**Retorno**

Essa função não retorna nada.

**Serial.available()**

Retorna a quantidades de bytes disponíveis para leitura no buffer de leitura. Essa função auxilia em loops onde a leitura dos dados só e realizada quando há dados disponível. A quantidade máxima de bytes no buffer é 64.

**Sintaxe:**

Serial.available();

**Parâmetros:**

Não passa nenhum parâmetro.

**Retorno:**

(int) - quantidade de bytes disponíveis para leitura

**Serial.read()**

Lê o byte mais recente apontado no buffer de entrada da serial.

**Sintaxe:**

Serial.read();

**Parâmetros:**

Não passa nenhum parâmetro.

**Retorno:**

(int) - O primeiro byte disponível no buffer da serial. Retorna -1 caso n tenha dado disponível.

**Serial.print()**

Escreve na serial texto em formato  ASCII. Essa função tem muitas possibilidades. Números inteiros são escritos usando um caractere ASCII para cada dígito. O mesmo ocorre para números flutuante e, por padrão, são escrito duas casas decimais. Bytes são enviados como caracteres únicos e strings e caracteres são enviados como escritos.

Vejamos alguns exemplos:

Serial.print ( 123 ); // Envia "123"

Serial.print ( 1.234567 ); // Envia "1.23"

Serial.print ( 'N' ); // Envia "N".

Serial.print ( "Hello world" ); // Envia "Hello world".

Obs.: caracteres são enviados com aspas simples e strings com aspas duplas.

Um segundo parâmetro opcional define a base numérica para formatar o valor enviado. São aceitos os seguintes parâmetros:

* BIN - binário, base 2
* OCT - octal, base 8
* HEX - hexadecimal, base 16
* DEC - decimal, base 10

Para números em ponto flutuante esse parâmetro define a quantidade de casas decimais a serem enviadas após o ponto. Exemplos:

* Serial.print(78, BIN) envia em binário "1001110"
* Serial.print(78, OCT) envia emr octal "116"
* Serial.print(78, DEC) envia em  decimal "78"
* Serial.print(78, HEX) envia em hexadecimal  "4E"
* Serial.println(1.23456, 0) envia apenas "1", sem casas decimais
* Serial.println(1.23456, 2)envia "1.23", ou seja, duas casas decimais
* Serial.println(1.23456, 4) envia  "1.2346", ou seja, 4 casas decimais

**Sintaxe:**

Serial.print(val)

Serial.print(val, format)

**Parâmetros:**

val:valor para ser escrito na serial - qualquer tipo de dado.

format: base numérica para tipos inteiros ou a quantidade de casas decimais para números flutuantes.

**Retorno:**

size\_t (long): retorna a quantidade de bytes escritos

**Serial.println()**

Funciona praticamente igual a função Serial.print(), a única diferença é que esta função acrescenta ao fim da mensagem o caractere de retorno de carro (ASCII 13 ou ‘\r’) e o caractere de nova linha(ASCII 10 ou ‘\n’). A sintaxe, os parâmetros e o retorno são os mesmos da função Serial.print().

**Serial.write()**

Escreve um byte na porta serial.

**Sintaxe:**

 Serial.write(val)

Serial.write(str)

Serial.write(buf, len)

**Parâmetros:**

val: um valor para ser enviado como um único byte.

str: uma string para ser enviada como uma sequência de bytes.

buf: um array para ser enviado como uma serie de bytes.

len: o tamanho do buffer a ser enviado.

**Retorno:**

(byte) - Retorna a quantidade de bytes escritos na serial. A leitura desse numero é opcional.

**Manipulação de dados através da comunicação serial**

**Exemplos:**

**Echo**

No Sketch a seguir é exibido como receber um caractere do computador e enviar este mesmo caractere para o computador, onde será exibido o que é digitado na terminal serial.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | /\*    echo    reenvia para o computador o dado recebido pela serial  \*/    byte byteRead;    void setup() {  //configura a comunicação seria com baud rate de 9600    Serial.begin(9600);  }    void loop() {      if (Serial.available())  //verifica se tem dados diponível para leitura    {      byteRead = Serial.read(); //le bytwe mais recente no buffer da serial      Serial.write(byteRead);   //reenvia para o computador o dado recebido    }  } |

Neste exemplo foi utilizada a função Serial.available() para verificar se há dado disponível no buffer da serial. Quando há um byte para leitura, o mesmo é lido pela função Serial.read() e armazenado na variavel byteRead. A próxima função, Serial.write(), imprime de volta o dado recebido para o computador.

**Dimmer**

Este exemplo demonstra como enviar dados do computador para controlar o brilho de um LED conectado a uma saída PWM. Este exemplo já vem com a plataforma e pode ser acessado em: *File -> Examples -> 04.Communication -> Dimmer.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | /\*  Dimmer  Demonstra como enviar dados do computador para controlar a intesindade  do brilho de um led conctado ao pino 9 do arduino.  \*/    const int ledPin = 9;      // the pin that the LED is attached to    void setup()  {    Serial.begin(9600); // configura a comunicação serial com 9600 bps    pinMode(ledPin, OUTPUT);  // configura pino do led como saída  }    void loop() {    byte brightness;  //cria uma variável para armazernar byte recebido      if (Serial.available()) //verifica se chegou algum dado na serial    {      brightness = Serial.read();//Lê o byte mais recente disponível na serial      analogWrite(ledPin, brightness);//atualiza a saída PWM do LED com valor recebido    }  } |

Com este exemplo pode -se  varia o brilho do LED conectado à saída PWM através de comandos enviados pelo PC. O byte recebido pela serial é atribuído a variável *brightness,* que na instrução a seguir é passado como parâmetro na função *analogWrite(),* definindo o brilho do LED. Junto com este exemplo é exibido um código em *processing* para variação do brilho através do clique do mouse no PC.

**Liga/Desliga LED**

Este exemplo exibe como ligar e desligar um LED conectado as saída digital da Arduino UNO através de comando enviados pelo computador.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | /\*  \* comandos via serial  \* inverte o estado do led conctado a saída 13 do arduino quando recebe o caracter 'A' pela serial  \*/    const int LED = 13;    void setup() {    Serial.begin(9600);    //configura comunicação serial com 9600 bps    pinMode(LED,OUTPUT);   //configura pino do led como saída  }    void loop() {     if (Serial.available()) //se byte pronto para leitura     {        switch(Serial.read())      //verifica qual caracter recebido      {        case 'A':                  //caso 'A'            digitalWrite(LED,!digitalRead(LED)); //inverte estado do LED          break;      }    }  } |

Neste exemplo o estado do LED ligado ao pino 13 é invertido sempre que o Arduino recebe o caractere 'A'. A estrutura desse sketch permite adicionar mais saídas para serem acionadas. Este exemplo pode ser aproveitado para uma aplicação gráfica no PC para controlar cargas automação residencial, por exemplo.

**Conclusão sobre comunicação serial na plataforma Arduino**

Dominar a comunicação serial na plataforma Arduino é essencial para o desenvolvedor de projetos. Muitos dispositivos e módulos possuem uma interface de comunicação, seja para configuração ou para comandos.

O uso de uma comunicação serial permite também o controle ou monitoramento de sistemas utilizando o computador ou mesmo outra placa eletrônica. É uma interface tradicional e bem conhecida e permite que ligue, de forma simples diferentes dispositivos.

A comunicação serial na plataforma Arduino aliada ao terminal da IDE, se torna uma ótima ferramenta para visualização de dados e Debug durante o processo de desenvolvimento, já que a plataforma não possui tais ferramentas de depuração.