

Programação em Arduino

Prof. Daniel R. Cassar and James Moraes de Almeida

Capacitação em Arduino

Contents

2	Módulo 3	1
2.1	Laço de repetição com <code>while</code>	1
2.2	Laço de repetição com <code>for</code>	3
2.3	Criando variáveis reais	4
2.4	Portas analógicas do Arduino e a função <code>analogRead</code>	4
2.5	Comunicação serial entre o Arduino e o computador	5
2.6	Criando variáveis inteiras tipo <code>long</code>	6
2.7	Portas digitais PWM e a função <code>analogWrite</code>	7
2.8	Usando bibliotecas	8
2.9	Sensor de distância ultrassônico (HC-SC04)	10
2.10	Sensor de temperatura e umidade (DHT11)	11
2.11	Comunicação tipo I ² C	14
2.12	Sensor de temperatura e pressão (BMP280)	16
2.13	Lendo e armazenando dados enviados pela comunicação serial usando Python	18

2 Módulo 3

2.1 Laço de repetição com `while`

O bloco de laço de repetição do Arduino (`void loop`) nos fornece uma maneira simples de indicar para o Arduino quais linhas de código ele deve repetir. Em certos casos, apenas o bloco de laço de repetição não é suficiente e precisamos de mais controle. O comando `while`¹ é uma forma de criar laços de repetição com condições controladas. Observe o exemplo abaixo.

¹Significa “enquanto” em inglês.

```

1  int contador = 0;
2
3  void setup() {
4  }
5
6  void loop() {
7      while (contador < 100) {
8          /* tudo que estiver neste bloco aqui
9             irá ser executado 100 vezes */
10         contador = contador + 1;
11     }
12
13     contador = 0; // Aqui estamos reinicializando o contador em zero
14
15     /* tudo que estiver fora do 'while' não é afetado por ele.
16        Enquanto o bloco dentro do 'while' rodou 100 vezes,
17        este código fora do 'while' irá rodar apenas 1 vez. */
18 }

```

Neste código temos o comando `while` seguido de um parêntese e um bloco de código entre chaves. O código dentro do parêntese é a chamada *condição* do `while`. O bloco de código entre chaves do comando `while` apenas será executado se a condição for verdadeira (`true`). Neste caso estamos checando se a variável `contador` é menor que a constante 100. Cada vez que o bloco de código do `while` se encerra, o Arduino retorna ao início do bloco do `while` e checa a condição novamente. Se a condição permanecer verdadeira o bloco do `while` é executado novamente, do contrário o bloco do `while` não é executado e o programa segue para a linha imediatamente após o final do bloco do `while`.

A variável `contador` do código acima é o que chamamos de *variável acumuladora*. Veja que é responsabilidade desta variável “contar” quantas vezes o bloco do `while` foi executado. Toda vez que o bloco é executado, esta variável é acrescida do número 1 (linha 10). O processo de somar 1 a uma variável é uma tarefa tão usada em programação que é comum existir um “açúcar sintático” para expressar essa tarefa. Em C++ podemos reescrever a linha 10 do código acima da seguinte maneira: “`contador++`” (ou seja, o nome da variável com um “++” no final).

Se a condição do `while` for sempre verdadeira, então o bloco do `while`

entrou em um *loop* infinito. Se a condição do `while` nunca for verdadeira, então o bloco do `while` jamais será executado. Para evitar problemas, é usual zerar o contador logo após sair do bloco do `while`. Isto está sendo feito na linha 13 do código acima.

2.2 Laço de repetição com for

Outro laço de repetição disponível é o `for`². O laço `for` tem uma sintaxe um pouco diferente da sintaxe do laço `while`, sendo que ele próprio “cuida” da sua variável acumuladora. Vamos ver o código anterior reescrito usando um laço `for`:

```
1 void setup() {
2 }
3
4 void loop() {
5     for (int contador = 0; contador < 100; contador++) {
6         /* tudo que estiver neste bloco aqui
7            irá ser executado 100 vezes */
8     }
9
10    /* tudo que estiver fora do 'for' não é afetado por ele.
11       Enquanto o bloco dentro do 'for' rodou 100 vezes,
12       este código fora do 'for' irá rodar apenas 1 vez. */
13 }
```

Observe que a instrução `for` requer três argumentos para ser executado.

1. O primeiro argumento é a *inicialização*. Ele é o primeiro comando do laço `for` a ser executado e só é executado uma vez ao entrar no laço. É aqui que nós criamos a nossa variável acumuladora para ser usada no laço `for`.
2. O segundo argumento é a *condição*. Ele é similar à condição que vimos no laço `while`. O bloco do laço `for` será executado se a condição for

²Significa “para” em inglês.

`true` e não será executado se a condição for `false`.

3. O terceiro argumento é o *incremento*. Ele é executado toda vez que a condição for `true` e serve para atualizar a variável acumuladora.

2.3 Criando variáveis reais

Variáveis inteiras têm duas limitações importantes: não permitem números decimais e estão restritas a números na faixa de -32.768 a 32.767 . Existem casos onde precisamos armazenar um número real em uma variável que não tenha essas limitações.

As variáveis tipo ponto flutuante (`float`) podem armazenar números decimais e valores desde $-3,40 \times 10^{38}$ até $3,40 \times 10^{38}$.

A sintaxe para criar variáveis do tipo `float` é similar à sintaxe para criar variáveis do tipo `int`. Veja no exemplo abaixo.

```
1 float variavel_float = 3.1415;
2 float variavel_com_notacao_e = 1.23456e4; // equivale a 12345.6
3
4 void setup() {
5 }
6
7 void loop() {
8 }
```

2.4 Portas analógicas do Arduino e a função `analogRead`

As portas digitais podem receber ou enviar apenas dois sinais distintos: `LOW` ou `HIGH`³. Já as portas analógicas do Arduino podem apenas *receber* sinal, porém não estão limitadas em apenas dois sinais distintos: elas recebem um sinal que é um valor entre 0 e 1023. Ter um maior intervalo de valores é particularmente interessante quando temos interesse em sensores que medem quantidades contínuas como temperatura, pressão e umidade.

³Falso ou verdadeiro, desligado ou ligado, zero ou um. Portas digitais são portas binárias.

As portas analógicas do Arduino são enumeradas e começam com a letra A⁴. Para ler o sinal elétrico que chega em uma porta analógica usamos a função `analogRead`. Esta função tem apenas um argumento, o número da porta analógica. Veja um exemplo abaixo.

```
1 int porta_analogica = A3; // observe a letra "A" na frente do número
2 int valor_lido;
3
4 void setup() {
5 }
6
7 void loop() {
8     valor_lido = analogRead(porta_analogica);
9     delay(1000);
10 }
```

Neste exemplo, dentro do bloco de laço de repetição a variável `valor_lido` recebe o valor lido na porta analógica A3 (que será um número inteiro de 0 a 1023). Após ler e armazenar o valor, o código aguarda 1 segundo e o laço se repete.

2.5 Comunicação serial entre o Arduino e o computador

Um Arduino pode enviar e receber informação de um computador através de sua comunicação serial. Para isso, o primeiro passo é o Arduino estar conectado ao computador por um cabo USB.

A comunicação serial é controlada pelo objeto `Serial` e precisa ser explicitamente inicializada antes do uso. Veja o exemplo abaixo.

```
1 int porta_analogica = A3; // observe a letra "A" na frente do número
2 int valor_lido;
3
4 void setup() {
```

⁴Em algumas placas, certas portas digitais funcionam também como portas analógicas. Para mais informações acesse o link <https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>

```

5     Serial.begin(9600);
6 }
7
8 void loop() {
9     valor_lido = analogRead(porta_analogica);
10    Serial.print(valor_lido);
11    delay(1000);
12 }

```

A linha 5 do código acima é a inicialização da comunicação serial entre o Arduino e o computador. Observe que a função `begin`⁵ faz parte do objeto `Serial`, sendo que para chamá-la é necessário primeiro escrever o nome do objeto seguido de um ponto (`Serial.begin`). Esta função tem apenas um argumento, sendo ele a velocidade da comunicação (em bits por segundo). Na maioria dos casos a velocidade de 9600 é a recomendada.

A linha 10 do código acima é um dos comandos disponíveis para enviar informação do Arduino para o computador. A função se chama `print`⁶ e ela faz parte do objeto `Serial`, logo deve ser chamada usando a sintaxe `Serial.print`. Esta função envia para o computador o seu argumento em formato de texto. No exemplo, o argumento é o valor lido na porta analógica A3.

O mesmo código acima pode ser escrito usando a função `Serial.println` no lugar da função `Serial.print`. A diferença é que a função `Serial.println` automaticamente inicia uma nova linha após executar o `print` e a função `Serial.print` não faz isso de forma automática.

2.6 Criando variáveis inteiras tipo long

Variáveis inteiras declaradas com `int` podem armazenar qualquer número inteiro entre -32.768 e 32.767. Existe um outro tipo de variável inteira que é capaz de armazenar números muito maiores, trata-se das variáveis tipo `long`. Estas variáveis são capazes de armazenar qualquer inteiro entre

⁵Significa “iniciar” em inglês.

⁶Em tradução livre do inglês seria “imprimir”. A palavra `print` em programação não significa literalmente imprimir um texto usando uma impressora. Usualmente a palavra `print` indica que alguma informação será mostrada para o usuário de forma visual no seu monitor. No caso do Arduino, os comandos tipo `print` enviam informação para o monitor da porta serial.

-2.147.483.648 e 2.147.483.647. A declaração de variáveis tipo `long` é similar à declaração de variáveis tipo `int`, veja um exemplo no código abaixo.

```
1  int variavel_tipo_int;
2  long variavel_tipo_long;
3
4  void setup() {
5  }
6
7  void loop() {
8  }
```

2.7 Portas digitais PWM e a função `analogWrite`

O Arduino não possui saída analógica, suas portas analógicas são apenas para recebimento de sinal. Certas portas digitais do Arduino conseguem *simular* um envio de sinal analógico através da técnica de Modulação por Largura de Pulso (Pulse Width Modulation do inglês, PWM). Esta técnica consiste em enviar um sinal digital na forma de ondas quadradas de alta frequência. O controle do tempo de permanência na crista e no vale da onda permite simular voltagens entre zero e a voltagem máxima de envio da porta digital do Arduino (3,3 ou 5 Volts, dependendo do modelo).⁷

O envio do sinal PWM é feito com a função `analogWrite`. Esta função tem dois argumentos: o primeiro é o número da porta digital e o segundo é a razão cíclica (duty cycle em inglês). A razão cíclica deve ser um valor entre 0 e 255 e indica a fração da voltagem máxima que será enviada. Por exemplo, um valor de 0 simula o envio de 0% da voltagem máxima (0V), um valor de 255 simula o envio de 100% da voltagem máxima e um valor de 127 simula o envio de 50% da voltagem máxima. Este tipo de sinal PWM pode ser usado para alterar o brilho de um LED e controlar a velocidade de um motor.

Apenas algumas das portas digitais do Arduino permitem o envio de sinal PWM. As portas e as frequências de operação podem ser consultadas na documentação oficial em <https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/>. No Arduino Leonardo, por exemplo, as portas com suporte ao envio de sinal PWM são as de número 3, 5, 6, 9,

⁷Para mais informações veja a documentação em <https://www.arduino.cc/en/pmwiki.php?n=Tutorial/PWM>.

10, 11 e 13 (todas operando na frequência de 490 Hz, com exceção das portas 3 e 11 que operam na frequência de 980 Hz).

Veja um exemplo do uso da função `analogWrite` abaixo. Neste código, a voltagem enviada na porta 6 sobe gradativamente com o passar do tempo até atingir a voltagem máxima, depois disso repete este processo novamente.

```
1  int porta = 10;  // Esta é uma porta com suporte a PWM em diversas placas
2
3  void setup() {
4      pinMode(porta, OUTPUT);
5  }
6
7  void loop() {
8      analogWrite(porta, 0);
9      delay(1000);
10     analogWrite(porta, 100);
11     delay(1000);
12     analogWrite(porta, 150);
13     delay(1000);
14     analogWrite(porta, 200);
15     delay(1000);
16     analogWrite(porta, 255);
17     delay(1000);
18 }
```

2.8 Usando bibliotecas

No contexto de programação, uma biblioteca é um conjunto de código reutilizável que nos ajuda a fazer uma certa tarefa. A vantagem de se usar bibliotecas é que não precisamos “reinventar a roda” cada vez que quisermos fazer uma tarefa que já foi implementada por outra pessoa.

Algumas bibliotecas fazem parte das chamadas “bibliotecas padrão” do Arduino e já vem instaladas “de fábrica” quando se instala o Arduino IDE. A biblioteca para gerenciar a memória EEPROM do Arduino e a biblioteca para comunicação com displays de LCD são dois exemplos de bibliotecas padrão.

É bastante comum a necessidade de usar bibliotecas que realizam tarefas não contempladas nas bibliotecas padrão. Existem duas formas de instalar bibliotecas usando o Arduino IDE:

1. Utilizando o gerenciador de bibliotecas disponível no menu **Sketch / Incluir Biblioteca / Gerenciar Bibliotecas**. Este é o procedimento preferencial caso a biblioteca de interesse esteja disponível no gerenciador. Após abrir o gerenciador, buscar a biblioteca desejada e instalar clicando no botão **Instalar**.
2. Instalando a biblioteca usando arquivo **.zip** disponível no menu **Sketch / Incluir Biblioteca / Adicionar Biblioteca .ZIP**. Este procedimento é necessário quando a biblioteca não está no gerenciador de bibliotecas do Arduino.

Antes de usar as funções e objetos contidos em uma determinada biblioteca é necessário carregar ela usando o comando **#include**. O código abaixo carrega a biblioteca **EEPROM** (isso ocorre na linha 1) e utiliza das funções **EEPROM.length** e **EEPROM.write** que são fornecidas por esta biblioteca. A biblioteca **EEPROM** é usada para ler e escrever a memória **EEPROM** interna do Arduino. O código abaixo serve para “limpar” a memória **EEPROM** convertendo todos seus valores para zero.

```
1  #include <EEPROM.h>
2
3  void setup() {
4      for (int i = 0 ; i < EEPROM.length() ; i++) {
5          EEPROM.write(i, 0);
6      }
7  }
8
9  void loop() {
10 }
```

Para saber mais sobre as diversas bibliotecas do Arduino veja a documentação oficial em <https://www.arduino.cc/reference/en/libraries/>.

2.9 Sensor de distância ultrassônico (HC-SC04)

O sensor de distância ultrassônico HC-SC04 é capaz de medir distâncias entre 2 centímetros até 4 metros. O sensor tem um emissor e um receptor de ultrassom e a medida é realizada computando o tempo entre o sinal emitido pelo emissor e recebido pelo receptor.

Este sensor tem 4 pinos:

- VCC: alimentação. Ligar na saída 5V do Arduino;
- Trig: ligar em uma porta digital do Arduino. É este pino que controla o envio da onda ultrassônica;
- Echo: ligar em uma porta digital do Arduino. É neste pino que recebemos a informação do recebimento da onda ultrassônica;
- GND: terra. Deve ser ligado ao GND do Arduino;

A operação deste sensor é facilitada com o uso de uma biblioteca. Existem diversas bibliotecas escritas para controlar este sensor. Uma delas pode ser instalada pelo próprio gerenciador de bibliotecas do Arduino IDE: a biblioteca *Ultrasonic* escrita pelo Erick Simões. O código desta biblioteca é aberto e pode ser conferido no Github em <https://github.com/ErickSimoes/Ultrasonic>. Veja um exemplo do uso desta biblioteca abaixo.

```
#include <Ultrasonic.h>

int pino_echo = 13;
int pino_trig = 12;
float leitura;

Ultrasonic sensor_us(pino_trig, pino_echo); // cria o objeto sensor_us

void setup() {
    Serial.begin(9600);
```

```

}

void loop() {
    leitura = sensor_us.read(); // lê a distância em centímetros
    Serial.print("Distancia em cm: ");
    Serial.print(leitura);
    Serial.println();
    delay(1000);
}

```

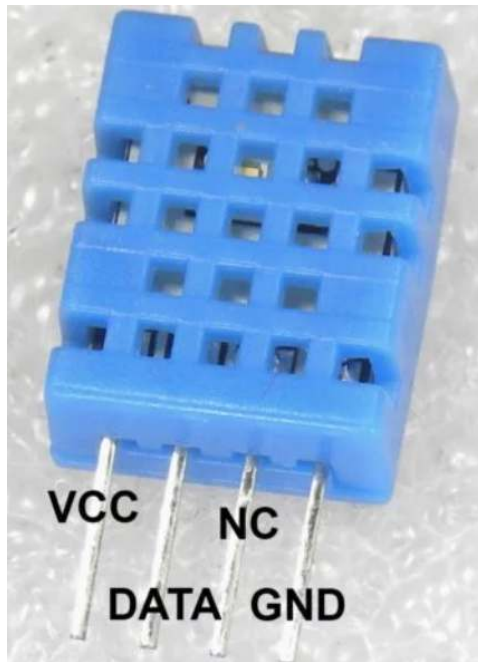
Assim como já vimos anteriormente, é necessário carregar a biblioteca usando o comando `#include` (linha 1). Depois disso, é necessário criar uma instância da classe `Ultrasonic`. A sintaxe para isso lembra a sintaxe de criação de variáveis e funções:

1. Primeiro é necessário declarar o tipo da classe. Neste caso estamos criando uma classe do tipo `Ultrasonic`. Esta classe é definida na biblioteca que importamos na linha 1.
2. Após declarar o tipo da classe escrevemos o nome do objeto que será criado. Neste caso demos o nome de `sensor_us`.
3. Finalmente, é necessário fornecer os argumentos de inicialização da classe. Para o sensor HC-SC04 precisamos de dois argumentos: a porta digital conectada ao pino Trig e a porta digital conectada ao pino Echo.

O objeto `sensor_us` é usado na linha 14 do código acima. Veja que para usar este objeto nós escrevemos o nome dele, um ponto e por fim o nome da função que desejamos usar. Neste caso é a função `read` que não recebe nenhum argumento e retorna a distância em centímetros.

2.10 Sensor de temperatura e umidade (DHT11)

O sensor de temperatura e umidade DHT11 tem 4 pinos não nomeados. Se você estiver olhando para a face com furinhos e os pinos estiverem apondo para baixo, então da esquerda para a direita temos:



- VCC: alimentação. Ligar na saída 5V do Arduino;
- DADOS: comunicação de dados. Deve ser ligado em uma porta analógica;
- NC: este pino não é utilizado;
- GND: terra. Deve ser ligado ao GND do Arduino;

A biblioteca recomendada para a operação deste sensor é a “DHT sensor library” desenvolvida pela Adafruit. Esta biblioteca pode ser instalada pelo gerenciador de bibliotecas do Arduino IDE. Veja abaixo um exemplo da utilização desta biblioteca.

```
1 #include <DHT.h>  
2
```

```

3  float leitura_umidade;
4  float leitura_temperatura;
5
6  int pino_dados = A1;
7
8  DHT sensor_dht(pino_dados, DHT11);
9
10 void setup() {
11     Serial.begin(9600);
12     sensor_dht.begin();
13 }
14
15 void loop() {
16     leitura_umidade = sensor_dht.readHumidity();
17     leitura_temperatura = sensor_dht.readTemperature();
18
19     Serial.print("Umidade (em porcentagem): ");
20     Serial.print(leitura_umidade);
21     Serial.println();
22
23     Serial.print("Temperatura (Celsius): ");
24     Serial.print(leitura_temperatura);
25     Serial.println();
26
27     delay(1000);
28 }

```

Novamente temos a inclusão da biblioteca com o comando `#include` na linha 1 e a criação de uma instância da classe fornecida por esta biblioteca na linha 8. A classe DHT requer dois argumentos para ser instanciada: o primeiro é o pino analógico onde o sensor está conectado e o segundo é o tipo de sensor. Para o sensor DHT11 basta escrever `DHT11` no segundo argumento. Na verdade a variável `DHT11` é definida dentro do código da própria biblioteca e é criada quando a biblioteca é carregada na linha 1.

No exemplo acima, o objeto que foi criado chama-se `sensor_dht`. Ele requer uma inicialização com a função `begin` antes de ser operado (veja a linha 12). Após ser inicializado, a leitura da umidade se dá pela função `readHumidity` e a leitura da temperatura se dá pela função `readTemperature`. Ambas são funções de zero argumentos.

2.11 Comunicação tipo I²C

O Arduino tem capacidade de controlar sensores através do protocolo de comunicação I²C. I²C é um protocolo relativamente simples e barato, porém não é indicado para tarefas que exigem alta velocidade de comunicação. Diversos sensores do Arduino têm suporte para este tipo de comunicação.

O protocolo I²C utiliza de uma linha de dados (chamada SDA) e um barramento com clock (chamado de SCL). Quando um sensor é controlado pelo Arduino seguindo este protocolo, é necessário conectar os pinos SDA e SCL do sensor nas portas corretas da placa Arduino.

O pino SCL deve estar ligado na porta SCL do Arduino Leonardo, na porta A5 do Arduino Nano ou na porta 21 do Arduino Due. Para outros modelos de Arduino você deve checar qual é a porta SCL da placa.

O pino SDA deve estar ligado na porta SDA do Arduino Leonardo, na porta A4 do Arduino Nano ou na porta 20 do Arduino Due. Para outros modelos de Arduino você deve checar qual é a porta SDA da placa.

Suponha que você acabou de montar um circuito eletrônico contendo um Arduino e pelo menos um sensor que se comunica utilizando o protocolo I²C. Como checar se a comunicação I²C está funcionando? Para isso, basta executar o código abaixo⁸. A biblioteca `Wire` importada na linha 1 é uma das bibliotecas embutidas do Arduino, não é necessário nenhuma instalação adicional. Os detalhes da implementação deste código não serão aprofundados aqui.

```
1 #include <Wire.h>
2
3 byte error, address;
4 int nDevices;
5
6 void setup() {
7     Wire.begin();
```

⁸Adaptado de `i2c_scanner` disponível em <https://playground.arduino.cc/Main/I2cScanner/>.

```

8      Serial.begin(9600);
9      while (!Serial);
10         Serial.println("\nI2C Scanner");
11  }
12
13  void loop() {
14      Serial.println("Escaneando...");
15
16      nDevices = 0;
17      for (address = 1; address < 127; address++) {
18          Wire.beginTransmission(address);
19          error = Wire.endTransmission();
20
21          if (error == 0) {
22              Serial.print("Dispositivo I2C encontrado no endereco 0x");
23              if (address<16)
24                  Serial.print("0");
25              Serial.print(address,HEX);
26              Serial.println("  !");
27              nDevices++;
28          }
29          else if (error==4) {
30              Serial.print("Erro desconhecido no endereco 0x");
31              if (address<16)
32                  Serial.print("0");
33              Serial.println(address,HEX);
34          }
35      }
36      if (nDevices == 0)
37          Serial.println("Nenhum dispositivo I2C encontrado\n");
38      else
39          Serial.println("Fim\n");
40
41      delay(5000);

```

Ao executar este código existem três possibilidades de resultados que podem ser acompanhados pelo monitor serial:

1. A comunicação I²C entre o Arduino e o sensor está funcionando. Neste

caso a informação do endereço de comunicação é exibido;

2. A comunicação I²C entre o Arduino e o sensor tem algum problema e resultou em um erro de comunicação;
3. Nenhum dispositivo com comunicação I²C foi encontrado. Se isso acontecer mesmo com o circuito aparentemente montado corretamente, cheque as conexões dos pinos. Pode ser um problema de mau contato!

2.12 Sensor de temperatura e pressão (BMP280)

O sensor de temperatura e pressão BMP280 tem 6 pinos e pode trabalhar em dois tipos de comunicação: I²C ou SPI. O protocolo SPI é mais rápido que o I²C, porém no caso específico deste sensor esta vantagem efetivamente não ocorre por conta do tempo necessário do sensor para realizar a medida. Na comunicação tipo I²C os pinos têm o seguinte significado:

- VCC: alimentação. Ligar na saída 3.3V do Arduino;
- GND: terra. Deve ser ligado ao GND do Arduino;
- SCL: pino serial clock;
- SDA: pino de linha de dados (serial data);
- CSB: pino chip select (não é utilizado se a comunicação for I²C);
- SDO: pino serial data out;

Os pinos SCL e SDA devem ser conectados nos pinos corretos assim como apresentado na seção anterior.

O pino SDO controla qual o endereço I²C que será usado durante a comunicação. Se este pino não recebe sinal elétrico então o endereço de comunicação utilizado será o 0x76. Se este pino está conectado a uma alimentação de 3.3V, então o endereço de comunicação utilizado será o 0x77. Você pode testar qual o endereço está sendo utilizado rodando o código apresentado na seção anterior. Diferentes bibliotecas consideram diferentes endereços de comunicação, então fique atento!

A biblioteca recomendada para este sensor é a “Adafruit BMP280 Library” desenvolvida pela Adafruit e disponível no gerenciador de bibliotecas do Arduino IDE. Atenção: esta biblioteca considera que o endereço de comunicação I²C é o 0x77. Leve isso em consideração quando for montar seu circuito! Sugestão: antes de rodar o código abaixo, garanta que a comunicação I²C entre o Arduino e o sensor está funcionando seguindo os passos da seção anterior.

O código abaixo mostra como medir a temperatura e a pressão usando o sensor BMP280. Se atente para a linha 1 onde carregamos a biblioteca do sensor, a linha 3 onde a classe do sensor é definida, a linha 7 onde o sensor é inicializado, as linhas 10-14 onde o sensor é configurado, e as linhas 19 e 22 onde as medidas de temperatura (em Celsius) e pressão (em Pascal) são realizadas.

```
1  #include <Adafruit_BMP280.h>
2
3  Adafruit_BMP280 sensor_bmp;
4
5  void setup() {
6      Serial.begin(9600);
7      sensor_bmp.begin();
8
9      // Configuração do sensor. Apenas altere se souber o que está fazendo
10     sensor_bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,
11                             Adafruit_BMP280::SAMPLING_X2,
12                             Adafruit_BMP280::SAMPLING_X16,
13                             Adafruit_BMP280::FILTER_X16,
14                             Adafruit_BMP280::STANDBY_MS_500);
15 }
16
```

```

17 void loop() {
18     Serial.print("Temperatura (Celsius) = ");
19     Serial.println(sensor_bmp.readTemperature());
20
21     Serial.print("Pressao (Pa) = ");
22     Serial.println(sensor_bmp.readPressure());
23
24     Serial.println();
25     delay(2000);
26 }

```

2.13 Lendo e armazenando dados enviados pela comunicação serial usando Python

O Arduino IDE conta com um monitor da porta serial onde podemos acompanhar as informações enviadas pelo Arduino ao computador. Este monitor serial costuma ser suficiente para grande parte dos casos onde queremos observar os dados de sensores adquiridos pelo Arduino; ele permite que os dados sejam copiados e colados em arquivos de texto ou de planilha para tratamento futuro.

Uma forma de automatizar a coleta de dados da porta serial é usando um *script* em Python. Para isso é necessário ter a biblioteca **pyserial** instalada⁹. Abaixo temos um exemplo de código que lê a porta serial e automaticamente armazena a leitura em um arquivo.

```

import serial

# Lembrar de alterar o nome do arquivo sempre que iniciar um experimento novo
NOME_DO_ARQUIVO = 'resultado.txt'

# Usar aqui o mesmo valor da velocidade de comunicação usada para iniciar a
# comunicação serial no Arduino
BAUD_RATE = 9600

# Alterar o nome da porta serial para a porta onde o Arduino está conectado
PORTA_SERIAL = 'COM3' # usar esta notação caso esteja no windows

```

⁹Disponível em <https://pypi.org/project/pyserial/>.

```
# PORTA_SERIAL = r'/dev/ttyACM0' # usar esta notação caso esteja no linux

# loop infinito
while True:

    # iniciando comunicação serial
    with serial.Serial(PORTA_SERIAL, BAUD_RATE) as pserial:

        # lendo a porta serial, decodificando e limpando a string
        line = pserial.readline().decode('utf-8').rstrip()

        # exibindo a leitura da porta serial no terminal Python
        print(line)

        # escrevendo resultado em um arquivo
        with open(NOME_DO_ARQUIVO, 'a') as arquivo:
            arquivo.write(line + '\n')
```