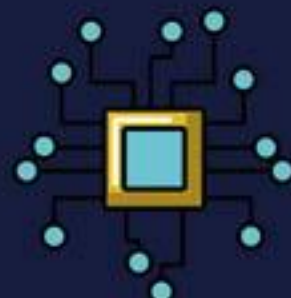




Co-funded by the  
Erasmus+ Programme  
of the European Union



**ROMUAS**  
KILLS  
ROUINO  
SING  
MACHINES  
OUTDATED  
ECOVER

# Recuperar máquinas (dispositivos) desactualizadas utilizando a tecnologia Arduino

## C2

**Expert program toolkit**  
PR1

All information and logos reported in this publication are the authors' and do not reflect necessarily the official position of the project. Names, logos and symbols of the United Kingdom and any other person appear in this report solely for identification purposes and do not imply any endorsement or affiliation with the project or the authors' information or content.



## Sumário

O Básico do Arduino e o Software de Programação .....	4
O que é o Arduino? .....	4
Os benefícios do Arduino .....	4
Arduino – escolha do hardware .....	5
Constituição do Arduino UNO R3 .....	5
Clones vs. Arduino Original .....	6
Instalação do Arduino IDE .....	8
O Básico da Programação do Arduino.....	11
O Básico da Programação do Arduino.....	11
Funções – o que é que significam? .....	12
Arduino UNO – pinagem .....	13
Exercício Prático de Saídas – LED .....	14
Exercício Prático de Entradas – Instrução Condicional (if).....	16
Exemplo – Interruptor de Luz com “Temporizador” .....	18
Exemplo – Semáforo .....	18
Exercício Prático – Função While .....	20
UART e Variáveis .....	24
Como é que o UART funciona? .....	24
Exercício Prático – Comunicação UART.....	26
Exercício Prático – Interação com o programa .....	27
Instrução #define .....	29
Variáveis.....	30
Declaração de Variáveis .....	31
Exercício Prático – Variáveis.....	32
Transmissão Bidirecional do Arduino.....	33
Exercício Prático de Interação com o Sistema – Controlo de LEDs via UART .....	34
O que é o sinal PWM? .....	35
Para que é que o sinal PWM é usado?.....	37
Exercício Prático de PWM – Controlo do brilho de um LED.....	37
Servo Motor .....	38
O que é um servo? .....	39
Como é que o servo funciona?.....	39
Alimentação do servo.....	40
Exercício Prático – Servomecanismo.....	40
Introdução aos Displays .....	41

Como é que o display funciona? .....	42
Ligação do display ao Arduino.....	43
Exercício Prático – Exibição de texto no display .....	45
Remover Conteúdos do Display .....	46
Exercício Prático – Regulação da luz de fundo.....	46
Controlo de Motores DC .....	48
Porquê juntar motores ao Arduino? .....	48
Quais são os motores que vamos utilizar?.....	48
Porque é que não podemos ligar o motor diretamente ao Arduino? .....	49
Porque é que os exercícios práticos não incluem motores?.....	49
Introdução às Pontes H .....	49
A sua primeira ponte H – L293D .....	50
Exercício Prático – Ponte H .....	51
Simulação do motor .....	52
Programação – Controlo da direção de rotação .....	53
Aceleração suave do motor .....	55
Impressão 3D.....	55
O Que É Impressão 3D?.....	56
Como Surgiu A Impressão 3D? .....	56
Para Que Serve A Impressão 3D? .....	57
Como Funciona A Impressão 3D? .....	57
Tipos De Impressora 3D .....	58
Quais Os Principais Materiais Utilizados Na Impressão 3D?.....	59
O que é o Tinkercad?.....	60
Quais são as principais ferramentas do Tinkercad?.....	60
É preciso fazer o download do TinkerCad? .....	61
Como dar os primeiros passos no software? .....	61
Quais são os recursos do software?.....	62
Como começar a modelar com o Tinkercad?.....	63

# O Básico do Arduino e o Software de Programação

## O que é o Arduino?

De forma geral, o Arduino é uma plataforma de prototipagem eletrônica, que integra o popular microcontrolador AVR, e permite o desenvolvimento de controlo de sistemas interativos. É um dispositivo de baixo custo e acessível a todos. Algumas características são:

- Não requer um programador externo;
- Interage, sem problemas, com um compilador dedicado;
- É compatível com inúmeras de placas de expansão (por exemplo: controladores de motor, displays, módulos, sensores, etc.).



No entanto, o verdadeiro poder do Arduino está na sua linguagem de programação baseada em C/C++.

A programação é realizada através de bibliotecas, graças às quais até a criação de um programa complicado está ao alcance de um programador iniciante.

## Os benefícios do Arduino

O projeto começou a ser desenvolvido em 2005, na Itália. Desde então, reuniu uma multidão de seguidores e utilizadores. Desde o início, o Arduino foi projetado para pessoas que tinham poucos conhecimentos de programação de microcontroladores. O seu excelente software, sintaxe amigável e preço reduzido tornaram o Arduino extremamente popular.



A comunidade construída em torno deste projeto é enorme. Isso traz muitos benefícios. Do ponto de vista do iniciante, os mais importantes são os seguintes:

- Um grande número de soluções prontas. Existe uma imensidão de projetos Arduino já criados. Se acha que criou algo “novo” e interessante, é provável que já algum dos utilizadores o tenha feito e publicado na Internet;

- A popularidade da plataforma significa que existem inúmeras variedades de placas e extensões criadas por diversos fabricantes – iremos falar sobre isto de seguida;
- Um grande número de utilizadores facilita a procura de ajuda quando ficar preso em alguma parte do projeto.

## Arduino – escolha do hardware

O Arduino é uma plataforma Open Hardware. Isto significa que todos os materiais necessários para criar o seu próprio dispositivo estão disponíveis. Por esta razão, existem imensas placas compatíveis com Arduino.

Atualmente existem mais de 20 modelos originais disponíveis. Em qualquer boa loja de eletrónica encontrará vários destes modelos. Para o propósito do curso, foi escolhida a placa mais popular – Arduino UNO R3.



## Constituição do Arduino UNO R3

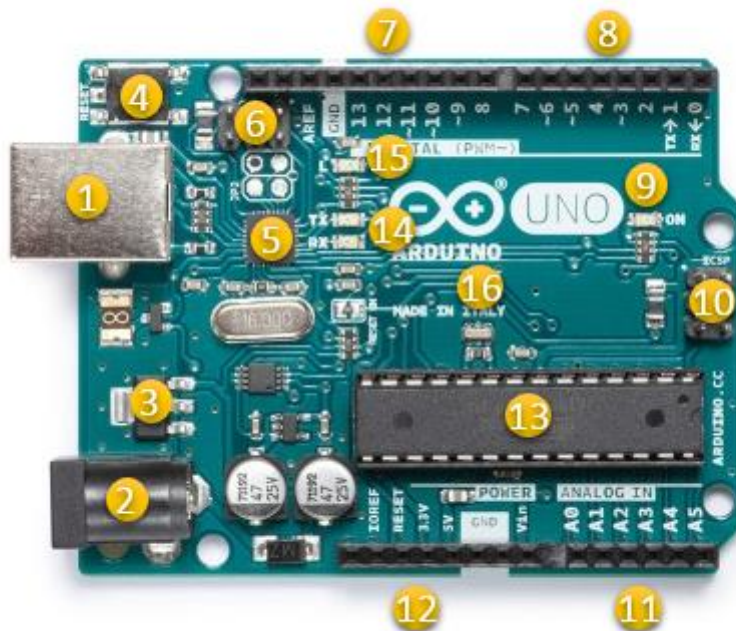
O coração do Arduino é o popular microcontrolador Atmel de 8 bits, o AVR ATmega328 que opera a 16MHz. Trabalhar com tal frequência, de forma muito simples, significa que o microcontrolador é capaz de realizar 16 milhões de operações por segundo!

Os conectores localizados nas laterais da placa, são as saídas dos sinais mais importantes. Aí encontramos 14 entradas/saídas (I/O) digitais programáveis. Seis delas podem ser usadas como saídas PWM (ex.: controlo de motores) e outras 6 como entradas analógicas. Também encontramos um botão de reset e um conector para a alimentação da placa.

O Arduino pode ser alimentado de várias formas. Os métodos mais populares são:

- Alimentação via cabo USB;
- Alimentação através de uma fonte de alimentação externa (baterias ou plug-in).

Os constituintes mais importantes estão sinalizados na imagem seguinte:



1. **Conector USB** – usado para alimentação, programação e comunicação com o computador;
2. **Conector Jack DC** – usado para alimentação (recomendado 7V a 12V);
3. **Estabilizador de Tensão** – transforma a tensão de entrada em 5V;
4. **Botão de Reset** – reinicia a placa Arduino;
5. **Microcontrolador** – responsável pela comunicação com o computador via USB;
6. **Terminais para Programação** para microcontrolador #5;
7. **Pinos Digitais I/O** (8 a 13) e GND Digital (terra);
8. **Pinos Digitais I/O** (3 a 7) e TX e RX (saída de série e entrada de série respetivamente);
9. **LED** – indica se a placa está ligada;
10. **Pinos para Programação Série** – permite programar o microcontrolador (pino #13) via comunicação série.
11. **Pinos de Entrada Analógicos** (0 a 5);
12. **Pinos de Alimentação e Reset** da placa;
13. **Microcontrolador AVR ATmega328** – o coração da placa arduino;
14. **LEDs** – indicam a transmissão do/para o computador;
15. **LED** – ligado ao pino #13, está à disposição do utilizador;
16. **Cristal 16MHz** – faz com que o microcontrolador funcione a uma frequência de 16MHz.

## Clones vs. Arduino Original

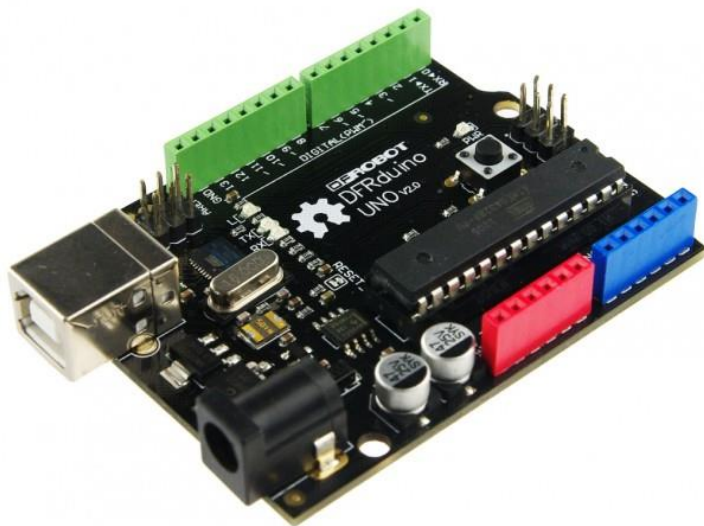
Como foi mencionado, o Arduino é uma plataforma de hardware aberto. Isto significa que todos podem criar o seu próprio Arduino ou projetar uma placa semelhante. Placas idênticas ao Arduino são coloquialmente chamadas de clones.

Os clones podem ser divididos em dois tipos:

- Falsificações integrais que imitam o original;
- Placas compatíveis com Arduino.

Pode correr o risco e adquirir uma falsificação por metade do preço (ou menos). Mas, a verdade, é que são feitas com materiais de menor qualidade, isto pode causar avarias e problemas com mais facilidade. Quanto às placas compatíveis, podem ser tão eficientes como as originais.

Há também outras placas mais baratas, que são compatíveis com Arduino, mas os fabricantes não fingem ser originais. Essas versões são produzidas, por exemplo, pela DFrobot, que vende as suas placas sob o nome de DFRduino.

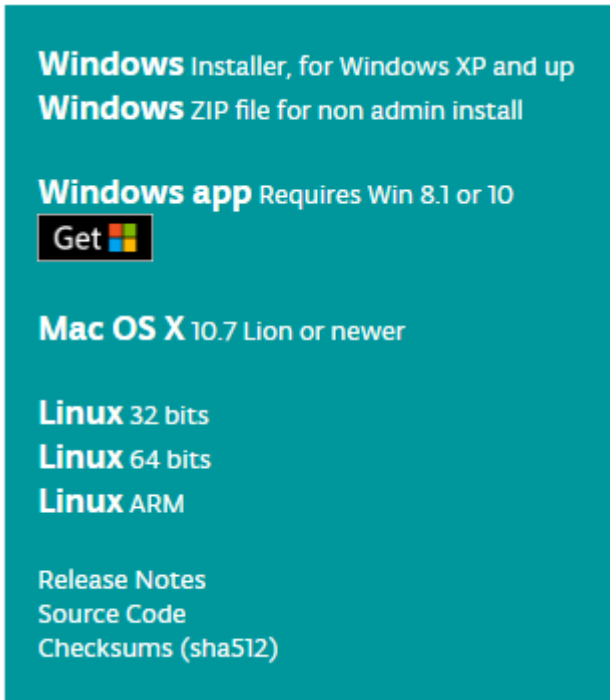


O **DFRduino** é produzido por um fabricante confiável, por isso pode facilmente adquiri-lo. Os produtos compatíveis com Arduino podem ser rapidamente identificados através do sufixo -uino.

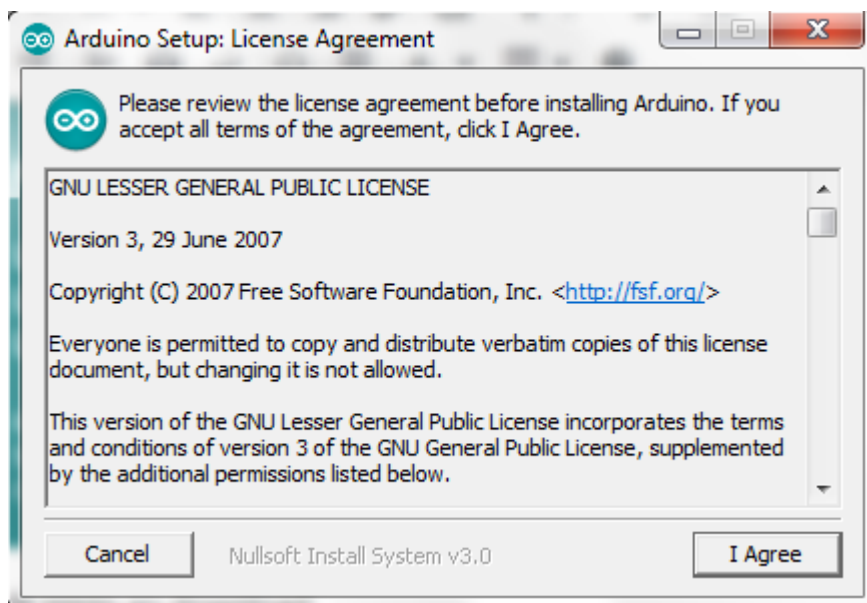
## Instalação do Arduino IDE

Antes de iniciar a programação, deverá ser instalado um software adequado. Pode fazer o download do mais recente Arduino IDE no site oficial Arduino. O software ocupa cerca de 90MB.

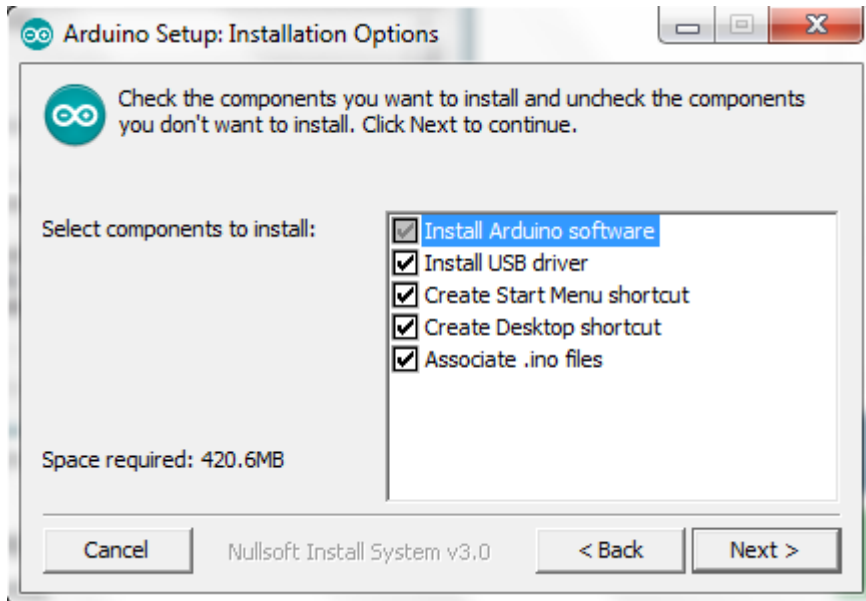
A instalação é básica. Para começar terá de escolher o seu sistema operativo.



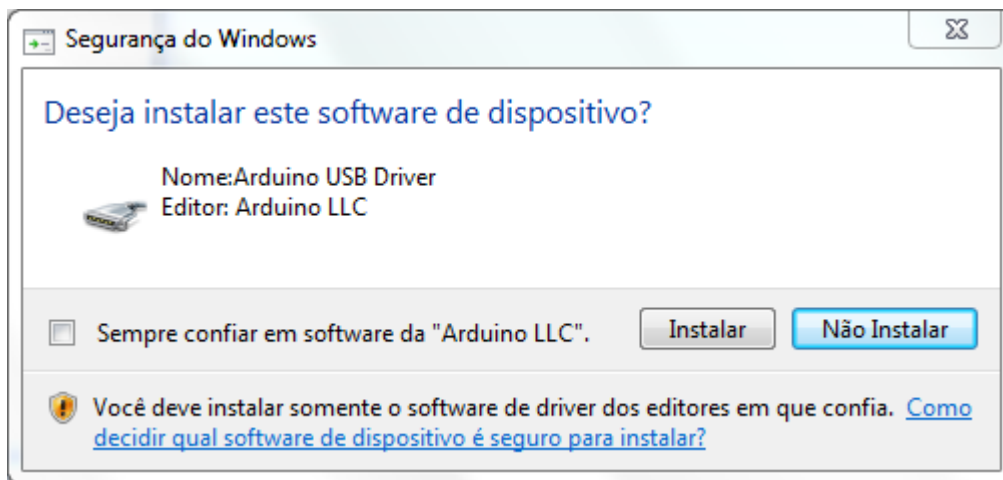
Depois do download estar feito, vamos dar início à instalação. Primeiro terá de aceitar a licença do produto:



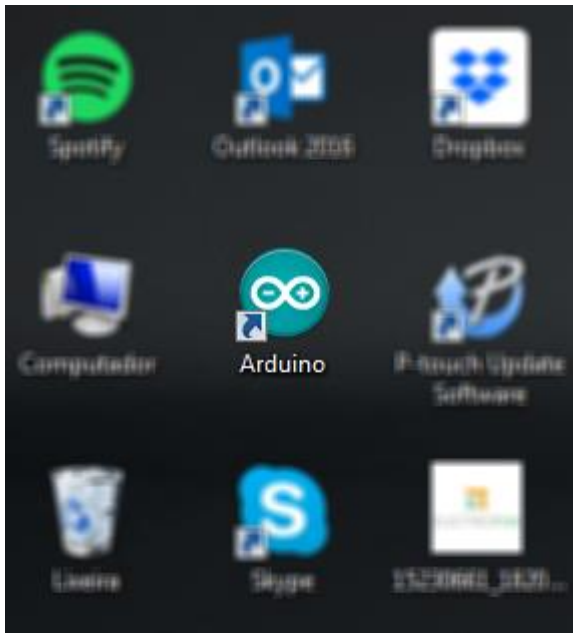
De seguida clique Next. Enquanto isso tenha atenção aos componentes a instalar:



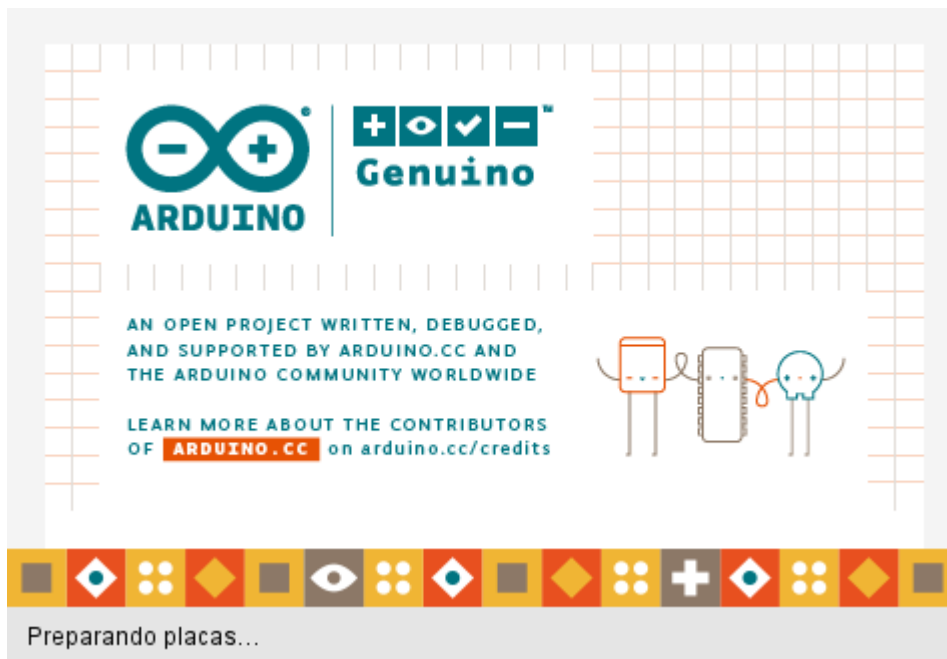
Se escolher instalar o USB driver (recomendável), receberá um aviso no fim da instalação com este aspeto:



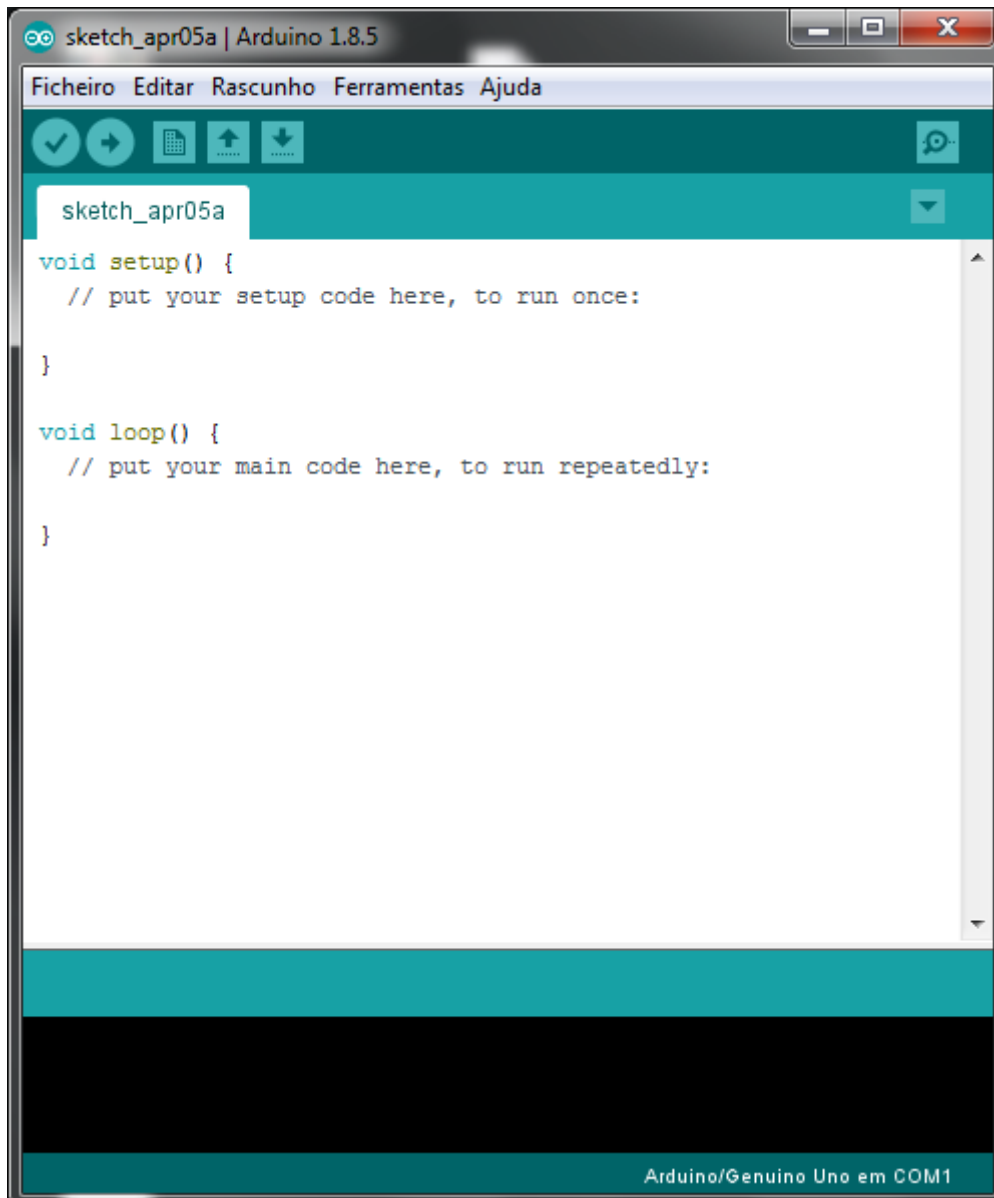
Clique em "Instalar". Não precisa de se preocupar com nada. Seguidamente clique em Close e o software está instalado. Deverá aparecer um ícone igual ao seguinte no seu ambiente de trabalho:



Se o software foi bem instalado, quando abrir o IDE, deverá aparecer uma imagem de boas-vindas deste género:



Depois de uns segundos, o editor:



## O Básico da Programação do Arduino

### O Básico da Programação do Arduino

Na linguagem C, todas as instruções que queremos dar, colocamos na função principal (mais sobre funções abaixo):

```
1 | int main() {  
2 |  
3 | //Conteúdo do Programa  
4 |  
5 | }
```

O símbolo “//” indica um comentário. É uma informação, de uma linha, que ajuda as pessoas a entender o programa. Durante a compilação, todos os comentários são omitidos. Se quiser escrever um comentário mais longo, deve colocá-lo entre “\*/”.

No Arduino, há aspetos que estão simplificados. Existem duas funções: uma delas executa a instrução uma vez, a outra executa a instrução definida em loop. Vejamos:

```
1 | void setup() {  
2 | //Instruções que são executadas apenas uma vez  
3 | }  
4 |  
5 | void loop() {  
6 | //Instruções que são executadas em loop  
7 | }
```

Na prática, a primeira função geralmente contém as configurações. Por exemplo: a definição dos pinos como entradas ou saídas. Nesta função, depois de ligar a placa, serão realizadas ações que é suposto só acontecerem uma vez.

Na segunda função, coloca-se o código que pretende executar o tempo todo (em loop).

## Funções – o que é que significam?

Os códigos podem ser escritos por si, ou pode usar um pré-feito, fornecido por programadores ou geeks que compartilham o seu próprio código.

Existe um conceito de função na linguagem C. Uma função, nas linguagens de programação, é um bloco (lista) de certos comandos extraídos do código principal, cujo desempenho fornece um resultado.

Cada função pode integrar vários argumentos e enviar um resultado. O programador pode determinar quais valores serão o resultado e os dados de entrada. Cada função tem o seu próprio tipo de resultado enviado (prefixos ex.: int, string, etc.) – pode ser um número, um sinal ou outra coisa. Há também um tipo específico de função – não envia qualquer valor (prefixo void).

Vamos concentrar-nos nas principais funções dos programas Arduino.

```
1 | void setup() {  
2 | }
```

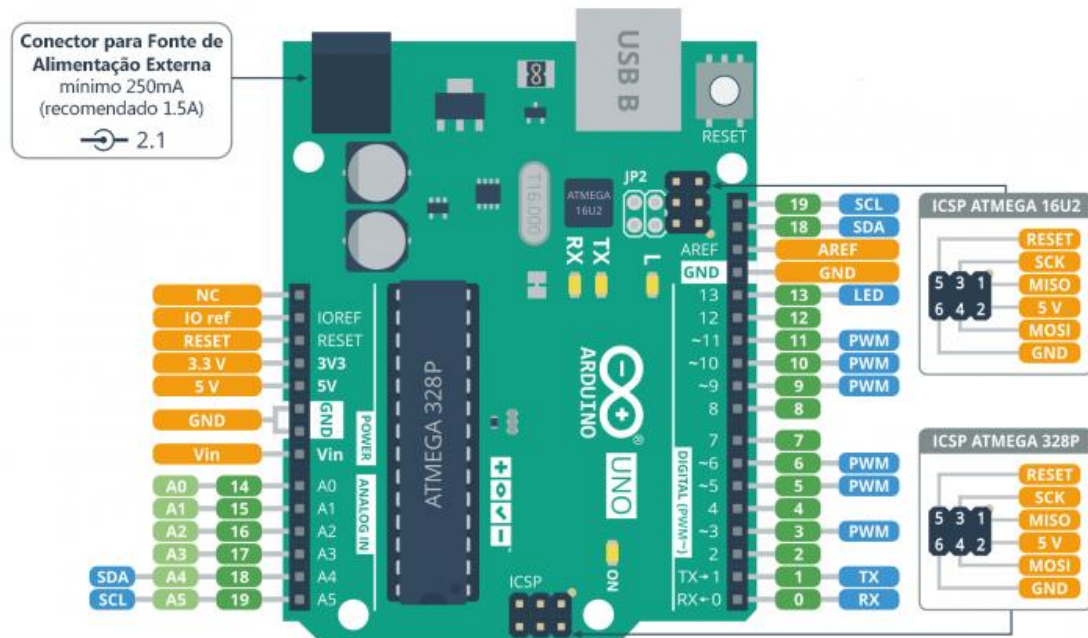
O procedimento setup, como já vimos, é indicado para executar as instruções definidas uma só vez. Como o nome sugere, é destinado principalmente a configurações gerais. Neste, o processador é iniciado, os periféricos são configurados, etc.

```
1 | void loop() {  
2 | }
```

A função (procedimento) loop é, como o nome indica, um loop infinito. É indicado para instruções que devem ser executadas o tempo todo.

## Arduino UNO – pinagem

Através de certos conectores, pode ligar elementos externos ao Arduino, como: LEDs e botões. No entanto, antes de chegarmos a essa parte, precisamos saber a descrição dos pinos/conectores existentes na placa. Abaixo pode ver um esquema com os principais pinos do Arduino UNO:



A **verde escuro** (#0 a #19) estão indicados os pinos digitais de entrada/saída (I/O). Quando usados como saídas, podemos definir como sendo 0V (nível lógico 0, LOW) ou 5V (nível lógico 1, HIGH). Quando são configurados como entradas, são capazes de detetar se o pino em questão possui tensão de 0V ou 5V.

As entradas analógicas (A0-A5) estão destacadas a **verde claro**. Estes são pinos únicos que permitem medir a tensão (0-5V). Como pode ver, a numeração destes pinos coincide com os pinos universais (#14 a #19). Trabalhar no modo analógico é uma função adicional deles.

Em **azul**, foram destacados pinos com funções alternativas. Isto significa que, além de serem pinos I/O normais, podem executar funções mais complexas. Explicação básica:

- **SDA, SCL** – barramentos I2C usados, por exemplo, para comunicação com sensores mais avançados. Existem dois pinos SDA e dois pinos SCL, no canto inferior esquerdo e superior direito da placa;
- **TX, RX** – interface UART, usados principalmente para comunicação com o computador;
- **PWM** – saídas nas quais é possível gerar um sinal retangular (variável). É função muito útil, por exemplo, no controlo de servos;
- **LED** – LED permanentemente instalado no Arduino, que está diretamente ligado ao pino #13.

A **cor-de-laranja** estão saídas que não são programáveis. Estas são responsáveis, principalmente, pela alimentação do sistema.



A programação da inclusão do LED é muito simples. Ligue o Arduino ao computador com o cabo USB. Abra o Arduino IDE e escreva o código abaixo. De seguida, faça o upload para a placa.

```
1 void setup() {
2   pinMode(8, OUTPUT);
3   digitalWrite(8, HIGH);
4 }
5
6 void loop() {
7 }
```

A **função pinMode** (Pin, Mode) permite seleccionar se o pino é uma entrada ou uma saída. O pino é um número inteiro entre 0 e 13 e o modo pode ser:

- INPUT
- OUTPUT
- INPUT\_PULLUP.

Se queremos controlar uma saída, usamos o modo OUTPUT.

Graças a esta configuração, pode definir o estado lógico na saída e, assim, ativar o LED. A **função digitalWrite** (Pin, Status) é usada para este propósito. O estado é um estado lógico que pode ser HIGH ou LOW (alto ou baixo).

Neste exemplo, o LED já foi ligado ao pino terra, então, o Arduino deve atingir um estado alto: digitalWrite (8, HIGH).

Depois de definir o pino num único estado, o seu valor não será alterado até definir um valor diferente. Portanto, o programa acima fará com que o LED permaneça ligado o tempo todo.

### Exercício Prático de Delays – LED a Piscar

Neste exercício, queremos colocar o LED a piscar. Para isso, é necessária uma nova função para inserir o delay. O esquema de ligação é exatamente igual ao anterior. O código deverá ficar assim:

```
1 void setup() {
2   pinMode(8, OUTPUT); //Definir o pino 8 como saída
3 }
4
5 void loop() {
6   digitalWrite(8, HIGH); //Ligar o LED
7   delay(1000); //Esperar 1 segundo
8   digitalWrite(8, LOW); //Desligar o LED
9   delay(1000); //Esperar 1 segundo
10 }
```

Na função loop, o estado é constantemente alternado. Foram adicionados atrasos ao programa através da função delay. Esta função assume um determinado número de milissegundos a atrasar.

Se não introduzisse os atrasos, o sistema mudaria de estado tão rapidamente que seria impossível ver a alternância a olho nu. Pode realizar esta experiência ao colocar 0ms como delay.

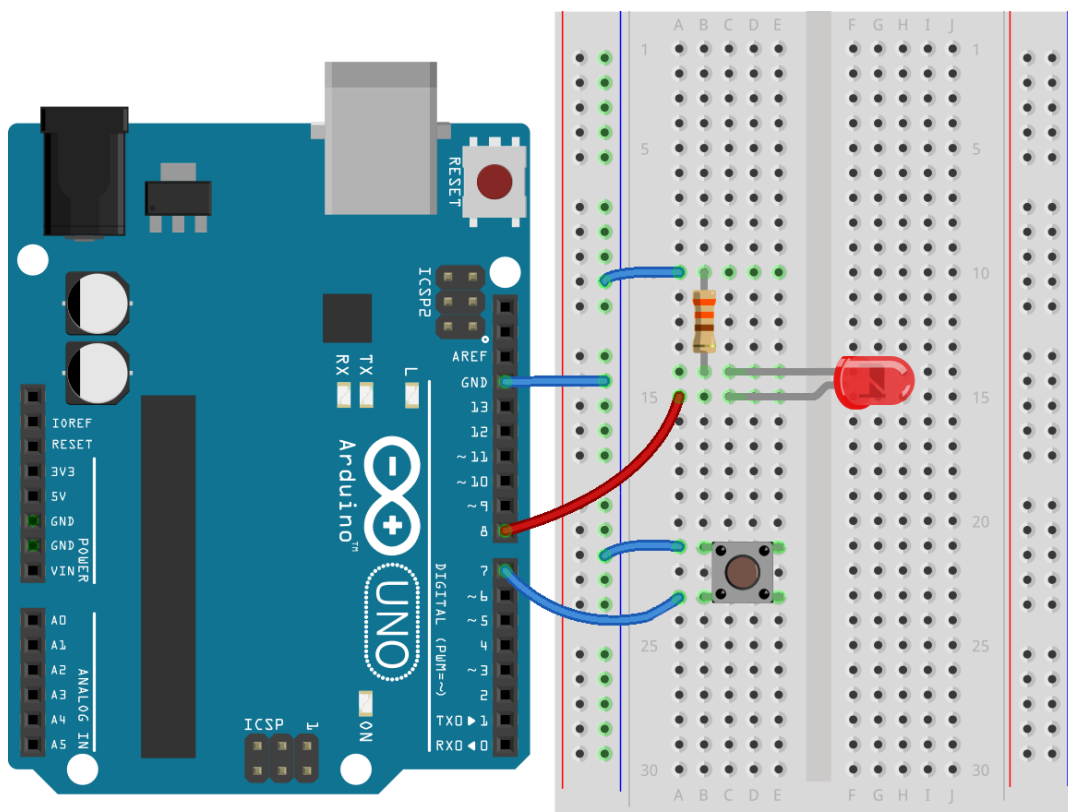
## Exercício Prático de Entradas – Instrução Condicional (if)

Muitas vezes é pretendido que o sistema reaja a sinais externos. Desta vez vamos ligar o botão de pressão ao Arduino, para além do LED.

### Material necessário:

- 1x Arduino UNO e cabo USB;
- 1x Breadboard;
- 1x LED;
- 1x Botão de Pressão;
- 1x Resistência de 330Ω;
- 5x Cabos jumper.

A ligação deverá ser realizada de acordo com o esquema abaixo. De um lado, o botão foi ligado ao terra (menos) e do outro lado ao pino 7.



O objetivo é criar um programa que ligue o LED quando o botão é pressionado. A tarefa é muito simples, mas vamos inserir algo novo – **instruções condicionais**.

Queremos que o programa esteja permanentemente num dos dois estados – o LED está ligado ou desligado. De início, é necessário ler o estado lógico do pino de entrada do botão.

Lembre-se **do modo INPUT\_PULLUP** mencionado anteriormente. A primeira parte do nome (input) obviamente significa entrada, enquanto o segundo (pullup) sugere a inclusão de uma resistência interna que verifica o estado do interruptor. Vamos usá-lo sempre que ligarmos um botão ao Arduino.

Para isso, é necessária a função **digitalRead** (pin), que envia a informação HIGH ou LOW, dependendo do estado. No entanto, apenas ler o estado da entrada não é o suficiente, devemos fazer com que o programa trabalhe em função dessa informação. Daí a instrução condicional (if). Graças a esta, pode executar uma determinada parte do código se houver o cumprimento de determinados requerimentos ou se se verificarem certas condições. Por exemplo, se pressionar um botão.

```
1 | [...]
2 | void loop() {
3 |
4 |   if (condição) {
5 |     /* A instrução é executada em loop somente
6 |     quando a condição é cumprida*/
7 |   }
8 |
9 | }
```

Esta função pode ser facilmente ampliada com uma parte do código que será executada somente se a condição não for verificada. A instrução else é usada para isso.

```
1 | [...]
2 | void loop()
3 | {
4 |
5 |   if ( condição) {
6 |     /* A instrução é executada em loop somente
7 |     quando a condição é cumprida */
8 |   } else {
9 |     /* A instrução é executada somente
10 |    quando a condição não é cumprida*/
11 |   }
12 |
13 | }
```

Ao combinar o conhecimento adquirido, pode criar um programa que realize a tarefa proposta. Analise o código e faça o upload para o Arduino.

```

1 void setup() {
2   pinMode(8, OUTPUT); //LED como saída
3   pinMode(7, INPUT_PULLUP); //Botão como entrada
4   digitalWrite(8, LOW); //Desliga o LED
5 }
6
7 void loop()
8 {
9   if (digitalRead(7) == LOW) { //Se o botão for pressionado
10    digitalWrite(8, HIGH); //Ligar o LED
11  } else { //Se o botão não for pressionado
12    digitalWrite(8, LOW); //Desligar LED
13  }
14 }

```

### Exemplo – Interruptor de Luz com “Temporizador”

Neste exemplo queremos que, tendo em conta o exemplo anterior, o LED esteja ligado por 10 segundos após pressionar o botão.

É capaz de escrever um programa adequado? Esperamos que sim! Se tiver dificuldades, pode sempre dar uma vista de olhos no nosso código:

```

1 void setup() {
2   pinMode(8, OUTPUT); //LED como saída
3   pinMode(7, INPUT_PULLUP); //Botão como entrada
4   digitalWrite(8, LOW); //Desliga o LED
5 }
6
7 void loop()
8 {
9   if (digitalRead(7) == LOW) { //Se o botão for pressionado
10    digitalWrite(8, HIGH); //Ligar o LED
11    delay(10000); //Esperar 10 segundos
12    digitalWrite(8, LOW); //Desligar LED
13  }
14 }

```

### Exemplo – Semáforo

O próximo exemplo é um sistema de semáforo sequencial. O principal objetivo é escrever um programa que, depois do botão ser pressionado, mostre uma sequência correta de luzes.

Vamos assumir o seguinte ciclo:

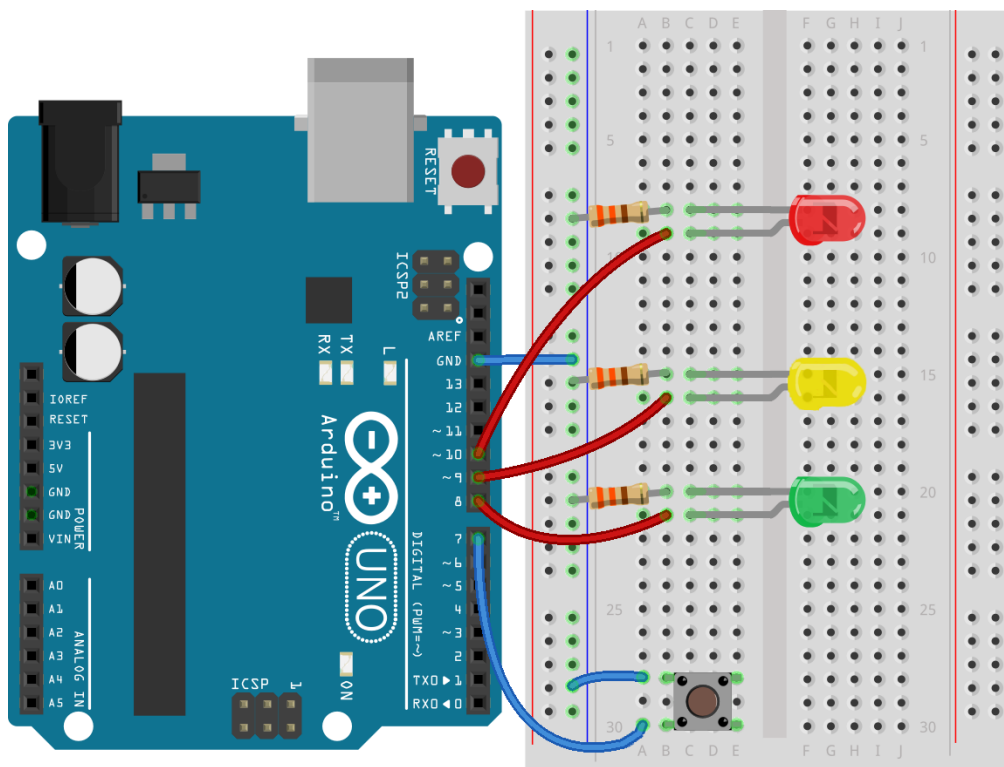
[...] Verde -> Amarelo -> Vermelho -> Amarelo -> Vermelho [...]

#### Material necessário:

- 1x Arduino UNO e cabo USB;

- 1x Breadboard;
- 1x Botão de pressão;
- 1x LED vermelho;
- 1x LED amarelo;
- 1x LED verde;
- 3x Resistências de 330Ω;
- 6x Cabos jumper.

Quando pressionar o botão, o sistema deve iniciar a sequência. Vamos fazê-lo por etapas. Primeiro, ligue os 3 LEDs conforme o esquema abaixo:



fritzing

Vamos preparar um programa que serve apenas para configurar as entradas e saídas.

```

1 void setup() {
2   pinMode(10, OUTPUT); //LED vermelho
3   pinMode(9, OUTPUT); //LED amarelo
4   pinMode(8, OUTPUT); //LED verde
5
6   pinMode(7, INPUT_PULLUP); //Botão
7
8   digitalWrite(10, LOW); //Desligar os LEDs
9   digitalWrite(9, LOW);
10  digitalWrite(8, LOW);
11 }

```

Agora que estão os LEDs e o botão configurados, vamos escrever um programa que mude as luzes automaticamente, a cada 1 segundo. O sketch completo deverá ficar assim:

```
1 void setup() {
2   pinMode(10, OUTPUT); //LED vermelho
3   pinMode(9, OUTPUT); //LED amarelo
4   pinMode(8, OUTPUT); //LED verde
5
6   pinMode(7, INPUT_PULLUP); //Botão
7
8   digitalWrite(10, LOW); //Desligar os LEDs
9   digitalWrite(9, LOW);
10  digitalWrite(8, LOW);
11 }
12
13 void loop()
14 {
15   digitalWrite(10, LOW); //Vermelho
16   digitalWrite(9, LOW); //Amarelo
17   digitalWrite(8, HIGH); //Verde
18
19   delay(1000); //Esperar 1 segundo
20
21   digitalWrite(10, LOW); //Vermelho
22   digitalWrite(9, HIGH); //Amarelo
23   digitalWrite(8, LOW); //Verde
24
25   delay(1000); //Esperar 1 segundo
26
27   digitalWrite(10, HIGH); //Vermelho
28   digitalWrite(9, LOW); //Amarelo
29   digitalWrite(8, LOW); //Verde
30
31   delay(1000); //Esperar 1 segundo
32
33   digitalWrite(10, HIGH); //Vermelho
34   digitalWrite(9, HIGH); //Amarelo
35   digitalWrite(8, LOW); //Verde
36
37   delay(1000); //Esperar 1 segundo
38 }
```

Faça o upload do programa no Arduino e verifique se está tudo a funcionar na perfeição.

## Exercício Prático – Função While

Até agora, usamos apenas o loop principal e obrigatório (função void loop). Agora é hora de conhecer um loop que podemos usar dentro dos nossos programas.

Agora vamos abordar o loop **while** (), que funciona enquanto uma certa condição for cumprida (verdadeira). O seu funcionamento é apresentado no seguinte código:

```

1 | [...]
2 | void loop()
3 | {
4 |
5 | while (condição) {
6 |
7 | /* O código é executado em loop enquanto
8 | a condição for verdadeira */
9 |
10 | }
11 |
12 | }

```

Para uma maior clareza, a função while apenas executa o código que fica entre as suas chavetas {}. O restante código não é executado naquele momento.

Vamos aproveitar o sistema de semáforo que foi montado anteriormente, para escrever um programa que fará piscar um LED de cada vez quando o botão for pressionado. Este exercício é uma tarefa mais difícil do que a anterior.

Confira o código que propomos:

```

1 | void setup() {
2 |   pinMode(10, OUTPUT); //LED vermelho
3 |   pinMode(9, OUTPUT); //LED amarelo
4 |   pinMode(8, OUTPUT); //LED verde
5 |
6 |   pinMode(7, INPUT_PULLUP); //Botão
7 |
8 |   digitalWrite(10, LOW); //Desligar LED
9 |   digitalWrite(9, LOW);
10 |  digitalWrite(8, LOW);
11 | }
12 |
13 | void loop() {
14 |
15 | while (digitalRead(7) == LOW) { //Quando o botão é pressionado
16 |   digitalWrite(10, LOW); //Vermelho desligado
17 |   delay(1000);
18 |   digitalWrite(10, HIGH); //Vermelho ligado
19 |   delay(1000);
20 | }
21 |
22 | }

```

Desta vez, as sequências devem ser exibidas até pressionar o botão. Assumimos que o botão é pressionado e largado muito rapidamente. O programa finalizado deve ter este aspeto:

```

1  void setup() {
2  pinMode(10, OUTPUT); //LED vermelho
3  pinMode(9, OUTPUT); //LED amarelo
4  pinMode(8, OUTPUT); //LED verde
5
6  pinMode(7, INPUT_PULLUP); //Botão
7
8  digitalWrite(10, LOW); //Desligar LED
9  digitalWrite(9, LOW);
10 digitalWrite(8, LOW);
11 }
12
13 void loop()
14 {
15 digitalWrite(10, LOW); //Vermelho
16 digitalWrite(9, LOW); //Amarelo
17 digitalWrite(8, HIGH); //Verde
18
19 while (digitalRead(7) == HIGH) {} //Quando o botão é pressionado
20
21 digitalWrite(10, LOW); //Vermelho
22 digitalWrite(9, HIGH); //Amarelo
23 digitalWrite(8, LOW); //Verde
24
25 while (digitalRead(7) == HIGH) {} //Quando o botão é pressionado
26
27 digitalWrite(10, HIGH); //Vermelho
28 digitalWrite(9, LOW); //Amarelo
29 digitalWrite(8, LOW); //Verde
30
31 while (digitalRead(7) == HIGH) {} //Quando o botão é pressionado
32
33 digitalWrite(10, HIGH); //Vermelho
34 digitalWrite(9, HIGH); //Amarelo
35 digitalWrite(8, LOW); //Verde
36
37 while (digitalRead(7) == HIGH) {} //Quando o botão é pressionado
38 }

```

Neste caso, o loop foi usado de uma maneira bastante estranha. Como pode verificar não há nada dentro das chavetas! Então, como é que o programa está a funcionar? Isso ocorre porque o programa usa loops para parar.

### Como é que está a funcionar?

1. Começamos a iluminar os LEDs de acordo com uma sequência;
2. Entramos na função loop while (), que está imediatamente abaixo;
3. As chavetas estão vazias, portanto, o programa está sempre em loop, sem fazer nada;
4. Somente depois do botão ser pressionado (a condição passa a ser falsa) o programa sai do loop;
5. A sequência seguinte é acionada e a situação repete-se.

Vamos agora verificar o programa na prática!

O que é que está a acontecer? Está tudo a funcionar como suposto? Claro que não! Mesmo quando o botão é pressionado por um curto período de tempo, às vezes o programa funciona corretamente, e outras vezes, salta algumas posições. Por que é que isso acontece?

Como se deve lembrar, o processador, de forma simplificada, realiza cerca de 16 milhões de operações por segundo. Portanto, ao pressionar o botão, o processador será capaz de ter acesso a todos os estados da nossa sinalização. Posto isto, depois de largar o botão poderá haver uma escolha aleatória na sequência.

Como resolver este problema? Muito simples! É suficiente alterar o programa, para que a mudança de luz não ocorra com mais frequência do que, por exemplo, a cada segundo. Para isso, podemos usar a função `delay ()` já conhecida.

```

1 void setup() {
2   pinMode(10, OUTPUT); //LED vermelho
3   pinMode(9, OUTPUT); //LED amarelo
4   pinMode(8, OUTPUT); //LED verde
5
6   pinMode(7, INPUT_PULLUP); //Botão
7
8   digitalWrite(10, LOW); //Desligar LED
9   digitalWrite(9, LOW);
10  digitalWrite(8, LOW);
11 }
12
13 void loop()
14 {
15   digitalWrite(10, LOW); //Vermelho
16   digitalWrite(9, LOW); //Amarelo
17   digitalWrite(8, HIGH); //Verde
18
19   delay(1000); //Parar o programa durante 1 segundo
20   while (digitalRead(7) == HIGH) {} //Quando o botão é pressionado
21
22   digitalWrite(10, LOW); //Vermelho
23   digitalWrite(9, HIGH); //Amarelo
24   digitalWrite(8, LOW); //Verde
25
26   delay(1000); //Parar o programa durante 1 segundo
27   while (digitalRead(7) == HIGH) {} //Quando o botão é pressionado
28
29   digitalWrite(10, HIGH); //Vermelho
30   digitalWrite(9, LOW); //Amarelo
31   digitalWrite(8, LOW); //Verde
32
33   delay(1000); //Parar o programa durante 1 segundo
34   while (digitalRead(7) == HIGH) {} //Quando o botão é pressionado
35
36   digitalWrite(10, HIGH); //Vermelho
37   digitalWrite(9, HIGH); //Amarelo
38   digitalWrite(8, LOW); //Verde
39
40   delay(1000); //Parar o programa durante 1 segundo
41   while (digitalRead(7) == HIGH) {} //Quando o botão é pressionado
42 }

```

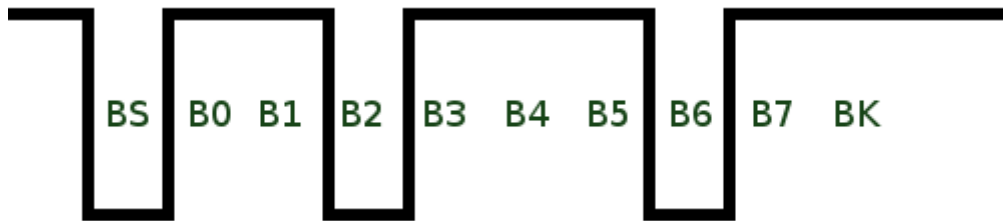
Agora já deve funcionar perfeitamente!

É importante ressaltar que as condições na função while () podem ser combinadas e muito mais complexas.

## UART e Variáveis

### Como é que o UART funciona?

O seu princípio de funcionamento baseia-se no envio série de uma sequência de bits, que são transformados em informação. Um conjunto de dados (data frame) é transmitido da seguinte forma:



A transmissão começa com o start bit, marcado como BS na figura. É sempre um bit que é zero lógico. Depois, dependendo da configuração, existem 7, 8 ou 9 data bits (marcados de B0-B7) que são as informações a ser enviadas. O stop bit (indicado como BK) é um bit um lógico – finaliza a transmissão.

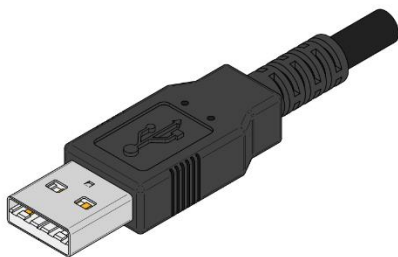
Ao usar o UART no Arduino, devemos ter em consideração dois pinos:

- Tx – envio de dados (pino 1);
- Rx – recepção de dados (pino 0).

Para que a transmissão funcione corretamente, deve ser definida a mesma velocidade de transferência de dados em ambos os sistemas – conhecida como **baud-rate** ou taxa de transmissão. Esta especifica o número de bits transmitidos por segundo. Os valores mais utilizados são: 9.600 e 112.500.

O computador com o qual pretendemos estabelecer comunicação também deve estar equipado com a interface apropriada. Infelizmente, os fabricantes de PCs pararam de inserir a porta série RS-232, que há alguns anos fazia parte do equipamento básico da maioria dos computadores.

Resta-nos a comunicação USB. Infelizmente, esta é uma tarefa bem difícil. Desta forma, geralmente são usados conversores USB-UART, o que simplifica bastante o trabalho. A boa notícia é que não precisa de se preocupar com isso na utilização do Arduino, o conversor já vem integrado na placa.



Portanto, tudo o que precisa de fazer é ligar o Arduino ao computador através do cabo USB (o mesmo que é usado na programação).

Vamos passar para os exemplos práticos. Os dois primeiros apenas requerem ligar o Arduino via USB ao computador. Só iremos adicionar periféricos ao sistema mais tarde.

## Exercício Prático – Comunicação UART

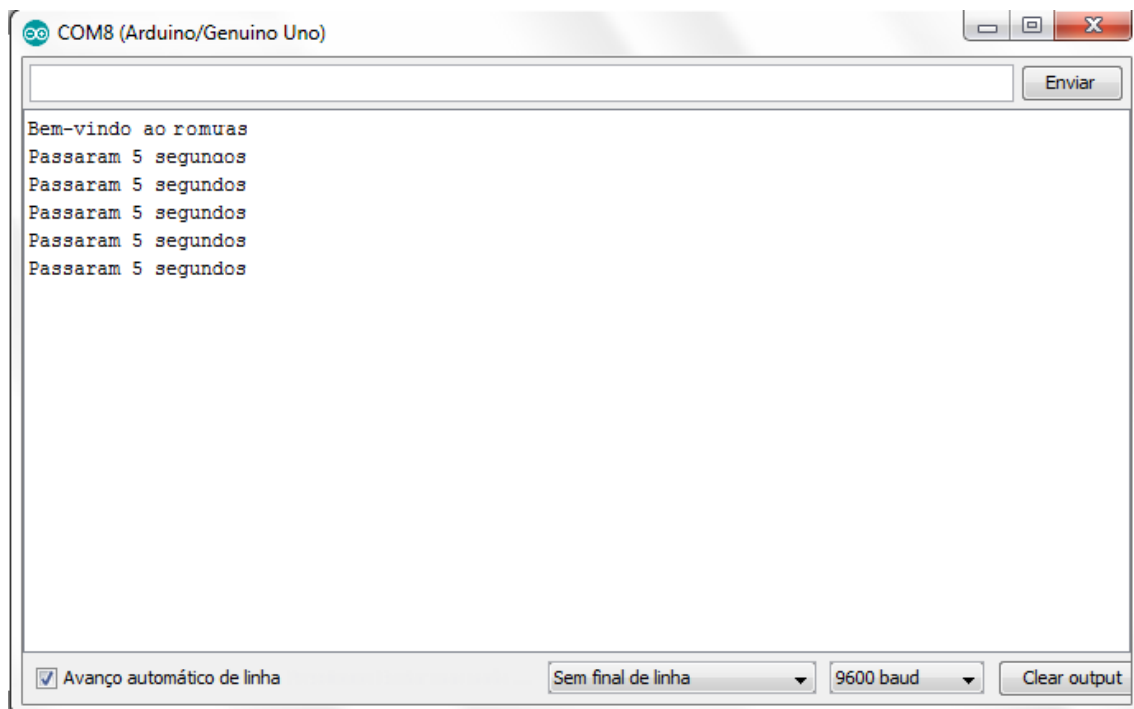
### Material necessário:

- 1x Arduino UNO e cabo USB.

O objetivo do programa abaixo é muito simples: enviar um texto para o computador:

```
1 void setup(){
2   Serial.begin(9600); //Configuração da velocidade de transmissão
3   Serial.println("Bem-vindo ao ROMUAS!"); //Envio de texto único
4 }
5 void loop() {
6   delay(5000);
7   Serial.println("Passaram 5 segundos"); //Envio de texto em loop
8 }
```

Após o upload do programa acima, aparentemente nada acontece. Para verificar o seu funcionamento, precisa de selecionar no menu do Arduino: Ferramentas -> Monitor Série. Depois disso, vai abrir uma nova janela. Aqui, podemos observar o que é enviado para/do Arduino através da porta COM, que é o nosso UART. Vejamos a operação em prática:



Vamos agora analisar o programa. A primeira coisa ser feita foi a definição da baud-rate. A função `Serial.begin()` é usada para este propósito, onde entre parênteses se encontra a velocidade de transmissão. Nesse caso, é 9600 baud/seg. Por outro lado, a função `Serial.println()` é usada para enviar uma informação (frase ou números).

O texto “Bem-vindo ao ROMUAS!” é exibido apenas uma vez, porque está inserido na função void setup e, como se deve lembrar do capítulo anterior do curso, as instruções inseridas nessa função são realizadas apenas uma vez.

A transmissão também pode ser observada nos LEDs integrados no Arduino (Tx e Rx)! Estes acendem quando os dados estão a ser transferidos para/da placa.

## Exercício Prático – Interação com o programa

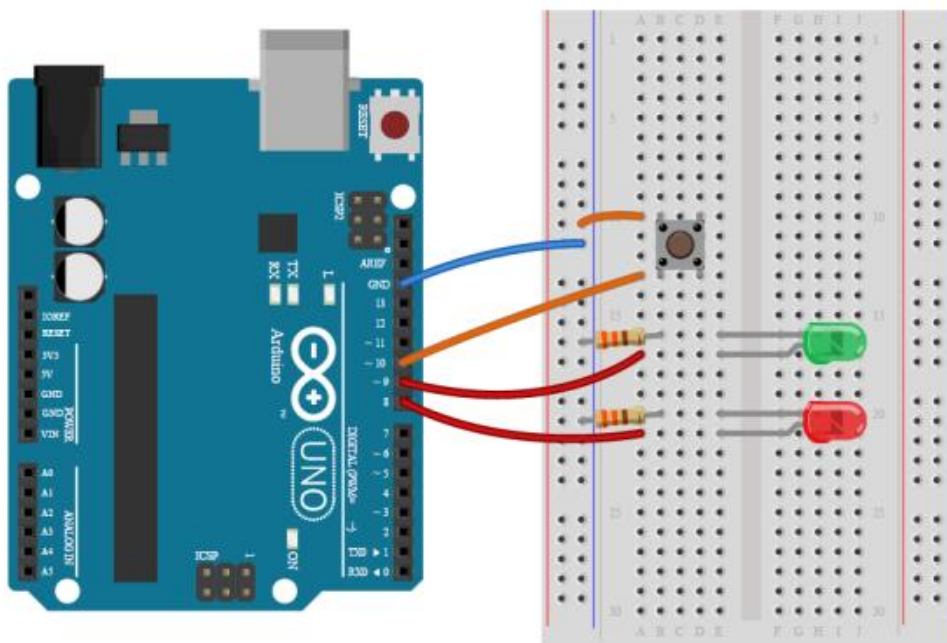
É claro que as informações não precisam de ser sempre enviadas por UART, a transmissão e recepção também podem ocorrer uma vez num momento escolhido. Isto é muito útil, por exemplo, para diagnosticar a operação do sistema ou sinalizar diferentes ocorrências.

Usando o conhecimento adquirido anteriormente, poderá escrever um sketch que ativa um LED quando uma janela está aberta. É claro que não vamos usar um sensor de abertura de janelas, vamos simular utilizado componentes mais simples. Um botão de pressão vai substituir o sensor e dois LEDs servirão para sinalização.

Material necessário:

- 1x Arduino UNO e cabo USB;
- 1x Breadboard;
- 1x LED vermelho;
- 1x LED verde;
- 1x Botão de pressão;
- 2x Resistências de 330Ω;
- 5x Cabos jumper.

Faça a ligação da seguinte forma:



Quando a janela está fechada (botão pressionado), o LED verde está aceso. Quando abrimos o circuito (paramos de carregar no botão) o LED vermelho acende e no monitor série vamos ler a mensagem “Atenção! Alarme! A janela está aberta!”.

```
1 void setup(){
2   Serial.begin(9600); //Configuração da velocidade de transmissão
3
4   pinMode(8, OUTPUT); //LED vermelho
5   pinMode(9, OUTPUT); //LED verde
6   pinMode(10, INPUT_PULLUP); //Botão
7
8   digitalWrite(8, LOW); //Desligar LEDs
9   digitalWrite(9, LOW);
10  }
11
12 void loop() {
13   if (digitalRead(10) == LOW) { //Se o botão for pressionado
14     digitalWrite(9, HIGH); //Liga o LED verde
15     digitalWrite(8, LOW); //Desliga o LED vermelho
16   } else { //Se o botão não for pressionado
17     digitalWrite(9, LOW); //Desliga o LED verde
18     digitalWrite(8, HIGH); //Liga o LED vermelho
19     Serial.println("Atenção! Alarme! A janela está aberta!");
20   }
21 }
```

Vamos verificar como é que o programa está a funcionar! Infelizmente, não está muito bem. A informação de alarme é enviada o tempo todo. Preferimos que seja enviada apenas uma vez. Sabe como mudar isso? Claro, inserindo a função while!

```
1 void setup(){
2   Serial.begin(9600); //Configuração da velocidade de transmissão
3
4   pinMode(8, OUTPUT); //LED vermelho
5   pinMode(9, OUTPUT); //LED verde
6   pinMode(10, INPUT_PULLUP); //Botão
7
8   digitalWrite(8, LOW); //Desligar LEDs
9   digitalWrite(9, LOW);
10  }
11
12 void loop() {
13
14   if (digitalRead(10) == LOW) { //Se o botão for pressionado
15     digitalWrite(9, HIGH); //Liga o LED verde
16     digitalWrite(8, LOW); //Desliga o LED vermelho
17   } else { //Se o botão não for pressionado
18     digitalWrite(9, LOW); //Desliga o LED verde
19     digitalWrite(8, HIGH); //Liga o LED vermelho
20     Serial.println("Atenção! Alarme! A janela está aberta!");
21
22     while (digitalRead(10) == HIGH) {
23       //Criação de um loop vazio para a janela voltar a fechar
24       delay(25); //Atraso de 25ms dentro do loop para minimizar interferências
25     }
26
27   }
28 }
```

## Instrução #define

Com o tempo, os nossos programas vão aumentar consideravelmente. E se for necessário alterar a conexão física de, por exemplo, um LED ou um botão? Alterar o número do pino de todo o código seria bastante difícil.

A instrução #define ajuda nesse aspeto. Permite que defina um símbolo para um determinado pino, que será substituído pelo número deste antes da compilação, em qualquer local do código. Por exemplo:

```
1 | #define ledPin 8
2 |
3 | void setup() {
4 |   pinMode(ledPin, OUTPUT); //Configuração do pino 8 como saída
5 | }
6 |
7 | void loop() {
8 |   digitalWrite(ledPin, HIGH); //Liga o LED
9 |   delay(1000); //Espera 1 segundo
10 |  digitalWrite(ledPin, LOW); //Desliga o LED
11 |  delay(1000); //Espera 1 segundo
12 | }
```

Colocando a linha: #define ledPin 8 no início do código, fazemos com que, antes da compilação, qualquer parte do programa que possua como pino o “ledPin”, seja automaticamente transformada no número definido para este, nomeadamente 8. É claro que o nome do pino pode ser diferente, o importante é estabelecer um nome único, que o ajude a escrever programas longos.

**IMPORTANTE:** Depois da instrução #define não se coloca ponto e vírgula (;).

De seguida, está a nova e melhorada versão do código do sensor de abertura de janelas:

```

1  #define LEDvermelho 8
2  #define LEDverde 9
3  #define Botão 10
4
5  void setup(){
6  Serial.begin(9600); //Configuração da velocidade de transmissão
7
8  pinMode(LEDvermelho, OUTPUT); //LED vermelho como saída
9  pinMode(LEDverde, OUTPUT); //LED verde como saída
10 pinMode(Botão, INPUT_PULLUP); //Botão
11
12 digitalWrite(LEDvermelho, LOW); //Desligar LEDs
13 digitalWrite(LEDverde, LOW);
14 }
15
16 void loop() {
17
18 if (digitalRead(Botão) == LOW) { //Se o botão for pressionado
19 digitalWrite(LEDverde, HIGH); //Liga LED verde
20 digitalWrite(LEDvermelho, LOW); //Desliga LED vermelho
21 } else { //Se o botão não for pressionado
22 digitalWrite(LEDverde, LOW); //Desliga LED verde
23 digitalWrite(LEDvermelho, HIGH); //Liga LED vermelho
24 Serial.println("Atenção! Alarme! A janela está aberta!");
25
26 while (digitalRead(Botão) == HIGH) {
27 //Criação de um loop vazio para a janela voltar a fechar
28 delay(25); //Atraso de 25ms dentro do loop para minimizar interferências
29 }
30
31 }
32 }

```

## Variáveis

Antes de passarmos para outros programas (incluindo envio de informações para o Arduino via UART), temos que saber o que são variáveis, e como é que estas funcionam.

As variáveis, de forma geral, são declarações de algum tipo de informação necessária para o código. Podem ser caracteres, palavras ou números. Na maioria das vezes, vamo-nos deparar com variáveis numéricas.

Quando é que as variáveis são necessárias? Quando queremos guardar um valor e executar vários tipos de operações com ele. Uma variável, assim como uma função, pode ter um tipo específico de informação, acerca de que tipo de dados pode armazenar.

Abaixo pode encontrar uma lista dos tipos de variáveis mais importantes:

```

1  boolean logica = false; //Boolean - verdadeiro (true) ou falso (false)
2
3  int numero = 30000; //Int - números inteiros no intervalo de -32768 a 32767
4  long numeroGrande = 2000000; //Long - números inteiros no intervalo de
5  -2147483648 a 2147483647
6
7  float numeroRacional = 6.28; //Float - números racionais que ocupem até 4
8  bytes de memória
9
10 char caractere = 'a'; //Char - caracteres
11 String frase = "Bem-vindo ao blog ElectroFun!"; //String - sequência de
12 caracteres

```

**NOTA:** Os valores máximos que podem ser gravados numa variável dependem da placa Arduino utilizada. Os valores acima são apropriados para o Arduino UNO.

De início, iremos usar mais frequentemente as seguintes variáveis:

- **Boolean** – como mencionado, é usado para armazenar valores verdadeiros ou falsos. Este tipo de variável, geralmente é usado para sinalizar ocorrências ou condições de controlo;
- **Int** – é a variável mais comum para o armazenamento de números inteiros. Pode armazenar informações como o número de toques no teclado, quantas vezes ocorreu uma determinada situação ou um valor dado pelo sensor de distância. E, claro, pode realizar operações matemáticas nas variáveis;
- **String** – é um conjunto de caracteres, ou seja, de forma simplificada, podemos guardar uma palavra, frase, legenda, mensagem, etc.

## Declaração de Variáveis

De forma a usar uma variável, é necessário declará-la, isto é, informar o compilador sobre o seu tipo e o seu nome. O nome de cada variável começa sempre com uma letra, nunca com um número, e não pode conter espaços. A declaração de uma variável deve ser feita da seguinte forma:

```
1 | tipo nome = 0;
```

**IMPORTANTE:** Note que o símbolo = é usado para atribuir um valor de variável, e o símbolo == serve para comparar a igualdade entre variáveis e valores.

Também é importante referir que se uma variável for colocada dentro de uma função, instrução ou subprograma, esta ficará invisível (não poderemos usá-la) noutras funções. Vejamos:

```
1 | int variavel = 0; //Variável global - pode ser usada em qualquer parte do
2 | programa
3 |
4 | void setup() {
5 |   int variavel2 = 0; //Variável local - só pode ser utilizada dentro da função
6 |   setup()
7 | }
8 |
9 | void loop() {
   |   int variavel3 = 0; //Variável local - só pode ser utilizada dentro da função
   |   loop()
   | }
```

É normal que, para já, esteja tudo ainda um pouco verde, mas com tempo aprenderá tudo na prática. De início, para facilitar a aprendizagem, usaremos apenas variáveis globais. Os programadores experientes podem não gostar, no entanto, chegaremos a tudo, pouco a pouco.

Há mais um aspeto importante a apontar – denominar as variáveis. Lembre-se de dar nomes às variáveis que determinem o seu propósito. Por exemplo, em vez de:

```
1 | string xx = "Manuel";
```

Coloque antes um nome único que identifique a variável:

```
1 | string nomePessoa = "Manuel";
```

O problema poderá surgir quando o propósito das variáveis for mais complicado de definir. Não tenha receio de criar nomes como:

```
1 | int vMotorEsquerdo = 100; //Velocidade do motor esquerdo
```

O mais importante é tornar o código legível e inteligível!

## Exercício Prático – Variáveis

A teoria já passou, agora é hora de praticar o que aprendeu. Vamos começar com algo muito simples. De início, deixe o nosso código escrever um valor de variável, que vai aumentando, em cada ciclo de *loop*.

```
1 | int contador = 0; //Declaração da variável
2 |
3 | void setup() {
4 |   Serial.begin(9600); //Configuração da velocidade de transmissão
5 | }
6 |
7 | void loop() {
8 |   Serial.println(contador); //Enviar o valor da variável
9 |   contador = contador + 1; //Somar 1 ao valor do contador
10 |   delay(100); //Atraso para tornar o programa mais perceptível
11 | }
```

Obviamente, a declaração de variável foi colocada no início, fora de qualquer função. Graças a isso, podemos ter acesso à variável em qualquer parte do programa.

Primeiro, a taxa de transmissão é definida e inicializada e, de seguida, a função `loop ()` executa 3 ações:

- 1) Invoca o local da memória, no qual declaramos uma variável denominada “contador”, e envia o valor encontrado via UART;
- 2) Aumenta +1 no valor recebido do contador;
- 3) Espera 100ms (para uma melhor percepção) e volta ao início do *loop*.

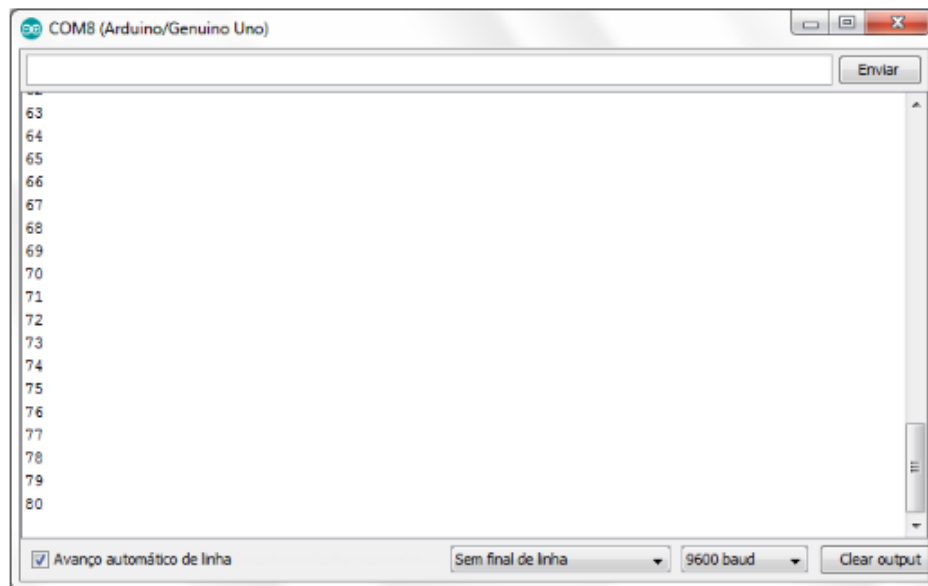
Passaremos a explicar o ponto 2, que é o aumento do valor da variável. A instrução utilizada foi a seguinte:

```
1 | contador = contador + 1; //Somar 1 ao valor do contador
```

Do ponto de vista matemático, onde o sinal “=” significa igualdade, a linha acima não deveria funcionar. No entanto, na programação, o sinal “=” significa atribuição. Na prática, o código acima deve ser entendido como a seguinte operação:

- 1) Ir buscar o valor da variável `contador`;
- 2) Somar 1 ao valor recebido;
- 3) Receber o resultado da operação e atribuí-lo à própria variável.

Faça o download do programa para o Arduino e confira se está tudo a funcionar corretamente. É claro que, para ver os resultados, deverá abrir o monitor série.



## Transmissão Bidirecional do Arduino

É claro que a comunicação, para ser útil, deve ocorrer de forma bidirecional. Até agora, tem sido o Arduino a enviar-nos informações. Está na hora de lhe respondermos!

O objetivo do primeiro programa é “ouvir” o nosso nome. Quando lhe enviarmos o nome, o Arduino deverá responder com a seguinte mensagem “Olá, Nome!”, onde, obviamente, o nome será o anteriormente enviado.

```
1 String dadosRecebidos = ""; //Conjunto vazio de dados recebidos
2
3 void setup() {
4   Serial.begin(9600); //Configuração da velocidade de transmissão
5 }
6
7 void loop() {
8   if(Serial.available() > 0) { //Se o Arduino receber dados
9     dadosRecebidos = Serial.readStringUntil('\n'); // Lê os dados recebidos e
10    guarda na própria variável
11    Serial.println("Bem-vindo " + dadosRecebidos + "!"); //Mostrar a mensagem
12  }
13 }
```

Primeiramente, declaramos a variável `dadosRecebidos`, para a qual o conjunto de caracteres recebido (nome) será copiado. De seguida, como de costume, definimos a taxa de transmissão e iniciá-mo-la. Depois introduzimos uma nova função: **Serial.available ()**. Esta envia o número de bytes que foram recebidos e que estão a aguardar suporte do Arduino.

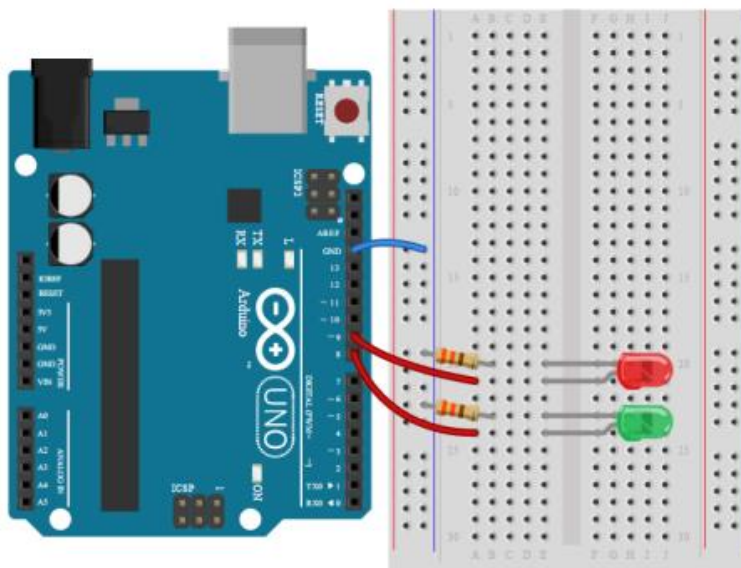
No caso dos dados já estarem disponíveis (maior que 0), são enviados para a variável `dadosRecebidos`. Isto é feito através da função **.readStringUntil** (término) que copia os dados do buffer até encontrar um caractere de finalização (neste caso, “\n”).

## Exercício Prático de Interação com o Sistema – Controle de LEDs via UART

### Material necessário:

- 1x Arduino UNO e cabo USB;
- 1x Breadboard;
- 1x LED verde;
- 1x LED vermelho;
- 2x Resistências de 330Ω;
- 3x Cabos jumper.

Neste exercício vamos usar capacidade de enviar texto para o Arduino para controlar LEDs. Para isso, ligue dois LEDs de acordo com o esquema abaixo (LEDs nos pinos 8 e 9):



O objetivo do programa é ligar o LED verde ou vermelho por 1 segundo, quando é enviado um comando apropriado para o Arduino. O código final é o seguinte:

```

1  #define verde 8
2  #define vermelho 9
3
4  String dadosRecebidos = ""; //Conjunto vazio de dados recebidos
5
6  void setup() {
7  Serial.begin(9600); //Configuração da velocidade de transmissão
8  pinMode(verde, OUTPUT); //Configuração dos LEDs como saídas
9  pinMode(vermelho, OUTPUT);
10 |
11  digitalWrite(verde, LOW); //Desligar LEDs
12  digitalWrite(vermelho, LOW);
13  }
14
15  void loop() {
16  if(Serial.available() > 0) { //Se o Arduino receber dados
17  dadosRecebidos = Serial.readStringUntil('\n'); //Lê os dados recebidos e guarda
18  na própria variável
19
20  if (dadosRecebidos == "verde") { //Se escrever a palavra "verde"
21  digitalWrite(verde, HIGH); //Liga o LED verde
22  delay(1000); //Espera 1 segundo
23  digitalWrite(verde, LOW); //Desliga o LED verde
24  }
25
26  if (dadosRecebidos == "vermelho") { //Se escrever a palavra "vermelho"
27  digitalWrite(vermelho, HIGH); //Liga o LED vermelho
28  delay(1000); //Espera 1 segundo
29  digitalWrite(vermelho, LOW); //Desliga o LED vermelho
30  }
31  }
}

```

Vamos agora analisar o funcionamento do programa. Inicialmente, os números dos pinos com LEDs são definidos e a variável para a qual os dados recebidos são copiados é declarada. No *loop*, é verificado se o Arduino recebeu os dados. Se sim, é averiguado se esses dados correspondem a uma das cores. Depois disto, é ligado o LED da cor indicada.

## O que é o sinal PWM?

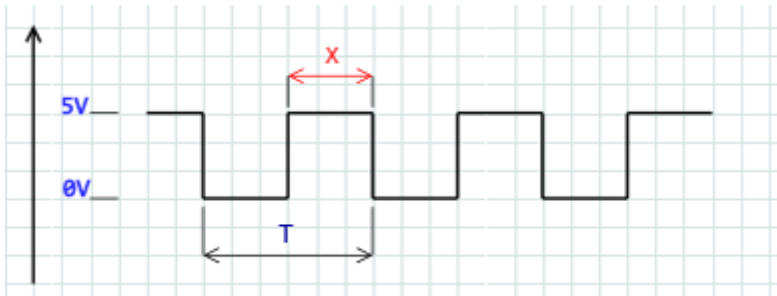
Suponha que liga um LED ao microcontrolador e programa-o para piscar em *loop*. O LED fica ligado por um segundo, no próximo permanece desligado e assim em diante:

```

1  void setup() {
2  pinMode(3, OUTPUT); //Configuração do LED como saída
3  }
4
5  void loop() {
6  digitalWrite(3, HIGH); //Liga o LED
7  delay(1000); //Espera 1 segundo
8  digitalWrite(3, LOW); //Desliga o LED
9  delay(1000); //Espera 1 segundo
10 | }

```

Se desenhássemos um gráfico que desmonstrasse a mudança de tensão em função do tempo do pino 3, obteríamos a seguinte onda:



O valor marcado como  $x$  é o tempo em que o LED está ligado. De forma oposta,  $T$  é o período de tempo em que o LED está desligado. Por sua vez, o seu inverso, isto é,  $1/T$ , indica a frequência. A relação entre o tempo que o LED está ligado e o tempo que o LED está desligado é 1:1. Por outras palavras, o LED está ativo durante 50% da operação do programa. Este aspeto é denominado **duty cycle**.

Para resumir as informações do sinal acima:

- Amplitude (valor máximo): 5V;
- Período (ciclo): 2 segundos;
- Frequência:  $1/2 = 0,5$  Hz;
- *Duty cycle*: 50%.

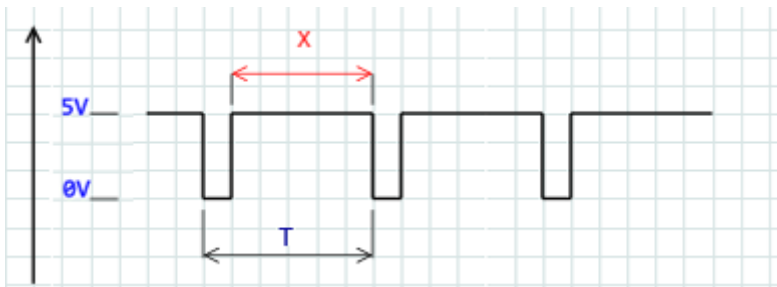
Agora outro exercício semelhante. No entanto, com um *duty cycle* diferente, ainda que mantendo o período. Como fazê-lo? Basta prolongar o tempo de operação do LED, reduzindo o tempo em que desligado. Por exemplo:

```

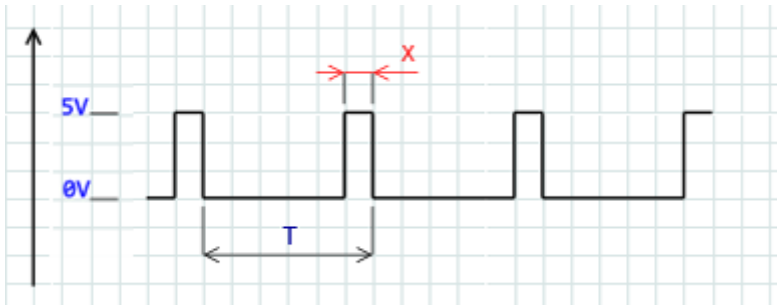
1 void setup() {
2   pinMode(3, OUTPUT); //Configuração do LED como saída
3 }
4
5 void loop() {
6   digitalWrite(3, HIGH); //Ligar o LED
7   delay(1667);
8   digitalWrite(3, LOW); //Desligar o LED
9   delay(333);
10 }

```

Desta vez, o LED fica ligado cerca de 5/6 do tempo. Então o *duty cycle* é cerca de 83%. Apresentando a situação num gráfico obtemos:



De forma oposta, se trocarmos os delays, o *duty cycle* do sinal fica cerca de 17%. Vejamos:



Dê uma vista de olhos nos exemplos acima. Qual dos parâmetros mudou em cada exemplo? A resposta é fácil: o *duty cycle*. A frequência permaneceu igual.

Agora imagine que os *delays* inseridos nos códigos acima eram muito mais pequenos, graças aos quais a frequência do sinal é muito maior... Parabéns! Acabou de entender o princípio do PWM. É um método de modular um sinal retangular através do ajuste da largura de pulso.

### Para que é que o sinal PWM é usado?

Este sinal é usado com muita frequência. Com este, pode controlar o brilho de um LED, a posição de um servo e a velocidade à qual um motor funciona! Como terá oportunidade de ver, possui inúmeras aplicações em robótica, bem como noutros projetos DIY.

### Exercício Prático de PWM – Controlo do brilho de um LED

Está na hora do primeiro exemplo prático de aplicação do PWM. Para já, vamos criar um programa muito simples, cujo objetivo será colocar regular o brilho de um LED.

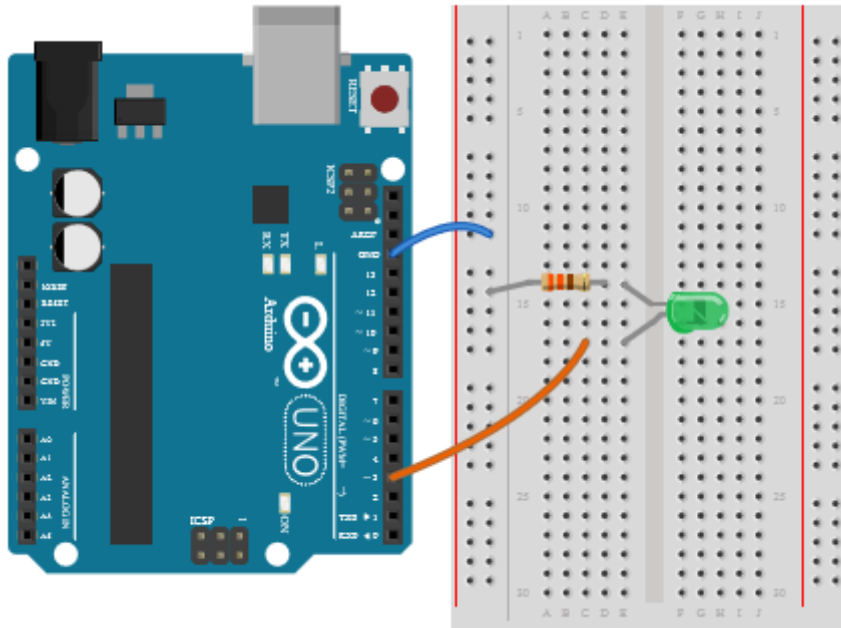
Material necessário:

- 1x Arduino UNO e cabo USB;
- 1x Breadboard;
- 1x LED;
- 1x Resistência de 330Ω;
- 2x Cabos jumper.

O Arduino está equipado com 6 canais PWM. Cada saída, na qual podemos obter o sinal PWM, foi marcada na placa com um til “~”.

Cada canal PWM disponível no Arduino UNO é de 8 bits. Isto significa que, o sinal que queremos receber na sua saída, pode ser definido por um número de 0 a 255, onde 255 significa 100% de *duty cycle*.

Para realizar o primeiro exercício, é necessário ligar o LED ao pino 3 da seguinte forma:



Agora vamos escrever o código. O objetivo é o LED acender lentamente.

```

1 | #define pinoLED 3
2 |
3 | int dutyCycle = 0;
4 | int mudanca = 5;
5 |
6 | void setup() {
7 |   pinMode(pinoLED, OUTPUT); //Configuração do LED como saída
8 | }
9 |
10 | void loop() {
11 |   analogWrite(pinoLED, dutyCycle); //Gerar um sinal com determinado duty cycle
12 |
13 |   if (dutyCycle < 255) { //Se o duty cycle for menor do que 100%
14 |     dutyCycle = dutyCycle + mudanca; //Aumenta o duty cycle
15 |   } else {
16 |     dutyCycle = 0; //Se o duty cycle for igual a 100%, volta ao início
17 |   }
18 |
19 |   delay(50); //Pequeno atraso para tornar o efeito visível
20 | }

```

Esperamos que tenha ficado claro. Podemos agora discutir a nova função inserida: `analogWrite (pino, duty cycle)`. O seu objetivo é gerar o sinal PWM no pino selecionado com o *duty cycle* indicado.

O programa acima pretende aumentar periodicamente o *duty cycle* de zero para o momento em que o seu valor é imediatamente menor do que 255 (100%). Quando é alcançado o *duty cycle* máximo, o LED desliga e o processo é repetido.

## Servo Motor

Está na altura de usar o servo motor! O que vamos utilizar é o **SG90**, do tipo micro, um dos mais pequenos disponíveis no mercado. Contudo, o seu tamanho não afeta o modo de

controlo. Depois de entender o princípio de operação, poderá aplicá-lo em servos maiores, mais potentes e mais rápidos.



## O que é um servo?

Um servomecanismo é um motor, caixa de velocidades e controlador num só dispositivo. No entanto, estes motores não são projetados para executar rotações completas. Na maioria das vezes, os servos possuem um ângulo de rotação de 0-180°. É importante saber que eles conhecem a sua posição atual, por isso não precisa de se preocupar com erros de posição.

Os princípios mais importantes da utilização dos servos:

1. Não se deve girar manualmente a posição do eixo, sem necessidade. Pode danificar as delicadas engrenagens de plástico;
2. Não se deve alimentar o servo diretamente da fonte de alimentação usada no restante sistema. Cada motor recebe uma corrente relativamente alta, especialmente no início do movimento. Isso pode perturbar o funcionamento dos outros dispositivos e, em casos extremos, danificá-los.

## Como é que o servo funciona?

Como é que o servomecanismo sabe para qual posição girar? Graças ao driver integrado. É ele que, com base no sinal PWM fornecido, controla o motor.

Um padrão aceitável é o envio de um sinal com um período de 20ms para o servo. O *duty cycle* é interpretado como a posição para a qual o servo deve ser movido. O *duty cycle* do sinal gerado deve encontrar-se entre 5 e 10%. Estes valores serão convertidos em duas posições extremas no servo (máximo esquerdo e máximo direito).

Fios de ligação do servo:

- **Vermelho** – alimentação;
- **Amarelo** ou **laranja** – controlo do sinal;
- **Preto** ou **castanho** – GND.

Dependendo do fabricante, as cores dos fios podem variar. No entanto, dois serão definitivamente preto/castanho e vermelho. O restante será o fio de sinal.

## Alimentação do servo

Como foi referido anteriormente, não deve alimentar o servo diretamente da mesma fonte que alimenta o microcontrolador. Portanto, devido ao fato de o motor consumir uma grande corrente, deve ser utilizada uma fonte adequada para alimentação do sistema.

Infelizmente, a alimentação a partir da porta USB, como havíamos feito até agora, não é suficiente. Portanto, pela primeira vez, vamos alimentar a placa com uma bateria de 9V!

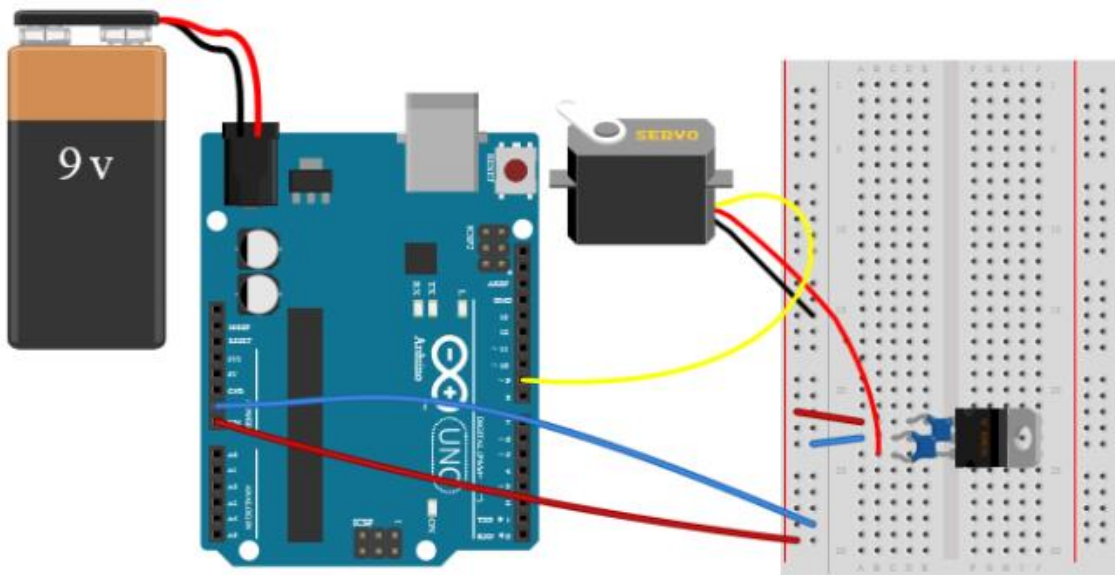
## Exercício Prático – Servomecanismo

Vamos passar ao primeiro programa com o servo motor. Para isto, ligue o sistema de acordo com o diagrama de montagem abaixo.

### Material necessário:

- 1x Arduino UNO e cabo USB;
- 1x Breadboard;
- 1x Servo Motor SG90;
- 1x Regulador de tensão LM7805;
- 1x Bateria de 9V;
- 1x Conector para bateria de 9V;
- 7x Cabos jumper.

Em primeiro lugar, é necessário ligar a bateria. Em segundo, é preciso incluir um regulador de tensão LM7805.



Agora o código que fará o servo mover-se gradualmente:

```

1  #include <Servo.h> //Biblioteca responsável pelo servo motor
2
3  Servo servomecanismo; //Declaração do servo como servomecanismo
4  int posicao = 0; //Posição atual do servo de 0-180 graus
5  int mudanca = 6; //Qual deve ser a posição do servo?
6
7  void setup()
8  {
9  servomecanismo.attach(9); //Servo ligado ao pino 9
10 }
11
12 void loop()
13 {
14 if (posicao < 180) { //Se a posição for inferior a 180 graus
15 servomecanismo.write(posicao); //Move-se
16 } else { //Caso contrário, volta ao início
17 posicao = 0;
18 }
19
20 posicao = posicao + mudanca; //Aumentar a posição atual do servo
21 delay(200); //Atraso para melhor efeito
22 }

```

Desta vez, adicionamos uma biblioteca que vai expandir as capacidades do programa com as nossas funções. O comando usado é:

```
1 | #include Servo.h
```

Neste caso, adicionamos o arquivo Servo.h, que contém instruções adicionais do servo. Graças a este, não precisamos de ser nós a controlar o sinal PWM. É suficiente indicar as posições (ângulo) para as quais queremos que o servo gire.

Para controlar o servo, é necessário declará-lo:

```
1 | Servo servomecanismo;
```

A **função attach (pino)** – para o objeto Servo – funciona de maneira semelhante ao **pinMode**. A partir desta instrução, será gerado, na saída indicada (neste caso 9), um sinal PWM.

Depois de iniciar o programa, o servo deve mover-se suavemente de uma posição extrema para a outra em *loop*. A instrução chave é:

```
1 | servomecanismo.write(posicao);
```

A posição deverá ser um ângulo entre 0 e 180°.

## Introdução aos Displays

Ligar e controlar um display gráfico ou de texto pode parecer muito difícil. Afinal, existem muitos pixels no ecrã e é preciso controlar tudo.

Cada letra consiste em vários pixels, dispostos em retângulos. O display que vamos utilizar é um LCD 16×2. Isto significa que possui 2 linhas, cada uma com 16 caracteres. Esta é uma limitação destes pequenos displays. Existem outros com muito mais capacidade de texto, onde se podem exibir imagens, etc.

Os displays LCD estão disponíveis em várias cores, como por exemplo verde, azul e preto. Pode utilizar o que lhe for mais conveniente.

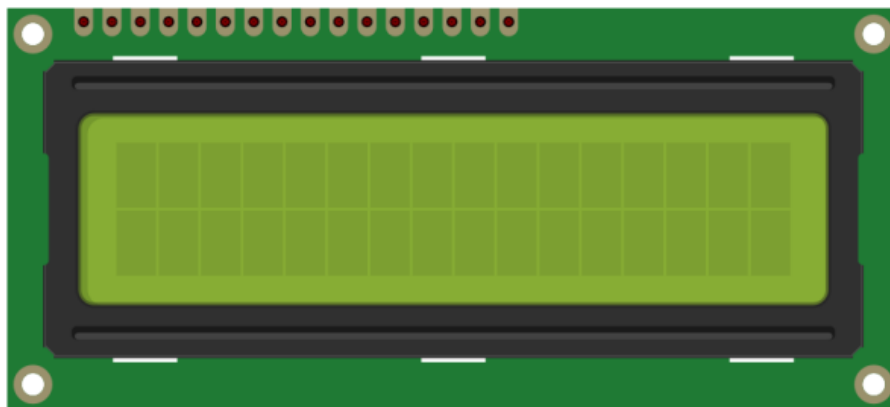
## Como é que o display funciona?

Como mencionado anteriormente, as letras do display consistem em pixels. Controlar cada um individualmente, significaria criar um código com uma imensidão de linhas. Mas, é claro que isso não é necessário, o display funciona muito mais facilmente graças aos drivers integrados. O mais conhecido é o HD44780.

Muitas vezes, na descrição de um determinado display, encontra-se indicação de que o display é compatível com o driver HD44780. Quando isso se verifica sabe-se que o seu funcionamento será muito simples!

Nesse caso, como é que se pode enviar um texto para o display? É necessário ligar aproximadamente 12 fios. É claro que, apenas alguns deles são usados na comunicação, os restantes são necessários para alimentação e outros sinais.

Na maioria das vezes, essa o display está equipado com um conector de 16 pinos:



Vejamos a legenda dos pinos, da esquerda para a direita:

1. VSS – terra (GND);
2. VDD – fonte de alimentação +5V;
3. VO – ajuste do contraste;
4. RS – seleção de registo;
5. R/W – seleção de escrita ou leitura;
6. E – habilitação de sinal;

7. DB0 – Dados;
8. DB1 – Dados;
9. DB2 – Dados;
10. DB3 – Dados;
11. DB4 – Dados;
12. DB5 – Dados;
13. DB6 – Dados;
14. DB7 – Dados;
15. LED+ – aumento da luz de fundo;
16. LED- – diminuição da luz de fundo.

Os pinos de 1 a 3 são usados para alimentar o display, os pinos de 4 a 14 para controlar o display, e os últimos dois (15 e 16) para regular a luz de fundo.

Os displays compatíveis com o driver HD44780 podem comunicar através de 4 e 8 bits. No primeiro modo, são necessárias 7 conexões com o Arduino. No entanto, no modo 8 bits, deve realizar até 11 conexões.

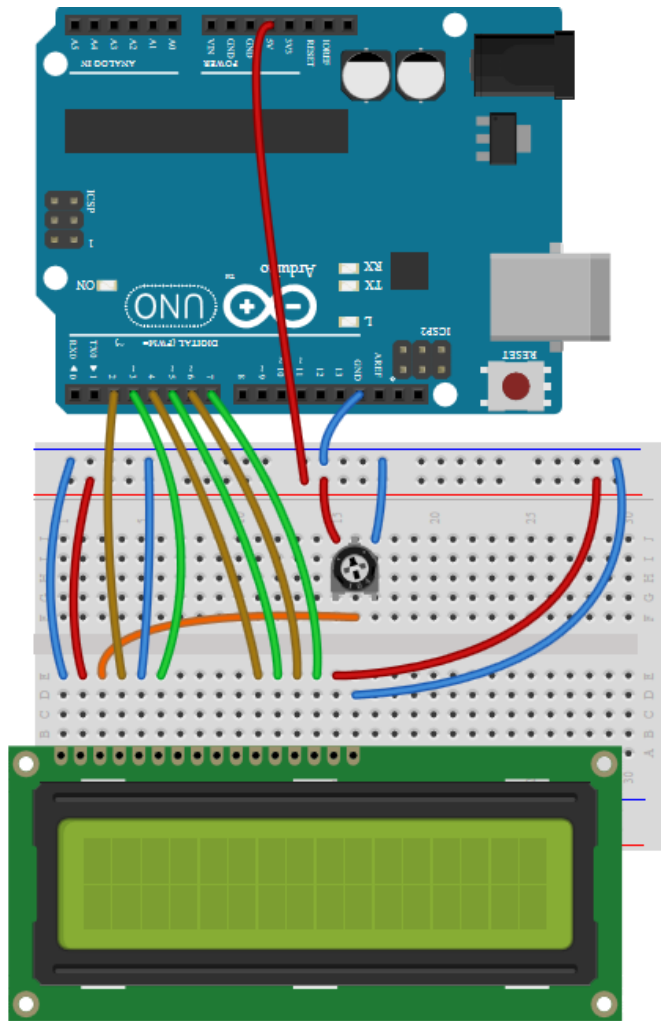
## **Ligação do display ao Arduino**

O Arduino possui uma biblioteca especial para a exibição de textos. Porém, antes de passarmos para a programação, é preciso fazer as ligações do sistema.

### **Material necessário:**

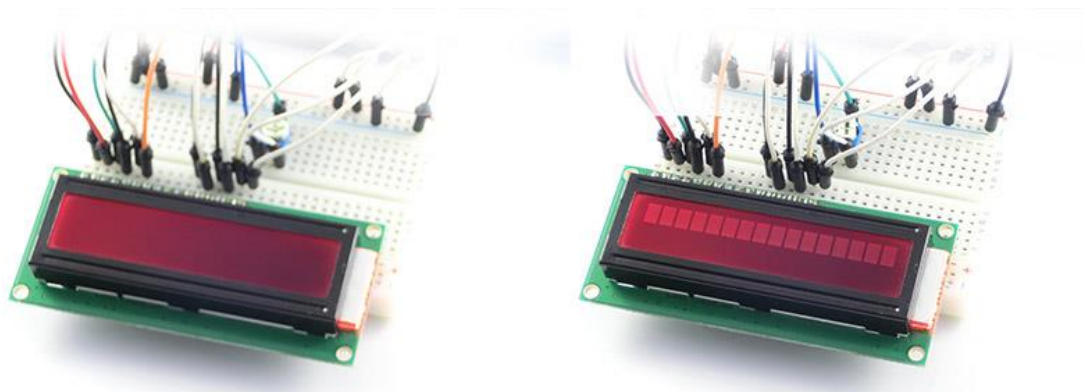
- 1x Arduino UNO e cabo USB;
- 1x Breadboard;
- 1x Display LCD 16x2;
- 1x Potenciômetro;
- 16x Cabos jumper.

Faça as ligações tendo em conta o diagrama abaixo:



Mas para que é que serve o potenciômetro neste caso? É responsável pelo ajuste do contraste do display. Quanto à luz de fundo, basta ligar à alimentação. No entanto, se pretender ajustar a luminosidade, poderá fazê-lo ligando o pino LED+ a uma saída PWM e programar como pretender. Falaremos disso mais para a frente.

Depois de ligar o Arduino à fonte de alimentação (através do cabo USB) o display deve acender, como demonstrado abaixo:



Ao girar o potenciômetro de contraste, poderá encontrar duas situações: o ecrã normal ou a linha superior preenchida com retângulos. Isso significa algum tipo de danos? Não, é um bom

aspecto. Um display eficiente ligado à corrente deve ser assim. Portanto, ajuste o potenciômetro de tal forma, que os retângulos sejam visíveis.

## Exercício Prático – Exibição de texto no display

De forma semelhante ao caso dos servo motores, também existe uma biblioteca indicada para o funcionamento dos displays. Desta vez, é intitulada **LiquidCrystal**. Vamos começar com um exemplo e depois explicamos o programa. Não faça já o download do código para o seu Arduino!

### Material necessário:

- 1x Arduino UNO e cabo USB;
- 1x Breadboard;
- 1x Display LCD 16x2;
- 1x Potenciômetro;
- 16x Cabos jumper.

```
1 | #include <LiquidCrystal.h> //Biblioteca responsável pelo display
2 | LiquidCrystal lcd(2, 3, 4, 5, 6, 7); //Informações sobre ligação do display
3 |
4 | void setup() {
5 |   lcd.begin(16, 2); //Declaração do tipo de LCD
6 |   lcd.setCursor(0, 0); //Configuração do cursor
7 |   lcd.print("Curso Arduino"); //Exibir texto
8 |   lcd.setCursor(0, 1); //Configuração do cursor
9 |   lcd.print("ROMUAS"); //Exibir texto
10 | }
11 |
12 | void loop() {
13 | }
```

A biblioteca responsável pelo display encontra-se no arquivo: *LiquidCrystal.h*. Para começar a trabalhar o display, deverá declará-lo. A linha usada para isso é a seguinte:

```
1 | LiquidCrystal lcd(2, 3, 4, 5, 6, 7);
```

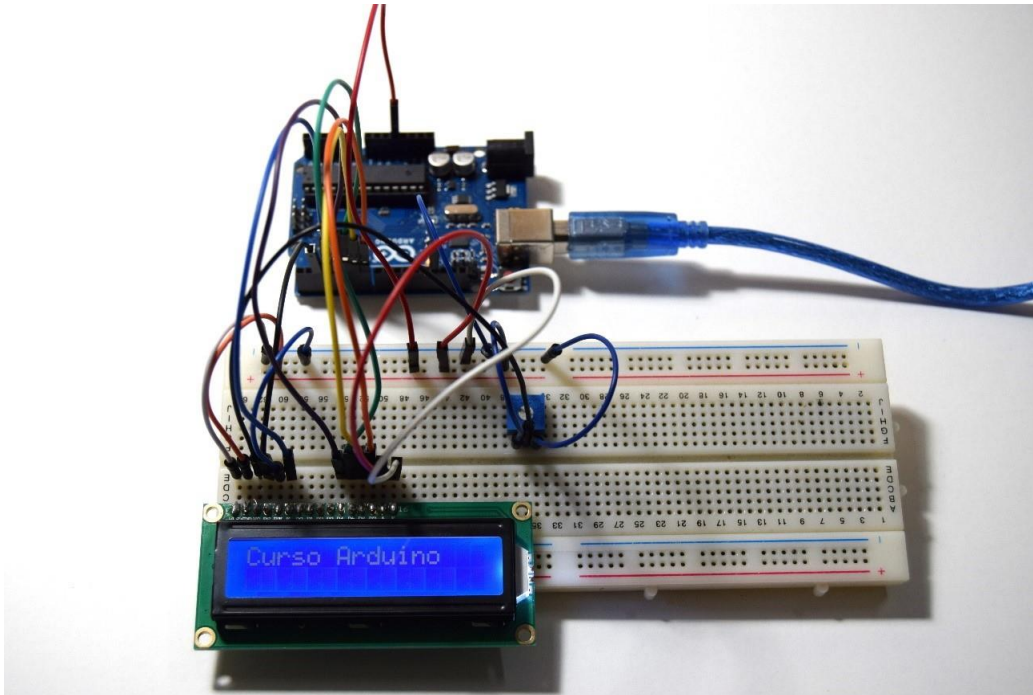
Tal informação indica que o display LCD foi ligado aos pinos de 2 a 7 do Arduino. Mas, é claro que, a seleção de pinos é livre.

A função *lcd.begin* (caracteres, linhas) define o número de caracteres e linhas nos quais o texto será exibido. Neste caso, o display permite a exibição de 16 caracteres em cada uma das duas linhas.

A função *lcd.setCursor* (posição, linha) define o cursor na posição indicada. Por exemplo, um registo (0,0) marca o início, o primeiro carácter da primeira linha. Por sua vez, um registo (0,1) indica o começo da segunda linha.

Neste caso, a função mais importante é a *lcd.print* ("texto"), pois imprime o texto declarado no ecrã. Ao apresentar letras consecutivas, o cursor move-se. É por isso que a chamada seguinte do *lcd.print* começa no lugar onde finalizou o texto anterior.

Agora é hora de fazer o upload do programa Arduino. O resultado deverá ser este:



### E se não funcionar?

Se não conseguir ver o texto no display, verifique os seguintes pontos, resolvem 99,99% dos problemas!

- 1) Alimentação do display;
- 2) Conexão dos pinos no Arduino;
- 3) Contraste.

O último ponto é o mais fácil de verificar, e os iniciantes, muitas vezes, esquecem-se disso! Esperamos que depois de verificar os pontos acima tudo funcione na perfeição. Agora podemos discutir as funções mais importantes relacionadas com o LCD.

### Remover Conteúdos do Display

Ao testar a exibição de textos no display, pode verificar que, às vezes, existem sobreposições. Isto acontece porque o display não limpa automaticamente o seu conteúdo (tem suas vantagens). No entanto, se quiser excluir os conteúdos anteriores, use a **função `lcd.clear ()`**.

### Exercício Prático – Regulação da luz de fundo

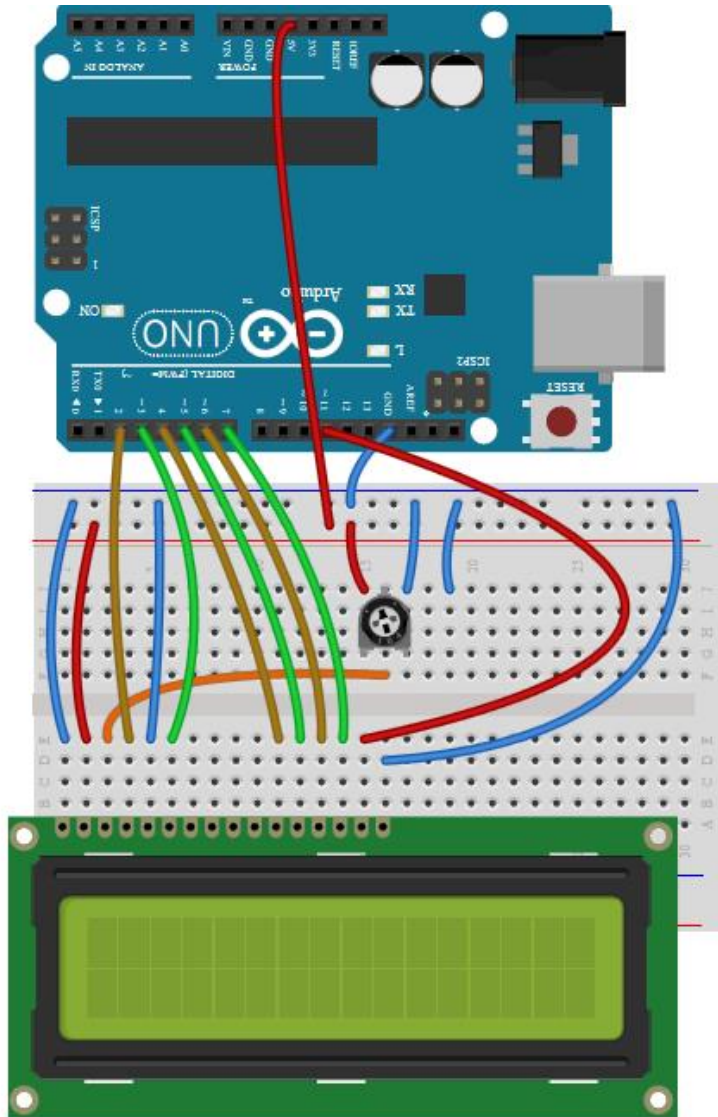
Como já foi referido anteriormente, a luz de fundo, como se trata de um LED, pode ser regulada. Vejamos como isso se faz!

#### Material necessário:

- 1x Arduino UNO e cabo USB;

- 1x Breadboard;
- 1x Display LCD 16x2;
- 1x Potenciômetro;
- 17x Cabos jumper.

Faça as ligações tendo em conta o diagrama seguinte:



Lembra-se das possibilidades do sinal PWM? Verifique como é que o display reage depois de fazer o upload do programa seguinte.

```

1  #include <LiquidCrystal.h> //Biblioteca responsável pelo display
2  LiquidCrystal lcd(2, 3, 4, 5, 6, 7); //Informações sobre ligação do display
3
4  void setup() {
5  lcd.begin(16, 2); //Declaração do tipo de LCD
6  lcd.clear();
7  lcd.setCursor(0,0);
8  lcd.print("Bem-Vindo ao");
9  lcd.setCursor(0,1);
10 lcd.print("Blog ElectroFun");
11 }
12 int brilho = 0;
13 int mudanca = 5;
14
15 void loop()
16 {
17 analogWrite(11, brilho); //Gerar sinal PWM com determinado duty cycle
18 brilho = brilho + mudanca; //Mudar o brilho para valor alterado da variável
19 if (brilho == 0 || brilho == 255) { //Se o duty cycle for 0% ou 100%
20 mudanca = 0-mudanca; //Alterar brilho na direção oposta
21 }
22
23 delay(30); //Atraso para melhor efeito
24 }

```

## Controlo de Motores DC

### Porquê juntar motores ao Arduino?

Controlar o sentido da rotação e a velocidade do motor abre a porta a muitas possibilidades. Com essas novas capacidades, podemos construir um robot móvel simples, que vagueie pela casa e evite obstáculos. Também podemos criar um veículo controlado remotamente através do nosso telemóvel. As possibilidades são, realmente, inúmeras!

### Quais são os motores que vamos utilizar?

Vamos abordar o princípio de controlar os populares motores DC. Estes são frequentemente utilizados em projetos de robótica.



Exemplo de um motor DC.

Ainda é necessário especificar o que são motores DC. De forma geral, os motores DC são dispositivos que convertem energia elétrica em energia mecânica. Neste caso específico, falaremos de motores que consomem, em média, menos de 1A, quando alimentados por 5-9V.

Essas restrições são condicionadas pelo controlador de motor utilizado, mas mais sobre o assunto adiante.

O princípio de controlo de motores é universal: quanto mais potente for o *driver*, maior poderá ser o motor controlado.

## **Porque é que não podemos ligar o motor diretamente ao Arduino?**

O Arduino, mais especificamente o microcontrolador integrado, controla os sinais. A eficiência de cada saída é relativamente pequena (cerca de 20mA). É fácil supor que, 99,999% dos motores que encontrar, vão precisar de muito mais corrente. Posto isto, ao ligar um motor ao Arduino corre o risco de danificar irreversivelmente a placa.

## **Porque é que os exercícios práticos não incluem motores?**

Tal como em tudo, para começar a controlar motores, terá de praticar (através dos exercícios práticos propostos). Como pode ver, nos exercícios propostos, existe um Arduino, uma bateria, um controlador de motor (L293D), condensadores, mas falta alguma coisa... não existem motores DC.

Mas, porquê? A verdade é que, incluir um conjunto de motores seria um desperdício do seu dinheiro. Sendo que se está a iniciar neste mundo, naturalmente não sabe quais projetos irá desenvolver futuramente. Sem saber isto, não poderá escolher os motores adequados. Porquê investir em algo que, se calhar, nunca irá utilizar?

Portanto, nos exercícios seguintes, os motores vão ser substituídos por LEDs. Será na mesma capaz de observar a mudança de direção da rotação (através do LED que acender) e a mudança da velocidade (através do brilho do LED).

Neste artigo precisará dos seguintes materiais:

- 1x Arduino UNO e cabo USB;
- 1x Breadboard;
- 1x Bateria de 9V;
- 1x Ligador para bateria de 9V;
- 1x Chip L293D;
- 1x Resistência de 1K $\Omega$ ;
- 1x Condensador Cerâmico 10-220uF;
- 1x Condensador Eletrolítico 10-220uF;
- 2x LEDs;
- Cabos jumper.

## **Introdução às Pontes H**

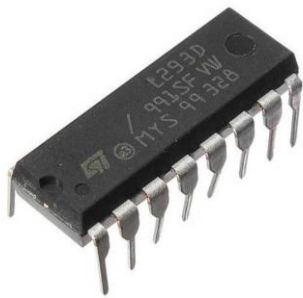
São necessários componentes intermédios entre o Arduino e os motores. Estes são frequentemente denominados pontes H. Estes drivers podem ser construídos a partir de vários transístores ou pode simplesmente usar um circuito integrado ponte H. Como é iniciante, o melhor é recorrer a um chip já pronto a utilizar.

O principal objetivo das pontes H é ler e converter os sinais enviados pelo microcontrolador em tamanhos compatíveis com o controlo do motor. Por exemplo, o Arduino, cujos sinais

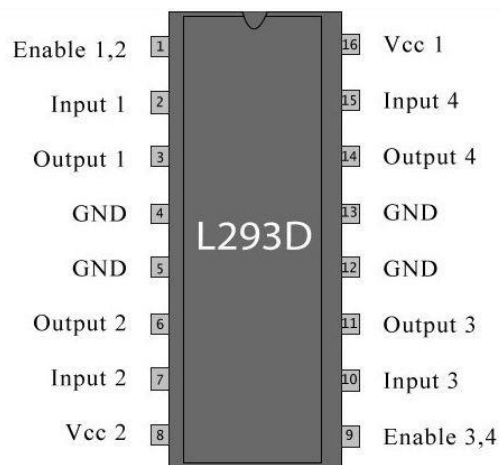
podem funcionar a um máximo de 5V/20mA, após a aplicação da ponte H, pode facilmente controlar um motor que requer 12V/1A para funcionar.

## A sua primeira ponte H – L293D

Vamos utilizar o chip L293D, que apesar de antigo, é barato e muito comum. A sua principal vantagem é o facto de possuir uma montagem THT, ou seja, podemos conectá-lo diretamente à breadboard.



Este sistema possui 16 pinos. Vejamos a legenda de cada um deles:



Uma das principais informações acerca da ponte H que devem ser verificadas é seu desempenho. Especificamente, a corrente que o motor poderá utilizar. O L293D possui uma corrente média por canal de 0,6A-1,2A. O que é que isso significa? Significa que, idealmente, os motores podem consumir até 0.6A, mas se, por um momento, a corrente aumentar para 1.2A, nada de mal irá acontecer. Desde que seja por um período de tempo reduzido!

Vamos agora abordar cada um dos pinos. Primeiro, os **pinos de alimentação**:

- **Terra (GND)** – 4, 5, 12, 13;
- **Alimentação da Parte Lógica 5V (Arduino)** – 16;
- **Alimentação para os Motores até 36V** – 8.

Lembre-se que cada ponte tem uma queda de tensão. Isso significa que, por exemplo, se fornecer uma alimentação de 9V, o L293D gasta parte dela e, os motores recebem, no máximo, 7V. Este sistema é antigo, por isso possui uma grande queda de tensão. Outras pontes mais recentes (por exemplo: TB6612) possuem uma queda menor.

#### Pinos de controlo dos motores:

- Entradas que definem a direção de rotação do 1º motor – 2, 7;
- Entradas que definem a direção de rotação do 2º motor – 10, 15;
- Entrada que define a velocidade do 1º motor – 1;
- Entrada que define a velocidade do 2º motor – 9.

Para parar ou mudar a direção de rotação do motor, deverá definir os sinais de acordo com o diagrama seguinte, denominado **tabela verdade**:

ENABLE	Input 1	Input 2	Motor
PWM	1	0	Esquerda
PWM	0	1	Direita
PWM	0	0	Parar
PWM	1	1	Parar

#### Pinos dos motores:

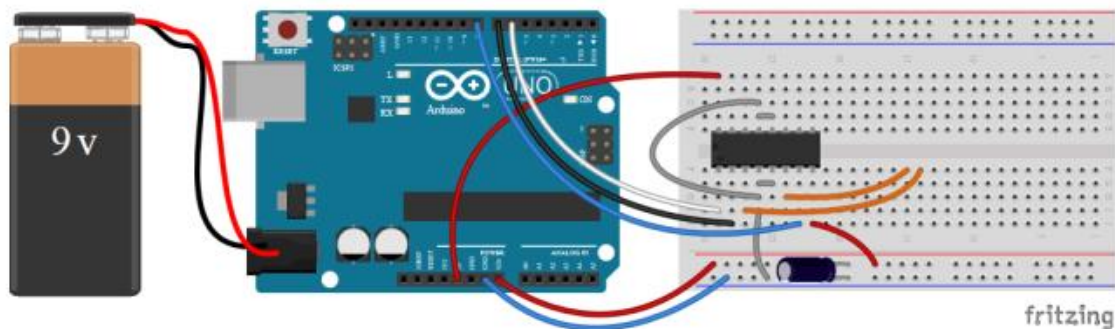
- Saídas do 1º motor – 3, 6;
- Saídas do 2º motor – 11, 14.

## Exercício Prático – Ponte H

Vamos falar da alimentação. Podemos alimentar o sistema com uma ou duas fontes de tensão. Se escolhermos a primeira opção, o Arduino e os motores serão alimentados a partir da mesma fonte – não aconselhamos esta configuração, pois pode levar a interrupções e *resets* frequentes.

Portanto, é mais seguro escolher a segunda opção, na qual o Arduino será alimentado a partir de uma fonte de alimentação separada. No nosso caso, vamos usar alimentação USB e uma bateria de 9V, a partir da qual alimentaremos os motores diretamente (o Arduino também pode ser alimentado dessa forma, mas através do estabilizador integrado).

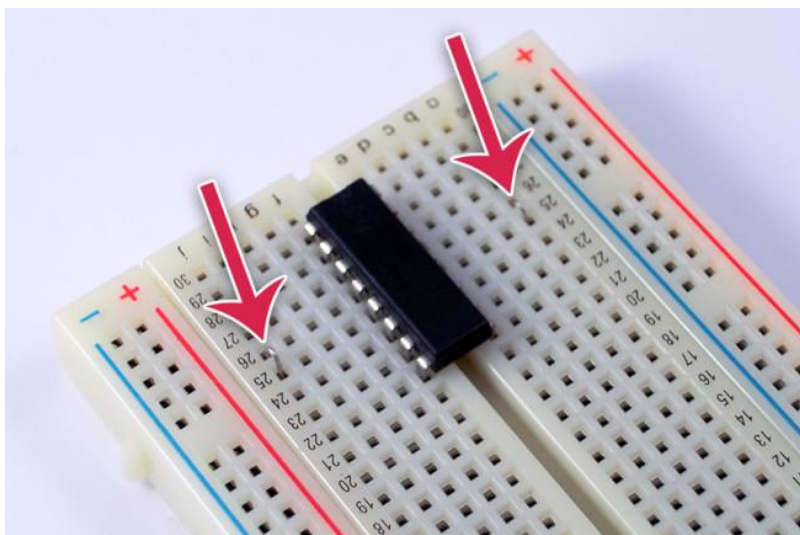
O sistema deve ser ligado de forma semelhante ao diagrama apresentado abaixo. Lembre-se que vamos ligar 9V à placa, e não 5V!



Aos cabos jumper laranja será ligado o “motor”.

Lembre-se de ligar os condensadores, pois são eles que filtram a tensão fornecida pela bateria de 9V.

O sistema é relativamente complicado, por isso, para tornar o esquema mais organizado, e para economizarmos espaço, conectamos os dois pinos GND de cada lado do chip. Isto pode ser feito da seguinte forma, com as pernas cortadas das resistências:

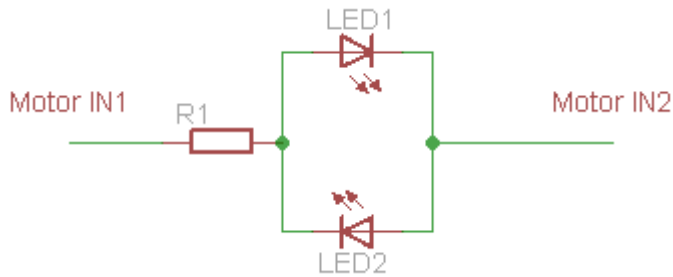


Está na hora de ligarmos o motor. Como foi referido anteriormente, vamos simular um motor com LEDs, para não ter de adquiri-lo. Mas como é que fazemos isso? Muito simples!

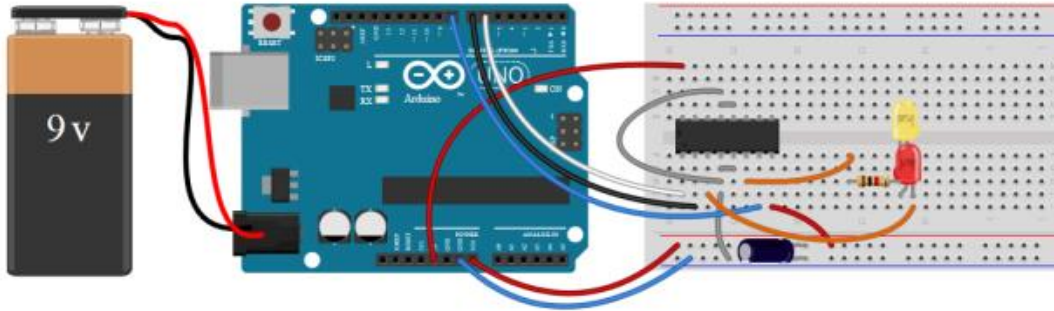
### **Simulação do motor**

Como sabe, um LED é um elemento semiconductor, que ilumina quando a corrente flui através dele na direção certa. Portanto, se ligarmos dois LEDs em paralelo de forma a estarem conectados de forma oposta, poderemos verificar em qual direção a corrente flui – quando um LED acender e, de seguida, o outro. Por sua vez, a velocidade do motor poderá ser verificada através do brilho do LED.

Posto isto, no lugar do motor, incluímos o seguinte circuito:



Assim, o esquema de ligação completo é o seguinte:



## Programação – Controlo da direção de rotação

Vamos agora programar, tendo por base o esquema ligado anteriormente. No início, vamos abordar o controlo da rotação do motor. Deixaremos a regulação da velocidade para mais tarde. Como pode verificar, os pinos do Arduino responsáveis pelo motor são os seguintes:

- 6 (PWM) – regulação da velocidade;
- 7, 8 – controlo do sentido de rotação.

Se não quisermos controlar a velocidade do motor, precisamos de definir o pino 6 como *high*. Também podíamos ligar o cabo diretamente a 5V. No entanto, como já temos as ligações feitas, vamos usar o Arduino:

```

1 void setup() {
2   pinMode(6, OUTPUT); //Sinal PWM do motor para controlo da velocidade
3   digitalWrite(6, HIGH); //Definir permanentemente o estado high no pino 6 -
4   velocidade máxima
5
6   pinMode(7, OUTPUT); //Pino que controla a direção da rotação do motor
7   pinMode(8, OUTPUT);
8 }
9
10 void loop() {
11 //Restante programa
12 }

```

Se o sistema foi ligado corretamente, depois de fazer o upload do programa não deverá acontecer nada! Agora é hora de adicionar o restante código à função *loop*.

Vamos supor que gostaríamos de rodar o motor 3 segundos num sentido e 3 segundos no sentido oposto (velocidade máxima). Para este fim, devemos adicionar uma parte muito simples ao programa:

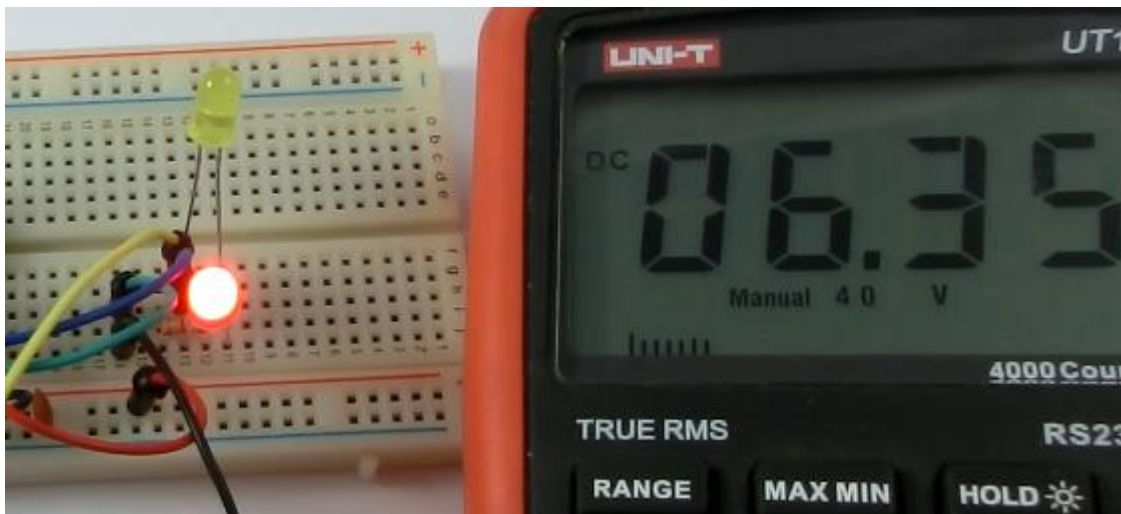
```

1 void setup() {
2   pinMode(6, OUTPUT); //Sinal PWM do motor para controlo da velocidade
3   digitalWrite(6, HIGH); //Definir permanentemente o estado high no pino 6 -
4   velocidade máxima
5
6   pinMode(7, OUTPUT); //Pino que controla a direção da rotação do motor
7   pinMode(8, OUTPUT);
8 }
9
10 void loop() {
11   digitalWrite(7, LOW); //Rodar para a esquerda
12   digitalWrite(8, HIGH);
13   delay(3000); //Durante 3 segundos
14
15   digitalWrite(7, HIGH); //Rodar para a direita
16   digitalWrite(8, LOW);
17   delay(3000); //Durante 3 segundos
18 }

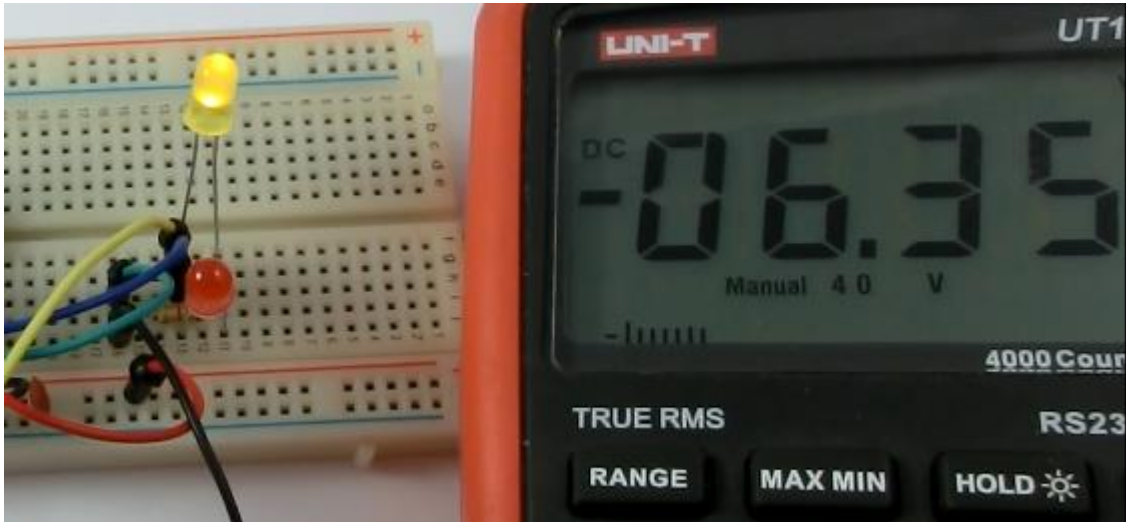
```

De acordo com a **tabela verdade** apresentada anteriormente, sabemos que o sistema funciona se uma das entradas tiver no estado *low* "0" e a outra no estado *high* "1". O sentido da rotação depende do sinal de entrada.

Depois de fazer o upload do programa para o Arduino, os dois LEDs devem começar a piscar alternadamente. Concentre-se e entenda por que é que isso está a acontecer. Os LEDs brilham alternadamente porque, a corrente flui da saída 1 para a saída 2, e vice-versa. O piscar é possível porque estes foram inseridos inversamente. Para facilitar a compreensão, observe as seguintes imagens, em que, para além dos LEDs, também existe um voltímetro:



Motor roda para a direita



Motor roda para a esquerda

Uma vez o resultado é positivo e de outra vez o resultado é negativo. Neste caso, o menos mostra que a corrente flui na direção oposta à inicial.

Como já conseguimos controlar a direção da rotação do motor, é hora de abordarmos a velocidade.

## Aceleração suave do motor

Como já conhecemos o ciclo *for*, podemos usá-lo para acelerar suavemente o motor. É suficiente para alterar o sinal PWM suavemente, por exemplo, a cada 25 ms.

O código é simples e a variável *i* é responsável pelo *duty cycle* do sinal PWM:

```
1 void setup() {
2   pinMode(6, OUTPUT); //Sinal PWM do motor para controlo da velocidade
3
4   pinMode(7, OUTPUT); //Pino que controla a direção da rotação do motor
5   pinMode(8, OUTPUT);
6
7   digitalWrite(7, LOW); //Rodar para a esquerda
8   digitalWrite(8, HIGH);
9 }
10
11 void loop() {
12   for (int i = 0; i <= 255; i++) {
13     analogWrite(6, i); //Aceleração suave do motor
14     delay(25);
15   }
16 }
```

Muitas vezes, os iniciantes têm problemas porque o robot não quer andar. Na maioria das vezes, verifica-se que, ou a fonte de alimentação utilizada é muito fraca, ou o sinal PWM tem pouco *duty cycle*. Se algum dia tiver algum problema, lembre-se de verificar estes dois aspetos!

## Impressão 3D

## O Que É Impressão 3D?

Impressão 3D é o nome dado a uma série de técnicas que **reproduzem objetos em três dimensões**.

Também chamado de prototipagem rápida, o processo permite a impressão de modelos através da fabricação aditiva.

Ou seja, as impressoras formam modelos tridimensionais a partir de uma técnica que **sobre põe finas camadas**, até que o objeto esteja pronto.

Peças de máquinas, objetos de decoração, joias, órgãos e até alimentos podem ser impressos a partir desse processo.

Em geral, os itens são feitos usando alguns tipos de **plástico como matéria-prima**, mas também é possível usar outros materiais, incluindo metal.

A impressão 3D tem revolucionado setores como medicina e engenharia, além de ser bastante útil para quem deseja **empreender** a partir de uma invenção.

Isso porque o processo encurtou o caminho para chegar a uma versão de teste de qualquer produto, chamada **protótipo**.

Anteriormente, era preciso idealizar, desenhar o objeto e desenvolver um projeto para que o protótipo fosse construído.

Com uma impressora 3D, basta desenhar o item em um **software específico** e converter o arquivo para um formato compatível com a máquina para obter um protótipo.

Claro que o uso do software requer certo conhecimento técnico, porém, a dinâmica ficou bem mais simples.

Dependendo do artigo desejado e do processo de impressão 3D, também é necessário **dar acabamento** à peça.

## Como Surgiu A Impressão 3D?

Embora a ideia de imprimir objetos 3D remeta a um cenário futurista, ela não é tão recente.

Em **1984**, a primeira técnica utilizada nesse processo, chamada estereolitografia, foi patenteada a pedido do engenheiro **Charles Hull**.

Ele criou uma máquina capaz de **imprimir lâmpadas especiais**, usadas na solidificação de resinas, além de peças de plástico, a partir da ação de um laser.

A iniciativa teve sucesso, levando Hull a fundar a **3D Systems Corp.** para vender seus serviços inovadores, dois anos depois.

Sempre focada em tecnologias inovadoras, a empresa foi crescendo e agregando **outros tipos de impressão 3D** em seu portfólio, a exemplo da Sinterização a Laser Seletiva (SLS) e sistemas baseados em pó para 3D.

Atualmente, a companhia tornou-se referência na produção de peças tridimensionais personalizadas, especialmente moldes utilizados em cirurgias e outras atividades na **área da saúde**.

Como reconhecimento à sua **contribuição para a sociedade**, o engenheiro Hull foi homenageado com o prêmio European Inventor Award, em 2014.

Além da 3D Systems Corp., vale citar a colaboração de empresas como a Stratasys para o campo da prototipação rápida.

A partir dos anos 1990, elas aperfeiçoaram metodologias e desenvolveram máquinas capazes de **imprimir diferentes modalidades de artefatos**.

No começo, as impressoras comercializadas eram mais robustas, caras e destinadas à produção industrial.

## Para Que Serve A Impressão 3D?

A impressão 3D serve para **construir diversos objetos personalizados**, de **maneira ágil** e relativamente simples.

Após serem prototipadas, as peças se tornam reais a partir desse processo, incluindo detalhes como fendas, formas e cores.

Por seu **caráter versátil**, a prototipagem rápida serve aos mais variados propósitos.

O mais comum é a impressão de um artigo único, desenhado para uso de uma pessoa ou um pequeno grupo.

É o caso de itens de papelaria, como porta canetas, e decorativos, como molduras diferenciadas, luminárias e chaveiros.

Mas o processo vem ganhando **popularidade na indústria**, onde auxilia na substituição de peças, que são produzidas sob demanda e a um custo menor do que se fossem fabricadas por meio da dinâmica tradicional.

Dependendo do material utilizado, até mesmo **móveis, tatuagens e próteses** podem ser impressos e realizar, com sucesso, as funções das estruturas que os inspiraram.

## Como Funciona A Impressão 3D?

Assim como uma impressora comum, que libera jatos de tinta em quantidade e formatos específicos para formar imagens e letras, a impressora 3D **injeta diferentes materiais** para compor um objeto.

A diferença é que, em vez do papel, é em uma bandeja que as máquinas modernas imprimem os itens.

Tudo começa com o **desenho da peça** em um software de modelagem em 3D, a exemplo do **AutoCad e Blender**.

Existem tipos variados de software, geralmente voltados para o setor do produto que será impresso – construção civil, papelaria, medicina e assim por diante.

Em seguida, com o desenho em mãos, a **matéria-prima é inserida** na impressora, que é ligada e começa a expelir a primeira camada na bandeja.

Ou seja, em vez de partir de uma grande peça que será cortada e moldada, a dinâmica monta os artigos unindo pequenas partes e camadas de material.

O processo sempre se inicia **de baixo para cima**, com a impressão de uma camada por vez, podendo durar de alguns minutos até dias para ser finalizado.

Apesar de seguirem um padrão semelhante, existem diferentes tipos de impressão em três dimensões, conforme detalhamos a seguir.

## **Tipos De Impressora 3D**

Há **modelos maiores e menores**, destinados à produção de pequenas peças até paredes e pequenos edifícios.

Elas utilizam **técnicas próprias de impressão**, dependendo de seus recursos e da matéria-prima com que trabalham.

Conheça, abaixo, as metodologias mais populares.

### *Modelagem Por Fusão E Depósito – FDM*

Também chamada de fabricação com filamento fundido (FFF), essa é uma das técnicas mais comumente utilizadas, devido ao seu baixo custo e relativa simplicidade.

Esse processo **usa fios de plástico** para imprimir os objetos – o que explica o nome da técnica.

Os filamentos são aquecidos, enquanto um cabeçote se movimenta por dois eixos diferentes e injeta o material em uma bandeja.

Em seguida, ocorre a fusão do plástico até formar o molde solicitado.

As impressoras que utilizam a modelagem por fusão e depósito têm **motores que exigem menor potência** e resfriamento para serem acionados, quando comparados aos lasers usados em outras modalidades de impressão 3D.

Por isso, máquinas que imprimem por meio da FDM podem ser instaladas fora de ambientes industriais, sendo empregadas principalmente para **fins acadêmicos** e impressão de protótipos que serão customizados.

Afinal, a técnica não permite muita variação na forma do objeto, pois tem um acabamento mais simples, adequado para a composição de itens menos sofisticados.

### *Estereolitografia*

Nos tópicos anteriores, contamos que a modalidade pioneira foi a estereolitografia, criada por Chuck Hull na década de 1980.

Essa técnica **emprega resina líquida** (acrílica, epóxi ou vinil) para compor cada detalhe da peça desejada.

A resina é endurecida por um feixe de laser ultravioleta, que é emitido pela impressora 3D para formar cada camada do artefato.

Por fim, o excesso de resina líquida deve ser removido e a peça **levada a um forno** para receber o acabamento necessário.

Esse método permite a impressão de objetos mais resistentes e complexos que a técnica FDM, no entanto, a um tempo e custo maiores.

#### *Sinterização Seletiva A Laser – SLS*

Esse processo é parecido com a estereolitografia, mas emprega **matéria-prima em pó** para formar as camadas do modelo tridimensional.

A SLS representa grande progresso para a prototipagem rápida, pois permite o uso de diferentes materiais na impressão, gerando peças feitas de poliamidas, elastômeros, cerâmicas e metais.

O processo implica no **uso de uma impressora robusta**, que possui uma câmara de impressão, na qual o pó é injetado, nivelado e atingido por um laser de alta potência, que o aquece e faz a fusão.

Depois de finalizar a primeira camada, o rolo da impressora passa sobre ela para que seja coberta com mais pó, que será **aquecido e fundido** até que o objeto esteja pronto.

Em seguida, um jato de ar comprimido e escovas retiram o excesso de pó do artefato.

#### *Sinterização Direta De Metal A Laser*

Conforme o nome sugere, a Sinterização Direta de Metal a Laser funciona a partir da combinação de **partículas feitas de metal**.

Assim como na SLS, um laser é aplicado para aquecer e unir o material, resultando em produtos exclusivos.

Uma das vantagens dessa técnica é a produção de itens com formato complexo, que normalmente não seriam fabricados na indústria.

#### *Polyjet*

A polyjet é a metodologia mais parecida com a impressão a jato de tinta, sobrepondo as camadas com perfeição e agilidade.

Usando um **tipo de plástico** (o fotopolímero) em forma líquida, esse processo resulta numa reprodução fiel do projeto.

## **Quais Os Principais Materiais Utilizados Na Impressão 3D?**

Listamos, a seguir, os principais materiais para a impressão de modelos tridimensionais:

- Papel
- Plástico
- Metais
- Borracha
- Vidro
- Resinas
- Cerâmica.

Os plásticos sintéticos são a **matéria-prima mais comum**, principalmente para impressoras que utilizam filamentos para compor as camadas dos objetos.

Ácido polilático biodegradável, acrilonitrila butadieno estireno e poliamidas (como o nylon) são exemplos de plásticos popularmente usados na impressão 3D.

## O que é o Tinkercad?



O Tinkercad é uma coleção de ferramentas de softwares, online e gratuita, para quem deseja criar modelos em 3D de forma bastante simples e intuitiva.

Utilizando essa ferramenta, tem acesso a diversas formas geométricas para construir o seu projeto. O slogan do Tinkercad traduz muito bem sua funcionalidade: “da mente ao projeto em minutos”.

Para utilizar a ferramenta basta criar uma conta gratuita em <https://www.tinkercad.com/>

### Quais são as principais ferramentas do Tinkercad?

Embora o Tinkercad seja perfeito para iniciantes, isso não significa que aqueles que têm mais experiência com modelagem 3D não gostarão desse software.

A ideia principal da ferramenta é que você crie modelos mais complexos a partir da composição de várias formas simples.

Além disso, o software permite adicionar circuitos eletrônicos aos projetos 3D para criar objetos com luz e movimento. O resultado final pode até ser simulado no software para verificar como os componentes responderão na vida real.

O Tinkercad disponibiliza formas como esferas, cilindros, caixas, cones, textos, números e conectores que podem compor o seu projeto.

Usando essas formas você poderá **agrupar**, **duplicar**, **desagrupar**, **alinhar** ou **espelhar** até conseguir chegar em um resultado satisfatório para a sua modelagem.

E para imprimir o modelo construído no Tinkercad, a boa notícia é que você conseguirá exportar seu arquivo em formato **STL** ou **OBJ** para poder realizar as configurações de pré-impressão no fatiador de sua preferência.

## É preciso fazer o download do TinkerCad?

Uma das principais vantagens do Tinkercad é ser totalmente online, ou seja, você não precisa fazer o download para utilizá-lo. No entanto, para utilizá-lo é necessário registrar-se. Depois de criar a conta de forma gratuita, basta começar a modelar!

## Como dar os primeiros passos no software?

Para começar, acesse o site do Tinkercad <https://www.tinkercad.com/>. Clique em “Inscrever-se” no canto superior direito (apenas se não tiver uma conta criada, caso contrário basta realizar o login clicando em “Entrar”).



Na tela de registro escolherá o país e colocará sua data de nascimento. Então clique em “Avançar”. Digite seu e-mail e a senha escolhida. Aceite os termos de serviço e privacidade e clique em “Criar conta”.



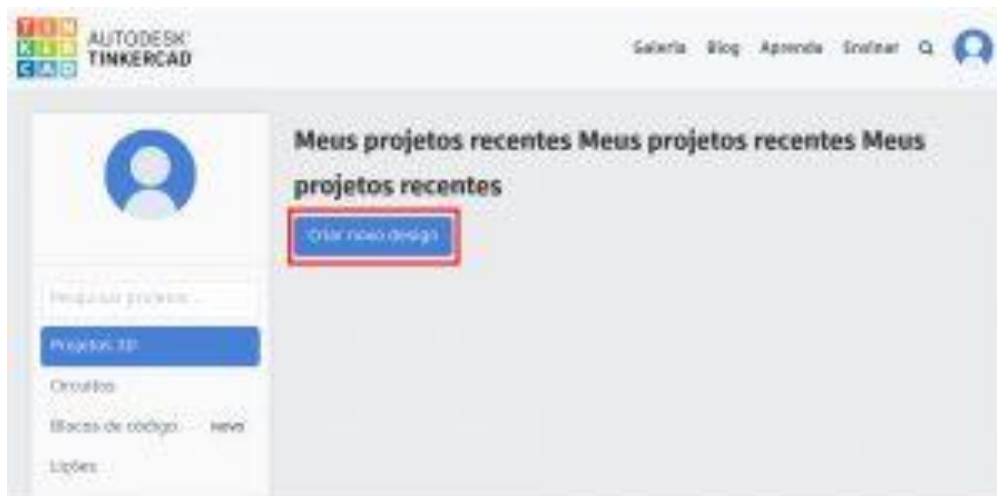
Com a conta criada o aplicativo abrirá automaticamente a tela inicial com o tutorial de primeiros passos. Caso queira pular essa etapa basta clicar no “X” do lado direito.



### Quais são os recursos do software?

O Tinkercad utiliza o mesmo princípio do Lego. Nele, você trabalhará principalmente com formas pré-definidas e estruturas geométricas. Você pode adicionar ou subtrair peças para criar furos ou objetos ocos.

O primeiro passo para começar o seu projeto é clicar em “Criar novo design”.



A tela de trabalho se abrirá como na imagem abaixo:



1. O retângulo indicado com o número 1 exibe os recursos de “Copiar” (Ctrl+C), “Colar” (Ctrl+V), “Duplicar” (Ctrl+D), “Excluir” (Delete), “Desfazer” (Ctrl+Z) e “Refazer” (Ctrl+Y).
2. O recurso indicado pelo retângulo vermelho de número 2 é utilizado para movimentação do plano de trabalho. O plano de trabalho também pode ser movimentado ao ser selecionado com o botão direito do mouse. Para mover em diferentes direções, mantenha pressionado o botão de seleção.
3. O retângulo de número 3 contém as funções mais importantes, que são “Mostrar tudo” (Ctrl+Shift+H), “Agrupar” (Ctrl+G), “Desagrupar” (Ctrl+Shift+G), “Alinhar” (L) e “Virar” (M);
4. O retângulo vermelho indicado com o número 4 refere-se às listas de formas, textos e demais recursos que o aplicativo tem a oferecer.

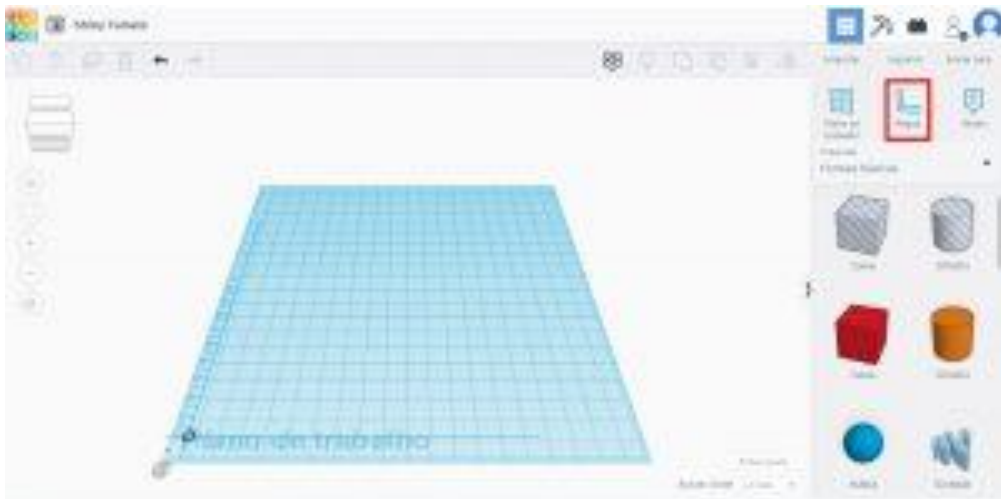
### **Como começar a modelar com o Tinkercad?**

Chegou a hora de iniciar os seus projetos no Tinkercad! Como exemplo vamos mostrar um modelo de identificador de bagagem. Mas pode explorar a criatividade, uma vez que o aplicativo lhe permite isso.

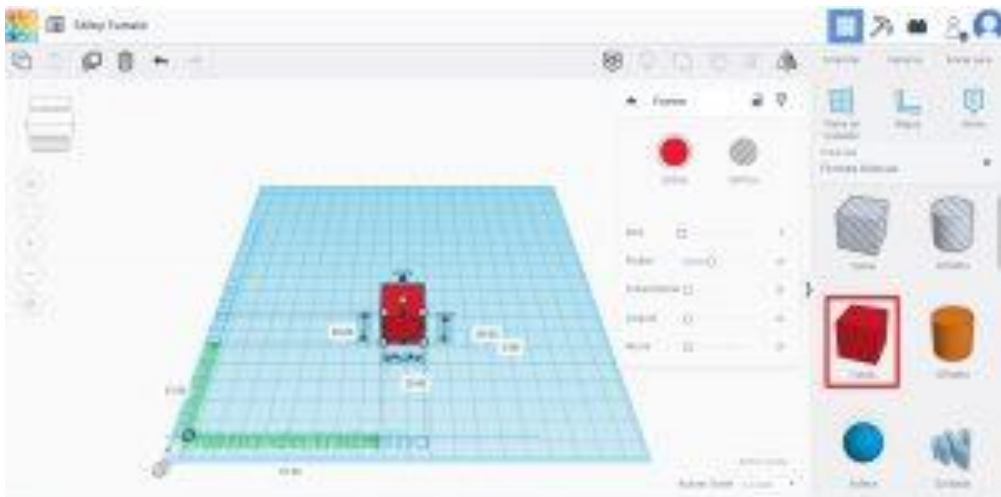


#### **Passo 1: utilizando a forma “Caixa”**

Ao iniciar o seu projeto, o primeiro (e essencial) passo é adicionar a ferramenta “Régua” localizada no canto direito superior no seu plano de trabalho. Dessa forma, você sempre terá uma referência de medidas.

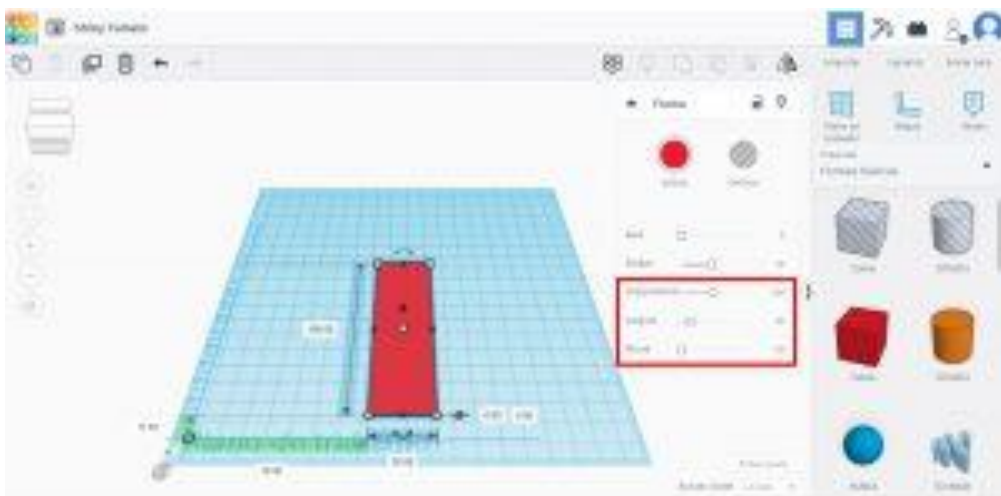


Em sequência, arraste a forma “Caixa” para iniciar a sua modelagem.

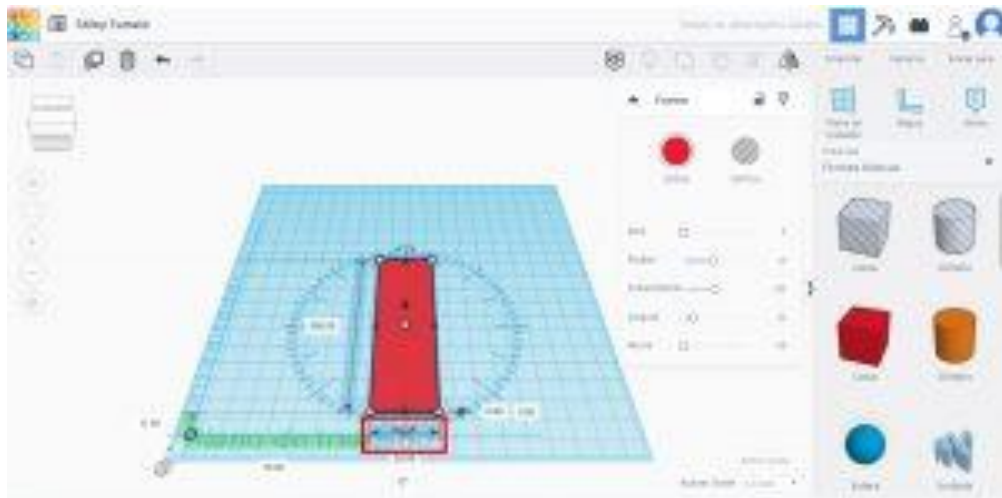


Clique sobre o objeto e altere no próprio menu da forma. Ou, se preferir, edite as dimensões desejadas sobre algum dos quadrados brancos que aparecerão ao redor da figura.

No nosso identificador de bagagem foi utilizado 100 mm de comprimento por 30 mm de largura. Na altura ele passou para 0,8 mm.

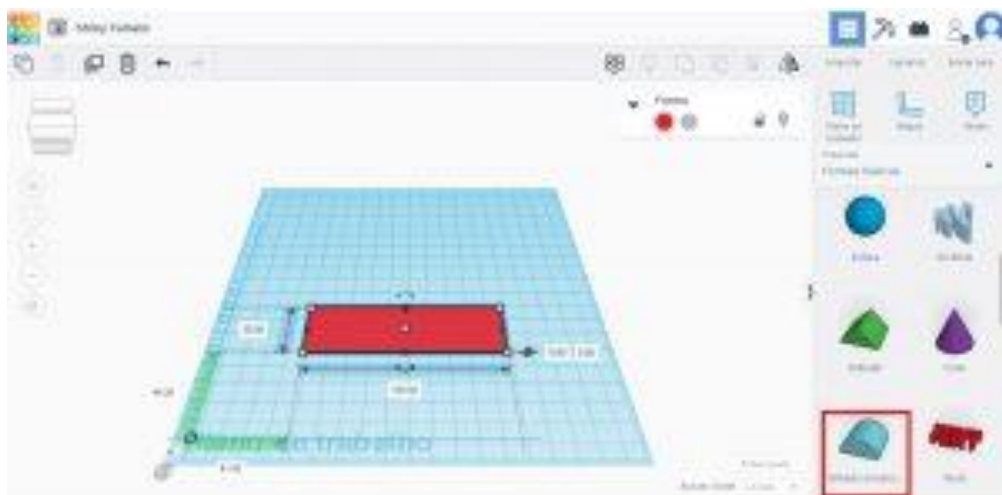


Para girar a peça sobre o plano de trabalho, basta clicar em cima da figura e mexer na rotação da mesma.

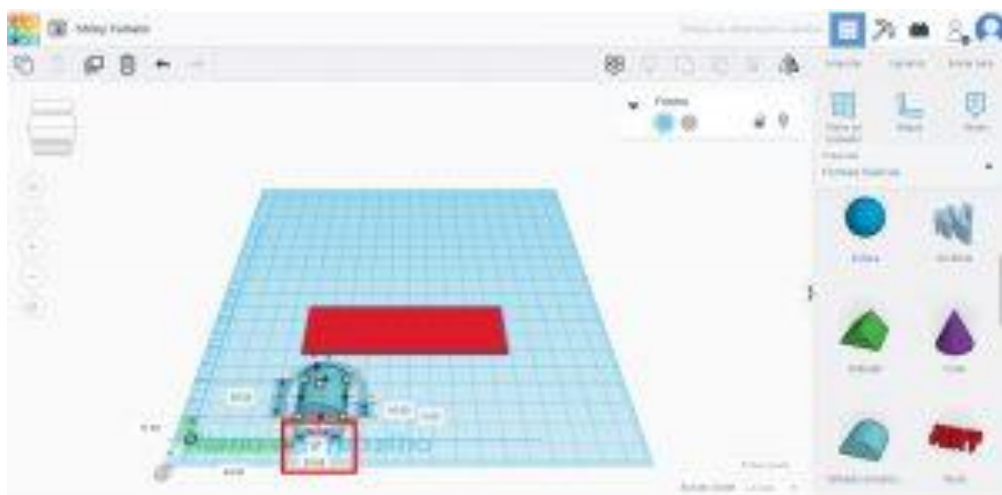


### Passo 2: utilizando a forma “telhado arredondado”

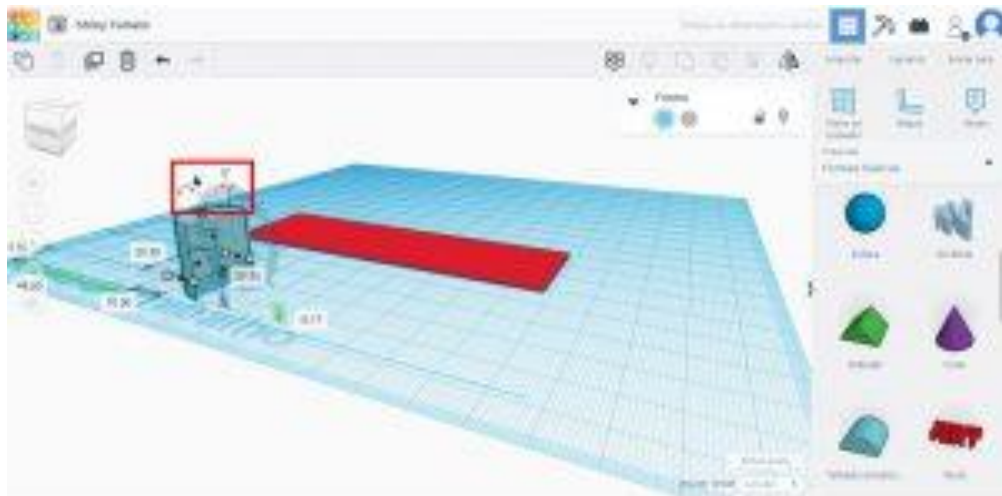
Na curva lateral da peça você pode utilizar duas formas diferentes: o cilindro (colocando metade dele para dentro da peça retangular), ou o telhado arredondado.



Aqui selecionamos o telhado arredondado, girando-o 90° no plano XY.



Para girar 90º no plano Z, movimente o plano de mesa. Dessa forma torna-se mais fácil visualizar.

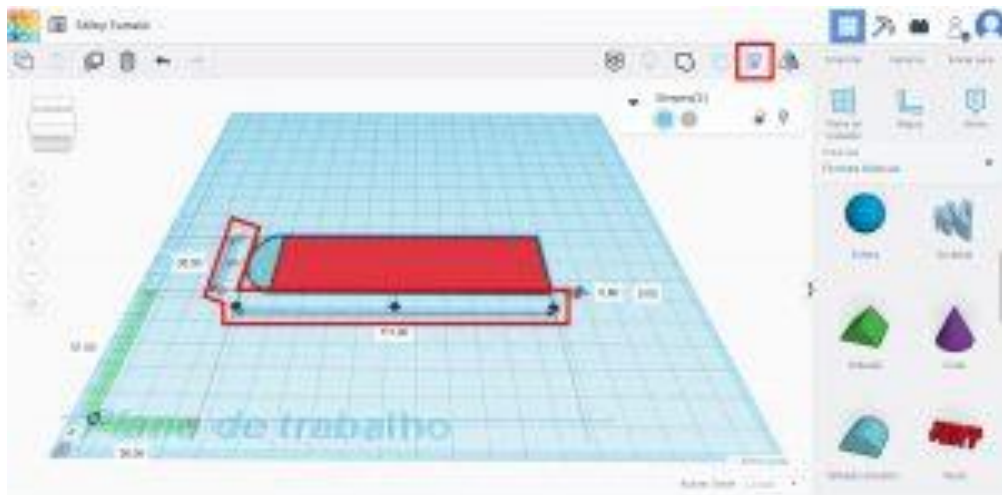


Em seguida, altere as dimensões do telhado para 10 mm de comprimento, 30 mm de largura e 0,8 mm de altura. Para tanto, clique nos quadrados brancos como foi mostrado no passo anterior.

### Passo 3: alinhando as partes

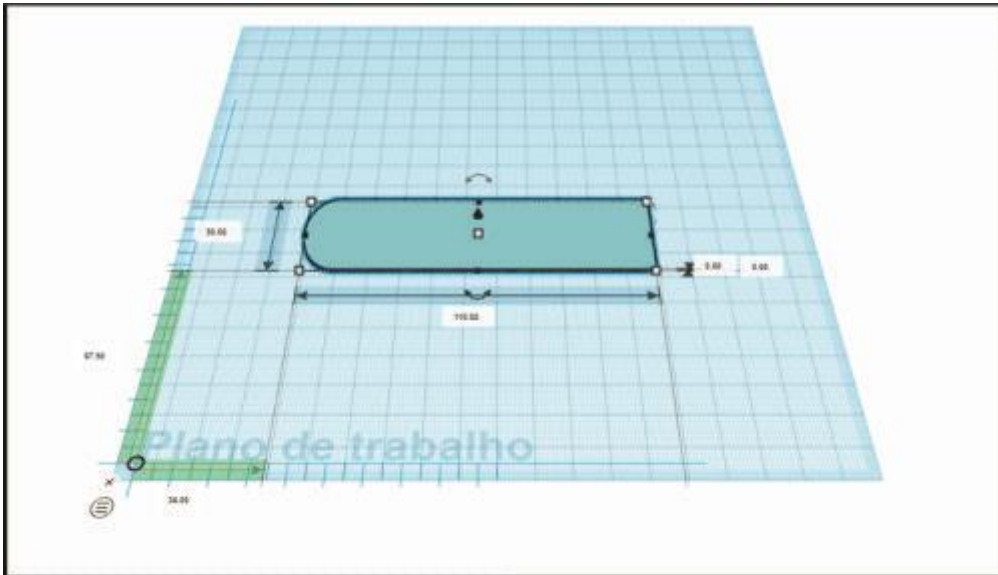
Encoste as duas peças que você criou. Para alinhá-las selecione uma delas, segure “Shift” e clique sobre a outra peça. Em seguida, no canto superior direito, selecione a ferramenta “Alinhar”.

Aparecerão pontos pretos para você escolher o tipo de alinhamento. No nosso caso, deixamos as peças centralizadas.



### Passo 4: agrupando as partes

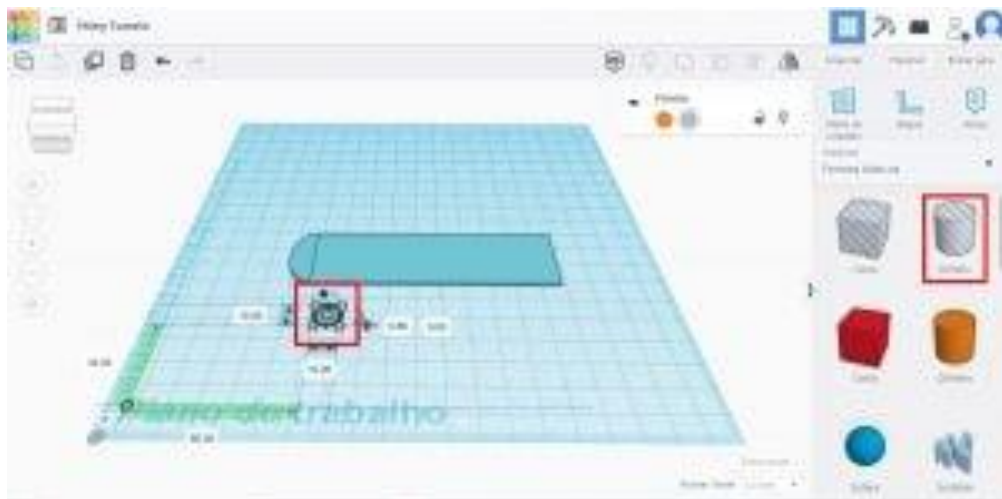
Agora iremos agrupá-las! Para isso basta manter as partes selecionadas e, no canto superior direito, clicar na opção “Agrupar”. A partir de agora as duas peças passarão a ser apenas uma.



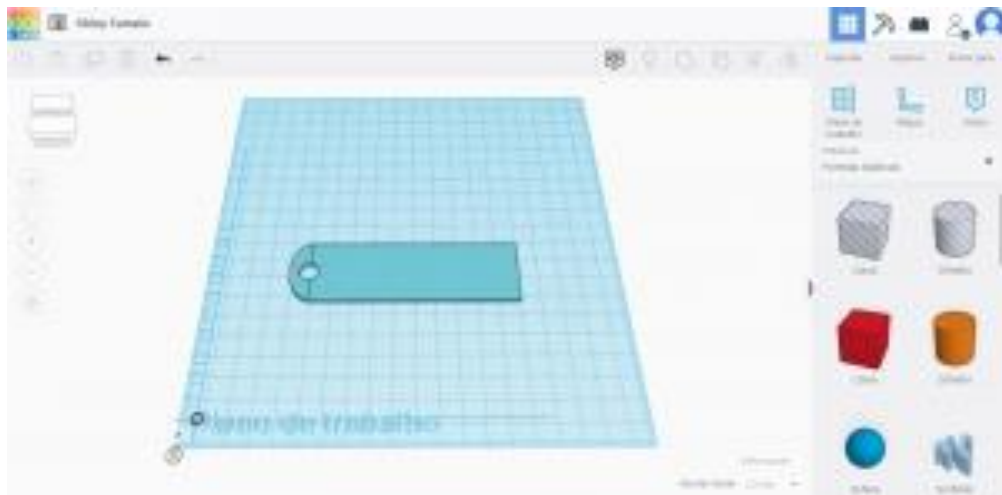
### Passo 5: fazendo um furo na peça

Existem duas formas de fazer um furo cilíndrico. A primeira é selecionando o cilindro cinza (que inclusive representa um orifício). Já a segunda alternativa é selecionar o cilindro sólido e alterá-lo para orifício. Em ambos os casos, arraste o cilindro por sobre a peça, altere as suas dimensões e alinhe as duas peças.

No identificador de bagagem 3D Lab, a dimensão usada foi de 10 mm de diâmetro.

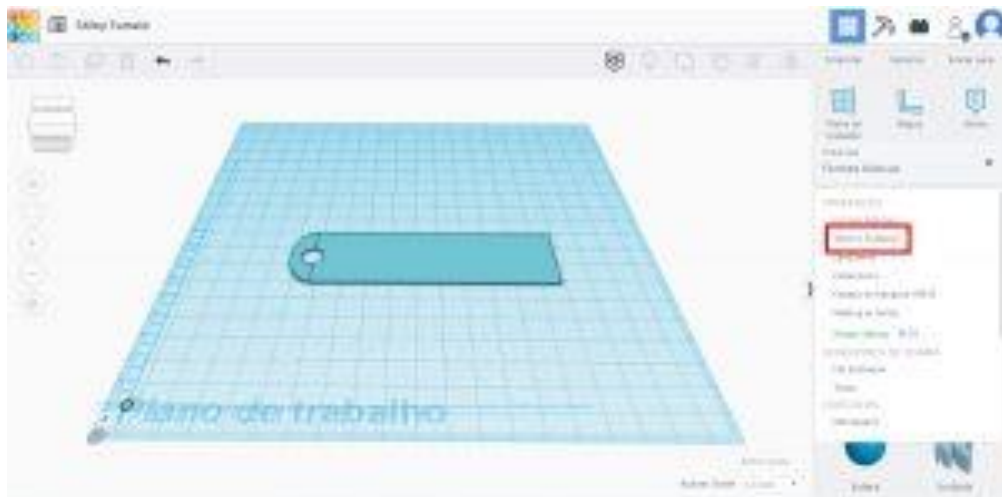


Em seguida, selecione as duas formas e clique "Agrupar". Aonde estava o cilindro será feito um furo na peça como mostrado abaixo.

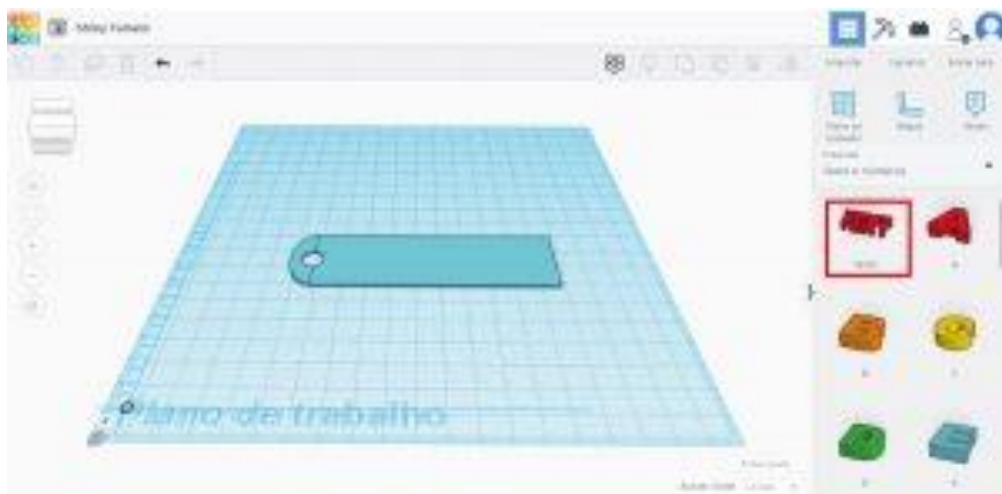


### Passo 6: acrescentando o texto

Do lado direito selecione “Texto e Números”.

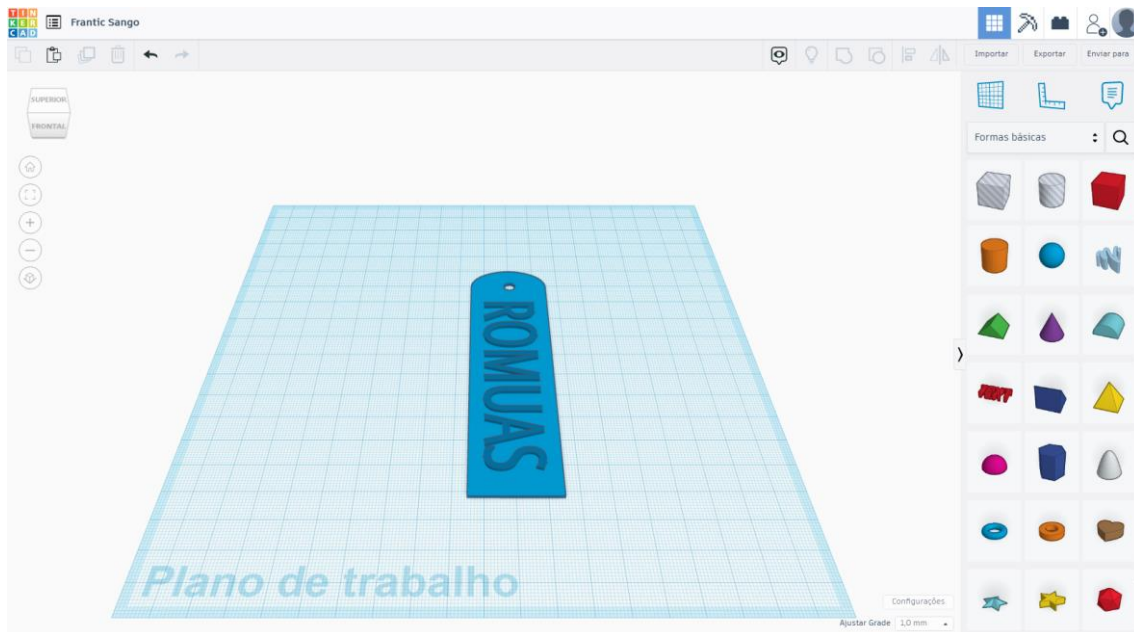


Então arraste a ferramenta “Texto” para dentro da peça. Neste momento uma aba lateral denominada “Forma” irá se abrir, e nela você pode alterar o “Texto” e a “Fonte”.

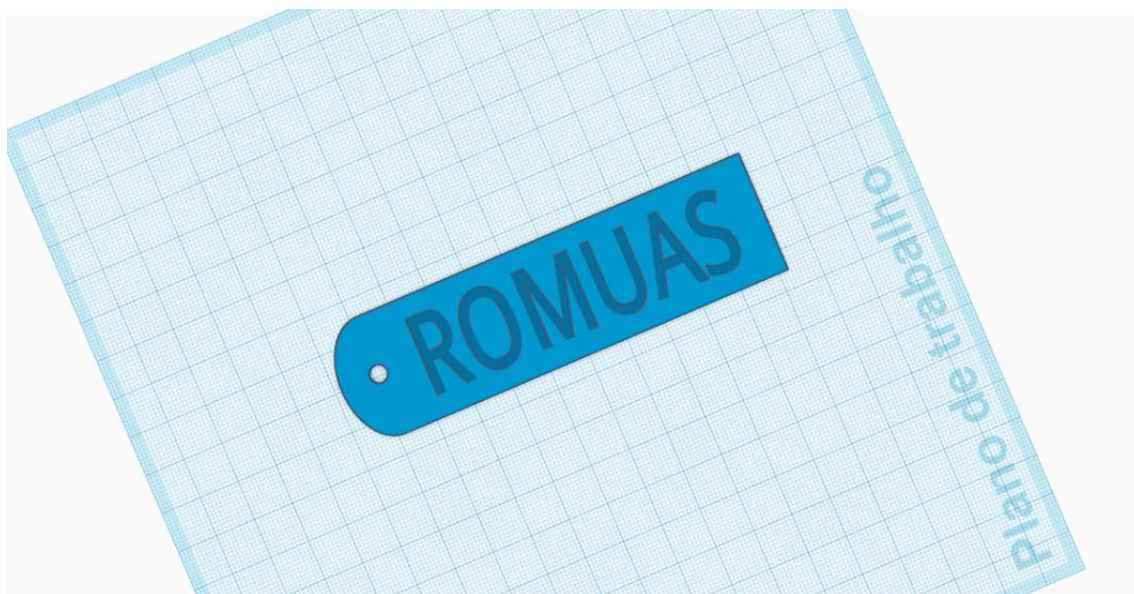


Escolhemos para o nosso exemplo a fonte Sans. As dimensões do texto foram 80 mm de comprimento e 20 mm de largura.

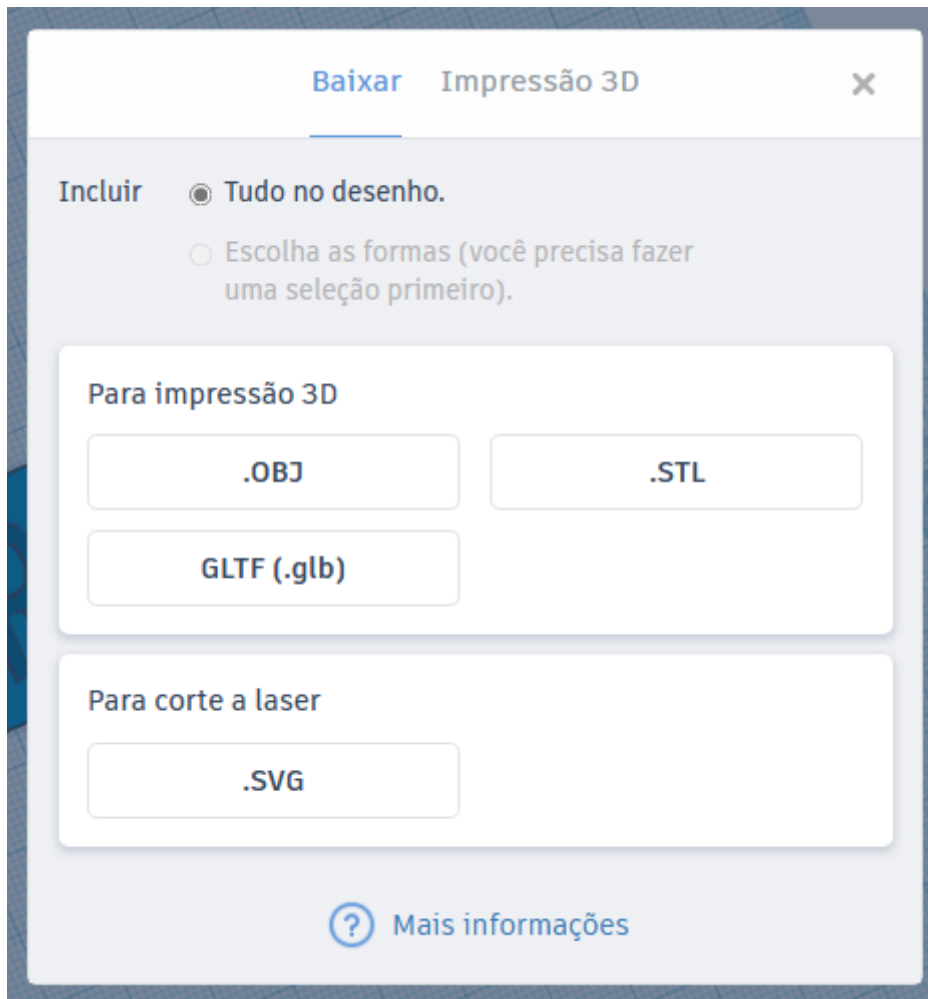
A altura utilizada para o texto foi de 2,0 mm a partir do plano de trabalho, uma vez que a peça e o texto estavam encostados na mesa.



Agora agrupe todas os componentes que você criou.



Feito isso o seu modelo está finalizado! Para fatiar e imprimir, basta exportar como STL através da ferramenta "Exportar". Ela está localizada no canto superior direito.



### Passo 7: imprimindo

Por fim, após ter o seu projeto salvo em STL abra o arquivo em um software de fatiamento de sua preferência. Selecione as configurações de impressão e coloque para imprimir na sua impressora 3D.

