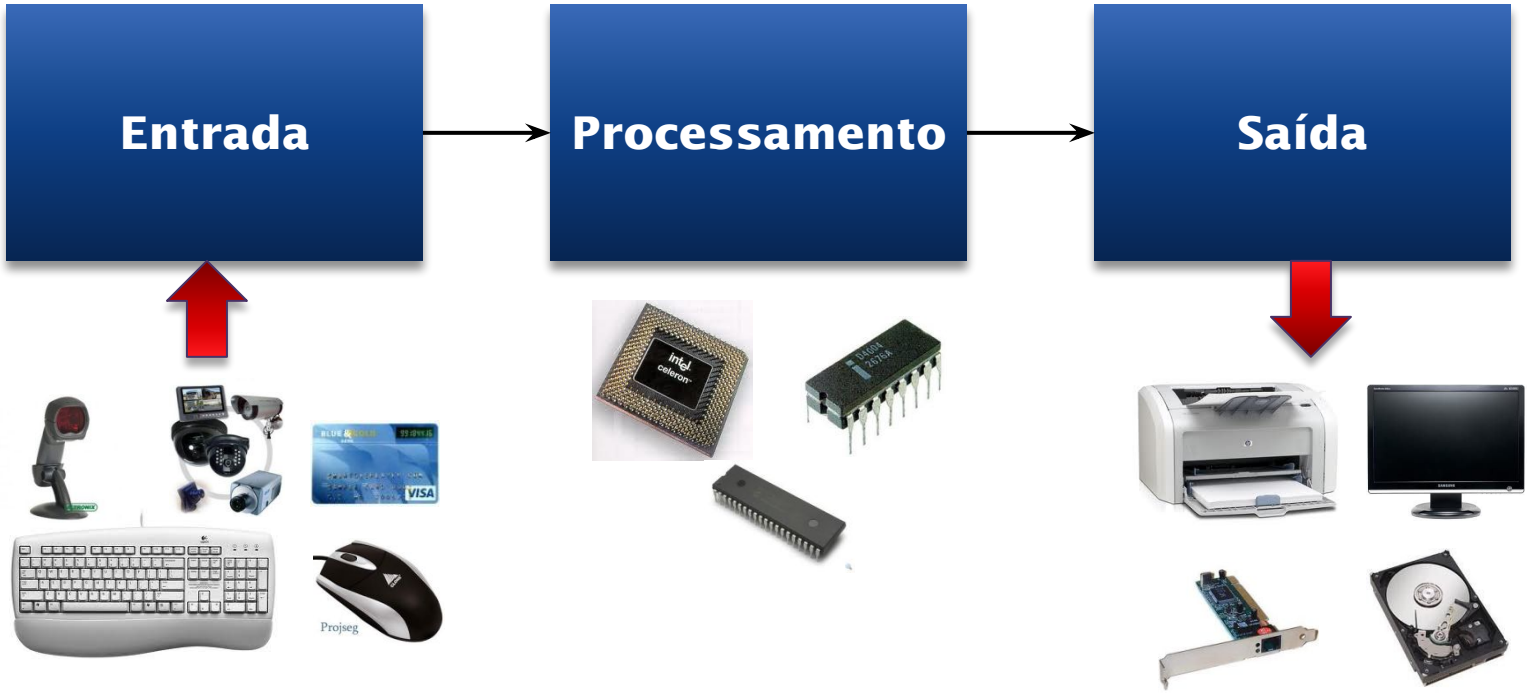
Microcontroladores

- 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.



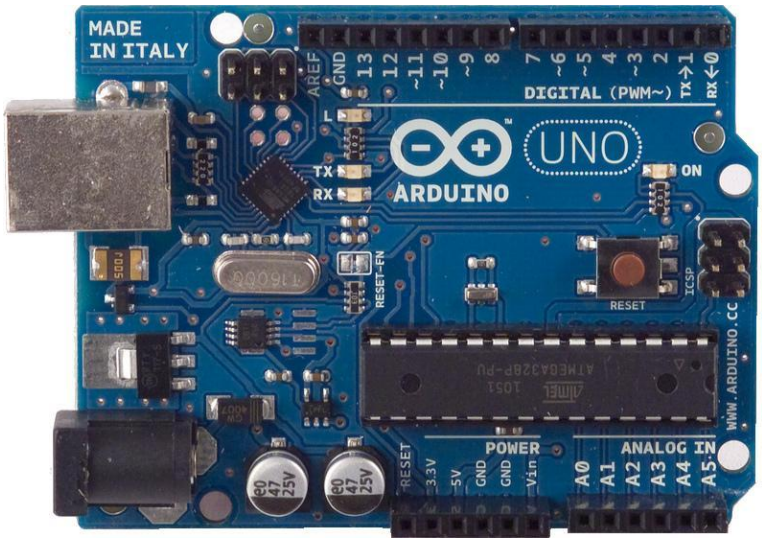
Microcontroladores

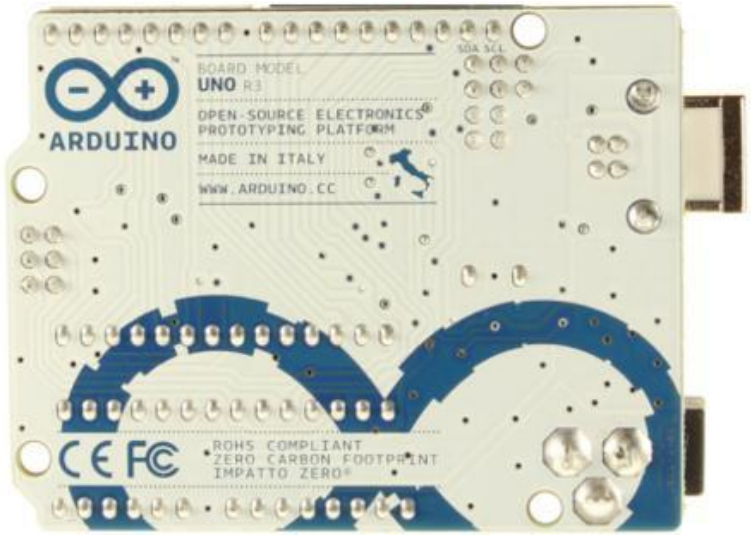
- Processamento de dados



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

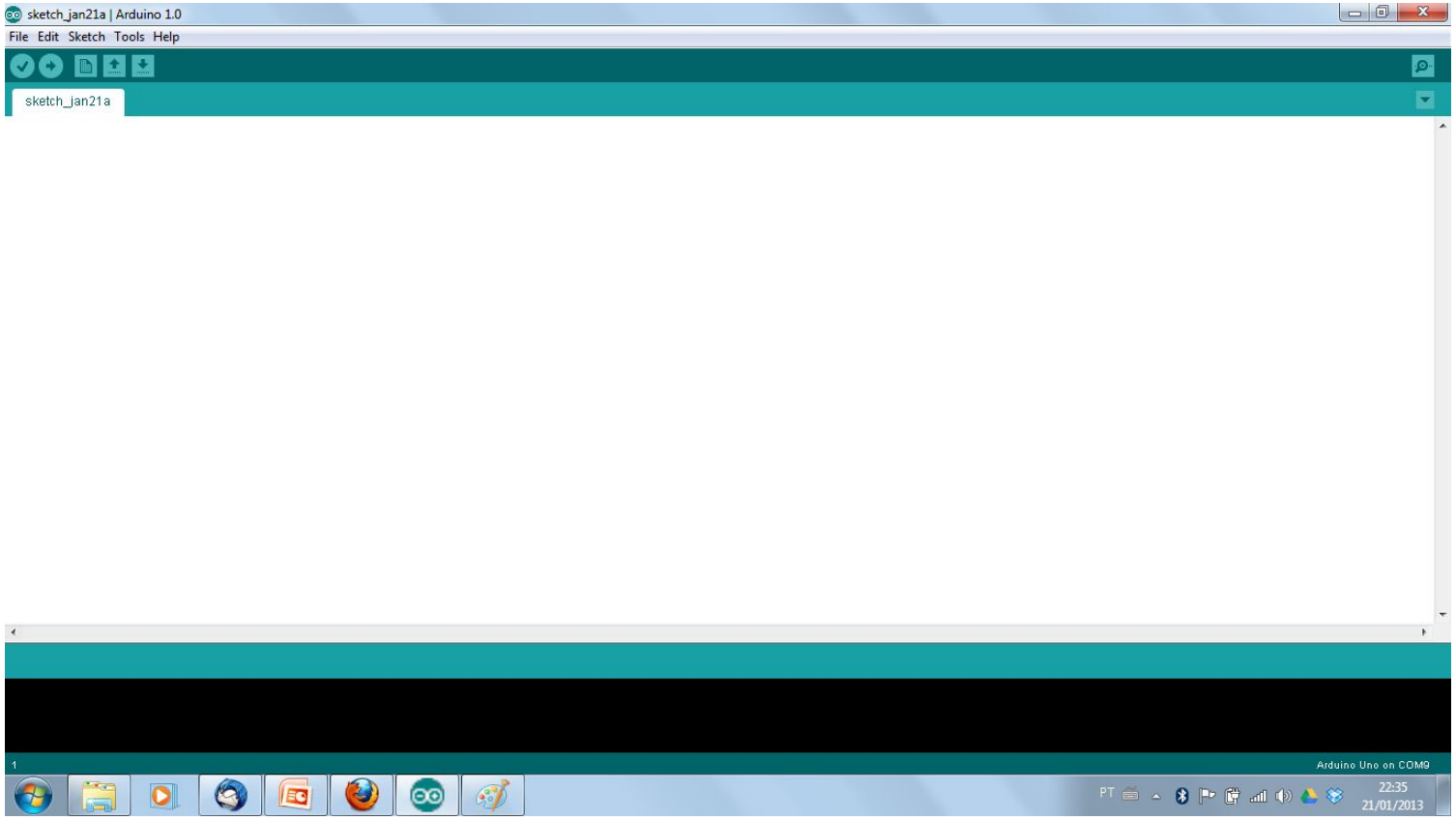
- Alimentação
  - O Arduino UNO pode ser alimentado pela porta USB ou por uma fonte externa DC.
  - A recomendação é que a fonte externa seja de 7 V a 12 V e pode ser ligada diretamente no conector de fonte ou nos pinos Vin e Gnd.

Ambiente de desenvolvimento

- O ambiente de desenvolvimento do Arduino (IDE) é gratuito e pode ser baixado no seguinte endereço: arduino.cc.
- 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
```

Exercício 1:

Crie um código em Arduino que simule o funcionamento de um semáforo para veículos, utilizando LEDs nas cores verde, amarelo e vermelho. O semáforo deve seguir a seguinte sequência:

1. **Verde:** Fica aceso por 5 segundos, indicando que os veículos podem prosseguir.
2. **Amarelo:** Fica aceso por 2 segundos, indicando que os veículos devem reduzir a velocidade.
3. **Vermelho:** Fica aceso por 5 segundos, indicando que os veículos devem parar.

O ciclo deve se repetir indefinidamente. Utilize os pinos digitais do Arduino para controlar os LEDs.

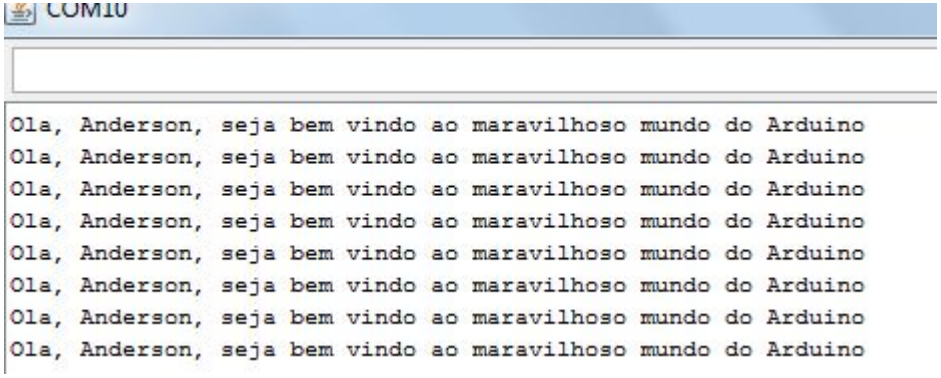
Monitor Serial

- 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 velocide de transmissão
4  }
5
6  void loop()
7  {
8    Serial.println("Ola, seu nome, seja bem vindo ao maravilhoso mundo do Arduino");
9  }
```



Exercício 2:

O monitor serial é uma ferramenta essencial para comunicação entre o Arduino e o computador, permitindo o envio e recebimento de dados em formato de texto. Ele é útil para depuração de código e interação com o Arduino através do teclado do PC, então como objetivo faça aparecer no monitor serial “Hello, World!”.

Portas digitais e analógicas

- 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 e analógicas

- 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*).

Portas digitais e analógicas

- Portas Analógicas
  - As portas analógicas são utilizadas para entrada de dados.
  - Os valores lidos em uma porta analógica variam de 0V a 5V.
  - Para ler uma valor em uma porta analógica basta utilizar a função analogRead(*pin*).
  - Os conversores analógicos-digitais (ADC) do Arduino são de 10 bits.
  - Os conversores ADC (do Inglês *Analog Digital Converter*) permitem uma precisão de 0.005V ou 5mV.
  - Os valores lidos em uma porta analógica variam de 0 a 1023 (10 bits), onde 0 representa 0V e 1023 representa 5V.

Portas digitais e analógicas

- 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)

Programando em Arduino

- Algoritmo
  - Sequência de passos que visa atingir um objetivo bem definido.
  - Exemplo: Receita caseira

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!

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  }
17
```

Programando em Arduino

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

Programando em Arduino

- Operadores aritméticos

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

Programando em Arduino

- Operadores relacionais

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

Programando em Arduino

- Operadores lógicos

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

Programando em Arduino

- Operadores compostos

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

Programando em Arduino

- 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.

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  }
14
```

Programando em Arduino

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

```
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      else {
24        if (led == 'r') {
25          digitalWrite(led_vermelho, LOW);  // Apaga led
26        }
27      }
28      if (led == 'G') {  // Led verde - green
29        digitalWrite(led_verde, HIGH); // Acende led
30      }
31      else {
32        if (led == 'g') {
33          digitalWrite(led_verde, LOW); // Apaga led
34        }
35      }
36      if (led == 'Y') {  // Led amarelo - yellow
37        digitalWrite(led_amarelo, HIGH); // Acende led
38      }
39      else {
40        if (led == 'y') {
41          digitalWrite(led_amarelo, LOW); // Apaga led
42        }
43      }
44    }
45  }
```

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: cmd1;
            break;
    case y: cmd2;
            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

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

Programando em Arduino

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

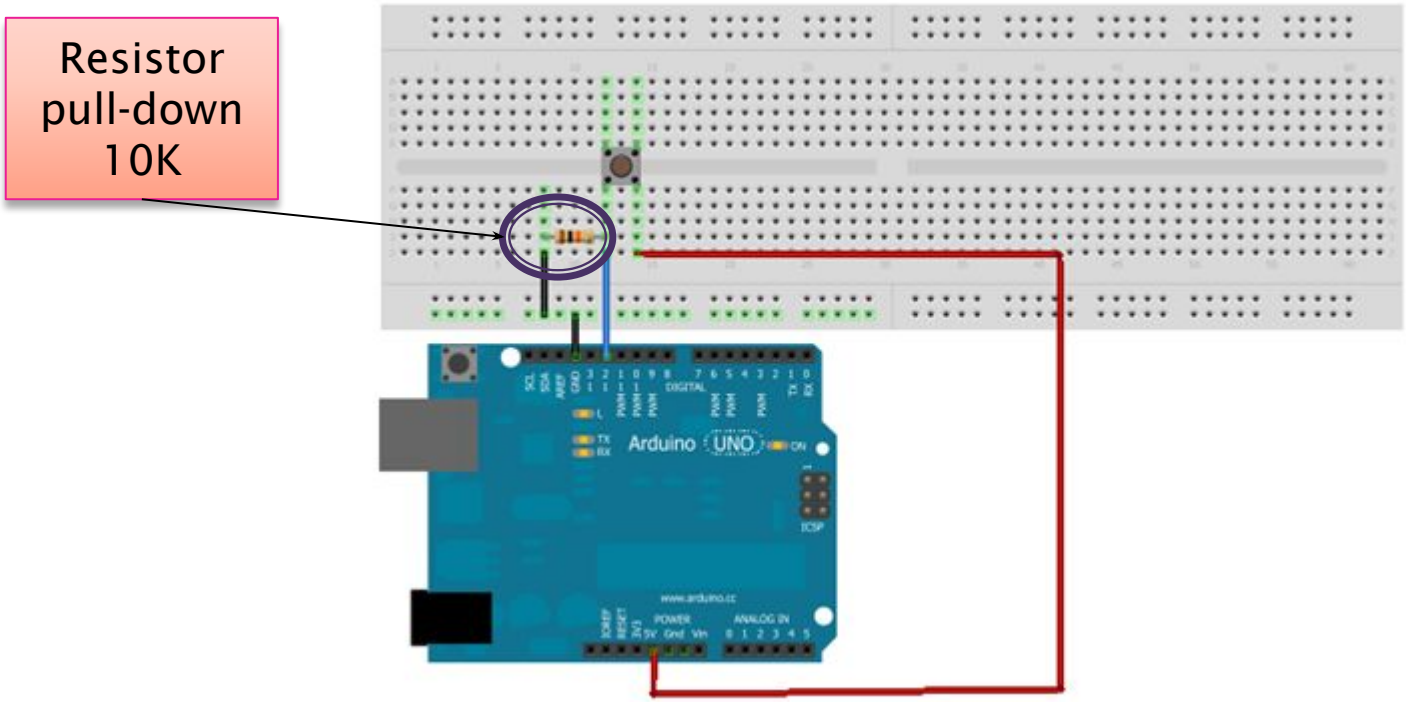
```
14  void loop()
15  {
16    if (Serial.available()) {
17      led = Serial.read();
18
19      switch (led) {
20        case 'R': digitalWrite(led_vermelho, HIGH);  // Acende led
21                  break;
22        case 'r': digitalWrite(led_vermelho, LOW);   // Apaga led
23                  break;
24        case 'G': digitalWrite(led_verde, HIGH);  // Acende led
25                  break;
26        case 'g': digitalWrite(led_verde, LOW);   // Apaga led
27                  break;
28        case 'Y': digitalWrite(led_amarelo, HIGH);  // Acende led
29                  break;
30        case 'y': digitalWrite(led_amarelo, LOW);   // Apaga led
31                  break;
32        default: Serial.println("Nenhum led selecionado!!!");
33      }
34    }
35  }
```

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



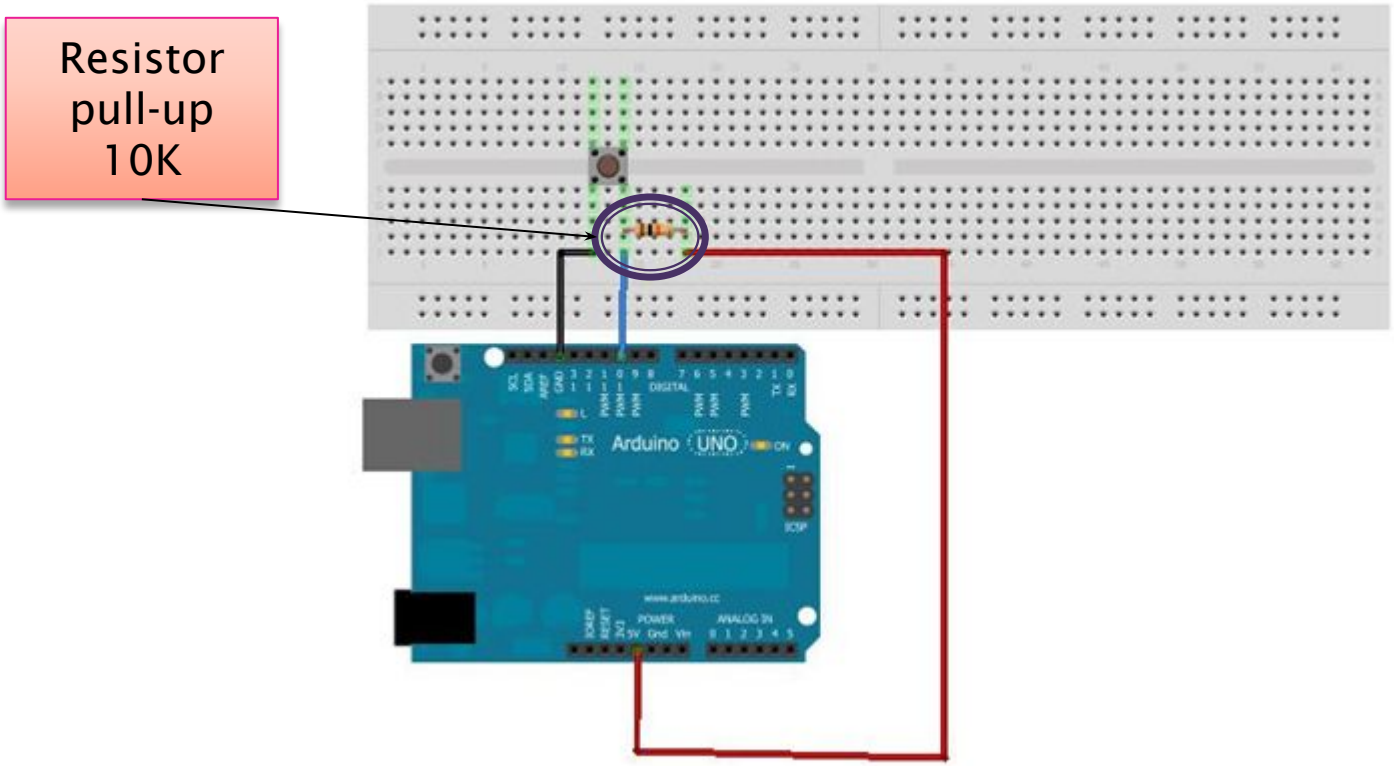
Programando em Arduino

- 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

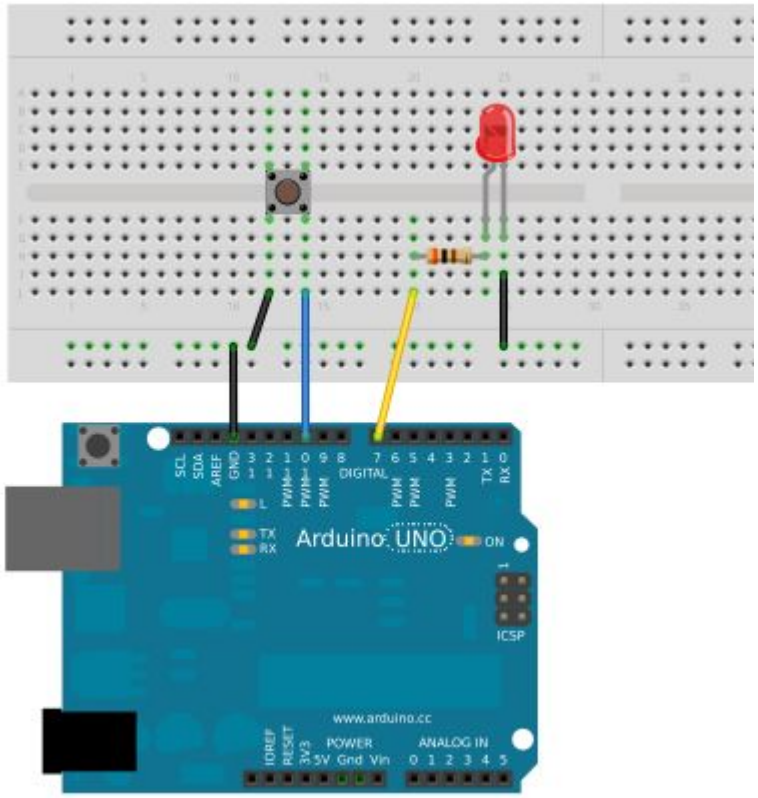
```
1   const int botao  = 8;
2
3   boolean vlr_btn = false;
4
5   void setup()
6   {
7     pinMode(botao, INPUT);
8     Serial.begin(9600);
9   }
10
11  void loop()
12  {
13    vlr_btn = digitalRead(botao);
14    if (vlr_btn == false) {
15      Serial.println("Botao pressionado!!!");
16    }
17  }
```

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

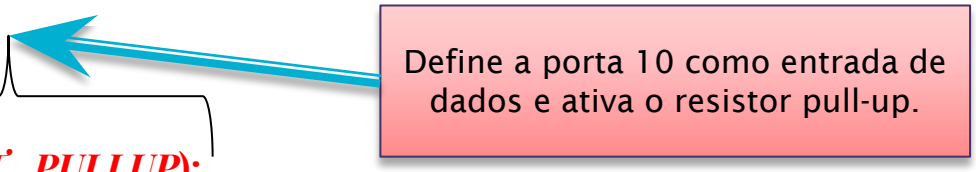
- Exemplo: ativando o resistor *pull-up* de uma porta digital
  - Quanto o botão for pressionado o led irá apagar

```
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  }
22
```

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);
}
```

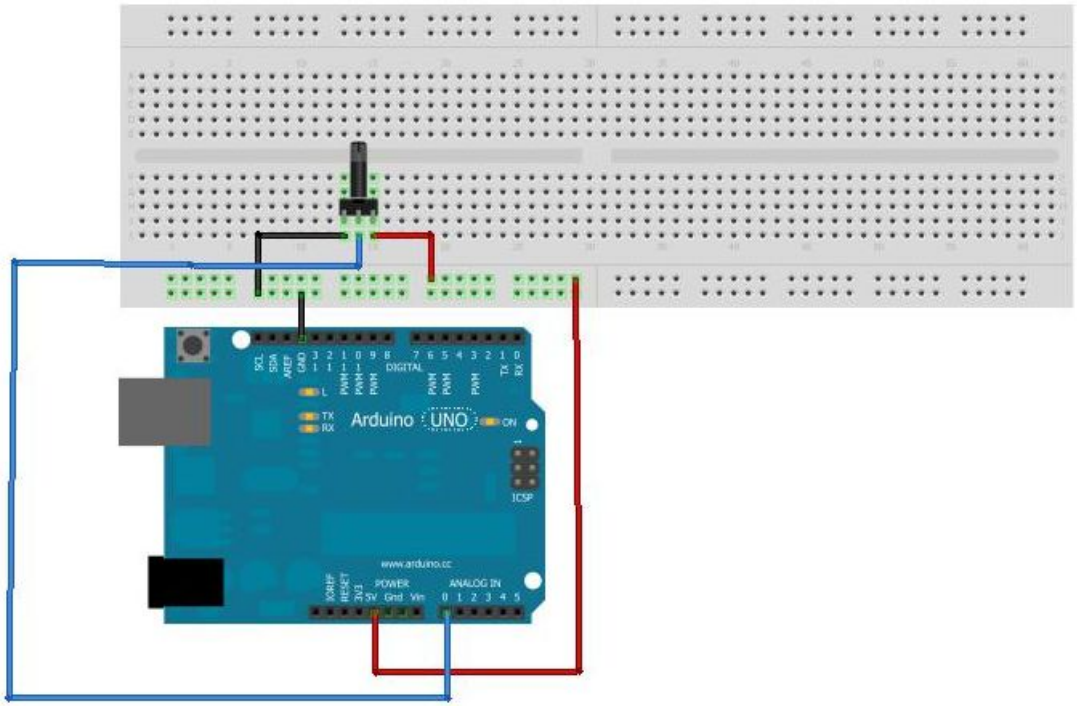


Programando em Arduino

- 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

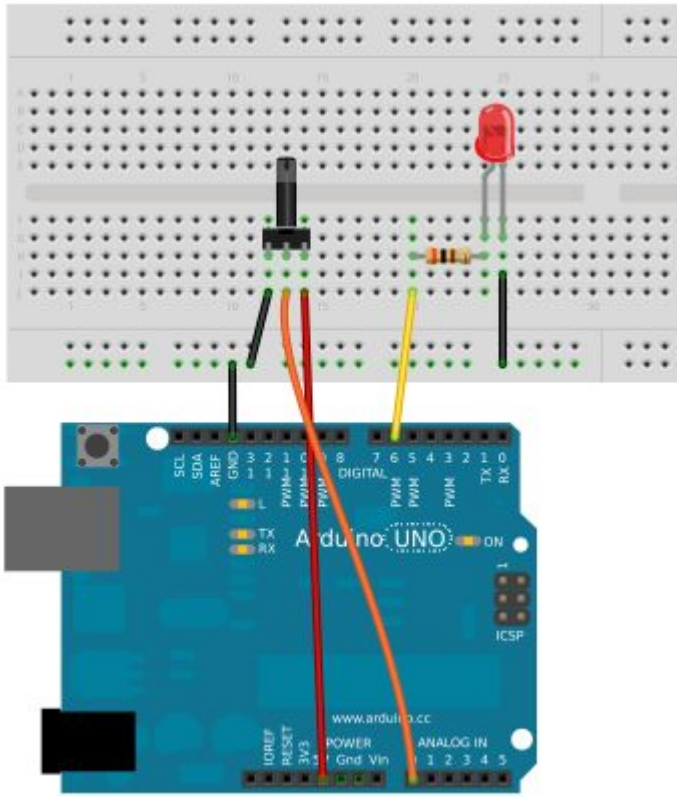
```
1   void setup()
2   {
3     Serial.begin(9600);
4   }
5
6   void loop()
7   {
8     int val = analogRead(0);
9     Serial.println(val);
10  }
11
```

Exercício 3:

Conecte um potenciômetro ao Arduino e crie um código que leia os valores analógicos do potenciômetro e os exibe no Monitor Serial. Além disso, utilize a função Serial.println(val) para visualizar o comportamento gráfico desses valores à medida que o potenciômetro é manipulado.

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

```
1   const int led = 6;
2
3   void setup()
4   {
5     pinMode(led, OUTPUT);
6     Serial.begin(9600);
7   }
8
9   void loop()
10  {
11    int val = analogRead(0);
12    Serial.println(val);
13    digitalWrite(led, HIGH);
14    delay(val);
15    digitalWrite(led, LOW);
16    delay(val);
17  }
```

Exercício 4:

Crie um circuito utilizando um Arduino, um potenciômetro e um LED (de qualquer cor), utilize também a função map que possui o seguinte formato “pwm=map(valorpot, 0, 1023, 0, 255);”, o circuito através de um potenciômetro capaz controle a intensidade do brilho do LED. O circuito deve seguir as seguintes especificações:

Componentes:

- 1 Arduino
- 1 LED (cor à sua escolha)
- 1 potenciômetro
- 1 resistor de 10k ohms
- Protoboard e jumpers

Programando em Arduino

- Lendo Portas Analógicas
  - Exemplo: lendo um sensor de temperatura
    - O 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

- Lendo Portas Analógicas
  - Exemplo: lendo um sensor de temperatura
    - **Equação de Steinhart-Hart**

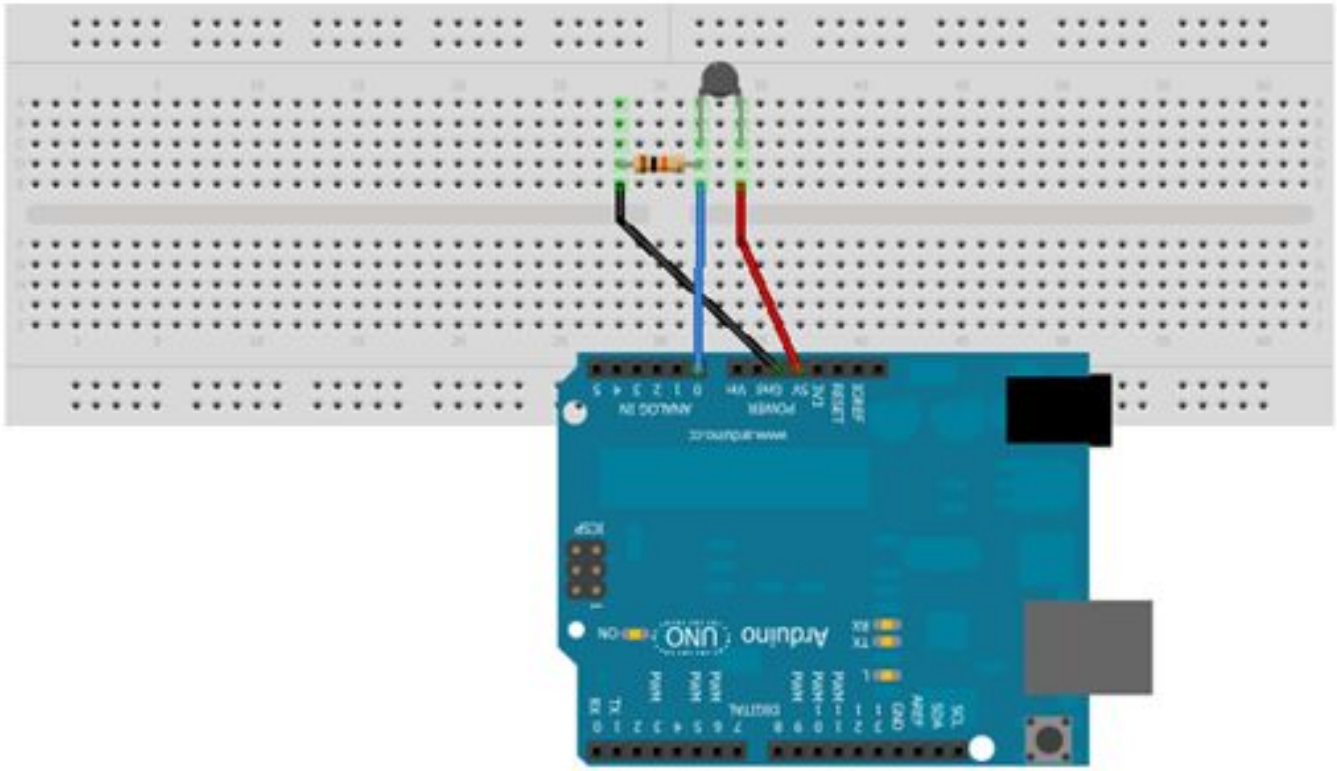
$$1/T = a + b * \ln(R) + c * (\ln(R))^3$$

      - 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

- Lendo Portas Analógicas
  - Exemplo: lendo um sensor de temperatura



Programando em Arduino

- Lendo Portas Analógicas
  - Exemplo: lendo um 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

- Lendo Portas Analógicas
  - Exemplo: lendo um sensor de temperatura

```
24  void setup()
25  {
26    Serial.begin(9600);
27  }
28
29  void loop()
30  {
31    int valor = analogRead(sensor);
32    double c = tempCelsius(valor);
33    Serial.println(valor);
34    Serial.println(c);
35    delay(100);
36  }
```

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

```
1   int vezes = 10;  // Quantidade de vezes que a mensagem será impressa no monitor serial
2   int executado = 0; // Quantidade de mensagens já impressas
3
4   void setup()
5   {
6     Serial.begin(9600);
7   }
8
9   void loop()
10  {
11    for (executado; executado < vezes; executado++) {
12      Serial.println("Testando o comando de repeticao for()");
13    }
14  }
```

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:

```
1   const int botao = 6;
2   const int led   = 10;
3
4   void setup()
5   {
6     pinMode(botao, INPUT);
7     pinMode(led, OUTPUT);
8     digitalWrite(botao, HIGH); // Ativa resistor pull-up
9   }
10
11  void loop()
12  {
13    // Teste do comando while()
14    while (digitalRead(botao));  // Espera até que o botão seja pressionado
15    digitalWrite(led, HIGH);
16    delay(2000);
17    digitalWrite(led, LOW);
18  }
```

Programando em Arduino

- Comandos de Repetição
  - Baseado em uma expressão 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 baseado em uma expressão 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 garatido que pelo menos uma vez o *cmd* será executado.

Programando em Arduino

- Comandos de Repetição
  - Baseado em uma expressão com teste no final
    - Exemplo:

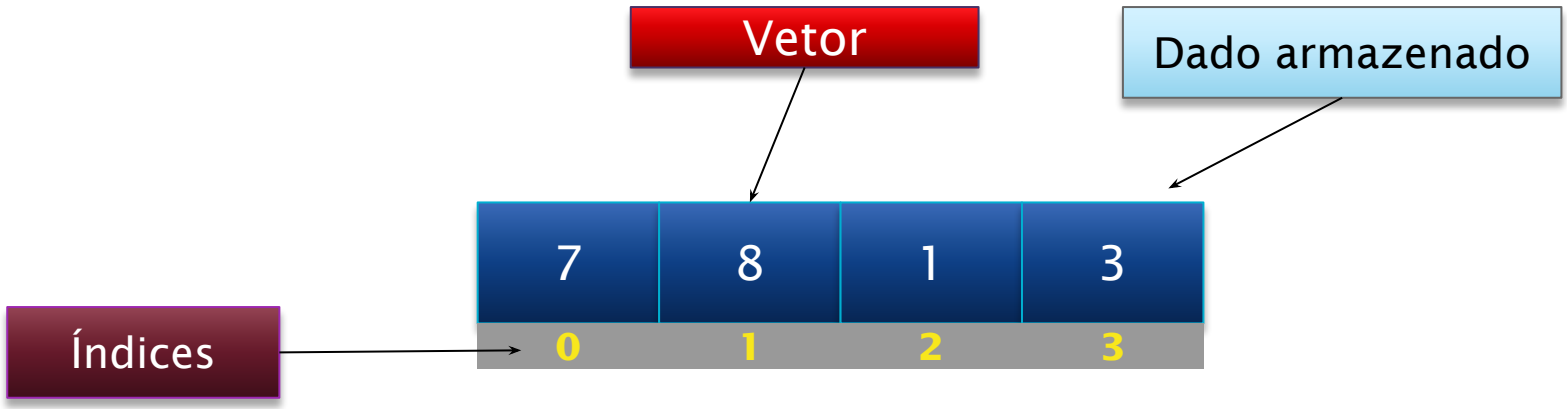
```
1   const int botao = 6;
2   const int led   = 10;
3
4   void setup()
5   {
6     pinMode(botao, INPUT);
7     pinMode(led, OUTPUT);
8     digitalWrite(botao, HIGH); // Ativa resistor pull-up
9   }
10
11  void loop()
12  {
13    // Teste do comando do while()
14    do {
15      digitalWrite(led, HIGH);
16      delay(2000);
17      digitalWrite(led, LOW);
18     delay(2000);
19    } while (digitalRead(botao));  // Enquanto o botão não for pressionado, pisca o led
20  }
21
```

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[o] = 7;
      - Atribui o valor 7 a posição o 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.

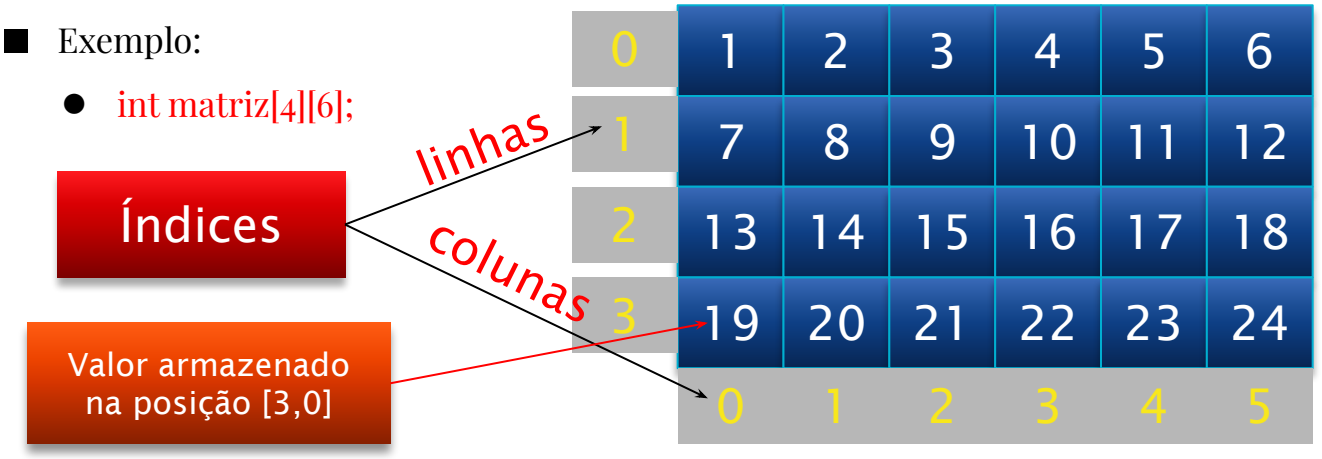
Programando em Arduino

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

```
1   int leds[5] = {2, 3, 4, 5, 6};  // Define as portas onde estão os leds
2
3   void setup()
4   {
5     int i;
6     for (i = 0; i < 5; i++) {
7       pinMode(leds[i], OUTPUT);  // Define as portas como saída
8     }
9   }
10
11  void loop()
12  {
13    int i;
14    for (i = 0; i < 5; i++) {
15      digitalWrite(leds[i], HIGH);  // Acende os leds
16      delay(1000);
17    }
18    for (i = 0; i < 5; i++) {
19      digitalWrite(leds[i], LOW);  // Apaga os leds
20      delay(1000);
21    }
22  }
```

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

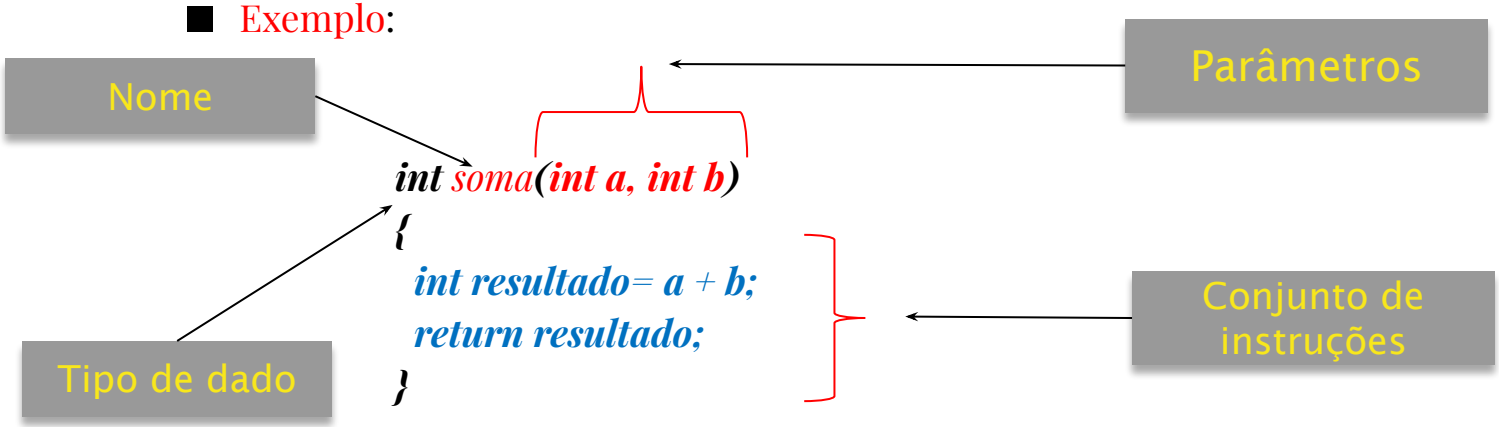
```
1   int matriz_leds[2][2] = {{2, 3}, {4, 5}};
2
3   void setup()
4   {
5     int i, j;
6     for (i = 0; i < 2; i++) {
7       for (j = 0; j < 2; j++) {
8         // Inicializa portas
9         pinMode(matriz_leds[i][j], OUTPUT);
10      }
11    }
12    randomSeed(analogRead(0));  // Define uma semente a partir da porta ananlógica 0
13  }
14
15  void loop()
16  {
17    int linha, coluna;
18
19    linha = random(2); // Gera um número aleatório entre 0 e 1
20    coluna = random(2); // Gera um número aleatório entre 0 e 1
21
22    // Acende led
23    digitalWrite(matriz_leds[linha][coluna], HIGH);
24    delay(500);
25    // Apaga led
26    digitalWrite(matriz_leds[linha][coluna], LOW);
27    delay(500);
28  }
```

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)

```
1  int matriz_leds[2][2] = {{5, 4}, {2, 3}};
2
3  void pisca_diagonal_principal()  // função para controlar os leds da diagonal principal
4  {
5    digitalWrite(matriz_leds[0][0], HIGH);
6    digitalWrite(matriz_leds[1][1], HIGH);
7    delay(1000);
8    digitalWrite(matriz_leds[0][0], LOW);
9    digitalWrite(matriz_leds[1][1], LOW);
10   delay(1000);
11 }
12
13 void pisca_diagonal_secundaria() // função para controlar os leds da diagonal secundária
14 {
15   digitalWrite(matriz_leds[0][1], HIGH);
16   digitalWrite(matriz_leds[1][0], HIGH);
17   delay(1000);
18   digitalWrite(matriz_leds[0][1], LOW);
19   digitalWrite(matriz_leds[1][0], LOW);
20   delay(1000);
21 }
```

Programando em Arduino

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

```
1   void setup()
2   {
3     int i, j;
4     for (i = 0; i < 2; i++) {
5       for (j = 0; j < 2; j++) {
6         // Inicializa portas
7         pinMode(matriz_leds[i][j], OUTPUT);
8       }
9     }
10  }
11
12  void loop()
13  {
14    pisca_diagonal_principal();
15    pisca_diagonal_secundaria();
16  }
```

Programando em Arduino

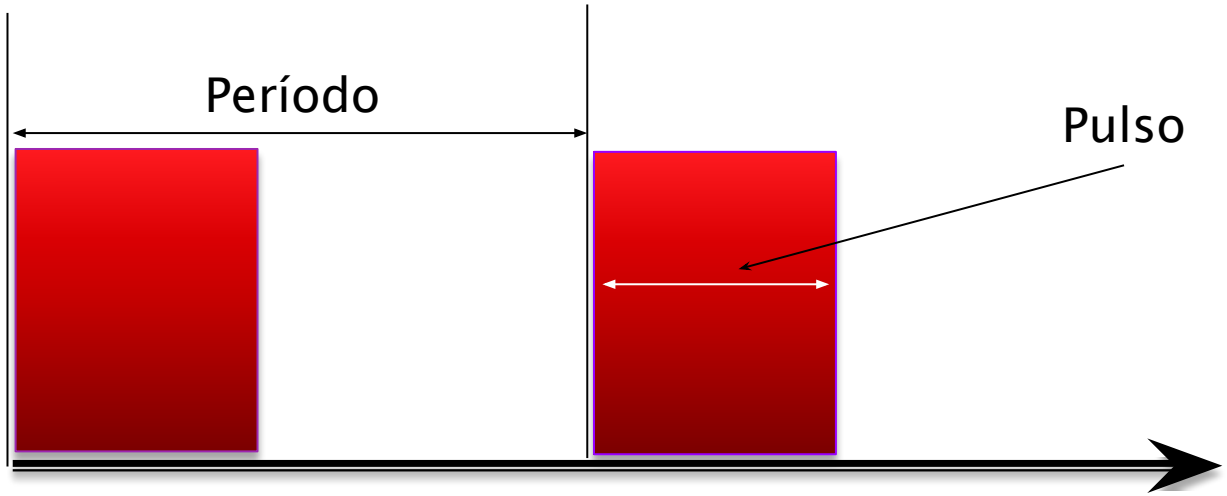
- Sinal PWM – *Pulse Width Modulation* (Modulação por Largura de Pulso)
  - O Arduino UNO possui 6 (seis) portas PWM, 3, 5, 6, 9, 10 e 11.
  - O sinal PWM pode variar de 0 a 255 e para ativá-lo basta usar a seguinte instrução em uma das portas PWM:
    - analogWrite(*pin, sinal_pwm*);
  - Note que as 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-Cicle*
    - 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.

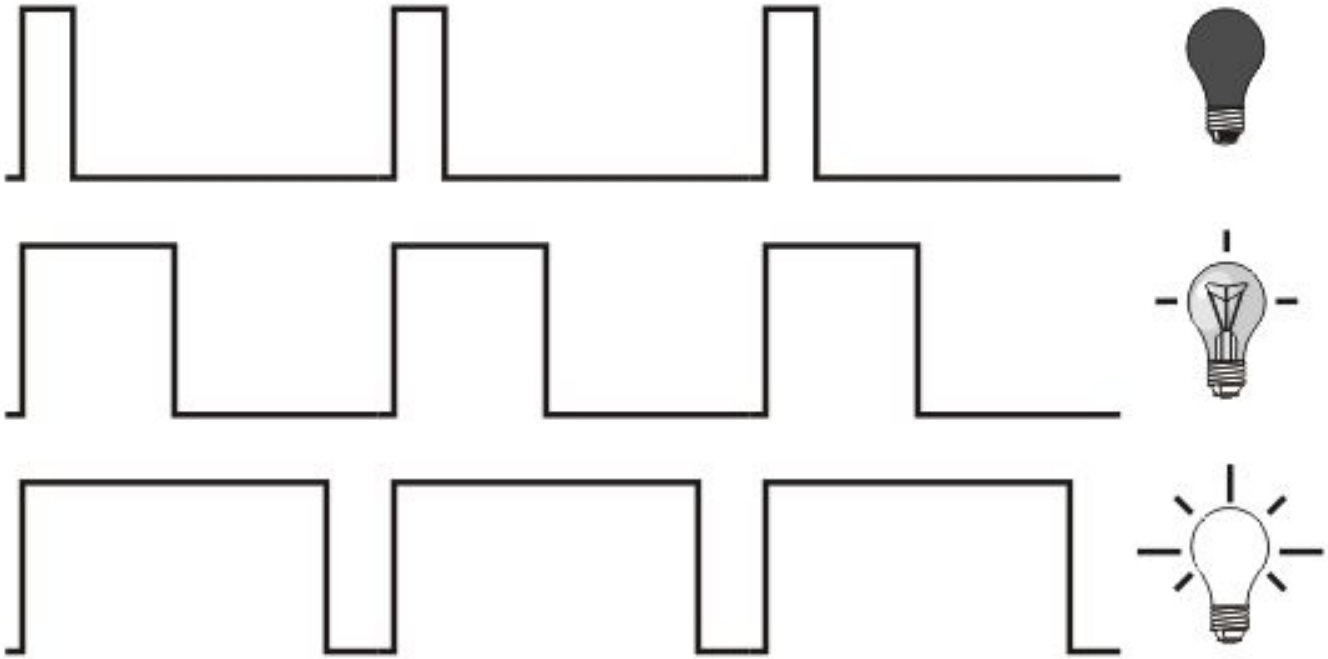
Programando em Arduino

- Sinal PWM – *Pulse Width Modulation* (Modulação por Largura de Pulso)
  - Ciclo de Trabalho – *Duty-Cicle*
    - *Duty cicle = (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

```
1   const int led_alto_brilho = 3;
2
3   void setup()
4   {
5     pinMode(led_alto_brilho, OUTPUT);
6     Serial.begin(9600);
7   }
8
9   void loop()
10  {
11    int i;
12    for (i = 10; i <= 255; i+=10) {
13      analogWrite(led_alto_brilho, i);  // Aumenta a intensidade do brilho
14      Serial.println(i);
15      delay(300);
16    }
17    for (i = 255; i >= 5; i-=10) {
18      analogWrite(led_alto_brilho, i);  // Diminui a intensidade do brilho
19      Serial.println(i);
20      delay(300);
21    }
22    delay(3000);
23  }
24
```

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