Workshop Arduino

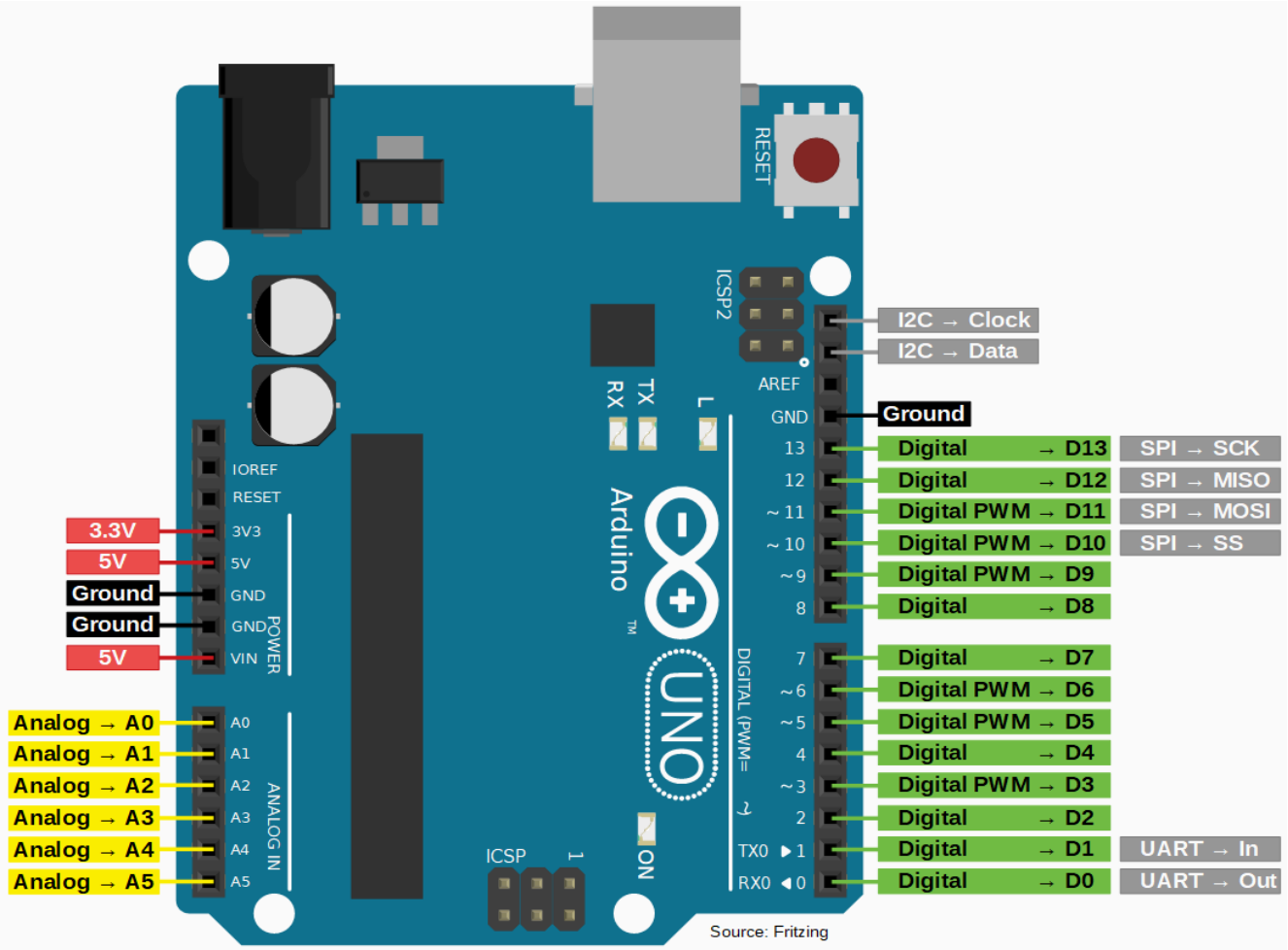
NEEEC-FEUP

Prática

Índice

NEEEC-FEUP

1- Pinos do Arduino



Sinal Digital

Um sinal digital no Arduino é um sinal binário, ou seja, possui apenas dois estados possíveis. High (1) e Low (0).

Os sinais digitais são utilizados para tarefas simples como por exemplo ligar ou desligar LEDs, ativar *relés* (dispositivo eletromecânico que funciona como um interruptor controlado por eletricidade) e ler informações de sensores digitais.

NEEEC-FEUP

Sinal Analógico

Um sinal analógico, ao contrário do sinal digital, não possui apenas 2 estados. Por ter um vasta gama de valores possíveis [0; 1023] *(inteiros)* são usados para tarefas mais complexas como por exemplo leitura de sinais de temperatura e controle de tensões através do uso de potenciômetros.

PWM

PWM (Pulse Width Modulation / Modulação por Largura de Pulso) é uma técnica utilizada para simular tensões intermediárias em saídas digitais, variando a largura de pulsos. Isto mostra-se útil quando estamos a falar de controlo de intensidade. Ao variar a largura dos pulsos, é possível ajustar a quantidade de energia que é entregue ao dispositivo, como por exemplo regular a intensidade de dispositivos LED.

Alimentação

Existem várias maneiras de alimentar um Arduino:

- **USB**: A porta USB é uma opção conveniente para alimentar e comunicar com o Arduino a partir de um computador.
- **Bateria**: Baterias recarregáveis ou pilhas podem ser usadas para tornar o Arduino portátil.
- **Transformador**: Geralmente entre 7-12V, podem ser usados através do conector de alimentação.
- **Pino Vin**: Podes alimentar o arduino aplicando 5V no pino Vin e o ground num pino GND do arduino.

É crucial fornecer a tensão e corrente corretas ao Arduino para evitar danos, tanto no arduino como no próprio PC.

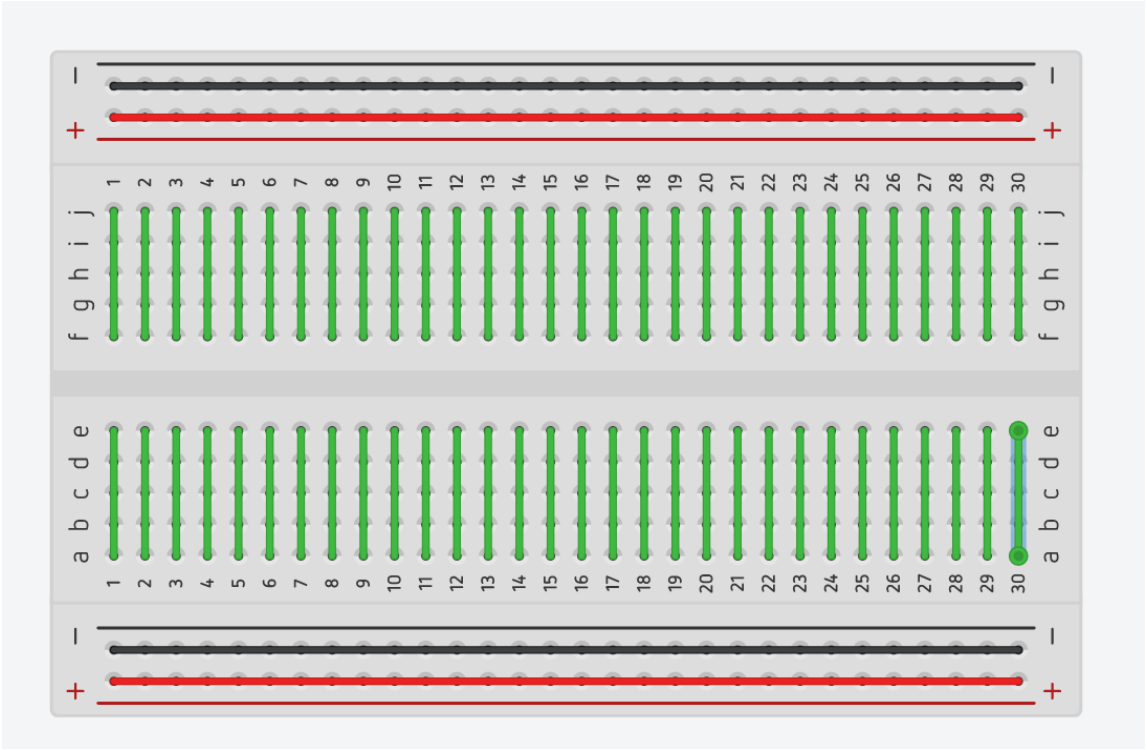
A tensão correta pode ser verificada consoante as especificações do Arduino para garantir que a tensão de entrada seja adequada. Para o Arduino funcionar tem que ter uma corrente mínima.

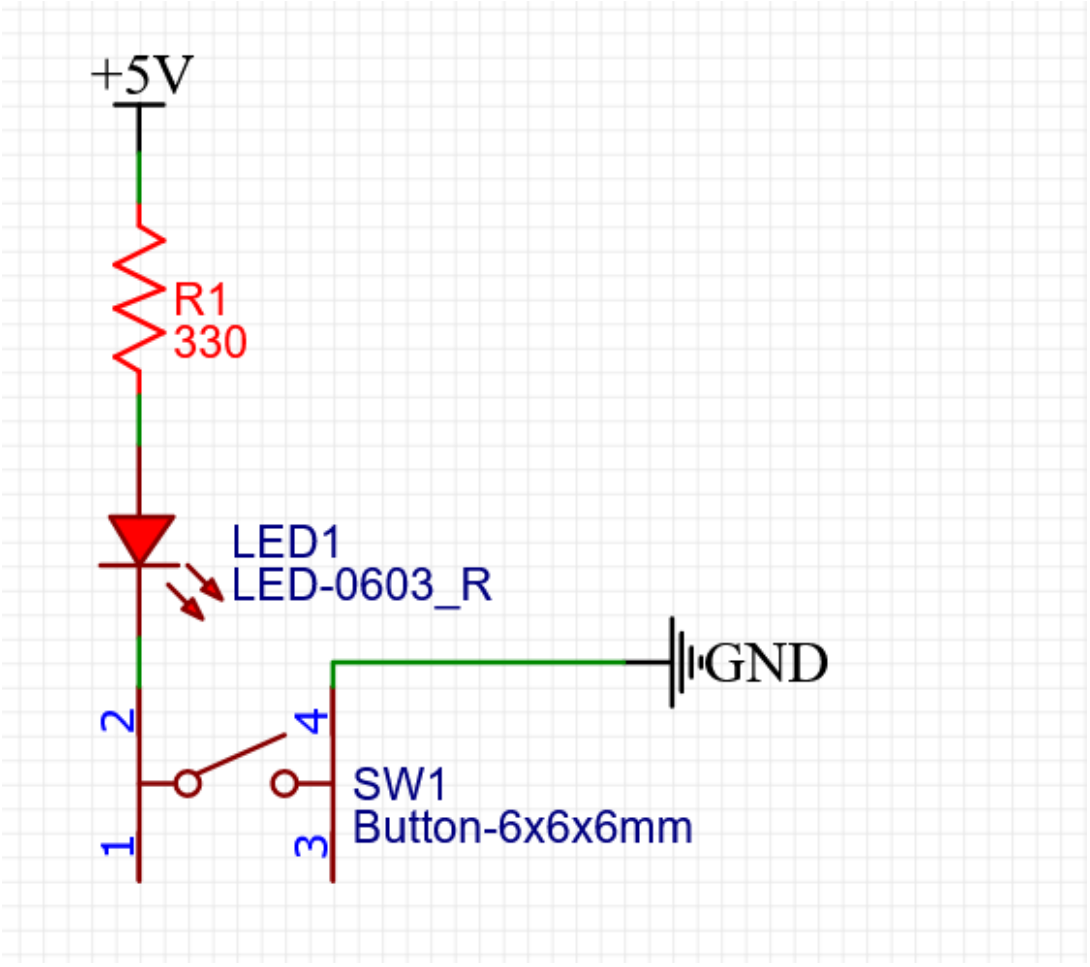
AVISOS!

Polaridade Correta: Conecta a polaridade da alimentação adequadamente para evitar danos.

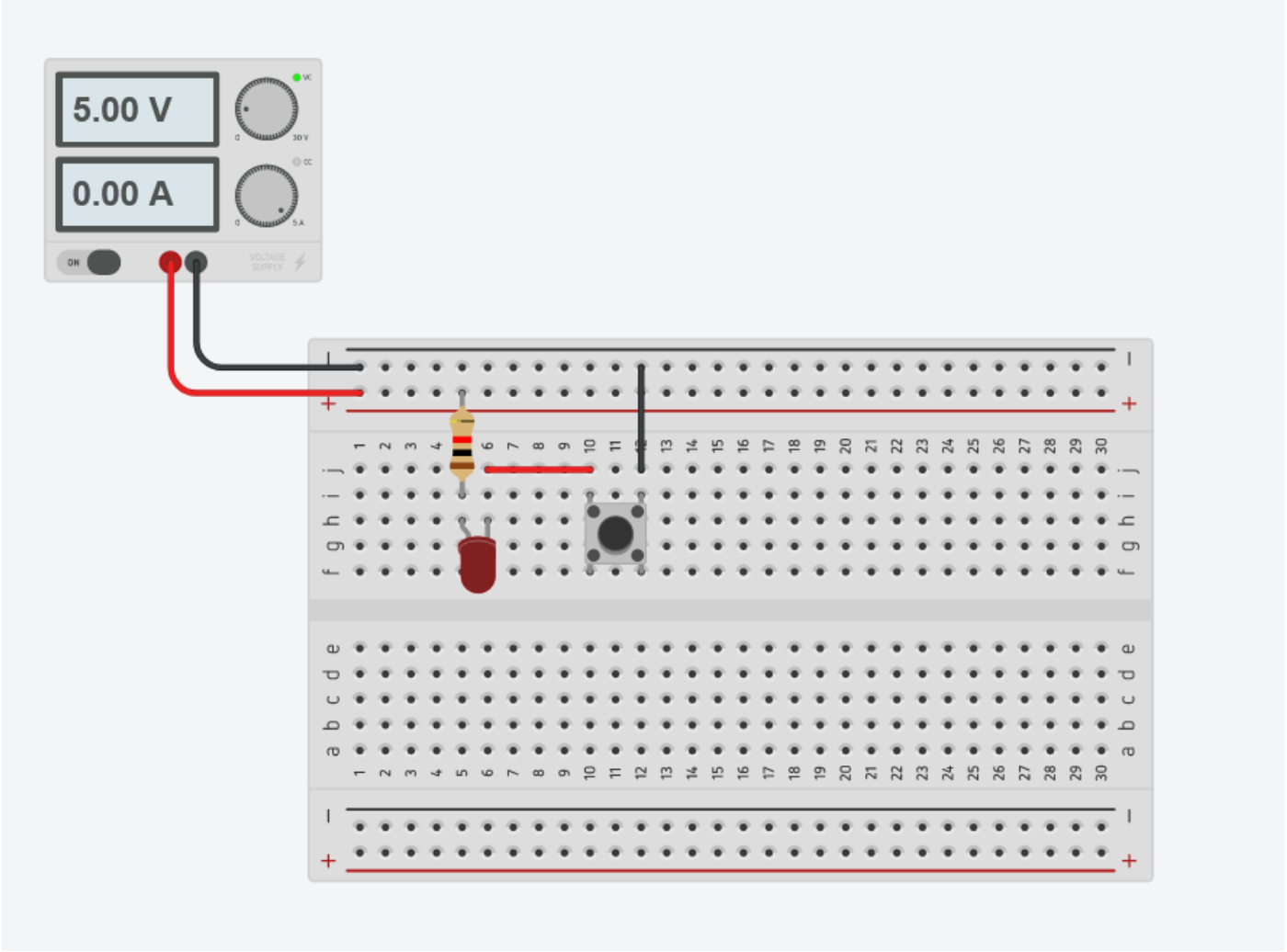
Sobrecarga das Portas USB: Evita sobrecarregar as portas USB do PC, pois cada porta USB tem uma capacidade máxima de corrente.

2-BreadBoards e Esquemático





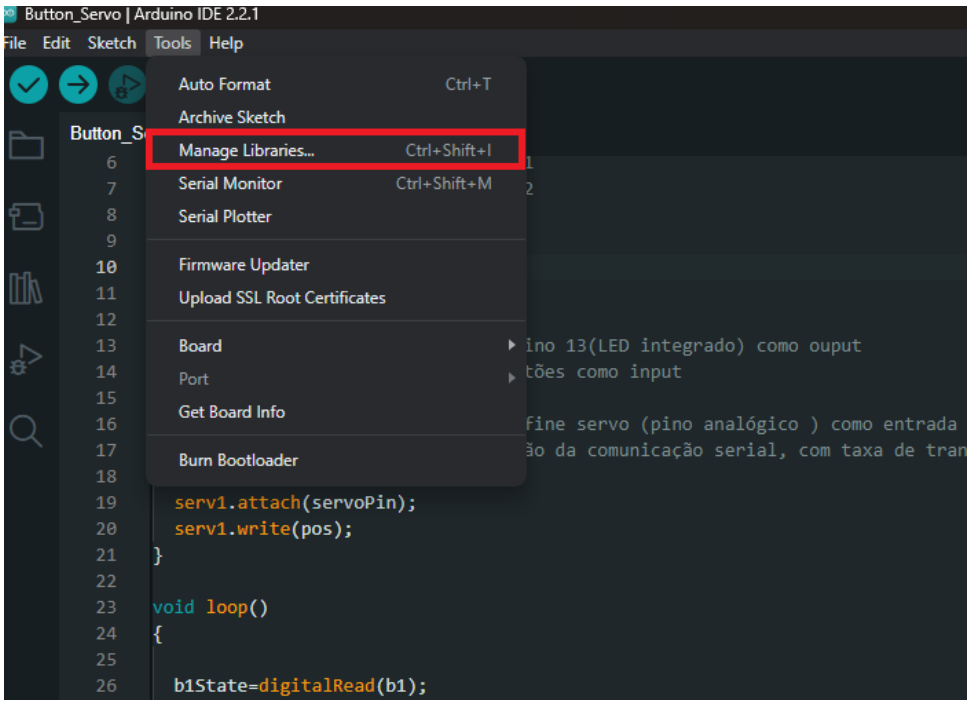
NEEEC-FEUP

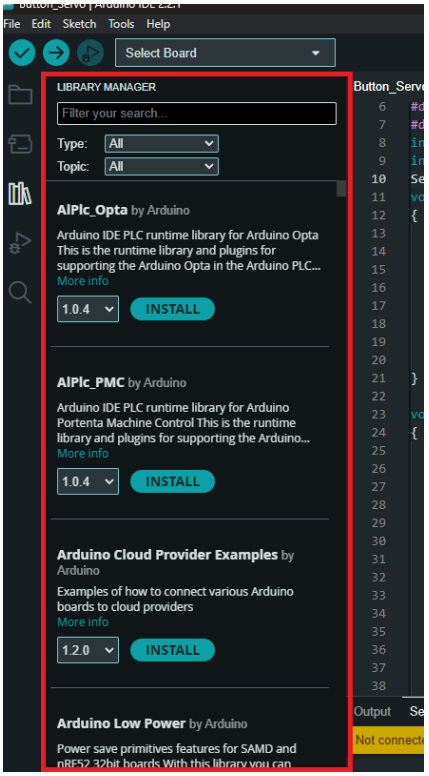


NEEEC-FEUP

3-Bibliotecas

Uma biblioteca em é um conjunto de funções já criadas que facilitam a realização de tarefas comuns ao programar dispositivos Arduino. Estas bibliotecas fornecem código pré-fabricado para fazer coisas como controlar motores, ler sensores ou interagir com componentes específicos, o que poupa tempo e torna o desenvolvimento de projetos mais simples.





NEEEC-FEUP

4- Desafios

4.1- “Hello, World!” e digitalWrite

O objeto é enviar a famosa mensagem "Hello, World!" para a porta serial do Arduino e, ao mesmo tempo, controlar o estado de um LED, ligando e desligando-o. Essas tarefas podem parecer básicas, mas são os primeiros passos cruciais no desenvolvimento de projetos mais complexos com Arduino.

1. Montagem:

- Insere o LED na breadboard, **lembrando que os LEDs têm um ânodo (+) e um cátodo (-).**
- Conecta o anodo do LED a um pino digital no Arduino, por exemplo.
- Conecta o catodo do LED a uma resistência de 220 ohms e, em seguida, conecta a outra extremidade da resistência ao GND do Arduino.

2. Faz o código e conecta o Arduino ao computador:

- Abre o software Arduino IDE no teu computador.
- Faz o código.
- Usa um cabo USB para conectar o Arduino ao teu computador.
- Carrega o código no arduino.

➔ Para carregar o código:
  - ◆ Vai até ‘Ferramentas > Placa’ e seleciona a placa que estás a usar;
  - ◆ Vai até ‘Ferramentas > Porta’ e seleciona a porta com a qual o arduino está conectado(se não souberes qual é a porta desconecta o arduino e volta a ligar e seleciona a porta que aparece);
  - ◆ Agora verifica o teu código clicando no ‘certo’ na parte superior esquerda do IDE;
  - ◆ Caso esteja tudo bem com o teu código basta clicar na ‘seta’ na parte superior esquerda do IDE.

4.2- Controlo de um Servo Motor com dois Botões

Neste desafio, exploraremos o mundo do controle de servomotores usando dois simples botões. Os servo motores são dispositivos versáteis, capazes de movimentar-se com precisão em ângulos específicos. Através da interação com os botões, aprenderemos a manipular a posição do servo, aprimorando assim o tópico de comunicação entre componentes.

1. Conecta os botões à placa Arduino:

- Conecta um pino de cada botão a pinos digitais no Arduino, por exemplo, os pinos 2 e 3.
- Conecta os pinos restantes dos botões a GND na placa Arduino.
- Utiliza resistências de 10k**Ω**+ entre os pinos 1 e 2 (ou 3 e 4) dos botões e o GND para pôr em modo pull down

2. Conecta o servo motor à placa Arduino:

- Conecta o fio de controle do servo a um pino PWM(~) na placa, por exemplo, o pino 9.
- Conecta o fio de alimentação do servo a uma saída de alimentação 5V na placa.
- Conecta o fio de terra do servo a uma saída GND na placa.

3. Cria um novo projeto e faz o código

Para facilitar a escrita do código, começa por atribuir variáveis para ler os valores dos botões, e definir também uma variável para guardar a posição máxima que o servo pode girar(180º); . Em seguida, verifica se a tua posição atual é menor que a máxima e se um dos botões está ativado.

Usa também a biblioteca Servo.h para os servos.

Exemplo:

```
#include <Servo.h>

Servo myservo;  // Cria um objeto do tipo servo para o controlares

int val=0;     // Variável com o ângulo que queres

void setup() {
  myservo.attach(9);  // anexa o pino 9 ao teu objeto
}

void loop() {
  if(val<180) {
        myservo.write(val); // Move o servo para o ângulo em val

        val++;
        }
}
```

4.3- Potenciómetro e Servo

Neste desafio, vamos explorar o poder do potenciômetro como uma ferramenta de controle dinâmica para os servo motores. O potenciômetro é um componente que permite variar a resistência elétrica, e com ele, seremos capazes de controlar a posição de um servo motor de forma contínua e intuitiva. Ao girar o potenciômetro, veremos o servo responder em tempo real, proporcionando uma experiência prática na interface entre entrada analógica e controle de servo motores.

1. Conecta o potenciômetro à placa Arduino:

- Conecta um dos pinos das extremidades aos 5V e o outro ao GND
- Conecta o pino central a uma porta analógica do arduino.

2. Conecta o servo motor à placa Arduino:

- Conecta o fio de controle do servo a um pino PWM(~) na placa;
- Conecta o fio de alimentação do servo a uma saída de alimentação 5V na placa;
- Conecta o fio de terra do servo a uma saída GND na placa;

3. Cria um novo projeto e faz o código

Usa a função analogRead() para transformar os valores do potenciômetro em inteiros de 0 a 1023. Reutiliza parte do código do exercício anterior e utiliza a função map() para para passar os valores do potenciômetro de [0;1023] para [0;180];

Arduino Quick Reference Cheatsheet

For more information visit: http://arduino.cc/Reference/

Program Flow / Control

```
/* Each Arduino Sketch must contain the
following two functions */

void setup()
{
  // runs only once.
}
void loop()
{
  // runs repeatedly.
}

delay(time_millis); // pauses program in ms
delayMicroseconds(time_micros); //pause µs
```

Basic Logic

Simple if()-else

```
if(condition)
{
      //true condition code here
}
else
{
     //false statement code here
}
```

Compound if()-else if()-else

```
if(condition1)
{
      //true condition1 code here
}
else if(condition2)
{
     //true condition2 code here
}
else
{
     //false statement code here
}
```

Pin Configuration - INPUT vs OUTPUT

```
pinMode(pin, INPUT/OUTPUT/INPUT_PULLUP);
```

OUTPUT Control

```
digitalWrite(pin, val); // val: HIGH or LOW
analogWrite(pin, val);  // val: 0 to 255.

tone(pin, freq); // freq in Hertz
tone(pin, freq, duration); //duration in ms
noTone(pin); // stop tone on pin
```

Reading INPUTs

```
buttonPress = digitalRead(pin); // any pin
sensorVal = analogRead(pin); // A0-A5 pins
```

Communication

```
Serial.begin(baudrate);
Serial.print("");   // print data out
Serial.println(""); // print with new line

x = Serial.read(); // reads a single byte
                   // data
x = Serial.parseInt(); // read the next
                       // available integer
```

Looping

```
while(condition)
{
}
for(init; condition; update variable)
{
}
```

Comments/Debug

```
/* this is a multiline comment. nothing
between here will be run or executed */

// this is a single
// line comment
```

Data \ Variable Types

```
const (indicates a constant data type)
void (null data type)
int (integer -32,768 to 32,767)
float (floating point / decimal numbers)
arrayName[] - list of elements (any type)
String (array of characters)
```

System constants / functions

```
HIGH / LOW
OUTPUT / INPUT / INPUT_PULLUP

millis();  //returns # of milliseconds
micros();  //returns # of microseconds
```

Math Operators

```
=   // assignment
+   // addition
-   // subtraction
*   // multiplication
/   // division
%   // modulus
```

Logic Operators

```
==   // is equal to?
!=   // is not equal to?
<    // less than
>    // greater than
<=   // less than or equal
>=   // greater than or equal
&&   // compound AND
||   // compound OR
!    // NOT (inverse)
```

Libraries

```
#include <libraryName.h>

libraryName objectName;

// read library documentation for usage.
```

rev. 0.2

NEEEC-FEUP