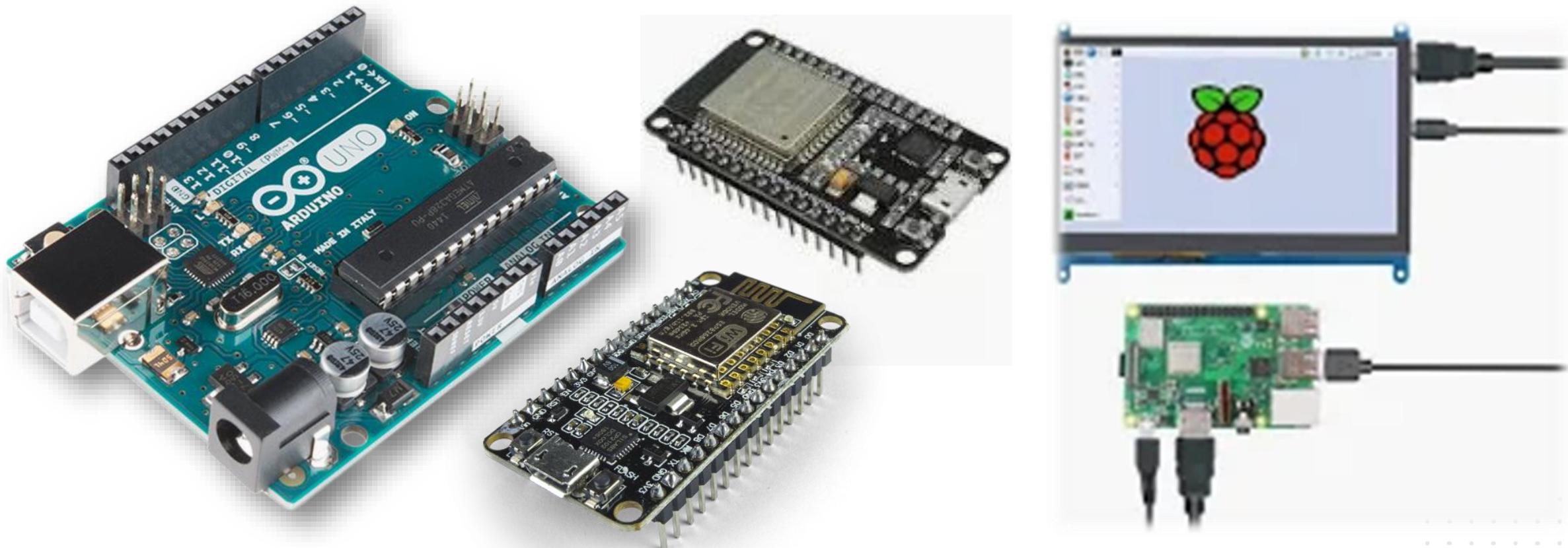


# INTRODUÇÃO:

## Do Arduino ao Raspberry Pi



**Contato:**

[massaki.igarashi@senai.edu.br](mailto:massaki.igarashi@senai.edu.br)

# 1 – Arduino?

“

O Arduino, essencialmente, é uma placa (de circuito impresso) única de hardware e software aberto com microcontrolador (geralmente Atmel AVR) que foi concebido para ser acessível e de fácil utilização; logo, por ser aberto, seu projeto pode ser compartilhado e alterado por todos.

Mas o nome Arduino®, que nasceu com as placas Arduino One e Duemilanove, acabou se tornando mais que estas simples placas, hoje é uma família de placas e ambiente de programação e compilação IDE (acrônimo em inglês para *Integrated Development Environment*); onde esta IDE pode ser previamente instalado ou executado de maneira online pelo navegador de internet.

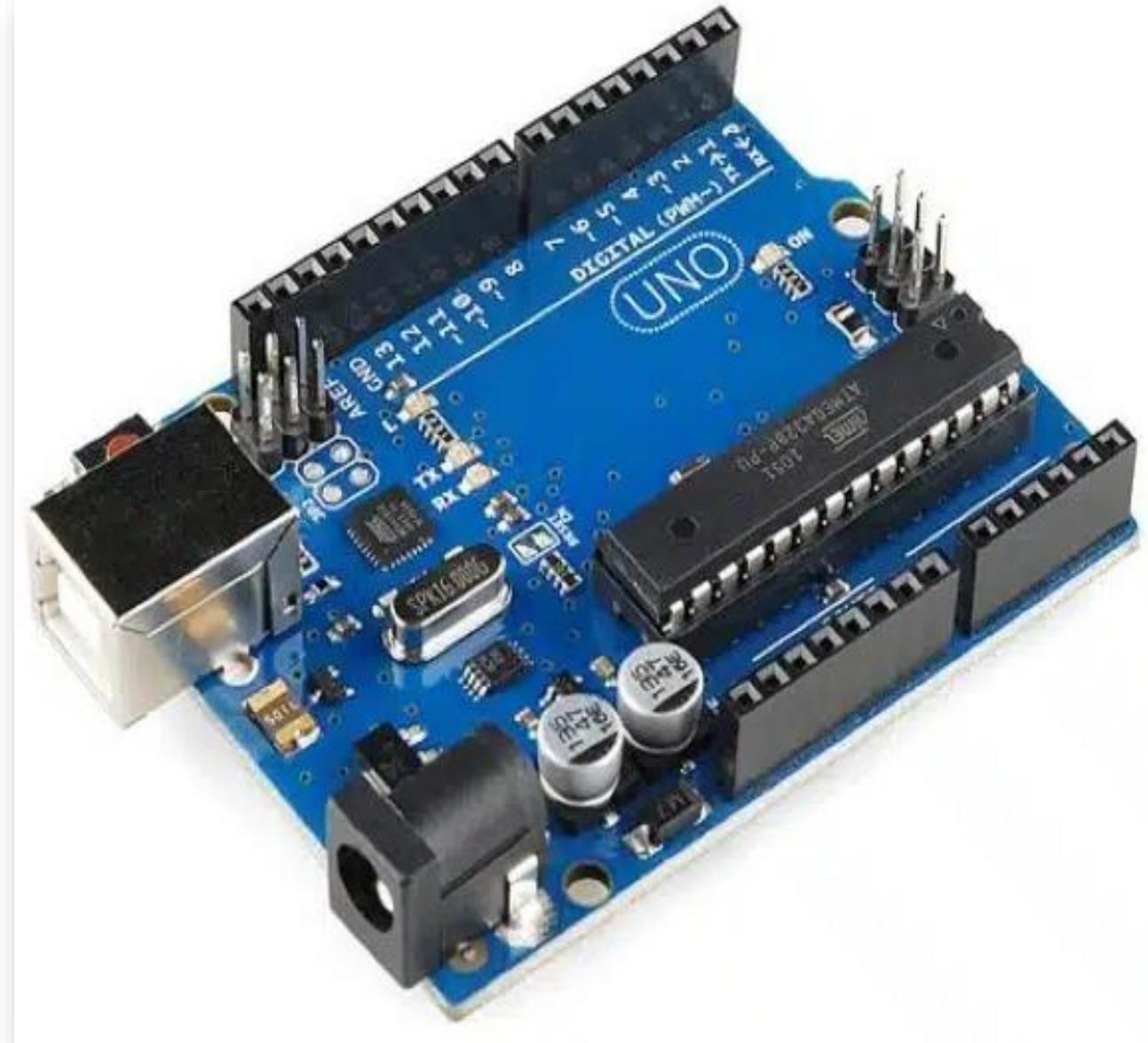
”

# O MICROCONTROLADOR

## Arduino Uno

Microcontrolador	ATmega328P
Tensão de Operação	5 Volts
Tensão de Entrada (Recomendada)	7-12V
Tensão de Entrada (Limites)	6-20V
Pinos I/O	14 (6 com PWM)
Pinos Analógicos	6
Corrente por pino I/O	40 mA
Corrente no pino 3.3v	50 mA
Memória Flash	32 kB
SRAM	2 kB
EEPROM	1 kB
Velocidade de Clock	16 MHz

Tabela 1 - Especificações do Arduino UNO



# Iniciando com a placa didática Arduino Uno



PRODUCTS ▾

EDUCATION

PROFESSIONAL

BACK TO SCHOOL

ARDUINO CLOUD



```
sketch_aug28a | Arduino IDE 2.1.1
Edit Sketch Tools Help
Arduino Uno
sketch_aug28a.ino
1 void setup() {
2 // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7 // put your main code here, to run repeatedly
8 }
9
10
```

## Função setup()

Aqui você insere as **configurações iniciais que executarão uma única vez**, ao ligar seu Arduino ou ao pressionar o botão reset da placa.

## Função loop()

Na função loop você insere **tudo o que precisará ser executado e repetido a cada ciclo de clock** do microcontrolador da placa.



**ATENÇÃO:** Antes de sair gravando seu microcontrolador, teste seu código em plataformas online como



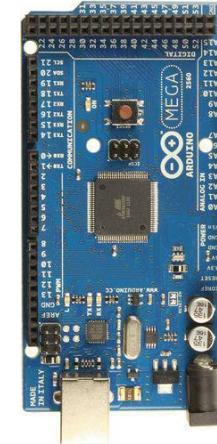
AUTODESK<sup>®</sup>  
TINKERCAD<sup>®</sup>

e/ou



# 1 – Plataforma Arduino

## 2.1 – Algunas placas Arduino

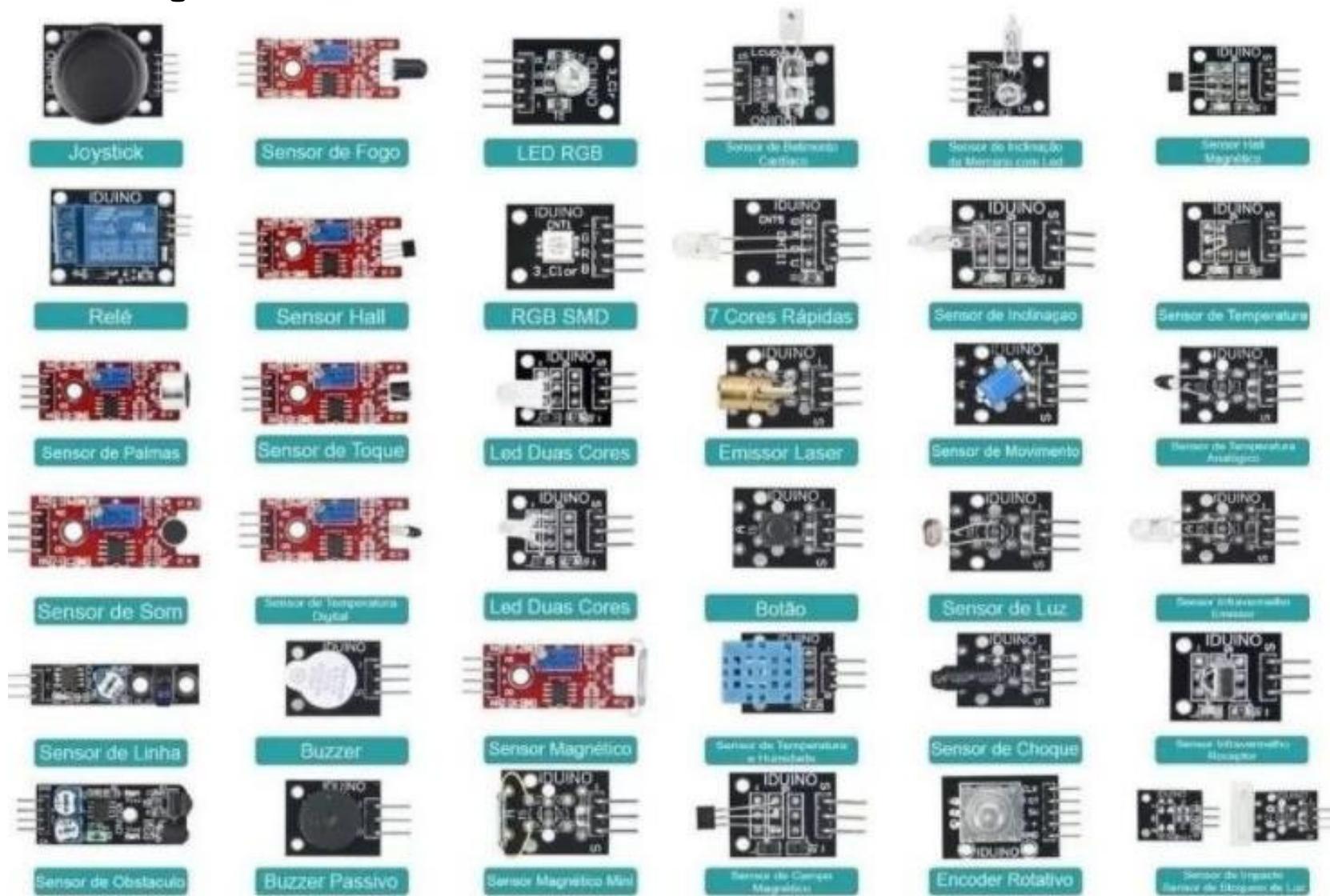
ARDUINO UNO REV3	ARDUINO NANO	ARDUINO UNO WIFI REV2	ARDUINO MEGA 2560 REV3	<u>ARDUINO NANO 33 BLE</u> <u>SENSE</u>	<u>Arduino Portenta</u> <u>Vision Shield</u>
					

# 1 – Plataforma Arduino

## 2.1 – Algunas placas Arduino

	ARDUINO UNO REV3	ARDUINO NANO	ARDUINO UNO WIFI REV2	ARDUINO MEGA 2560 REV3	ARDUINO NANO 33 BLE SENSE	Arduino Portenta Vision Shield
MICROCONTROLLER	<u>ATmega328P</u>	ATmega328	ATmega4809	<u>ATMEGA2560</u>	<u>ARDUINO NANO 33 BLE SENSE</u>	STM32H747XI DUAL CORTEX M7+M4 32 BITS LOW POWER MCU ARM c/ engine gráfica
Operating Voltage	5V	5V	5V	5V	3.3V	3.3 V
Input Voltage (limit)	20	12	12	6-20V	21V	5V
Digital I/O Pins	14 (of which 6 provide PWM output)	22 (6 of which are PWM)	14 — 5 Provide PWM Output	54 (of which 15 provide PWM output)	14	80
Analog Pins	6 (ADC 8 bits)	8 (ADC 8 bits)	6 (ADC 8 bit)	16 (ADC 10 bits)	8 (ADC 12 bit 200 ksamples)	5 (3 ADCs 16 bits and 2 DACs 12 bits 1MHz)
DC Current per I/O Pin	20 mA	40 mA	20 mA	20 mA	15 mA	15 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader	32 KB of which 2 KB used by bootloader	48 KB (ATmega4809)	256 KB of which 8 KB used by bootloader	1MB (nRF52840)	Flash Memory Flash NOR 16 MB
SRAM	2 KB (ATmega328P)	2 KB (ATmega328P)	6,144 Bytes (ATmega4809)	8 KB	256KB (nRF52840)	0, 8, 16, 32 and 64 Mbytes SDRAM 8MB
EEPROM	1 KB (ATmega328P)	1 KB (ATmega328P)	256 Bytes (ATmega4809)	4 KB	none	external QSPI Flash Memory 0, 2, 16, 128 MByte
Clock Speed	16 MHz	16 MHz	<b>16 MHz</b>	16 MHz	64MHz	<b>480 MHz/240 MHz</b>
Radio module	none	none	<b>u-blox NINA-W102</b>	none	UBLOX NINA B306 (nordic nRF52)	<b>Murata 1DX dual Wifi 802.11b/g/n 65 Mbps and Bluetooth 5.1 BR/EDR/LE</b>
Secure Element	none	none	ATECC608A	none	none	<b>NXP SE0502</b>
Inertial Measurement Unit	none	none	<b>LSM6DS3TR ((6 axis)</b>	none	<b>LSM9DS1 (6 axis)</b>	None
Weight	25 g	7 g	25g	37 g	5g	7g

# Shields para Arduino



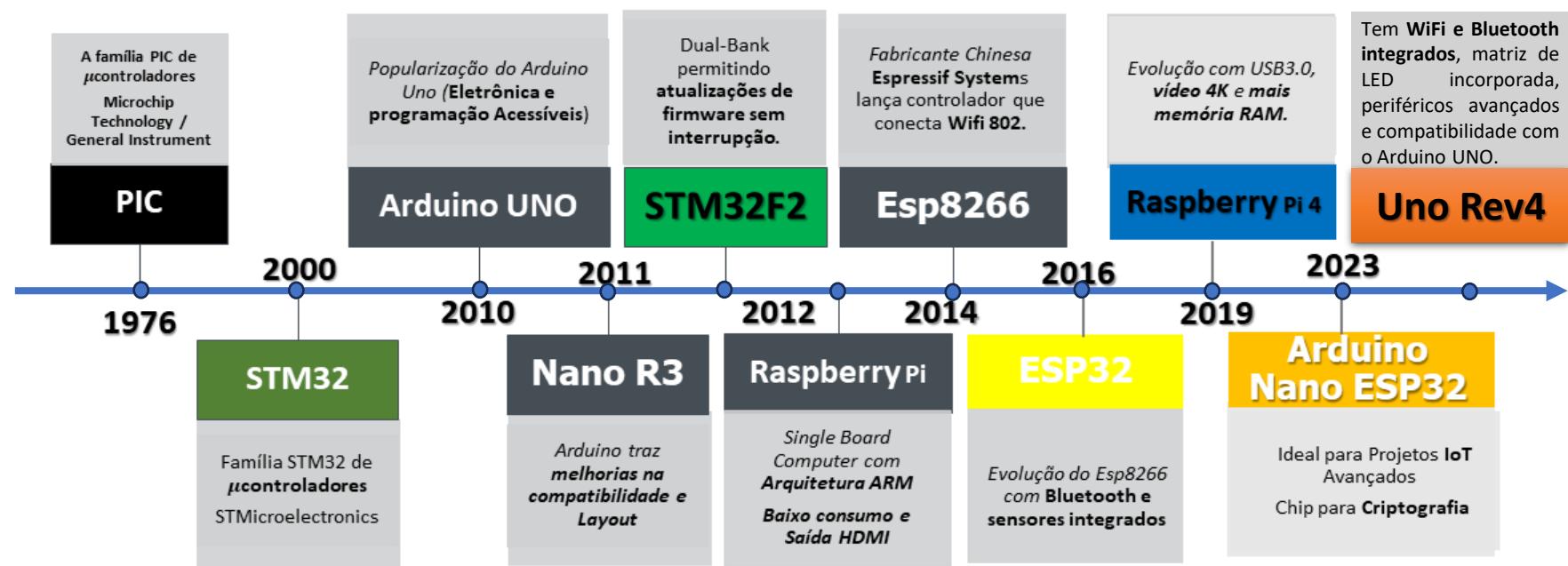
# MICROCONTROLADORES

## Linha do Tempo

Os **microcontroladores** são compactos, com recursos integrados e são usados em aplicações embarcadas, enquanto **microprocessadores** são mais poderosos e usados em computadores e sistemas complexos. Suas diferenças incluem capacidade de processamento, memória, portas de E/S e consumo de energia elétrica.

**Aplicações:** controlar máquinas, processar dados e implementar comunicação e interfaces de usuário, possibilitando a eficiência e inovação nos sistemas industriais. Compreender suas funções, diferenças e aplicações é fundamental para quem trabalha nesta área.

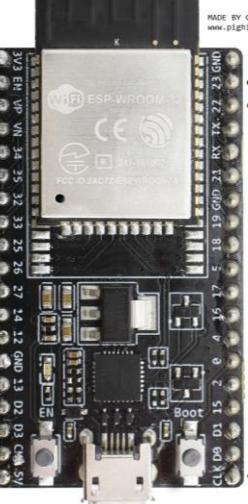
## Linha do Tempo dos uControladores usados para fins didáticos-profissionais



Um **sistema embarcado** é um sistema de computação dedicado a realizar uma ou mais funções específicas. É integrado ao hardware do qual faz parte e é projetado para operar de forma independente ou como parte de um sistema maior. Principais características: **especialização** (projetados para executar tarefas específicas); operam em **Tempo Real**; apresentam **Confiabilidade; Eficiência Energética** (menor consumo de energia); **Tamanho Compacto e Baixo Custo.**;

# 2 – Plataforma ESP32

## 3.1 – Algunas placas ESPRESSIF

ESP32	ESP32 S2
	

	ESP32 Series – S1	ESP32-S2-Saola-1MI
MICROCONTROLLER	Xtensa® single-/dual-core 32-bit LX6	ESP32-S2 embedded, Xtensa® single-core 32-bit LX7 microprocessor,
Operating Voltage	3.3 – 3.6 V	3.3 – 3.6 V
Input Voltage (limit)	3.6 V	3.6 V
Digital I/O Pins	34	43
Analog Pins	18 (12-bit SAR ADC) 2 (8 bits DAC)	20 (12-bit ADC) 2 (8-bit DAC)
DC Current per I/O Pin	40 mA	28 mA – 40 mA
SRAM	• 520 KB SRAM • 16 KB SRAM in RTC (QSPI supports multiple flash/SRAM chip)	• 320 KB SRAM • 16 KB SRAM in RTC
EEPROM	• 448 KB ROM	128 KB ROM
Clock Speed	160 MHz – 240 MHz	160 MHz – 240 MHz
Radio module	802.11 b/g/n Bluetooth V4.2 BR/EDR & LE	802.11 b/g/n Bluetooth V4.2 BR/EDR & LE
Secure Element	Secure boot Flash encryption 1024-bit OTP	Secure boot Flash encryption 4096-bit OTP Supports External encryption/decryption based on XTS-AES
Inertial Measurement Unit	None	none
Weight	10 g	25

# 3 – Conceitos básicos

## 1.1 – Sistema eletrônico

Entrada



**Push Button** **Sensor Hall** **Ultrassônico**  
**De Distância**



**Sensor de  
Luminosidade  
(LDR)**

**Sensor  
Temperatura e  
Humidade**

**Sensor  
Temperatur  
a**

PROCESSAMENTO

Microcontrolador  
Arduino  
ESP32  
...

...



**LEDs de uso geral**  
**(270 nm a 590 nm)**



**IR LED**  
**>880 nm**



**UV LED**  
**<270 nm**



**RGB LED**



**Servo  
Motor**



**Relé SSR  
Shield**



**Buzzer**

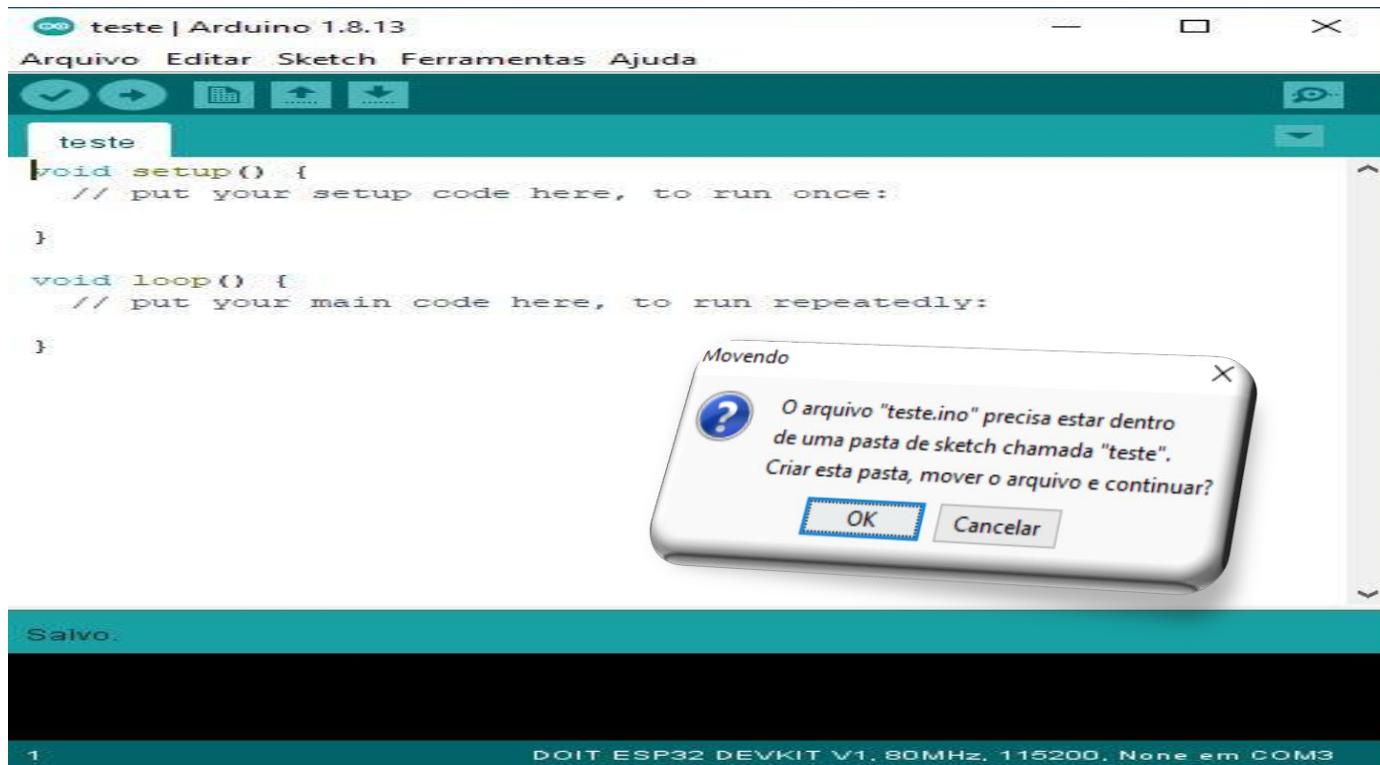
Saída

# 4 – IDE Arduino

## SKETCH

No mundo do Arduino, os programas são conhecidos como “Sketches” (rascunho ou esboço) .

Os Sketches são criados e programados na IDE do Arduino



Linguagem de  
programação

C++

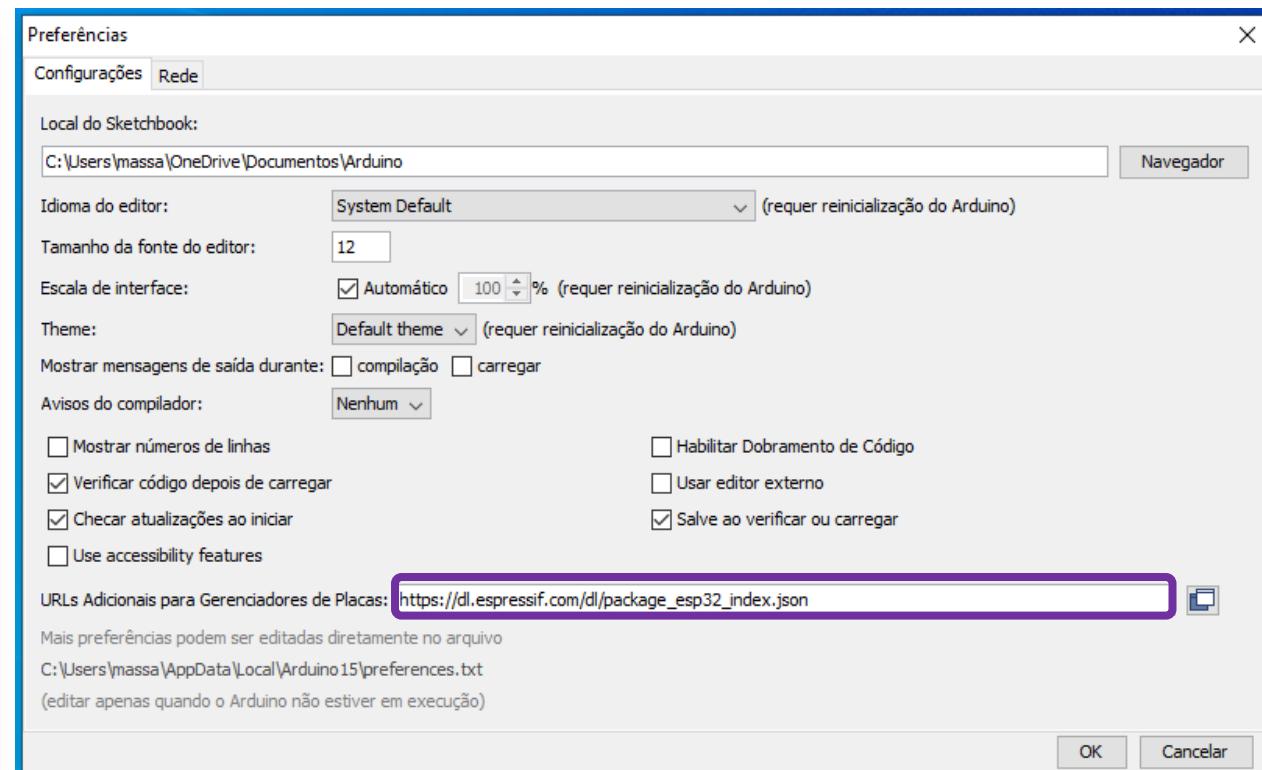
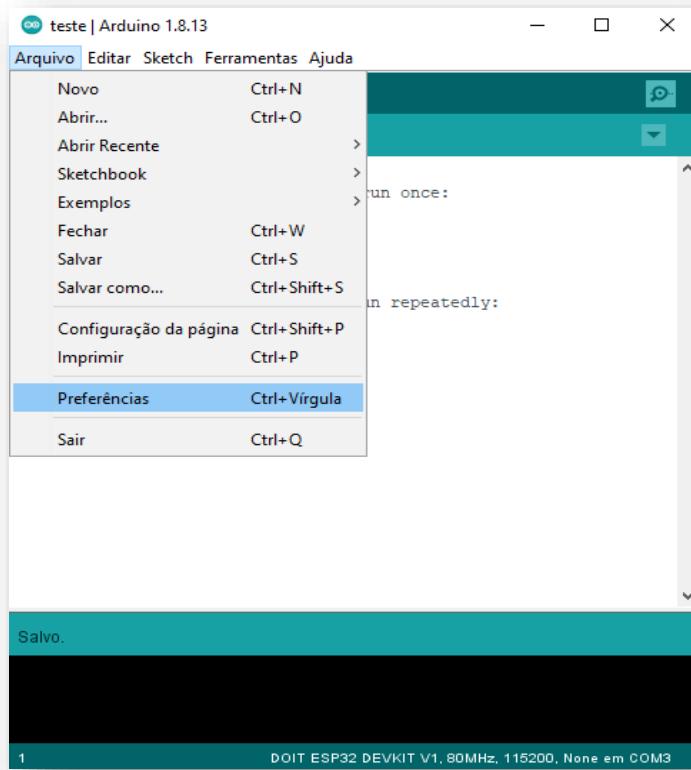
# 4 – IDE Arduino

## 4.1 – Preparando IDE Arduino para programar o ESP32

### ADICIONAR O CAMINHO

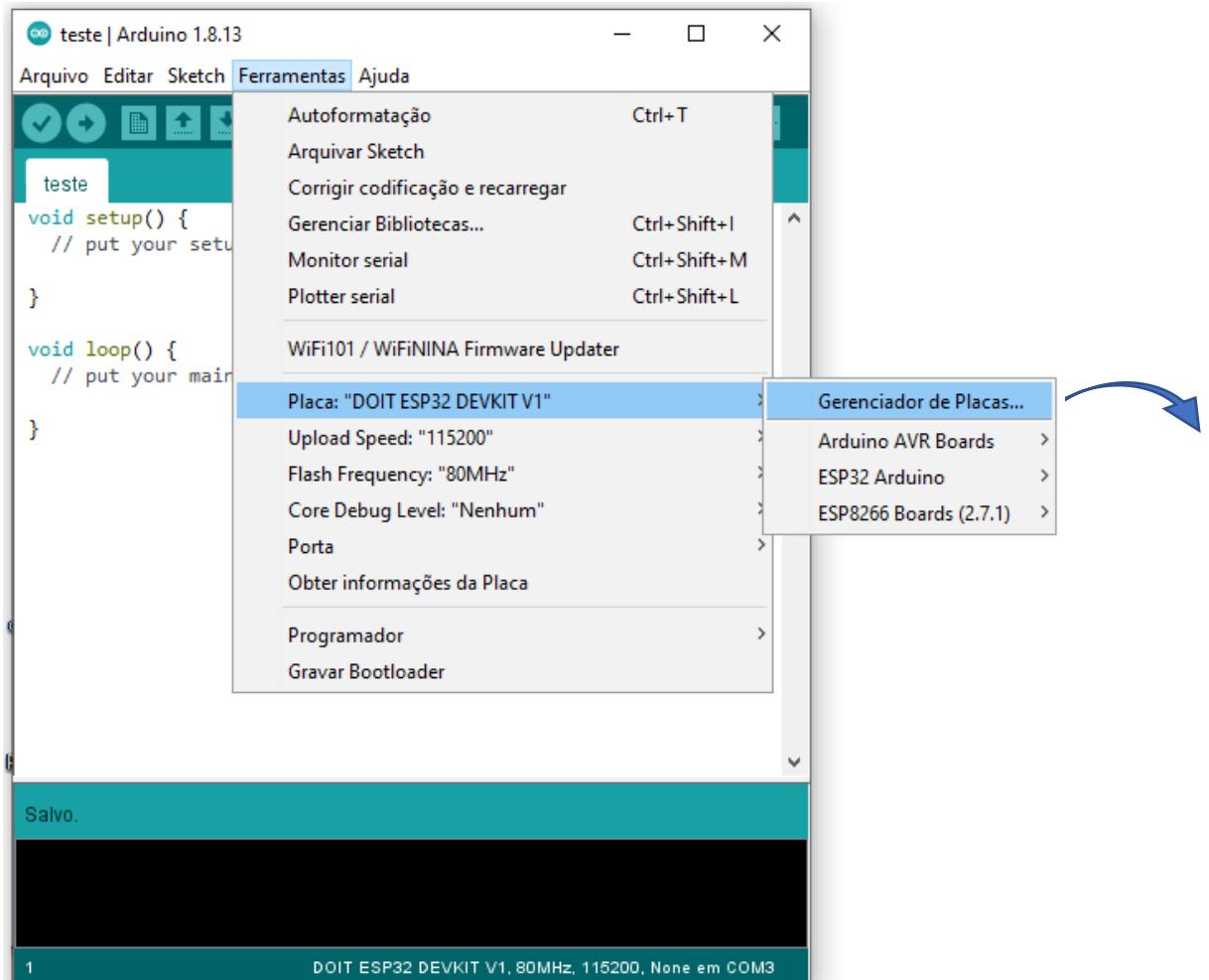
Menu ARQUIVO > PREFERÊNCIAS > URLs Adicionais para Gerenciadores de Placas:

[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)

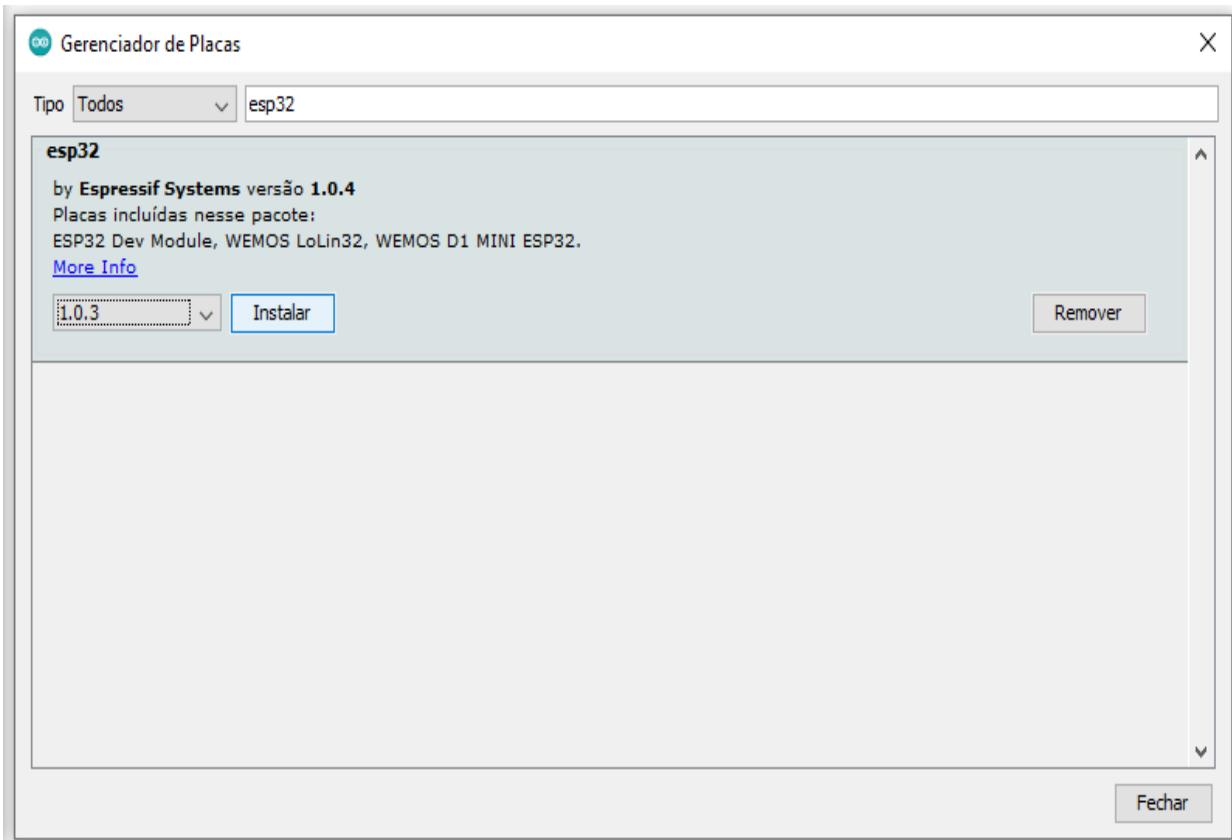


# 4 – IDE Arduino

## 4.1 – Preparando IDE Arduino para programar o ESP32

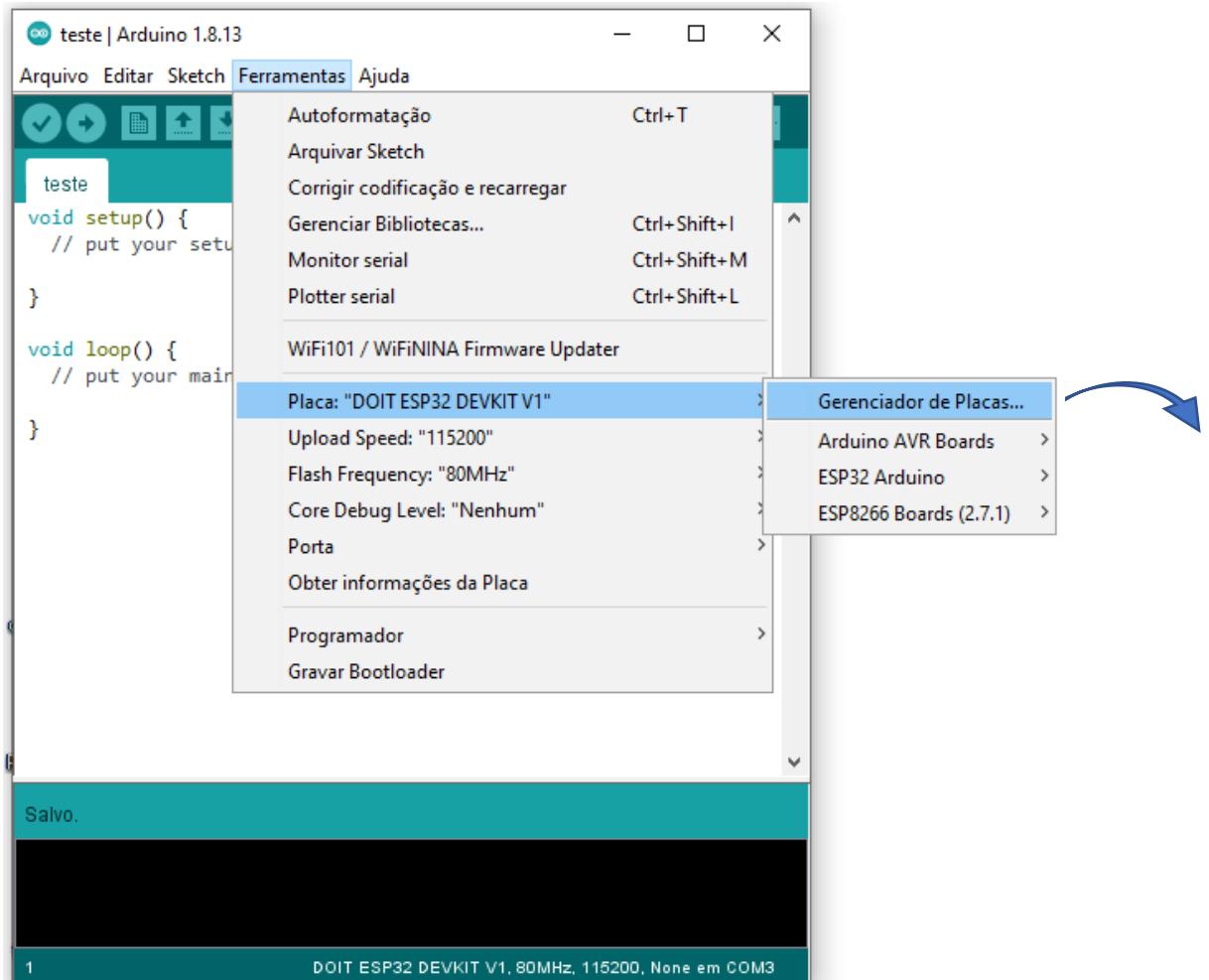


Menu Ferramentas > Placa: ... > Gerenciador de Placas  
Digitar esp32 e clicar INSTALAR

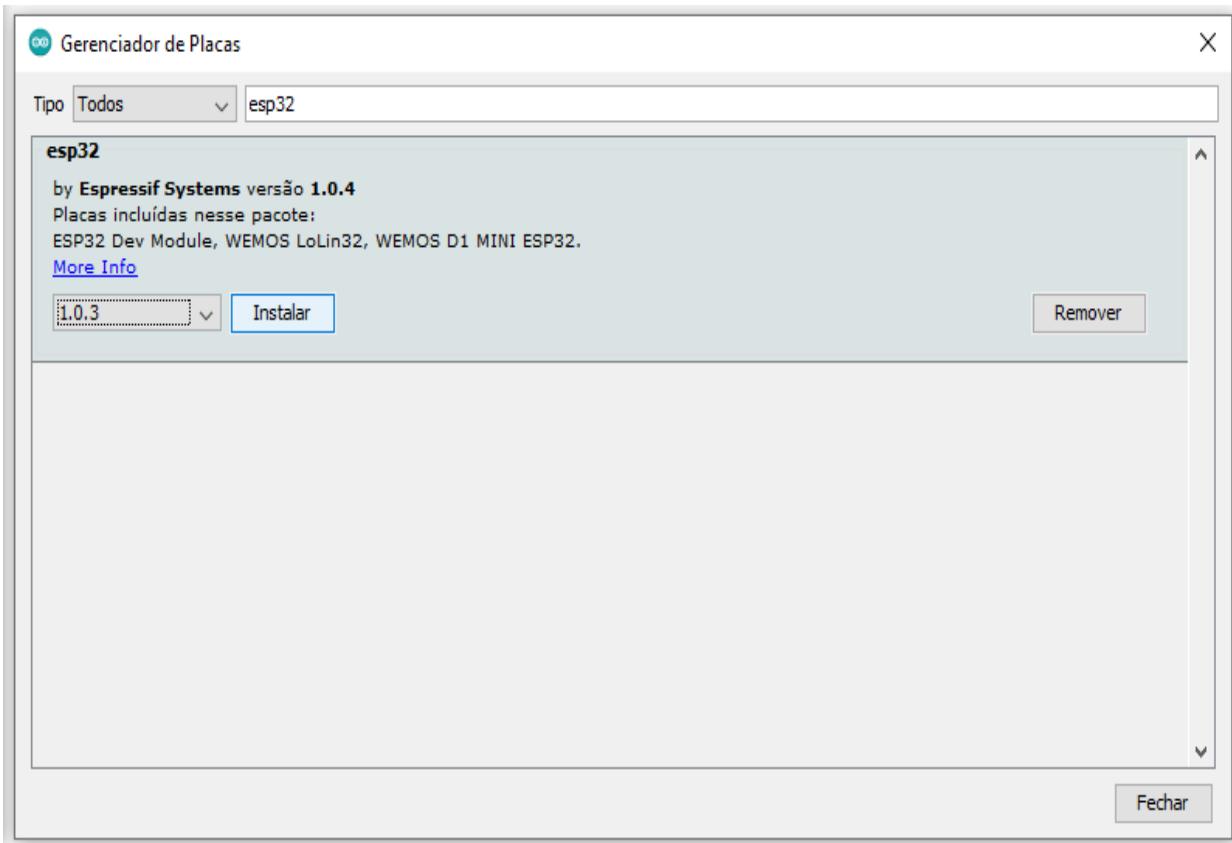


# 4 – IDE Arduino

## 4.1 – Preparando IDE Arduino para programar o ESP32

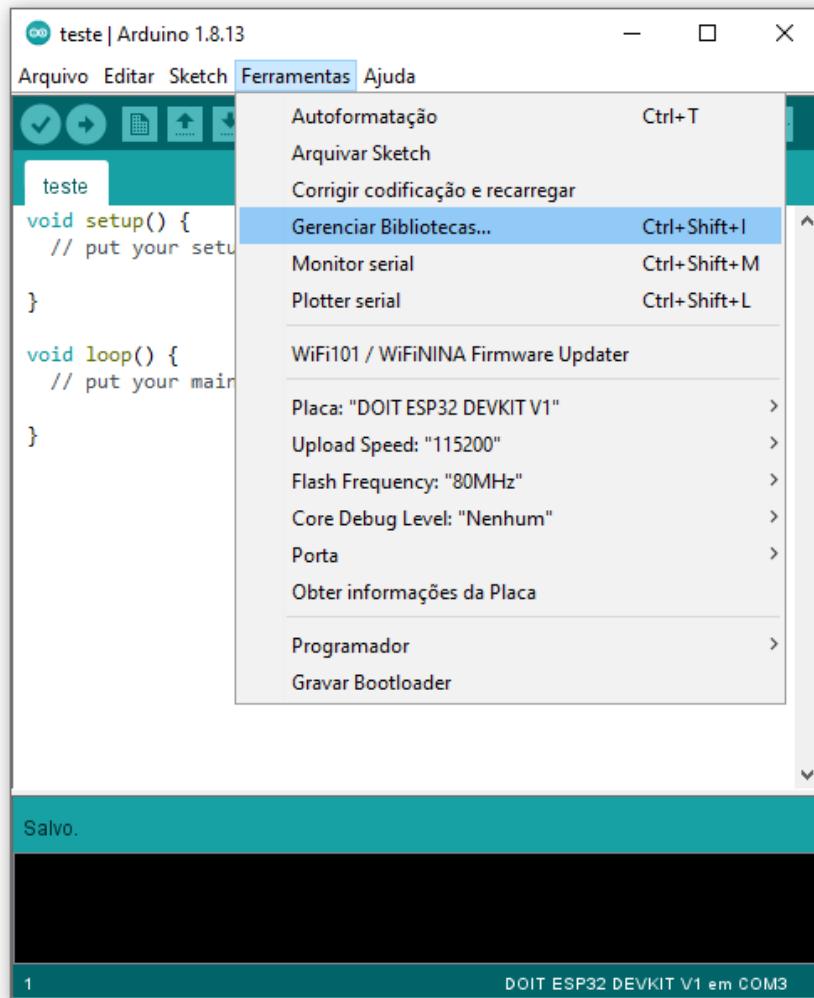


Menu Ferramentas > Placa: ... > Gerenciador de Placas  
Digitar esp32 e clicar INSTALAR

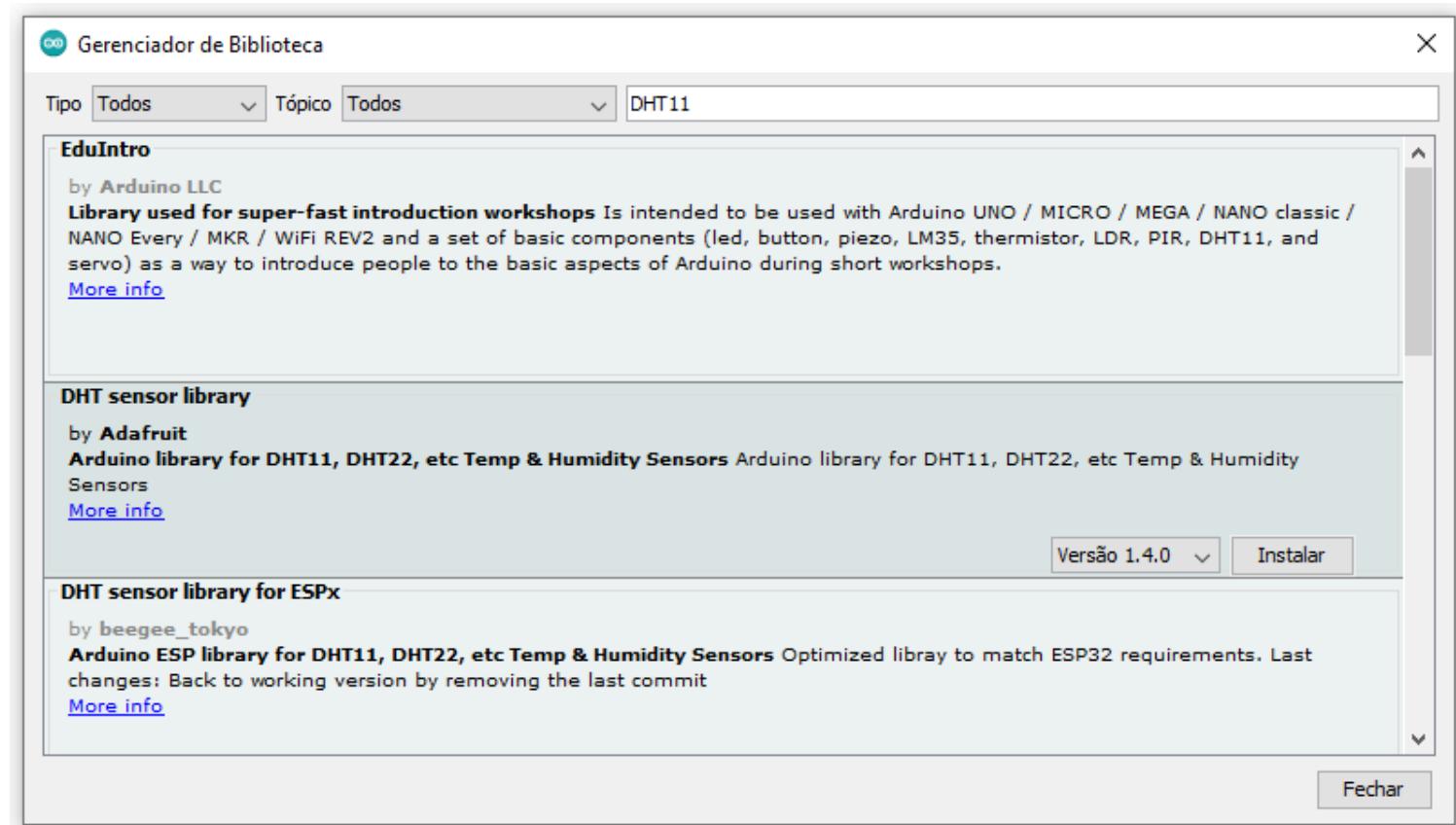


# 4 – IDE Arduino

## 4.2 – Adicionando BIBLIOTECAS

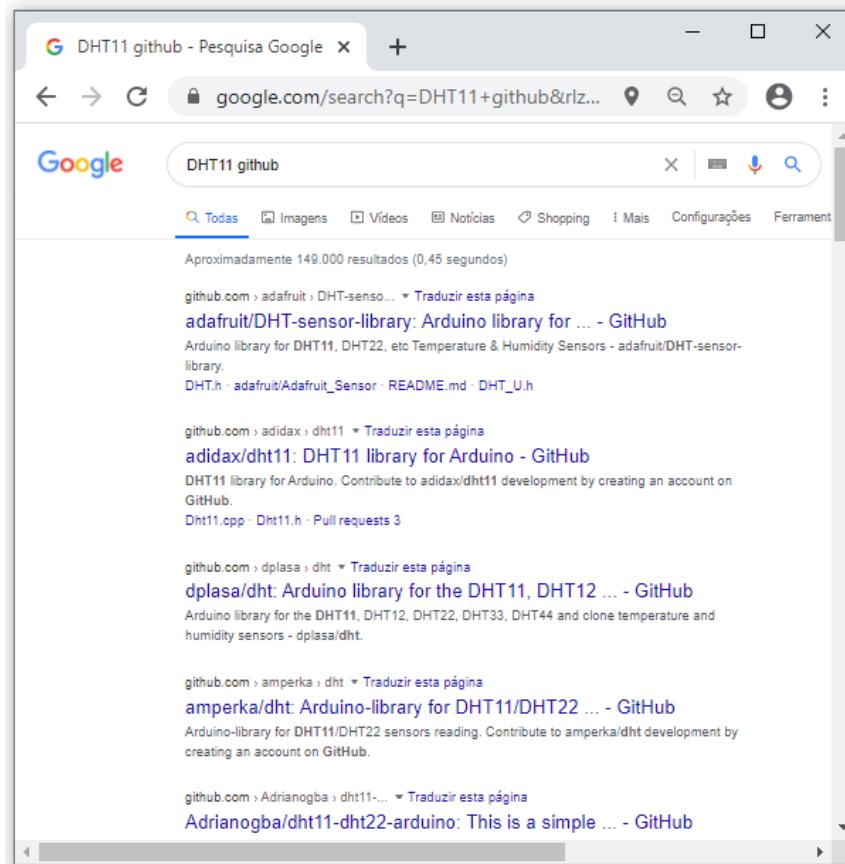


Menu Ferramentas > Gerenciador de Bibliotecas



# 4 – IDE Arduino

## 4.3 – Adicionando BIBLIOTECAS pelo Git Hub



Clicar em CODE > Download ZIP

Why GitHub? Team Enterprise Explore Marketplace Pricing

Search

Sign in Sign up

adafruit / DHT-sensor-library

Code Issues 9 Pull requests 12 Actions Projects Security Insights

Join GitHub today

GitHub is home to over 50 million developers working together to host and review code, manage projects, and build software together.

Dismiss

Sign up

master 3 branches 19 tags

Go to file Code

dherrada Bump to 1.4.0

.github actionified, formatted and doxy'd

examples Change initial HIGH delay, fix comment

.gitignore Add .gitignore

CONTRIBUTING.md [Update URL]

DHT.cpp Merge pull request #159 from Rotzbu...

DHT.h Adjust comments to comply with doxygen.

DHT\_U.cpp actionified, formatted and doxy'd

DHT\_U.h actionified, formatted and doxy'd

Clone HTTPS GitHub CLI

Use Git or checkout with SVN using the web URL

Open with GitHub Desktop

Download ZIP

About

Arduino library for DHT11, DHT22, etc Temperature & Humidity Sensors

learn.adafruit.com/dht

Readme

MIT License

Releases 19

1.4.0 - Added missing micros... Latest

on 16 Sep

+ 18 releases

<https://github.com/adafruit/DHT-sensor-library/archive/master.zip>

# 4 – IDE Arduino

## 4.4 – Utilizando os EXEMPLOS

The image shows two screenshots of the Arduino IDE. The left screenshot shows the 'Exemplos' (Examples) menu item selected in the 'Arquivo' (File) menu. A submenu titled 'Exemplos embutidos' (Built-in Examples) is open, displaying various example sketches like '01.Basics', '02.Digital', '03.Analog', etc. The right screenshot shows the 'Blink' sketch open in the editor. The code for 'Blink' is displayed, showing comments explaining the setup and loop functions.

**COMENTÁRIOS NO SKETCH**

/\*  
Comentário de  
várias linhas  
\*/

//Ou comentário de 1 linha

```
/*
Comentário de
várias linhas
*/
//Ou comentário de 1 linha

// a função SETUP é executada apenas quando você pressiona RESET ou liga a placa
void setup() {
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED_BUILTIN, OUTPUT);
}

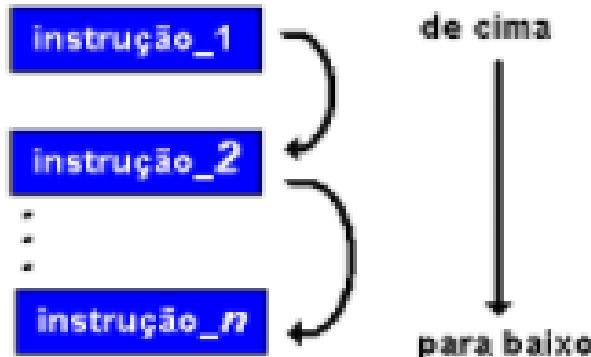
// a função LOOP roda a cada ciclo do microcontrolador
void loop() {
    digitalWrite(LED_BUILTIN, HIGH);      // turn the LED on (HIGH is the voltage level)
    delay(1000);                       // wait for a second
    digitalWrite(LED_BUILTIN, LOW);       // turn the LED off by making the voltage LOW
    delay(1000);                       // wait for a second
}
```

# 5 – Programação

## 5.1 - Estrutura de controle

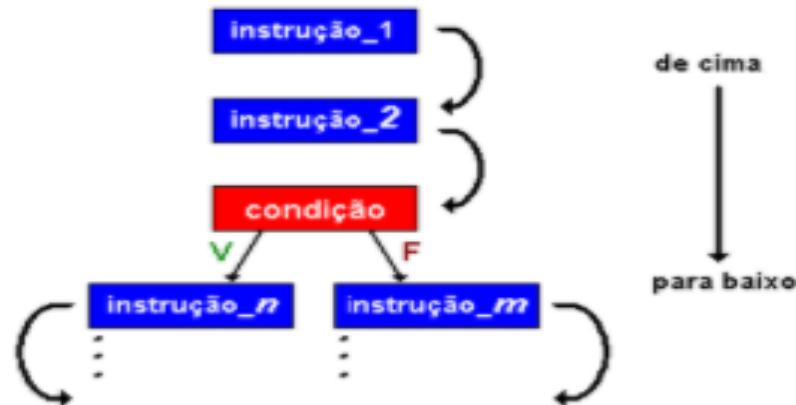
As instruções são programadas **SEQUENCIALMENTE** para definir as ações a serem executadas pelo microcontrolador. Basicamente, as **ESTRUTURAS DE UM PROGRAMA** podem ser:

### Seqüencial



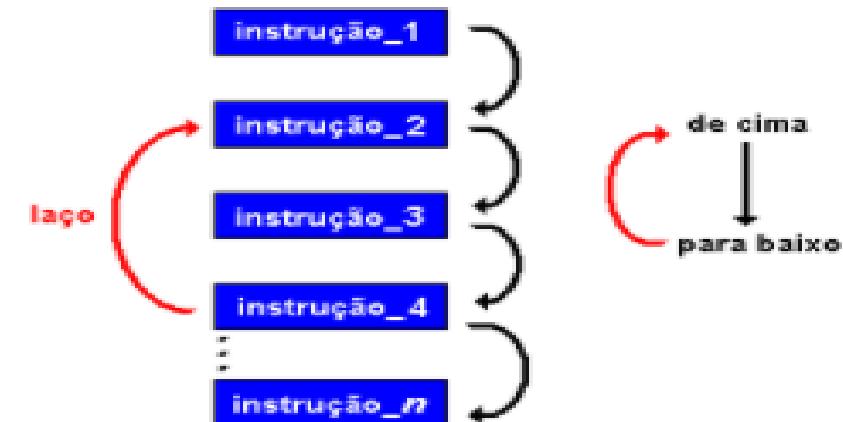
Conjunto de instruções executados em uma sequência linear, obedecendo a pontuação e o alinhamento (endentação) definido nas instruções envolvidas.

### Seleção ou Condicional



A partir de um teste condicional elaborado com operadores relacionais e até lógicos, uma instrução, ou um conjunto de instruções, podem ser executados ou não, dependendo exatamente do resultado do teste efetuado (lógica convencional - **V = verdadeiro** ou **F = falso**).

### Repetição ou Laço

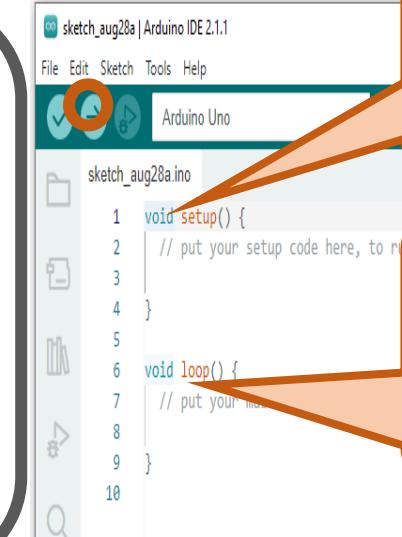


Conjunto de instruções, ou uma única instrução, que será executada repetidamente, de acordo com o resultado de um teste condicional que compõem a instrução de repetição e permite o laço para nova execução destas instruções que compõem o bloco de repetição.

# Exemplos de configurações usadas na função Loop

```
void setup() {  
  
Serial.begin(9600); // Inicializa Comunicação Serial a 9600 bps  
  
lcd.begin(16, 2); // Inicializa display LCD (16 colunas x 2 linhas)  
  
pinMode(10, INPUT); // Configura Pino Digital D10 p/ entrada digital (Botão)  
  
pinMode(13, OUTPUT); // Configura Pino Digital D13 como saída digital  
  
}
```

Os símbolos { e } servem, respectivamente, p/ informar o **início** e o **fim** de uma função ou para especificar diferentes ações a serem executadas dentro de uma estrutura de controle de fluxo (condicional ou loop repetição);:



```
sketch_aug28a | Arduino IDE 2.1.1  
File Edit Sketch Tools Help  
Arduino Uno  
sketch_aug28a.ino  
1 void setup() {  
2 // put your setup code here, to run  
3 // once before the loop()  
4 }  
5  
6 void loop() {  
7 // put your main code here,  
8 // to run repeatedly  
9 }  
10
```

## Função setup()

Aqui você insere as **configurações iniciais** que executarão uma **única vez**, ao ligar seu Arduino.

## Função loop()

Na função loop você insere **tudo o que precisará ser executado e repetido a cada ciclo de clock** do microcontrolador da placa.

/\* Já os símbolos de /\* e \*/ são usados, respectivamente, no início e no final de um comentário de várias linhas ou, caso seja desejado apenas um comentário único, ou seja, uma explicação junto à seu comando de execução, basta usar //duas barras de divisão consecutivas no início do comentário.\*/



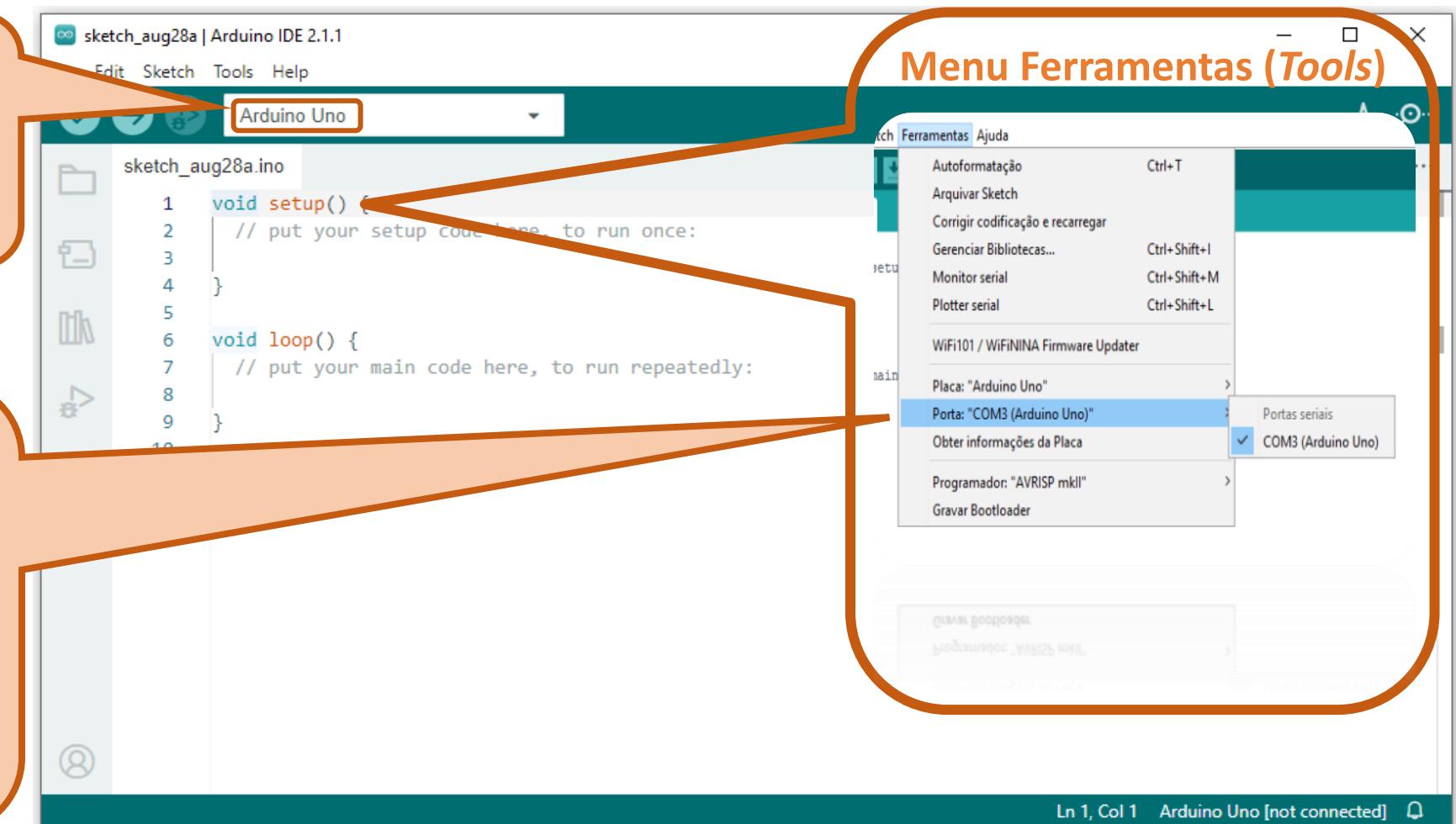
# DICA Nº 1

## 1º) Selecione a Placa correta!

Preste atenção para escolher a placa correta para que não ocorram erros de gravação no seu kit de microcontrolador; isto pode ocasionar queima da memória flash do seu microcontrolador.c

## 2º) Selecione a Porta Serial!

Praticamente todas as placas didáticas que existem no mercado se comunicarão com seu computador e serão gravadas por meio da porta serial de dados; portanto, é importante que você preste muita atenção para selecionar a porta correta que será usada por seu microcontrolador.

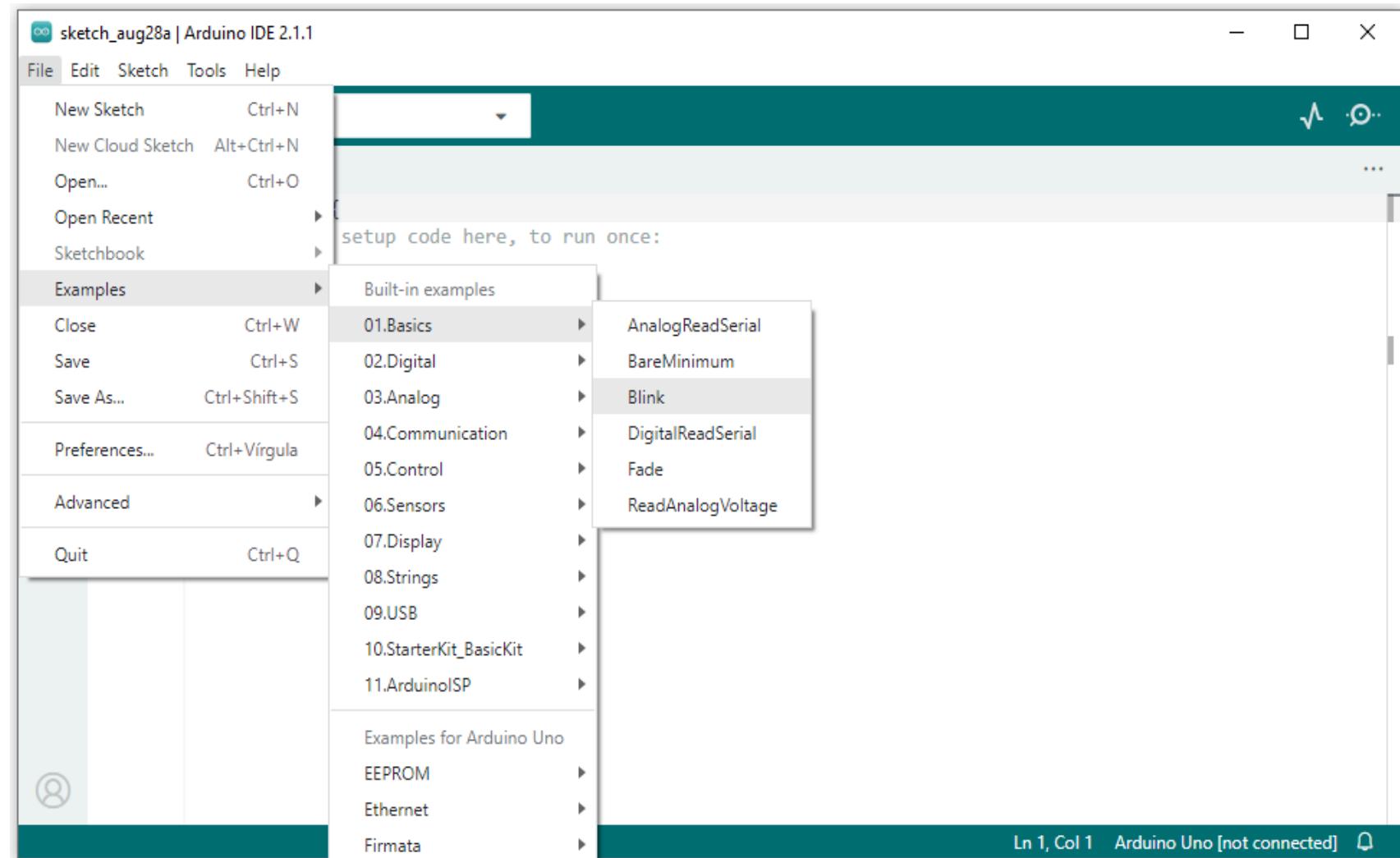


# DICA Nº 2

**Sempre testar a sua placa antes de embarcar qualquer novo código: para isso programe o código do exemplo *Blink* que se encontra no *Menu***

*File > Examples > 01. Basics > Blink*

Isto poupará sua dor de cabeça e tempo caso a placa tenha algum problema de fabricação; independente do seu código ou qualquer erro na lógica do programa.



# DICA Nº 3

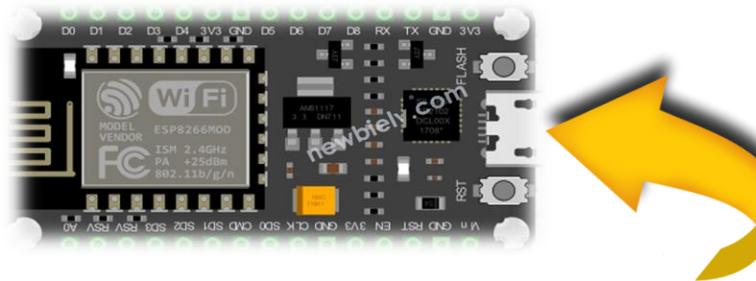


## ATENÇÃO

Nunca se esqueça de ir no **Menu Ferramentas (Tools) > Gerenciador de bibliotecas (Manage Libraries)** e instale as bibliotecas necessárias para que todos os periféricos adicionados por você ao seu projeto funcionem adequadamente.



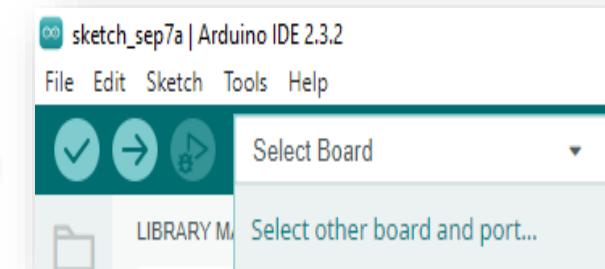
## DICA Nº 4



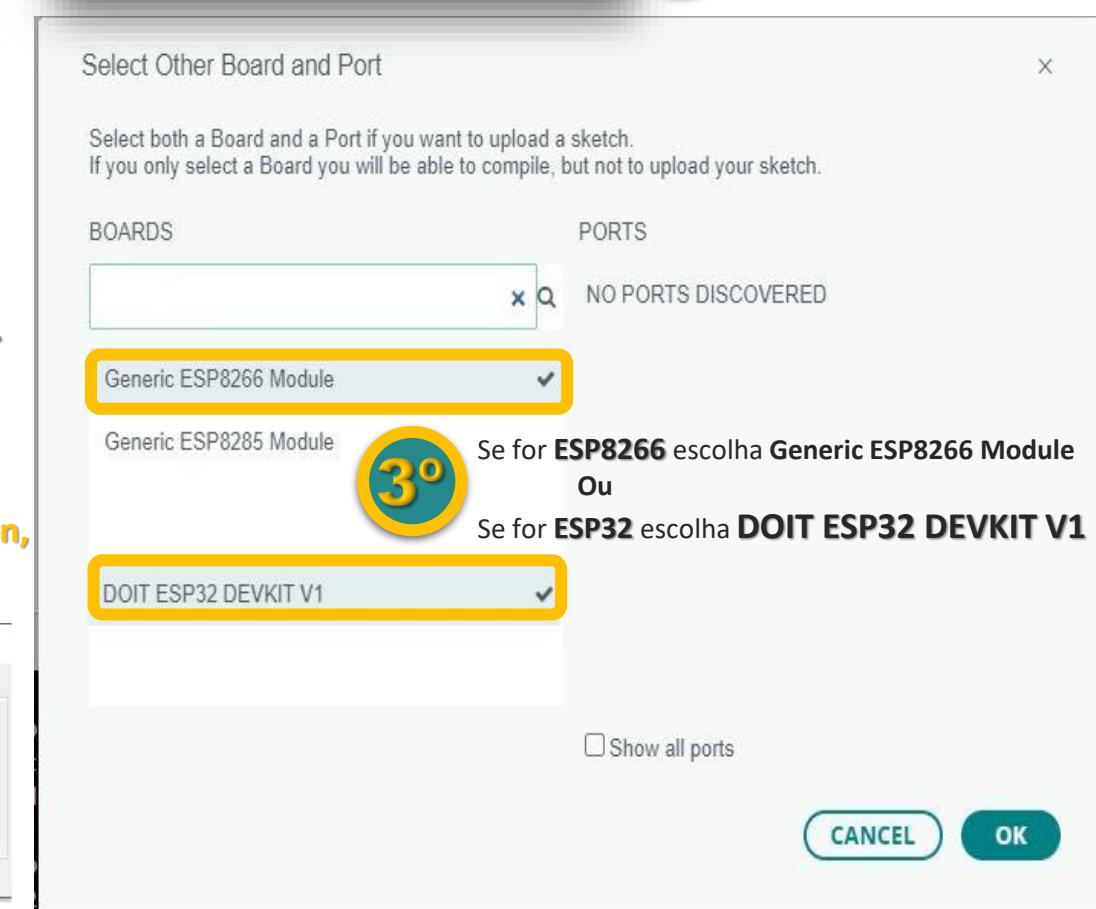
### Configurando o Arduino IDE para programar o ESPXX

Antes de começar a programar o ESP8266 e/ou ESP32 é necessário realizar algumas configurações no Arduino IDE. Primeiro deve-se atualizar o Gerenciador de Placas do Arduino IDE, que é bem simples. Para isso, abra o Arduino IDE e vá para **Arquivo>Preferências**. Em seguida, **copie a URL abaixo na caixa de URLs adicionais para Gerenciadores de Placas**, localizada no inferior da janela.

**1º** [https://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](https://arduino.esp8266.com/stable/package_esp8266com_index.json),  
[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)



**2º** Seleione a Placa corretamente



# DEFINIÇÃO E CONCEITO



# 01 - INTRODUÇÃO



## 01.1 – Variáveis

### Na Linguagem de programação



Em programação, um **identificador** é um nome atribuído a uma **variável** de memória, uma estrutura de dados, uma classe, um objeto, um procedimento, uma função, um comando ou palavra reservada da linguagem.

Variável é tudo que está **sujeito a variações**, que é **incerto, instável ou inconstante**. Quando se fala de **computadores**, é preciso ter em mente que o volume de dados a serem tratados é grande e diversificado. Dessa forma, os **dados a serem processados são bastante variáveis**.

**Todo dado a ser armazenado na memória de um computador deve ser previamente identificado segundo seu tipo**, ou seja, primeiramente é necessário saber o tipo do dado para depois fazer seu armazenamento adequado. **Armazenado o dado, ele pode ser utilizado e processado a qualquer momento**.

#### Regras para identificadores:

Um identificador pode ter até 32 caracteres de comprimento.

❖ O 1º caractere deve ser uma letra do alfabeto.

❖ Os demais caracteres podem ser letras, números ou sinal *underscore* (\_)

❖ Não usar sinais de pontuação, caracteres acentuados, cedilha (ç) ou espaço em branco.

❖ Exemplos de identificadores: **Nome, Nome\_Cliente, Nome\_Completo, Data, Data\_Emissao, Mensagem, Aviso, 1aSemana, Endereco, a, b, c, ..., X, Y, Z;**

## 01.2 – Tipos de Dados na ling. C++ para ARDUINO

bool  
boolean  
byte  
char  
double  
float  
int  
long  
short  
size\_t  
string  
String()  
unsigned char  
unsigned int  
unsigned long  
vetor  
void  
word

Uma variável representa um **contêiner ou espaço na memória** física ou virtual de um computador, onde diferentes tipos de dados (valores) são armazenados durante a execução de um programa. A cada variável é atribuído um nome descritivo ou identificador que se refere ao valor salvo.

A seguir alguns tipos de variáveis usadas na linguagem C++ para sistemas embarcados:

**Int** armazena inteiros (números inteiros), sem decimais, como 123 ou -123

**double** armazena números de ponto flutuante, com decimais, como 19,99 ou -19,99

**char** armazena caracteres únicos, como 'a' ou 'B'. Os valores de caracteres estão entre aspas simples

**string** armazena texto, como "Olá Mundo". Os valores da string estão entre aspas duplas

**bool** armazena valores com dois estados: verdadeiro ou falso

## 01.2) TIPOS DE DADOS na ling. C++ do ARDUINO

TIPO DE DADO	CONSUMO DE MEMÓRIA (Bytes)	INTERVALO DE DADOS	OBSERVAÇÕES
<b>boolean</b>	<b>1</b>	<b>Verdadeiro (True = 1) ou Falso (False = 0)</b>	
<b>char</b>	<b>1</b>	<b>-128 até +128</b>	<b>Usado para representar código de caractere ASCII (Ex.: Caracter A = 65)</b>
byte	1	0 até 255	
<b>int</b>	<b>2</b>	<b>-32.768 até +32.767</b>	
unsigned int	2	0 até 65.536	Usado para ter precisão extra ao tipo int quando não há nºs negativos
long	4	-2147483648, ..., -1, 0, 1, ..., 2147483647	Quando necessita-se representar números muito grandes
<b>unsigned long</b>	<b>4</b>	<b>0 até 4294967295</b>	
float	4	-3.4028235E+38 até +3.4028235E+38	Números com 7 dígitos significativos, Emáx=38
double	4	-3.4028235E+38 até +3.4028235E+38	Excepcionalmente no Arduino, ao contrário do que ocorre na linguagem C++ padrão, o tipo double terá comportamento idêntico ao float
<b>String()</b>	Cria uma instância String <sup>1</sup> a partir de diferentes tipos de dados; esta pode ser usada para armazenar texto/caracteres e pode ser útil para delimitar casas decimais no tipo float.		

<sup>1</sup> Veja item 1.5 – Instância String na página 15 deste livro.

## Operadores Aritméticos

### 1.2.1

### Operadores Aritméticos

Operadores aritméticos: são símbolos especiais que realizam as operações aritméticas básicas

OPERAÇÃO	SÍMBOLO	CÓDIGO EM C++	DESCRIÇÃO
ADIÇÃO	+	$x + y$	Efetua a soma $x + y$
SUBTRAÇÃO	-	$x - y$	Efetua a subtração $x - y$
MULTIPLICAÇÃO	*	$x * y$	Efetua a multiplicação $x * y$
DIVISÃO	/	$x / y$	Efetua a divisão $x / y$
RESTO DA DIVISÃO	%	$x \% y$	Obtém o resto da divisão entre dois inteiros
INCREMENTO	++	$+=$	Incrementa o valor da variável em 1 unidade
DECREMENTO	--	$-=$	Decrementa o valor da variável em 1 unidade

## 1.2.2

### Operadores Relacionais

O símbolo  $=$  é

considerado operador

de atribuição; usado

para atribuir valores

para constantes ou



**Exemplo:**

$X = 2$

$Y = 3$

atribuir cálculos.

$soma = X + Y$

OPERAÇÃO	SÍMBOLO	CÓDIGO EM PYTHON	Descrição
Menor que	<	$x < y$	Verdadeiro se x Menor y
Maior que	>	$x > y$	Verdadeiro se x Maior y
Igual	$==$	$x == y$	Verdadeiro se x Igual a y
ATRIBUIÇÃO	$=$ $\leftarrow$	$X = 2$	Atribui um valor ou resultado de cálculo à uma constante
Diferente	$!=$	$x != y$	Verdadeiro se x Diferente de y
Menor igual	$<=$	$x <= y$	Verdadeiro se x Menor ou Igual a y
Maior igual	$>=$	$x >= y$	Verdadeiro se x Maior ou Igual a y

## 01.3) IMPORTÂNCIA DOS TIPOS DE DADOS na ling. C++ do ARDUINO

Função para calcular o valor do *cosseno de x* a partir da soma das  $n$  primeiras parcelas da série:

O tipo da variável X impacta na  
precisão do resultado!  
Justamente pelo Fatorial!

Quanto mais  
parcelas, mais  
preciso o resultado!

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

Efetuar o cálculo da série utilizando apenas as quatro operações elementares da aritmética:  $+$ ,  $-$ ,  $*$  e  $/$ .

**Observação:** todas as series trigonométricas trabalham com valores em radianos.

**OBS:** No C++ incluo a Biblioteca `#include <cmath>` e uso `pow(x,y)` para calcular Potência  $X^Y$

# CONTROLE DE FLUXO

Os comandos de controle de fluxo são essenciais a qualquer linguagem de programação; já que eles determinam a ordem dos comandos e especificam, numa dada condição, quais destes devem ser executados (comandos **de teste de condição**) e se devem ser repetidos (comandos de **controle de loop**).

CONDICIONAL

- { 1. Testes de condição (ou Estrutura de Decisão): popularmente conhecida como estrutura condicional ou de seleção, realiza diferentes ações dependendo se a condição for verdadeira ou falsa. A condição é uma expressão processada em um valor booleano.

LOOP

- { 2. Controle de loop (ou Estrutura de Repetição): realiza e repete uma ou mais ações dependendo da condição (se verdadeira ou falsa). Condição esta que é processada em uma valor booleano como no teste de condição.

# LOOP FOR

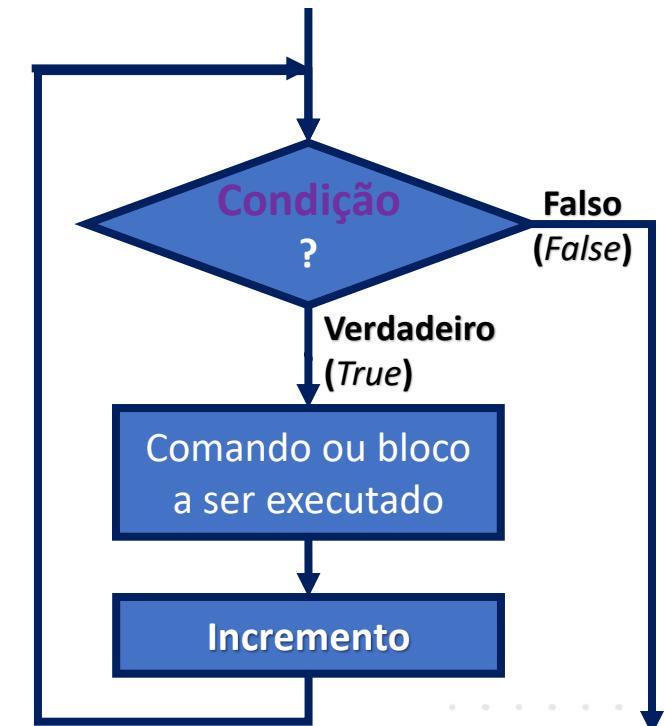
2. Controle de loop (ou Estrutura de Repetição): realiza e repete uma ou mais ações dependendo da condição (se verdadeira ou falsa). Condição esta que é processada em uma valor booleano como no teste de condição.

Necessita de uma **variável de controle** que **define o número de execuções** do laço logico.



**Atenção!**  
A estrutura de repetição **for** realiza essencialmente uma **CONTAGEM NUMÉRICA** (por isso necessita um valor inicial, um valor final da contagem e o passo)

## Fluxograma do Loop FOR:

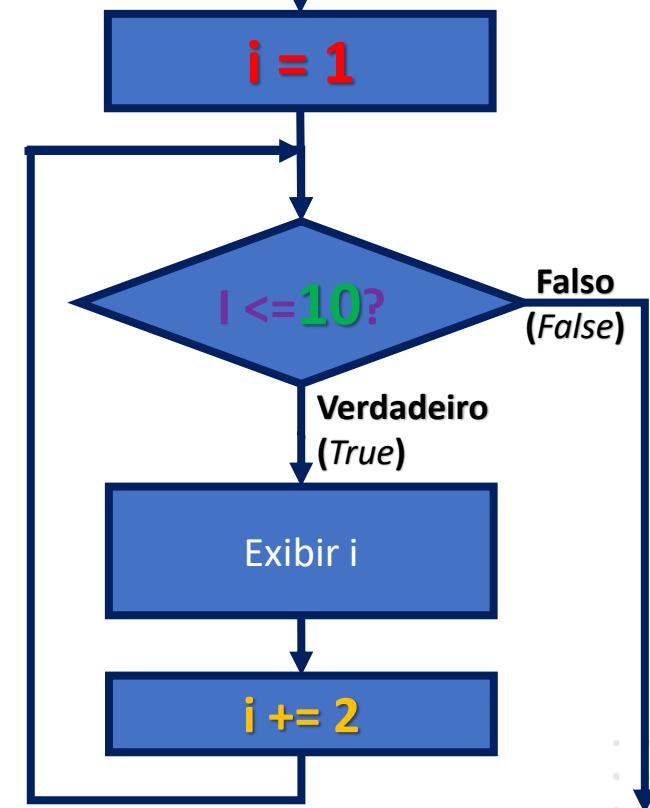


# LOOP FOR

Quando você souber exatamente quantas vezes deseja percorrer um bloco de código, use o loop for em vez de um loop while. Neste exemplo a variável de controle **i** definirá o número de execuções do Loop.

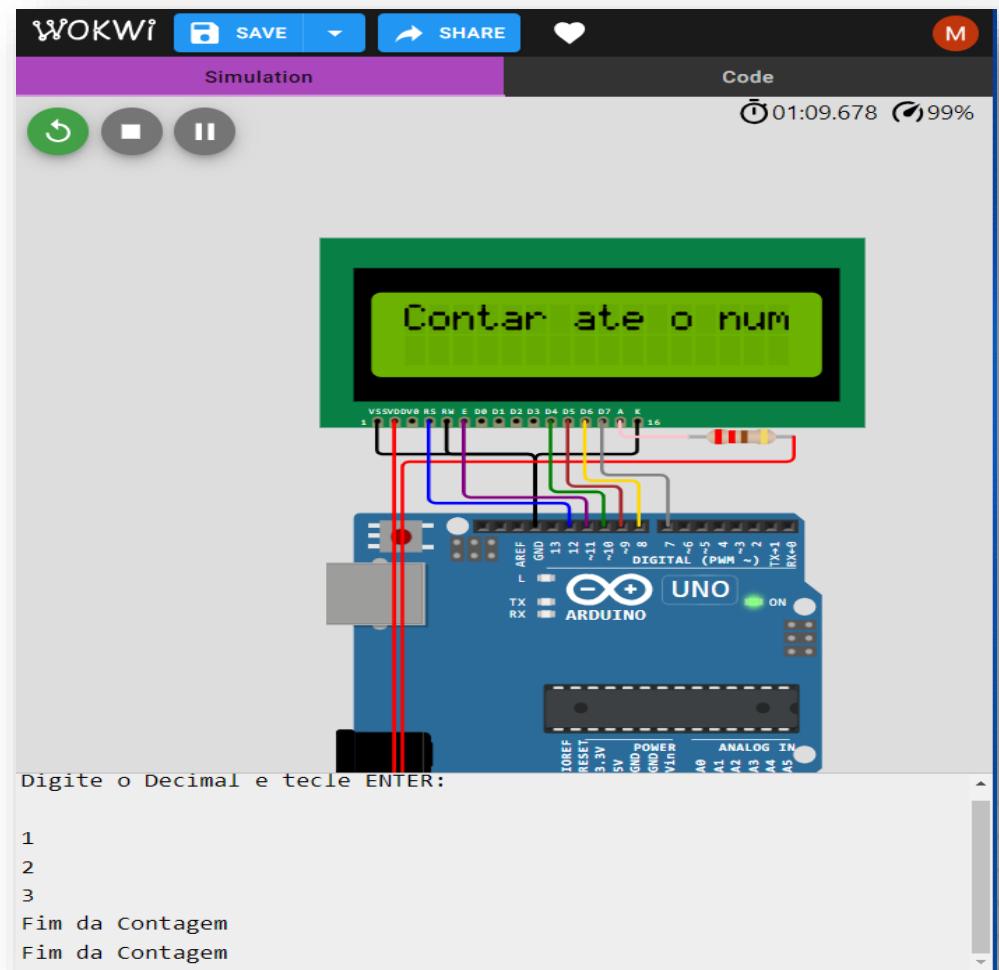
```
for variável (Valor_Inicial, Valor_Final, Passo)
{
    Instrução ou Bloco de instruções;
}
```

Fluxograma p/ contar de 1 até 10 com passo 2:



# Exercício

Agora que você aprendeu a fazer uma contagem usando o LOOP FOR, pede-se a você usar o simulador WOKWI e criar uma simulação com um microcontrolador (pode ser Arduino, EXPxx, etc.) que peça ao usuário digitar, e receber o valor digitado pela porta serial do microcontrolador; em seguida será exibido num display LCD e também escrito via porta serial a contagem de 1 até o número digitado.



# 01.3 – Blink.ino

## Exemplo básico

Blink | Arduino 1.8.18

Arquivo Editar Sketch Ferramentas

Verificar

1º Clicar no Botão Upload  
Ao clicar no botão upload você compilará seu código e em seguida gravará o mesmo na memória flash do microcontrolador da sua placa.

```
/*
Blink
Liga e Desliga o LED padrão da placa no intervalo de 1s
https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/
void setup() {
  //LED_BUILTIN REFERE-SE AO LED PRÉ-INSTALADO NA PLACA
  pinMode(LED_BUILTIN, OUTPUT); //Configura o pino LED_BUILTIN como Saída Digital
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // Ativa saída digital LED em Nível ALTO (HIGH = 1)
  delay(1000); // Aguarda 1s
  digitalWrite(LED_BUILTIN, LOW); // Desativa saída digital LED (LOW = 0)
  delay(1000); // Aguarda 1s
}
```

2º Carregado.

O sketch usa 924 bytes (2%) de espaço de armazenamento para programas. O máximo são 32256 bytes.  
Variáveis globais usam 9 bytes (0%) de memória dinâmica, deixando 2039 bytes para variáveis locais

Arduino Uno em COM4

Sempre testar a sua placa antes de embarcar qualquer novo código:  
para isso programe o código do exemplo **Blink** que se encontra no  
**Menu File > Examples > 01. Basics > Blink**



### PinMode()

Configura o pino especificado como entrada ou saída. Para mais informações veja: [Descrição dos pinos digitais](#). Desde o Arduino 1.0.1 é possível usar o resistor pull-up interno com modo INPUT\_PULLUP para definir. Se definido apenas como modo INPUT desativa completamente o resistor pull-up interno.

#### Sintaxe

`pinMode(Nº_pino, Modo)`

#### Parâmetros

- Nº\_pino: O número do pino do Arduino para ativar o modo de pino.
- Modo: INPUT, OUTPUT ou INPUT\_PULLUP. (veja: [Descrição dos pinos digitais](#))

### digitalWrite()

Envia nível lógico **1 (HIGH)** ou **0 (LOW)** para um pino digital se o pino foi configurado como uma **Saída (OUTPUT)** com a função `pinMode()`; sua voltagem será ajustada para o valor 5V (ou 3,3V em placas de 3,3V) no uso do HIGH e 0V (GND) (LOW). Mas se o pino estiver configurado como um **INPUT**, irá habilitar (**HIGH**) ou desabilitar (**LOW**) o *pullup* interno no pino de entrada. É recomendado definir o `pinMode()` para INPUT\_PULLUP para habilitar o resistor pull-up interno. Veja o tutorial [Digital Pins](#) para mais informações. Se você não definir o `pinMode()` para OUTPUT, e conecte um LED a um pino, ao chamar `digitalWrite (HIGH)`, o LED pode parecer fraco. Sem definir explicitamente `pinMode()`, `digitalWrite()` terá habilitado o resistor pull-up interno, que atua como um grande resistor limitador de corrente.

#### Sintaxe

`digitalWrite(Nº_pino, valor)`

#### Parâmetros

- Nº\_pino: o número do pino do Arduino
- Valor: ALTO = 1 (HIGH) ou Baixo = GND (LOW)

# 6 – A plataforma Thinkercad

## 6.1 – Acessando a plataforma Thinkercad



The image shows two side-by-side screenshots of the Tinkercad platform. On the left, the 'Login | Tinkercad' page is displayed, featuring a blue header with the Tinkercad logo and a central login form. The form includes fields for 'E-mail ou nome de usuário' (Email or user name), 'Entrar com o Google' (Sign in with Google), and 'Fazer login com a Apple' (Log in with Apple). Below the form are links for 'Mais opções de login...' (More login options...) and 'Entrar no Tinkercad' (Log in to Tinkercad). At the bottom, there's a 'Privacy settings' section and a download link for 'download.png'. On the right, the 'Painel | Tinkercad' dashboard is shown, specifically the 'Circuits' section. It features a sidebar with the user's profile picture, a search bar, and links for 'Classes', 'Galeria', 'Blog', 'Aprenda', and 'Ensinar'. The main area displays various circuit projects, each with a thumbnail image, title, and status (e.g., 'Copy of Operador Lógico OR', 'Operador Lógico NOT', etc.).

# 01.4 – Uso de Constantes

Ao definir o Pino e Estado do Botão



1.4 - Uso de Constantes para acender LED com botão

Todas as alterações salvas

Código Iniciar simulação Enviar para 1 (Arduino Uno R3)

**const**

O identificador **const** altera o comportamento da variável para "somente leitura". Se você atribuir um novo valor à variável, receberá um erro do compilador.

**delay()**

Interrompe o programa pelo tempo especificado como parâmetro (em milissegundos). (Há 1000 milissegundos em um segundo.)

**Sintaxe** **delay(ms)**

**Parâmetro** **ms:** o número de milissegundos para pausar.

```
const int BOTAO = 7; //Define a constante BOTAO (tipo inteiro) no pino D7
const int LED = 10; //Define a constante ledPin tipo inteiro no pino D10

int estadoBOTAO = 0; //Variável p/ armazenar estado do botão (ligado/desligado)

void setup()
{
    pinMode(LED, OUTPUT); //Define LED (pino 10) como saída
    pinMode(BOTAO, INPUT); //Define buttonPin (pino 7) como entrada
}

void loop()
{
    estadoBOTAO = digitalRead(BOTAO); //Lê valor do BOTAO e armazena em estadoBOTAO
    if (estadoBOTAO == HIGH) { //Se estadoBOTAO for igual a HIGH ou 1
        digitalWrite(LED, HIGH); //Define LED como HIGH, ligando o LED
        delay(100); //Intervalo de 100 milissegundos
    }
    else { //Senão = estadoButton for igual a LOW ou 0
        digitalWrite(LED, LOW); //Define ledPin como LOW, desligando o LED
    }
}
```

Monitor serial

# 01.5 – Instância **String()**

## Muito mais que um tipo de dado

Cria uma instância tipo String. Existem várias versões que criam strings a partir de diferentes tipos de dados (ou seja, formatam-nas como strings).

### Sintaxe

```
1 String(val)
2 String(val, Precisão_Decimal)
2 String(val, base)
```

### Parâmetro

- **val**: Uma variável que pode ser usada como String deve ser formatada. Tipos de dados permitidos: string, char, byte, int, long, unsigned int, unsigned long, float, double.
- **Precisão\_Decimal**: Somente se para variáveis tipo float ou double): As casas decimais desejadas.
- **base**(Opcional): A base na qual formatar um valor inteiro.

### Valor de retorno

Uma instância de String

# 01.5.1 – INSTÂNCIA STRING():

## definindo a precisão de casas decimais

### String()\*

Cria uma instância String exibindo o valor em questão com a **precisão** fornecida na chamada da função (**OBS:** este valor precisa ser do tipo float ou double).

#### Sintaxe

```
String(val, Precisao)
```

#### Parâmetros

**Val:** valor a ser convertido para tipo string de dado.

**Precisao:** Quantidade de casas decimais de precisão.

### Serial.Println()\*

Imprime dados na porta serial como texto ASCII legível, seguido por um retorno de carro (ASCII 13 ou '\r') e um caractere de nova linha (ASCII 10 ou '\n'). Este comando tem o mesmo formato que [Serial.print\(\)](#).

#### Sintaxe

```
Serial.println(val)
```

```
Serial.println(val, Precisao)
```

#### Parâmetros

**Val:** valor a ser convertido para tipo string de dado.

**Precisao:** Quantidade de casas decimais de precisão.

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** 1.5.1 - Instância String()
- Toolbar:** Includes icons for file operations (New, Open, Save, Print, Find, Copy, Paste, Undo, Redo), a text editor, and simulation controls (Start Simulation, Send to).
- Code Editor:** Displays the following sketch:

```
1 #include <math.h> //Inclui a biblioteca Math.h para acessar a constante PI
2 float piValue = PI;
3 String ValorPI = "";
4
5 void setup() {
6   Serial.begin(9600); //Configura porta serial para velocidade 9600 bps
7   ValorPI = String(piValue, 2); //Uso da Instância String para formatar a Variável piValue tipo Float com 3 casas decimais
8   Serial.print("Valor de Pi com 2 casas decimais usando String()= ");
9   Serial.println(ValorPI); //Exibição do Valor usando a função println
10  Serial.print("Valor de Pi com 3 casas decimais usando Serial.print= ");
11  Serial.println(piValue, 3); //Uso da função println para formatar a Variável piValue tipo Float com 3 casas decimais
12 }
13 void loop() {
14   //Nesta simulação optamos por inserir o código no setup para que seja executado apenas uma única vez ao inicializar
15 }
```
- Monitor Serial:** Shows the output of the serial port:

```
Valor de Pi com 2 casas decimais usando String()= 3.14
Valor de Pi com 3 casas decimais usando Serial.print= 3.142
```
- Bottom Buttons:** Includes buttons for Env., Apag., and a plot icon.

# 01.5.2 – INSTÂNCIA **STRING()**:

## conversão entre bases numéricas

### **String()\*\***

Cria uma instância String convertendo o valor em questão na **base** fornecida na chamada da função.

#### Sintaxe

**VarRetorno** = String(**val**, **base**)

#### Parâmetros

**VarRetorno**: variável que armazenará o retorno da conversão do valor em questão na base desejada.

**Val**: valor a ser convertido para base desejada. O retorno será do tipo string de dado.

**base**: base de destino da conversão do valor fornecido na chamada da função. A base de destino pode ser:

**BIN** = Binária,

**HEX** = Hexadecimal

**OCT** = Octal

**DEC** = Decimal

### **char()**

Converte um valor para tipo char.

#### Sintaxe

char(**val**)

#### Parâmetros

**Val**: valor a ser convertido para tipo char.

**OBS**: Caso o valor seja inteiro o char em questão corresponderá ao caractere ASCII de referência ao número inteiro decimal em questão (veja tabela ASCII).

The screenshot shows the Arduino IDE interface with the title "1.5.2 – Instância String()". The code in the editor is as follows:

```
1 int num; //Variável num tipo inteira para armazenar dado digitado
2 String binaryString = ""; //Variável tipo String para armazenar binário correspondente
3 String hexString = ""; //Variável tipo String para armazenar hexadecimal correspondente
4 String octString = ""; //Variável tipo String para armazenar octal correspondente
5 String ASCII = ""; //Variável tipo String para armazenar ASCII correspondente
6 void setup() {
7   Serial.begin(9600); //Configura porta serial para velocidade 9600 bps
8   Serial.print("Digite numero Decimal e tecle ENTER: \n");
9 }
10 void loop() {
11   if (Serial.available() > 0) {
12     num = Serial.parseInt(); //A função parseInt recebe e armazena numa variável tipo inteiro o número enviado a ela
13     Serial.print("Para o num digitado ");
14     Serial.println(String(num)); //Converte e exibe o número inteiro digitado como tipo String de dados
15     binaryString = String(num, BIN); //Converte o num digitado para o valor BINÁRIO correspondente
16     hexString = String(num, HEX); //Converte o num digitado para o valor HEXADECIMAL correspondente
17     octString = String(num, OCT); //Converte o num digitado para o valor OCTAL correspondente
18     ASCII = char(num);
19     Serial.print("Num em Binario = \t\t");
20     Serial.println(binaryString); //Exibe o valor BINÁRIO correspondente armazenado na variável binaryString
21     Serial.print("Num em Hexadecimal = \t\t");
22     Serial.println(hexString); //Exibe o valor HEXADECIMAL correspondente armazenado na variável hexString
23     Serial.print("Num em OCTAL = \t\t");
24     Serial.println(octString); //Exibe o valor OCTAL correspondente armazenado na variável octString
25     Serial.print("ASCII correspondente a num = \t");
26     Serial.println(ASCII); //Exibe o ASCII correspondente armazenado na variável ASCII
27     delay(2000);
28     Serial.print("\033[2J\033[H"); //Seqüencia ANSI enviada para porta serial limpar o buffer dela
29 }
```

The "Monitor serial" window shows the output:

Para o num digitado 248  
Num em Binario = 11111000  
Num em Hexadecimal = f8  
Num em OCTAL = 370  
ASCII correspondente a num = Ⓛ

# CÓDIGOS ALFANUMÉRICOS

## Código ASCII

O *American Standard Code for Information Interchange - ASCII*, é um conjunto de códigos que representam caracteres alfanuméricos e símbolos utilizados na comunicação de dados em computadores e dispositivos eletrônicos. Criado na década de 1960, o ASCII é amplamente utilizado até os dias atuais como padrão para a codificação de texto em sistemas de computação.

**Foi desenvolvido** pela American National Standards Institute (ANSI) **para padronizar a representação de caracteres em computadores**. Antes da criação do ASCII, cada fabricante de computador utilizava seu próprio conjunto de códigos para representar caracteres, o que dificultava a **interoperabilidade entre sistemas**.

O ASCII é composto por 128 códigos diferentes, que vão desde letras maiúsculas e minúsculas até números, pontuações e símbolos especiais. Cada caractere é representado por um número decimal de 7 bits, o que permite a sua codificação e transmissão de forma eficiente em sistemas de computação.

O ASCII desempenha um papel fundamental na Internet, sendo utilizado para representar texto em páginas da web, e-mails, arquivos de texto e outros tipos de dados. Mesmo com o surgimento de padrões mais avançados de codificação de caracteres, como o Unicode, o ASCII ainda é amplamente utilizado devido à sua simplicidade e compatibilidade com sistemas legados.

Além dos 128 caracteres padrão do ASCII, existem versões estendidas que incluem caracteres adicionais, como letras acentuadas, símbolos matemáticos e outros caracteres especiais.

# CÓDIGOS ALFANUMÉRICOS

## Código ASCII

ASCII Art	ASCII NA PROGRAMAÇÃO
<p>Uma forma criativa de utilizar o ASCII é na criação de arte digital, conhecida como ASCII art. Nessa técnica, os caracteres do ASCII são combinados de forma a criar imagens e ilustrações, que podem ser visualizadas em terminais de texto e páginas da web. O ASCII art é uma forma única de expressão artística que remonta aos primórdios da computação.</p>	<p>Na programação de computadores, o ASCII é frequentemente utilizado para representar caracteres em linguagens de programação, arquivos de código-fonte e comunicação entre sistemas. Os códigos ASCII são facilmente convertidos em valores numéricos que podem ser manipulados e processados por algoritmos e programas de computador.</p>

### Compatibilidade do ASCII com Unicode

O ASCII é compatível com o Unicode, um padrão mais abrangente de codificação de caracteres que suporta múltiplos idiomas e alfabetos. Os primeiros 128 caracteres do Unicode correspondem aos códigos ASCII padrão, o que facilita a conversão e a interoperabilidade entre os dois padrões de codificação.

### Vantagens e Desvantagens do ASCII

Uma das principais vantagens do ASCII é a sua simplicidade e eficiência na representação de caracteres alfanuméricos em sistemas de computação. No entanto, o ASCII possui limitações em relação à representação de caracteres não pertencentes ao alfabeto inglês, o que levou ao desenvolvimento de padrões mais avançados, como o Unicode, para suportar idiomas e alfabetos diversos.

# CÓDIGOS ALFANUMÉRICOS

## Código ASCII

### ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

# CÓDIGOS ALFANUMÉRICOS

## Código ASCII



[https://bit.ly/ASCII\\_arduino](https://bit.ly/ASCII_arduino)

```
ExemploASCII
Todas as alterações salvas
Código Iniciar simulação Enviar para
1 (Arduino Uno R3)
Texto
22 // prints title with ending line break
23 Serial.println("ASCII Table ~ Character Map");
24 }
25 // first visible ASCII character '!' is number 33:
26 int thisByte = 33;
27 // you can also write ASCII characters in single quotes.
28 // for example, '!' is the same as 33, so you could also use this
29 // int thisByte = '!';
30 void loop() {
31 // prints value unaltered, i.e. the raw binary version of the k
32 // The Serial Monitor interprets all bytes as ASCII, so 33, the
33 // ...
Monitor serial
j, dec: 106, hex: 6A, oct: 152, bin: 1101010
k, dec: 107, hex: 6B, oct: 153, bin: 1101011
l, dec: 108, hex: 6C, oct: 154, bin: 1101100
m, dec: 109, hex: 6D, oct: 155, bin: 1101101
n, dec: 110, hex: 6E, oct: 156, bin: 1101110
o, dec: 111, hex: 6F, oct: 157, bin: 1101111
```

<https://docs.arduino.cc/built-in-examples/communication/ASCIITable/>