



# Arduino

Marcos Antonio Jeremias Coelho

[marcos.coelho@satc.edu.br](mailto:marcos.coelho@satc.edu.br)

# 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*.

# Tipos de Arduino

- Existem vários tipos de Arduino com especificidades de hardware. O site oficial do Arduino lista os tipos, alguns como:

- ~ UNO
- ~ Leonardo
- ~ Due
- ~ Esplora
- ~ Mega
- ~ Mega ADK
- ~ Ethernet
- ~ Mini
- ~ LilyPad
- ~ Micro
- ~ Nano
- ~ ProMini
- ~ Pro
- ~ Fio

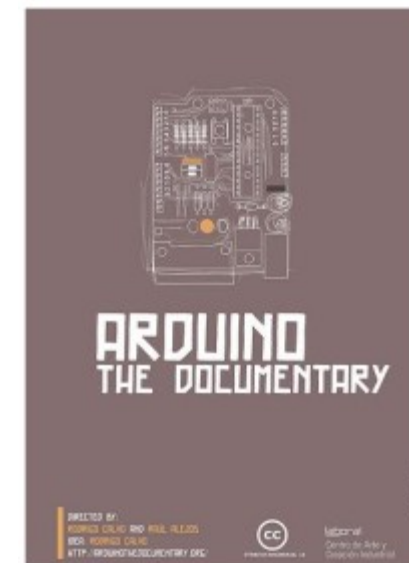
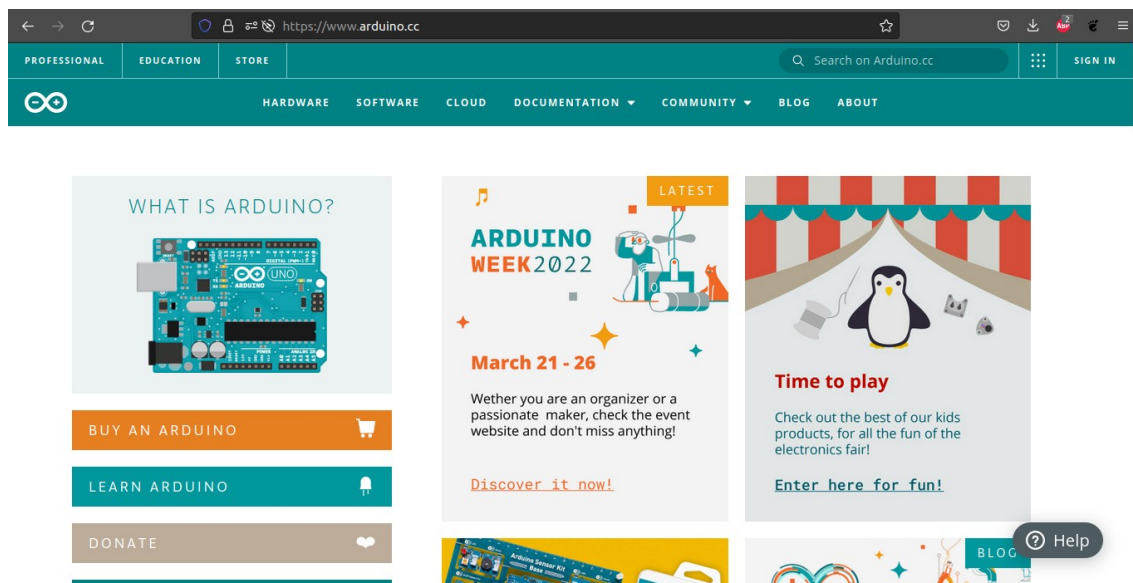


# Referências na WEB:

- O site **oficial** do Arduino é:

<http://arduino.cc>

- Um **documentário** sobre o Arduino pode ser assistido em:  
<http://arduinothedocumentary.org/>



# Microcontroladores

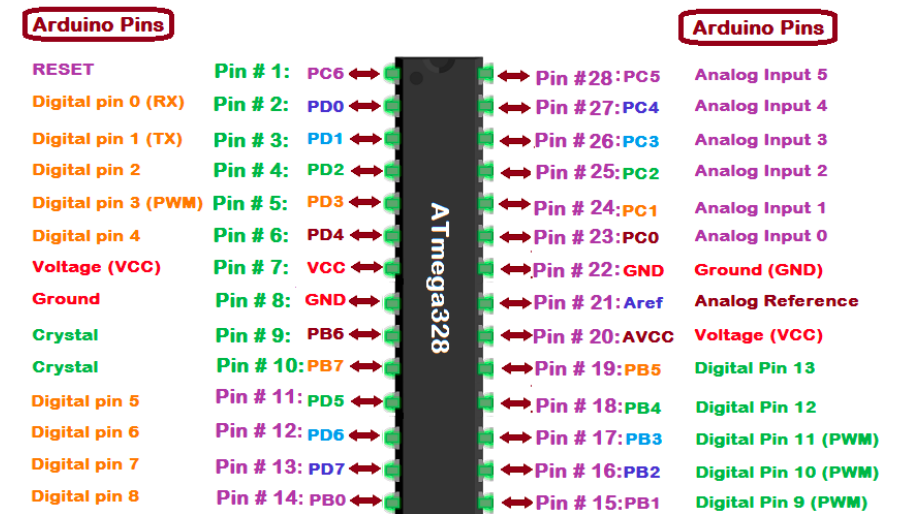
- Um microcontrolador é um CI que incorpora várias funcionalidades.
- Alguns vezes os microcontroladores são chamados de “computador de um único chip”.
- São utilizados em diversas aplicações de sistemas embarcados, tais como:

~ carros,

~ eletrodomésticos,

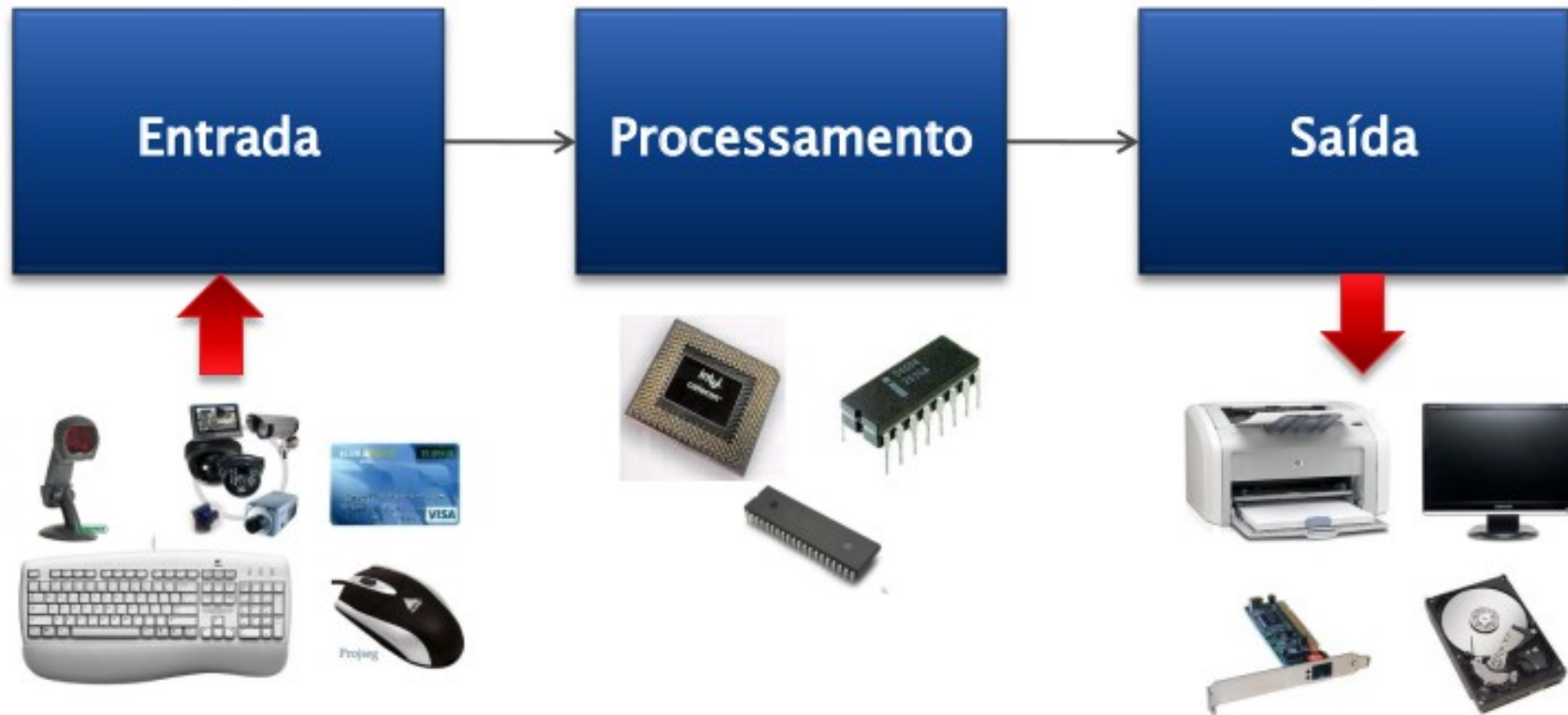
~ aviões,

~ automação residencial, etc.



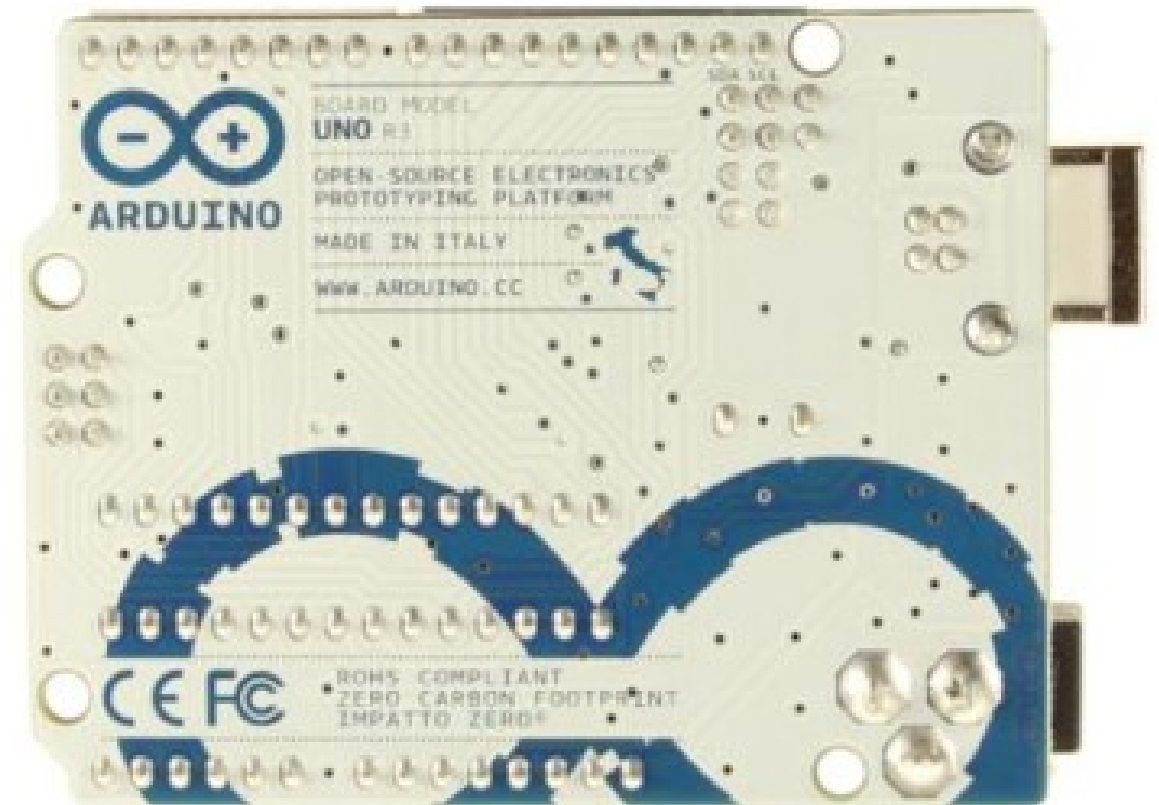
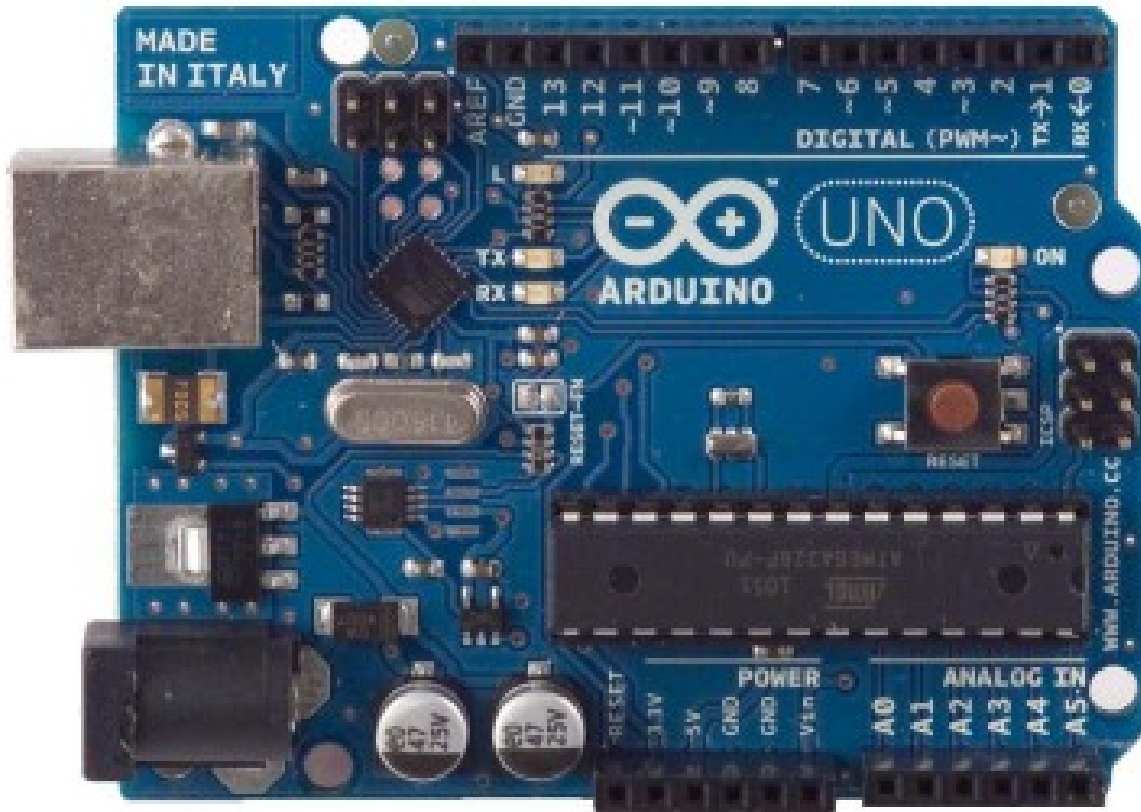
# Microcontroladores

- Processamento de dados



# Arduino Uno

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



# Arduino UNO

## Características

- Microcontrolador: **ATmega328**
- Tensão de operação: **5 V**
- Tensão recomendada (entrada): **7-12 V**
- Limite da tensão de entrada: **6-20 V**
- Pinos digitais: 14 (**seis** pinos com saída **PWM**)
- Entrada analógica: **6** pinos
- Corrente contínua por pino de entrada e saída: 40 mA



# Arduino UNO

## Características

- 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**

# Alimentação

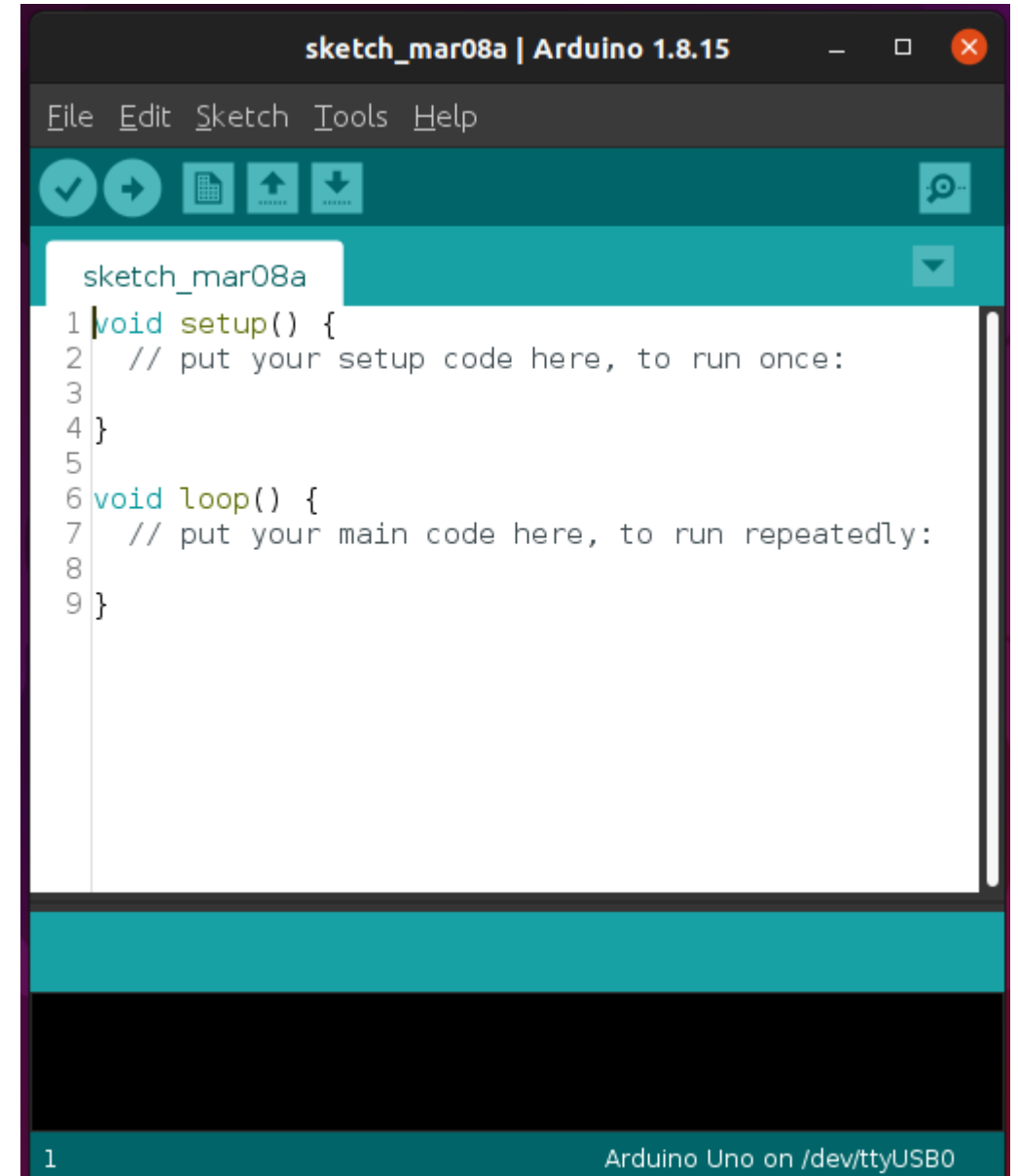
- O Arduino UNO pode ser alimentado pela **porta USB** ou por uma **fonte externa DC**.
- A recomendação é que a fonte externa seja de **7 V a 12 V** e pode ser ligada diretamente no conector de fonte ou nos pinos  $V_{in}$  e Gnd.

# Ambiente de Desenvolvimento

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

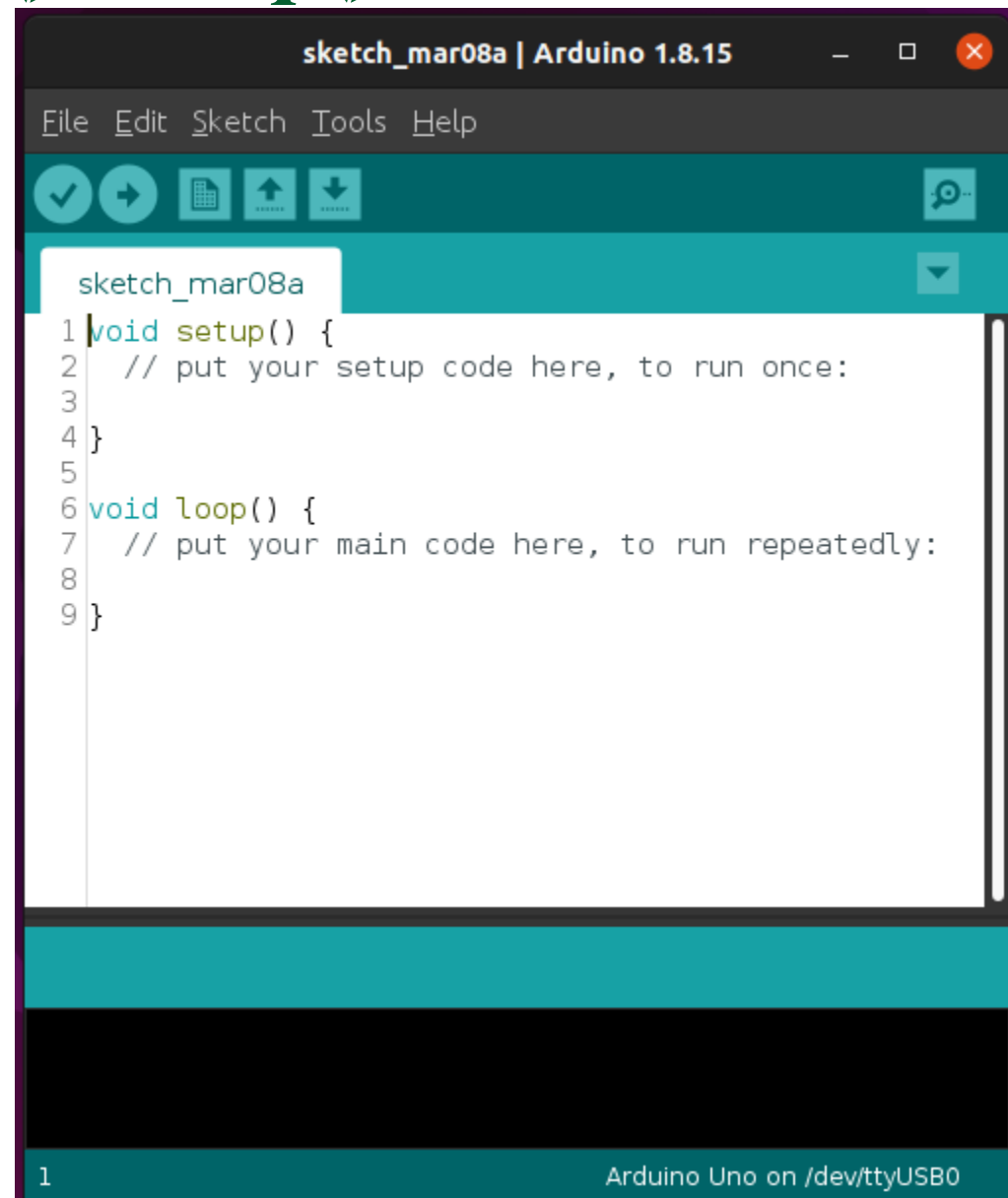


## Funções *setup()* e *loop()*

- 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()*

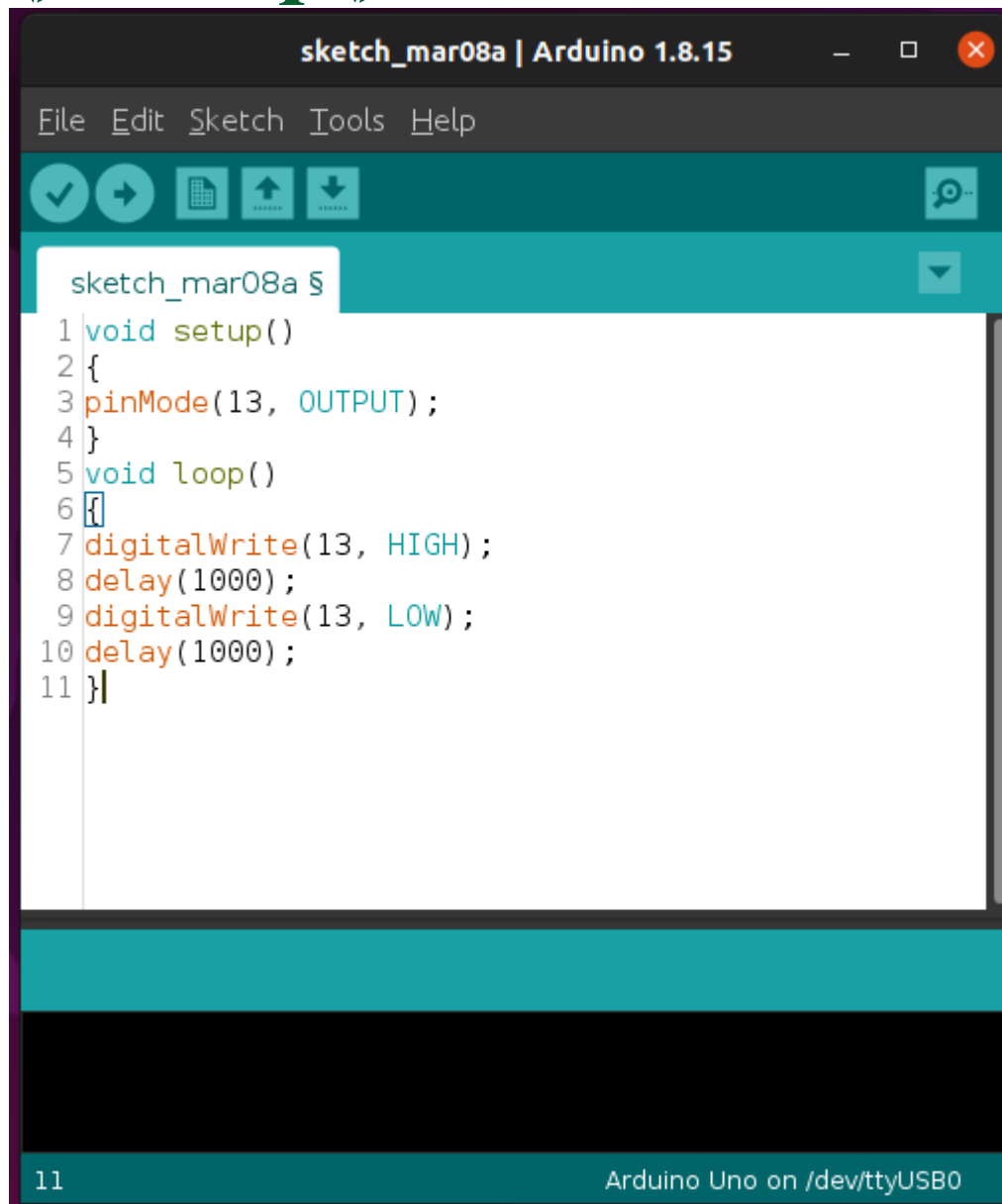
- Exemplo 1: formato das funções *setup()* e *loop()*



```
sketch_mar08a | Arduino 1.8.15
File Edit Sketch Tools Help
[Icons: Checkmark, Run, Upload, Download, Serial Monitor]
sketch_mar08a
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
1
Arduino Uno on /dev/ttyUSB0
```

# Funções *setup()* e *loop()*

- Exemplo 2: formato das funções *setup()* e *loop()*



```
sketch_mar08a | Arduino 1.8.15
File Edit Sketch Tools Help
[Icons: Check, Run, Upload, Download, Search]
sketch_mar08a §
1 void setup()
2 {
3   pinMode(13, OUTPUT);
4 }
5 void loop()
6 {
7   digitalWrite(13, HIGH);
8   delay(1000);
9   digitalWrite(13, LOW);
10  delay(1000);
11 }
11 Arduino Uno on /dev/ttyUSB0
```

# C modificado ou Assembly

blink	Blink	Blink.S
<pre>const int LED_Pin = 8; void setup() {   pinMode(LED_Pin, OUTPUT); }  void loop() {   digitalWrite(LED_Pin, LOW);   delay(1000);   digitalWrite(LED_Pin, HIGH);   delay(1000); }</pre>		<pre>#define __SFR_OFFSET 0x20  #include "avr/io.h"  .global start .global forever  start:   LDI R16, 0x01      ; Setting 1st bit of PORTB as output   STS DDRB, R16   LDI R17, 0x00   STS PORTB, R17     ; Writing 0 to PORTB   LDI R16, 0x00   STS TCCR1A, R16    ; Setting all bits of TCCR1A as 0   RET  forever:   LDI R16, 0xC2   STS TCNT1H, R16    ; Writing 0xC2 into TCNT1H (8-bit)   LDI R16, 0xF7   STS TCNT1L, R16    ; Writing 0xF7 into TCNT1H (8-bit)   LDI R16, 0x05   STS TCCR1B, R16    ; Writing 0x05 into TCCR1B L:LDS R0, TIFR1      ; Load the value of TIFR1 into R0   SBRS R0, 0         ; Skip the next statement if overflow has occurred.   RJMP L             ; Loop until overflow occurs.   LDI R16, 0x00   STS TCCR1B, R16    ; Stop the Timer/Counter1   LDI R16, 0x01   STS TIFR1, R16     ; Clear the overflow flag by writing 1 to it   COM R17            ; Complement R17 register   STS PORTB, R17     ; Toggle the LED output   RET</pre>



# 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 no monitor serial

# 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

- As portas digitais trabalham com valores bem definidos, ou seja, no caso do Arduino esses valores são 0 V e 5 V.
  - ~ 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 Analógicas

- As portas analógicas são utilizadas para entrada de dados.
- Os valores lidos em uma porta analógica variam de 0V a 5V.
- Para ler um 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 0.005 V ou 5 mV.
- Os **valores lidos** em uma porta analógica variam de **0 a 1023** (10 bits), onde 0 representa 0 V e 1023 representa 5 V.

# 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);*

# Algoritmo

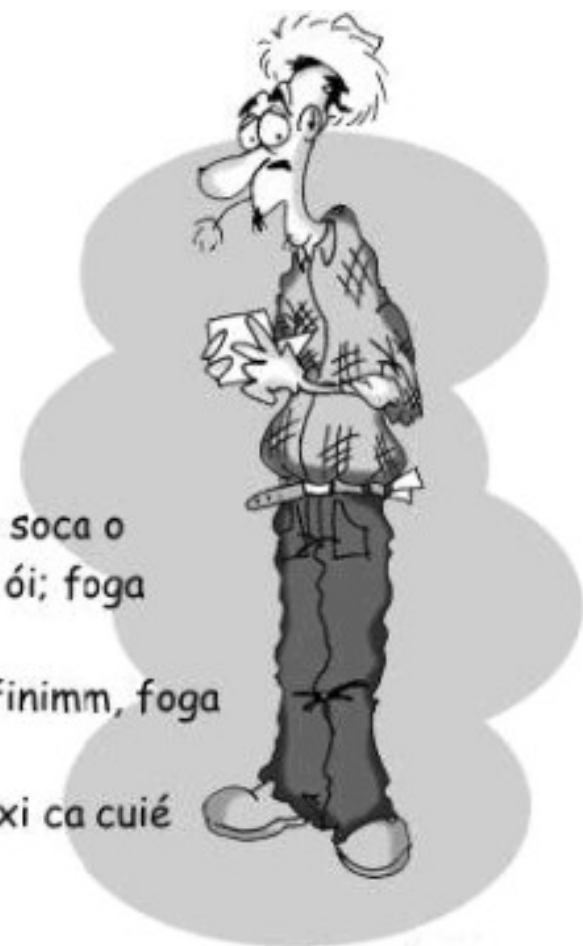
- Sequência de passos que visa atingir um objetivo bem definido.
- Exemplo: Receita caseira

## 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!



# 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



# Constantes

A criação de constantes no Arduino pode ser feita de duas maneiras:

- Usando a palavra reservada `const`

~ Exemplo:

~ *`const int x = 100;`*

- Usando a palavra reservada `define`

~ Exemplo:

~ *`#define X 100`*

# Constantes

No Arduino **existem algumas constantes previamente definidas** e são consideradas palavras reservadas.

- As constantes definidas são:
  - ~ **true** – indica valor lógico verdadeiro
  - ~ **false** – indica valor lógico falso
  - ~ **HIGH** – indica que uma porta está ativada, ou seja, está em 5V.
  - ~ **LOW** – indica que uma porta está desativada, ou seja, está em 0V.
  - ~ **INPUT** – indica que uma porta será de entrada de dados.
  - ~ **OUTPUT** – indica que uma porta será de saída de dados.

# Variáveis

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

# Constantes e Variáveis

- ATENÇÃO!!!

**Um identificador de uma variável ou constante não pode ser formado por caracteres especiais ou palavras reservadas da linguagem**

# Variáveis

## Tipos de Variáveis no Arduino

- ***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.

# Variáveis

## Tipos de Variáveis no Arduino

- ***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 (2 bytes). 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.
- ***long***: Armazena números de até 32 bits (4 bytes). A faixa de valores é de -2.147.483.648 até 2.147.483.647.

# Variáveis

## Tipos de Variáveis no Arduino

- ***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.

# Declaração de Variáveis e Constantes

- Exemplo: declaração de duas constantes e uma variável



# 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**.

# Atribuição de valores a variáveis e constantes

- Exemplo: lendo dados do monitor serial

# Operadores

Em uma linguagem de programação existem vários **operadores** que permitem operações do tipo:

- Aritmética
- Relacional
- Lógica
- Composta

# Operadores aritméticos

+ Adição

- Subtração

\* Multiplicação

/ Divisão

% Módulo (resto da divisão inteira)

# Operadores relacionais

> Maior

< Menor

>= Maior ou igual

<= Menor ou igual

== Igual

!= Diferente

# Operadores lógicos

&& E (and)

|| OU (or)

! Não (not)

# Operadores compostos

++ Incremento

– Decremento

+= Adição com atribuição

-= Subtração com atribuição

\*= Multiplicação com atribuição

/= Divisão com atribuição

# 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 */`



# Comentários

Nota:

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

# Comandos 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.

# Comandos de Seleção Simples

- Exemplo: acendendo *leds* pelo monitor serial

# Comandos 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  
}
```

# Comandos de Seleção composta

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

# Comandos 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;  
}
```

# Comandos de Seleção de múltipla escolha

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

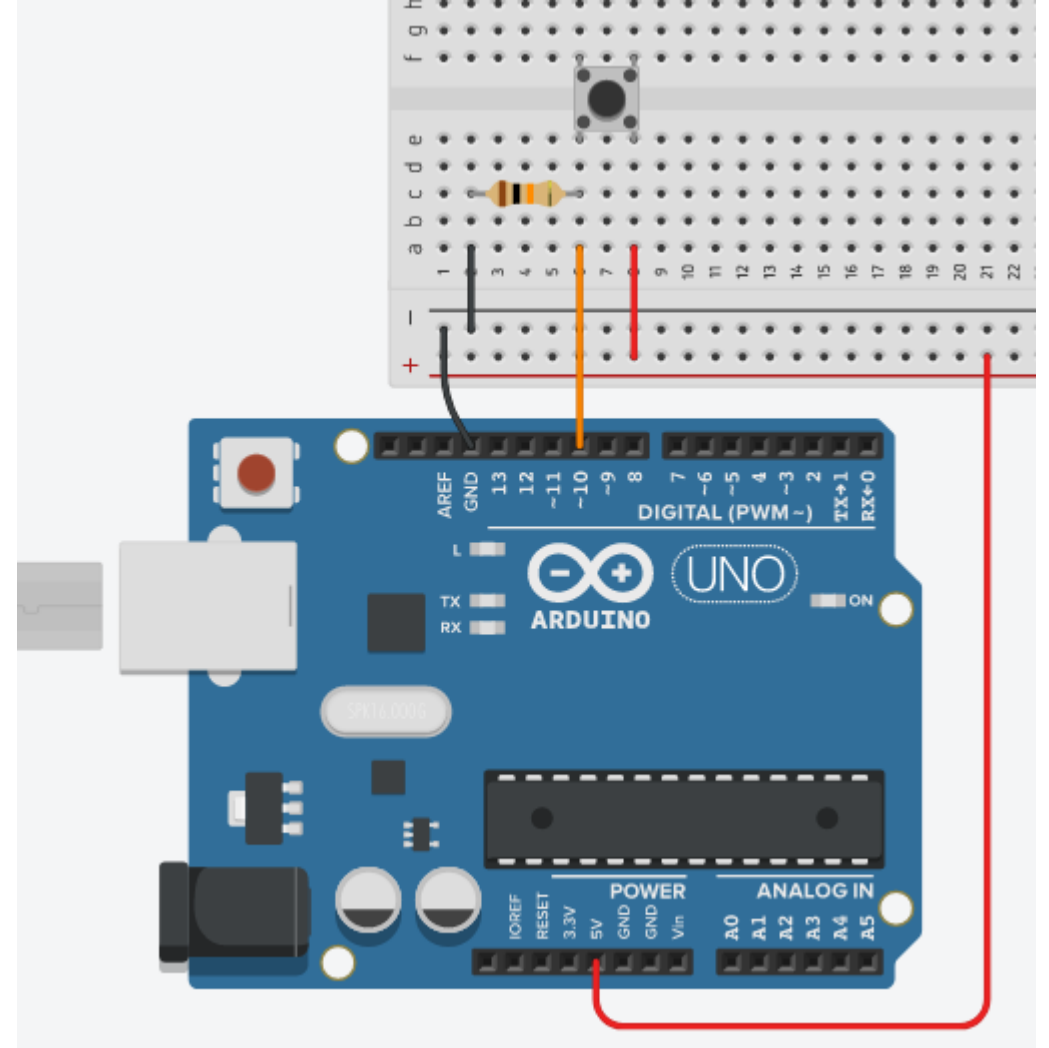


# Lendo um botão

- 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.

# Lendo um botão

- Ligação no protoboard com resistor pull-down

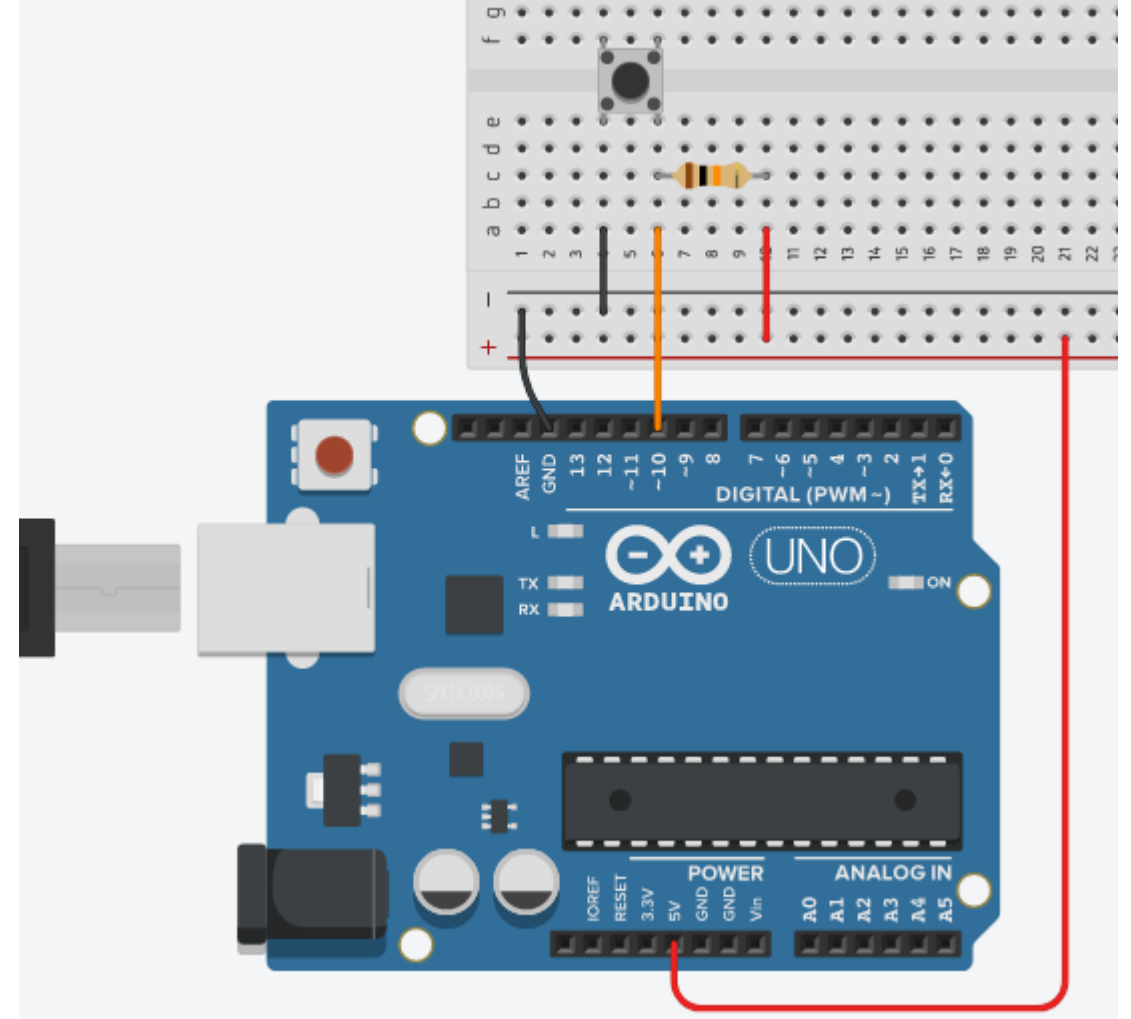


# Lendo um botão

- Exemplo: Com resistor pull-down

# Lendo um botão

- Ligação no protoboard com resistor pull-up



# Lendo um botão

- Exemplo: Com resistor pull-up

## Lendo um botão

- 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)*

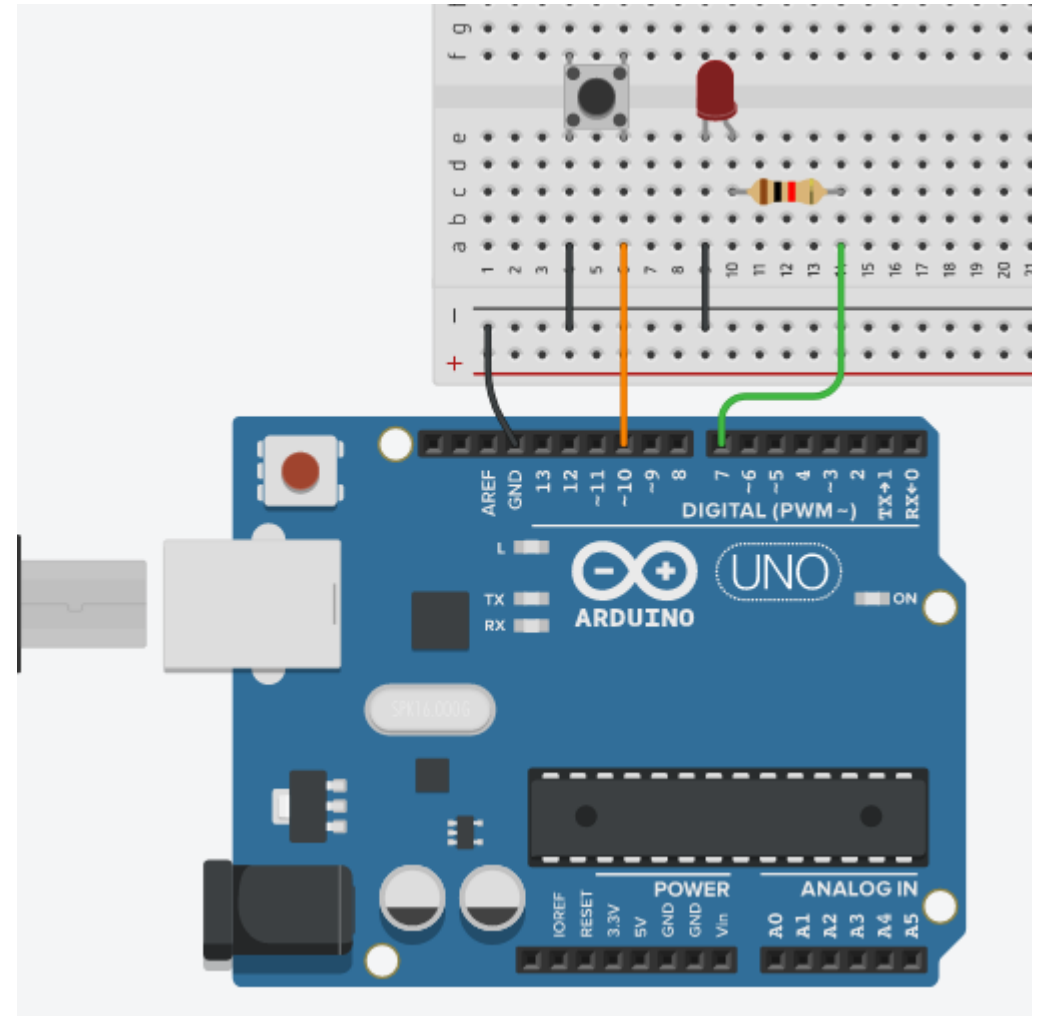
# Lendo um botão

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

# Lendo um botão

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

~ Quanto o botão for  
pressionado o led irá  
apagar





# Lendo um botão

- 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: dados e ativa o resistor pull-up.

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