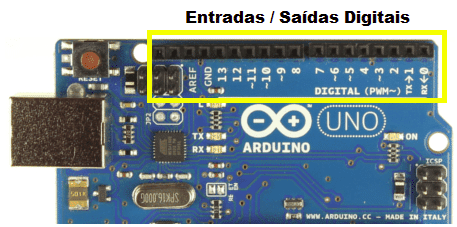
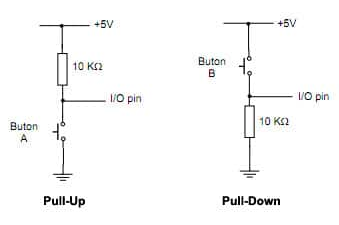
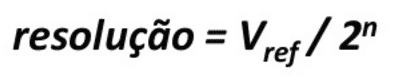
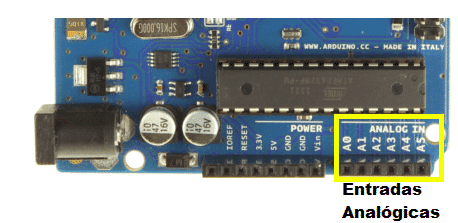
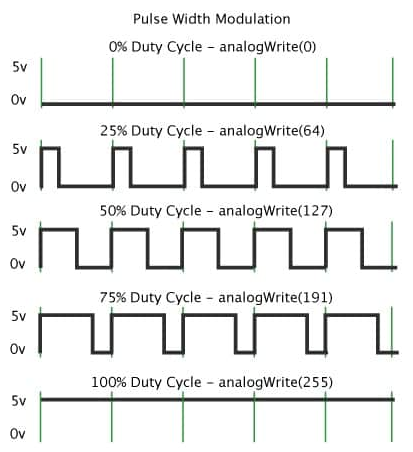
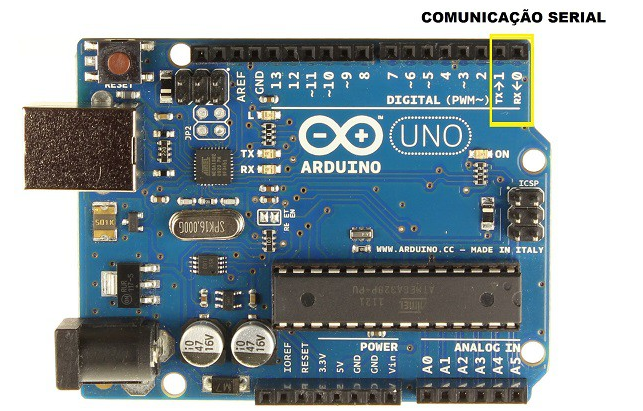
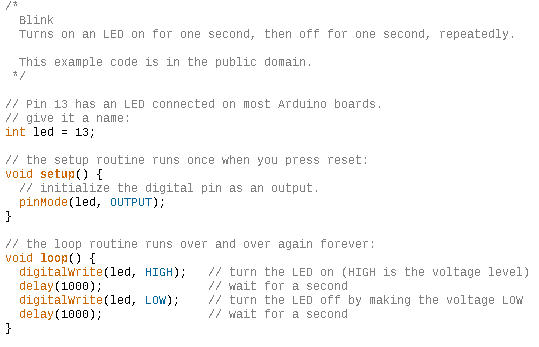
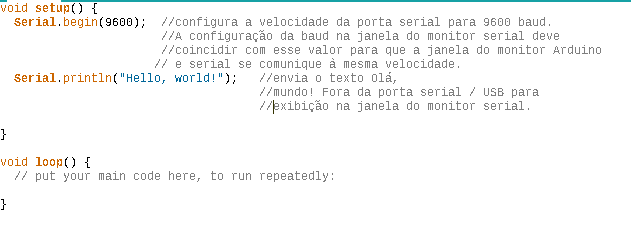
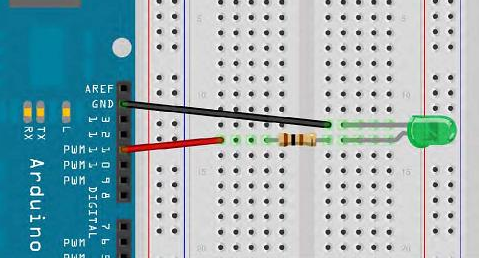
Victor Dallagnol Bento - 201520835.

Universidade Federal de Santa Maria, Engenharia de Computação.  
GEPSE - Grupo de Estudos e Pesquisa em Sistemas Embarcados.

# Trabalho 1

* Microcontrolador é um dispositivo que mistura hardware com software(circuito integrado), geralmente são embarcados(é dedicado ao dispositivo/sistema que ele controla) e ficam no interior de outro dispositivo eletrônico para que possam controlar suas funções. Podemos controlar o hardware para fazer funções específicas de uma maneira simples, flexível e potente.  
   Sendo um circuito integrado, o microcontrolador tem a possibilidade de ser programado para desempenhar diferentes tarefas específicas. Muitos kits de desenvolvimento possuem hardwares para se usar com o microcontrolador, como display de LED, botões, luzes, acionador de motor, sensores e muitos outros, no qual sua escolha depende do projeto desejado.  
   Um microcontrolador é praticamente um computador em um chip (com capacidades bem menores que um computador), que contem todos os itens, como processador, memória ROM, memória RAM, periféricos de entrada/saída, conversor analógico/digital, etc.
* Arduino é uma placa fabricada na Itália, com um projeto que teve inicio em 2015. As unidades são constituídas por controladora Atmel AVR de 8 bits, pinos digitais e analógicos de entrada e saída, entrada USB ou seria, e possui código aberto, que quando modificado, da origem a outros derivados “ino”.  
  ➙ Arduino UNO: possui processador ATMEGA328, 14 portas digitais (6 são saídas PWM), 6 são portas analógicas), a alimentação pode vir da USB o do conector para alimentação externa (7 à 12 Vdc);  
  ➙ Arduino MEGA 2560: microcontrolador ATmega2560, 54 portas digitais(15 podem ser usadas como PWM), 15 portas analógicas, clock de 16Mhz, conexão USB e conector de alimentação externa;  
  ➙ Arduino LEONARDO: microcontrolador ATMEGA32u4, 20 portas digitais(7 podem ser usadas como PWM), 12 como portas analógicas, clock de 16Mhz, conexão micro-USB e conexão para alimentação externa;  
  ➙ Arduino DUE: microcontrolador ARM de 32 bits e 512kb de memória, 54 portas digitais(12 podem ser usadas como PWM), 12 portas analógicas, 4 chips controladores de poetas seriais, conexão USB e conector para alimentação externa;  
  ➙ Arduino MEGA ADK: microcontrolador ATMEGA2560, conexão USB dedicada a ligação de dispositivos Android, 54 portas digitais(15 podem ser usadas como PWM), 16 portas analógicas, 4 chips dedicados a comunicação serial, clock de 16Mhz, conexão USB e conector para alimentação externa;  
  ➙ Arduino NANO: microcontrolador ATMEGA328, não possui conector de alimentação externa, só é alimentada por um conector USB Mini-B, possui memória de 32Kb ou 16Kb (dependendo da versão) e 2K da memória são usados como bootloader.  
  ➙ Arduino PRO MINI: microcontrolador ATMEGA168, roda a 8Mhz ou 16Mhz (dependendo da versão), possui 14 portas digitais(6 podem ser usadas como PWM), 8 portas analógicas, não possui conexão USB nem conector para alimentação externa.
* A alimentação externa é feita através do conector Jack com positivo no centro, onde o valor de tensão da fonte externa deve estar entre os limites 6V. a 20V, porém se alimentada com uma tensão abaixo de 7V, a tensão de funcionamento da placa, que no Arduino Uno é 5V, pode ficar instável e nao chegar a 5V, e quando alimentada com tensão acima de 12V, o regulador de tensão da placa pode sobreaquecer e danificar a placa. Dessa forma, é recomendado para tensões de fonte externa valores de 7V a 12V.  
   As saídas digitais do Arduino suportam no máximo 40mA, por fator de segurança é recomendável trabalhar com no máximo 20mA. Porém, não basta controlar o total e corrente fornecida por cada pino digital, pois na documentação do Arduino diz que a corrente total que o microcontrolador é capaz de suportar são de 200mA. Se as somas de todas as correntes fornecidas pelo Arduino ultrapassar esse limite, o mesmo será danificado.
* ➙ E/S Digitais:   
   A placa do Arduino UNO possui 14 pinos que podem ser configurados como entradas ou saídas digitais. Estes pinos são numerados de 0 a 13.  
    
  Antes de usar esses pinos precisamos configura-los como entrada ou saída digital, conforme necessitamos. Por padrão os pinos estão configurados como entradas digitais, dessa forma o pino é colocado em um estado de alta impedância, equivalente a um resistor de 100 Megaohms em série com o circuito monitorado. O pino absorve uma corrente muito baixa do circuito que está monitorando. Devido a essa característica de alta impedância, quando um pino colocado com entrada digital encontrasse flutuando (sem ligação definida), o nível de tensão presente nesse pino fica variando não podendo ser determinado um valor estável devido a ruído elétrico e até mesmo capacitância de entrada do pino. Para resolver esse problema é necessário colocar um resistor de pull up (ligado a +5V) ou um resistor de pull down (ligado a GND) conforme a necessidade. Esses resistores garantem nível lógico estável.  
    
     
   Quando um pino é configurado com saída, ele se encontra em estado de baixa impedância. Dessa forma, o pino pode fornecer ou drenar corrente para um circuito externo. A corrente máxima que um pino pode fornecer ou drenar é de 40 mA, porém a soma das correntes não pode ultrapassar 200 mA. Deve-se ficar atento a corrente maiores que este limite e a curto-circuitos que podem danificar o transistor de saída danificando o pino e até mesmo queimando o microcontrolador. Essa é uma característica perigosa para a placa Arduino e seria interessante se tivessem resistores ou algum tipo de proteção em todos os pinos utilizados como saída para limitar a corrente em uma situação anormal.  
   A plataforma Arduino possui funções para trabalhar com entradas e saídas digitais que abstraem toda a configurações dos registradores que configuram e acessam os pino de I/O. Isso torna a programação do Arduino realmente fácil e esse é seu encanto. Essas funções são:  
     
   ***void pinMode();***  
   Essa função é utilizada para configurar um pino como entrada ou saída digital. Ela geralmente é utilizada dentro da função setup(). Apresenta as seguintes características:  
  Sintaxe:  
     
   ***pinMode(pino, modo);***Parâmetros:  
  *pino*: Número correspondente ao pino que se deseja configurar, conforme a placa que está trabalhando. No caso da Arduino UNO  pode ser de 0 a 13;  
  *modo*: Modo que deseja configurar o pino. INPUT, INPUT\_PULLUP, OUTPUT.  
  *INPUT*: Entrada digital;  
  *INPUT\_PULLUP*: Entrada digital com resistor de pull-up (ligado ao VCC) interno habilitado;  
  O*UTPUT*: Saída digital;  
    
  Retorno:  
  Não tem retorno.  
    
   ***Int digitalRead();***Essa função lê o valor presente em um pino digital. Este valor pode ser HIGH ou LOW. Apresenta as seguintes características:  
  Sintaxe:  
    
   ***digitalRead(pino);***Parâmetros:  
  *pino*: valor correspondente ao pino que se deseja ler.  
    
  Retorno: HIGH ou LOW.  
    
     
   ***Void digitalWrite();***A função digitalWrite() coloca um nível lógico Alto (HIGH, 5V) ou baixo (LOW, 0V) em um pino configurado como saída digital.  
  Sintaxe:  
    
   ***digitalWrite(pino, valor);***Parâmetros:   
  *pino*: Número correspondente ao pino;  
  *valor*: HIGH OU LOW  
    
  Retorno: Não tem retorno  
    
    
  ➙ E/S Analógicas:  
   As entradas digitais só podem assumir dois estados, HIGH e LOW, ou seja, 0 V ou 5 V. Dessa forma só é possível ler apenas dois estados.   
   O microcontrolador da Arduino trabalha internamente com dados digitais, portanto é necessário traduzir um sinal analógico para um valor digital. A técnica utilizada para leitura de um sinal analógico pelo Arduino é a conversão analógica digital. Essa técnica consiste em converter o sinal analógico para um valor digital, dessa forma se pode quantificar o sinal presente no pino. Esse processo é feito pelo conversor Analógico digital, ADC ou conversor A/D.  
   Um conversor A/D quantifica o valor analógico conforme a quantidade de bits da sua resolução. A resolução de um conversor A/D é dada pela seguinte equação:  
     
  onde:  
  *Vref*: tensão de referência do conversor A/D;  
  *n*: número de bits do conversor.  
   O conversor A/D do microcontrolador ATmega328 possui 10 bits de resolução, a sua tensão de entrada pode variar de 0 V até o valor de VCC e possui referência interna selecionável de 1,1 V.  
   Dessa forma quando está trabalhando com a referência em VCC o menor valor que pode ser lido será:  
  5 V/1024 = 4,88 mV. Esse é o valor de degrau para uma conversão em 10 bits com referência em 5 V.  
   Caso  trabalhe com a referência interna de 1,1V a resolução será:  
     
  Nota-se que o passo é bem menor para esse valor de referência.  
   Se a referência externa  for selecionada, a resolução dependerá do valor de tensão aplicada ao pino AREF.  
   A placa Arduino UNO possui 6 canais de conversor analógico digital. Essas entradas são nomeadas de A0 a A5:  
    
   A plataforma Arduino possui funções para trabalhar com entradas analógicas, as quais abstraem a configuração dos registradores de configuração do conversor A/D, assim como selecionam o canal conforme o pino passado como parâmetro.   
    
   ***analogReference(tipo)***Essa função configura a referência de tensão para a conversão analógica/digital, usando esse valor como o máximo para a entrada analógica.   
   Os tipos possíveis de configurações são:  
  *DEFAULT*: a tensão padrão para conversão é a tensão de alimentação da placa. 5 V para placas alimentadas com 5 V  e 3,3 V  para placas alimentadas com 3,3 V;  
  *INTERNAL*: referência interna de 1,1V no Atmega168 e Atmega328, e 2,56 V no ATmega8;  
  *INTERNAL1V1*: referência de 1,1V, apenas no Arduino Mega;  
  *INTERNAL2V56*: referência interna de 5,6 V, apenas no Arduino Mega;  
  *EXTERNAL*: referência de tensão aplicada no pino AREF (valor entre 0 e 5V).  
    
  Sintaxe:  
    
   **analogReference(tipo);**  
  Parâmetros:  
  Tipo: DEFAULT, INTENAL, INTERNAL1V1, INTERNAL2V56, EXTERNAL.  
    
  Retorno: Essa função não tem retorno algum.  
    
    
   **Int analogRead(pino);**  
  Essa função lê o valor presente em um pino configurado como entrada analógica. Internamente o Arduino possui um conversor A/D de 10 bits. Dessa forma o valor retornado por esta função estará na faixa de 0 a 1023 conforme o valor presente no pino.  
   O tempo para leitura pela função analogRead() é por volta de 100 micro segundos, dessa forma a máxima frequência de leitura que se pode ter é de 10000 vezes por segundo.  
  Sintaxe:  
    
   **analogRead(pino);**  
  Parâmetros:  
  Pino: valor do pino configurado como entrada analógica (0 a 5 na maioria da placas, 0 a 7 na MINI e NANO, 0 a 15 na MEGA).  
    
  Retorno: int (0 a 1023).
* PWM é uma tecnica para variação do valor médio de uma forma de onda periódica. Que pode ser usada como controle de velocidade de motores, variação da luminosidade de leds, geração de sinais analógicos, geração de sinais de áudio, etc. Essa tecnica consiste em manter a frequencia de uma onda quadratica fixa e variar o tempo que o sinal fica em nivel lógico alto(duty cycle).  
     
  Para cada variação do Duty cycle teremos uma variação na tensão de saída. A formula para obter a tensão média de saída é:  
   Vout = (dusy cycle/100) \* Vcc  
  Vout = Tensão de saída (V).  
  Duty Cycle = Valor do ciclo ativo do PWM em %.  
  Vcc = Tensão de alimentação em V.  
   No Arduino UNO os pinos específicos para saídas PWM estão indicados pelo caracter ‘~’ na frente de seu número.  
   A função analogWrite() escreve um valor de PWM em um pino digital que possui a função PWM. Após a chamada dessa função, o pino passa a operar com uma onda quadrada de frequência fixa e com duty cycle conforme valor passado pela função. A frequência dessa onda, na maioria dos pinos é em torno de 490 Hz, porém, os pinos 5 e 6 da Arduino UNO operam em 980 Hz.  
   Para utilizar a função analogWrite() , deve-se configurar o pino correspondente como saída digital. É interessante notar que essas saídas não são conversores digital-analógico como o nome sugere, e estes pinos não estão relacionados às entradas analógicas.  
    
  Sintaxe:  
    
   ***analogWrite(pino, valor);***Onde:  
  *Pino:* corresponde ao pino que será gerado o sinal PWM;  
  *Valor:* corresponde ao duty cycle, ou seja, o valor que permanecerá em nível alto o sinal.  
   O valor deve ser de 0 a 255 onde com 0 a saída permanece sempre em nível baixo e 255 a saída permanece sempre em nível alto.
* A UART (Comunicação Serial) possibilita a comunicação entre a placa e um computador (ou outro dispositivo). Através desse canal é realizado o upload do código para a placa.  
   No Arduino UNO esse canal está ligado aos pinos 0 (RX) e 1(TX), que também são responsáveis pela tradução do sinal para comunicação USB com o computador.  
    
   Dispositivos conectados nesses pinos podem interferir no upload do programa. Por isso, é recomendável desconectar os dispositivos ligados a esses pinos antes de fazer o upload.  
   O sinal de comunicação é um sinal TTL de 5V. Para conectar com um dispositivo que não tem o mesmo nível de tensão é necessário usar um conversor de nível.  
   Além do recurso de upload através da comunicações serial, a IDE trás um terminal serial que auxilia no recebimento e envio de dados para a placa sem a necessidade de recorrer a uma ferramenta externa. Para acessar essa ferramenta basta clicar no ícone Serial Monitor ou acessar o menu Tools> Serial Monitor.   
   A ferramenta é bem simples, contendo apenas alguns parâmetros de configuração, onde se pode definir a taxa de envio (baud rate). Possui dois campos, um onde pode ser inserido a mensagem a ser enviada e outro maior onde é exibido os caracteres enviados pela placa para o computador.  
   A plataforma Arduino possui em sua biblioteca uma variedade de funções para manipulação de dados através de comunicação serial. Essas funções auxiliam o desenvolvedor em tarefas mais complexas de envio e recebimento de dados.   
    
   Serial.begin()  
  É a primeira função a ser utilizada quando vai trabalhar com a comunicação serial. Ela configura a taxa de comunicação em bits por segundo (baud rate). Possui um segundo parâmetro opcional para a definição da quantidade de bits, paridade e stop bits. Se for omitido esse parâmetro o padrão será 8 bits, sem paridade e 1 stop bit.  
    
  Sintaxe:  
   ***Serial.begin(speed)  
   Serial.begin(speed, config)***Parâmetros:  
  speed: velocidade em bit por segundo (baud rate) - long  
  config: configura a quantidade de bits, paridade e stop bits. Os valores válidos são :  
  SERIAL\_5N1  
  SERIAL\_6N1  
  SERIAL\_7N1  
  SERIAL\_8N1 (padrão)  
  SERIAL\_5N2  
  SERIAL\_6N2  
  SERIAL\_7N2  
  SERIAL\_8N2  
  SERIAL\_5E1  
  SERIAL\_6E1  
  SERIAL\_7E1  
  SERIAL\_8E1  
  SERIAL\_5E2  
  SERIAL\_6E2  
  SERIAL\_7E2  
  SERIAL\_8E2  
  SERIAL\_5O1  
  SERIAL\_6O1  
  SERIAL\_7O1  
  SERIAL\_8O1  
  SERIAL\_5O2  
  SERIAL\_6O2  
  SERIAL\_7O2  
  SERIAL\_8O2  
    
  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.
* Apesar da placa ser falsificada, por usar um sistema linux, nao foi necessário baixar o driver do conversor. A ARDUINO IDE foi instalada via terminal através do comendo “sudo apt-get install arduino”.
* As bibliotecas que já vem com a IDE são:  
  ➙ *EEPROM*  - Lendo e escrevendo para armazenamento “permanente”.  
  ➙ *Ethernet / Ethernet 2* - Para conectar-se à internet usando o Arduino Ethernet Shield, Arduino Ethernet Shield 2 e Arduino Leonardo ETH  
  ➙ *Firmata -* Para se comunicar com aplicativos no computador usando um protocolo serial padrão.  
  ➙ *GSM - Para conectar-se a uma rede GSM / GRPS com o escudo shield.*  
  ➙ LiquidCrystal - para controlar displays de cristal líquido (LCDs)  
  ➙ SD - para ler e escrever cartões SD  
  ➙ Servo - para controle de servo-motores  
  ➙ SPI - para comunicação com dispositivos usando o barramento de interface periférica serial (SPI)  
  ➙ SoftwareSerial - para comunicação serial em qualquer pino digital.A versão 1.0 e posterior da Arduino incorporam a biblioteca NewSoftSerial da Mikal Hart como SoftwareSerial.  
  ➙ Stepper - para controlar motores passo a passo  
  ➙ TFT - para desenhar texto, imagens e formas na tela Arduino TFT  
  ➙ WiFi - para conectar-se à internet usando o escudo Wi-Fi Arduino  
  ➙ Wire - Two Wire Interface (TWI / I2C) para enviar e receber dados através de uma rede de dispositivos ou sensores.  
  ➙ As bibliotecas Matrix e Sprite não fazem mais parte da distribuição principal.  
   Para acessar os exemplos precisamos ir em “File ➙ Examples” e selecionar o exemplo desejado.  
  1. Basics:  
  ➙ Analog Read Serial: Leia um potenciômetro, imprima seu estado para o Arduino Serial Monitor.  
  ➙ Bare Minimum: O mínimo de código necessário para iniciar um esboço Arduino.  
  ➙ Blink: liga e desliga o LED.  
  ➙ Digital Read Serial: Leia um interruptor, imprima o estado no Arduino Serial Monitor.  
  ➙ Fade: Demonstra o uso da saída analógica para desvanecer um LED.  
  ➙ Read Analog Voltage: lê uma entrada analógica e imprime a tensão no Monitor Serial.  
    
  2. Digital  
  ➙ Blink Without Delay: pisca um LED sem usar a função delay ().  
  ➙ Button: use um botão para controlar um LED.  
  ➙ Debounce: Leia um botão, ruído de filtragem.  
  ➙ Digital Input Pullup: Demonstra o uso de INPUT\_PULLUP com pinMode ().  
  ➙ Stage Change Detection: conte o número de pressionamentos de botão.  
  ➙ Tone Keyboard: Um teclado musical de três teclas que usa sensores de força e um alto-falante piezo.  
  ➙ Tone Melody: Reproduz uma melodia com um alto-falante Piezo.  
  ➙ Tone Multiple: Reproduzir tons em vários alto-falantes sequencialmente usando o comando tone ().  
  ➙ Tone Pitch Follower: Jogue um tom em um altofalante piezo dependendo de uma entrada analógica.  
    
  3. Analógico  
  ➙ Analog In Out Serial: Leia um pino de entrada analógico, mapeie o resultado e, em seguida, use esses dados para diminuir ou iluminar um LED.  
  ➙ Analog Input: use um potenciômetro para controlar o piscar de um LED.  
  ➙ Analog Write Mega: Fade 12 LEDs ligados e desligados, um por um, usando uma placa Arduino Mega.  
  ➙ Calibration: Defina um valor máximo e mínimo para os valores esperados do sensor analógico.  
  ➙ Fading: Use uma saída analógica (PWM pin) para desvanecer um LED.  
  ➙ Smoothing: liso leituras múltiplas de uma entrada analógica.  
    
  4. Comunicação  
  Estes exemplos incluem código que permite que o Arduino converse com o processamento de esboços executados no computador. Para obter mais informações ou para baixar Processamento, consulte processing.org. Existem também patches Max / MSP que podem se comunicar com cada esboço Arduino também. Para mais informações sobre Max / MSP, consulte Ciclismo 74.  
  ➙ ASCIITável: demonstra as funções avançadas de saída serial da Arduino.  
  ➙ Dimmer: mova o mouse para alterar o brilho de um LED.  
  ➙ Graph: envie dados para o computador e grafica-o em Processamento.  
  ➙ Midi: envie mensagens de nota MIDI em série.  
  ➙ Multi Serial Mega: Use duas das portas seriais disponíveis no Arduino Mega.  
  ➙ Physical Pixel: ligue e desligue um LED enviando dados para seu Arduino de Processamento ou Max / MSP.  
  ➙ Read ASCII String: analise uma seqüência de números inteiros separados por vírgulas para desaparecer um LED.  
  ➙ Serial Call Response: envie várias variáveis ​​usando um método de chamada e resposta (handshaking).  
  ➙ Serial Call Response ASCII: envie múltiplas variáveis ​​usando uma chamada-e-Método de resposta (handshaking) e codificação ASCII dos valores antes de enviar.  
  ➙ Serial Event: demonstra o uso do SerialEvent ().  
  ➙ Virtual Color Mixer: envie múltiplas variáveis ​​do Arduino para o seu computador e lê-las em Processamento ou Max / MSP.  
    
  5. Estruturas de controle  
  ➙ Arrays: uma variação no exemplo For Loop que demonstra como usar uma matriz.  
  ➙ For Loop Iteration: Controle LEDs múltiplos com um loop for.  
  ➙ If Statement Conditional: use uma 'declaração if' para alterar as condições de saída com base na alteração das condições de entrada.  
  ➙ Switch Case: Como escolher entre um número discreto de valores.  
  ➙ Switch Case 2: Um segundo exemplo de exemplo de troca,Mostrando como tomar diferentes ações com base nos caracteres recebidos na porta serial.  
  ➙ While Statement Conditional: Como usar um loop while para calibrar um sensor enquanto um botão está sendo lido.  
    
  6. Sensores  
  ➙ ADXL3xx: Leia um acelerômetro ADXL3xx.  
  ➙ Knock: Detecta batidas com um elemento piezo.  
  ➙ Memsic2125: acelerômetro de dois eixos.  
  ➙ Ping: Detectar objetos com um buscador de alcance ultra-sônico.  
    
  7. Exibir  
  ➙ Bar Graph: faça um gráfico de barras de LEDs.  
  ➙ RowColumnScanning: controle uma matriz 8x8 de LEDs.  
    
  8. Cordas  
  ➙ Character Analysis: use os operadores para reconhecer o tipo de personagem com o qual estamos lidando.  
  ➙ String Addition Operator: adicione strings juntos de várias maneiras.  
  ➙ String Append Operator: Use o operador + = e o método concat () para acrescentar coisas a Strings.  
  ➙ String Case Changes: Mude o caso de uma string.  
  ➙ String Characters: Obter / definir o valor de um caractere específico em uma string.  
  ➙ String Comparison Operators: compare cadeias em ordem alfabética.  
  ➙ String Constructors: Inicialize objetos de string.  
  ➙ String Index of: Procure a primeira / última instância de um personagem em uma string.  
  ➙ String Length & String Length Trim: Obter e cortar o comprimento de uma string.  
  ➙ String Replace: Substitua caracteres individuais em uma string.  
  ➙ String Start With Ends With: Verifique quais os caracteres / substrings com os quais uma determinada cadeia começa ou termina.  
  ➙ String Substring: Procure por "frases" dentro de uma string determinada.  
  ➙ String To Int: Permite converter uma String para um número inteiro.  
    
  9. USB  
  Os exemplos de teclado e mouse são exclusivos do Leonardo, Micro e Due. Eles demonstram o uso de bibliotecas exclusivas do quadro.  
  ➙ Keyboard Logout: efetua o registro do usuário atual com os comandos das teclas.  
  ➙ Keyboard Message: envia uma string de texto quando um botão é pressionado.  
  ➙ Keyboard Reprogram: abre uma nova janela no IDD Arduino e reprograma o Leonardo com um simples programa de piscadela.  
  ➙ Keyboard Serial: lê um byte da porta serial e envia uma batida de tecla.  
  ➙ Keyboard Mouse Control: demonstra os comandos do mouse e do teclado em um único programa.  
  ➙ Button Mouse Control: controle o movimento do cursor com 5 botões.  
  ➙ Joystick Mouse Contol: Controla o movimento do cursor de um computador com um Joystick quando um botão é pressionado.  
    
  10. Starterkit & BasicKit  
  Os Tutoriais para os Exemplos do StarterKit estão disponíveis no Livro do Projeto incluído no kit, se você comprar um BasicKit você poderá acessar os projetos on-line no Project Ignite.  
    
  11. Arduino ISP  
  O Arduino ISP transforma seu Arduino em um programador em circuito para re-programar chips AtMega. Útil quando você precisa recarregar o carregador de inicialização em um Arduino, se você estiver indo de Arduino para um AtMega em um boardboard, ou se você estiver fazendo o seu próprio circuito compatível com Arduino em uma placa de teste.
* BLINK CODE:  
  File > Examples > 01.Basics > Blink  
    
  Depois clicamos em Upload para enviar para o Arduino.
* HELLO, WORLD!  
  File > Examples > 01.Basics > BareMinimum  
    
  Depois disso precisamos salvar o programa.   
  File > Save As...  
  Depois de salvo, Clicamos em upload para enviar o código para o Arduino e então clicamos em Serial Monitor para visualizar o terminal.
* Como o LED Onboard do Arduino não possui saída PWM(~), fica impossivel controlar o nível de brilho, pois sua saída é apenas HIGH (5V) ou LOW (0V). A saída PWM emula uma saída analógica. Concluindo isso, foi necessário escolher um pino que suporta saida PWM, pegamos um resistor para limitar a corrente que chega no LED e conectamos todos em série, como a imagem abaixo:   
    
  Como o arduino trabalha com 5V, o LED usado ascendia entre 1,6V e 3V, por isso o resistor limitou a corrente até essa faixa. O código usado está abaixo:  
  