



FACULDADE DE TECNOLOGIA DE PORTO ALEGRE– FAQI



Automação com Arduino na prática

Professor



- **Silvio Cesar Viegas**
- **Doutorando no Ensino de Ciências e Matemática**
- **Mestre no Ensino de Ciências e Matemática**
- **Bacharel em Informática**
- **Email: scviegas@gmail.com / silvio.viegas@qi.edu.br**
- **Curriculum Lattes: <http://lattes.cnpq.br/5020505141968701>**





ARDUINO

Massimo Banzi says "You don't need anybody's permission to create something great." So what you waiting for?

Variáveis

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

Variáveis

► **ATENÇÃO!!!**

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

Tipos de Dados

▶ TIPOS DE DADOS DEFINEM:

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

Tipos de Dados

▶ TIPOS DE VARIÁVEIS NO ARDUINO

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

Tipos de Dados

▶ TIPOS DE VARIÁVEIS NO ARDUINO

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

Operadores

- ▶ EM UMA LINGUAGEM DE PROGRAMAÇÃO EXISTEM VÁRIOS **OPERADORES** QUE PERMITEM OPERAÇÕES DO TIPO:

- Aritmética
- Relacional
- Lógica
- Composta

Operadores

▶ OPERADORES ARITMÉTICOS

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

Operadores

▶ OPERADORES RELACIONAIS

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

Operadores

▶ OPERADORES LÓGICOS

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

Operadores

▶ OPERADORES COMPOSTOS

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

Operadores

▶ OPERADOR DE ATRIBUIÇÃO

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

Operadores

► OPERADOR DE ATRIBUIÇÃO

atribuicao

```
int numero = 1; // inicialização

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

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

Monitor Serial

- ▶ O MONITOR SERIAL É UTILIZADO PARA COMUNICAÇÃO

entre o Arduino e o computador (PC).

- ▶ O monitor serial pode ser aberto no menu *tools* opção *serial monitor*, ou pressionando as teclas CTRL+SHIFT+M.
- ▶ As principais funções do monitor serial são: *begin()*, *read()*, *write()*, *print()*, *println()* e *available()*.

Monitor Serial

▶ ALGUMAS FUNÇÕES BASTANTE USADAS:

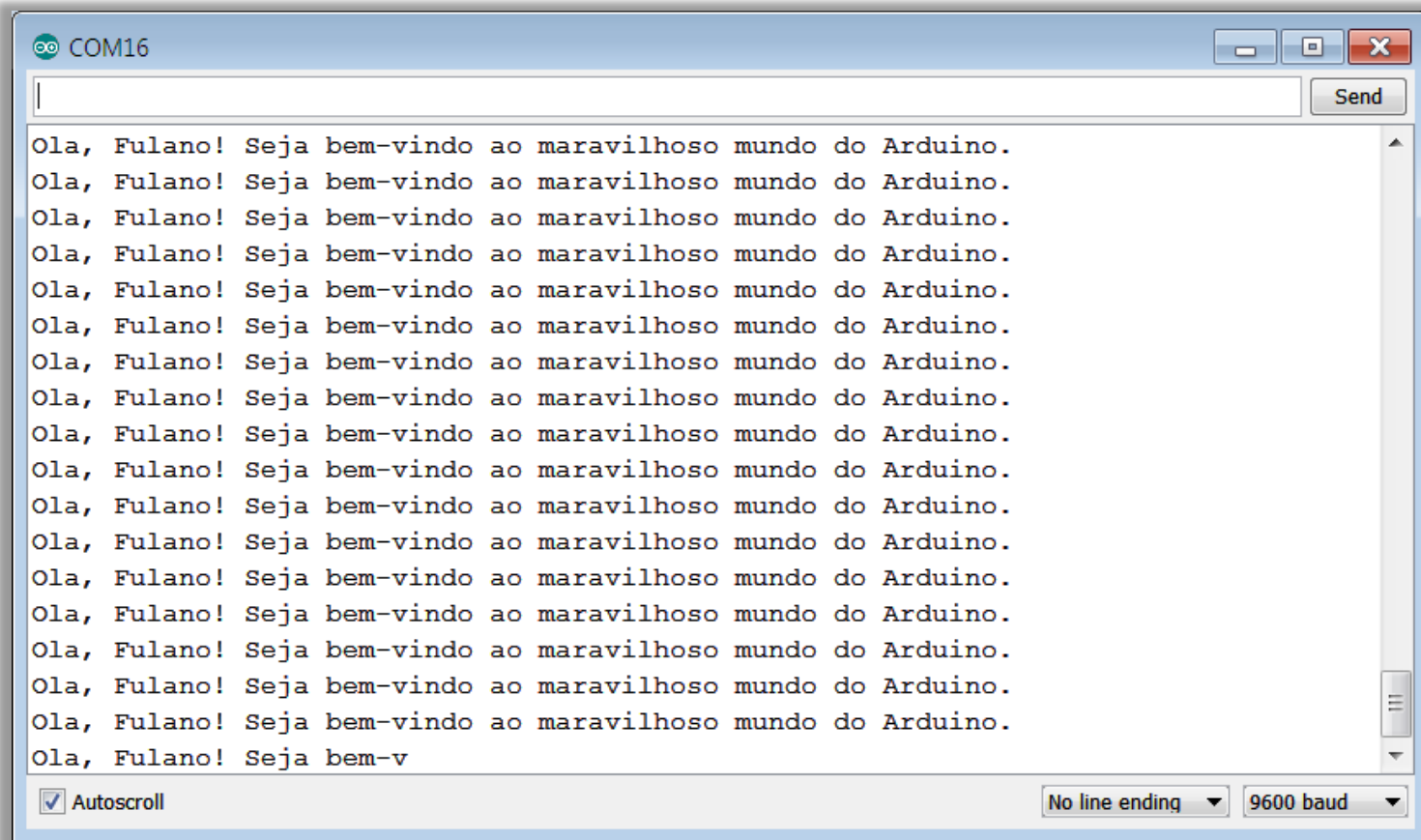
- *begin()*: inicializa a comunicação entre o Arduino e um computador;
- *read()*: recebe caracteres inseridos no monitor serial;
- *print()*: imprime caracteres no monitor serial;
- *println()*: imprime caracteres no monitor serial, mas causa uma quebra de linha no final;
- *available()*: retorna o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

Monitor Serial

```
monitor_serial $  
void setup()  
{  
  Serial.begin(9600);  
}  
  
void loop()  
{  
  Serial.print("Ola, Fulano! Seja bem-vindo ao ");  
  Serial.println("maravilhoso mundo do Arduino.");  
}
```

Monitor Serial

► SAÍDA NO MONITOR SERIAL



Comandos de Repetição

▶ **MUITAS VEZES** É repetir uma determinada instrução ou conjunto de instruções.

- ▶ Os comandos de repetição mantêm em um “laço” uma instrução ou conjunto de instruções enquanto uma condição estiver sendo satisfeita.
- ▶ O Arduino possui 3 comandos de repetição:
 1. `while ()`
 2. `for (; ;)`
 3. `do while ()`

Comandos de Repetição

► WHILE ()

- Este comando de repetição avalia uma expressão no início do laço e, caso esta seja verdadeira, a(s) instrução(ções) dentro do “laço” permanecem sendo executadas.

- Sintaxe do comando while:

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

- onde:

- *expr* – é uma expressão que pode ser lógica, relacional ou aritmética.
- *cmd* – um ou mais comandos a serem repetidos.

-

enquanto a expressão for verdadeira.

Como a avaliação da expressão é realizada no início do laço, pode ser que o *cmd* não execute nenhuma vez.

Comandos de Repetição

► WHILE ()

```
while
char ch;

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

void loop()
{
  while (Serial.available() == 0) {
    Serial.println("Informe um caractere.");
    delay(300);
  }

  ch = Serial.read();
  Serial.print("ch = ");
  Serial.println(ch);
  delay(2000);
}
```

Comandos de Repetição



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

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

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

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

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

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

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

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

Comandos de Repetição



```
while_2
int contador;

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

    contador = 0;
    while (contador < 10) {
        Serial.println("Oficina de Robotica");
        contador++;
    }
}

void loop() {}
```


Comandos de Repetição

► FOR (; ;)

- Este comando

determinada instrução deve ser executada.

- O comando for permite que escrevamos de forma mais organizada os laços de repetição baseados em um contador.
- Sintaxe do comando for:

```
for (inicialização; condição; incremento) {  
    cmd;  
}
```
- onde:
 - *inicialização* – inicialização do contador;
 - *condição* – é uma expressão relacional ou lógica;
 - *incremento* – atualização do contador.

Comandos de Repetição

► FOR (; ;)

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

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

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

void loop()
{
}
```

Comandos de Repetição



NOT

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

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

Comandos de Repetição

▶ DO WHILE ()

- Semelhante ao comando while. A única diferença é que a condição é testada no final do laço.
- Sintaxe do comando do while:
do {
 cmd;
} while (*expr*);

Como a avaliação da expressão é realizada no final do laço, é garantido que o *cmd* será executado pelo menos uma vez.

Comandos de Seleção

- ▶ EM VÁRIOS MOMENTOS EM UM PROGRAMA PRECISAMOS VERIFICAR UMA DETERMINADA CONDIÇÃO AFIM DE SELECIONAR UMA AÇÃO OU AÇÕES QUE SERÃO EXECUTADAS.
- ▶ UM COMANDO DE SELEÇÃO TAMBÉM É CONHECIDO POR DESVIO CONDICIONAL, OU SEJA, DADA UM CONDIÇÃO, UMA PARTE DO PROGRAMA É EXECUTADA.
 - ▶ Os comandos de seleção podem ser do tipo:
 - Seleção simples
 - Seleção composta
 - Seleção de múltipla escolha

Comandos de Seleção

▶ SELEÇÃO SIMPLES

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

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

Comandos de Seleção

- ▶ Seleção simples
 - Acendendo leds pelo monitor serial

```
selecao_simples
#define LED_VERDE    5
#define LED_AMARELO  6
#define LED_VERMELHO 7

char opcao;

void setup()
{
    pinMode(LED_VERDE, OUTPUT);
    pinMode(LED_AMARELO, OUTPUT);
    pinMode(LED_VERMELHO, OUTPUT);
    Serial.begin(9600);
}

void loop()
{
    opcao = Serial.read();

    if (opcao == 'G')
        digitalWrite(LED_VERDE, HIGH);
    if (opcao == 'Y')
        digitalWrite(LED_AMARELO, HIGH);
    if (opcao == 'R')
        digitalWrite(LED_VERMELHO, HIGH);
}
```

Comandos de Seleção

► Seleção simples

- Verificando se há caracteres no buffer de leitura

```
serial_available
char caractere;

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

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


Comandos de Seleção

▶ SELEÇÃO COMPOSTA

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

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

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

Comandos de Seleção



```
selecao_composta_01
char caractere;

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

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

Comandos de Seleção

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

```
selecao_composta_02
#define LED_VERDE 2
#define LED_AMARELO 3
#define LED_VERMELHO 4

char opcao;

void setup()
{
  pinMode(LED_VERDE, OUTPUT);
  pinMode(LED_AMARELO, OUTPUT);
  pinMode(LED_VERMELHO, OUTPUT);
  Serial.begin(9600);
}

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

    if (opcao == 'G')
      digitalWrite(LED_VERDE, HIGH);
    else if (opcao == 'g')
      digitalWrite(LED_VERDE, LOW);
    else if (opcao == 'Y')
      digitalWrite(LED_AMARELO, HIGH);
    else if (opcao == 'y')
      digitalWrite(LED_AMARELO, LOW);
    else if (opcao == 'R')
      digitalWrite(LED_VERMELHO, HIGH);
    else if (opcao == 'r')
      digitalWrite(LED_VERMELHO, LOW);
    else
      Serial.println("Opcao Invalida.");
  }
}
```

Comandos de Seleção

▶ SELEÇÃO DE MÚLTIPLA ESCOLHA

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

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

- onde:

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

Comandos de Seleção

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

```
selecao_multipla_escolha
#define LED_VERDE 2
#define LED_AMARELO 3
#define LED_VERMELHO 4

char opcao;

void setup()
{
    pinMode(LED_VERDE, OUTPUT);
    pinMode(LED_AMARELO, OUTPUT);
    pinMode(LED_VERMELHO, OUTPUT);
    Serial.begin(9600);
}

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

        switch (opcao) {
            case 'G': digitalWrite(LED_VERDE, HIGH);
                       break;
            case 'g': digitalWrite(LED_VERDE, LOW);
                       break;
            case 'Y': digitalWrite(LED_AMARELO, HIGH);
                       break;
            case 'y': digitalWrite(LED_AMARELO, LOW);
                       break;
            case 'R': digitalWrite(LED_VERMELHO, HIGH);
                       break;
            case 'r': digitalWrite(LED_VERMELHO, LOW);
                       break;
            default: Serial.println("Opcao Invalida.");
        }
    }
}
```

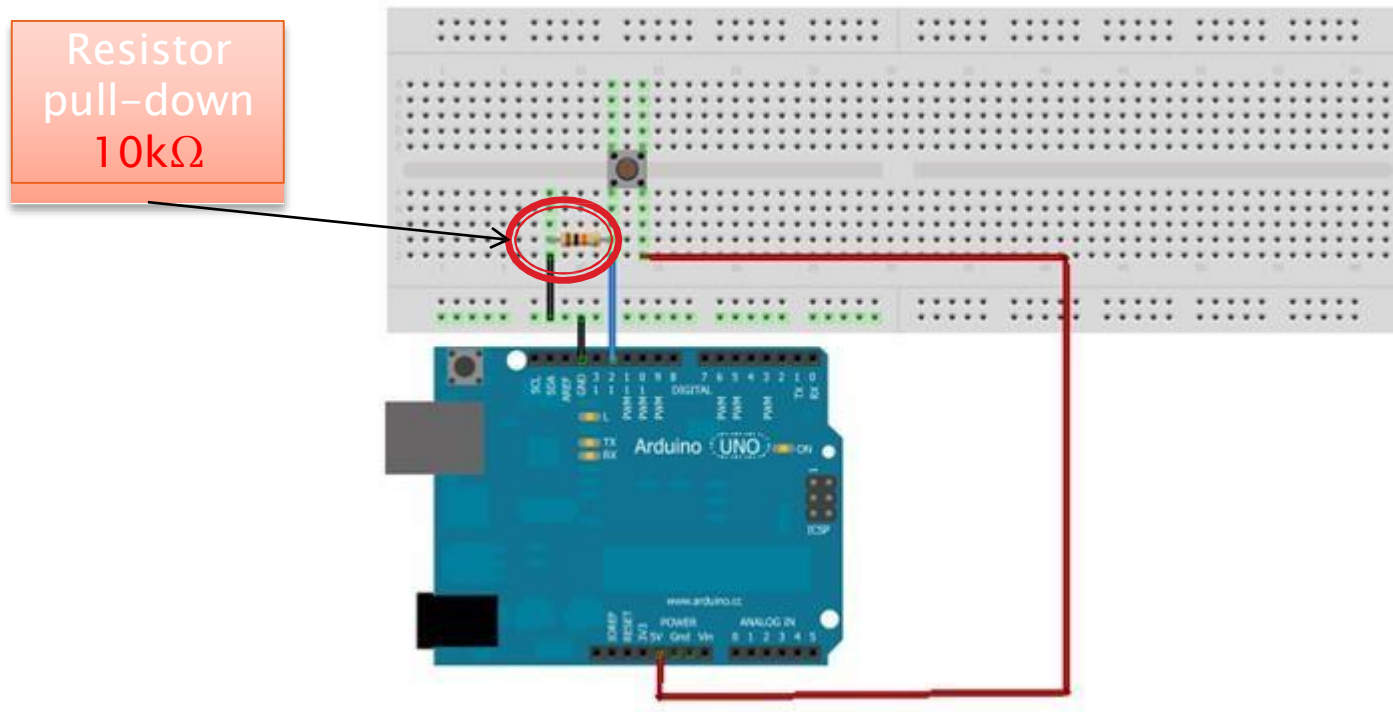
Entrada Digital de Dados

▶ VERIFICANDO O ESTADO DE UM BOTÃO

- Para verificar se um botão está pressionado ou não basta conectá-lo a uma porta digital.
- Para que um circuito com botão funcione adequadamente é necessário o uso de resistores *pull-down* ou *pull-up*.
- Os resistores *pull-down* e *pull-up* garantem que os níveis lógicos estarão próximos às tensões esperadas.

Entrada Digital de Dados

- ▶ LEITURA DE UM BOTÃO COM RESISTOR *PULL-DOWN*
 - Ligação na protoboard



Entrada Digital de Dados

- ▶ Leitura de um botão com resistor *pull-down*
 - Programa

```
pull-down
#define BOTAO 12

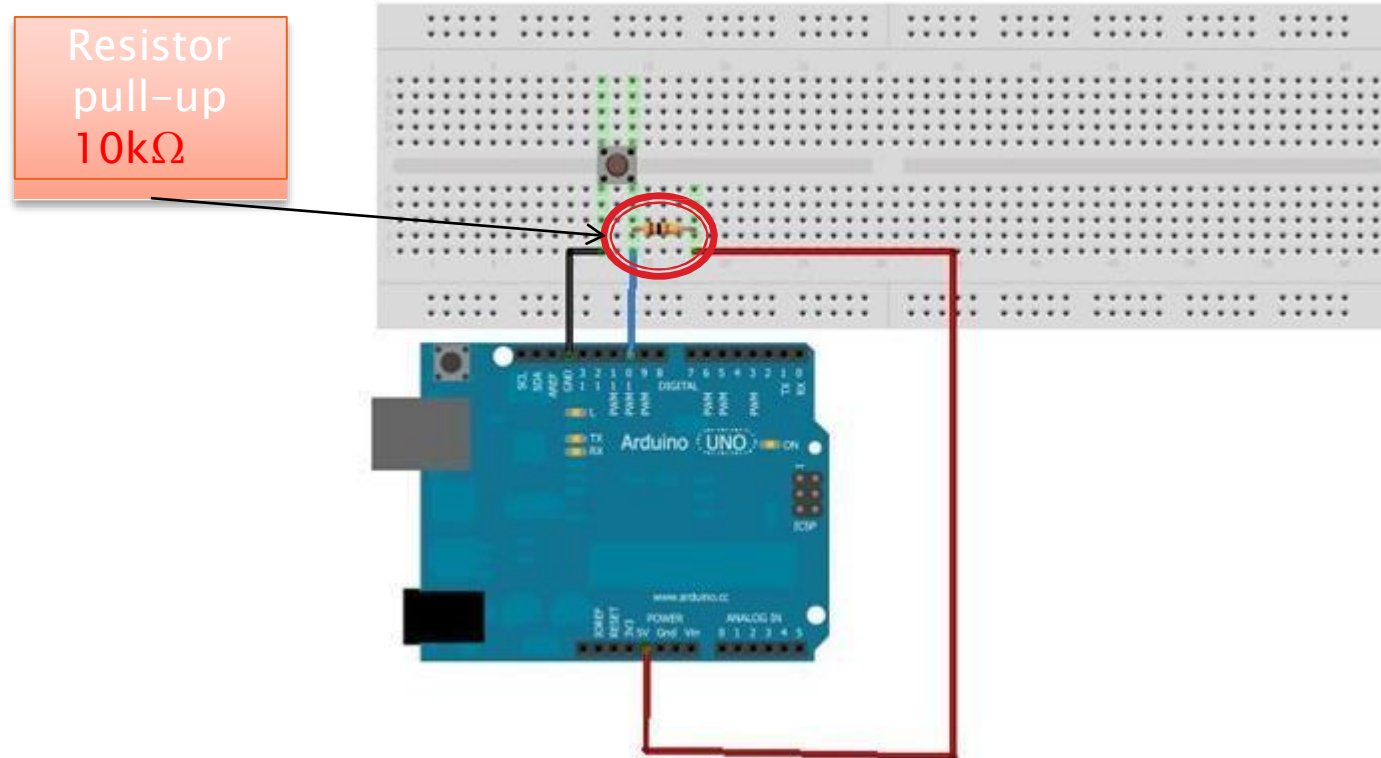
void setup()
{
    pinMode(BOTAO, INPUT);
    Serial.begin(9600);
}

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


Entrada Digital de Dados

► LEITURA DE UM BOTÃO COM RESISTOR

PULL-UP



Entrada Digital de Dados

- ▶ Leitura de um botão com resistor *pull-up*
 - Programa

```
pullup
#define BOTAO 10

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

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

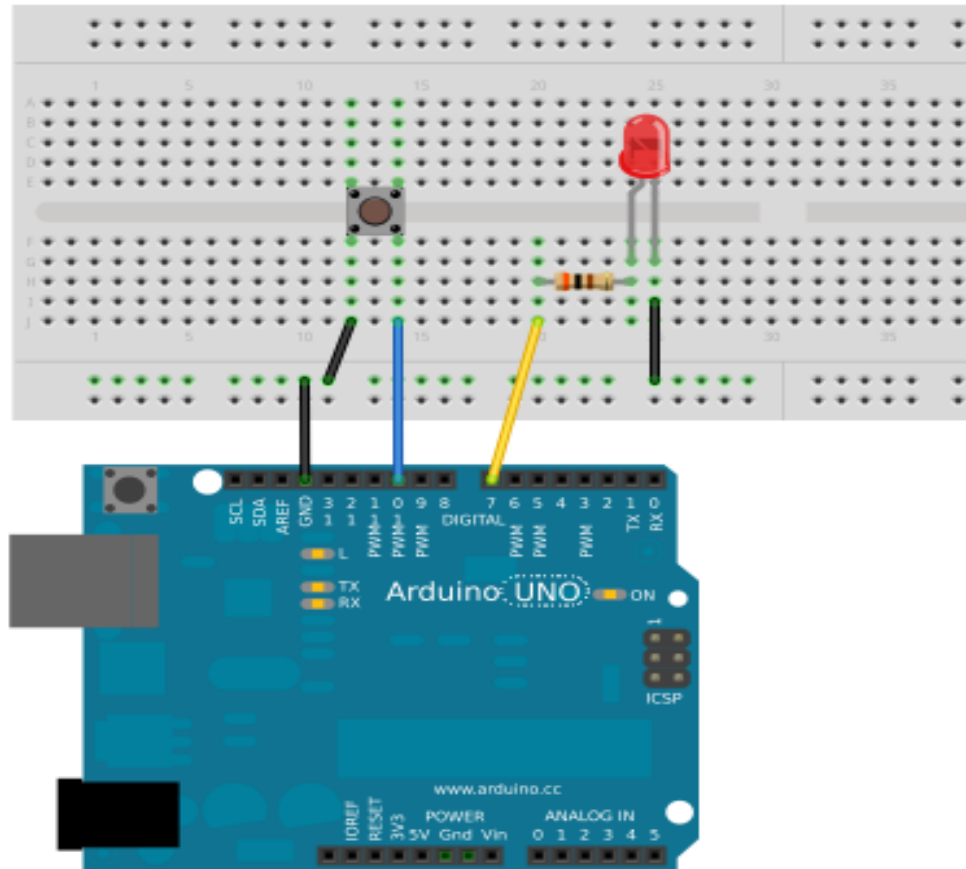
Entrada Digital de Dados

▶ NOTA

- O Arduino possui resistores *pull-up* nas portas digitais.
- Para **ativar** os resistores *pull-up* de uma porta digital **basta defini-la como entrada e colocá-la em nível alto (HIGH)**.
 - `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)`

Entrada Digital de Dados

- ▶ Ativando o *pull-up* de uma porta digital
 - Ligação na protoboard



Entrada Digital de Dados

- ▶ Ativando o *pull-up* de uma porta digital
 - Programa

```
pullup_arduino
#define BOTAO 10
#define LED 7

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

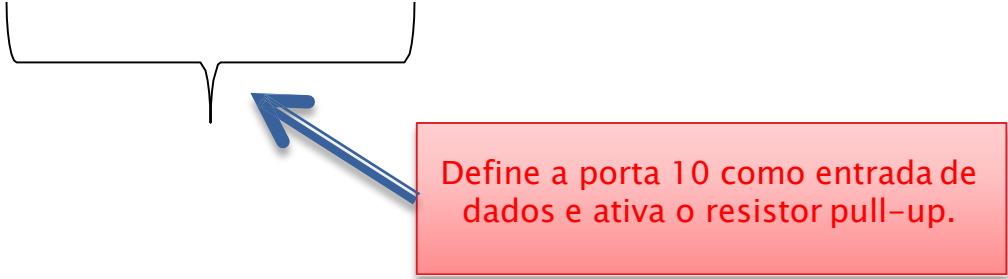
void loop()
{
    if (digitalRead(BOTAO) == LOW)
        digitalWrite(LED, HIGH);
    else
        digitalWrite(LED, LOW);
}
```

Entrada Digital de Dados

▶ NOTA

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

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



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

Exercícios

- ▶ Fazer um contador, com um botão pra incremento e um botão para decremento de uma variável. O valor da variável deve ser mostrado em tela.
- ▶ Criar um controle de LEDs (vermelho, amarelo e verde), onde o botão que for pressionado deve acender o LED correspondente a ele, e mostrar em tela qual LED está acesso.