



**PUC Minas**

# Desenvolvimento e Infraestrutura

Prof. Ilo Rivero



**PUC Minas**

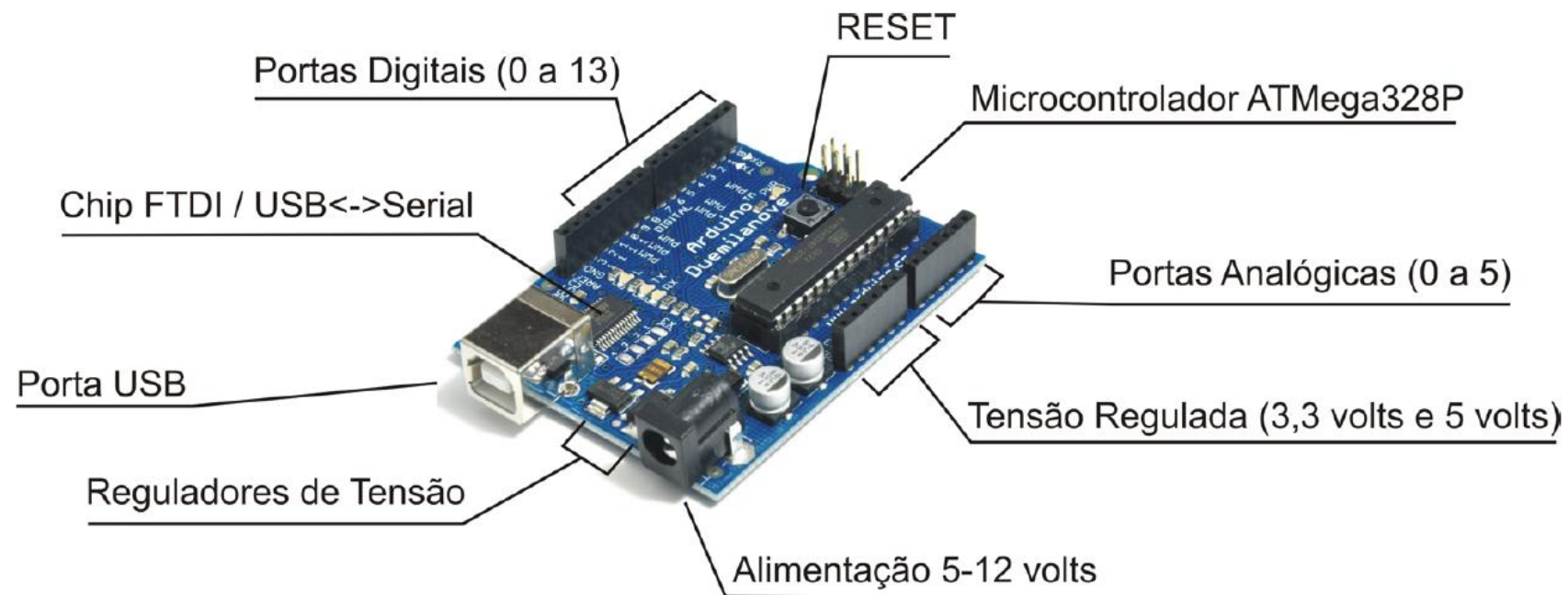
# Plataforma Arduino

Prof. Ilo Rivero

- Plataforma de prototipagem rápida
  - É baseado em hardware livre
  - Pode ser modificado sem o pagamento de royalties
- Utiliza o conceito de Shields:
  - Placas podem ser adicionadas para aumentar a funcionalidade



- **Plataforma *Arduino***

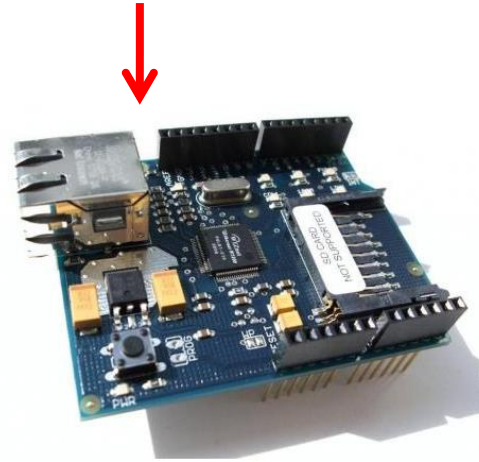


- **Plataforma *Arduino***

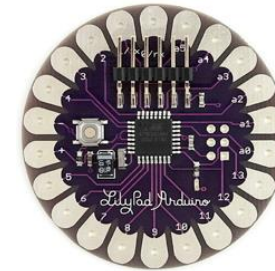


Arduino Uno

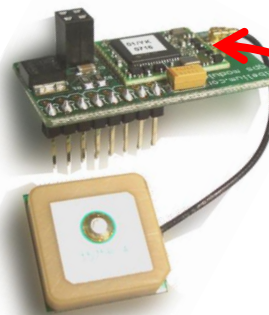
Módulo Ethernet



LilyPad - Wearable



Módulo GPRS

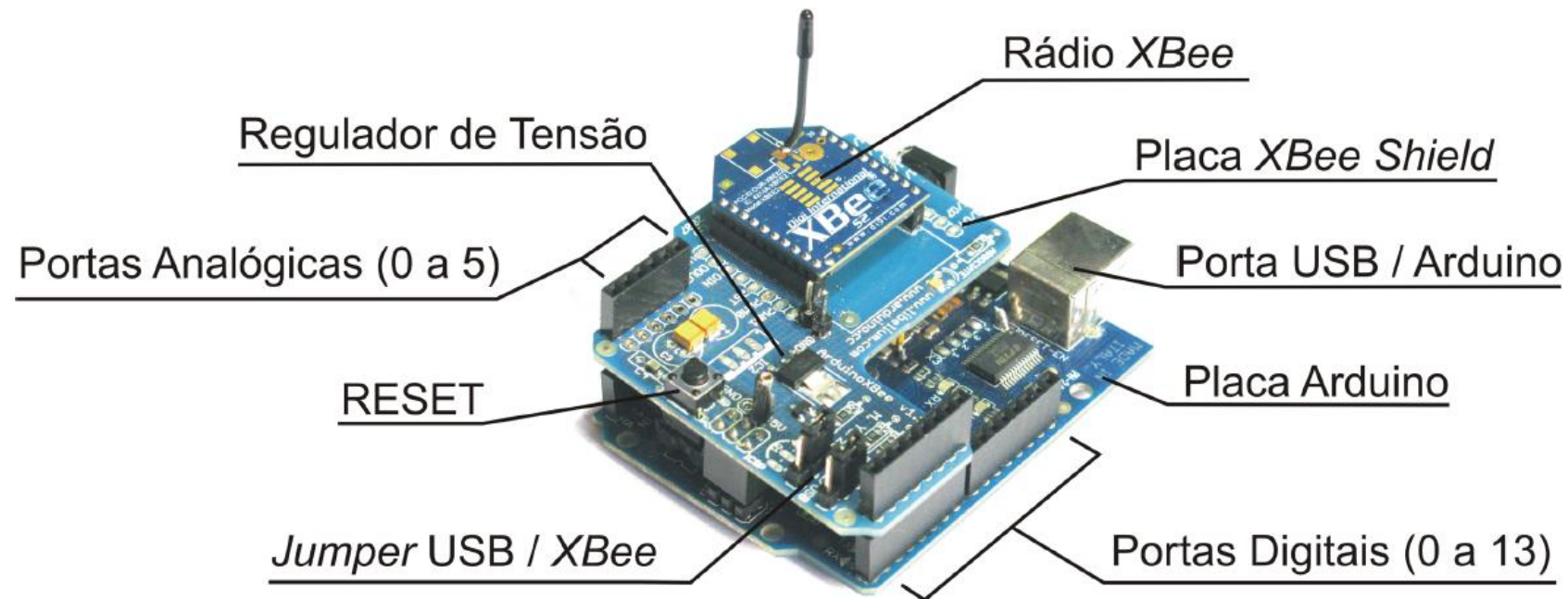


Módulo GPS

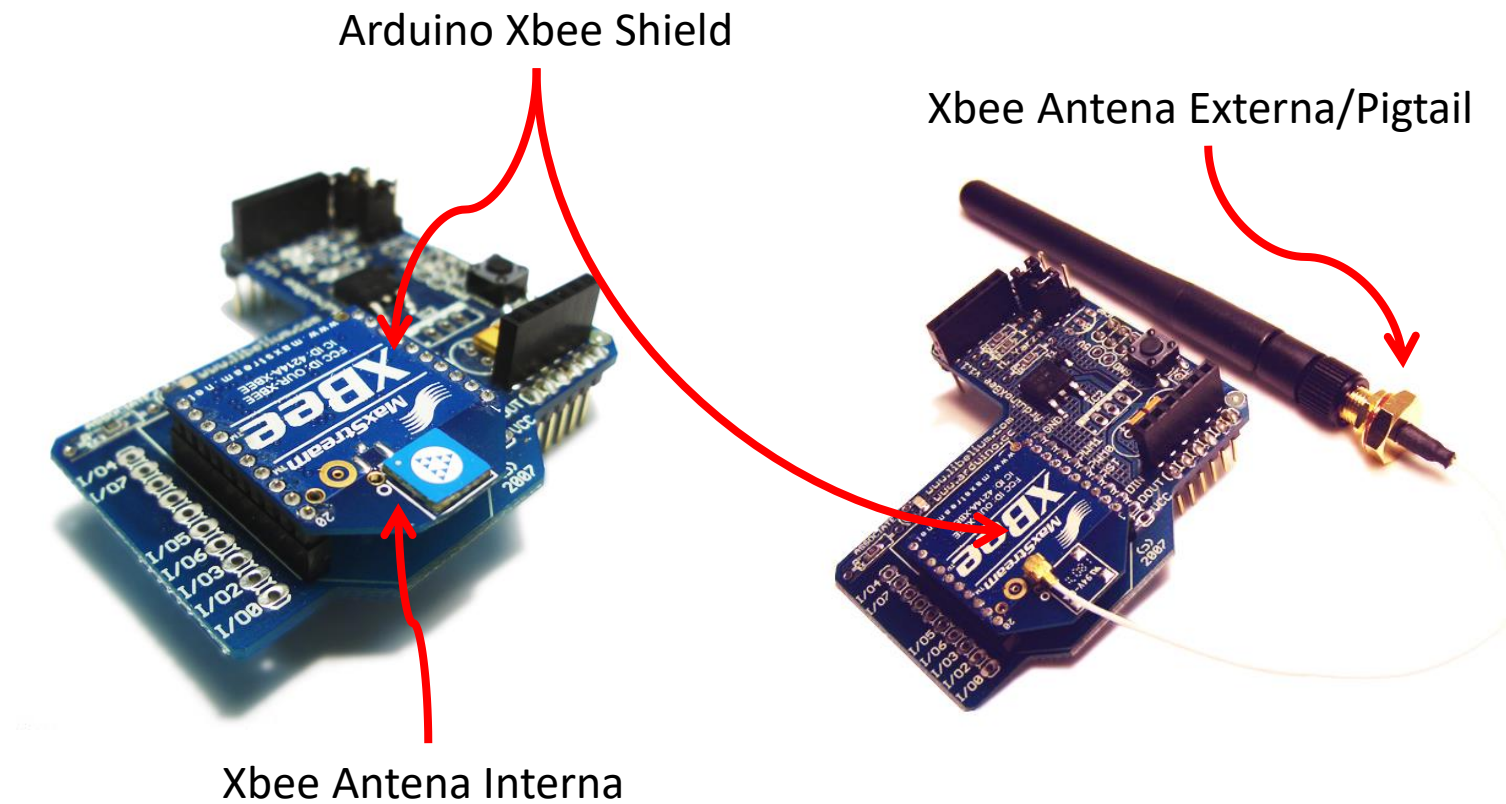


Arduino Bluetooth

- **Plataforma *Arduino* com *XBee Shield***



- **Plataforma *Arduino* com *XBee Shield***





[HARDWARE](#) [SOFTWARE](#) [CLOUD](#) [DOCUMENTATION](#) [COMMUNITY](#) [BLOG](#) [ABOUT](#)

[Summary](#)  
[Entry Level](#)  
[Enhanced Features](#)  
[IoT](#)  
[Education](#)  
[Retired](#)







## Arduino Products

Browse the full range of official Arduino products, including Boards, Modules (a smaller form-factor of classic boards), Shields (elements that can be plugged onto a board to give it extra features), and Kits.

If you are wondering if your Arduino product is authentic you can [learn how to spot a counterfeit board here](#).

### Entry Level

Get started with Arduino using Entry Level products: easy to use and ready to power your first creative projects. These boards and modules are the best to start learning and tinkering with electronics and coding. The StarterKit includes a book with 15 tutorials that will walk you through the basics up to complex projects.





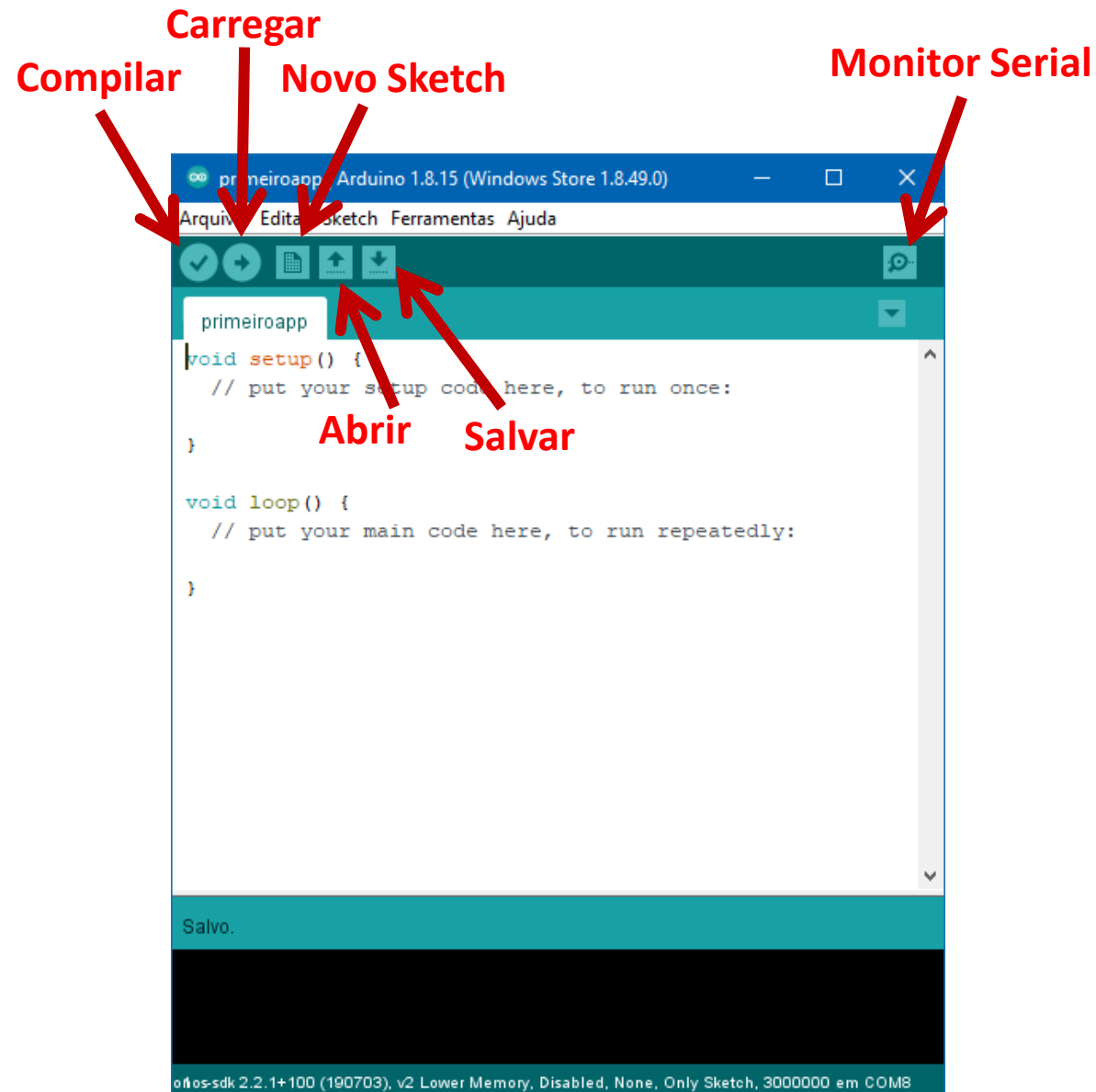
- O Arduino utiliza uma variação da linguagem C
- Possui duas estruturas básicas:

### SETUP

- Responsável por inicializar parâmetros do microcontrolador

### LOOP

- Permite a execução das rotinas ciclicamente



```

#define toleranciaExterno 00
int nivelDC = hibernacao, nHibern = 0;
const int idLum = 0x01;
volatile boolean f_wdt=1;
long tempo;
#define portaMov 3
#define portaLed 3
int movimento;
void setup(){
  Serial.begin(57600);
  Serial.println(" - configurando - ");
  delay (10);
  pinMode(portaMov, INPUT);
  pinMode(portaLed, OUTPUT);}
void loop(){
  movimento=analogRead(portaMov);
  Serial.println(movimento);
  piscaLed();
}
void piscaLed() {
  digitalWrite(portaLed, HIGH);
  delay(100);
  digitalWrite(portaLed, LOW);
    
```

- Microcontrolador ATmega328:
  - Possui 32Kb de memória
  - Arquitetura RISC
  - Velocidade de 16Mhz
  - De fácil implementação



## Portas Analógicas

- As portas analógicas lêem valores em milivolts e convertem esses valores em um intervalo entre 0 e 1023
- Conversor Analógico Digital de 10 bits

## Portas Digitais

- As portas digitais lêem e escrevem valores 1 (ligado) ou 0 (desligado)
- Também atuam como portas moduladas por largura de pulso, com valores de 0 a 255, denominado PWM

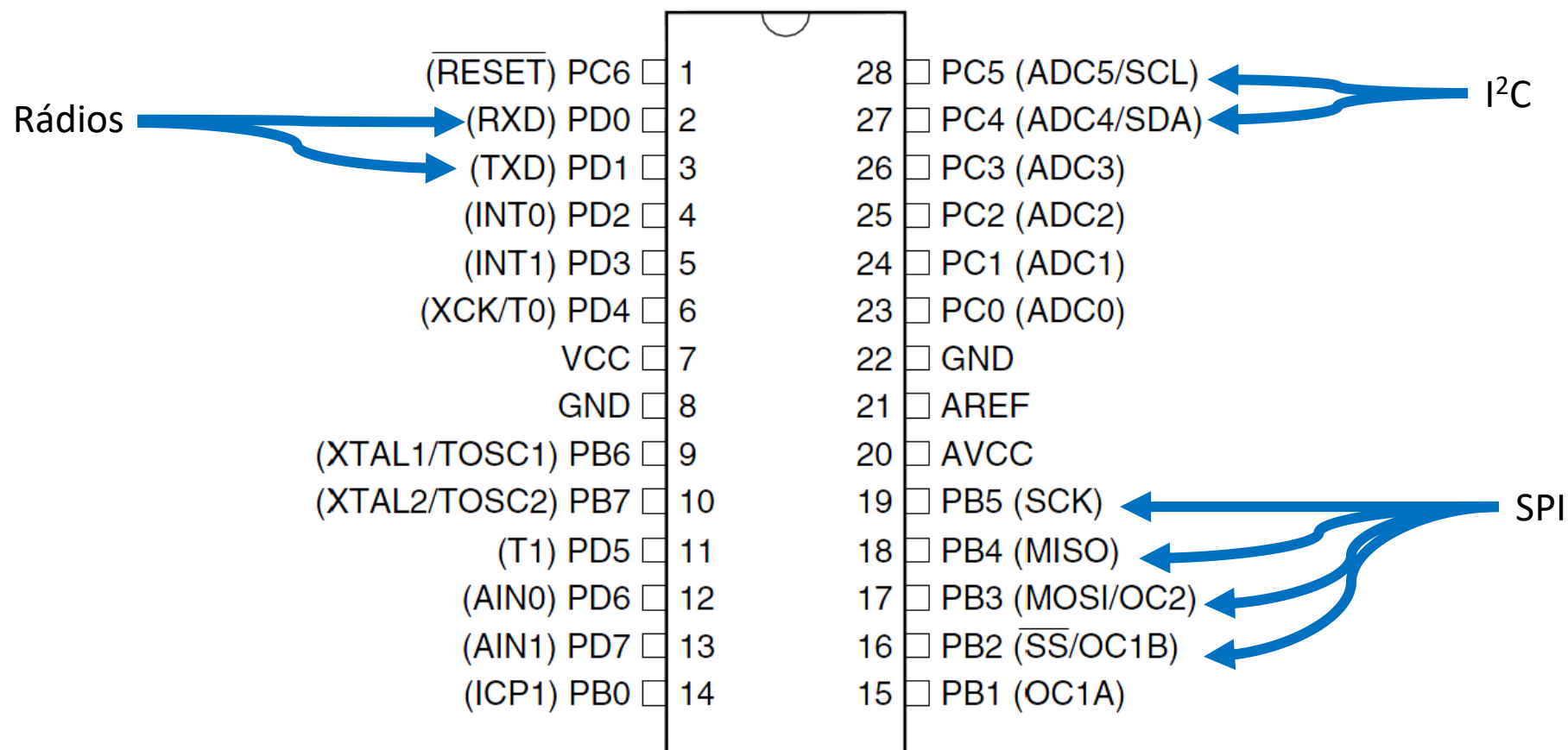
## Barramento I<sup>2</sup>C

- Inter-Integrated Circuit
- Utiliza 1 fio para sincronismo (SCL) e 1 fio para dados (SDA), além da alimentação VCC e GND
- Permite a conexão de vários dispositivos sensores na mesma porta de conexão nas placas de desenvolvimento

## SPI

- Serial Peripheral Interface
- Comunicação serial com 4 fios: CS, SCLK, MOSI e MISO
- Implementação mais complexa do que o I<sup>2</sup>C

- Pinagem do Microcontrolador ATmega 328



- Por exemplo, ao ler o valor da umidade do ar, o sensor retorna o valor 750 ao Arduino
- Esse valor é convertido pela fórmula:

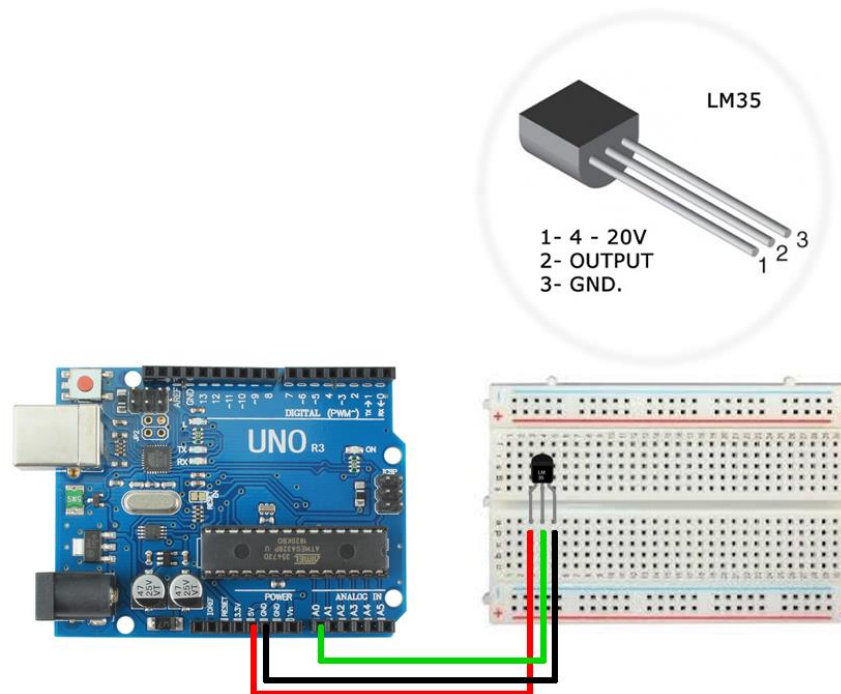
$$\frac{(\text{valor medido} * 5 / 1024)}{X}$$
$$32.25 - 25.81$$

- O que dá o valor da umidade em 89,1%
- Normalmente essa fórmula é fornecida pelo fabricante do sensor e depende do nível de tensão do Arduino



- Para conhecer a temperatura, o valor é convertido pela fórmula:

$$\text{valor medido} * ( \text{tensão} * 10 / 1024 ) - 500 ) / 10$$





**PUC Minas**

# Programando com Arduino IDE

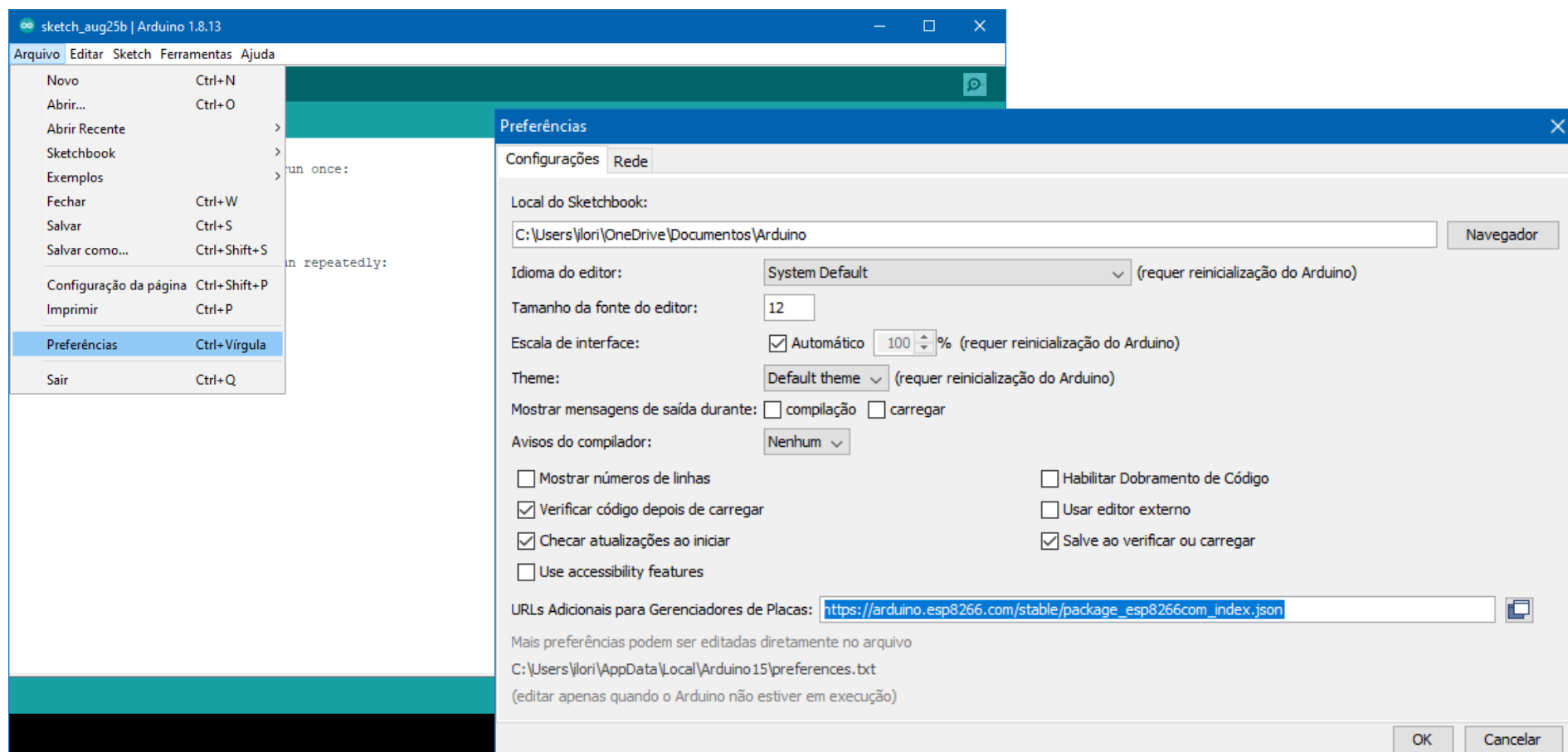
Prof. Ilo Rivero

# Programando o ESP32 / 8266

- Vá em Arquivo -> Preferências e adicione as linhas abaixo em URL Adicionais de Placas

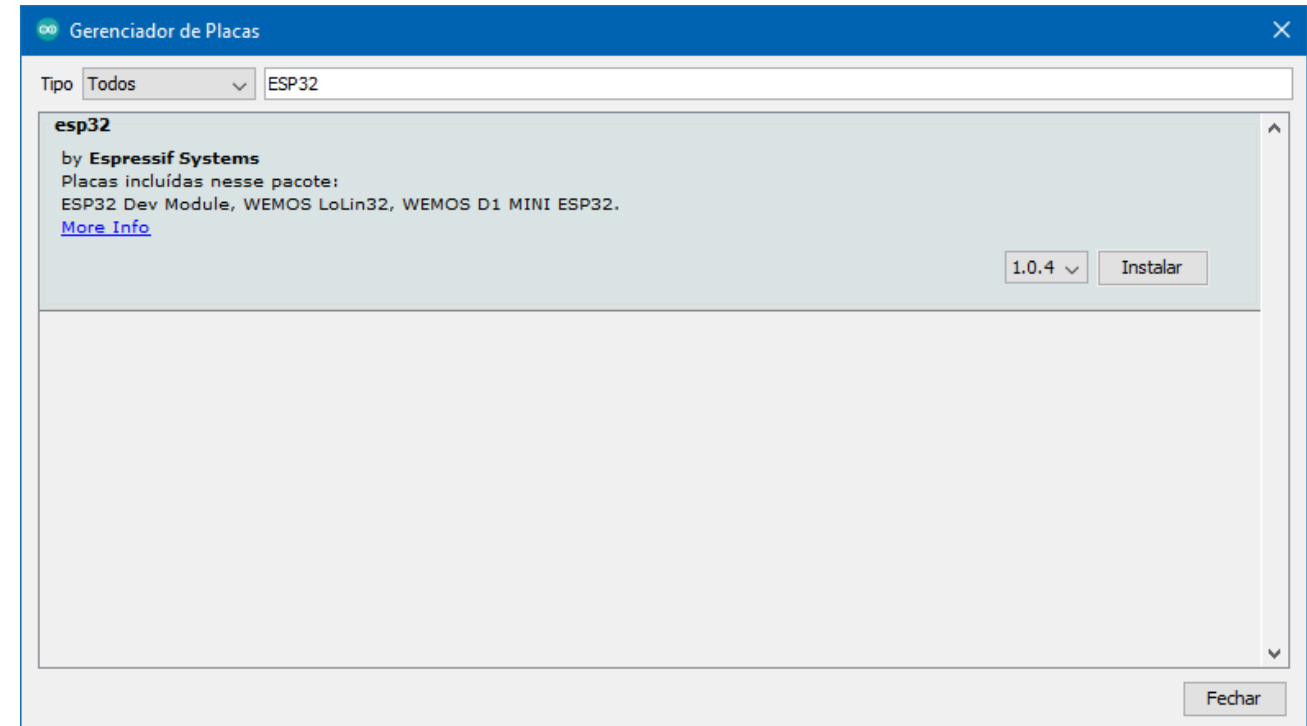
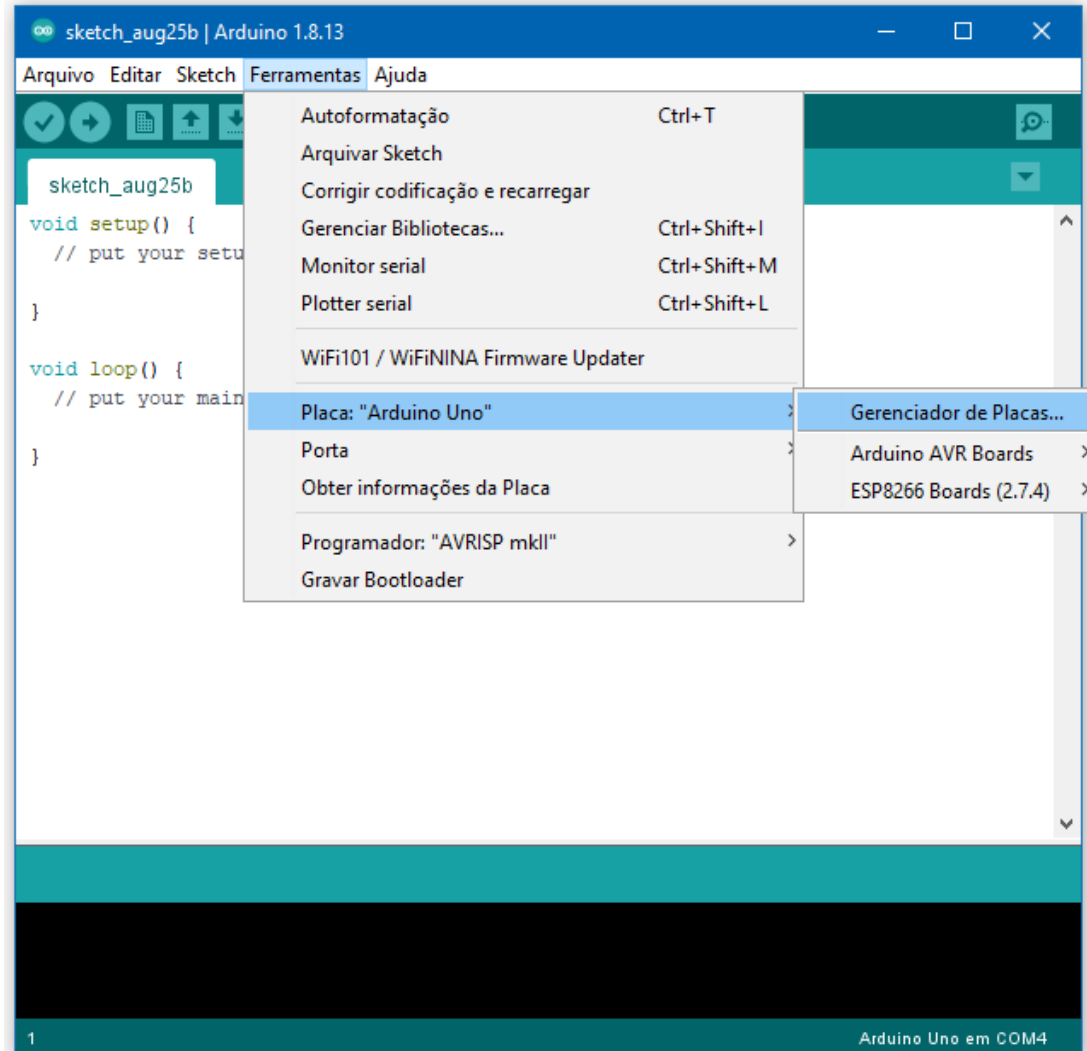
ESP32: [https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)

ESP8266: [https://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](https://arduino.esp8266.com/stable/package_esp8266com_index.json)



# Programando o ESP32 / 8266

- Vá em Ferramentas -> Placas -> Gerenciador de Placas -> Digite (no caso) ESP32, clique em instalar e fechar



- Testando a Conexão

```
#include <ESP8266WiFi.h>
```

```
//Configuração da Rede Wifi
const char* ssid = "AulaloT";
const char* senha = "secreta123";
```

```
void setup() {  
  Serial.begin(115200);  
  delay(10);  
}
```

```
Serial.print("Conectando na rede ");  
Serial.println(ssid);
```

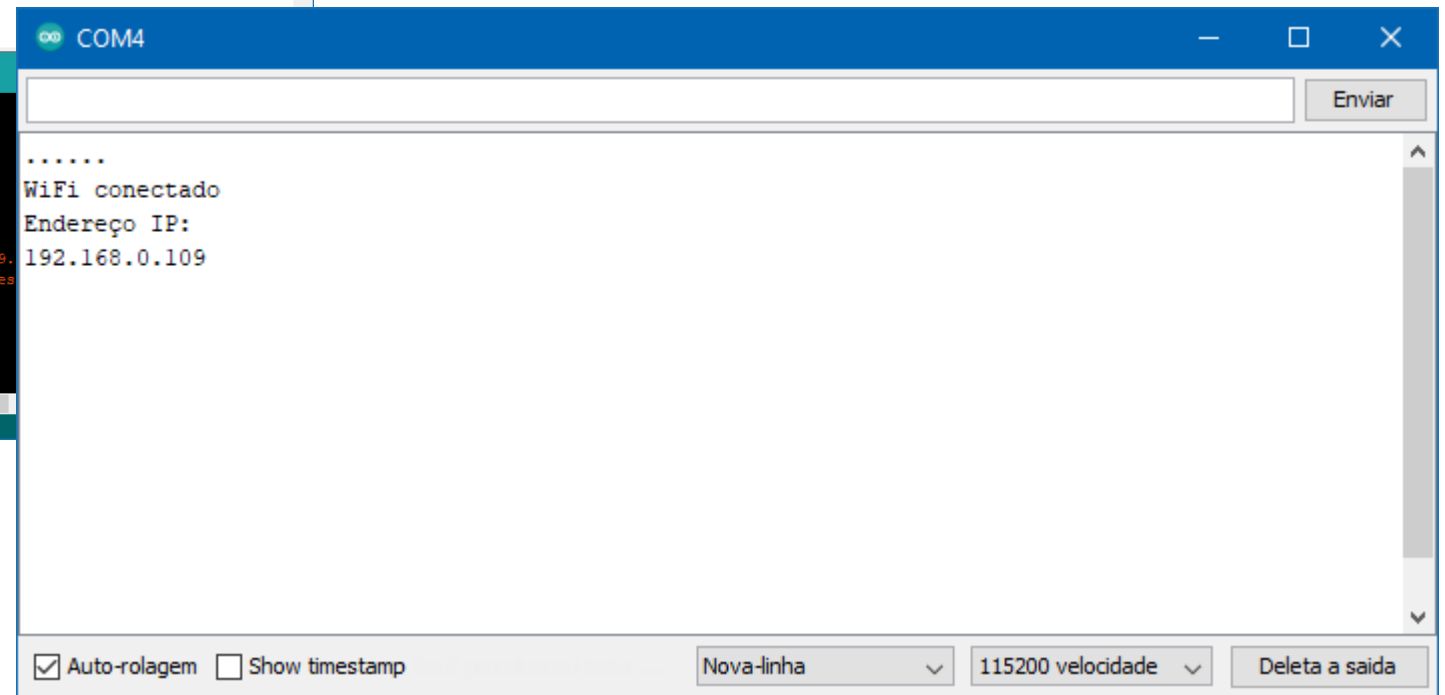
```
// Iniciando a conexão WiFi
WiFi.begin(ssid, senha);
```

```
// Aguarda a conexão WIFI
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
```

```
// Tudo certo na conexão – anote o endereço IP
Serial.println("");
Serial.println("WiFi conectado");
Serial.println("Endereço IP: ");
Serial.println(WiFi.localIP());
}
```

```
void loop(){
```

```
}
```



- Detectando um cliente e exibindo uma página web

```
#include <ESP8266WiFi.h>
```

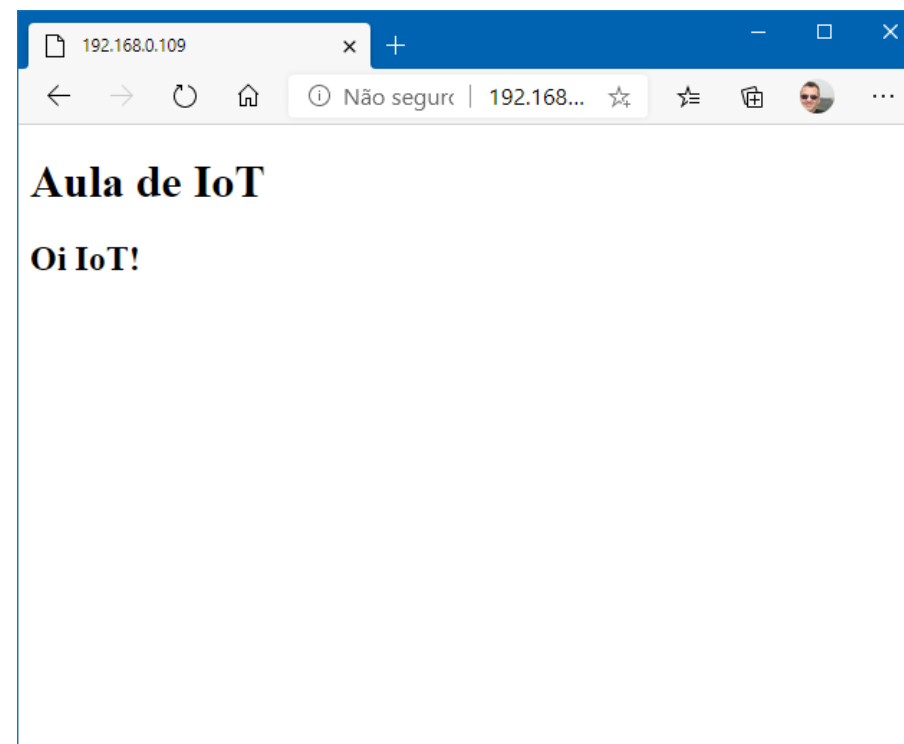
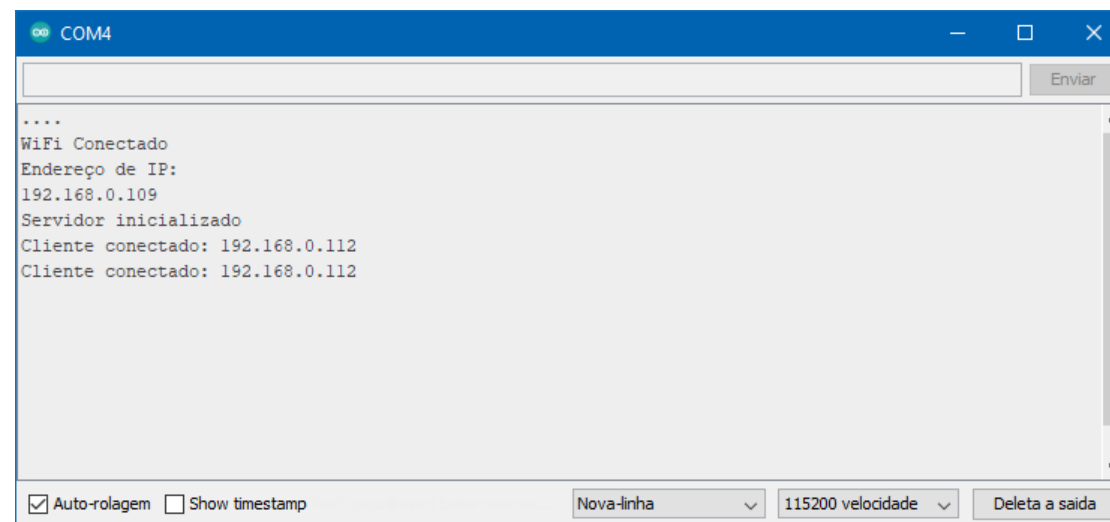
```
char ssid [] = "AulaloT";  
char senha[] = "secreta123";  
WiFiServer server(80);
```

```
void setup() {  
  Serial.begin(115200);  
  delay(10);  
  conectaWiFi(ssid,senha);  
  server.begin();  
  Serial.println("Servidor inicializado");  
}
```

```
void conectaWiFi(char SSID[],char SENHA[]){  
  Serial.print("Conectando a rede ");  
  Serial.println(SSID);  
  WiFi.begin(SSID,SENHA);  
  while(WiFi.status() != WL_CONNECTED){  
    delay(500);  
    Serial.print(".");  
  }  
  
  Serial.println(" ");  
  Serial.println("WiFi Conectado");  
  Serial.println("Endereço de IP: ");  
  Serial.println(WiFi.localIP());  
}
```

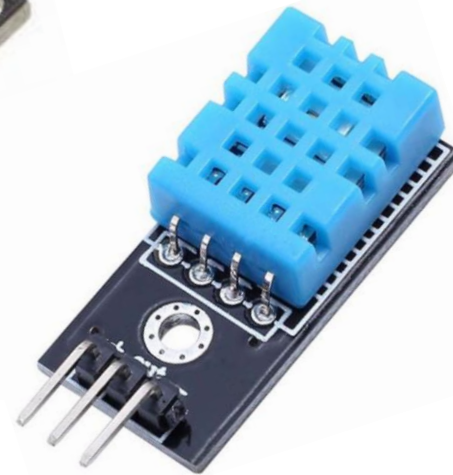
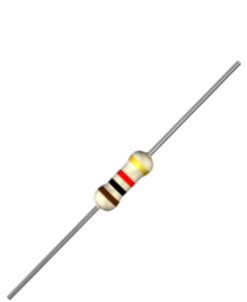


```
void loop() {  
  WiFiClient client = server.available();  
  if(!client){  
    return;  
  } else {  
    String ipCliente = client.remoteIP().toString();  
    Serial.print("Cliente conectado: ");  
    Serial.println(ipCliente);  
  }  
  
  while(!client.available()){  
    delay(1);  
  }  
  
  client.println("HTTP/1.1 200 OK");  
  client.println("Content-Type: text/html");  
  client.println("");  
  client.println("<!DOCTYPE HTML>");  
  client.println("<html>");  
  client.println("<body>");  
  client.println("<h1>Aula de IoT</h1>");  
  client.println("<h2>Oi IoT!</h2>");  
  client.println("</body>");  
  client.println("</html>");  
}
```

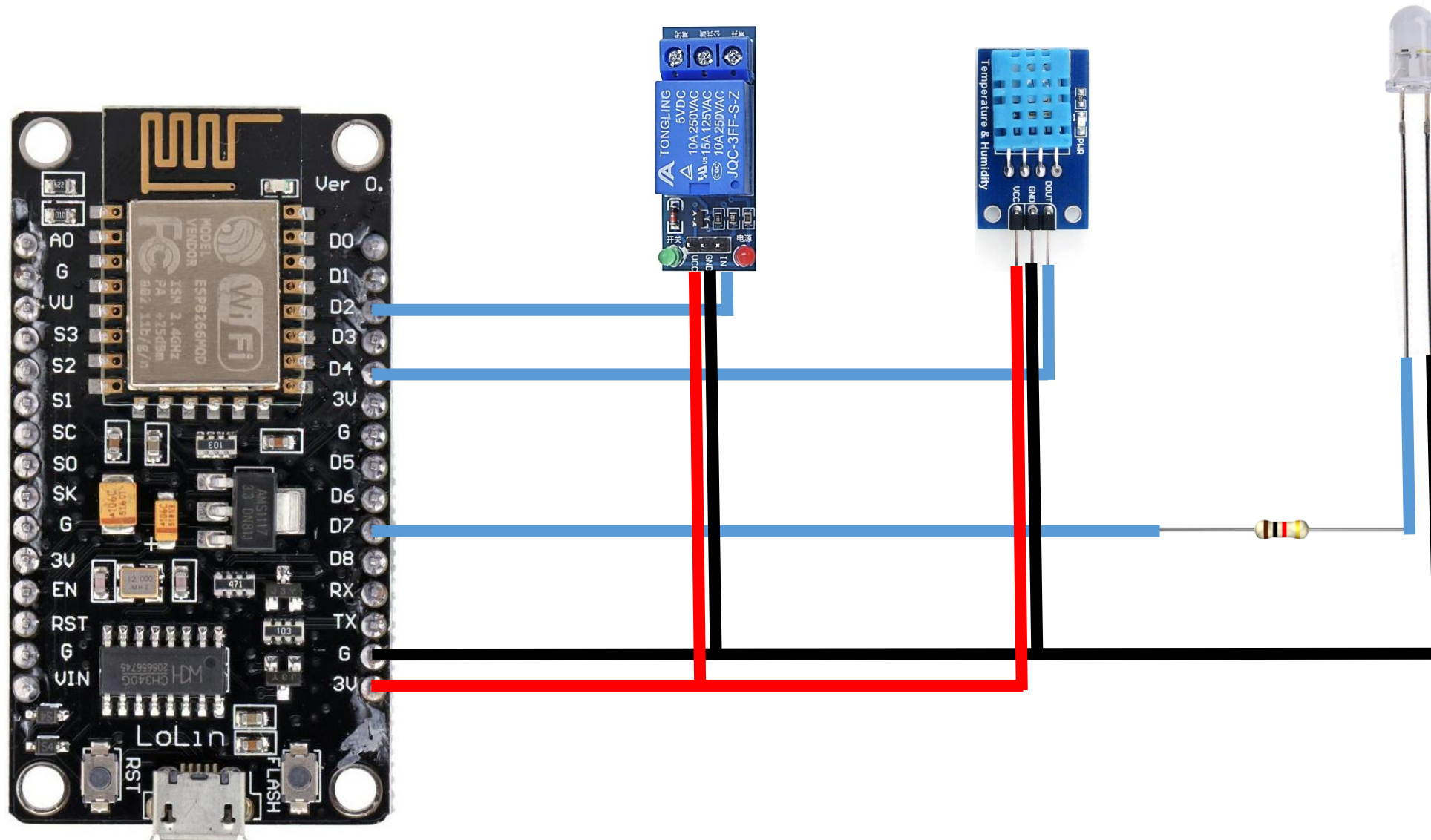


# Programando o ESP32/8266

- Adicionando Sensores e Atuadores
  - Requisitos da Aplicação:
    - Medir temperatura e umidade (sensor DHT11)
    - Ligar e desligar um led
    - Ligar e desligar um relé
    - Mostrar o status do dispositivo em uma página web local



- Adicionando Sensores e Atuadores



- Adicionando Sensores e Atuadores
  - Declarando as portas e sensores

```
#include <ESP8266WiFi.h>
```

```
#include <ESP8266WebServer.h>
```

```
#include "DHT.h"
```

```
#define DHTTYPE DHT11
```

```
uint8_t DHTPin = 2;
```

```
DHT dht(DHTPin, DHTTYPE);
```

```
const char* ssid = "AulaloT";
```

```
const char* senha = "secreta123";
```

```
float temp;
```

```
float umi;
```

```
const int ledPin=13;
```

```
const int relePin=4;
```

```
bool statusLed = false;
```

```
bool statusRele = false;
```

```
ESP8266WebServer server(80);
```

- Adicionando Sensores e Atuadores
  - Inicializando os sensores e servidores

```
void setup() {  
  Serial.begin(115200);  
  pinMode(ledPin, OUTPUT);  
  pinMode(relePin, OUTPUT);  
  pinMode(DHTPin, INPUT);  
  
  dht.begin();  
  
  Serial.print("Conectando na rede: ");  
  Serial.println(ssid);  
  WiFi.begin(ssid, senha);  
  while (WiFi.status() != WL_CONNECTED) {  
    piscaLed();  
    Serial.print(".");  
  }  
}
```

```
Serial.println("");  
Serial.println("WiFi conectado");  
Serial.println("Endereço IP: ");  
Serial.println(WiFi.localIP());  
  
server.on("/", enviaInfo);  
server.on("/ledOn", ligaLed);  
server.on("/ledOff", desligaLed);  
server.on("/releOn", ligaRele);  
server.on("/releOff", desligaRele);  
server.onNotFound(envia404);  
  
server.begin();  
Serial.println("Servidor HTTP Iniciado");  
}
```



- Adicionando Sensores e Atuadores
  - Métodos de resposta a eventos

```
void piscaLed() {  
    digitalWrite(ledPin, HIGH);  
    delay(100);  
    digitalWrite(ledPin, LOW);  
    delay(100);  
}
```

```
void loop(){  
    server.handleClient();  
}  
void ligaLed(){  
    digitalWrite(ledPin,HIGH);  
    statusLed = true;  
    enviaInfo();  
}  
void desligaLed(){  
    digitalWrite(ledPin,LOW);  
    statusLed= false;  
    enviaInfo();  
}  
void ligaRele(){  
    digitalWrite(relePin,HIGH);  
    statusRele = true;  
    enviaInfo();  
}  
void desligaRele(){  
    digitalWrite(relePin,LOW);  
    statusRele = false;  
    enviaInfo();  
}
```

- Adicionando Sensores e Atuadores
  - Leitura do sensor DHT11 e envio para renderizar o HTML

```
void enviaInfo() {
```

```
    temp = dht.readTemperature();
```

```
    umi = dht.readHumidity();
```

```
    Serial.println(temp);
```

```
    Serial.println(umi);
```

```
    server.send(200, "text/html", respHTML(temp,umi, statusLed, statusRele));
```

```
}
```

```
void envia404(){
```

```
    server.send(404, "text/plain", "Não Encontrado");
```

```
}
```

- Adicionando Sensores e Atuadores
  - Renderização do HTML

**String respHTML(float temperatura, float umidade, bool led, bool rele){**

```
String ptr = "<!DOCTYPE html> <html>\n";
ptr += "<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
ptr += "<meta http-equiv='refresh' content='2'>";
ptr += "<meta charset='UTF-8'>";
ptr += "<title>Aula IoT</title>\n";
ptr += "<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";
ptr += "body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";
ptr += ".button {display: block;width: 80px;background-color: #1abc9c;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";
ptr += ".buttontemp {display: block;width: 100px;background-color: #11bc11;border: 2px;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";
ptr += ".buttonumi {display: block;width: 100px;background-color: #11bc11;border: 2px;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";
ptr += ".button-on {background-color: #1abc9c;}\n";
ptr += ".temperatura {background-color: #1abc9c;}\n";
ptr += ".button-on:active {background-color: #16a085;}\n";
ptr += ".button-off {background-color: #34495e;}\n";
ptr += ".button-off:active {background-color: #2c3e50;}\n";
ptr += "p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";
ptr += "</style>\n";
ptr += "</head>\n";
ptr += "<body>\n";
ptr += "<div id=\"webpage\">\n";
ptr += "<h1>Aula IoT</h1>\n";
```

- Adicionando Sensores e Atuadores
  - Renderização do HTML – Exibição da temperatura, umidade e botões para ligar e desligar o led e relé

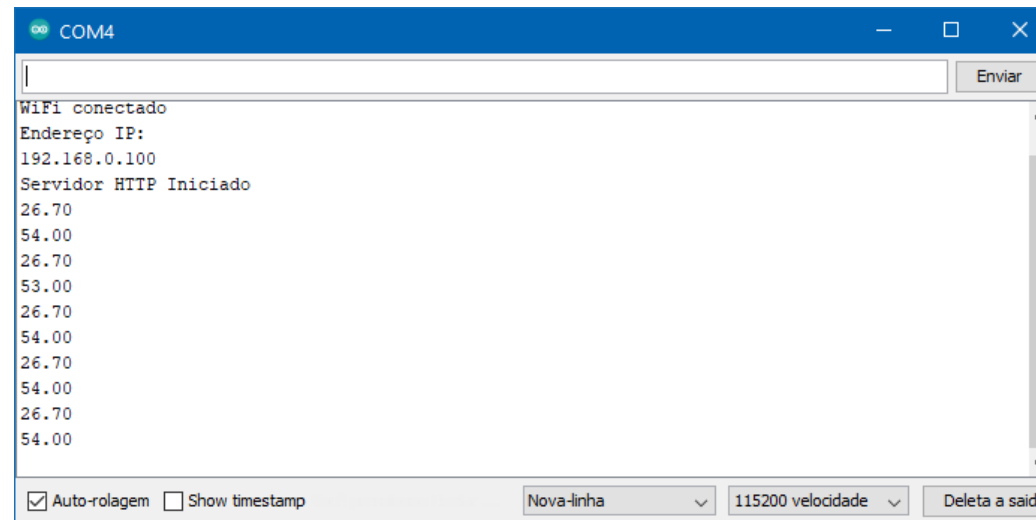
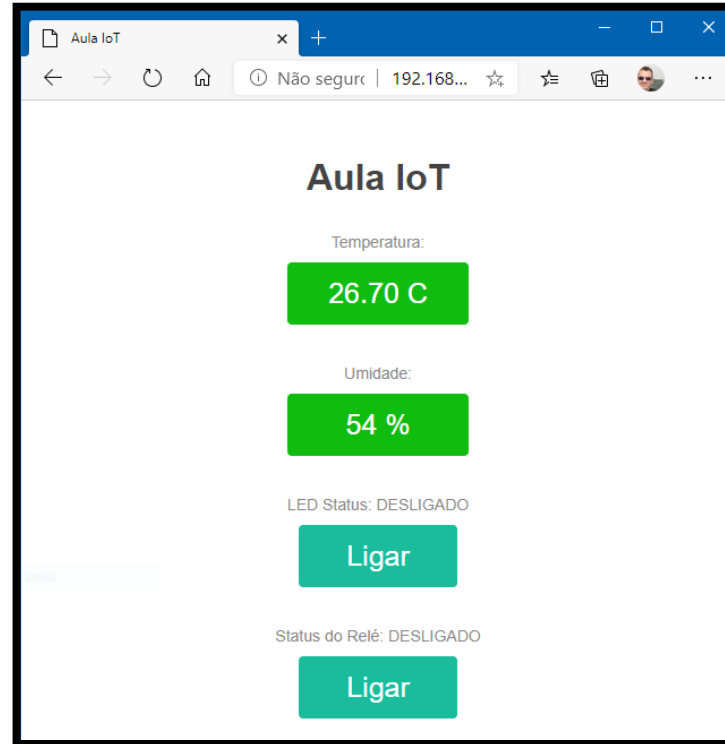
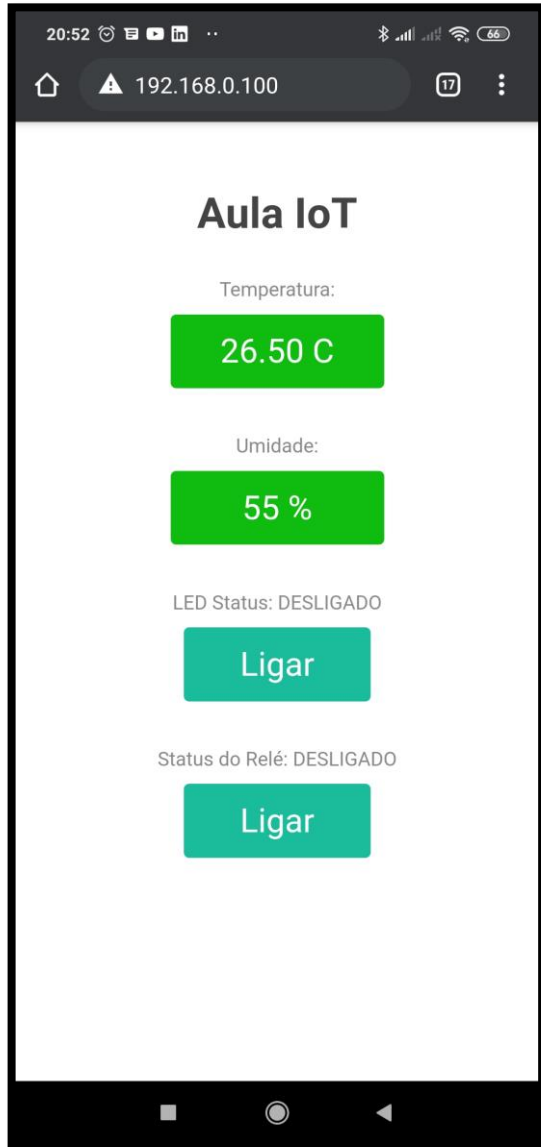
```
ptr += "<p>Temperatura: </p><p class=\"buttontemp\">";
ptr += (float)temperatura;
ptr += " C</p>";
ptr += "<p>Umidade: </p><p class=\"buttonumi\">";
ptr += (int)umidade;
ptr += " %</p>";
ptr += "</div>\n";

if(led)
{ptr += "<p>LED Status: LIGADO</p><a class=\"button button-off\" href=\"/ledOff\">Desligar</a>\n";}
else
{ptr += "<p>LED Status: DESLIGADO</p><a class=\"button button-on\" href=\"/ledOn\">Ligar</a>\n";}

if(rele)
{ptr += "<p>Status do Relé: LIGADO</p><a class=\"button button-off\" href=\"/releOff\">Desligar</a>\n";}
else
{ptr += "<p>Status do Relé: DESLIGADO</p><a class=\"button button-on\" href=\"/releOn\">Ligar</a>\n";}

ptr += "</body>\n";
ptr += "</html>\n";
return ptr;
}
```

- Resultado



- Melhorando o App: Alterando a cor do botão de temperatura no HTML caso passe de um determinado valor

- Vamos criar uma variável de alerta:

```
bool statusLed = false;  
bool statusRele = false;  
bool statusAlerta = false;
```

- No método de captura dos valores dos sensores, se a temperatura for maior do que 35°C, seta o alerta para true

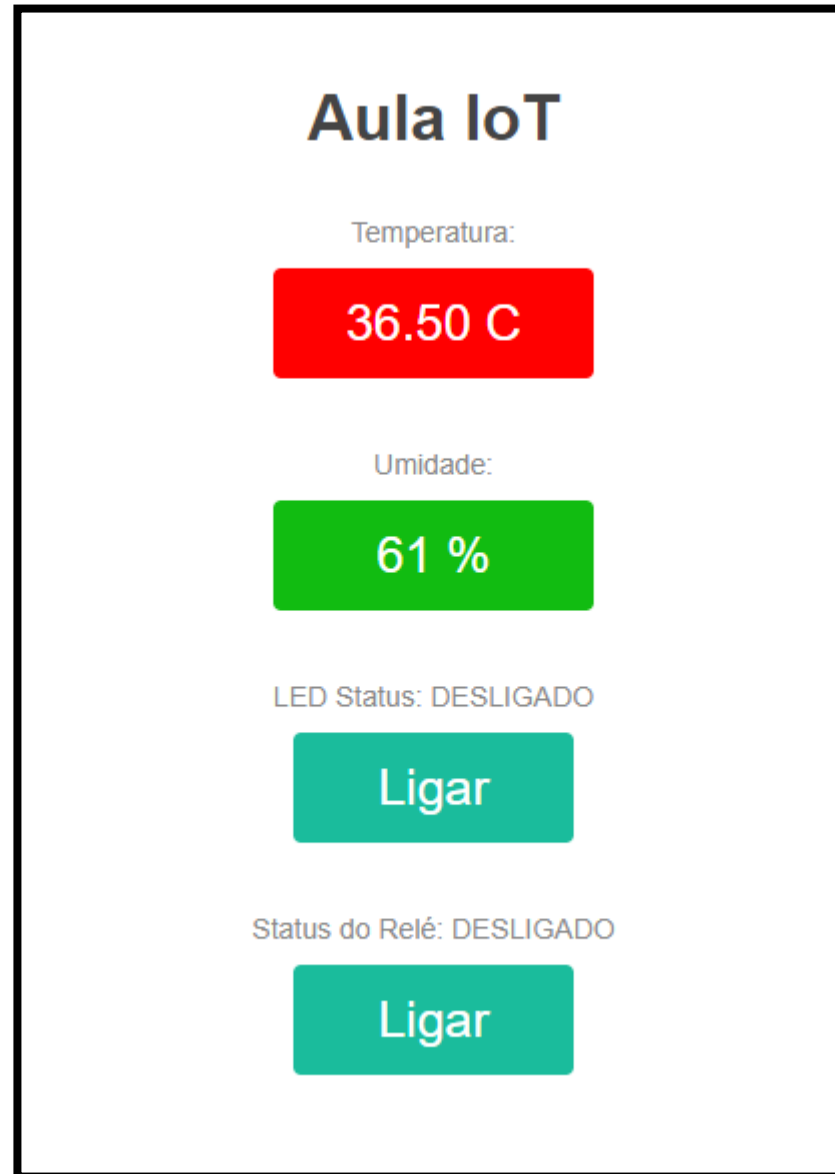
```
if (temp > 35.0) {  
    statusAlerta=true;  
}  
else {  
    statusAlerta=false;  
}
```



- Melhorando o App: Alterando a cor do botão de temperatura no HTML caso passe de um determinado valor
  - Na renderização do HTML, adicionamos uma condição onde se o alerta estiver true, o método envia o CSS com o botão vermelho, ou verde caso esteja falso:
  - Vá na linha onde está o CSS buttontemp e o substitua pelo código abaixo:

```
if (!statusAlerta) {  
    ptr += ".buttontemp {display: block;width: 100px;background-color: #11bc11;border: 2px;color:  
white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor:  
pointer;border-radius: 4px;}}\n";  
}  
else {  
    ptr += ".buttontemp {display: block;width: 100px;background-color: #ff0000;border: 2px;color:  
white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor:  
pointer;border-radius: 4px;}}\n";  
}
```

- Melhorando o App: Alterando a cor do botão de temperatura no HTML caso passe de um determinado valor:



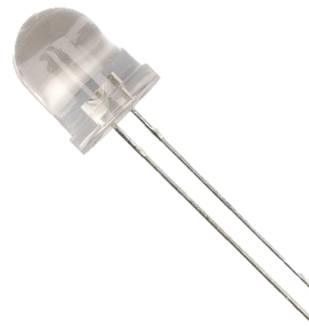
# Programando Raspberry Pi com C#

# Projeto do Pisca-pisca

- Para conhecer o funcionamento de um sistema de IoT, vamos iniciar com uma tarefa simples: criar um pisca-pisca
- O pisca-pisca consiste em um Led conectado à uma das portas de entrada e saída do Raspberry Pi
- Os LEDs trabalham geralmente com tensões entre 1,7 e 2,5 volts e o Raspberry Pi com tensões de 3,3 e 5 volts. Por este motivo é necessária a utilização de um outro componente, o resistor.



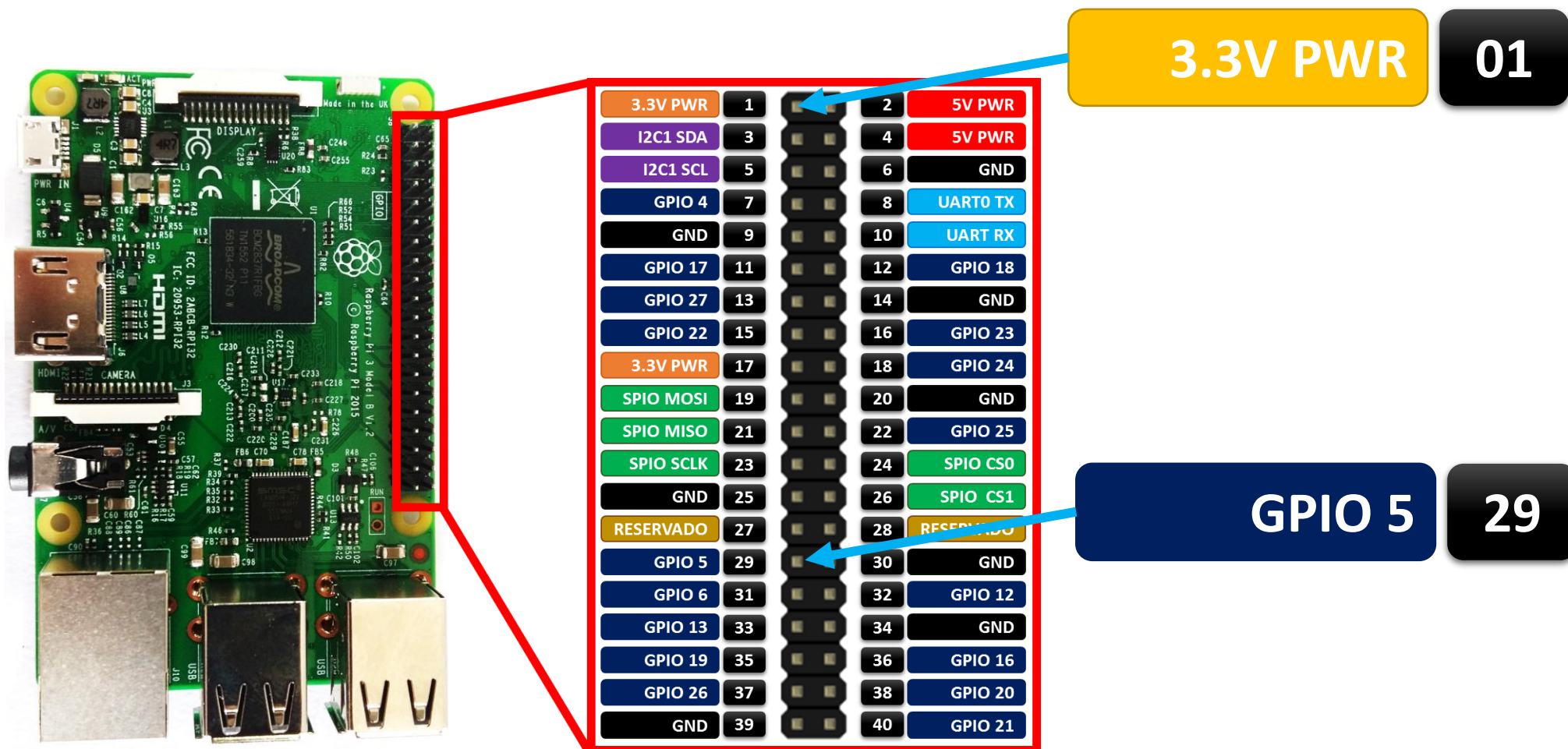
O resistor é um componente eletrônico cuja função é, no caso deste projeto, reduzir a tensão de 3,3 volts do Raspberry para um valor adequado ao led



LED - Light Emitting Diode (Diodo Emissor de Luz) é um componente eletrônico polarizado, ou seja, ele tem uma posição correta de conexão – tem um pino ligado ao positivo (perna mais comprida) e outro ao negativo (perna mais curta).

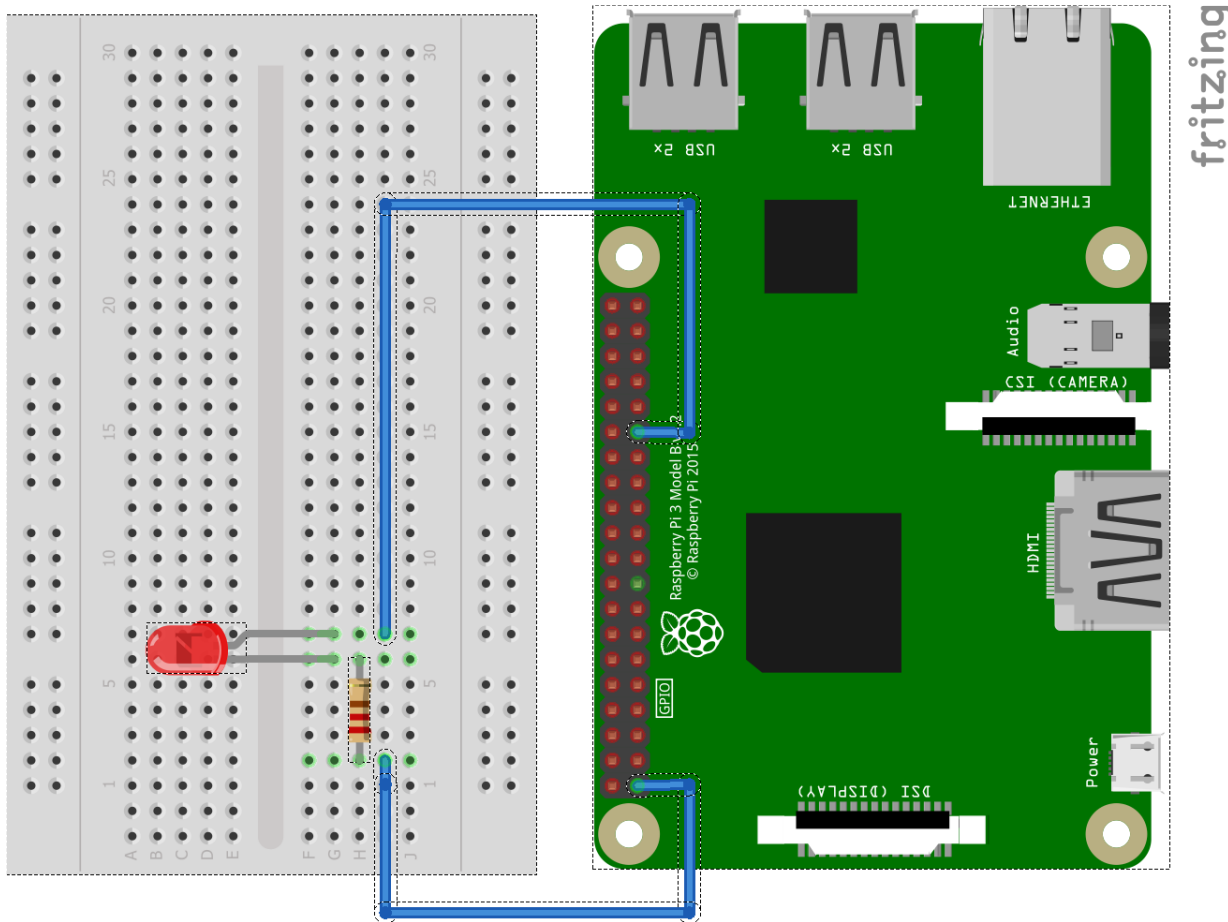
# Adicionando o Led ao Raspberry Pi

- O Raspberry Pi possui várias portas de entrada e saída, que permitem a conexão de sensores e atuadores. Nesse exemplo, vamos usar a porta GPIO 5, pino 29 no conector de entrada e saída do Raspberry



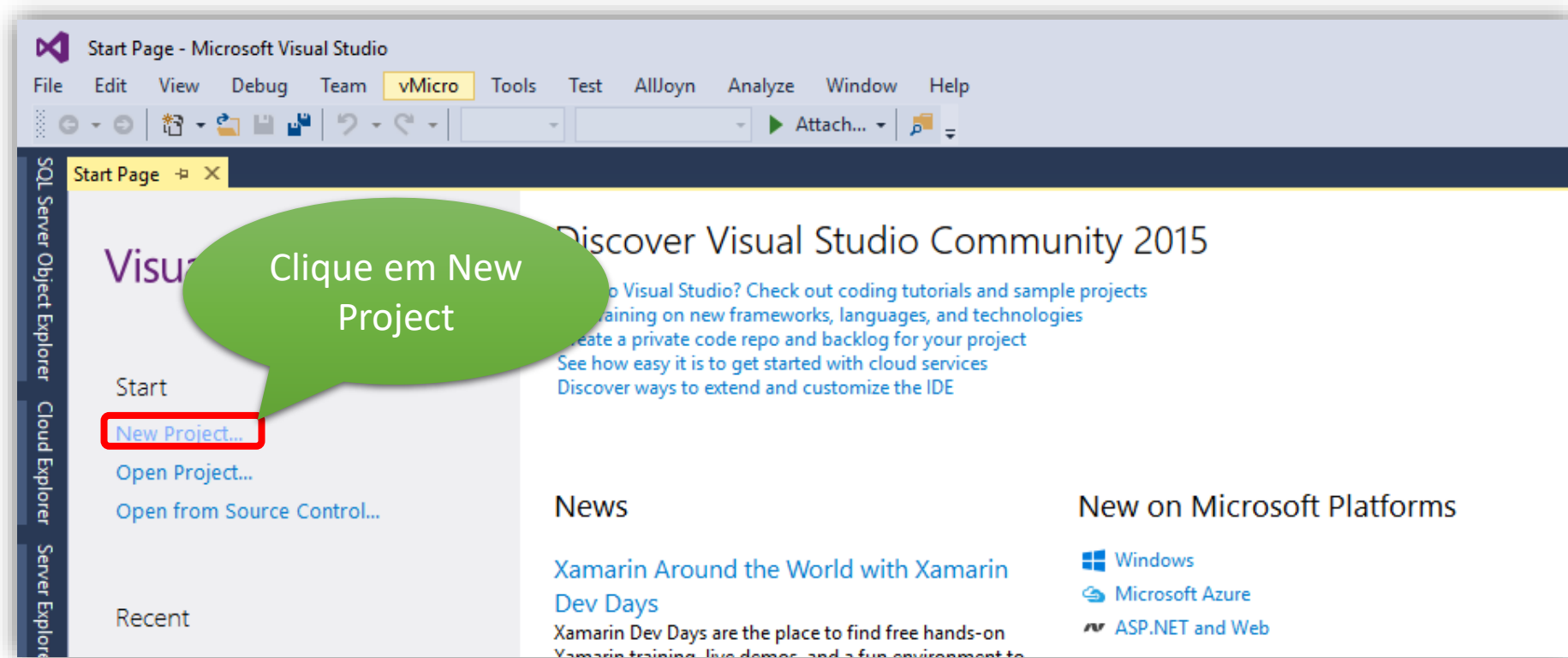
# Adicionando o Led ao Raspberry Pi

- Utilizando o Protoboard, vamos conectar os componentes ao Raspberry Pi, tomando cuidado com a polaridade do led, conectando a perna maior do led ao resistor, a perna menor à porta 5 do Raspberry Pi e a outra perna do resistor à saída de 3,3 volts do Raspberry Pi

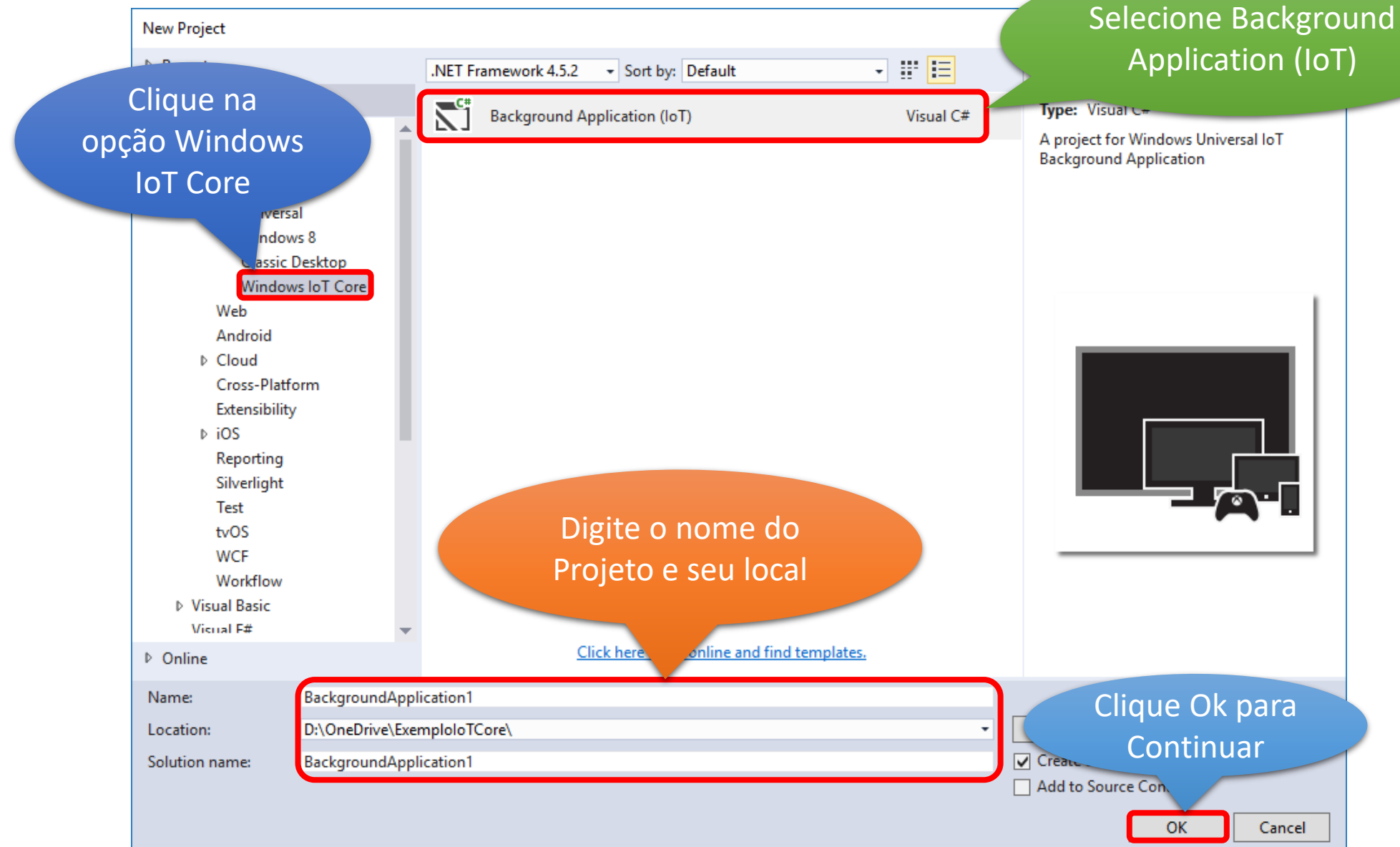


O diagrama esquemático ao lado foi desenvolvido no Fritzing, que é uma ferramenta gratuita para criar projetos de eletrônica. Mais informações em <http://www.fritzing.org>

- No Visual Studio, clique em New Project



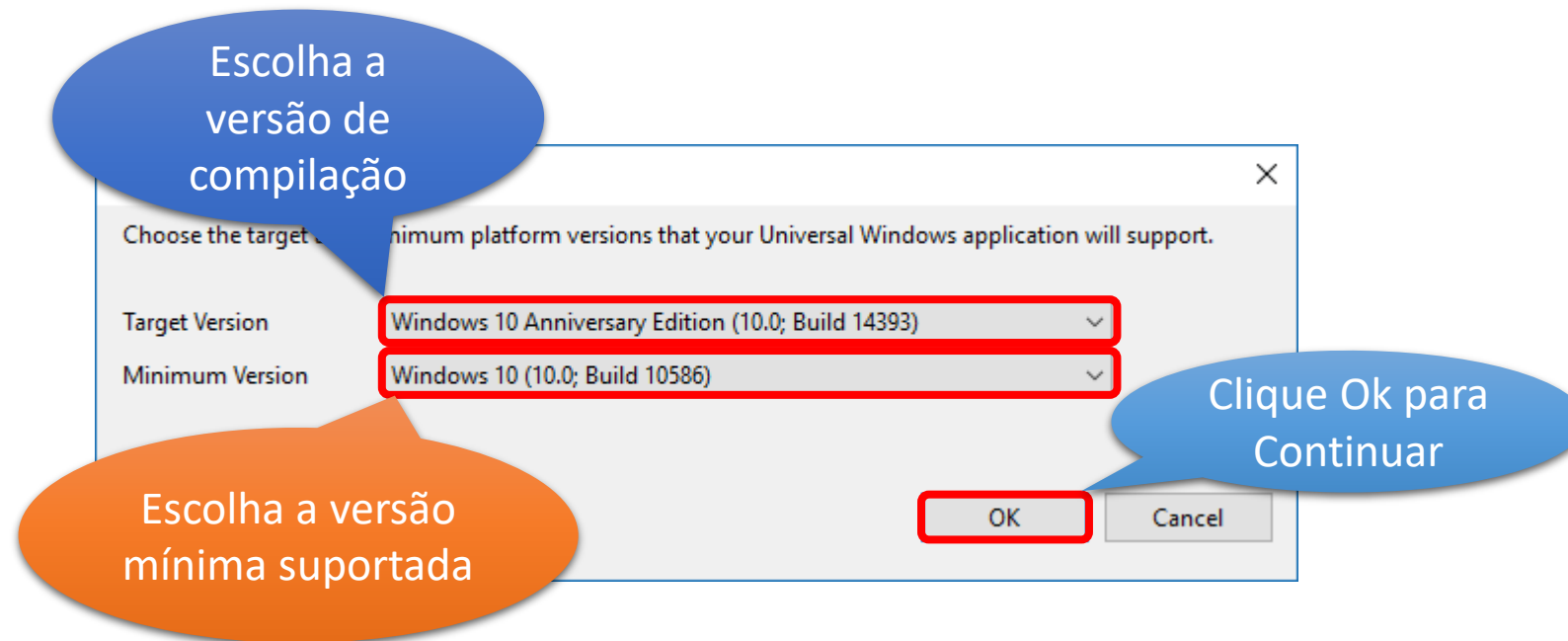
# Criando o projeto de software





# Escolhendo a versão do Windows IoT

- Você pode desenvolver aplicações de IoT para diferentes versões de Windows IoT
- Selecione a versão a ser desenvolvida. Geralmente a opção padrão é a mais indicada



## Adicionando o código ao projeto

- Quando um projeto de IoT é criado no Visual Studio, ele adiciona uma classe chamada `StartupTask`, herdando `IBackgroundTask`, responsável por executar seus métodos em segundo plano quando a aplicação é inicializada. O código a ser executado inicialmente fica dentro do método `Run`
- Vamos criar uma aplicação para piscar um led.
  - Primeiro passo é adicionar as bibliotecas `System`, `ApplicationModel.Background`, `Gpio` e `Threading`.
    - `System` é a biblioteca padrão do C#
    - `ApplicationModel.Background` é a biblioteca para tratar esse app como aplicação do tipo `Background application`
    - `GPIO` é a biblioteca que permite ao Windows IoT Core acessar as portas de entrada e saída do dispositivo IoT
    - `Threading` é a biblioteca para que o dispositivo possa executar o ato de piscar o led em uma thread

```
using System;  
using Windows.ApplicationModel.Background;  
using Windows.Devices.Gpio;  
using Windows.System.Threading;
```

# Inicializando a aplicação

- Vamos criar uma aplicação para piscar um led.
  - Temos então a declaração do namespace da aplicação e inicialização das variáveis e constantes

```
namespace BackgroundApplication
{
    public sealed class StartupTask : IBackgroundTask
    {
        BackgroundTaskDeferral deferral;
        private GpioPinValue value = GpioPinValue.High;
        private const int LED_PIN = 36;
        private GpioPin pin;
        private ThreadPoolTimer timer;
```

Definimos aqui que o estado inicial do led é LIGADO

Definimos que a porta do dispositivo a ser utilizada pelo LED é a porta 36

Definimos um temporizador para especificar o tempo que a porta vai ficar ativa

# Método Run

- Vamos criar uma aplicação para piscar um led.
  - O método Run é o primeiro método a ser executado pela aplicação

```
public void Run(IBackgroundTaskInstance taskInstance)
{
    deferral = taskInstance.GetDeferral();
    InitGPIO();
    timer = ThreadPoolTimer.CreatePeriodicTimer(Timer_Tick, TimeSpan.FromMilliseconds(500));
}
```

Inicializa as portas de entrada e saída do dispositivo

Cria um timer para executar o método para piscar o led em um intervalo de tempo

Chama o método Timer\_Tick no intervalo de tempo especificado em TimeSpan

Define a intermitência do pisca-pisca em 500 milisegundos

# Inicializando as Portas

- Vamos criar uma aplicação para piscar um led.
- O método InitGPIO é responsável por inicializar as portas de entrada e saída do Raspberry Pi
- Neste exemplo, a porta 5, definida anteriormente como LED\_PIN é configurada como Output (saída) e inicializada como High (Ligado)

```
private void InitGPIO()  
{  
    pin = GpioController.GetDefault().OpenPin(LED_PIN);  
    pin.Write(GpioPinValue.High);  
    pin.SetDriveMode(GpioPinDriveMode.Output);  
}
```

Solicita ao dispositivo a utilização de um pino, no caso, pino 5

Define a saída do pino 5 como LIGADO (High)

Declara o pino como sendo uma porta de saída

# Timer Tick

- Vamos criar uma aplicação para piscar um led.
- O método `Timer_Tick` é executado a cada 500 milissegundos pela aplicação e é responsável por alternar, a cada execução, o valor da variável `value`.
- `Value` irá alterar de High para Low, ou seja, de Ligado para Desligado a cada 500 milissegundos

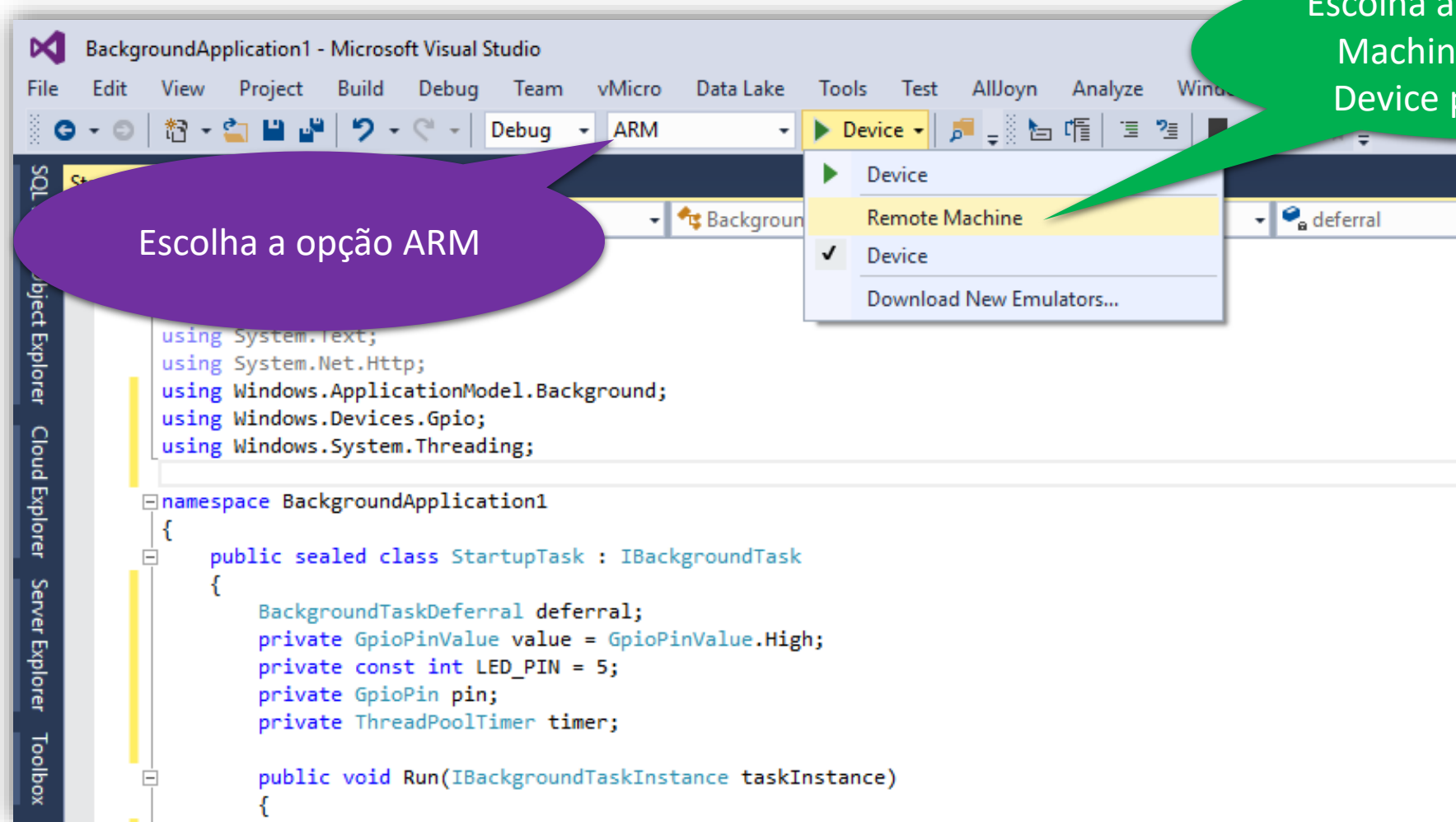
```
private void Timer_Tick(ThreadPoolTimer timer)
{
    value = (value == GpioPinValue.High) ? GpioPinValue.Low : GpioPinValue.High;
    pin.Write(value);
}
}
```

Ativa ou desativa o led  
de acordo com o valor  
da variável `value`

Se `value` era High  
anteriormente, ele passa  
para Low e vice-versa

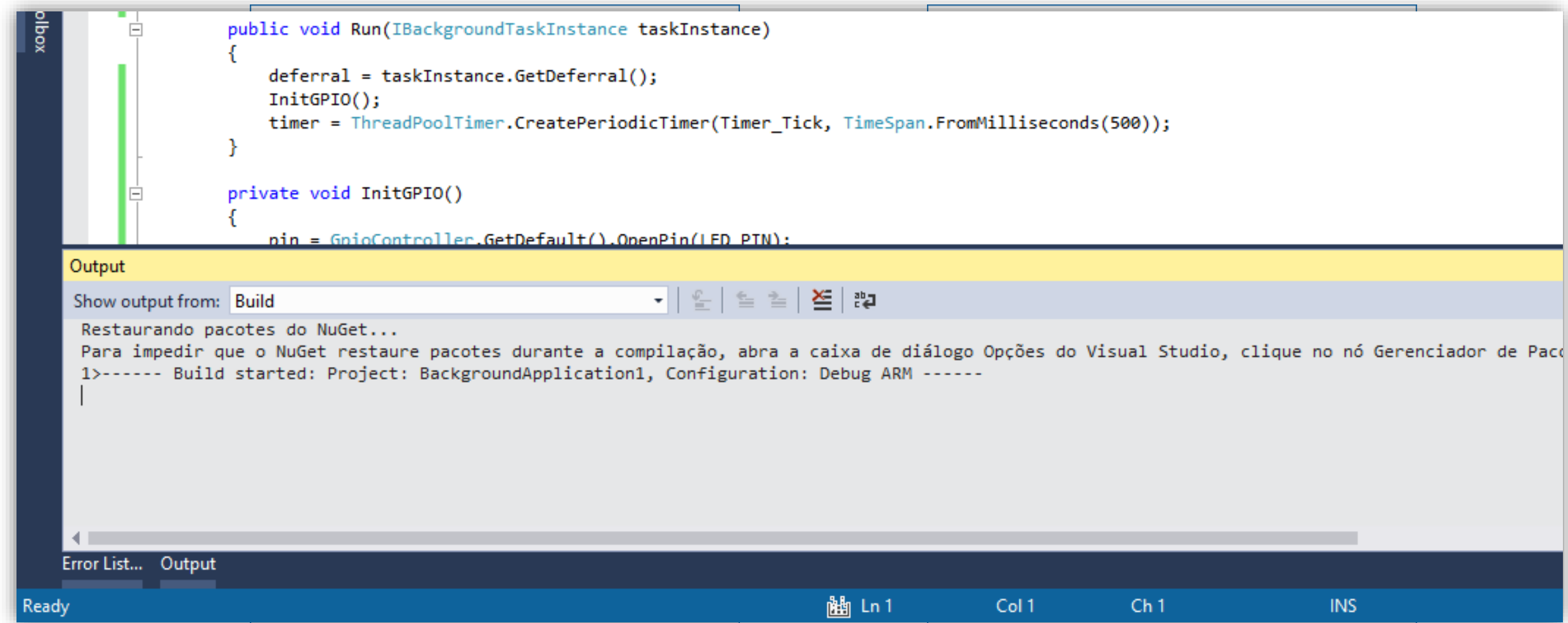
# Executando a aplicação

- Com o programa pronto, vamos compilar para um dispositivo do tipo ARM e fazer o upload para o Raspberry Pi utilizando a opção Remote Machine, ao invés de compilar diretamente



# Escolhendo o dispositivo

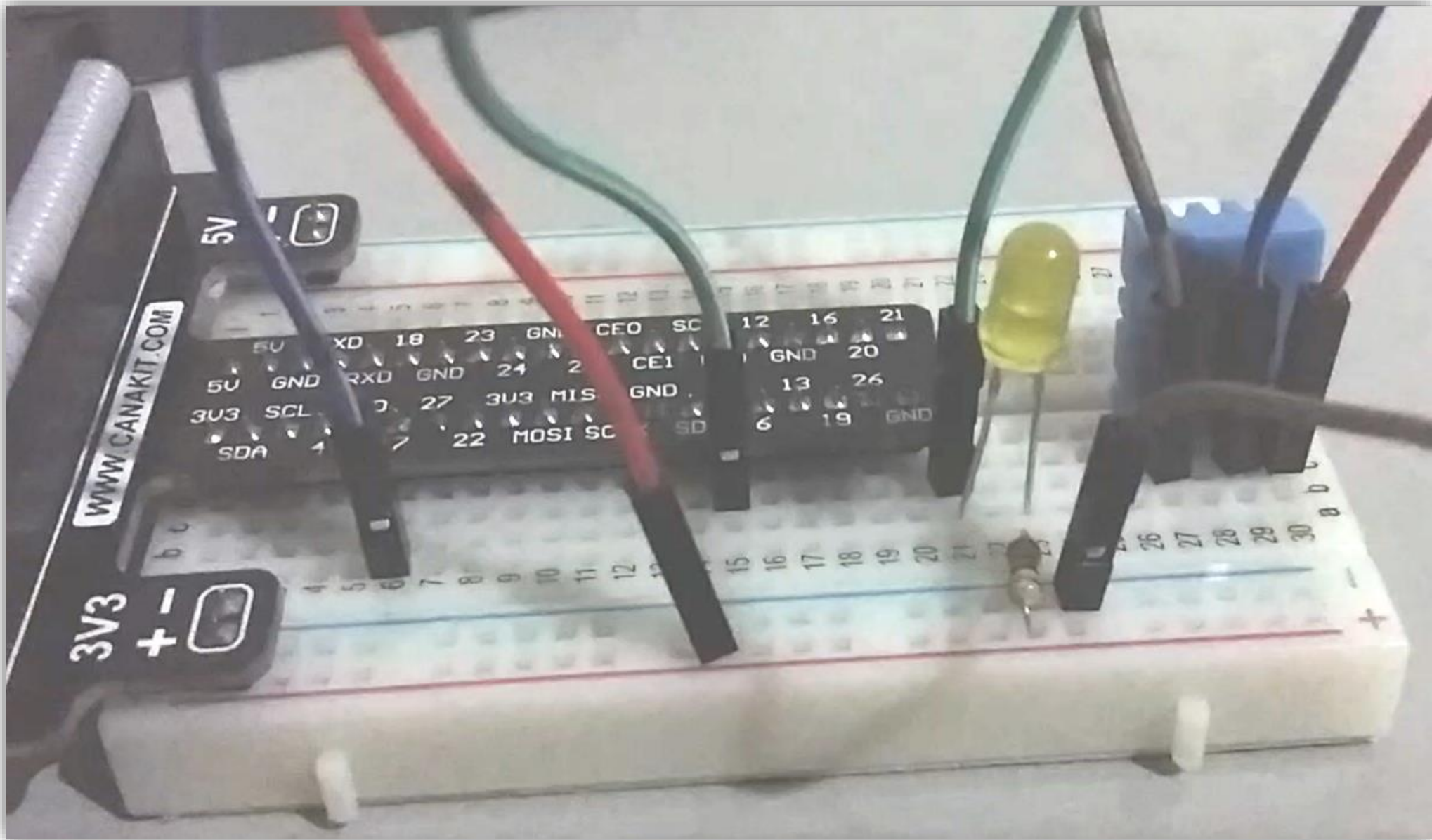
- O Visual Studio irá compilar e fazer o upload para o Raspberry Pi automaticamente e o aplicativo deverá ser executado como Background Application no dispositivo.
- Configuration. Clique em Select para compilar para este dispositivo.





## Pisca-pisca sendo executado

- O resultado é o led piscando no intervalo de 500 milissegundos



## Outros exemplos de código

- Você pode conferir esse exemplo de código e muitos outros em:

<https://github.com/ms-iot/samples>

This repository Search

Pull requests Issues Gist

ms-iot / samples

Watch 264 Star 727 Fork 1,125

Code Issues 14 Pull requests 5 Projects 0 Wiki Pulse Graphs

Windows 10 IoT Core Samples

1,331 commits 3 branches 4 releases 47 contributors MIT

Branch: develop New pull request

Create new file Upload files Find file Clone or download

bfjelds committed on GitHub Merge pull request #341 from namkedia/develop

AirHockeyRobot	Update pfx to new shared MSFT pfx
AllJoyn	Adding readmes to alljoyn sample code
AppServiceBlinky/C#	Add AppServices Samples
AppServiceSharedNotepad/C#	Add AppServices Samples
ArduinoLibraryBlinky	Added links to basic Arduino Wiring setup page

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

<https://github.com/ms-iot/samples.git>

Open in Desktop Download ZIP

7 months ago

6 months ago