

O Guia Maker



Sobre o Autor:



Olá , me chamo Allan Araújo , sou fundador da Startup Basic Code , na qual trabalhamos com educação a distância no ramo de T. I e Telecomunicações . Sou programador e cursei a graduação de Sistemas de telecomunicações pelo Instituto Federal do Amazonas .

Nesse guia você aprenderá de tudo um pouco para dar os primeiros passos para criações de projetos com arduino .

Contato :

Fb.com/basicode

Instagram.com/basicodetech

Website : www.basiccode.com.br

Arduino Maker

O Arduino é uma plataforma de protótipo (código aberto) baseada em hardware e software fáceis de usar. Consiste em uma placa de circuito, que pode ser programada (conhecida como microcontrolador) e em um software pronto chamado Arduino IDE (Ambiente de Desenvolvimento Integrado), que é usado para escrever e carregar o código do computador na placa física.

O Arduino fornece um fator de forma padrão que divide as funções do microcontrolador em um pacote mais acessível.

Público

Este e-book tutorial é destinado a estudantes ou entusiastas entusiastas. Com o Arduino, é possível conhecer rapidamente o básico de microcontroladores e sensores e começar a construir protótipos com muito pouco investimento.

Este tutorial destina-se a facilitar a introdução ao Arduino e suas várias funções.

Pré-requisitos

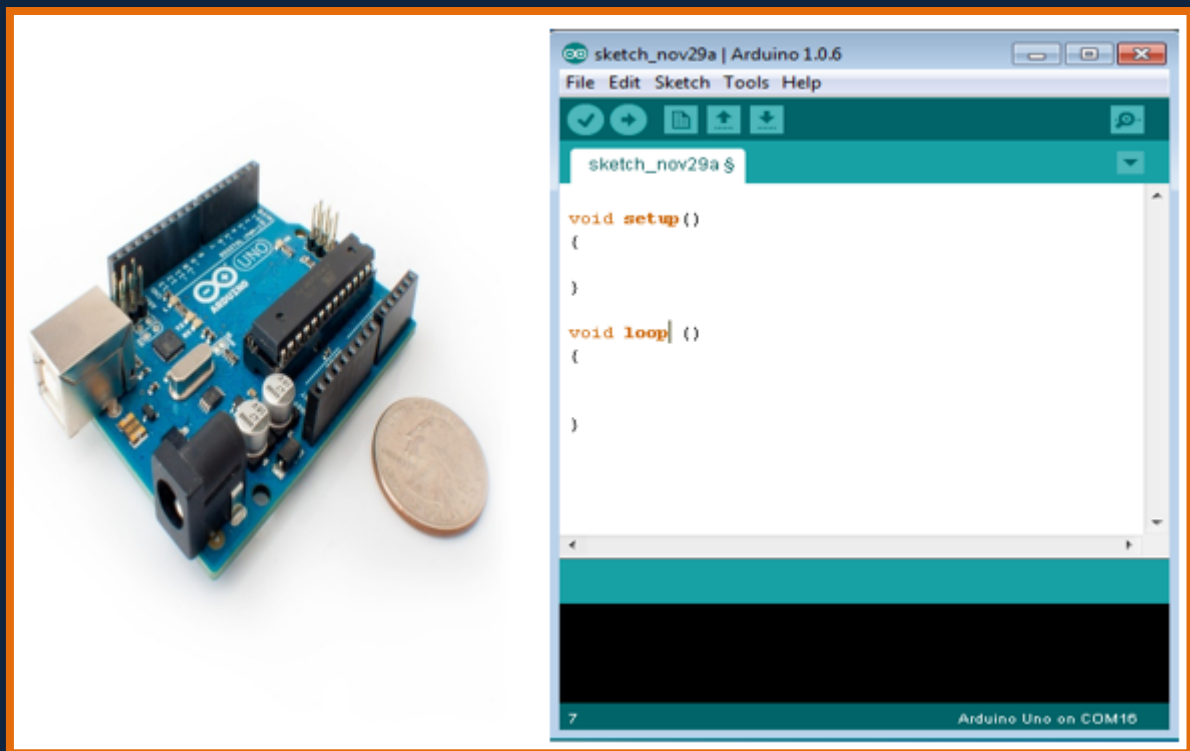
Antes de começar a prosseguir com este e-book, presumimos que você já esteja familiarizado com o básico de C e C ++.

Arduino - Visão Geral

O Arduino é uma plataforma de protótipo (código aberto) baseada em hardware e software fáceis de usar. Consiste em uma placa de circuito, que pode ser programada (conhecida como microcontrolador) e em um software pronto chamado Arduino IDE (Ambiente de Desenvolvimento Integrado), que é usado para escrever e carregar o código do computador na placa física.

Os principais recursos são -

- As placas Arduino são capazes de ler sinais de entrada analógicos ou digitais de diferentes sensores e transformá-lo em uma saída, como ativar um motor, ligar / desligar o LED, conectar-se à nuvem e muitas outras ações.
- Você pode controlar as funções da sua placa enviando um conjunto de instruções ao microcontrolador na placa via Arduino IDE (referido como upload de software).
- Diferentemente da maioria das placas de circuito programáveis anteriores, o Arduino não precisa de um hardware extra (chamado de programador) para carregar um novo código na placa. Você pode simplesmente usar um cabo USB.
- Além disso, o Arduino IDE usa uma versão simplificada do C ++, facilitando o aprendizado do programa.
- Finalmente, o Arduino fornece um fator de forma padrão que divide as funções do microcontrolador em um pacote mais acessível.



Tipos de placa

Vários tipos de placas Arduino estão disponíveis, dependendo dos diferentes microcontroladores usados. No entanto, todas as placas do Arduino têm uma coisa em comum: elas são programadas através do IDE do Arduino.

As diferenças são baseadas no número de entradas e saídas (número de sensores, LEDs e botões que você pode usar em uma única placa), velocidade, tensão operacional, fator de forma etc. Algumas placas são projetadas para serem incorporadas e não têm programação interface (hardware), que você precisaria comprar separadamente. Alguns podem funcionar diretamente a partir de uma bateria de 3,7V, outros precisam de pelo menos 5V.

Aqui está uma lista de diferentes placas Arduino disponíveis.

Placas Arduino baseadas no microcontrolador ATMEGA328

Nome do Conselho	Volt de operação	Velocidade do relógio	E / S digital	Entradas Analógicas	PWM	UART	Interface de programação
Arduino Uno R3	5V	16MHz	14	6	6	1	USB via ATmega16U2
Arduino Uno R3	5V	16MHz	14	6	6	1	USB via ATmega16U2

SMD							
Placa vermelha	5V	16MHz	14	6	6	1	USB via FTDI
Arduino Pro 3.3v / 8 MHz	3.3V	8MHz	14	6	6	1	Cabeçalho compatível com FTDI
Arduino Pro 5V / 16MHz	5V	16MHz	14	6	6	1	Cabeçalho compatível com FTDI
Arduino mini 05	5V	16MHz	14	8	6	1	Cabeçalho compatível com FTDI
Arduino Pro mini 3.3v / 8mhz	3.3V	8MHz	14	8	6	1	Cabeçalho compatível com FTDI
Arduino Pro mini 5v / 16mhz	5V	16MHz	14	8	6	1	Cabeçalho compatível com FTDI
Arduino Ethernet	5V	16MHz	14	6	6	1	Cabeçalho compatível com FTDI
Arduino Fio	3.3V	8MHz	14	8	6	1	Cabeçalho compatível com FTDI
Placa principal do LilyPad Arduino 328	3.3V	8MHz	14	6	6	1	Cabeçalho compatível com FTDI

Quadro simples LilyPad Arduino	3.3V	8MHz	9	4	5	0 0	Cabeçalho compatível com FTDI
--------------------------------	------	------	---	---	---	-----	-------------------------------

Placas Arduino baseadas no microcontrolador ATMEGA32u4

Nome do Conselho	Volt de operação	Velocidade do relógio	E / S digital	Entradas Analógicas	PWM	UART	Interface de programação
Arduino Leonardo	5V	16MHz	20	12	7	1	USB nativo
Pro micro 5V / 16MHz	5V	16MHz	14	6	6	1	USB nativo
Pro micro 3.3V / 8MHz	5V	16MHz	14	6	6	1	USB nativo
LilyPad Arduino USB	3.3V	8MHz	14	6	6	1	USB nativo

Placas Arduino baseadas no microcontrolador ATMEGA2560

Nome do Conselho	Volt de operação	Velocidade do relógio	E / S digital	Entradas Analógicas	PWM	UART	Interface de programação
Arduino Mega 2560 R3	5V	16MHz	54	16	14	4	USB via ATmega16U2B
Mega Pro 3.3V	3.3V	8MHz	54	16	14	4	Cabeçalho compatível com FTDI
Mega Pro 5V	5V	16MHz	54	16	14	4	Cabeçalho compatível com

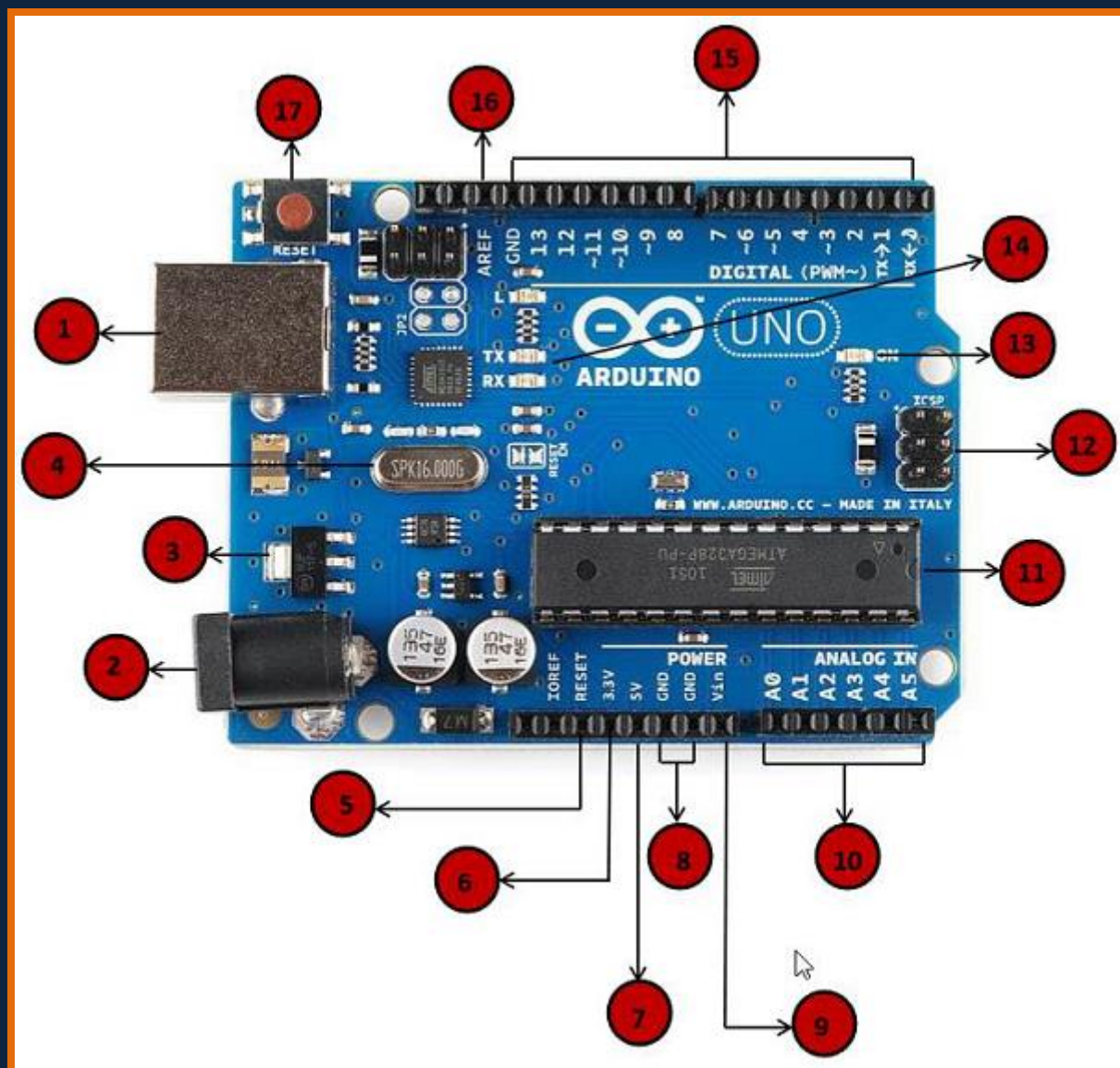
							FTDI
Mega Pro Mini 3.3V	3.3V	8MHz	54	16	14	4	Cabeçalho compatível com FTDI




Placas Arduino baseadas no microcontrolador AT91SAM3X8E






Nome do Conselho	Volt de operação	Velocidade do relógio	E / S digital	Entradas Analógicas	PWM	UART	Interface de programação
Arduino Mega 2560 R3	3.3V	84MHz	54	12	12	4	Nativo USB

Arduino - Descrição da Placa

Neste capítulo, aprenderemos sobre os diferentes componentes na placa Arduino. Estudaremos a placa Arduino UNO porque é a placa mais popular da família de placas Arduino. Além disso, é a melhor placa para começar com eletrônica e codificação. Algumas placas parecem um pouco diferentes das fornecidas abaixo, mas a maioria dos Arduinos tem a maioria desses componentes em comum.



	USB de alimentação A placa Arduino pode ser alimentada usando o cabo USB do seu computador. Tudo o que você precisa fazer é conectar o cabo USB à conexão USB (1).
	Potência (Barrel Jack) As placas Arduino podem ser alimentadas diretamente da fonte de alimentação CA conectando-a ao Barrel Jack (2).
	Regulador de voltagem A função do regulador de tensão é controlar a tensão fornecida à placa Arduino e estabilizar as tensões CC usadas pelo processador e outros elementos.

	<p>Oscilador de cristal</p> <p>O oscilador de cristal ajuda o Arduino a lidar com problemas de tempo. Como o Arduino calcula o tempo? A resposta é, usando o oscilador de cristal. O número impresso na parte superior do cristal do Arduino é 16.000H9H. Diz-nos que a frequência é 16.000.000 Hertz ou 16 MHz.</p>
	<p>Arduino Reset</p> <p>Você pode redefinir sua placa Arduino, ou seja, iniciar seu programa desde o início. Você pode redefinir a placa UNO de duas maneiras. Primeiro, usando o botão de reset (17) na placa. Segundo, você pode conectar um botão de reset externo ao pino do Arduino chamado RESET (5).</p>
	<p>Pinos (3.3, 5, GND, Vin)</p> <ul style="list-style-type: none"> • 3.3V (6) - Alimentação 3.3 volt de saída • 5V (7) - Fornece 5 volts de saída • A maioria dos componentes usados na placa Arduino funciona bem com 3,3 volts e 5 volts. • GND (8) (terra) - Existem vários pinos GND no Arduino, qualquer um dos quais pode ser usado para aterrar seu circuito. • Vin (9) - Esse pino também pode ser usado para alimentar a placa Arduino a partir de uma fonte de energia externa, como a fonte de alimentação CA.
	<p>Pinos analógicos</p> <p>A placa UNO do Arduino possui seis pinos de entrada analógica A0 a A5. Esses pinos podem ler o sinal de um sensor analógico, como o sensor de umidade ou sensor de temperatura, e convertê-lo em um valor digital que pode ser lido pelo microprocessador.</p>
	<p>Microcontrolador principal</p> <p>Cada placa Arduino possui seu próprio microcontrolador (11). Você pode assumi-lo como o cérebro do seu quadro. O principal IC (circuito integrado) do Arduino é um pouco diferente de placa para placa. Os microcontroladores são geralmente da empresa ATMEL. Você deve saber qual é o IC da sua placa antes de carregar um novo programa no IDE do Arduino. Esta informação está disponível na parte superior do IC. Para mais detalhes sobre a construção e funções do IC, consulte a folha de dados.</p>

	<p>Pino ICSP</p> <p>Principalmente, o ICSP (12) é um AVR, um pequeno cabeçalho de programação para o Arduino que consiste em MOSI, MISO, SCK, RESET, VCC e GND. É frequentemente chamado de SPI (Serial Peripheral Interface), que pode ser considerada uma "expansão" da saída. Na verdade, você está escravizando o dispositivo de saída para o mestre do barramento SPI.</p>
	<p>LED indicador de energia</p> <p>Esse LED deve acender quando você conecta o seu Arduino a uma fonte de energia para indicar que sua placa está ligada corretamente. Se essa luz não acender, há algo errado com a conexão.</p>
	<p>LEDs TX e RX</p> <p>Na sua placa, você encontrará dois rótulos: TX (transmissão) e RX (recebimento). Eles aparecem em dois lugares no quadro da Arduino UNO. Primeiro, nos pinos digitais 0 e 1, para indicar os pinos responsáveis pela comunicação serial. Segundo, os LEDs TX e RX (13). O led TX pisca com velocidade diferente ao enviar os dados seriais. A velocidade do piscar depende da taxa de transmissão usada pela placa. RX pisca durante o processo de recebimento.</p>
	<p>E / S digital</p> <p>A placa Arduino UNO possui 14 pinos de E / S digitais (15) (dos quais 6 fornecem saída PWM (Pulse Width Modulation)). Esses pinos podem ser configurados para funcionar como pinos digitais de entrada para ler valores lógicos (0 ou 1) ou como digitais pinos de saída para acionar módulos diferentes, como LEDs, relés, etc. Os pinos rotulados como "~" podem ser usados para gerar PWM.</p>
	<p>AREF</p> <p>AREF significa Referência Analógica. Às vezes, é usado para definir uma tensão de referência externa (entre 0 e 5 Volts) como o limite superior para os pinos de entrada analógicos.</p>

Arduino - Instalação

Depois de aprender sobre as principais partes da placa UNO do Arduino, estamos prontos para aprender como configurar o IDE do Arduino. Depois que aprendermos isso, estaremos prontos para fazer upload de nosso programa no quadro do Arduino.

Nesta seção, aprenderemos em etapas fáceis, como configurar o Arduino IDE em nosso computador e preparar a placa para receber o programa via cabo USB.

Etapa 1 - Primeiro você deve ter sua placa Arduino (você pode escolher sua placa favorita) e um cabo USB. Caso você utilize o Arduino UNO, o Arduino Duemilanove, o Nano, o Arduino Mega 2560 ou o Diecimila, será necessário um cabo USB padrão (plugue A para plugue B), do tipo que você conectaria a uma impressora USB, conforme mostrado na imagem a seguir.

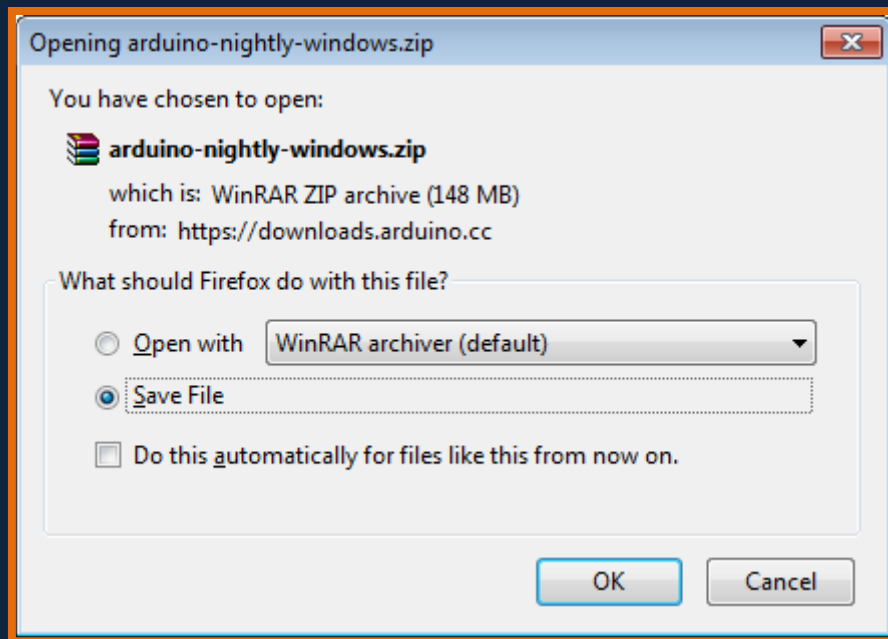


Caso você use o Arduino Nano, precisará de um cabo A para Mini-B, conforme mostrado na imagem a seguir.



Etapa 2 - Faça o download do software IDE do Arduino.

Você pode obter diferentes versões do IDE do Arduino na página Download no site oficial do Arduino. Você deve selecionar seu software, compatível com seu sistema operacional (Windows, IOS ou Linux). Após a conclusão do download do arquivo, descompacte o arquivo.



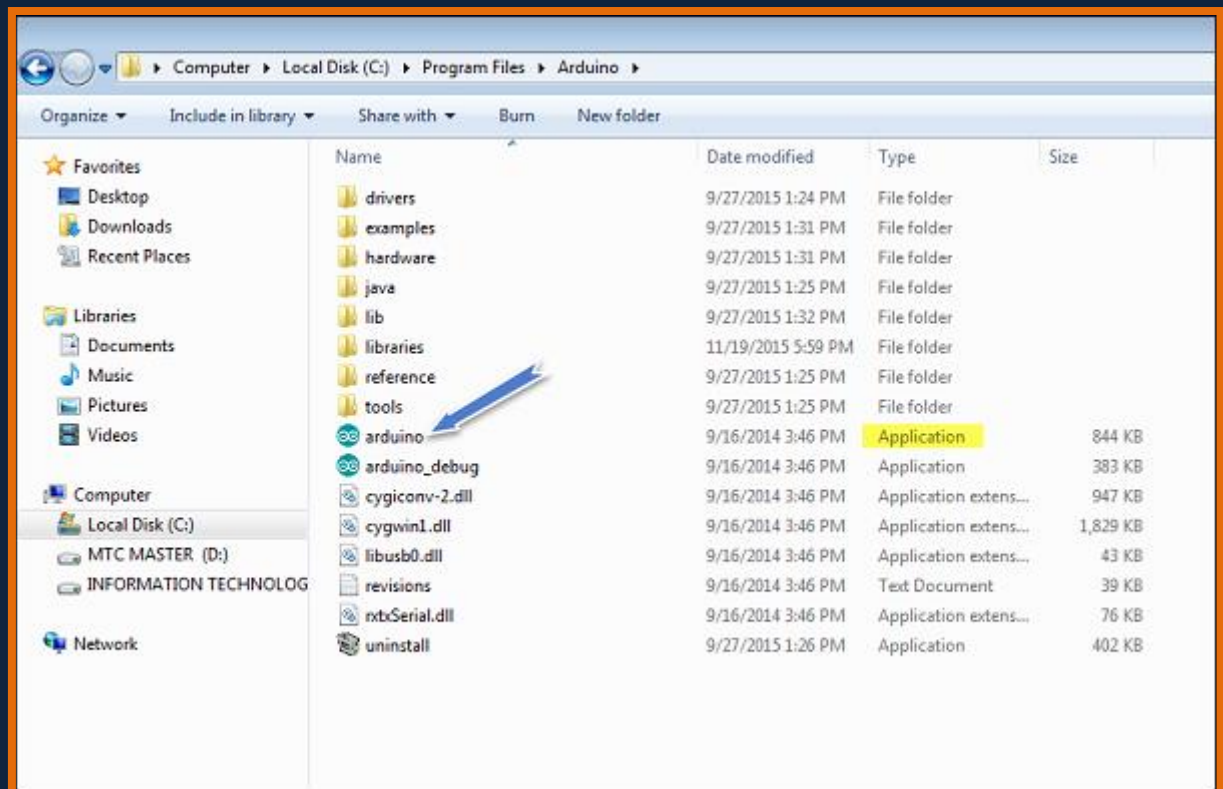
Etapa 3 - Ligue sua placa.

O Arduino Uno, Mega, Duemilanove e Arduino Nano consomem energia automaticamente, da conexão USB ao computador ou de uma fonte de alimentação externa. Se você estiver usando um Arduino Diecimila, verifique se a placa está configurada para extrair energia da conexão USB. A fonte de alimentação é selecionada com um jumper, um pequeno pedaço de plástico que se encaixa em dois dos três pinos entre as tomadas USB e de alimentação. Verifique se ele está nos dois pinos mais próximos da porta USB.

Conecte a placa Arduino ao seu computador usando o cabo USB. O LED verde de energia (rotulado como PWR) deve acender.

Etapa 4 - Inicie o Arduino IDE.

Após o download do software IDE do Arduino, é necessário descompactar a pasta. Dentro da pasta, você pode encontrar o ícone do aplicativo com um rótulo infinito (application.exe). Clique duas vezes no ícone para iniciar o IDE.

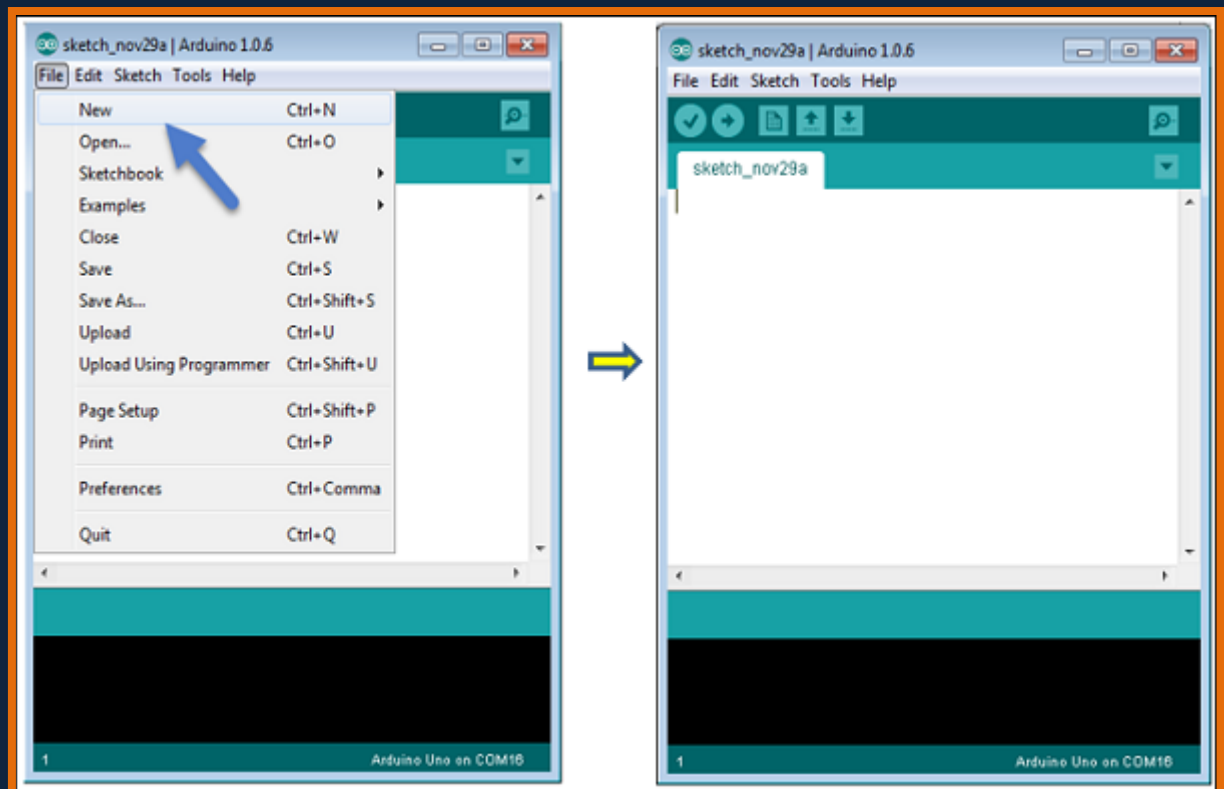


Etapa 5 - Abra seu primeiro projeto.

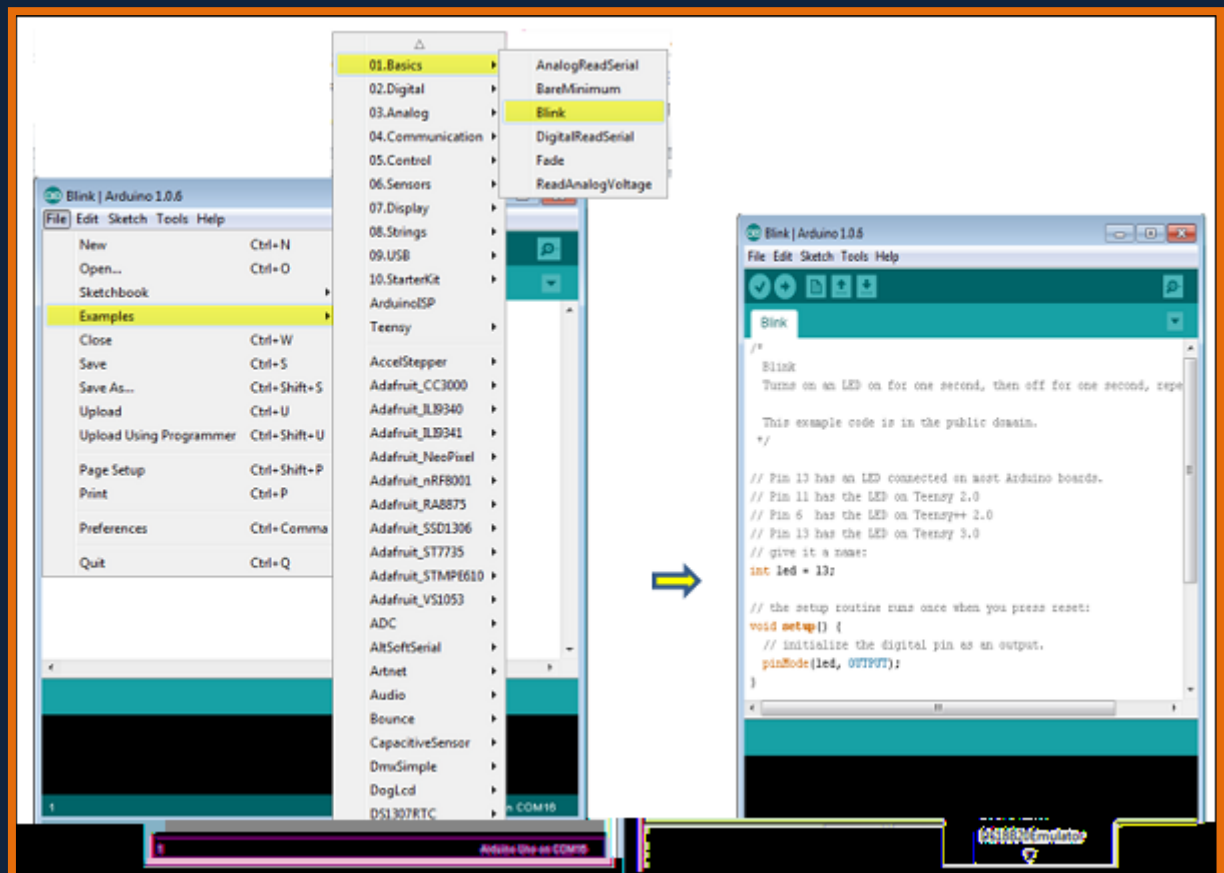
Depois que o software inicia, você tem duas opções -

- Crie um novo projeto.
- Abra um exemplo de projeto existente.

Para criar um novo projeto, selecione Arquivo → **Novo** .



Para abrir um exemplo de projeto existente, selecione Arquivo → Exemplo → Básico → Piscar.

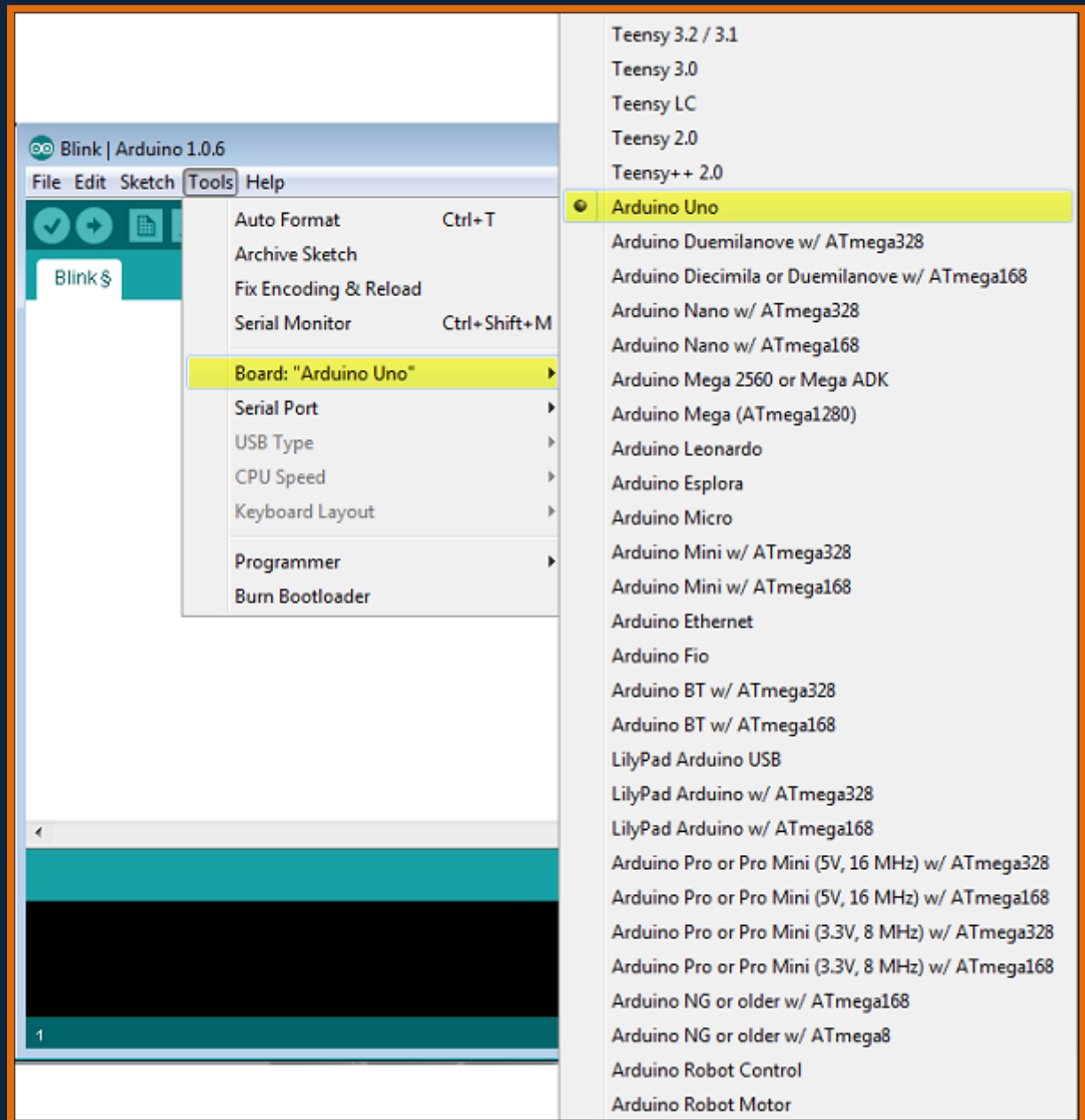


Aqui, estamos selecionando apenas um dos exemplos com o nome **Blink** . Acende e apaga o LED com algum atraso. Você pode selecionar qualquer outro exemplo da lista.

Etapas 6 - Selecione sua placa Arduino.

Para evitar qualquer erro ao carregar o programa na placa, você deve selecionar o nome correto da placa Arduino, que corresponda à placa conectada ao seu computador.

Vá em Ferramentas → Quadro e selecione seu quadro.

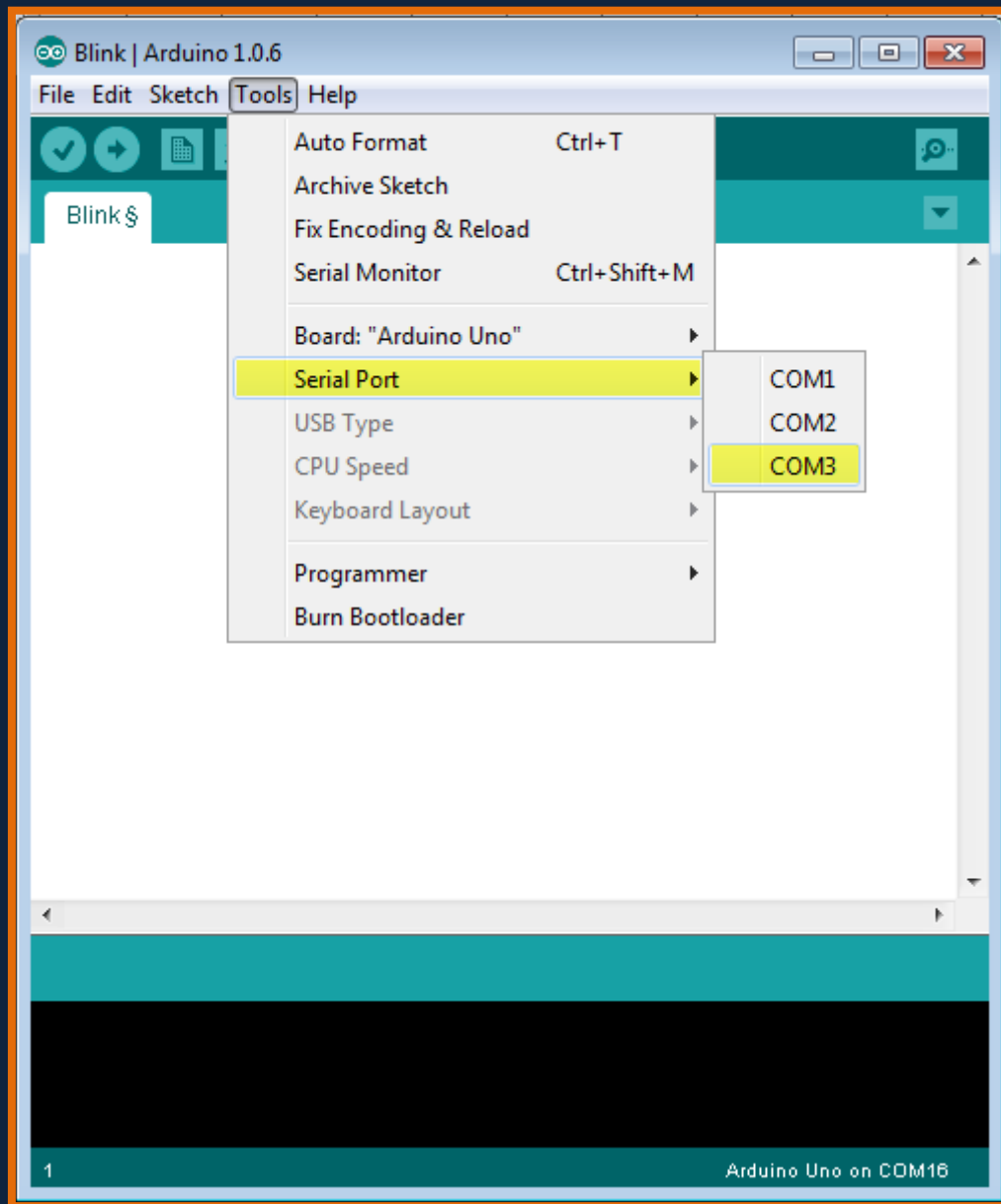


Aqui, selecionamos o painel Arduino Uno de acordo com o nosso tutorial, mas você deve selecionar o nome correspondente ao painel que está usando.

Etapas 7 - Selecione sua porta serial.

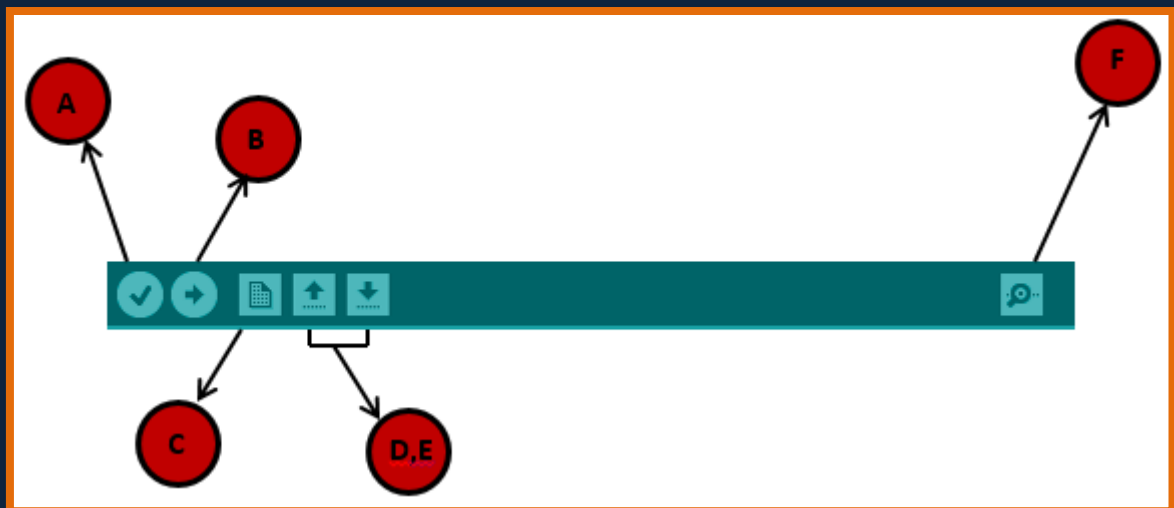
Selecione o dispositivo serial da placa Arduino. Vá para o **menu Ferramentas → Porta serial** . É provável que seja COM3 ou superior (COM1 e COM2 são geralmente

reservados para portas seriais de hardware). Para descobrir, você pode desconectar a placa do Arduino e reabrir o menu; a entrada que desaparecer deve ser da placa do Arduino. Reconecte a placa e selecione essa porta serial.



Etapa 8 - Carregue o programa na sua placa.

Antes de explicar como podemos fazer upload de nosso programa no quadro, devemos demonstrar a função de cada símbolo que aparece na barra de ferramentas do Arduino IDE.



A - Usado para verificar se há algum erro de compilação.

B - Usado para carregar um programa na placa do Arduino.

C - Atalho usado para criar um novo esboço.

D - Usado para abrir diretamente um dos esboços de exemplo.

E - Usado para salvar seu esboço.

F - Monitor serial usado para receber dados seriais da placa e enviar os dados seriais para a placa.

Agora, basta clicar no botão "Upload" no ambiente. Aguarde alguns segundos; você verá os LEDs RX e TX na placa piscando. Se o upload for bem-sucedido, a mensagem "Concluído upload" aparecerá na barra de status.

Nota - Se você possui um Arduino Mini, NG ou outra placa, precisa pressionar fisicamente o botão de reset na placa, imediatamente antes de clicar no botão de upload no software Arduino.

Arduino - Estrutura do Programa

Neste capítulo, estudaremos em profundidade a estrutura do programa Arduino e aprenderemos mais novas terminologias usadas no mundo Arduino. O software Arduino é de código aberto. O código-fonte para o ambiente Java é liberado sob a GPL e as bibliotecas de microcontroladores C / C ++ estão sob a LGPL.

Esboço - A primeira nova terminologia é o programa Arduino chamado " esboço ".

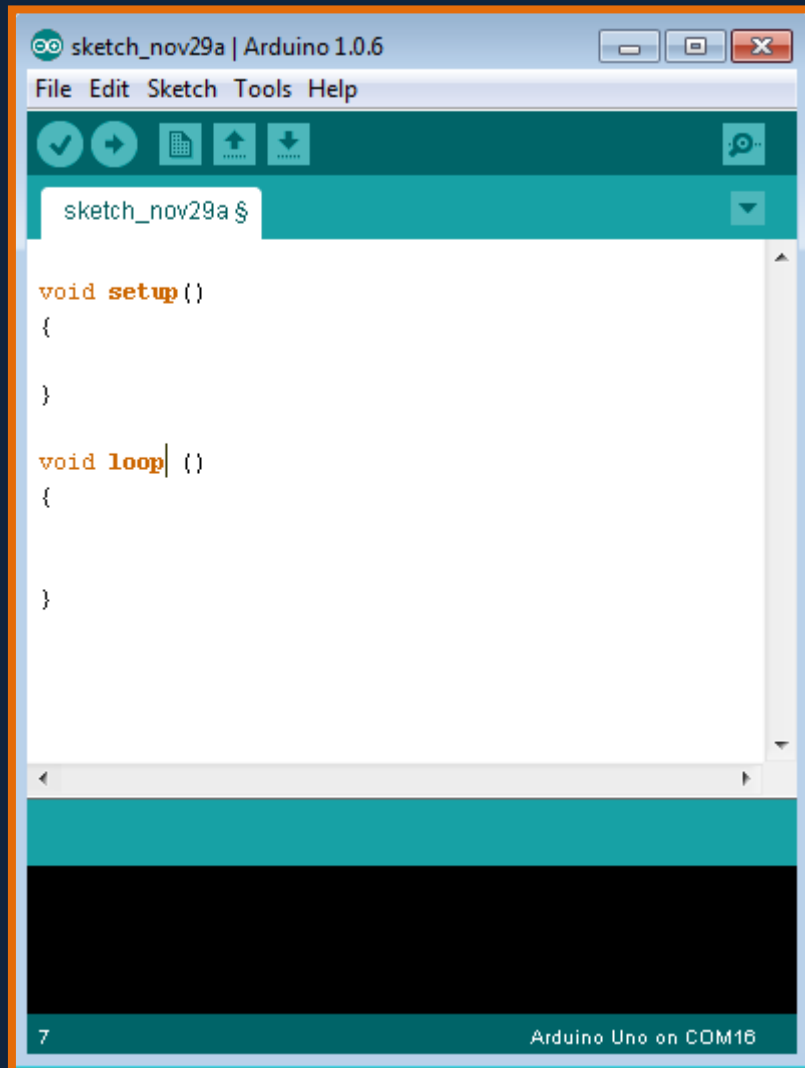
Estrutura

Os programas do Arduino podem ser divididos em três partes principais: **Estrutura**, **Valores** (variáveis e constantes) e **Funções** . Neste tutorial, aprenderemos sobre o

programa de software Arduino, passo a passo, e como podemos escrever o programa sem nenhum erro de sintaxe ou compilação.

Vamos começar com a **estrutura** . A estrutura do software consiste em duas funções principais -

- Função Setup ()
- Função Loop ()



```
Void setup ( ) {  
  
}
```

- **OBJETIVO** - A função **setup ()** é chamada quando um esboço é iniciado. Use-o para inicializar as variáveis, modos de pinos, começar a usar bibliotecas etc. A função de configuração será executada apenas uma vez, após cada inicialização ou redefinição da placa Arduino.
- **ENTRADA** - -
- **SAÍDA** - -
- **RETORNO** - -

```
Void Loop ( ) {  
  
}
```

- **OBJETIVO** - Depois de criar uma função **setup ()** , que inicializa e define os valores iniciais, a função **loop ()** faz exatamente o que o nome sugere e faz um loop consecutivo, permitindo que o programa mude e responda. Use-o para controlar ativamente a placa Arduino.
- **ENTRADA** - -
- **SAÍDA** - -
- **RETORNO** - -

Arduino - Tipos de Dados

Os tipos de dados em C se referem a um sistema extenso usado para declarar variáveis ou funções de tipos diferentes. O tipo de uma variável determina quanto espaço ela ocupa no armazenamento e como o padrão de bits armazenado é interpretado.

A tabela a seguir fornece todos os tipos de dados que você usará durante a programação do Arduino.

vazio	booleano	Caracteres	Caracter não identificado	byte	int	Int não assinado	palavra
longo	Não assinado por muito tempo	baixo	flutuador	Duplo	matriz	Matriz string-char	Objeto String

vazio

A palavra-chave void é usada apenas em declarações de função. Indica que a função deve retornar nenhuma informação para a função da qual foi chamada.

Exemplo

```
Void Loop ( ) {  
  // rest of the code  
}
```

booleano

Um booleano contém um dos dois valores, verdadeiro ou falso. Cada variável booleana ocupa um byte de memória.

Exemplo

```
boolean val = false ; // declaration of variable with type boolean and initialize it with false
boolean state = true ; // declaration of variable with type boolean and initialize it with true
```

Caracteres

Um tipo de dados que ocupa um byte de memória que armazena um valor de caractere. Literais de caracteres são escritos em aspas simples como esta: 'A' e para vários caracteres, as seqüências de caracteres usam aspas duplas: "ABC".

No entanto, os caracteres são armazenados como números. Você pode ver a codificação específica no gráfico ASCII. Isso significa que é possível executar operações aritméticas em caracteres, nos quais o valor ASCII do caractere é usado. Por exemplo, 'A' + 1 tem o valor 66, pois o valor ASCII da letra maiúscula A é 65.

Exemplo

```
Char chr_a = 'a' ; //declaration of variable with type char and initialize it with character a
Char chr_c = 97 ; //declaration of variable with type char and initialize it with character 97
```

Ascii Chart																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPC	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
9	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
10	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
11	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
12	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
13	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
14	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
15	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
16	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

caracter não identificado

Caracter não assinado é um tipo de dados não assinado que ocupa um byte de memória. O tipo de dados char não assinado codifica números de 0 a 255.

Exemplo

Unsigned Char chr_y = 121 ; // declaration of variable with type Unsigned char and initialize it with character y

byte

Um byte armazena um número não assinado de 8 bits, de 0 a 255.

Exemplo

```
byte m = 25 ;//declaration of variable with type byte and initialize it with 25
```

int

Os números inteiros são o tipo de dados principal para armazenamento de números. `int` armazena um valor de 16 bits (2 bytes). Isso gera um intervalo de -32.768 a 32.767 (valor mínimo de -2^{15} e um valor máximo de $(2^{15}) - 1$).

O tamanho `int` varia de placa para placa. No Arduino Due, por exemplo, um `int` armazena um valor de 32 bits (4 bytes). Isso gera um intervalo de -2.147.483.648 a 2.147.483.647 (valor mínimo de -2^{31} e um valor máximo de $(2^{31}) - 1$).

Exemplo

```
int counter = 32 ;// declaration of variable with type int and initialize it with 32
```

Int não assinado

Entradas não assinadas (números inteiros não assinados) são iguais a `int` da maneira que armazenam um valor de 2 bytes. Em vez de armazenar números negativos, no entanto, eles apenas armazenam valores positivos, produzindo um intervalo útil de 0 a 65.535 ($2^{16} - 1$). O Due armazena um valor de 4 bytes (32 bits), variando de 0 a 4.294.967.295 ($2^{32} - 1$).

Exemplo

```
Unsigned int counter = 60 ; // declaration of variable with  
type unsigned int and initialize it with 60
```

Palavra

No Uno e em outras placas baseadas em ATMEGA, uma palavra armazena um número não assinado de 16 bits. No vencimento e no zero, ele armazena um número não assinado de 32 bits.

Exemplo

```
word w = 1000 ;//declaration of variable with type word and initialize it with 1000
```

Longo

Variáveis longas são variáveis de tamanho estendido para armazenamento de números e armazenam 32 bits (4 bytes), de -2.147.483.648 a 2.147.483.647.

Exemplo

```
Long velocity = 102346 ;//declaration of variable with type Long and initialize it with 102346
```

não assinado por muito tempo

Variáveis longas não assinadas são variáveis de tamanho estendido para armazenamento de números e armazenam 32 bits (4 bytes). Diferentemente dos longos padrão, os longos não assinados não armazenam números negativos, variando de 0 a 4.294.967.295 ($2^{32} - 1$).

Exemplo

```
Unsigned Long velocity = 101006 ;// declaration of variable with type Unsigned Long and initialize it with 101006
```

baixo

Um curto é um tipo de dados de 16 bits. Em todos os Arduinos (baseados em ATmega e ARM), um pequeno armazena um valor de 16 bits (2 bytes). Isso gera um intervalo de -32.768 a 32.767 (valor mínimo de -2^{15} e um valor máximo de $(2^{15} - 1)$).

Exemplo

```
short val = 13 ;//declaration of variable with type short and initialize it with 13
```

Flutuador

O tipo de dados para o número de ponto flutuante é um número que possui um ponto decimal. Números de ponto flutuante são frequentemente usados para aproximar os valores analógicos e contínuos porque eles têm maior resolução que números inteiros.

Os números de ponto flutuante podem ser tão grandes quanto $3,4028235E + 38$ e tão baixos quanto $-3,4028235E + 38$. Eles são armazenados como 32 bits (4 bytes) de informação.

Exemplo

```
float num = 1.352; //declaration of variable with type float and initialize it with 1.352
```

Duplo

No Uno e em outras placas baseadas em ATMEGA, o número de ponto flutuante de precisão dupla ocupa quatro bytes. Ou seja, a implementação dupla é exatamente igual à flutuação, sem ganho de precisão. No Arduino Due, os duplos têm precisão de 8 bytes (64 bits).

Exemplo

```
double num = 45.352 ;// declaration of variable with type double and initialize it wit
```

Arduino - Variáveis e Constantes

Antes de começarmos a explicar os tipos de variáveis, um assunto muito importante que precisamos ter certeza de que você entende completamente é chamado de **escopo de variável** .

O que é escopo variável?

Variáveis na linguagem de programação C, que o Arduino usa, possuem uma propriedade chamada scope. Um escopo é uma região do programa e há três locais onde variáveis podem ser declaradas. Eles são -

- Dentro de uma função ou bloco, que é chamado de **variáveis locais** .
- Na definição de parâmetros de função, que é chamada de **parâmetros formais** .
- Fora de todas as funções, que são chamadas de **variáveis globais** .

Variáveis locais

Variáveis declaradas dentro de uma função ou bloco são variáveis locais. Eles podem ser usados apenas pelas instruções que estão dentro dessa função ou bloco de código. As variáveis locais não são conhecidas por funcionar fora de suas próprias. A seguir, o exemplo usando variáveis locais -

```
Void setup () {  
  
}  
  
Void loop () {  
  int x , y ;  
  int z ; Local variable declaration  
  x = 0;  
  y = 0; actual initialization  
  z = 10;  
}
```


Variáveis globais

Variáveis globais são definidas fora de todas as funções, geralmente na parte superior do programa. As variáveis globais manterão seu valor durante toda a vida útil do seu programa.

Uma variável global pode ser acessada por qualquer função. Ou seja, uma variável global está disponível para uso em todo o programa após a declaração.

O exemplo a seguir usa variáveis globais e locais -

```
Int T , S ;  
float c = 0 ; Global variable declaration  
  
Void setup () {  
  
}  
  
Void loop () {  
    int x , y ;  
    int z ; Local variable declaration  
    x = 0;  
    y = 0; actual initialization  
    z = 10;  
}
```

Arduino - Operadores

Um operador é um símbolo que informa ao compilador para executar funções matemáticas ou lógicas específicas. A linguagem C é rica em operadores internos e fornece os seguintes tipos de operadores -

- Operadores aritméticos
- Operadores de comparação
- Operadores booleanos
- Operadores bit a bit
- Operadores compostos

Operadores aritméticos

Assuma que a variável A possui 10 e a variável B possui 20, em seguida -

Mostrar Exemplo

Nome do	Operador	Descrição	Exemplo
---------	----------	-----------	---------

operador	simples		
operador de atribuição	=	Armazena o valor à direita do sinal de igual na variável à esquerda do sinal de igual.	A = B
Adição	+	Adiciona dois operandos	A + B dará 30
subtração	-	Subtrai o segundo operando do primeiro	A - B dará -10
multiplicação	*	Multiplique os dois operandos	A * B dará 200
divisão	/	Divida o numerador pelo denominador	B / A dará 2
módulo	%	Operador de módulo e restante após uma divisão inteira	B % A dará 0

Operadores de comparação

Assuma que a variável A possui 10 e a variável B possui 20, em seguida -

Mostrar Exemplo

Nome do operador	Operador simples	Descrição	Exemplo
igual a	==	Verifica se o valor de dois operandos é igual ou não, se sim, a condição se torna verdadeira.	(A == B) não é verdadeiro
não é igual a	!=	Verifica se o valor de dois operandos é igual ou não, se os valores não são iguais,	(A != B) é

		a condição se torna verdadeira.	verdadeiro
Menor que	<	Verifica se o valor do operando esquerdo é menor que o valor do operando direito; se sim, a condição se torna verdadeira.	(A < B) é verdadeiro
Maior que	>	Verifica se o valor do operando esquerdo é maior que o valor do operando direito; se sim, a condição se torna verdadeira.	(A > B) não é verdadeiro
menos que ou igual a	<=	Verifica se o valor do operando esquerdo é menor ou igual ao valor do operando direito; se sim, a condição se torna verdadeira.	(A <= B) é verdadeiro
Melhor que ou igual a	>=	Verifica se o valor do operando esquerdo é maior ou igual ao valor do operando direito; se sim, a condição se torna verdadeira.	(A >= B) não é verdadeiro

Operadores booleanos

Assuma que a variável A possui 10 e a variável B possui 20, em seguida -

Mostrar Exemplo

Nome do operador	Operador simples	Descrição	Exemplo
e	&&	Chamado operador AND lógico. Se os dois operandos forem diferentes de zero, a condição se tornará verdadeira.	(A && B) é verdadeiro
ou		Chamado operador lógico ou. Se qualquer um dos dois operandos for diferente de zero, a condição se tornará verdadeira.	(A B) é verdadeiro

não	!	Chamado operador NÃO lógico. Use para reverter o estado lógico de seu operando. Se uma condição for verdadeira, o operador Logical NOT se tornará falso.	! (A && B) é falso
-----	---	--	--------------------

Operadores bit a bit

Assuma que a variável A possui 60 e a variável B possui 13 e então -

Mostrar Exemplo

Nome do operador	Operador simples	Descrição	Exemplo
e	&	O operador AND binário copia um pouco para o resultado, se existir nos dois operandos.	(A & B) dará 12, que é 0000 1100
ou		Operador OR binário copia um pouco se ele existir em qualquer operando	(A B) dará 61, que é 0011 1101
xor	^	O operador binário XOR copia o bit se ele estiver definido em um operando, mas não em ambos.	(A ^ B) dará 49, que é 0011 0001
não	~	O operador de complemento binário é unário e tem o efeito de 'inverter' os bits.	(~ A) dará - 60, que é 1100 0011
deslocar para a esquerda	<<	Operador de deslocamento à esquerda binário. O valor dos operandos da esquerda é movido para a esquerda pelo número de bits especificado pelo operando da direita.	Um << 2 dará 240, que é 1111 0000
mudar para a direita	>>	Operador de deslocamento à direita binário. O valor dos operandos da esquerda é movido para a direita pelo número de bits	A >> 2 dará 15, que é 0000

		especificado pelo operando da direita.	1111
--	--	--	------

Operadores compostos

Assuma que a variável A possui 10 e a variável B possui 20, em seguida -

Mostrar Exemplo

Nome do operador	Operador simples	Descrição	Exemplo
incremento	++	Operador de incremento, aumenta o valor inteiro em um	A ++ dará 11
diminuir	-	Operador de decremento, diminui o valor inteiro em um	A-- dará 9
adição de compostos	+=	Adicionar E operador de atribuição. Ele adiciona o operando direito ao operando esquerdo e atribui o resultado ao operando esquerdo	B += A é equivalente a B = B + A
subtração composta	-=	Subtrair E operador de atribuição. Subtrai o operando direito do operando esquerdo e atribui o resultado ao operando esquerdo	B -= A é equivalente a B = B - A
multiplicação composta	*=	Operador de multiplicação e atribuição. Multiplica o operando direito pelo operando esquerdo e atribui o resultado ao operando esquerdo	B *= A é equivalente a B = B * A
divisão composta	/=	Operador de divisão E atribuição. Ele divide o operando esquerdo com o operando direito e atribui o resultado ao operando esquerdo	B /= A é equivalente a B = B / A

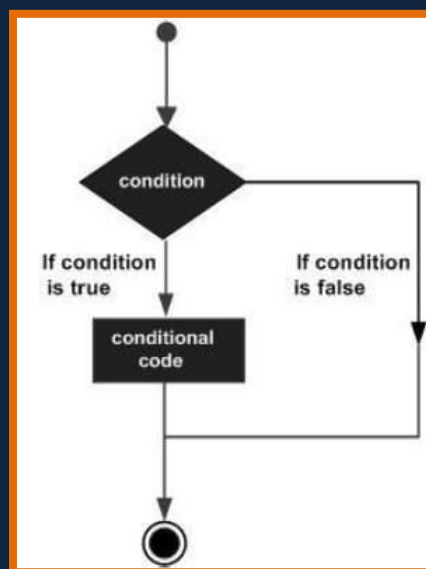
módulo composto	$\% =$	Módulo E operador de atribuição. Ele pega o módulo usando dois operandos e atribui o resultado ao operando esquerdo	$B\% = A$ é equivalente a $B = B\% A$
composto bit a bit ou	$ =$	operador OR inclusivo e de atribuição bit a bit	$A = 2$ é o mesmo que $A = A 2$
composto bit a bit e	$\& =$	Operador de atribuição AND bit a bit	$A \& = 2$ é o mesmo que $A = A \& 2$

Arrisque-se! “Para ser grande, às vezes é necessário correr riscos enormes.”
Bill Gates

Arduino - Declarações de controle

As estruturas de tomada de decisão exigem que o programador especifique uma ou mais condições a serem avaliadas ou testadas pelo programa. Deveria estar junto com uma instrução ou instruções a serem executadas se a condição for determinada como verdadeira e, opcionalmente, outras instruções a serem executadas se a condição for determinada como falsa.

A seguir está a forma geral de uma estrutura típica de tomada de decisão encontrada na maioria das linguagens de programação -



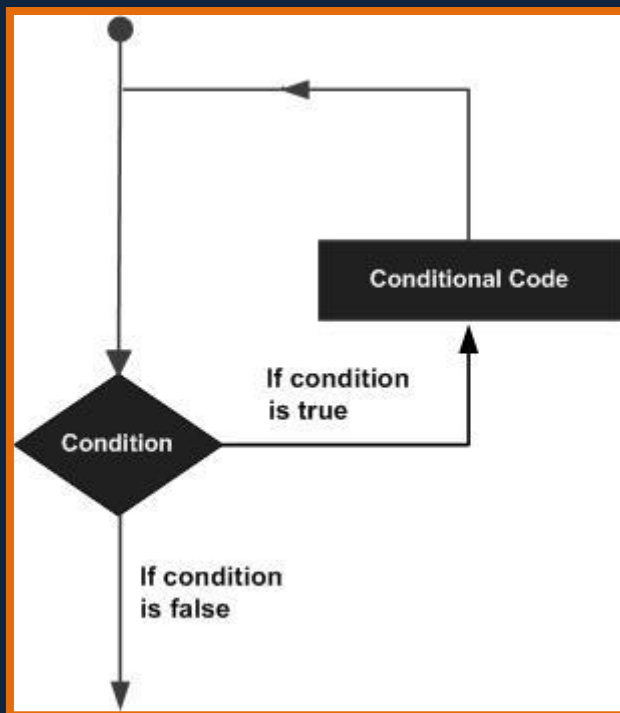
As declarações de controle são elementos no código-fonte que controlam o fluxo de execução do programa. Eles são -

S.NO.	Declaração e descrição de controle
1	<p>Se declaração</p> <p>É preciso uma expressão entre parênteses e uma declaração ou bloco de declarações. Se a expressão for verdadeira, a instrução ou bloco de instruções será executado, caso contrário, essas instruções serão ignoradas.</p>
2	<p>Declaração if... else</p> <p>Uma instrução if pode ser seguida por uma instrução else opcional, que é executada quando a expressão é falsa.</p>
3	<p>Declaração if... else if... else</p> <p>A instrução if pode ser seguida por uma instrução opcional else if ... else, que é muito útil para testar várias condições usando a instrução single if ... else if.</p>
4	<p>declaração de caso de mudança</p> <p>Semelhante às instruções if, switch ... case controla o fluxo de programas, permitindo que os programadores especifiquem códigos diferentes que devem ser executados em várias condições.</p>
5	<p>Operador condicional? :</p> <p>O operador condicional? : é o único operador ternário em C.</p>

Arduino - Loops

As linguagens de programação fornecem várias estruturas de controle que permitem caminhos de execução mais complicados.

Uma declaração de loop nos permite executar uma declaração ou grupo de declarações várias vezes e a seguir é a forma geral de uma declaração de loop na maioria das linguagens de programação -



A linguagem de programação C fornece os seguintes tipos de loops para lidar com os requisitos de loop.

S.NO.	Loop & Descrição
1	<p>enquanto loop</p> <p>enquanto os loops se repetem continuamente e infinitamente, até que a expressão entre parênteses, () se torne falsa. Algo deve mudar a variável testada, ou o loop while nunca sairá.</p>
2	<p>fazer ... enquanto loop</p> <p>O loop do... while é semelhante ao loop while. No loop while, a condição de continuação do loop é testada no início do loop antes de executar o corpo do loop.</p>
3	<p>para laço</p> <p>Um loop for executa instruções um número predeterminado de vezes. A expressão de controle para o loop é inicializada, testada e manipulada inteiramente dentro dos parênteses do loop for.</p>
4	<p>Loop aninhado</p> <p>A linguagem C permite que você use um loop dentro de outro loop. O exemplo a seguir ilustra o conceito.</p>

5	<p>Loop infinito</p> <p>Como o loop não tem condição de término, o loop se torna infinito.</p>
---	--

Arduino - Funções

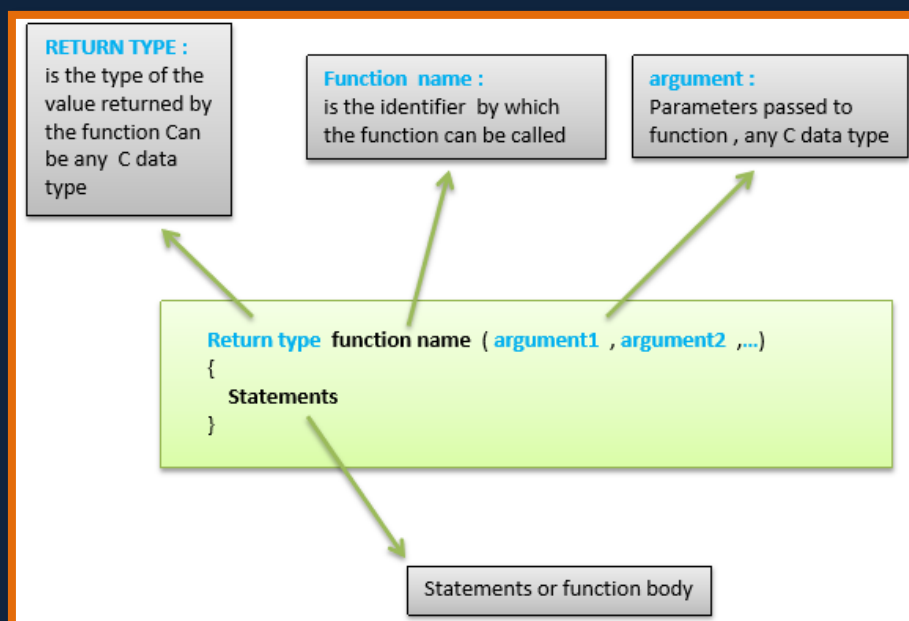
As funções permitem estruturar os programas em segmentos de código para executar tarefas individuais. O caso típico para criar uma função é quando é necessário executar a mesma ação várias vezes em um programa.

A padronização de fragmentos de código em funções tem várias vantagens -

- As funções ajudam o programador a se organizar. Muitas vezes, isso ajuda a conceituar o programa.
- As funções codificam uma ação em um só lugar, para que a função precise ser pensada e depurada apenas uma vez.
- Isso também reduz as chances de erros na modificação, se o código precisar ser alterado.
- As funções tornam o esboço inteiro menor e mais compacto porque as seções do código são reutilizadas várias vezes.
- Eles facilitam a reutilização de código em outros programas, tornando-o modular, e o uso de funções geralmente torna o código mais legível.

Existem duas funções necessárias em um esboço ou programa do Arduino, ou seja, `setup ()` e `loop ()`. Outras funções devem ser criadas fora dos colchetes dessas duas funções.

A sintaxe mais comum para definir uma função é -



Declaração de Função

Uma função é declarada fora de qualquer outra função, acima ou abaixo da função de loop.

Podemos declarar a função de duas maneiras diferentes -

A primeira maneira é apenas escrever a parte da função chamada **protótipo de função** acima da função de loop, que consiste em -

- Tipo de retorno de função
- Nome da função
- Tipo de argumento de função, sem necessidade de escrever o nome do argumento

O protótipo de função deve ser seguido por um ponto e vírgula (;).

O exemplo a seguir mostra a demonstração da declaração da função usando o primeiro método

Exemplo

```
int sum_func (int x, int y) // function declaration {  
    int z = 0;  
    z = x+y ;  
    return z; // return the value  
}  
  
void setup () {  
    Statements // group of statements  
}  
  
Void loop () {  
    int result = 0 ;  
    result = Sum_func (5,6) ; // function call  
}
```

A segunda parte, chamada de definição ou declaração de função, deve ser declarada abaixo da função de loop, que consiste em -

- Tipo de retorno de função
- Nome da função
- Tipo de argumento de função, aqui você deve adicionar o nome do argumento
- O corpo da função (instruções dentro da função em execução quando a função é chamada)

O exemplo a seguir demonstra a declaração de função usando o segundo método.

Exemplo

```
int sum_func (int , int ) ; // function prototype
```

```

void setup () {
  Statements // group of statements
}

Void loop () {
  int result = 0 ;
  result = Sum_func (5,6) ; // function call
}

int sum_func (int x, int y) // function declaration {
  int z = 0;
  z = x+y ;
  return z; // return the value
}

```

O segundo método apenas declara a função acima da função de loop.

Arduino - String

Strings são usadas para armazenar texto. Eles podem ser usados para exibir texto em um LCD ou na janela Monitor serial do Arduino IDE. Strings também são úteis para armazenar a entrada do usuário. Por exemplo, os caracteres que um usuário digita em um teclado conectado ao Arduino.

Existem dois tipos de strings na programação do Arduino -

- Matrizes de caracteres, que são as mesmas que as strings usadas na programação C.
- A String do Arduino, que nos permite usar um objeto de string em um esboço.

Neste capítulo, aprenderemos Strings, objetos e o uso de strings nos esboços do Arduino. No final do capítulo, você aprenderá que tipo de string usar em um esboço.

Matrizes de caracteres de sequência

O primeiro tipo de string que aprenderemos é a string que é uma série de caracteres do tipo **char**. No capítulo anterior, aprendemos o que é uma matriz; uma série consecutiva do mesmo tipo de variável armazenada na memória. Uma string é uma matriz de variáveis de caracteres.

Uma string é uma matriz especial que possui um elemento extra no final da string, que sempre tem o valor 0 (zero). Isso é conhecido como "sequência terminada nula".

Exemplo de matriz de caracteres de sequência

Este exemplo mostra como criar uma string e imprimi-la na janela do monitor serial.

Exemplo

```

void setup() {
  char my_str[6]; // an array big enough for a 5 character string
  Serial.begin(9600);
  my_str[0] = 'H'; // the string consists of 5 characters
  my_str[1] = 'e';
  my_str[2] = 'l';
  my_str[3] = 'l';
  my_str[4] = 'o';
  my_str[5] = 0; // 6th array element is a null terminator
  Serial.println(my_str);
}

void loop() {
}

```

O exemplo a seguir mostra do que uma string é composta; uma matriz de caracteres com caracteres imprimíveis e 0 como o último elemento da matriz para mostrar que é aí que a cadeia termina. A sequência pode ser impressa na janela Monitor serial do Arduino IDE usando **Serial.println ()** e passando o nome da sequência.

Este mesmo exemplo pode ser escrito de uma maneira mais conveniente, como mostrado abaixo -

Exemplo

```

void setup() {
  char my_str[] = "Hello";
  Serial.begin(9600);
  Serial.println(my_str);
}

void loop() {
}

```

Neste esboço, o compilador calcula o tamanho da matriz de cadeias e também nula automaticamente termina a cadeia com um zero. Uma matriz com seis elementos e composta por cinco caracteres seguidos por um zero é criada exatamente da mesma maneira que no esboço anterior.

Manipulando matrizes de sequência de caracteres

Podemos alterar uma matriz de cadeias de caracteres dentro de um esboço, conforme mostrado no esboço a seguir.

Exemplo

```

void setup() {
  char like[] = "I like coffee and cake"; // create a string
  Serial.begin(9600);
  // (1) print the string
}

```

```
Serial.println(like);
// (2) delete part of the string
like[13] = 0;
Serial.println(like);
// (3) substitute a word into the string
like[13] = ' '; // replace the null terminator with a space
like[18] = 't'; // insert the new word
like[19] = 'e';
like[20] = 'a';
like[21] = 0; // terminate the string
Serial.println(like);
}

void loop() {
}
```

Resultado

I like coffee and cake

I like coffee

I like coffee and tea

O esboço funciona da seguinte maneira.

Criando e imprimindo a string

No esboço fornecido acima, uma nova sequência é criada e impressa para exibição na janela Serial Monitor.

Encurtando a String

A cadeia de caracteres é reduzida, substituindo o 14º caractere por um zero que termina nulo (2). Este é o elemento número 13 na matriz de cadeias, contando de 0.

Quando a sequência é impressa, todos os caracteres são impressos até o novo zero final nulo. Os outros caracteres não desaparecem; eles ainda existem na memória e a matriz de cadeias ainda tem o mesmo tamanho. A única diferença é que qualquer função que funcione com cadeias verá apenas a cadeia até o primeiro terminador nulo.

Alterando uma palavra na string

Por fim, o esboço substitui a palavra "bolo" por "chá" (3). Primeiro, ele deve substituir o terminador nulo em [13] por um espaço para que a string seja restaurada para o formato criado originalmente.

Novos caracteres substituem "cak" da palavra "bolo" pela palavra "chá". Isso é feito sobrescrevendo caracteres individuais. O 'e' do "bolo" é substituído por um novo caractere final nulo. O resultado é que a string é realmente finalizada com dois caracteres nulos, o original no final da string e o novo que substitui o 'e' no "bolo". Isso não faz diferença quando a nova string é impressa, porque a função que imprime a

string para de imprimir os caracteres da string quando encontra o primeiro terminador nulo.

Funções para manipular matrizes de sequência de caracteres

O esboço anterior manipulou a sequência de maneira manual, acessando caracteres individuais na sequência. Para facilitar a manipulação de matrizes de sequência de caracteres, você pode escrever suas próprias funções para fazer isso ou usar algumas das funções de sequência de caracteres da biblioteca de idiomas C.

Dada a seguir, é apresentada a lista Funções para manipular matrizes de string

O próximo esboço usa algumas funções da string C.

Exemplo

```
void setup() {
  char str[] = "This is my string"; // create a string
  char out_str[40]; // output from string functions placed here
  int num; // general purpose integer
  Serial.begin(9600);

  // (1) print the string
  Serial.println(str);

  // (2) get the length of the string (excludes null terminator)
  num = strlen(str);
  Serial.print("String length is: ");
  Serial.println(num);

  // (3) get the length of the array (includes null terminator)
  num = sizeof(str); // sizeof() is not a C string function
  Serial.print("Size of the array: ");
  Serial.println(num);

  // (4) copy a string
  strcpy(out_str, str);
  Serial.println(out_str);

  // (5) add a string to the end of a string (append)
  strcat(out_str, " sketch.");
  Serial.println(out_str);
  num = strlen(out_str);
  Serial.print("String length is: ");
  Serial.println(num);
  num = sizeof(out_str);
  Serial.print("Size of the array out_str[]: ");
  Serial.println(num);
}
```

```
void loop() {  
  
}
```

Resultado

This is my string
String length is: 17
Size of the array: 18
This is my string
This is my string sketch.
String length is: 25
Size of the array out_str[]: 40

O esboço funciona da seguinte maneira.

Imprimir a sequência

A sequência recém-criada é impressa na janela Serial Monitor, como feito nos esboços anteriores.

Obter o comprimento da string

A função `strlen ()` é usada para obter o comprimento da string. O comprimento da sequência é apenas para caracteres imprimíveis e não inclui o terminador nulo.

A sequência contém 17 caracteres, portanto vemos 17 impressos na janela Serial Monitor.

Obter o comprimento da matriz

O operador `sizeof ()` é usado para obter o comprimento da matriz que contém a string. O comprimento inclui o terminador nulo; portanto, o comprimento é um a mais que o comprimento da cadeia.

`sizeof ()` parece uma função, mas tecnicamente é um operador. Não faz parte da biblioteca de cadeias C, mas foi usada no esboço para mostrar a diferença entre o tamanho da matriz e o tamanho da cadeia (ou comprimento da cadeia).

Copiar uma String

A função `strcpy ()` é usada para copiar a string `str []` para a matriz `out_num []`. A função `strcpy ()` copia a segunda string passada para ela na primeira string. Uma cópia da string agora existe na matriz `out_num []`, mas ocupa apenas 18 elementos da matriz, portanto ainda temos 22 elementos `char` gratuitos na matriz. Esses elementos livres são encontrados após a sequência na memória.

A string foi copiada para a matriz, para que tivéssemos algum espaço extra na matriz para usar na próxima parte do esboço, que está adicionando uma string ao final de uma string.

Anexar uma String a uma String (Concatenate)

O esboço une uma sequência a outra, conhecida como concatenação. Isso é feito usando a função `strcat()`. A função `strcat()` coloca a segunda string passada no final da primeira string passada a ele.

Após a concatenação, o comprimento da string é impresso para mostrar o novo comprimento da string. O comprimento da matriz é impresso para mostrar que temos uma cadeia de 25 caracteres em uma matriz de 40 elementos.

Lembre-se de que a cadeia longa de 25 caracteres ocupa 26 caracteres da matriz devido ao zero que termina nulo.

Limites da matriz

Ao trabalhar com strings e matrizes, é muito importante trabalhar dentro dos limites de strings ou matrizes. No esboço de exemplo, foi criada uma matriz, com 40 caracteres, para alocar a memória que poderia ser usada para manipular seqüências de caracteres.

Se a matriz fosse muito pequena e tentássemos copiar uma sequência maior que a matriz, ela seria copiada no final da matriz. A memória além do final da matriz pode conter outros dados importantes usados no esboço, que serão substituídos pela nossa string. Se a memória além do final da sequência for excedida, poderá travar o esboço ou causar comportamento inesperado.

Arduino - Objeto String

O segundo tipo de string usado na programação do Arduino é o String Object.

O que é um objeto?

Um objeto é uma construção que contém dados e funções. Um objeto String pode ser criado como uma variável e atribuído um valor ou string. O objeto String contém funções (chamadas "métodos" na programação orientada a objetos (OOP)) que operam nos dados da string contidos no objeto String.

O seguinte esboço e explicação tornarão claro o que é um objeto e como o objeto String é usado.

Exemplo

```
void setup() {  
  String my_str = "This is my string."  
  Serial.begin(9600);  
  
  // (1) print the string  
  Serial.println(my_str);  
}
```



```

// (2) change the string to upper-case
my_str.toUpperCase();
Serial.println(my_str);

// (3) overwrite the string
my_str = "My new string.";
Serial.println(my_str);

// (4) replace a word in the string
my_str.replace("string", "Arduino sketch");
Serial.println(my_str);

// (5) get the length of the string
Serial.print("String length is: ");
Serial.println(my_str.length());
}

void loop() {
}

```

Resultado

This is my string.
 THIS IS MY STRING.
 My new string.
 My new Arduino sketch.
 String length is: 22

Um objeto de sequência é criado e recebe um valor (ou sequência) na parte superior do esboço.

```
String my_str = "This is my string." ;
```

Isso cria um objeto String com o nome **my_str** e atribui a ele o valor "Esta é minha string".

Isso pode ser comparado à criação de uma variável e à atribuição de um valor, como um número inteiro -

```
int my_var = 102;
```

O esboço funciona da seguinte maneira.

Imprimindo a sequência

A sequência pode ser impressa na janela Serial Monitor, como uma sequência de caracteres.

Converter a sequência de caracteres em maiúsculas

O objeto de cadeia `my_str` que foi criado, possui várias funções ou métodos que podem ser operados nele. Esses métodos são chamados usando o nome dos objetos seguido pelo operador de ponto (`.`) E, em seguida, o nome da função a ser usada.

```
my_str.toUpperCase();
```

A função **`toUpperCase()`** opera na string contida no objeto **`my_str`**, que é do tipo `String` e converte os dados da string (ou texto) que o objeto contém em caracteres maiúsculos. Uma lista das funções que a classe `String` contém pode ser encontrada na referência de `String` do Arduino. Tecnicamente, `String` é chamado de classe e é usado para criar objetos `String`.

Sobrescrever uma String

O operador de atribuição é usado para atribuir uma nova sequência ao objeto **`my_str`** que substitui a sequência antiga

```
my_str = "My new string." ;
```

O operador de atribuição não pode ser usado em cadeias de caracteres, mas funciona apenas em objetos `String`.

Substituindo uma palavra na cadeia

A função `replace()` é usada para substituir a primeira string passada a ela pela segunda string passada a ela. `replace()` é outra função incorporada à classe `String` e, portanto, está disponível para uso no objeto `String my_str`.

Obtendo o comprimento da string

É fácil obter o comprimento da string usando `length()`. No esboço de exemplo, o resultado retornado por `length()` é passado diretamente para `Serial.println()` sem usar uma variável intermediária.

Quando usar um objeto String

Um objeto `String` é muito mais fácil de usar do que uma matriz de caracteres `string`. O objeto possui funções internas que podem executar várias operações em sequências de caracteres.

A principal desvantagem do uso do objeto `String` é que ele usa muita memória e pode usar rapidamente a memória RAM do Arduino, o que pode causar o Arduino travar, travar ou se comportar inesperadamente. Se um esboço em um Arduino é pequeno e limita o uso de objetos, não deve haver problemas.

As sequências de caracteres da matriz são mais difíceis de usar e pode ser necessário escrever suas próprias funções para operar com esses tipos de sequências. A vantagem é que você tem controle sobre o tamanho das matrizes de sequência de caracteres que você cria, para manter as matrizes pequenas para economizar memória.

Você precisa se certificar de que não escreve além do final dos limites da matriz com matrizes de string. O objeto String não possui esse problema e cuidará dos limites da string para você, desde que haja memória suficiente para operar. O objeto String pode tentar gravar na memória que não existe quando fica sem memória, mas nunca gravará no final da string em que está operando.

Onde as cordas são usadas

Neste capítulo, estudamos as seqüências de caracteres, como elas se comportam na memória e em suas operações.

Os usos práticos de strings serão abordados na próxima parte deste curso, quando estudarmos como obter a entrada do usuário na janela Serial Monitor e salvar a entrada em uma string.

Arduino - Tempo

O Arduino fornece quatro funções diferentes de manipulação do tempo. Eles são -

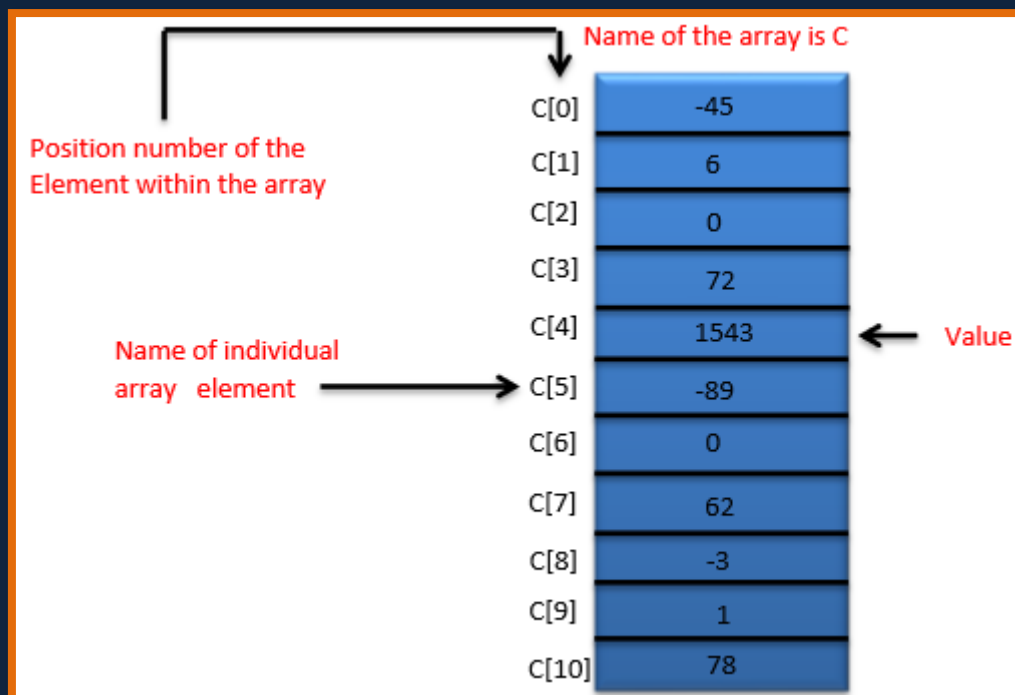
S.No.	Descrição da função
1	função delay () O funcionamento da função delay () é bastante simples. Ele aceita um único argumento inteiro (ou número). Este número representa o tempo (medido em milissegundos).
2	Função delayMicroseconds () A função delayMicroseconds () aceita um único argumento inteiro (ou número). Existem mil microssegundos em um milissegundo e um milhão de microssegundos em um segundo.
3	função millis () Esta função é usada para retornar o número de milissegundos no momento em que a placa Arduino começa a executar o programa atual.
4	função micros () A função micros () retorna o número de microssegundos a partir do momento em que a placa Arduino começa a executar o programa atual. Esse número ultrapassa o valor, ou seja, volta ao zero após aproximadamente 70 minutos.

Arduino - Matrizes

Uma matriz é um grupo consecutivo de locais de memória que são do mesmo tipo. Para se referir a um local ou elemento específico na matriz, especificamos o nome da matriz e o número da posição do elemento específico na matriz.

A ilustração abaixo mostra uma matriz inteira chamada C que contém 11 elementos. Você se refere a qualquer um desses elementos, fornecendo o nome da matriz seguido pelo número da posição do elemento em colchetes ([]). O número da posição é formalmente chamado de índice subscrito ou índice (esse número especifica o número de elementos desde o início da matriz). O primeiro elemento possui o índice 0 (zero) e às vezes é chamado de elemento zeros.

Assim, os elementos da matriz C são C [0] (pronunciado “C sub zero”), C [1], C [2] e assim por diante. O maior índice subscrito na matriz C é 10, que é 1 menor que o número de elementos na matriz (11). Os nomes de matrizes seguem as mesmas convenções que outros nomes de variáveis.



Um subscrito deve ser um número inteiro ou expressão inteira (usando qualquer tipo integral). Se um programa usar uma expressão como um subscrito, o programa avaliará a expressão para determinar o subscrito. Por exemplo, se assumirmos que a variável a é igual a 5 e que a variável b é igual a 6, a instrução adiciona 2 ao elemento C da matriz [11].

Um nome de matriz subscrito é um lvalue, pode ser usado no lado esquerdo de uma atribuição, assim como nomes de variáveis que não são de matriz.

Vamos examinar a matriz C na figura fornecida, mais de perto. O nome de toda a matriz é C. Seus 11 elementos são referidos como C [0] a C [10]. O valor de C [0] é -45, o valor de C [1] é 6, o valor de C [2] é 0, o valor de C [7] é 62 e o valor de C [10] é 78

Para imprimir a soma dos valores contidos nos três primeiros elementos da matriz C, escreveríamos:

```
Serial.print (C[ 0 ] + C[ 1 ] + C[ 2 ] );
```

Para dividir o valor de C [6] por 2 e atribuir o resultado à variável x, escreveríamos:

```
x = C[ 6 ] / 2;
```

Declarando matrizes

Matrizes ocupam espaço na memória. Para especificar o tipo dos elementos e o número de elementos exigidos por uma matriz, use uma declaração do formulário -

```
type arrayName [ arraySize ] ;
```

O compilador reserva a quantidade apropriada de memória. (Lembre-se de que uma declaração, que reserva memória, é mais conhecida como definição). O `arraySize` deve ser uma constante inteira maior que zero. Por exemplo, para dizer ao compilador para reservar 11 elementos para a matriz inteira C, use a declaração -

```
int C[ 12 ]; // C is an array of 12 integers
```

As matrizes podem ser declaradas para conter valores de qualquer tipo de dados sem referência. Por exemplo, uma matriz do tipo `string` pode ser usada para armazenar cadeias de caracteres.

Exemplos usando matrizes

Esta seção fornece muitos exemplos que demonstram como declarar, inicializar e manipular matrizes.

Exemplo 1: Declarando uma matriz e usando um loop para inicializar os elementos da matriz

O programa declara uma matriz inteira de 10 elementos `n`. As linhas a – b usam a instrução **For** para inicializar os elementos da matriz em zeros. Como outras variáveis automáticas, as matrizes automáticas não são implicitamente inicializadas em zero. A primeira instrução de saída (linha c) exibe os cabeçalhos das colunas impressas na instrução subsequente (linhas d – e), que imprime a matriz em formato tabular.

Exemplo

```
int n[ 10 ] ; // n is an array of 10 integers

void setup () {

}

void loop () {
  for ( int i = 0; i < 10; ++i ) // initialize elements of array n to 0 {
    n[ i ] = 0; // set element at location i to 0
    Serial.print (i) ;
```

```

    Serial.print ('\r') ;
  }
  for ( int j = 0; j < 10; ++j ) // output each array element's value {
    Serial.print (n[j]) ;
    Serial.print ('\r') ;
  }
}

```

Resultado - produz o seguinte resultado -

Element	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Exemplo 2: Inicializando uma matriz em uma declaração com uma lista de inicializadores

Os elementos de uma matriz também podem ser inicializados na declaração da matriz, seguindo o nome da matriz com um sinal de igual para e uma lista de inicializadores separados por vírgula, separados por vírgula. O programa usa uma lista de inicializadores para inicializar uma matriz inteira com 10 valores (linha a) e imprime a matriz em formato tabular (linhas b – c).

Exemplo

```

// n is an array of 10 integers
int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 } ;

void setup () {

}

void loop () {
  for ( int i = 0; i < 10; ++i ) {
    Serial.print (i) ;
  }
}

```

```

Serial.print ('\r') ;
}
for ( int j = 0; j < 10; ++j ) // output each array element's value {
    Serial.print (n[j]) ;
    Serial.print ('\r') ;
}
}

```

Resultado - produz o seguinte resultado -

Element	Value
0	32
1	27
2	64
3	18
4	95
5	14
6	90
7	70
8	60
9	37

Exemplo 3: Somando os elementos de uma matriz

Freqüentemente, os elementos de uma matriz representam uma série de valores a serem usados em um cálculo. Por exemplo, se os elementos de uma matriz representam notas do exame, um professor pode desejar totalizar os elementos da matriz e usar essa soma para calcular a média da classe para o exame. O programa soma os valores contidos na matriz inteira de 10 elementos **a** .

Exemplo

```

const int arraySize = 10; // constant variable indicating size of array
int a[ arraySize ] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
int total = 0;

void setup () {

}

void loop () {
    // sum contents of array a
    for ( int i = 0; i < arraySize; ++i )
        total += a[ i ];
}

```

```
Serial.print ("Total of array elements : ") ;  
Serial.print(total) ;  
}
```

Resultado - produz o seguinte resultado -

Total of array elements: 849

As matrizes são importantes para o Arduino e precisam de muito mais atenção. Os seguintes conceitos importantes relacionados à matriz devem ser claros para um Arduino -

S.NO.	Conceito e descrição
1	<u>Passando matrizes para funções</u> Para passar um argumento da matriz para uma função, especifique o nome da matriz sem colchetes.
2	<u>Matrizes multidimensionais</u> Matrizes com duas dimensões (isto é, subscritos) geralmente representam tabelas de valores que consistem em informações organizadas em linhas e colunas.

Arduino - Funções de E / S

Os pinos na placa Arduino podem ser configurados como entradas ou saídas. Explicaremos o funcionamento dos pinos nesses modos. É importante observar que a maioria dos pinos analógicos do Arduino pode ser configurada e usada exatamente da mesma maneira que os pinos digitais.

Pinos configurados como INPUT

Os pinos do Arduino são configurados por padrão como entradas, portanto, eles não precisam ser declarados explicitamente como entradas com **pinMode ()** quando você os estiver usando como entradas. Dizem que os pinos configurados dessa maneira estão em um estado de alta impedância. Os pinos de entrada fazem demandas extremamente pequenas no circuito que estão amostrando, equivalente a um resistor em série de 100 megaohm na frente do pino.

Isso significa que é necessária pouca corrente para alternar o pino de entrada de um estado para outro. Isso torna os pinos úteis para tarefas como implementar um sensor de toque capacitivo ou ler um LED como um fotodiodo.

Os pinos configurados como pinMode (pino, INPUT) sem nada conectado a eles ou com fios conectados a eles que não estão conectados a outros circuitos, relatam

alterações aparentemente aleatórias no estado do pino, captam ruído elétrico do ambiente ou acoplam capacitivamente o estado de um alfinete próximo.

Resistores de pull-up

Os resistores pull-up são geralmente úteis para direcionar um pino de entrada para um estado conhecido se nenhuma entrada estiver presente. Isso pode ser feito adicionando um resistor de pull-up (a + 5V) ou um resistor de pull-down (resistor ao terra) na entrada. Um resistor de 10K é um bom valor para um resistor pull-up ou pull-down.

Usando o resistor pull-up embutido com pinos configurados como entrada

Existem 20.000 resistores pull-up embutidos no chip Atmega que podem ser acessados a partir de software. Esses resistores pull-up embutidos são acessados configurando o **pinMode ()** como INPUT_PULLUP. Isso inverte efetivamente o comportamento do modo INPUT, onde ALTO significa que o sensor está DESLIGADO e BAIXO significa que o sensor está LIGADO. O valor desse pull-up depende do microcontrolador usado. Na maioria das placas baseadas em AVR, o valor é garantido entre 20kΩ e 50kΩ. No Arduino Due, ele está entre 50kΩ e 150kΩ. Para o valor exato, consulte a folha de dados do microcontrolador em sua placa.

Ao conectar um sensor a um pino configurado com INPUT_PULLUP, a outra extremidade deve ser conectada ao terra. No caso de uma chave simples, isso faz com que o pino leia ALTO quando a chave está aberta e BAIXO quando a chave é pressionada. Os resistores pull-up fornecem corrente suficiente para acender um LED vagamente conectado a um pino configurado como entrada. Se os LEDs de um projeto parecem estar funcionando, mas de maneira muito fraca, é provável que isso esteja acontecendo.

Os mesmos registros (locais de memória interna do chip) que controlam se um pino é ALTO ou BAIXO controlam os resistores de pull-up. Conseqüentemente, um pino configurado para ter resistores de pull-up ativados quando o pino está no modo INPUT, terá o pino configurado como HIGH se o pino for então alternado para o modo OUTPUT com **pinMode ()**. Isso também funciona na outra direção, e um pino de saída que é deixado no estado HIGH terá o resistor de pull-up definido se alternado para uma entrada com **pinMode ()**.

Exemplo

```
pinMode(3,INPUT) ; // set pin to input without using built in pull up resistor
pinMode(5,INPUT_PULLUP) ; // set pin to input using built in pull up resistor
```

Pinos configurados como OUTPUT

Os pinos configurados como OUTPUT com **pinMode ()** estão em um estado de baixa impedância. Isso significa que eles podem fornecer uma quantidade substancial de corrente para outros circuitos. Os pinos Atmega podem fornecer (fornecer corrente positiva) ou afundar (fornecer corrente negativa) até 40 mA (miliamperes) de corrente para outros dispositivos / circuitos. Isso é corrente suficiente para acender brilhantemente um LED (não se esqueça do resistor em série) ou acionar muitos sensores, mas não corrente suficiente para acionar relés, solenóides ou motores.

Tentar executar dispositivos de alta corrente a partir dos pinos de saída, pode danificar ou destruir os transistores de saída no pino ou danificar todo o chip Atmega. Frequentemente, isso resulta em um pino "morto" no microcontrolador, mas os chips restantes ainda funcionam adequadamente. Por esse motivo, é uma boa idéia conectar os pinos de SAÍDA a outros dispositivos através de resistores de 470Ω ou 1k, a menos que seja necessária uma corrente máxima extraída dos pinos para uma aplicação específica.

Função pinMode ()

A função `pinMode ()` é usada para configurar um pino específico para se comportar como uma entrada ou uma saída. É possível ativar os resistores pull-up internos com o modo `INPUT_PULLUP`. Além disso, o modo `INPUT` desativa explicitamente as pull-ups internas.

Sintaxe da Função pinMode ()

```
Void setup () {  
  pinMode (pin , mode);  
}
```

- **pin** - o número do pino cujo modo você deseja definir
- **modo** - `INPUT`, `OUTPUT` ou `INPUT_PULLUP`.

Exemplo

```
int button = 5 ; // button connected to pin 5  
int LED = 6; // LED connected to pin 6  
  
void setup () {  
  pinMode(button , INPUT_PULLUP);  
  // set the digital pin as input with pull-up resistor  
  pinMode(button , OUTPUT); // set the digital pin as output  
}  
  
void loop () {  
  If (digitalRead(button ) == LOW) // if button pressed {  
    digitalWrite(LED,HIGH); // turn on led  
    delay(500); // delay for 500 ms  
    digitalWrite(LED,LOW); // turn off led  
    delay(500); // delay for 500 ms  
  }  
}
```

Função digitalWrite ()

A função **digitalWrite ()** é usada para gravar um valor ALTO ou BAIXO em um pino digital. Se o pino tiver sido configurado como `OUTPUT` com `pinMode ()`, sua tensão será ajustada para o valor correspondente: 5V (ou 3,3V em placas de 3,3V) para ALTO, 0V (terra) para BAIXO. Se o pino estiver configurado como `INPUT`,

`digitalWrite ()` habilitará (HIGH) ou desabilitará (LOW) o pullup interno no pino de entrada. É recomendável definir o `pinMode ()` como `INPUT_PULLUP` para ativar o resistor de pull-up interno.

Se você não definir o `pinMode ()` como `OUTPUT` e conectar um LED a um pino, ao chamar `digitalWrite (HIGH)`, o LED poderá parecer escuro. Sem definir explicitamente `pinMode ()`, o `digitalWrite ()` ativará o resistor de pull-up interno, que age como um grande resistor limitador de corrente.

Sintaxe da função `digitalWrite ()`

```
Void loop() {  
  digitalWrite (pin ,value);  
}
```

- **pin** - o número do pino cujo modo você deseja definir
- **valor** - ALTO ou BAIXO.

Exemplo

```
int LED = 6; // LED connected to pin 6  
  
void setup () {  
  pinMode(LED, OUTPUT); // set the digital pin as output  
}  
  
void loop () {  
  digitalWrite(LED,HIGH); // turn on led  
  delay(500); // delay for 500 ms  
  digitalWrite(LED,LOW); // turn off led  
  delay(500); // delay for 500 ms  
}
```

função `analogRead ()`

O Arduino é capaz de detectar se há uma tensão aplicada a um de seus pinos e relatá-la através da função `digitalRead ()`. Há uma diferença entre um sensor liga / desliga (que detecta a presença de um objeto) e um sensor analógico, cujo valor muda continuamente. Para ler esse tipo de sensor, precisamos de um tipo diferente de pino.

Na parte inferior direita da placa Arduino, você verá seis pinos marcados com “Entrada Analógica”. Esses pinos especiais não apenas informam se há tensão aplicada a eles, mas também seu valor. Usando a função **`analogRead ()`**, podemos ler a tensão aplicada a um dos pinos.

Esta função retorna um número entre 0 e 1023, que representa tensões entre 0 e 5 volts. Por exemplo, se houver uma tensão de 2,5 V aplicada ao número de pino 0, `analogRead (0)` retornará 512.

função `analogRead ()` Sintaxe

```
analogRead(pin);
```

- **pin** - o número do pino de entrada analógica para leitura (0 a 5 na maioria das placas, 0 a 7 no Mini e Nano, 0 a 15 no Mega)

Exemplo

```
int analogPin = 3;//potentiometer wiper (middle terminal)
// connected to analog pin 3
int val = 0; // variable to store the value read

void setup() {
  Serial.begin(9600); // setup serial
}

void loop() {
  val = analogRead(analogPin); // read the input pin
  Serial.println(val); // debug value
}
```

Arduino - Função Avançada de E / S

Neste capítulo, aprenderemos algumas funções avançadas de entrada e saída.

Função analogReference ()

Configura a tensão de referência usada para a entrada analógica (ou seja, o valor usado como o topo da faixa de entrada). As opções são -

- **PADRÃO** - A referência analógica padrão de 5 volts (em placas Arduino de 5V) ou 3,3 volts (em placas Arduino de 3,3V)
- **INTERNO** - Uma referência interna, igual a 1,1 volts no ATmega168 ou ATmega328 e 2,56 volts no ATmega8 (não disponível no Arduino Mega)
- **INTERNAL1V1** - Uma referência 1.1V integrada (somente Arduino Mega)
- **INTERNAL2V56** - Uma referência interna de 2,56 V (somente Arduino Mega)
- **EXTERNO** - A tensão aplicada ao pino AREF (apenas 0 a 5V) é usada como referência

Função da função analogReference ()

analogReference (type);

type - pode usar qualquer tipo do seguinte (PADRÃO, INTERNO, INTERNAL1V1, INTERNAL2V56, EXTERNO)

Não use nada menos que 0V ou mais que 5V para tensão de referência externa no pino AREF. Se você estiver usando uma referência externa no pino AREF, defina a referência analógica como EXTERNAL antes de chamar a função **analogRead**

() . Caso contrário, você reduzirá a tensão de referência ativa (gerada internamente) e o pino AREF, possivelmente danificando o microcontrolador na placa Arduino.



Como alternativa, você pode conectar a tensão de referência externa ao pino AREF através de um resistor de 5K, permitindo alternar entre as tensões de referência externas e internas.

Observe que o resistor altera a voltagem usada como referência, porque existe um resistor interno de 32K no pino AREF. Os dois atuam como um divisor de tensão. Por exemplo, 2,5V aplicados através do resistor produzirão $2,5 * 32 / (32 + 5) = \sim 2,2V$ no pino AREF.

Exemplo

```
int analogPin = 3; // potentiometer wiper (middle terminal) connected to analog pin 3
int val = 0; // variable to store the read value

void setup() {
  Serial.begin(9600); // setup serial
  analogReference(EXTERNAL); // the voltage applied to the AREF pin (0 to 5V only)
  // is used as the reference.
}

void loop() {
  val = analogRead(analogPin); // read the input pin
  Serial.println(val); // debug value
}
```

Arduino - Funções dos Personagens

Todos os dados são inseridos nos computadores como caracteres, o que inclui letras, dígitos e vários símbolos especiais. Nesta seção, discutiremos os recursos do C++ para examinar e manipular caracteres individuais.

A biblioteca de manipulação de caracteres inclui várias funções que executam testes e manipulações úteis de dados de caracteres. Cada função recebe um caractere, representado como `int`, ou EOF como argumento. Os caracteres são frequentemente manipulados como números inteiros.

Lembre-se de que o EOF normalmente tem o valor -1 e que algumas arquiteturas de hardware não permitem que valores negativos sejam armazenados em variáveis de caracteres. Portanto, as funções de manipulação de caracteres manipulam caracteres como números inteiros.

A tabela a seguir resume as funções da biblioteca de manipulação de caracteres. Ao usar funções da biblioteca de manipulação de caracteres, inclua o cabeçalho `<cctype>`.

S.No.	Protótipo e descrição
1	<code>int isdigit (int c)</code> Retorna 1 se <code>c</code> for um dígito e 0 caso contrário.
2	<code>int isalpha (int c)</code> Retorna 1 se <code>c</code> for uma letra e 0 caso contrário.
3	<code>int isalnum (int c)</code> Retorna 1 se <code>c</code> for um dígito ou uma letra e 0 caso contrário.
4	<code>int isxdigit (int c)</code> Retorna 1 se <code>c</code> for um caractere de dígito hexadecimal e 0 caso contrário. (Consulte o Apêndice D, Sistemas de números, para obter uma explicação detalhada dos números binários, octais, decimais e hexadecimais.)
5	<code>int islower (int c)</code> Retorna 1 se <code>c</code> for uma letra minúscula e 0 caso contrário.
6	<code>int isupper (int c)</code> Retorna 1 se <code>c</code> for uma letra maiúscula; 0 caso contrário.
7	<code>int isspace (int c)</code> Retorna 1 se <code>c</code> for um caractere de espaço em branco - nova linha (<code>\n</code>), espaço (<code> </code>), avanço de página (<code>\f</code>), retorno de carro (<code>\r</code>), guia horizontal (<code>\t</code>) ou guia vertical (<code>\v</code>) - e 0 caso contrário.

8	int isctrl (int c) Retorna 1 se c for um caractere de controle, como nova linha ('\ n'), avanço de formulário ('\ f'), retorno de carro ('\ r'), guia horizontal ('\ t'), guia vertical ('\ v '), alerta ('\ a ') ou backspace ('\ b ') - e 0 caso contrário.
9	int ispunct (int c) Retorna 1 se c for um caractere de impressão que não seja um espaço, um dígito ou uma letra e 0 caso contrário.
10	int isprint (int c) Retorna 1 se c for um caractere de impressão, incluindo espaço (") e 0, caso contrário.
11	int isgraph (int c) Retorna 1 se c for um caractere de impressão diferente de espaço (") e 0 caso contrário.

Exemplos

O exemplo a seguir demonstra o uso das funções **isdigit**, **isalpha**, **isalnum** e **isxdigit**. A função **isdigit** determina se seu argumento é um dígito (0–9). A função **isalpha** determina se seu argumento é uma letra maiúscula (AZ) ou uma letra minúscula (a – z). A função **isalnum** determina se o argumento é uma letra maiúscula, minúscula ou um dígito. A função **isxdigit** determina se seu argumento é um dígito hexadecimal (A – F, a – f, 0–9).

Exemplo 1

```
void setup () {
  Serial.begin (9600);
  Serial.print ("According to isdigit:\r");
  Serial.print (isdigit( '8' ) ? "8 is a": "8 is not a");
  Serial.print (" digit\r");
  Serial.print (isdigit( '8' ) ?"# is a": "# is not a") ;
  Serial.print (" digit\r");
  Serial.print ("\rAccording to isalpha:\r" );
  Serial.print (isalpha('A' ) ?"A is a": "A is not a");
  Serial.print (" letter\r");
  Serial.print (isalpha('A' ) ?"b is a": "b is not a");
  Serial.print (" letter\r");
  Serial.print (isalpha('A') ?"& is a": "& is not a");
  Serial.print (" letter\r");
  Serial.print (isalpha( 'A' ) ?"4 is a": "4 is not a");
```



```

Serial.print (" letter\r");
Serial.print ("\rAccording to isalnum:\r");
Serial.print (isalnum( 'A' ) ? "A is a" : "A is not a" );

Serial.print (" digit or a letter\r" );
Serial.print (isalnum( '8' ) ? "8 is a" : "8 is not a" );
Serial.print (" digit or a letter\r");
Serial.print (isalnum( '#' ) ? "# is a" : "# is not a" );
Serial.print (" digit or a letter\r");
Serial.print ("\rAccording to isxdigit:\r");
Serial.print (isxdigit( 'F' ) ? "F is a" : "F is not a" );
Serial.print (" hexadecimal digit\r" );
Serial.print (isxdigit( 'J' ) ? "J is a" : "J is not a" );
Serial.print (" hexadecimal digit\r" );
Serial.print (isxdigit( '7' ) ? "7 is a" : "7 is not a" );

Serial.print (" hexadecimal digit\r" );
Serial.print (isxdigit( '$' ) ? "$ is a" : "$ is not a" );
Serial.print (" hexadecimal digit\r" );
Serial.print (isxdigit( 'f' ) ? "f is a" : "f is not a");

}

void loop () {

}

```

Resultado

According to isdigit:
 8 is a digit
 # is not a digit
 According to isalpha:
 A is a letter
 b is a letter
 & is not a letter
 4 is not a letter
 According to isalnum:
 A is a digit or a letter

 8 is a digit or a letter
 # is not a digit or a letter
 According to isxdigit:
 F is a hexadecimal digit
 J is not a hexadecimal digit
 7 is a hexadecimal digit

 \$ is not a hexadecimal digit
 f is a hexadecimal digit

Usamos o operador condicional (**? :**) com cada função para determinar se a sequência "é uma" ou a sequência "não é" deve ser impressa na saída para cada caractere testado. Por exemplo, a linha **a** indica que se '8' for um dígito - ou seja, se **isdigit** retornar um valor verdadeiro (diferente de zero) - a sequência "8 é a" será impressa. Se '8' não for um dígito (ou seja, se o **isdigit** for retornar 0), a string "8 não é a" será impressa.

Exemplo 2

O exemplo a seguir demonstra o uso das funções **islower** e **isupper**. A função **islower** determina se seu argumento é uma letra minúscula (a – z). A função **isupper** determina se o argumento é uma letra maiúscula (A – Z).

```
int thisChar = 0xA0;

void setup () {
  Serial.begin (9600);
  Serial.print ("According to islower:\r") ;
  Serial.print (islower( 'p' ) ? "p is a" : "p is not a" );
  Serial.print ( " lowercase letter\r" );
  Serial.print ( islower( 'P' ) ? "P is a" : "P is not a" );
  Serial.print ("lowercase letter\r");
  Serial.print (islower( '5' ) ? "5 is a" : "5 is not a" );
  Serial.print ( " lowercase letter\r" );
  Serial.print ( islower( '!' )? "! is a" : "! is not a" );
  Serial.print ("lowercase letter\r");

  Serial.print ("\rAccording to isupper:\r") ;
  Serial.print (isupper( 'D' ) ? "D is a" : "D is not an" );
  Serial.print ( " uppercase letter\r" );
  Serial.print ( isupper( 'd' )? "d is a" : "d is not an" );
  Serial.print ( " uppercase letter\r" );
  Serial.print (isupper( '8' ) ? "8 is a" : "8 is not an" );
  Serial.print ( " uppercase letter\r" );
  Serial.print ( islower( '$' )? "$ is a" : "$ is not an" );
  Serial.print ("uppercase letter\r ");
}

void setup () {
}
```

Resultado

```
According to islower:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter
```

According to isupper:
D is an uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
\$ is not an uppercase letter

Exemplo 3

O exemplo a seguir demonstra o uso das funções **isspace**, **isctrl**, **ispunct**, **isprint** e **isgraph**.

- A função **isspace** determina se seu argumento é um caractere de espaço em branco, como espaço (" "), avanço de formulário ("\f"), nova linha ("\n"), retorno de carro ("\r"), guia horizontal ("\t") ou guia vertical ("\v").
- A função **isctrl** determina se o argumento é um caractere de controle, como guia horizontal ("\t"), guia vertical ("\v"), feed de formulário ("\f"), alerta ("\a"), backspace ("\b"), retorno de carro ("\r") ou nova linha ("\n").
- A função **ispunct** determina se seu argumento é um caractere de impressão que não seja um espaço, dígito ou letra, como \$, #, (,), [,], {, }, ., :, ou %.
- A função **isprint** determina se o argumento é um caractere que pode ser exibido na tela (incluindo o caractere de espaço).
- A função **isgraph** testa os mesmos caracteres que isprint, mas o caractere de espaço não está incluído.

```
void setup () {
  Serial.begin (9600);
  Serial.print ( " According to isspace:\r\nNewline " );
  Serial.print ( isspace( '\n' ) ? " is a" : " is not a" );
  Serial.print ( " whitespace character\r\nHorizontal tab" );
  Serial.print ( isspace( '\t' ) ? " is a" : " is not a" );
  Serial.print ( " whitespace character\n" );
  Serial.print ( isspace('%') ? " % is a" : " % is not a" );

  Serial.print ( " \rAccording to isctrl:\r\nNewline" );
  Serial.print ( isctrl( '\n' ) ? "is a" : " is not a" );
  Serial.print ( " control character\r");
  Serial.print ( isctrl( '$' ) ? " $ is a" : " $ is not a" );
  Serial.print ( " control character\r");
  Serial.print ( "\rAccording to ispunct:\r");
  Serial.print ( ispunct(';') ? " is a" : " ; is not a" );
  Serial.print ( " punctuation character\r");
  Serial.print ( ispunct('Y') ? "Y is a" : "Y is not a" );
  Serial.print ( "punctuation character\r");
  Serial.print ( ispunct('#') ? "# is a" : "# is not a" );
  Serial.print ( "punctuation character\r");

  Serial.print ( "\r According to isprint:\r");
  Serial.print ( isprint('$') ? "$ is a" : "$ is not a" );
  Serial.print ( " printing character\r\nAlert " );
}
```

```

Serial.print (isprint('\a' ) ?" is a" : " is not a" );
Serial.print ( " printing character\rSpace ");
Serial.print (isprint(' ' ) ?" is a" : " is not a" );
Serial.print ( " printing character\r");

Serial.print ("\r According to isgraph:\r");
Serial.print (isgraph ('Q' ) ?"Q is a" : "Q is not a" );
Serial.print ("printing character other than a space\rSpace ");
Serial.print (isgraph ( ' ' ) ?" is a" : " is not a" );
Serial.print ("printing character other than a space ");
}

void loop () {
}

```

Resultado

According to isspace:
 Newline is a whitespace character
 Horizontal tab is a whitespace character
 % is not a whitespace character
 According to iscntrl:
 Newline is a control character
 \$ is not a control character
 According to ispunct:
 ; is a punctuation character
 Y is not a punctuation character
 # is a punctuation character
 According to isprint:
 \$ is a printing character
 Alert is not a printing character
 Space is a printing character
 According to isgraph:
 Q is a printing character other than a space
 Space is not a printing character other than a space

Arduino - Biblioteca de Matemática

A biblioteca Arduino Math (math.h) inclui várias funções matemáticas úteis para manipular números de ponto flutuante.

Macros de biblioteca

A seguir, estão as macros definidas no cabeçalho math.h -

Dada a seguir, é apresentada a lista de macros definidas no cabeçalho math.h

Funções da Biblioteca

As seguintes funções são definidas no cabeçalho **math.h** -

Dada a seguir, a lista de funções é definida no cabeçalho math.h

Exemplo

O exemplo a seguir mostra como usar as funções mais comuns da biblioteca math.h -

```
double double__x = 45.45 ;
double double__y = 30.20 ;

void setup() {
  Serial.begin(9600);
  Serial.print("cos num = ");
  Serial.println (cos (double__x) ); // returns cosine of x
  Serial.print("absolute value of num = ");
  Serial.println (fabs (double__x) ); // absolute value of a float
  Serial.print("floating point modulo = ");
  Serial.println (fmod (double__x, double__y)); // floating point modulo
  Serial.print("sine of num = ");
  Serial.println (sin (double__x) ); // returns sine of x
  Serial.print("square root of num : ");
  Serial.println ( sqrt (double__x) ); // returns square root of x
  Serial.print("tangent of num : ");
  Serial.println ( tan (double__x) ); // returns tangent of x
  Serial.print("exponential value of num : ");
  Serial.println ( exp (double__x) ); // function returns the exponential value of x.
  Serial.print("cos num : ");

  Serial.println (atan (double__x) ); // arc tangent of x
  Serial.print("tangent of num : ");
  Serial.println (atan2 (double__y, double__x) ); // arc tangent of y/x
  Serial.print("arc tangent of num : ");
  Serial.println (log (double__x) ) ; // natural logarithm of x
  Serial.print("cos num : ");
  Serial.println ( log10 (double__x)); // logarithm of x to base 10.
  Serial.print("logarithm of num to base 10 : ");
  Serial.println (pow (double__x, double__y) ); // x to power of y
  Serial.print("power of num : ");
  Serial.println (square (double__x)); // square of x
}

void loop() {
}
```

Resultado

cos num = 0.10
absolute value of num = 45.45
floating point modulo = 15.25
sine of num = 0.99
square root of num : 6.74
tangent of num : 9.67
exponential value of num : ovf
cos num : 1.55
tangent of num : 0.59
arc tangent of num : 3.82
cos num : 1.66
logarithm of num to base 10 : inf
power of num : 2065.70

Arduino - Funções trigonométricas

Você precisa usar a trigonometria praticamente como calcular a distância para o objeto em movimento ou a velocidade angular. O Arduino fornece funções trigonométricas tradicionais (sin, cos, tan, asin, acos, atan) que podem ser resumidas escrevendo seus protótipos. Math.h contém o protótipo da função trigonometria.

Sintaxe exata trigonométrica

```
double sin(double x); //returns sine of x radians  
double cos(double y); //returns cosine of y radians  
double tan(double x); //returns the tangent of x radians  
double acos(double x); //returns A, the angle corresponding to cos (A) = x  
double asin(double x); //returns A, the angle corresponding to sin (A) = x  
double atan(double x); //returns A, the angle corresponding to tan (A) = x
```

Exemplo

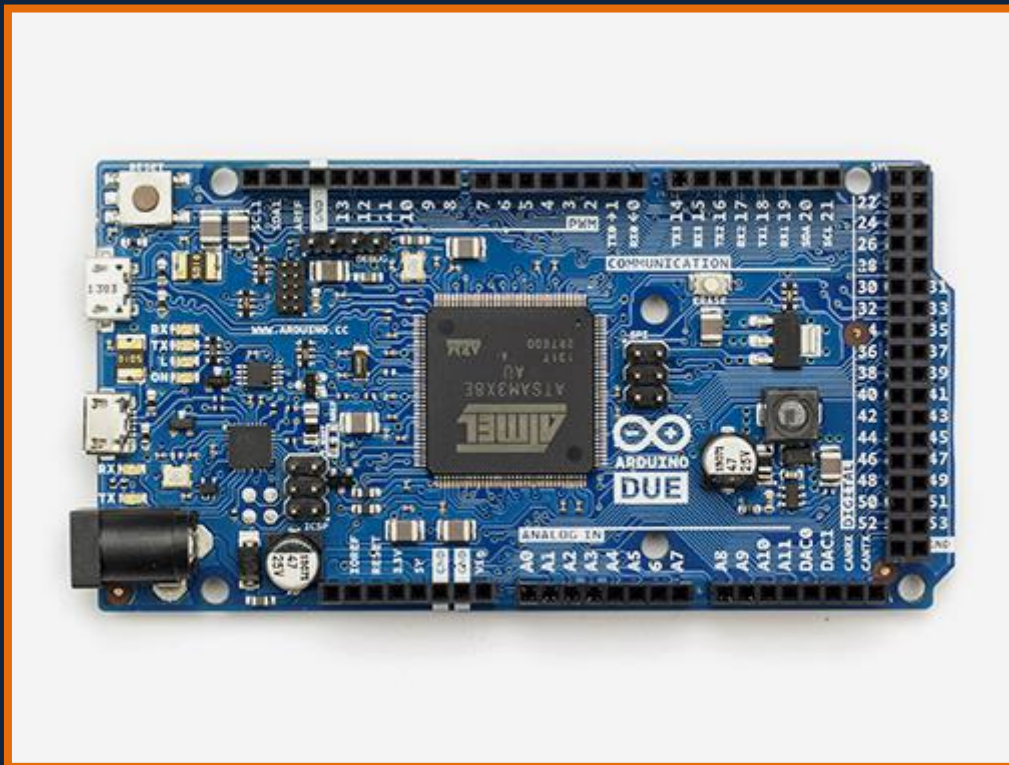
```
double sine = sin(2); // approximately 0.90929737091  
double cosine = cos(2); // approximately -0.41614685058  
double tangent = tan(2); // approximately -2.18503975868
```

Arduino - Due & Zero

O Arduino Due é uma placa de microcontrolador baseada na CPU Atmel SAM3X8E ARM Cortex-M3. É a primeira placa Arduino baseada em um microcontrolador de núcleo ARM de 32 bits.

Recursos importantes -

- Possui 54 pinos de entrada / saída digitais (dos quais 12 podem ser usados como saídas PWM)
- 12 entradas analógicas
- 4 UARTs (portas seriais de hardware)
- Relógio de 84 MHz, uma conexão compatível com USB OTG
- 2 DAC (digital para analógico), 2 TWI, uma tomada de força, um cabeçalho SPI, um cabeçalho JTAG
- Botão de redefinição e um botão de apagar



Características da placa do Arduino Due

Volt de operação	Velocidade da cpu	Entrada / saída analógica	IO / PWM digitais	EEPROM [KB]	SRAM [KB]	Flash [KB]	USB	UART
3.3 Volt	84 Mhz	02/12	54/12	-	96	512	2 micro	4

Comunicação

- 4 UARTs de hardware
- 2 I2C

- 1 Interface CAN (protocolo de comunicação automotiva)
- 1 SPI
- 1 interface JTAG (10 pinos)
- 1 host USB (como Leonardo)
- 1 porta de programação

Diferentemente da maioria das placas Arduino, a placa Arduino Due funciona a 3,3V. A tensão máxima que os pinos de E / S podem tolerar é de 3,3V. A aplicação de tensões superiores a 3,3V a qualquer pino de E / S pode danificar a placa.

A placa contém tudo o necessário para suportar o microcontrolador. Você pode simplesmente conectá-lo a um computador com um cabo micro-USB ou alimentá-lo com um adaptador CA-CC ou bateria para começar. O Due é compatível com todos os escudos do Arduino que funcionam em 3.3V.

Arduino Zero

O Zero é uma extensão simples e poderosa de 32 bits da plataforma estabelecida pela ONU. A placa Zero expande a família, fornecendo desempenho aprimorado, possibilitando uma variedade de oportunidades de projeto para dispositivos, e atua como uma ótima ferramenta educacional para aprender sobre o desenvolvimento de aplicativos de 32 bits.

Recursos importantes são -

- Os aplicativos Zero abrangem desde dispositivos inteligentes de IoT, tecnologia vestível, automação de alta tecnologia, até robótica maluca.
- A placa é alimentada pelo SAMD21 MCU da Atmel, que possui um núcleo ARM Cortex® M0 + de 32 bits.
- Um de seus recursos mais importantes é o Embedded Debugger (EDBG) da Atmel, que fornece uma interface de depuração completa sem a necessidade de hardware adicional, aumentando significativamente a facilidade de uso para depuração de software.
- O EDBG também suporta uma porta COM virtual que pode ser usada para programação de dispositivo e carregador de inicialização.



Características da placa Arduino Zero

Volt de operação	Velocidade da cpu	Entrada / saída analógica	IO / PWM digitais	EEPROM [KB]	SRAM [KB]	Flash [KB]	USB	UART
3.3 Volt	48 Mhz	1/6	14/10	-	32.	256	2 micro	2

Diferentemente da maioria das placas Arduino e Genuino, o Zero funciona em 3,3V. A tensão máxima que os pinos de E / S podem tolerar é de 3,3V. A aplicação de tensões superiores a 3,3V a qualquer pino de E / S pode danificar a placa.

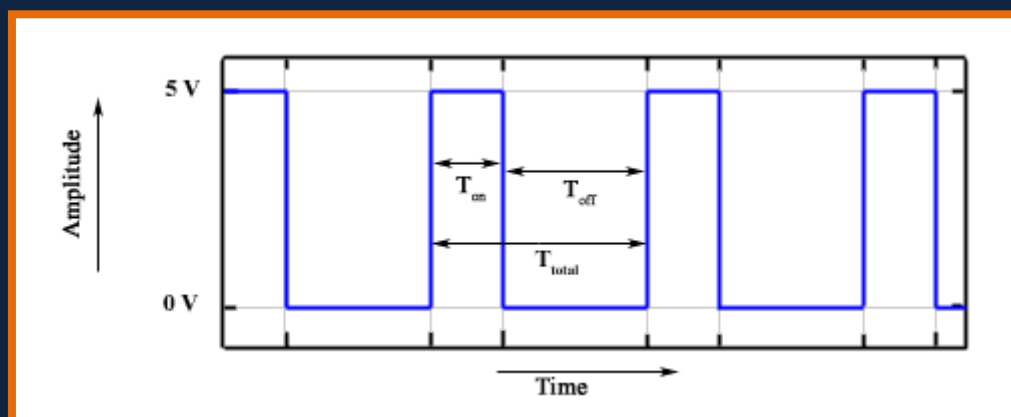
A placa contém tudo o necessário para suportar o microcontrolador. Você pode simplesmente conectá-lo a um computador com um cabo micro-USB ou alimentá-lo com um adaptador CA-CC ou uma bateria para começar. O Zero é compatível com todos os escudos que funcionam em 3.3V.

Arduino - Modulação por Largura de Pulso

A modulação por largura de pulso ou PWM é uma técnica comum usada para variar a largura dos pulsos em um trem de pulsos. O PWM possui muitas aplicações, como o controle de servos e controladores de velocidade, limitando a potência efetiva de motores e LEDs.

Princípio básico de PWM

A modulação da largura de pulso é basicamente uma onda quadrada com um tempo alto e baixo variável. Um sinal PWM básico é mostrado na figura a seguir.



Existem vários termos associados ao PWM -

- **On-Time** - A duração do sinal de tempo é alta.

- **Off-Time** - O sinal de duração é baixo.
- **Período** - É representado como a soma do tempo de ativação e desativação do sinal PWM.
- **Ciclo de serviço** - É representado como a porcentagem de sinal de tempo que permanece durante o período do sinal PWM.

Período

Como mostrado na figura, T_{on} indica o tempo de ativação e T_{off} indica o tempo de desativação do sinal. Período é a soma dos tempos de ativação e desativação e é calculado como mostrado na equação a seguir -

$$T_{total} = T_{ativado} + T_{desativado}$$

Ciclo de trabalho

O ciclo de serviço é calculado como o horário do período. Usando o período calculado acima, o ciclo de serviço é calculado como -

$$D = \frac{T_{on}}{T_{on} + T_{off}} = \frac{T_{on}}{T_{total}}$$

Função analogWrite ()

A função **analogWrite ()** grava um valor analógico (onda PWM) em um pino. Pode ser usado para acender um LED com brilho variável ou acionar um motor em várias velocidades. Após uma chamada da função **analogWrite ()**, o pino gerará uma onda quadrada constante do ciclo de trabalho especificado até a próxima chamada para **analogWrite ()** ou uma chamada para **digitalRead ()** ou **digitalWrite ()** no mesmo pino. A frequência do sinal PWM na maioria dos pinos é de aproximadamente 490 Hz. Nas placas Uno e similares, os pinos 5 e 6 têm uma frequência de aproximadamente 980 Hz. Os pinos 3 e 11 no Leonardo também funcionam a 980 Hz.

Na maioria das placas Arduino (aquelas com ATmega168 ou ATmega328), essa função funciona nos pinos 3, 5, 6, 9, 10 e 11. No Arduino Mega, funciona nos pinos 2 - 13 e 44 - 46. Arduino mais antigo as placas com um ATmega8 suportam apenas **analogWrite ()** nos pinos 9, 10 e 11.



O Arduino Due suporta **analogWrite ()** nos pinos 2 a 13 e nos pinos DAC0 e DAC1. Ao contrário dos pinos PWM, o DAC0 e o DAC1 são conversores de Digital para Analógico e atuam como saídas analógicas verdadeiras.

Você não precisa chamar **pinMode ()** para definir o pino como uma saída antes de chamar **analogWrite ()**.

Função de **analogWrite ()** Sintaxe

analogWrite (pin , value) ;

valor - o ciclo de serviço: entre 0 (sempre desativado) e 255 (sempre ativado).

Exemplo

```
int ledPin = 9; // LED connected to digital pin 9
int analogPin = 3; // potentiometer connected to analog pin 3
int val = 0; // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT); // sets the pin as output
}

void loop() {
  val = analogRead(analogPin); // read the input pin
  analogWrite(ledPin, (val / 4)); // analogRead values go from 0 to 1023,
  // analogWrite values from 0 to 255
}
```

Arduino - Números Aleatórios

Para gerar números aleatórios, você pode usar as funções de números aleatórios do Arduino. Temos duas funções -

- **randomSeed (semente)**
- **aleatória()**

randomSeed (semente)

A função **randomSeed (seed)** redefine o gerador de números pseudo-aleatórios do Arduino. Embora a distribuição dos números retornados por **random ()** seja essencialmente aleatória, a sequência é previsível. Você deve redefinir o gerador para algum valor aleatório. Se você tiver um pino analógico desconectado, ele poderá captar ruídos aleatórios no ambiente ao redor. Podem ser ondas de rádio, raios cósmicos, interferência eletromagnética de telefones celulares, luzes fluorescentes e assim por diante.

Exemplo

```
randomSeed(analogRead(5)); // randomize using noise from analog pin 5
```

aleatória()

A função aleatória gera números pseudo-aleatórios. A seguir está a sintaxe.

Sintaxe de declarações random ()

```
long random(max) // it generate random numbers from 0 to max
```

```
long random(min, max) // it generate random numbers from min to max
```

Exemplo

```
long randNumber;

void setup() {
  Serial.begin(9600);
  // if analog input pin 0 is unconnected, random analog
  // noise will cause the call to randomSeed() to generate
  // different seed numbers each time the sketch runs.
  // randomSeed() will then shuffle the random function.
  randomSeed(analogRead(0));
}

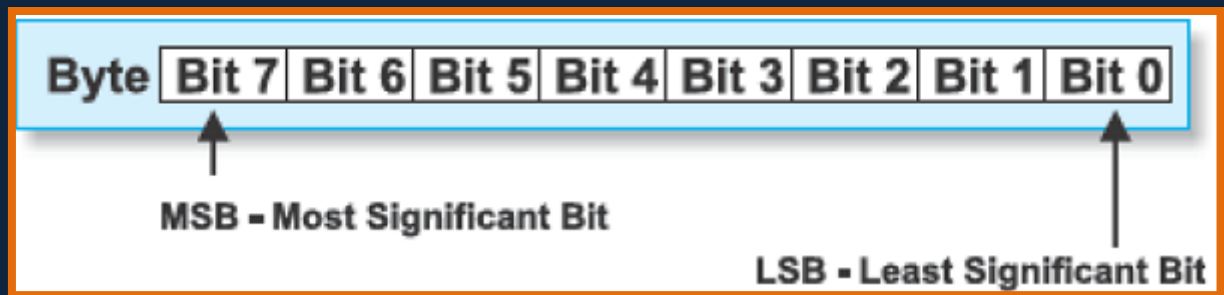
void loop() {
  // print a random number from 0 to 299
  Serial.print("random1=");
  randNumber = random(300);
  Serial.println(randNumber); // print a random number from 0 to 299
  Serial.print("random2=");
  randNumber = random(10, 20); // print a random number from 10 to 19
  Serial.println(randNumber);
  delay(50);
}
```

Vamos agora atualizar nosso conhecimento sobre alguns dos conceitos básicos, como bits e bytes.

Bits

Um bit é apenas um dígito binário.

- O sistema binário usa dois dígitos, 0 e 1.
- Semelhante ao sistema de números decimais, no qual os dígitos de um número não têm o mesmo valor, a 'significância' de um bit depende de sua posição no número binário. Por exemplo, os dígitos no número decimal 666 são iguais, mas têm valores diferentes.



Bytes

Um byte consiste em oito bits.

- Se um bit é um dígito, é lógico que os bytes representem números.
- Todas as operações matemáticas podem ser executadas sobre elas.
- Os dígitos em um byte também não têm o mesmo significado.
- O bit mais à esquerda tem o maior valor chamado de bit mais significativo (MSB).
- O bit mais à direita tem o menor valor e, portanto, é chamado de bit menos significativo (LSB).
- Como oito zeros e os de um byte podem ser combinados de 256 maneiras diferentes, o maior número decimal que pode ser representado por um byte é 255 (uma combinação representa um zero).

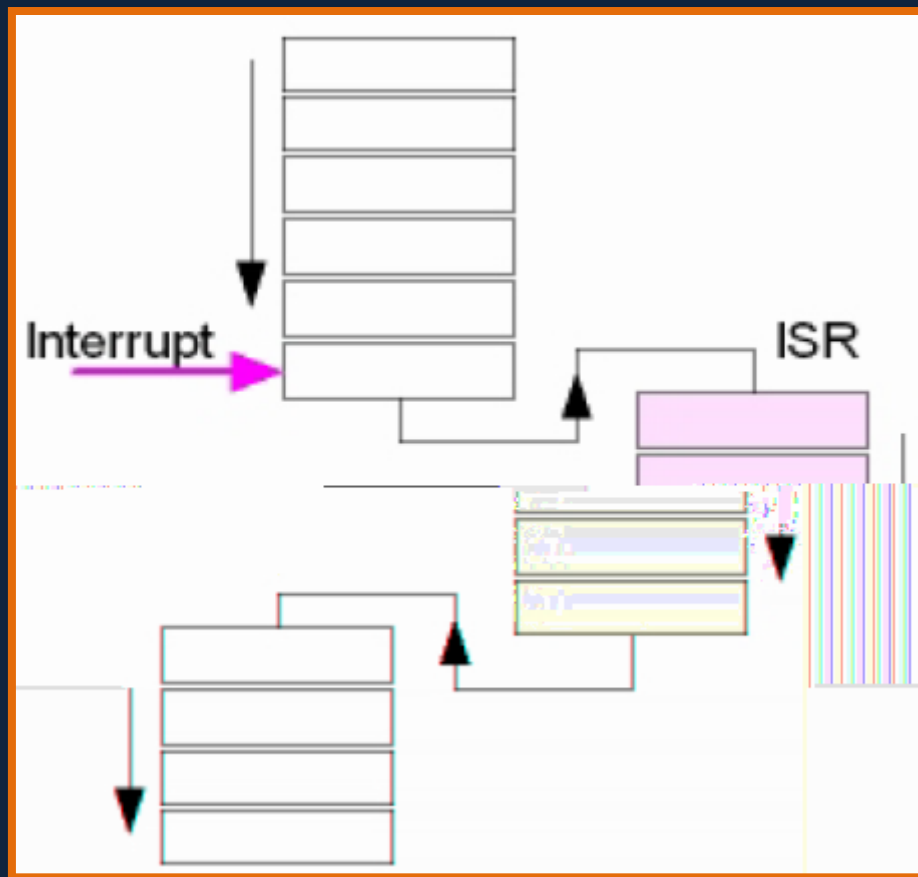
Arduino - Interrupções

As interrupções interrompem o trabalho atual do Arduino, de forma que algum outro trabalho possa ser feito.

Suponha que você esteja sentado em casa, conversando com alguém. De repente, o telefone toca. Você para de conversar e pega o telefone para falar com o chamador. Quando você termina sua conversa telefônica, volta a conversar com a pessoa antes de o telefone tocar.

Da mesma forma, você pode pensar na rotina principal como conversar com alguém; o toque do telefone faz com que você pare de conversar. A rotina do serviço de interrupção é o processo de falar ao telefone. Quando a conversa telefônica termina, você volta à sua rotina principal de conversar. Este exemplo explica exatamente como uma interrupção faz com que um processador atue.

O programa principal está sendo executado e executando algumas funções em um circuito. No entanto, quando ocorre uma interrupção, o programa principal é interrompido enquanto outra rotina é executada. Quando essa rotina termina, o processador volta à rotina principal novamente.



Características importantes

Aqui estão alguns recursos importantes sobre interrupções -

- As interrupções podem vir de várias fontes. Nesse caso, estamos usando uma interrupção de hardware que é acionada por uma alteração de estado em um dos pinos digitais.
- A maioria dos projetos do Arduino possui duas interrupções de hardware (chamadas de "interrupt0" e "interrupt1") conectadas aos pinos de E / S digitais 2 e 3, respectivamente.
- O Arduino Mega possui seis interrupções de hardware, incluindo as interrupções adicionais ("interrupt2" a "interrupt5") nos pinos 21, 20, 19 e 18.
- Você pode definir uma rotina usando uma função especial denominada "Rotina de serviço de interrupção" (geralmente conhecida como ISR).
- Você pode definir a rotina e especificar condições na borda ascendente, descendente ou em ambos. Nessas condições específicas, a interrupção seria atendida.
- É possível executar essa função automaticamente, sempre que um evento ocorre em um pino de entrada.

Tipos de interrupções

Existem dois tipos de interrupções -

- **Interrupções de hardware** - Elas ocorrem em resposta a um evento externo, como um pino de interrupção externo ficando alto ou baixo.
- **Interrupções de software** - ocorrem em resposta a uma instrução enviada no software. O único tipo de interrupção suportado pelo "idioma do Arduino" é a função `attachInterrupt()`.

Usando interrupções no Arduino

As interrupções são muito úteis nos programas do Arduino, pois ajudam a resolver problemas de tempo. Uma boa aplicação de uma interrupção é ler um codificador rotativo ou observar uma entrada do usuário. Geralmente, um ISR deve ser o mais curto e rápido possível. Se o seu esboço usa vários ISRs, apenas um pode ser executado por vez. Outras interrupções serão executadas depois que a atual terminar em uma ordem que depende da prioridade que elas têm.

Normalmente, variáveis globais são usadas para transmitir dados entre um ISR e o programa principal. Para garantir que as variáveis compartilhadas entre um ISR e o programa principal sejam atualizadas corretamente, declare-as como voláteis.

Sintaxe da instrução `attachInterrupt`

```
attachInterrupt(digitalPinToInterrupt(pin),ISR,mode);//recommended for arduino board
attachInterrupt(pin, ISR, mode) ; //recommended Arduino Due, Zero only
//argument pin: the pin number
//argument ISR: the ISR to call when the interrupt occurs;
//this function must take no parameters and return nothing.
//This function is sometimes referred to as an interrupt service routine.
//argument mode: defines when the interrupt should be triggered.
```

As três constantes a seguir são predefinidas como valores válidos -

- **LOW** para acionar a interrupção sempre que o pino estiver baixo.
- **MUDAR** para acionar a interrupção sempre que o pino mudar de valor.
- **QUEDA** sempre que o pino passa de alto a baixo.

Exemplo

```
int pin = 2; //define interrupt pin to 2
volatile int state = LOW; // To make sure variables shared between an ISR
//the main program are updated correctly,declare them as volatile.

void setup() {
  pinMode(13, OUTPUT); //set pin 13 as output
  attachInterrupt(digitalPinToInterrupt(pin), blink, CHANGE);
  //interrupt at pin 2 blink ISR when pin to change the value
}
void loop() {
```

```

digitalWrite(13, state); //pin 13 equal the state value
}

void blink() {
  //ISR function
  state = !state; //toggle the state when the interrupt occurs
}

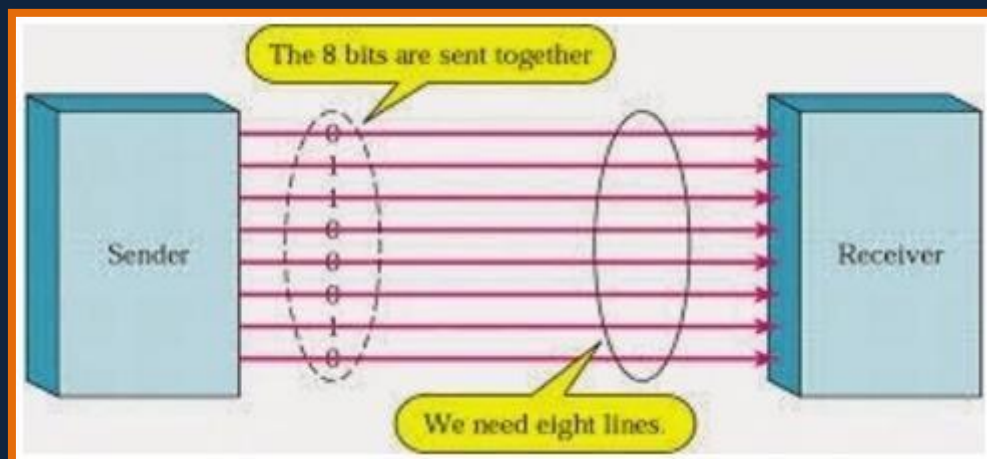
```

Arduino - Comunicação

Centenas de protocolos de comunicação foram definidos para alcançar essa troca de dados. Cada protocolo pode ser categorizado em uma das duas categorias: paralela ou serial.

Comunicação Paralela

A conexão paralela entre o Arduino e os periféricos via portas de entrada / saída é a solução ideal para distâncias menores até vários metros. No entanto, em outros casos, quando é necessário estabelecer comunicação entre dois dispositivos para distâncias maiores, não é possível usar a conexão paralela. Interfaces paralelas transferem vários bits ao mesmo tempo. Eles geralmente exigem barramentos de dados - transmitindo através de oito, dezesseis ou mais fios. Os dados são transferidos em enormes ondas de 1 e 0.



Vantagens e desvantagens da comunicação paralela

A comunicação paralela certamente tem suas vantagens. É mais rápido do que serial, direto e relativamente fácil de implementar. No entanto, requer muitas portas e linhas de entrada / saída (E / S). Se você já teve que mudar um projeto de um Arduino Uno básico para um Mega, sabe que as linhas de E / S em um microprocessador podem ser preciosas e poucas. Portanto, preferimos a comunicação serial, sacrificando a velocidade potencial para o imobilizado de pinos.

Módulos de comunicação serial

Hoje, a maioria das placas Arduino são construídas com vários sistemas diferentes para comunicação serial como equipamento padrão.

Qual desses sistemas é usado depende dos seguintes fatores -

- Com quantos dispositivos o microcontrolador precisa trocar dados?
- Qual a velocidade da troca de dados?
- Qual a distância entre esses dispositivos?
- É necessário enviar e receber dados simultaneamente?

Uma das coisas mais importantes em relação à comunicação serial é o **Protocolo**, que deve ser rigorosamente observado. É um conjunto de regras que devem ser aplicadas para que os dispositivos possam interpretar corretamente os dados que eles trocam mutuamente. Felizmente, o Arduino cuida disso automaticamente, para que o trabalho do programador / usuário seja reduzido a simples gravação (dados a serem enviados) e leitura (dados recebidos).

Tipos de comunicação serial

A comunicação serial pode ser classificada ainda como -

- **Síncrono** - Os dispositivos sincronizados usam o mesmo relógio e seu tempo está sincronizado.
- **Assíncrono** - Os dispositivos assíncronos têm seus próprios relógios e são acionados pela saída do estado anterior.

É fácil descobrir se um dispositivo é síncrono ou não. Se o mesmo relógio for dado a todos os dispositivos conectados, eles serão síncronos. Se não houver linha de relógio, é assíncrona.

Por exemplo, o módulo UART (Transmissor universal de receptor assíncrono) é assíncrono.

O protocolo serial assíncrono possui várias regras internas. Essas regras nada mais são do que mecanismos que ajudam a garantir transferências de dados robustas e sem erros. Esses mecanismos, que temos para evitar o sinal do relógio externo, são -

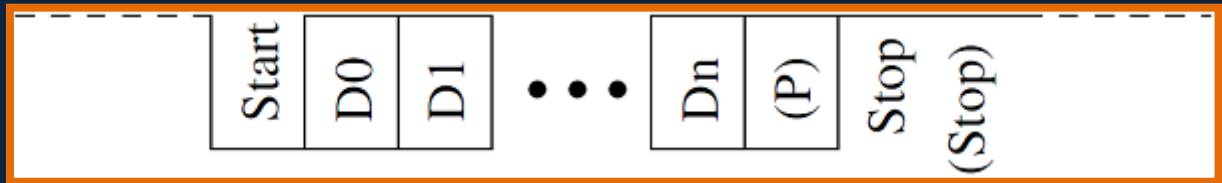
- Bits de sincronização
- Bits de dados
- Bits de paridade
- Taxa de transmissão

Bits de sincronização

Os bits de sincronização são dois ou três bits especiais transferidos com cada pacote de dados. Eles são o bit inicial e o (s) bit (s) de parada. Fiel ao seu nome, esses bits marcam o início e o fim de um pacote, respectivamente.

Sempre existe apenas um bit de início, mas o número de bits de parada é configurável para um ou dois (embora normalmente seja deixado em um).

O bit inicial é sempre indicado por uma linha de dados ociosa indo de 1 a 0, enquanto o (s) bit (s) de parada retornam ao estado ocioso mantendo a linha em 1.



Bits de dados

A quantidade de dados em cada pacote pode ser configurada para qualquer tamanho de 5 a 9 bits. Certamente, o tamanho padrão dos dados é o byte básico de 8 bits, mas outros tamanhos têm seus usos. Um pacote de dados de 7 bits pode ser mais eficiente que 8, especialmente se você estiver apenas transferindo caracteres ASCII de 7 bits.

Bits de paridade

O usuário pode selecionar se deve haver um bit de paridade ou não e, se sim, se a paridade deve ser ímpar ou par. O bit de paridade é 0 se o número de 1s entre os bits de dados for par. Paridade ímpar é exatamente o oposto.


Taxa de transmissão

O termo taxa de transmissão é usado para indicar o número de bits transferidos por segundo [bps]. Observe que se refere a bits, não bytes. Geralmente, é exigido pelo protocolo que cada byte seja transferido junto com vários bits de controle. Isso significa que um byte no fluxo de dados serial pode consistir em 11 bits. Por exemplo, se a taxa de transmissão for de 300 bps, o máximo de 37 e o mínimo de 27 bytes poderão ser transferidos por segundo.

Arduino UART

O código a seguir fará com que o Arduino envie alô mundo quando for inicializado.

```
void setup() {  
  Serial.begin(9600); //set up serial library baud rate to 9600  
  Serial.println("hello world"); //print hello world  
}  
  
void loop() {  
  
}
```

Após o upload do esboço do Arduino para o Arduino, abra o monitor Serial  na seção superior direita do IDE do Arduino.

Digite qualquer coisa na caixa superior do Monitor serial e pressione enviar ou entrar no teclado. Isso enviará uma série de bytes para o Arduino.

O código a seguir retorna o que recebe como entrada.

O código a seguir fará com que o Arduino entregue a saída, dependendo da entrada fornecida.

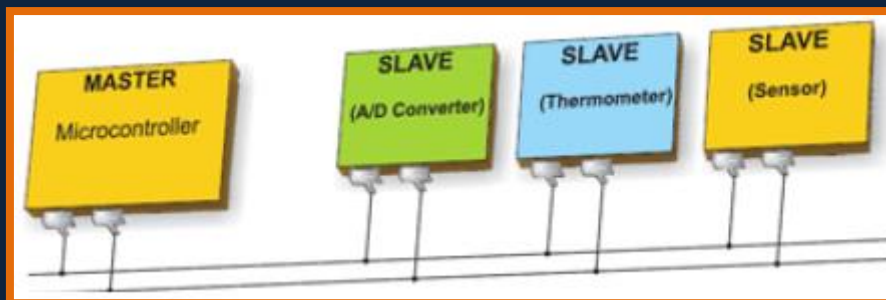
```
void setup() {  
  Serial.begin(9600); //set up serial library baud rate to 9600  
}  
  
void loop() {  
  if(Serial.available()) //if number of bytes (characters) available for reading from {  
    serial port  
    Serial.print("I received:"); //print I received  
    Serial.write(Serial.read()); //send what you read  
  }  
}
```

Observe que **Serial.print** e **Serial.println** enviarão de volta o código ASCII real, enquanto **Serial.write** enviará de volta o texto real. Consulte os códigos ASCII para obter mais informações.

Arduino - Circuito Integrado Inter

O circuito integrado (I2C) é um sistema para troca de dados seriais entre os microcontroladores e os circuitos integrados especializados de uma nova geração. É usado quando a distância entre eles é curta (o receptor e o transmissor geralmente estão no mesmo cartão impresso). A conexão é estabelecida através de dois condutores. Um é usado para transferência de dados e o outro é usado para sincronização (sinal de relógio).

Como visto na figura a seguir, um dispositivo é sempre um mestre. Ele executa o endereçamento de um chip escravo antes do início da comunicação. Dessa maneira, um microcontrolador pode se comunicar com 112 dispositivos diferentes. A taxa de transmissão geralmente é de 100 Kb / s (modo padrão) ou 10 Kb / s (modo de velocidade lenta). Sistemas com taxa de transmissão de 3,4 Mb / s apareceram recentemente. A distância entre os dispositivos, que se comunicam através de um barramento I2C, é limitada a vários metros.



Placa I2C Pins

O barramento I2C consiste em dois sinais - SCL e SDA. SCL é o sinal do relógio e SDA é o sinal de dados. O atual mestre do barramento sempre gera o sinal do relógio. Alguns dispositivos escravos podem forçar o relógio a reduzir o tempo às vezes para atrasar o envio de mais dados pelo mestre (ou exigir mais tempo para preparar os dados antes que o mestre tente registrá-los). Isso é conhecido como "alongamento do relógio".

A seguir, estão os pinos para diferentes placas Arduino -

- Uno, Pro Mini A4 (SDA), A5 (SCL)
- Mega, pino 20 (SDA), 21 (SCL)
- Leonardo, Yun 2 (SDA), 3 (SCL)

Arduino I2C

Temos dois modos - código mestre e código escravo - para conectar duas placas Arduino usando I2C. Eles são -

- Transmissor Mestre / Receptor Escravo
- Receptor Mestre / Transmissor Escravo

Transmissor Mestre / Receptor Escravo

Vamos agora ver o que é o transmissor mestre e o receptor escravo.

Transmissor Mestre

As seguintes funções são usadas para inicializar a biblioteca do Wire e ingressar no barramento I2C como mestre ou escravo. Isso normalmente é chamado apenas uma vez.

- **Wire.begin (endereço)** - O endereço é o endereço escravo de 7 bits no nosso caso, pois o mestre não está especificado e ele ingressará no barramento como mestre.
- **Wire.beginTransmission (address)** - Inicie uma transmissão para o dispositivo escravo I2C com o endereço fornecido.
- **Wire.write (value)** - **enfileira** os bytes para transmissão de um dispositivo mestre para um escravo (chamadas intermediárias para beginTransmission () e endTransmission ()).
- **Wire.endTransmission ()** - Termina uma transmissão para um dispositivo escravo iniciado por beginTransmission () e transmite os bytes que foram enfileirados por wire.write ().

Exemplo

```
#include <Wire.h> //include wire library
```

```

void setup() //this will run only once {
  Wire.begin(); // join i2c bus as master
}

short age = 0;

void loop() {
  Wire.beginTransaction(2);
  // transmit to device #2
  Wire.write("age is = ");
  Wire.write(age); // sends one byte
  Wire.endTransmission(); // stop transmitting
  delay(1000);
}

```

Receptor Escravo

As seguintes funções são usadas -

- **Wire.begin (endereço)** - Endereço é o endereço escravo de 7 bits.
- **Wire.onReceive (manipulador de dados recebido)** - Função a ser chamada quando um dispositivo escravo recebe dados do mestre.
- **Wire.available ()** - Retorna o número de bytes disponíveis para recuperação com `Wire.read ()`. Isso deve ser chamado dentro do manipulador `Wire.onReceive ()`.

Exemplo

```

#include <Wire.h> //include wire library

void setup() { //this will run only once
  Wire.begin(2); // join i2c bus with address #2
  Wire.onReceive(receiveEvent); // call receiveEvent when the master send any thing
  Serial.begin(9600); // start serial for output to print what we receive
}

void loop() {
  delay(250);
}

//-----this function will execute whenever data is received from master-----//

void receiveEvent(int howMany) {
  while (Wire.available()>1) // loop through all but the last {
    char c = Wire.read(); // receive byte as a character
    Serial.print(c); // print the character
  }
}

```

Receptor Mestre / Transmissor Escravo

Vamos agora ver o que é o receptor principal e o transmissor escravo.

Receptor Mestre

O mestre é programado para solicitar e, em seguida, ler bytes de dados que são enviados do Slave Arduino endereçado exclusivamente.

A seguinte função é usada -

Wire.requestFrom (endereço, número de bytes) - Usado pelo mestre para solicitar bytes de um dispositivo escravo. Os bytes podem então ser recuperados com as funções `wire.available ()` e `wire.read ()`.

Exemplo

```
#include <Wire.h> //include wire library
void setup() {
  Wire.begin(); // join i2c bus (address optional for master)
  Serial.begin(9600); // start serial for output
}

void loop() {
  Wire.requestFrom(2, 1); // request 1 bytes from slave device #2
  while (Wire.available()) // slave may send less than requested {
    char c = Wire.read(); // receive a byte as character
    Serial.print(c); // print the character
  }
  delay(500);
}
```

Transmissor Escravo

A seguinte função é usada.

Wire.onRequest (manipulador) - Uma função é chamada quando um mestre solicita dados deste dispositivo escravo.

Exemplo

```
#include <Wire.h>

void setup() {
  Wire.begin(2); // join i2c bus with address #2
  Wire.onRequest(requestEvent); // register event
}

Byte x = 0;

void loop() {
  delay(100);
}
```

```
// function that executes whenever data is requested by master
// this function is registered as an event, see setup()

void requestEvent() {
  Wire.write(x); // respond with message of 1 bytes as expected by master
  x++;
}
```

Arduino - Interface Periférica Serial

Um barramento de interface periférica serial (SPI) é um sistema para comunicação serial, que utiliza até quatro condutores, geralmente três. Um condutor é usado para recebimento de dados, um para envio de dados, um para sincronização e outro para selecionar um dispositivo com o qual se comunicar. É uma conexão full duplex, o que significa que os dados são enviados e recebidos simultaneamente. A taxa de transmissão máxima é superior à do sistema de comunicação I2C.

Pinos do SPI da placa

O SPI usa os quatro fios a seguir -

- **SCK** - Este é o relógio serial acionado pelo mestre.
- **MOSI** - Esta é a saída principal / entrada escrava acionada pelo mestre.
- **MISO** - Esta é a entrada principal / saída escrava acionada pelo mestre.
- **SS** - Este é o fio de seleção de escravos.

As seguintes funções são usadas. Você precisa incluir o SPI.h.

- **SPI.begin ()** - Inicializa o barramento SPI configurando SCK, MOSI e SS para saídas, puxando SCK e MOSI para baixo e SS alto.
- **SPI.setClockDivider (divider)** - Para definir o divisor do relógio SPI em relação ao relógio do sistema. Nas placas baseadas em AVR, os divisores disponíveis são 2, 4, 8, 16, 32, 64 ou 128. A configuração padrão é SPI_CLOCK_DIV4, que define o relógio SPI para um quarto da frequência do relógio do sistema (5 Mhz para o placas a 20 MHz).
- **Divisor** - pode ser (SPI_CLOCK_DIV2, SPI_CLOCK_DIV4, SPI_CLOCK_DIV8, SPI_CLOCK_DIV16, SPI_CLOCK_DIV32, SPI_CLOCK_DIV64, SPI_CLOCK_DIV128).
- **SPI.transfer (val)** - a transferência SPI é baseada em um envio e recebimento simultâneos: os dados recebidos são retornados em receivedVal.
- **SPI.beginTransaction (SPISettings (speedMaximum, dataOrder, dataMode))** - speedMaximum é o relógio, dataOrder (MSBFIRST ou LSBFIRST), dataMode (SPI_MODE0, SPI_MODE1, SPI_MODE2 ou SPI_MODE3).

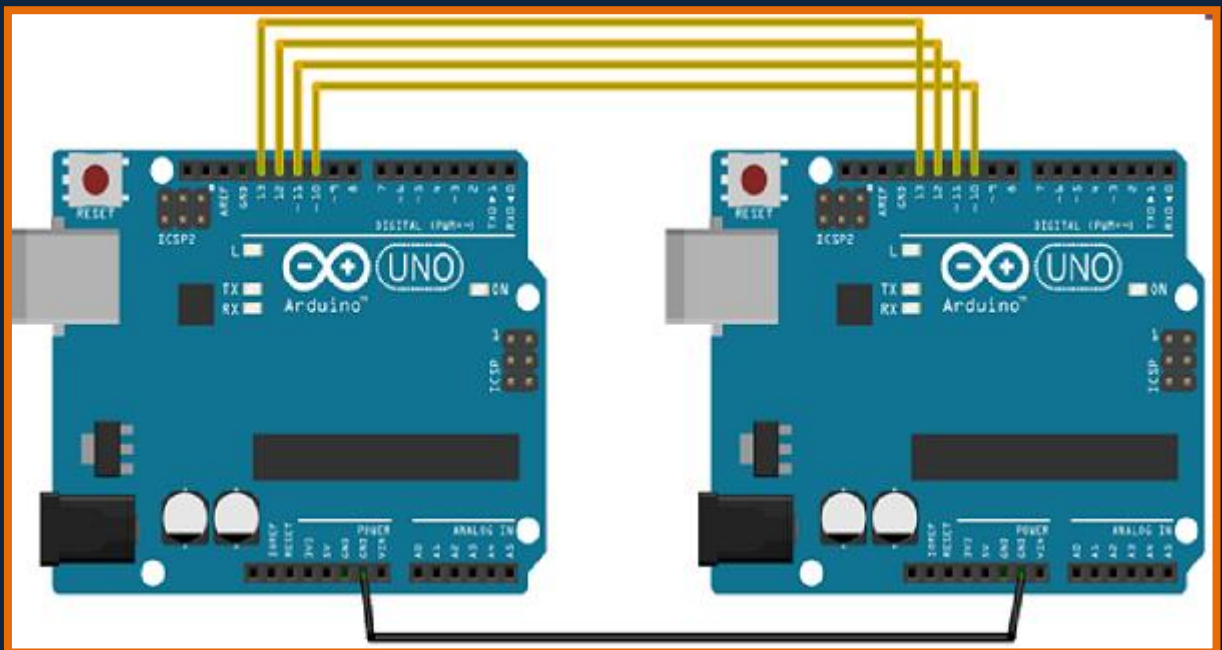
Temos quatro modos de operação no SPI da seguinte forma:

- **Modo 0 (o padrão)** - O relógio normalmente é baixo (CPOL = 0) e os dados são amostrados na transição de baixa para alta (borda superior) (CPHA = 0).
- **Modo 1** - O relógio normalmente é baixo (CPOL = 0) e os dados são amostrados na transição de alto para baixo (borda de fuga) (CPHA = 1).
- **Modo 2** - O relógio normalmente é alto (CPOL = 1) e os dados são amostrados na transição de alto para baixo (borda principal) (CPHA = 0).
- **Modo 3** - O relógio normalmente é alto (CPOL = 1) e os dados são amostrados na transição de baixa para alta (borda de fuga) (CPHA = 1).
- **SPI.attachInterrupt (manipulador)** - Função a ser chamada quando um dispositivo escravo recebe dados do mestre.

Agora, conectaremos duas placas Arduino UNO juntas; um como mestre e o outro como escravo.

- (SS): pino 10
- (MOSI): pino 11
- (MISO): pino 12
- (SCK): pino 13

O chão é comum. A seguir, é apresentada a representação esquemática da conexão entre as duas placas -



Vamos ver exemplos de SPI como mestre e SPI como escravo.

SPI como MASTER

Exemplo

```
#include <SPI.h>

void setup (void) {
  Serial.begin(115200); //set baud rate to 115200 for usart
  digitalWrite(SS, HIGH); // disable Slave Select
  SPI.begin ();
  SPI.setClockDivider(SPI_CLOCK_DIV8); //divide the clock by 8
}

void loop (void) {
  char c;
  digitalWrite(SS, LOW); // enable Slave Select
  // send test string
  for (const char * p = "Hello, world!\r" ; c = *p; p++) {
    SPI.transfer (c);
    Serial.print(c);
  }
  digitalWrite(SS, HIGH); // disable Slave Select
  delay(2000);
}
```

SPI como ESCRAVO

Exemplo

```
#include <SPI.h>
char buff [50];
volatile byte indx;
volatile boolean process;

void setup (void) {
  Serial.begin (115200);
  pinMode(MISO, OUTPUT); // have to send on master in so it set as output
  SPCR |= _BV(SPE); // turn on SPI in slave mode
  indx = 0; // buffer empty
  process = false;
  SPI.attachInterrupt(); // turn on interrupt
}
ISR (SPI_STC_vect) // SPI interrupt routine {
  byte c = SPDR; // read byte from SPI Data Register
  if (indx < sizeof buff) {
    buff [indx++] = c; // save data in the next index in the array buff
    if (c == '\r') //check for the end of the word
      process = true;
  }
}
```



```

    }
}

void loop (void) {
  if (process) {
    process = false; //reset the process
    Serial.println (buff); //print the array on serial monitor
    indx= 0; //reset button to zero
  }
}

```

Projetos com Arduino

Arduino - LED piscando

Os LEDs são luzes pequenas e poderosas usadas em muitas aplicações diferentes. Para começar, trabalharemos para piscar um LED, o Hello World dos microcontroladores. É tão simples quanto acender e apagar uma luz. O estabelecimento dessa importante linha de base fornecerá uma base sólida à medida que trabalhamos em direção a experimentos mais complexos.

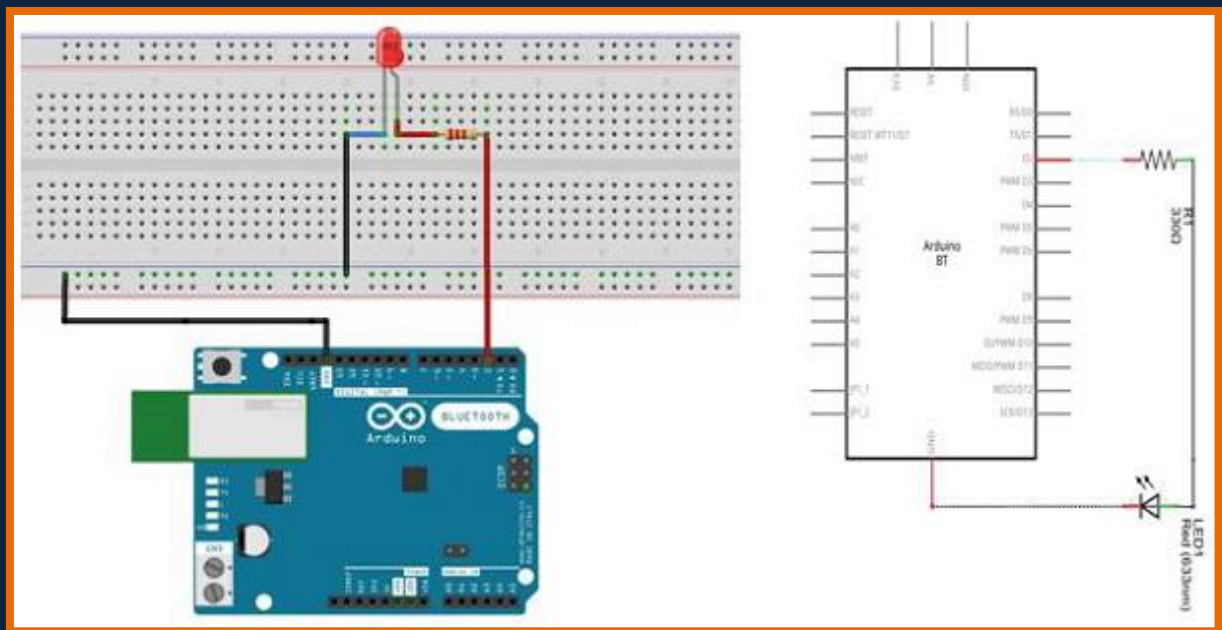
Componentes Necessários

Você precisará dos seguintes componentes -

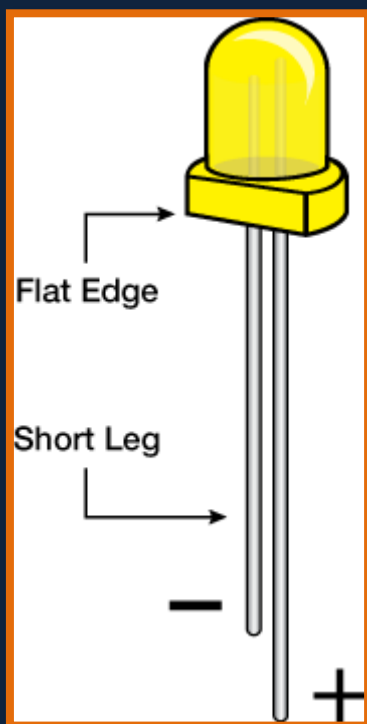
- 1 × Breadboard
- 1 × Arduino Uno R3
- 1 × LED
- Resistor 1 × 330Ω
- 2 × Jumper

Procedimento

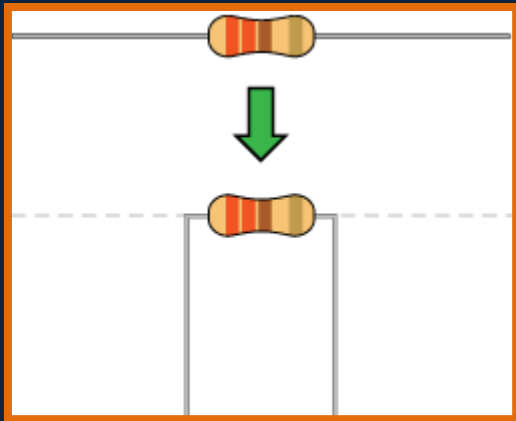
Siga o diagrama do circuito e conecte os componentes na tábua de pão, como mostra a imagem abaixo.



Nota - Para descobrir a polaridade de um LED, observe-o de perto. A mais curta das duas pernas, em direção à borda plana da lâmpada, indica o terminal negativo.

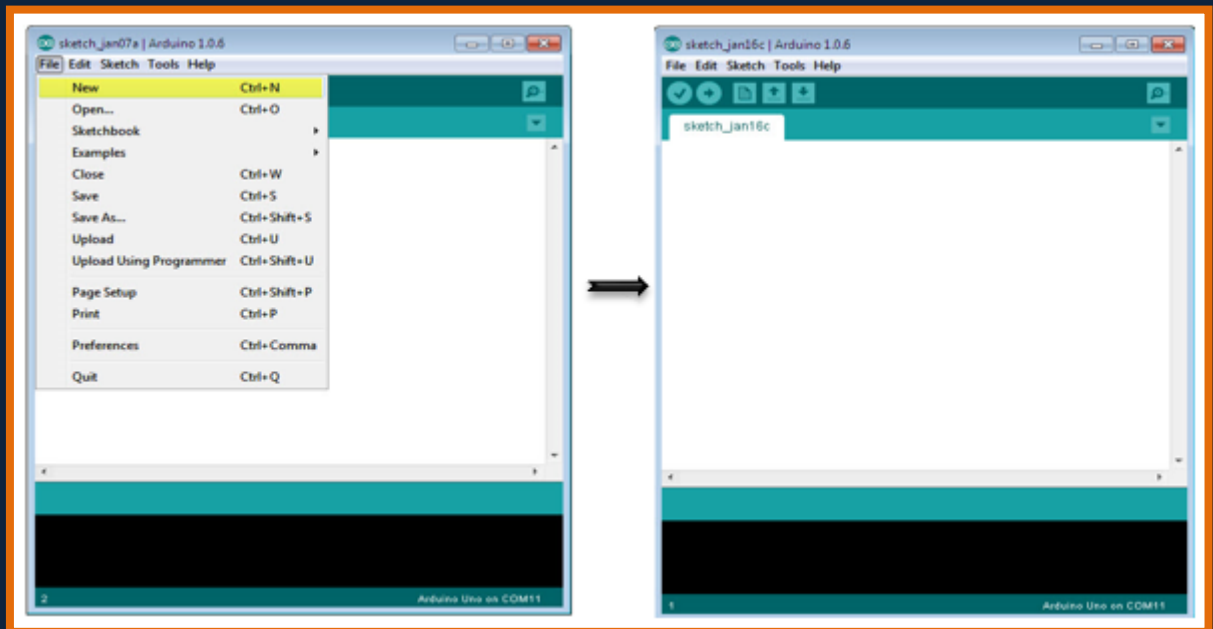


Componentes como resistores precisam ter seus terminais dobrados em ângulos de 90 ° para encaixar corretamente os soquetes da placa de ensaio. Você também pode cortar os terminais mais curtos.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra o novo arquivo de esboço clicando em Novo.



Código Arduino

```

/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
*/

// the setup function runs once when you press reset or power the board

void setup() { // initialize digital pin 13 as an output.
  pinMode(2, OUTPUT);
}

```

```
// the loop function runs over and over again forever

void loop() {
  digitalWrite(2, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(2, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

Código a Nota

pinMode (2, OUTPUT) - Antes de poder usar um dos pinos do Arduino, você precisa informar ao Arduino Uno R3 se é um INPUT ou OUTPUT. Usamos uma "função" interna chamada pinMode () para fazer isso.

digitalWrite (2, HIGH) - Quando você está usando um pino como uma SAÍDA, pode comandá-lo como ALTO (saída 5 volts) ou BAIXO (saída 0 volts).

Resultado

Você deve ver o LED acender e apagar. Se a saída necessária não for vista, verifique se você montou o circuito corretamente e verifique e carregou o código na sua placa.

Arduino - LED de desvanecimento

Este exemplo demonstra o uso da função analogWrite () para apagar um LED. O AnalogWrite usa modulação por largura de pulso (PWM), ativando e desativando um pino digital muito rapidamente com diferentes proporções entre ativado e desativado, para criar um efeito de desbotamento.

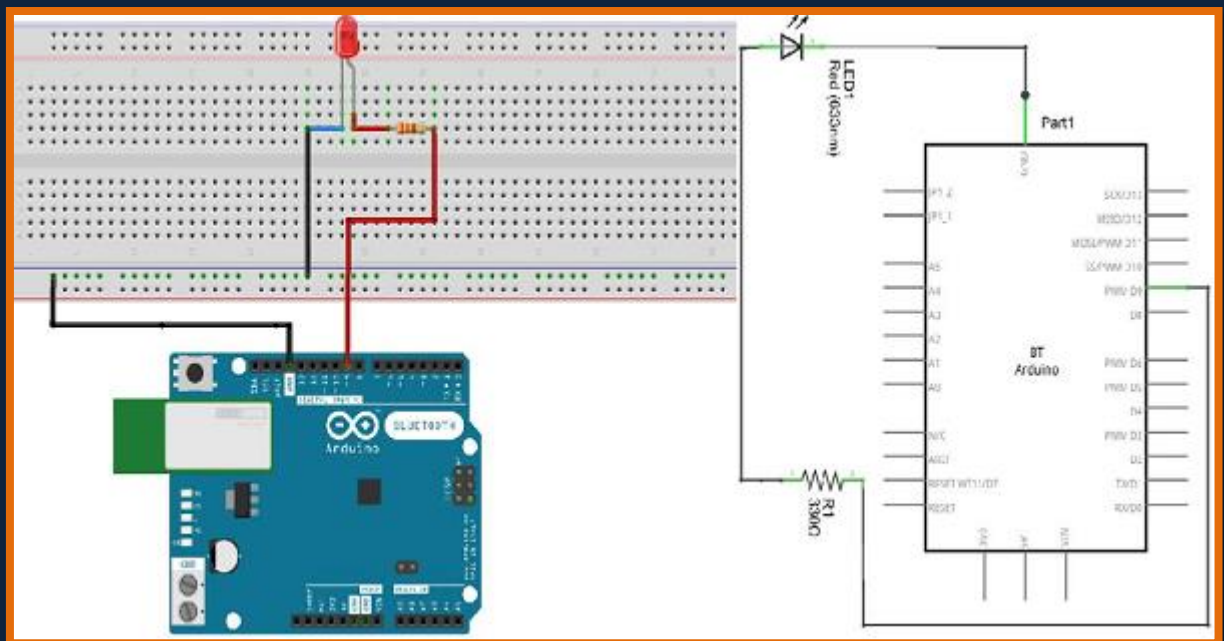
Componentes Necessários

Você precisará dos seguintes componentes -

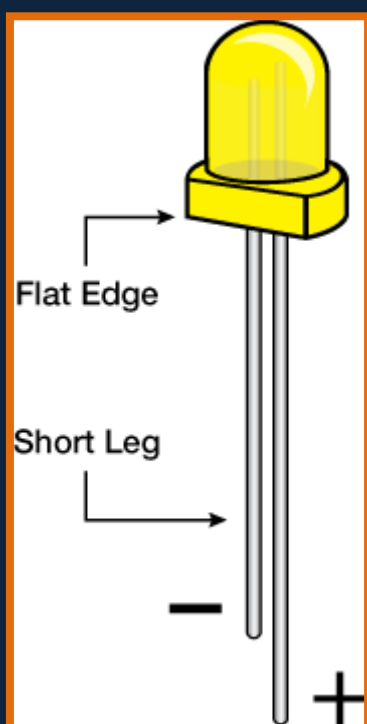
- 1 × Breadboard
- 1 × Arduino Uno R3
- 1 × LED
- Resistor 1 × 330Ω
- 2 × Jumper

Procedimento

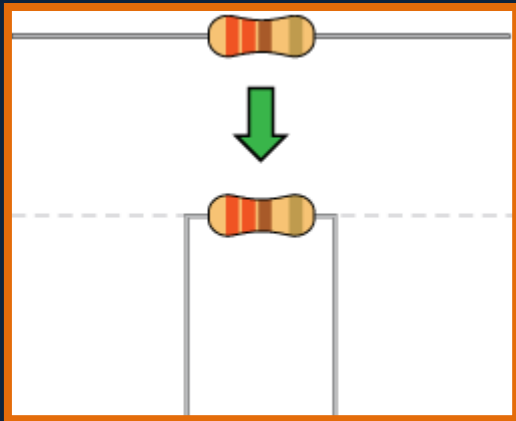
Siga o diagrama do circuito e conecte os componentes na tábua de pão, como mostra a imagem abaixo.



Nota - Para descobrir a polaridade de um LED, observe-o de perto. A mais curta das duas pernas, em direção à borda plana da lâmpada, indica o terminal negativo.

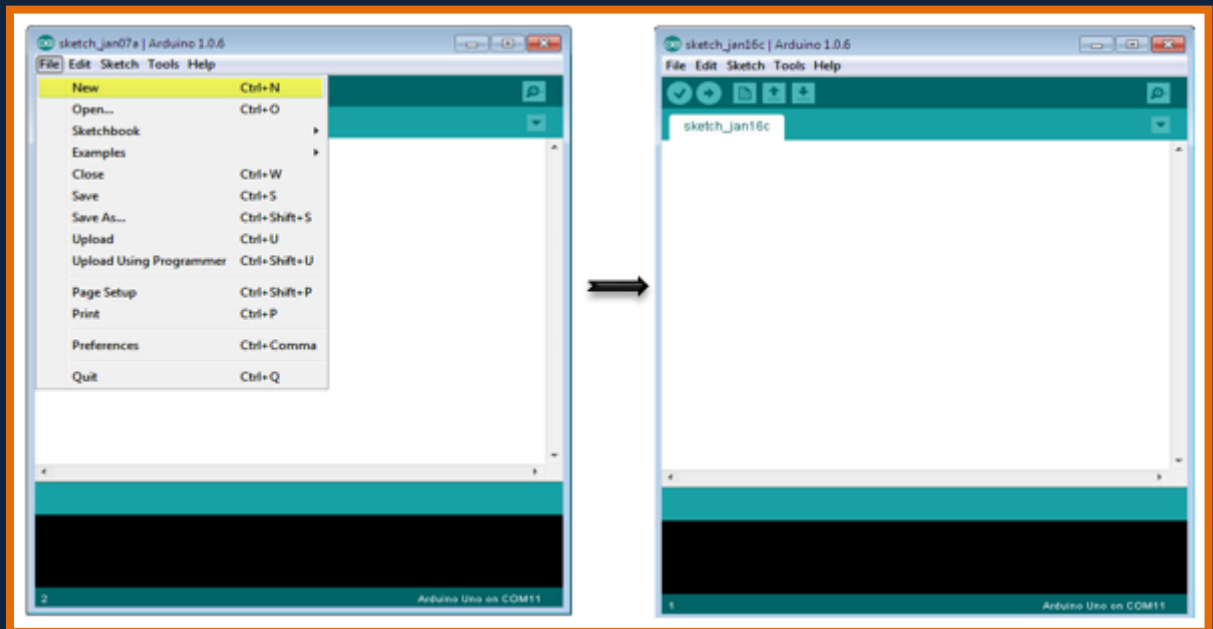


Componentes como resistores precisam ter seus terminais dobrados em ângulos de 90° para encaixar corretamente os soquetes da placa de ensaio. Você também pode cortar os terminais mais curtos.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra o novo arquivo de esboço clicando em Novo.



Código Arduino

```

/*
  Fade
  This example shows how to fade an LED on pin 9 using the analogWrite() function.

  The analogWrite() function uses PWM, so if you want to change the pin you're using,
  be
  sure to use another PWM capable pin. On most Arduino, the PWM pins are identified
  with
  a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11.
  */

```

```

int led = 9; // the PWM pin the LED is attached to
int brightness = 0; // how bright the LED is
int fadeAmount = 5; // how many points to fade the LED by
// the setup routine runs once when you press reset:

void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:

void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);
  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;
  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(300);
}

```

Código a Nota

Depois de declarar o pino 9 como seu pino de LED, não há nada a fazer na função `setup ()` do seu código. A função `analogWrite ()` que você usará no loop principal do seu código requer dois argumentos: um, informando a função em que pino gravar e o outro indicando qual valor PWM a ser gravado.

Para apagar e acender o LED, aumente gradualmente os valores de PWM de 0 (até o fim) para 255 (até o fim) e depois volte para 0 para concluir o ciclo. No esboço fornecido acima, o valor PWM é definido usando uma variável chamada **brilho**. Cada vez que passa pelo loop, ele aumenta pelo valor da variável **fadeAmount**.

Se o brilho estiver no extremo de seu valor (0 ou 255), `fadeAmount` será alterado para negativo. Em outras palavras, se `fadeAmount` for 5, será definido como -5. Se for -5, será definido como 5. Na próxima vez que ocorrer um loop, essa alteração fará com que o brilho mude também a direção.

analogWrite () pode alterar o valor PWM muito rapidamente, portanto o atraso no final do esboço controla a velocidade do desbotamento. Tente alterar o valor do atraso e veja como ele altera o efeito de desbotamento.

Resultado

Você deve ver o brilho do LED mudar gradualmente.

Arduino - Leitura de tensão analógica

Este exemplo mostra como ler uma entrada analógica no pino analógico 0. A entrada é convertida de `analogRead ()` em voltagem e impressa no monitor serial do Arduino Software (IDE).

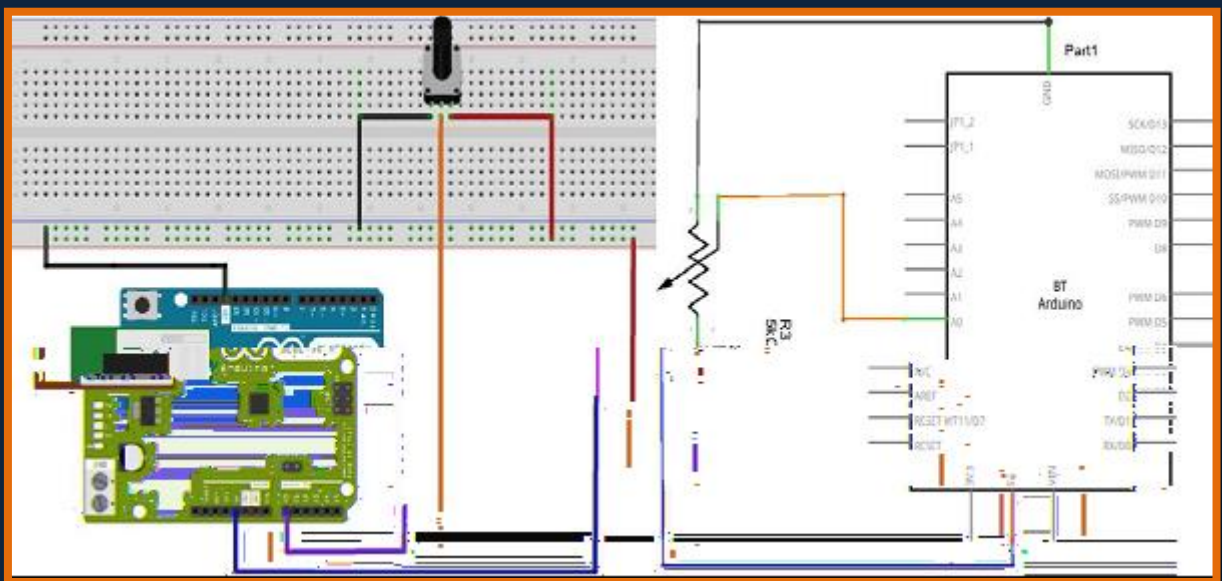
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × Breadboard
- 1 × Arduino Uno R3
- Resistor variável de 1 × 5K (potenciômetro)
- 2 × Jumper

Procedimento

Siga o diagrama do circuito e conecte os componentes na tábua de pão, como mostra a imagem abaixo.

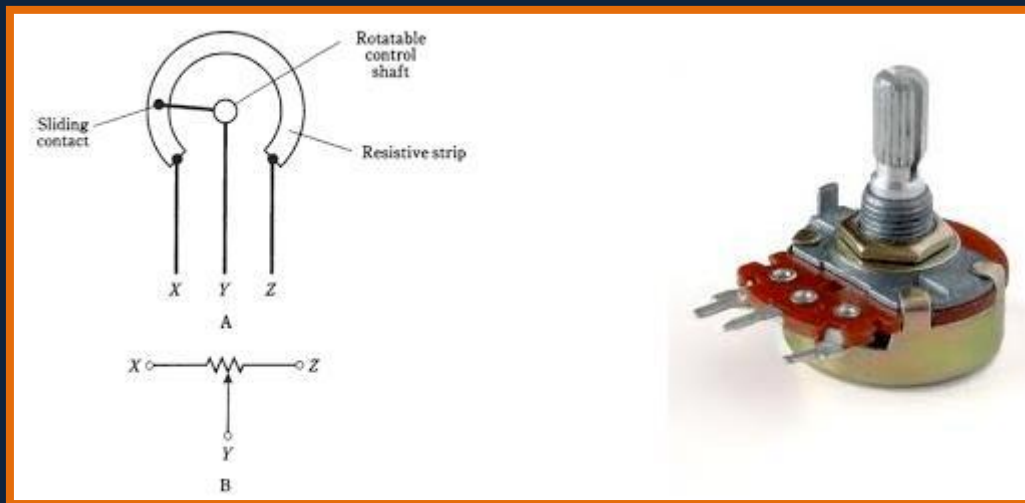


Potenciômetro

Um potenciômetro (ou pot) é um transdutor eletromecânico simples. Ele converte o movimento rotativo ou linear do operador de entrada em uma mudança de resistência. Essa alteração é (ou pode ser) usada para controlar qualquer coisa, desde o volume de um sistema de alta fidelidade até a direção de um enorme navio porta-contêineres.

O pote como o conhecemos era originalmente conhecido como reostato (essencialmente um resistor variável de fio enrolado). A variedade de pots disponíveis agora é bastante surpreendente, e pode ser muito difícil para o iniciante (em particular)

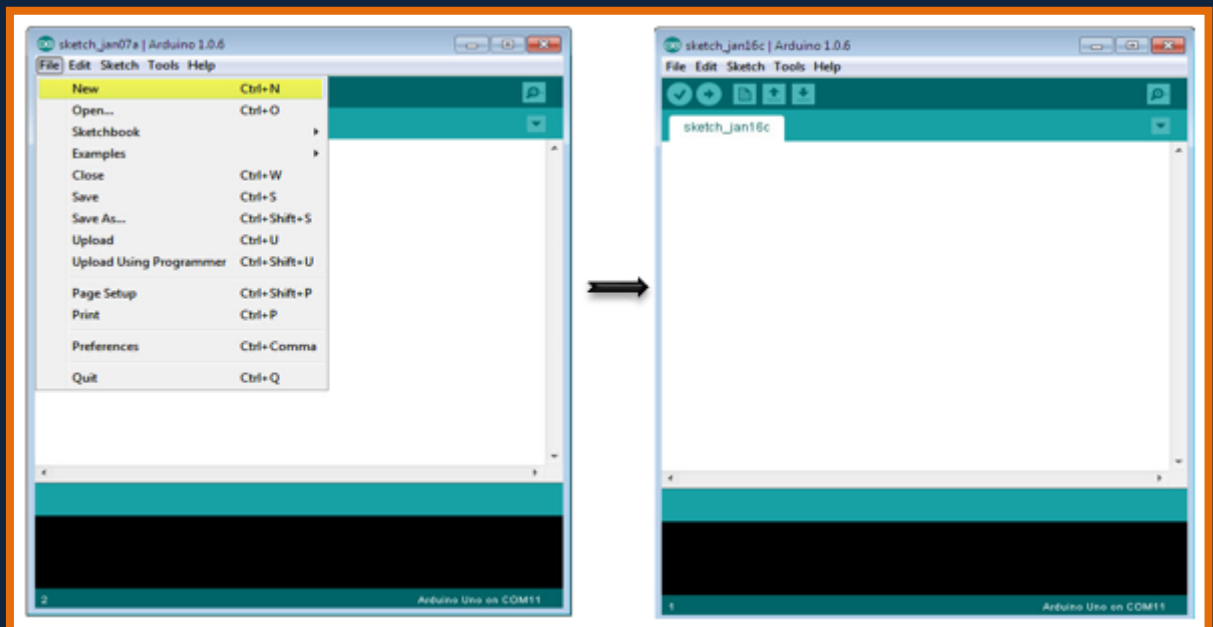
descobrir qual tipo é adequado para uma determinada tarefa. Alguns tipos diferentes de pots, que podem ser usados para a mesma tarefa, dificultam o trabalho.



A imagem à esquerda mostra o símbolo esquemático padrão de um pote. A imagem à direita é o potenciômetro.

Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

```
/*  
ReadAnalogVoltage  
Reads an analog input on pin 0, converts it to voltage,
```

```

and prints the result to the serial monitor.
Graphical representation is available using serial plotter (Tools > Serial Plotter menu)
Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and
ground.
*/

// the setup routine runs once when you press reset:

void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:

void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}

```

Código a Nota

No programa ou esboço fornecido abaixo, a primeira coisa que você faz na função de configuração é iniciar as comunicações seriais, a 9600 bits por segundo, entre a placa e o computador com a linha -

```
Serial.begin(9600);
```

No loop principal do seu código, você precisa estabelecer uma variável para armazenar o valor da resistência (que será entre 0 e 1023, perfeito para um tipo de dados int) proveniente do seu potenciômetro -

```
int sensorValue = analogRead(A0);
```

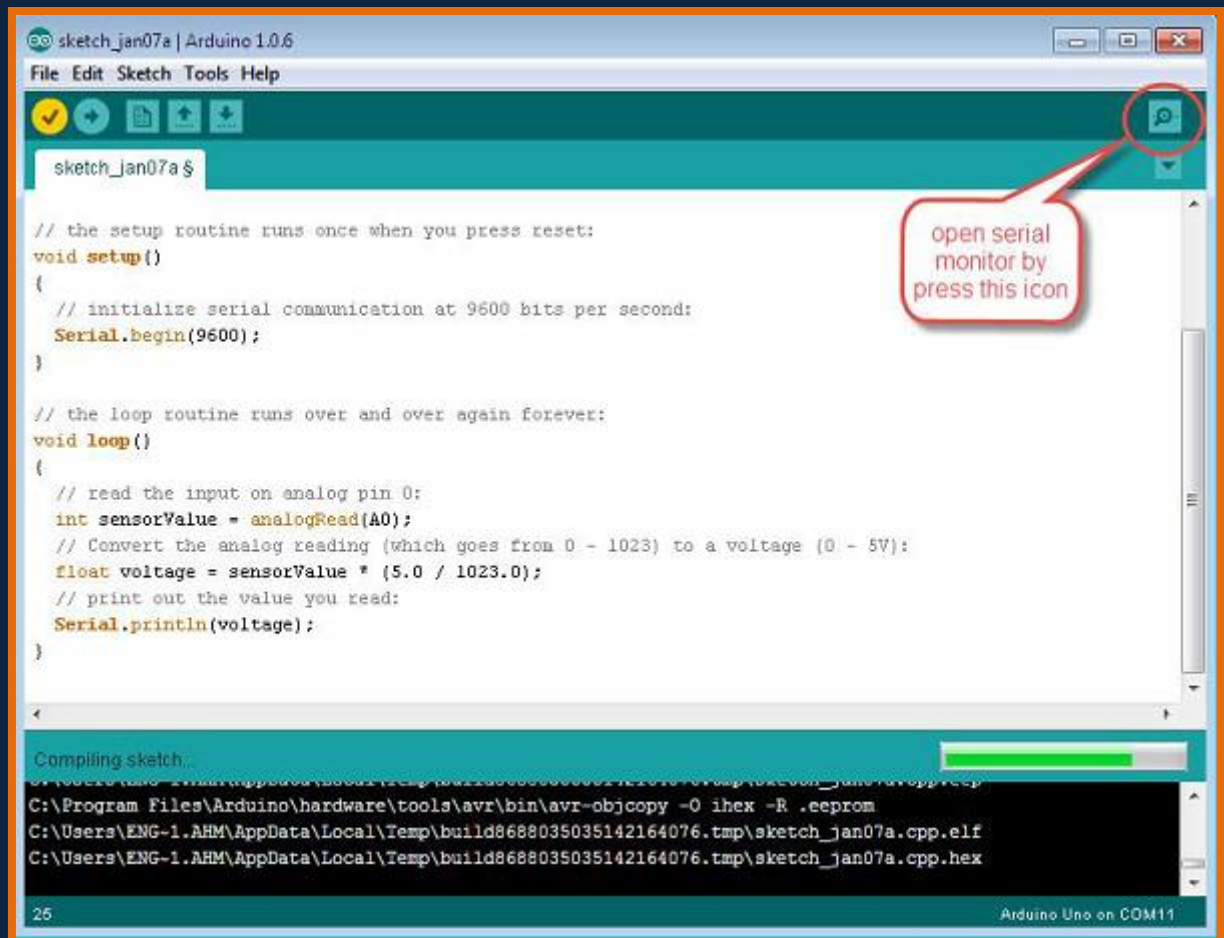
Para alterar os valores de 0 a 1023 para uma faixa que corresponde à tensão, o pino está lendo, você precisa criar outra variável, um flutuador e fazer um pequeno cálculo. Para dimensionar os números entre 0,0 e 5,0, divida 5,0 por 1023,0 e multiplique por sensorValue -

```
float voltage= sensorValue * (5.0 / 1023.0);
```

Finalmente, você precisa imprimir essas informações na sua janela serial. Você pode fazer isso com o comando Serial.println () na sua última linha de código -

```
Serial.println(voltage)
```

Agora, abra o Serial Monitor no Arduino IDE clicando no ícone no lado direito da barra verde superior ou pressionando Ctrl + Shift + M.



Resultado

Você verá um fluxo constante de números variando de 0,0 a 5,0. À medida que você gira o pote, os valores mudam, correspondendo à tensão no pino A0.

Arduino - Gráfico de barras do LED

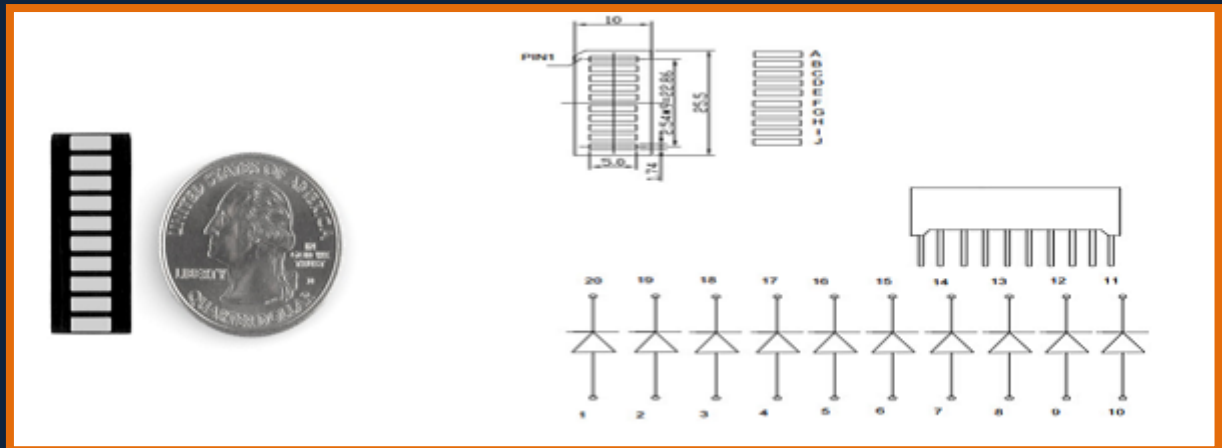
Este exemplo mostra como ler uma entrada analógica no pino analógico 0, converter os valores de `analogRead ()` em tensão e imprimi-la no monitor serial do Arduino Software (IDE).

Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × Breadboard
- 1 × Arduino Uno R3
- Resistor variável de 1 × 5k ohm (potenciômetro)
- 2 × Jumper

Gráfico de barras do LED de 10 segmentos



Esses LEDs de gráfico de barras de 10 segmentos têm muitos usos. Com um tamanho compacto, conexão simples, eles são fáceis para protótipos ou produtos acabados. Essencialmente, são 10 LEDs azuis individuais alojados juntos, cada um com uma conexão individual de ânodo e cátodo.

Eles também estão disponíveis nas cores amarelo, vermelho e verde.

Nota - Os pinos desses gráficos de barra podem variar do que está listado na folha de dados. Girar o dispositivo 180 graus corrigirá a alteração, tornando o pino 11 o primeiro pino da linha.

Código Arduino

```
/*  
  LED bar graph  
  Turns on a series of LEDs based on the value of an analog sensor.  
  This is a simple way to make a bar graph display.  
  Though this graph uses 8 LEDs, you can use any number by  
  changing the LED count and the pins in the array.  
  This method can be used to control any series of digital  
  outputs that depends on an analog input.  
*/  
  
// these constants won't change:  
const int analogPin = A0; // the pin that the potentiometer is attached to  
const int ledCount = 8; // the number of LEDs in the bar graph  
int ledPins[] = {2, 3, 4, 5, 6, 7, 8, 9}; // an array of pin numbers to which LEDs are  
attached  
  
void setup() {  
  // loop over the pin array and set them all to output:  
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {  
    pinMode(ledPins[thisLed], OUTPUT);  
  }  
}
```

```

void loop() {
  // read the potentiometer:
  int sensorReading = analogRead(analogPin);
  // map the result to a range from 0 to the number of LEDs:
  int ledLevel = map(sensorReading, 0, 1023, 0, ledCount);
  // loop over the LED array:
  for (int thisLed = 0; thisLed < ledCount; thisLed++) {
    // if the array element's index is less than ledLevel,
    // turn the pin for this element on:
    if (thisLed < ledLevel) {
      digitalWrite(ledPins[thisLed], HIGH);
    } else { // turn off all pins higher than the ledLevel:
      digitalWrite(ledPins[thisLed], LOW);
    }
  }
}
}

```

Código a Nota

O esboço funciona da seguinte maneira: primeiro, você lê a entrada. Você mapeia o valor de entrada para o intervalo de saída, neste caso, dez LEDs. Em seguida, você configura um **loop for** para iterar sobre as saídas. Se o número da saída na série for menor que o intervalo de entrada mapeado, você o ativará. Caso contrário, desligue-o.

Resultado

Você verá o LED acender um a um quando o valor da leitura analógica aumentar e desligar um a um enquanto a leitura estiver diminuindo.

Arduino - Logout do teclado

Este exemplo usa a biblioteca do teclado para fazer o logout da sua sessão do usuário no computador quando o pino 2 no ARDUINO UNO é puxado para o chão. O esboço simula o pressionamento de teclas em sequência de duas ou três teclas ao mesmo tempo e, após um pequeno atraso, as libera.

Aviso - Quando você usa o comando **Keyboard.print ()**, o Arduino assume o teclado do seu computador. Para garantir que você não perca o controle do seu computador enquanto executa um esboço com esta função, configure um sistema de controle confiável antes de ligar para **Keyboard.print ()**. Este esboço foi desenvolvido para enviar apenas um comando do teclado depois que um pino foi puxado para o chão.

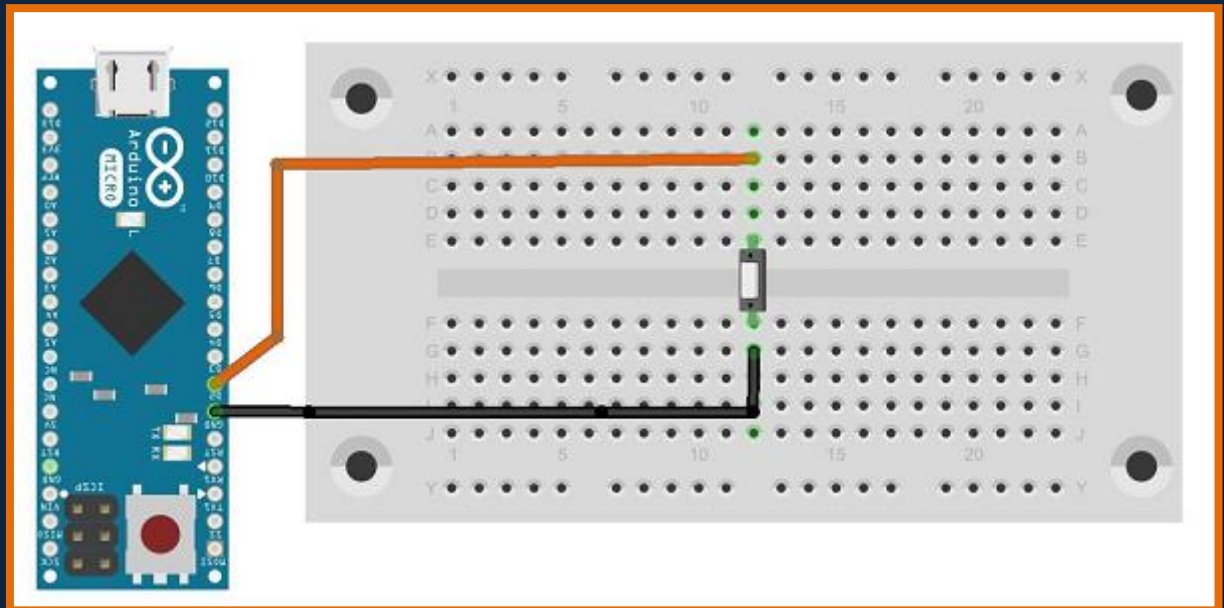
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × Breadboard
- 1 × placa Leonardo, Micro ou Due Arduino
- 1 × botão
- 1 × jumper

Procedimento

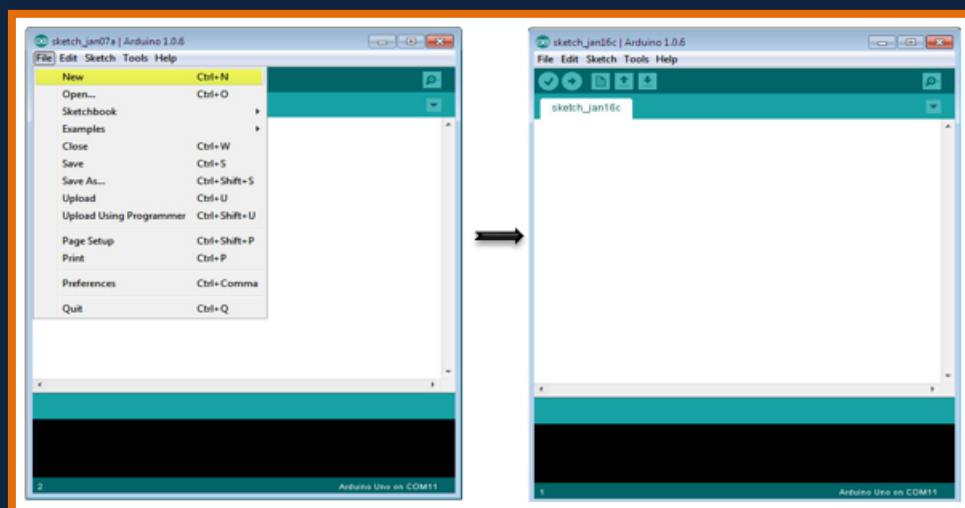
Siga o diagrama do circuito e conecte os componentes na tábua de pão, como mostra a imagem abaixo.



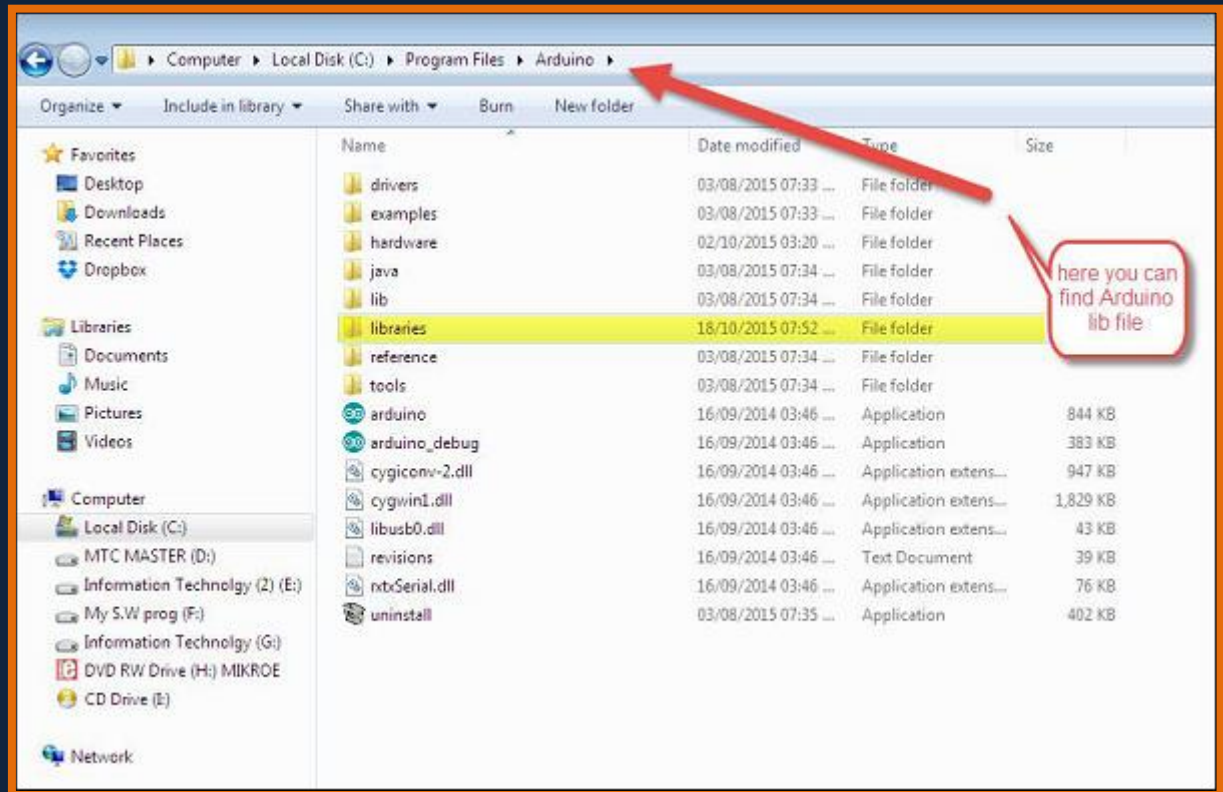
Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.

Neste exemplo, você precisa usar o Arduino IDE 1.6.7



Nota - Você deve incluir a biblioteca do teclado no seu arquivo de biblioteca do Arduino. Copie e cole o arquivo da biblioteca do teclado dentro do arquivo com as bibliotecas de nomes (destacadas), conforme mostrado na captura de tela a seguir.



Código Arduino

```

/*
  Keyboard logout
  This sketch demonstrates the Keyboard library.
  When you connect pin 2 to ground, it performs a logout.
  It uses keyboard combinations to do this, as follows:
  On Windows, CTRL-ALT-DEL followed by ALT-I
  On Ubuntu, CTRL-ALT-DEL, and ENTER
  On OSX, CMD-SHIFT-q
  To wake: Spacebar.
  Circuit:
  * Arduino Leonardo or Micro
  * wire to connect D2 to ground.
*/

#define OSX 0
#define WINDOWS 1
#define UBUNTU 2

#include "Keyboard.h"

// change this to match your platform:

```

```

int platform = WINDOWS;

void setup() {
  // make pin 2 an input and turn on the
  // pullup resistor so it goes high unless
  // connected to ground:

  pinMode(2, INPUT_PULLUP);
  Keyboard.begin();
}

void loop() {
  while (digitalRead(2) == HIGH) {
    // do nothing until pin 2 goes low
    delay(500);
  }

  delay(1000);

  switch (platform) {
    case OSX:
      Keyboard.press(KEY_LEFT_GUI);

      // Shift-Q logs out:
      Keyboard.press(KEY_LEFT_SHIFT);
      Keyboard.press('Q');
      delay(100);

      // enter:
      Keyboard.write(KEY_RETURN);
      break;

    case WINDOWS:
      // CTRL-ALT-DEL:
      Keyboard.press(KEY_LEFT_CTRL);
      Keyboard.press(KEY_LEFT_ALT);
      Keyboard.press(KEY_DELETE);
      delay(100);
      Keyboard.releaseAll();

      //ALT-I:
      delay(2000);
      Keyboard.press(KEY_LEFT_ALT);
      Keyboard.press('I');
      Keyboard.releaseAll();
      break;

    case UBUNTU:
      // CTRL-ALT-DEL:
      Keyboard.press(KEY_LEFT_CTRL);

```

```

Keyboard.press(KEY_LEFT_ALT);
Keyboard.press(KEY_DELETE);

delay(1000);
Keyboard.releaseAll();

// Enter to confirm logout:
Keyboard.write(KEY_RETURN);
break;
}

// do nothing:
while (true);
}

Keyboard.releaseAll();

// enter:
Keyboard.write(KEY_RETURN);
break;
case WINDOWS:

// CTRL-ALT-DEL:
Keyboard.press(KEY_LEFT_CTRL);
Keyboard.press(KEY_LEFT_ALT);
Keyboard.press(KEY_DELETE);
delay(100);
Keyboard.releaseAll();

//ALT-l:
delay(2000);
Keyboard.press(KEY_LEFT_ALT);
Keyboard.press('l');
Keyboard.releaseAll();
break;

case UBUNTU:
// CTRL-ALT-DEL:
Keyboard.press(KEY_LEFT_CTRL);
Keyboard.press(KEY_LEFT_ALT);
Keyboard.press(KEY_DELETE);
delay(1000);
Keyboard.releaseAll();

// Enter to confirm logout:
Keyboard.write(KEY_RETURN);
break;
}

// do nothing:

```

```
while (true);  
}
```

Código a Nota

Antes de carregar o programa em sua placa, certifique-se de atribuir o sistema operacional correto que você está usando atualmente à variável de plataforma.

Enquanto o esboço estiver em execução, pressionar o botão conectará o pino 2 ao chão e a placa enviará a sequência de logout ao PC conectado via USB.

Resultado

Quando você conecta o pino 2 ao terra, ele executa uma operação de logout.

Ele usa as seguintes combinações de teclado para fazer logout -

- No **Windows** , CTRL-ALT-DEL seguido por ALT-I
- No **Ubuntu** , CTRL-ALT-DEL e ENTER
- No **OSX** , CMD-SHIFT-q

Arduino - Mensagem do Teclado

Neste exemplo, quando o botão é pressionado, uma sequência de texto é enviada ao computador como entrada do teclado. A string informa o número de vezes que o botão é pressionado. Depois de programar e instalar o Leonardo, abra seu editor de texto favorito para ver os resultados.

Aviso - Quando você usa o comando **Keyboard.print ()** , o Arduino assume o teclado do seu computador. Para garantir que você não perca o controle do seu computador enquanto executa um esboço com esta função, configure um sistema de controle confiável antes de ligar para **Keyboard.print ()** . Este esboço inclui um botão para alternar o teclado, de modo que só seja executado depois que o botão for pressionado.

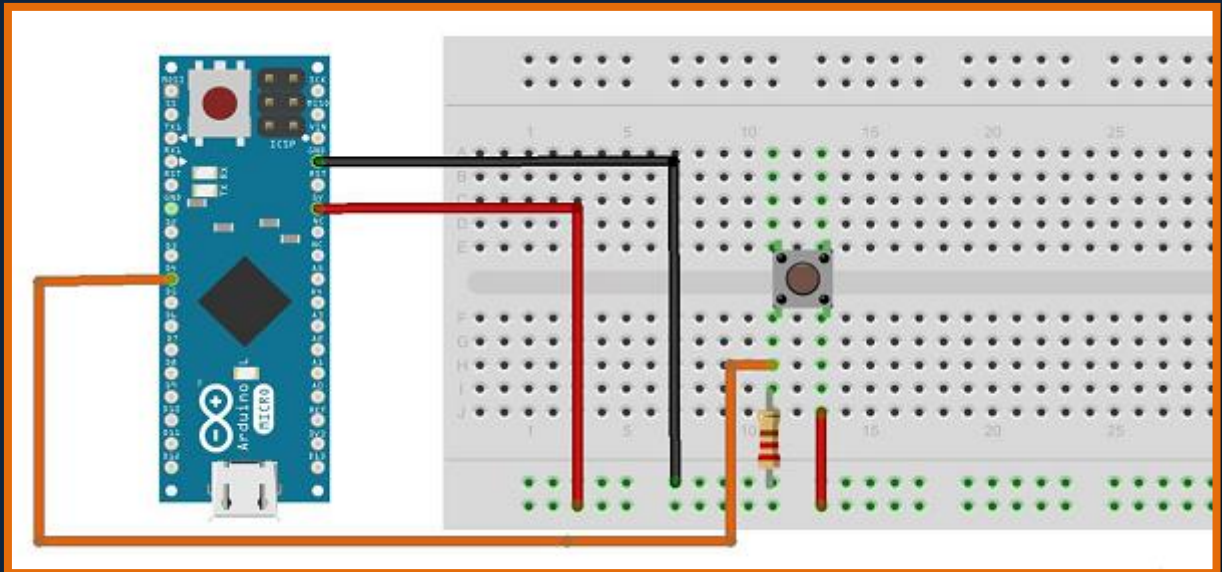
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × Breadboard
- 1 × placa Leonardo, Micro ou Due Arduino
- 1 × botão momentâneo
- Resistor de 1 × 10k ohm

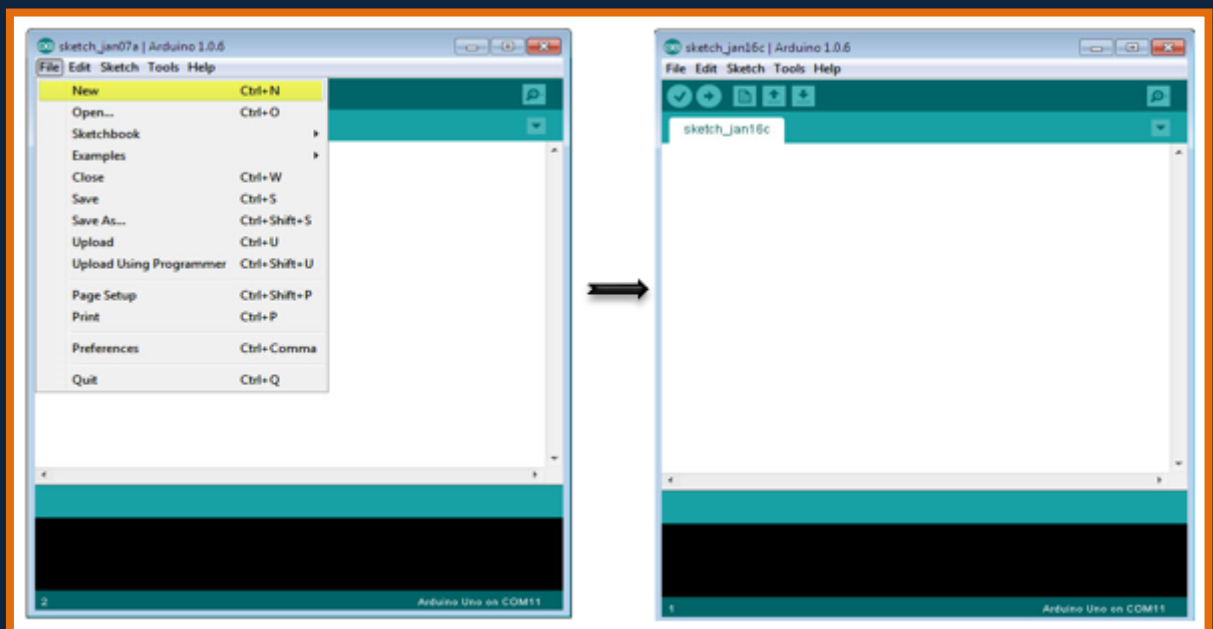
Procedimento

Siga o diagrama do circuito e conecte os componentes na tábua de pão, como mostra a imagem abaixo.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

```
/*
```

```

Keyboard Message test For the Arduino Leonardo and Micro,
  Sends a text string when a button is pressed.
The circuit:
* pushbutton attached from pin 4 to +5V
* 10-kilohm resistor attached from pin 4 to ground
*/

#include "Keyboard.h"
const int buttonPin = 4; // input pin for pushbutton
int previousButtonState = HIGH; // for checking the state of a pushButton
int counter = 0; // button push counter

void setup() {
  pinMode(buttonPin, INPUT); // make the pushButton pin an input:
  Keyboard.begin(); // initialize control over the keyboard:
}

void loop() {
  int buttonState = digitalRead(buttonPin); // read the pushbutton:
  if ((buttonState != previousButtonState)&& (buttonState == HIGH)) // and it's
currently pressed: {
    // increment the button counter
    counter++;
    // type out a message
    Keyboard.print("You pressed the button ");
    Keyboard.print(counter);
    Keyboard.println(" times.");
  }
  // save the current button state for comparison next time:
  previousButtonState = buttonState;
}

```

Código a Nota

Conecte um terminal do botão ao pino 4 no Arduino. Conecte o outro pino a 5V. Use o resistor como um pull-down, fornecendo uma referência ao terra, conectando-o do pino 4 ao solo.

Depois de programar sua placa, desconecte o cabo USB, abra um editor de texto e coloque o cursor na área de digitação. Conecte a placa ao seu computador via USB novamente e pressione o botão para escrever no documento.

Resultado

Usando qualquer editor de texto, ele exibirá o texto enviado via Arduino.

Arduino - Controle de botão do mouse

Usando a biblioteca Mouse, você pode controlar o cursor na tela de um computador com um Arduino Leonardo, Micro ou Due.

Este exemplo em particular usa cinco botões para mover o cursor na tela. Quatro dos botões são direcionais (cima, baixo, esquerda, direita) e um é para um clique esquerdo do mouse. O movimento do cursor do Arduino é sempre relativo. Sempre que uma entrada é lida, a posição do cursor é atualizada em relação à sua posição atual.

Sempre que um dos botões direcionais é pressionado, o Arduino move o mouse, mapeando uma entrada HIGH para um intervalo de 5 na direção apropriada.

O quinto botão é para controlar um clique esquerdo do mouse. Quando o botão é liberado, o computador reconhece o evento.

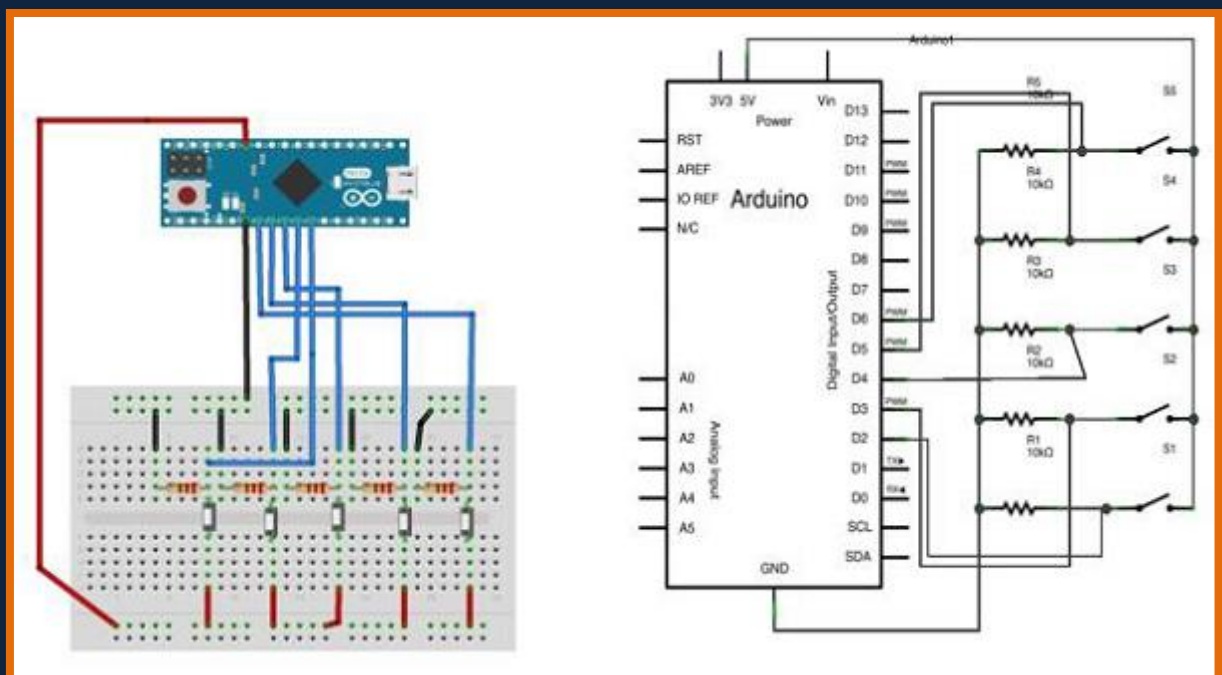
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × Breadboard
- 1 × Arduino Leonardo, placa Micro ou Due
- Resistor de 5 × 10k ohm
- 5 × botões momentâneos

Procedimento

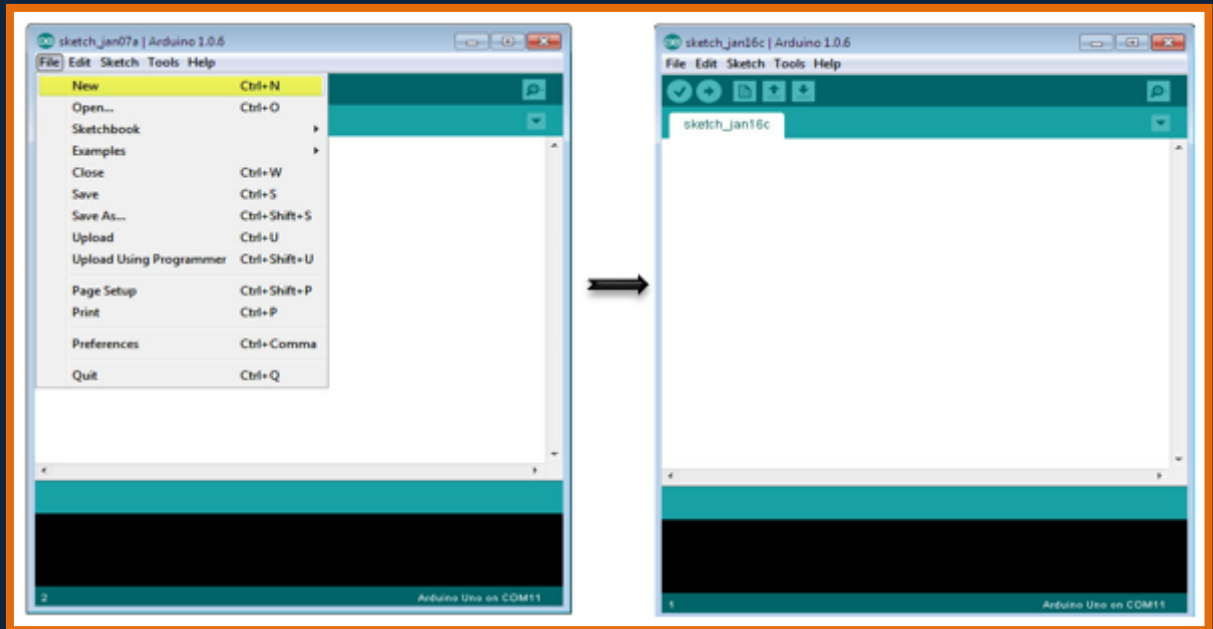
Siga o diagrama do circuito e conecte os componentes na tábua de pão, como mostra a imagem abaixo.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.

Neste exemplo, você precisa usar o Arduino IDE 1.6.7



Código Arduino

```
/*  
  Button Mouse Control  
  For Leonardo and Due boards only .Controls the mouse from  
  five pushbuttons on an Arduino Leonardo, Micro or Due.  
  Hardware:  
  * 5 pushbuttons attached to D2, D3, D4, D5, D6  
  The mouse movement is always relative. This sketch reads  
  four pushbuttons, and uses them to set the movement of the mouse.  
  WARNING: When you use the Mouse.move() command, the Arduino takes  
  over your mouse! Make sure you have control before you use the mouse commands.  
*/  
  
#include "Mouse.h"  
// set pin numbers for the five buttons:  
const int upButton = 2;  
const int downButton = 3;  
const int leftButton = 4;  
const int rightButton = 5;  
const int mouseButton = 6;  
int range = 5; // output range of X or Y movement; affects movement speed  
int responseDelay = 10; // response delay of the mouse, in ms
```



```

void setup() {
  // initialize the buttons' inputs:
  pinMode(upButton, INPUT);
  pinMode(downButton, INPUT);
  pinMode(leftButton, INPUT);
  pinMode(rightButton, INPUT);
  pinMode(mouseButton, INPUT);
  // initialize mouse control:
  Mouse.begin();
}

void loop() {
  // read the buttons:
  int upState = digitalRead(upButton);
  int downState = digitalRead(downButton);
  int rightState = digitalRead(rightButton);
  int leftState = digitalRead(leftButton);
  int clickState = digitalRead(mouseButton);
  // calculate the movement distance based on the button states:
  int xDistance = (leftState - rightState) * range;
  int yDistance = (upState - downState) * range;
  // if X or Y is non-zero, move:
  if ((xDistance != 0) || (yDistance != 0)) {
    Mouse.move(xDistance, yDistance, 0);
  }

  // if the mouse button is pressed:
  if (clickState == HIGH) {
    // if the mouse is not pressed, press it:
    if (!Mouse.isPressed(MOUSE_LEFT)) {
      Mouse.press(MOUSE_LEFT);
    }
  } else { // else the mouse button is not pressed:
    // if the mouse is pressed, release it:
    if (Mouse.isPressed(MOUSE_LEFT)) {
      Mouse.release(MOUSE_LEFT);
    }
  }
  // a delay so the mouse does not move too fast:
  delay(responseDelay);
}

```

Código a Nota

Conecte sua placa ao seu computador com um cabo micro-USB. Os botões estão conectados às entradas digitais dos pinos 2 a 6. Certifique-se de usar resistores pull-down de 10k.

Arduino - Teclado Serial

Este exemplo escuta um byte vindo da porta serial. Quando recebida, a placa envia um pressionamento de tecla de volta ao computador. O pressionamento de tecla enviado é um maior do que o recebido; portanto, se você enviar um "a" do monitor serial, receberá um "b" da placa conectada ao computador. Um "1" retornará um "2" e assim por diante.

Aviso - Quando você usa o comando **Keyboard.print ()**, a placa Leonardo, Micro ou Due assume o teclado do seu computador. Para garantir que você não perca o controle do seu computador enquanto executa um esboço com esta função, configure um sistema de controle confiável antes de ligar para **Keyboard.print ()**. Este esboço foi desenvolvido para enviar apenas um comando do teclado depois que a placa recebeu um byte pela porta serial.

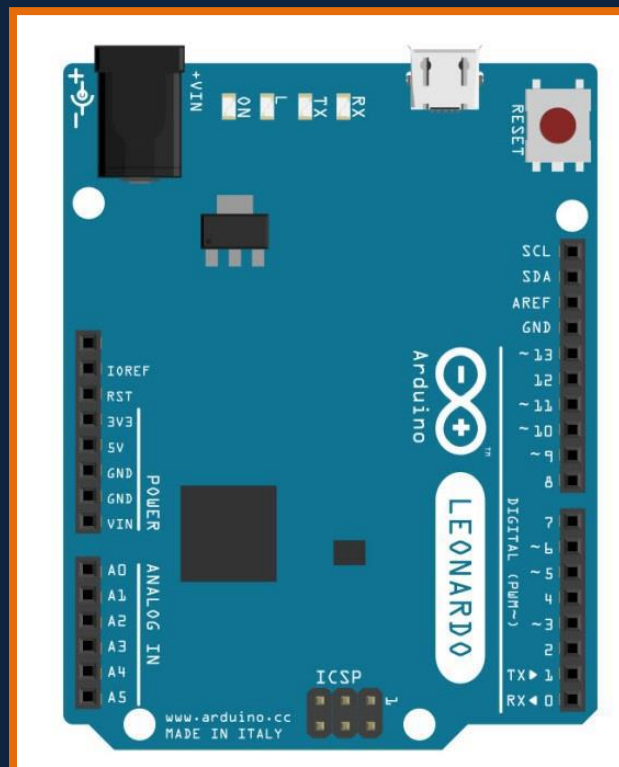
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × placa Leonardo, Micro ou Due Arduino

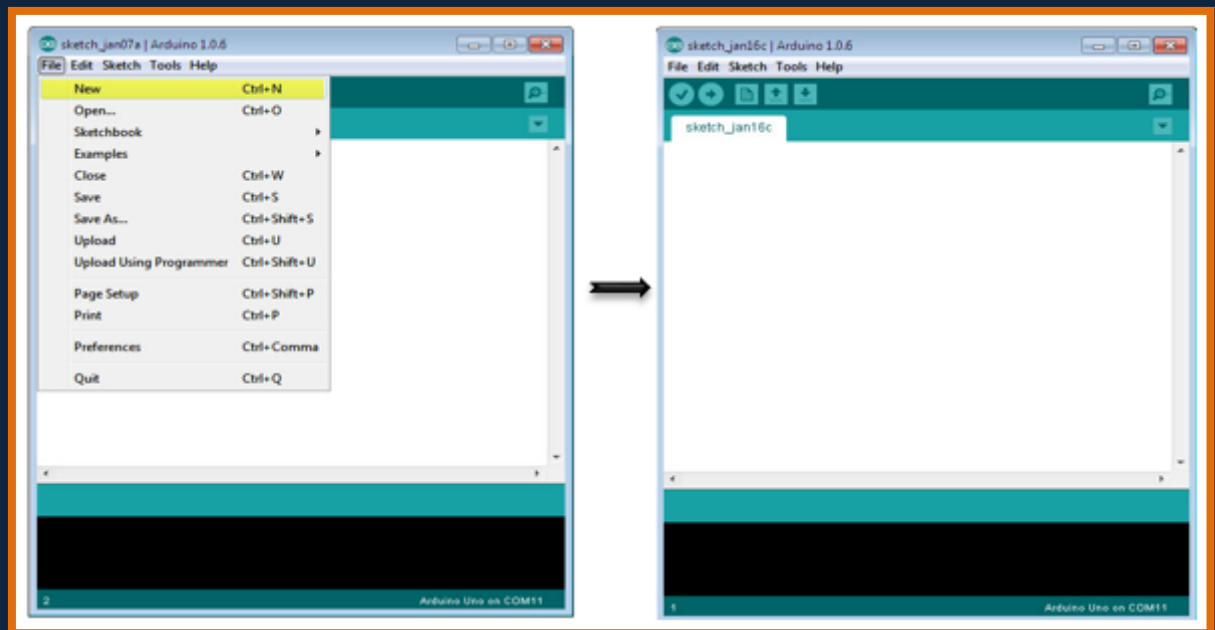
Procedimento

Basta conectar sua placa ao computador usando o cabo USB.

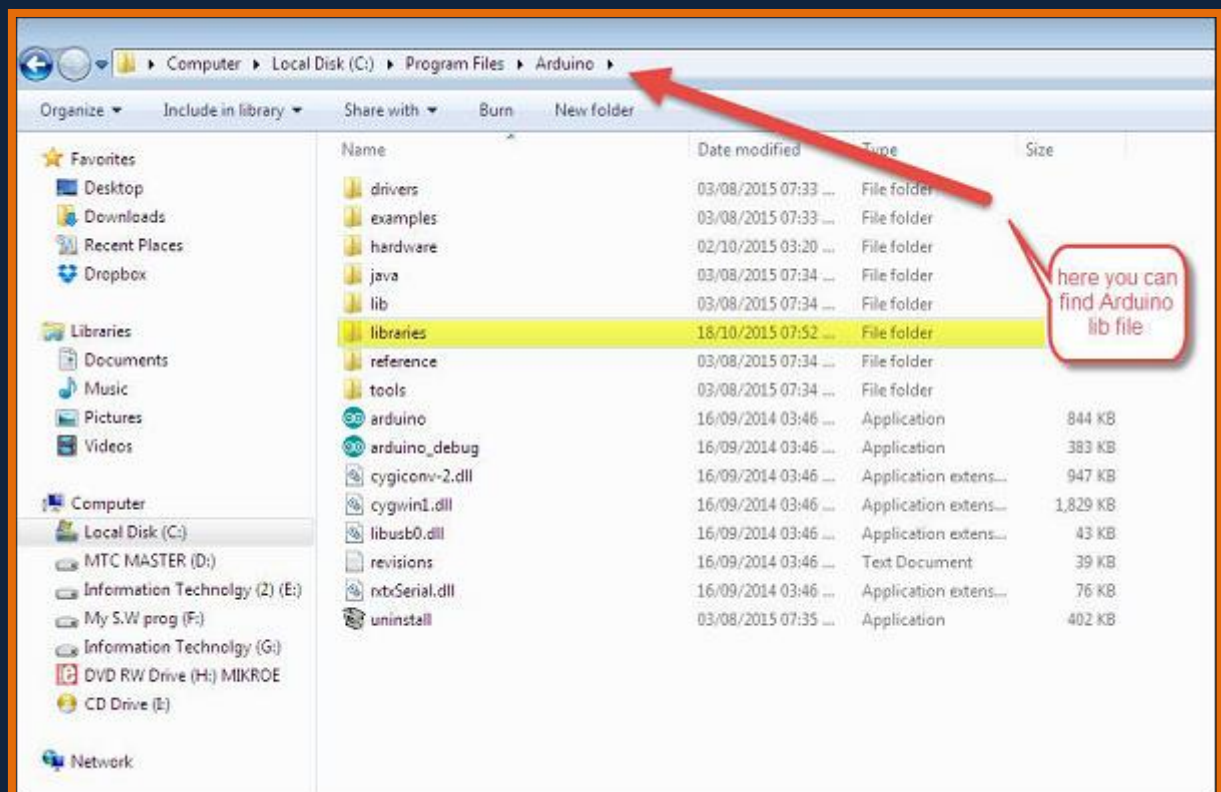


Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Notas - Você deve incluir a biblioteca do teclado no seu arquivo de biblioteca do Arduino. Copie e cole o arquivo da biblioteca do teclado dentro do arquivo com o nome 'bibliotecas' destacado em amarelo.



Código Arduino

```
/*
  Keyboard test
  For the Arduino Leonardo, Micro or Due Reads
  a byte from the serial port, sends a keystroke back.
  The sent keystroke is one higher than what's received, e.g. if you send a, you get b,
  send
  A you get B, and so forth.
  The circuit:
  * none
*/

#include "Keyboard.h"

void setup() {
  // open the serial port:
  Serial.begin(9600);
  // initialize control over the keyboard:
  Keyboard.begin();
}

void loop() {
  // check for incoming serial data:
  if (Serial.available() > 0) {
    // read incoming serial data:
    char inChar = Serial.read();
    // Type the next ASCII value from what you received:
    Keyboard.write(inChar + 1);
  }
}
```

Código a Nota

Uma vez programado, abra seu monitor serial e envie um byte. O quadro responderá com um pressionamento de tecla, que é um número maior.

Resultado

A placa responderá com um pressionamento de tecla um número mais alto no monitor serial do Arduino IDE quando você enviar um byte.

Arduino - Sensor de Umidade

Nesta seção, aprenderemos como conectar nossa placa Arduino com diferentes sensores. Vamos discutir os seguintes sensores -

- Sensor de umidade (DHT22)

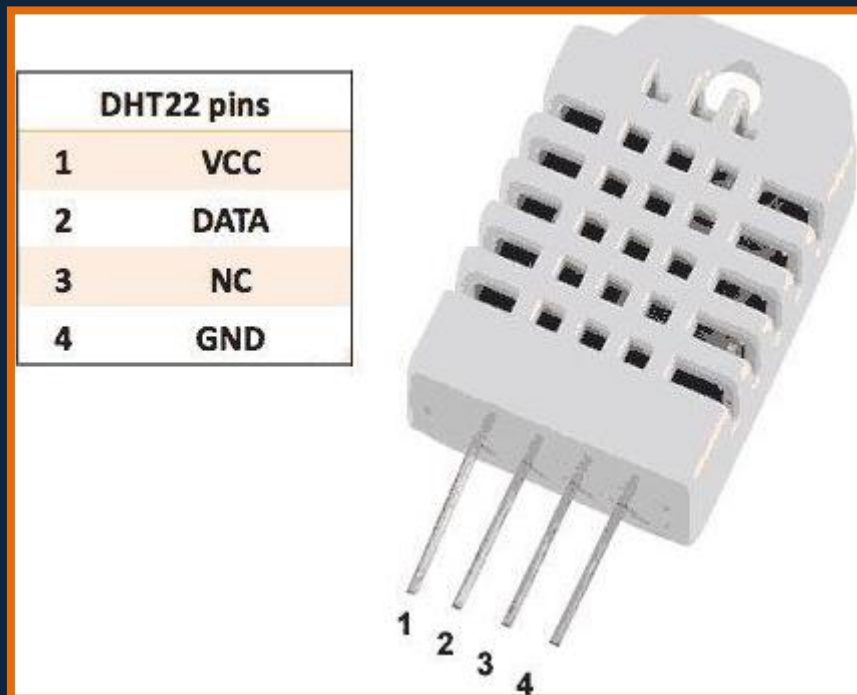
- Sensor de temperatura (LM35)
- Sensor detector de água (Simple Water Trigger)
- SENSOR PIR
- SENSOR ULTRASÔNICO
- GPS

Sensor de umidade (DHT22)

O DHT-22 (também chamado de AM2302) é um sensor de saída digital, umidade relativa e temperatura. Ele usa um sensor de umidade capacitivo e um termistor para medir o ar circundante e envia um sinal digital no pino de dados.

Neste exemplo, você aprenderá como usar esse sensor com o Arduino UNO. A temperatura e a umidade da sala serão impressas no monitor serial.

O sensor DHT-22



As conexões são simples. O primeiro pino da esquerda para a potência de 3-5V, o segundo pino do pino de entrada de dados e o pino mais à direita do solo.

Detalhes técnicos

- **Potência** - 3-5V
- **Corrente máxima** - 2.5mA
- **Umidade** - 0-100%, precisão de 2-5%
- **Temperatura** - 40 a 80 ° C, precisão de $\pm 0,5$ ° C

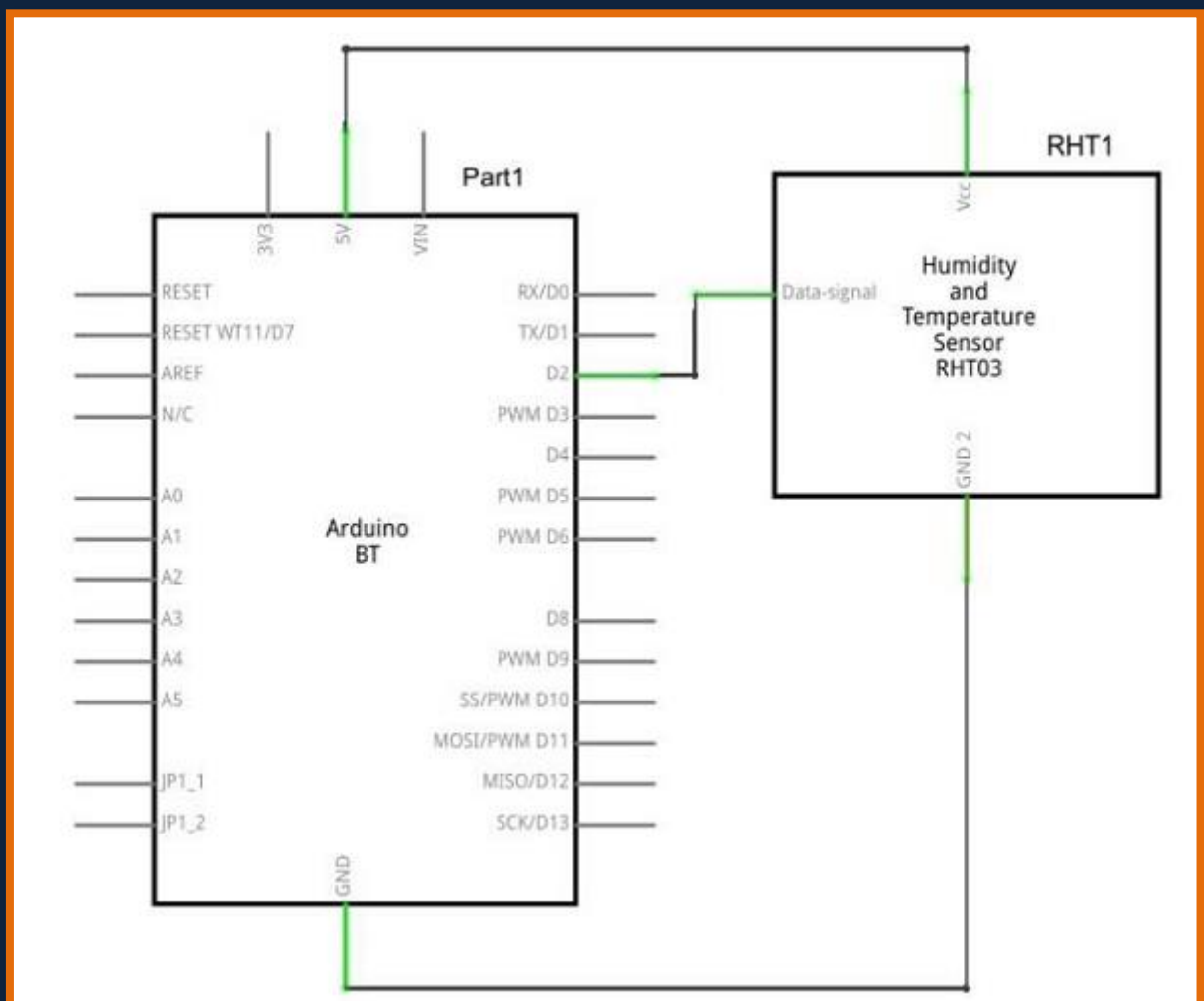
Componentes Necessários

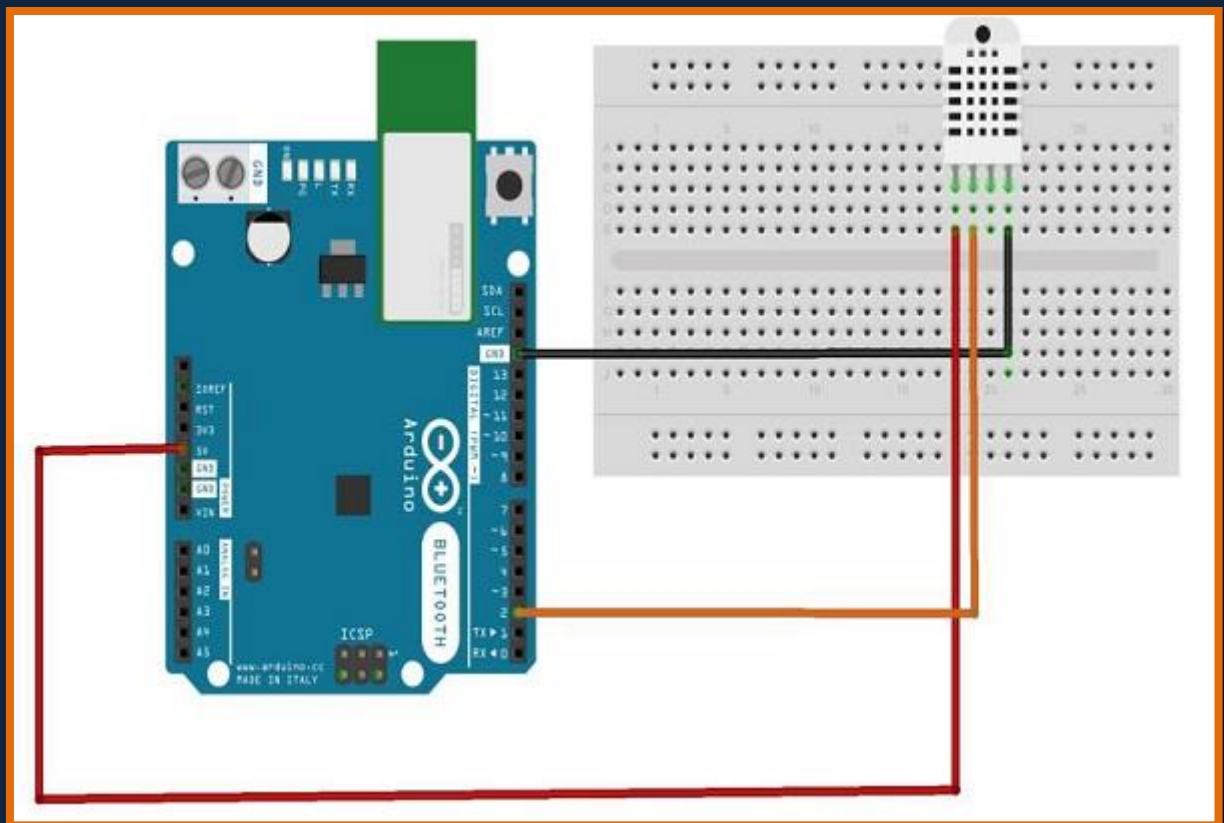
Você precisará dos seguintes componentes -

- 1 × Breadboard
- 1 × Arduino Uno R3
- 1 × DHT22
- Resistor de 1 × 10K ohm

Procedimento

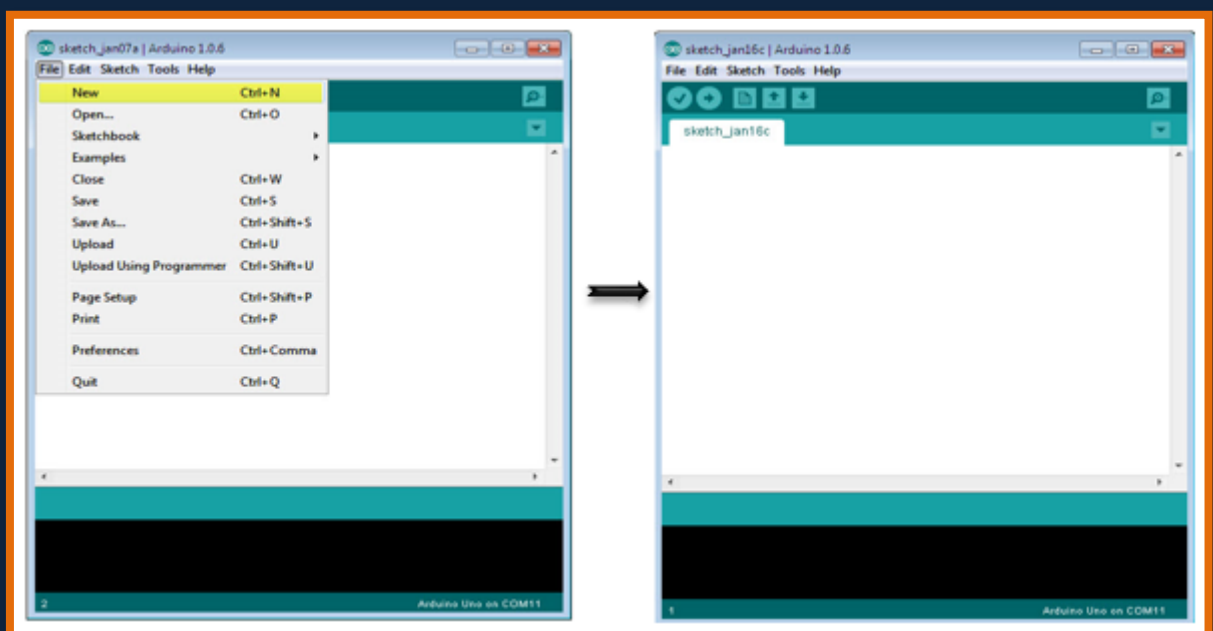
Siga o diagrama do circuito e conecte os componentes na tábua de pão, como mostra a imagem abaixo.





Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

```
// Example testing sketch for various DHT humidity/temperature sensors

#include "DHT.h"
#define DHTPIN 2 // what digital pin we're connected to
// Uncomment whatever type you're using!
// #define DHTTYPE DHT11 // DHT 11
#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
// #define DHTTYPE DHT21 // DHT 21 (AM2301)
// Connect pin 1 (on the left) of the sensor to +5V
// NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
// to 3.3V instead of 5V!
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor
// Initialize DHT sensor.
// Note that older versions of this library took an optional third parameter to
// tweak the timings for faster processors. This parameter is no longer needed
// as the current DHT reading algorithm adjusts itself to work on faster procs.
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHTxx test!");
  dht.begin();
}

void loop() {
  delay(2000); // Wait a few seconds between measurements
  float h = dht.readHumidity();
  // Reading temperature or humidity takes about 250 milliseconds!
  float t = dht.readTemperature();
  // Read temperature as Celsius (the default)
  float f = dht.readTemperature(true);
  // Read temperature as Fahrenheit (isFahrenheit = true)
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Compute heat index in Fahrenheit (the default)
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);
  Serial.print ("Humidity: ");
  Serial.print (h);
  Serial.print (" %\n");
  Serial.print ("Temperature: ");
```



```
Serial.print (t);  
Serial.print (" *C ");  
Serial.print (f);  
Serial.print (" *F\\t");  
Serial.print ("Heat index: ");  
Serial.print (hic);  
Serial.print (" *C ");  
Serial.print (hif);  
Serial.println (" *F");  
}
```

Código a Nota

O sensor DHT22 possui quatro terminais (V_{cc} , DATA, NC, GND), que são conectados à placa da seguinte maneira -

- Pino de dados no pino número 2 do Arduino
- V_{cc} pin a 5 volts da placa Arduino
- Pino GND no chão da placa Arduino
- Precisamos conectar um resistor de 10k ohm (resistor de pull up) entre o DATA e o pino V_{cc}

Depois que as conexões de hardware forem concluídas, você precisará adicionar a biblioteca DHT22 ao seu arquivo de biblioteca do Arduino, conforme descrito anteriormente.

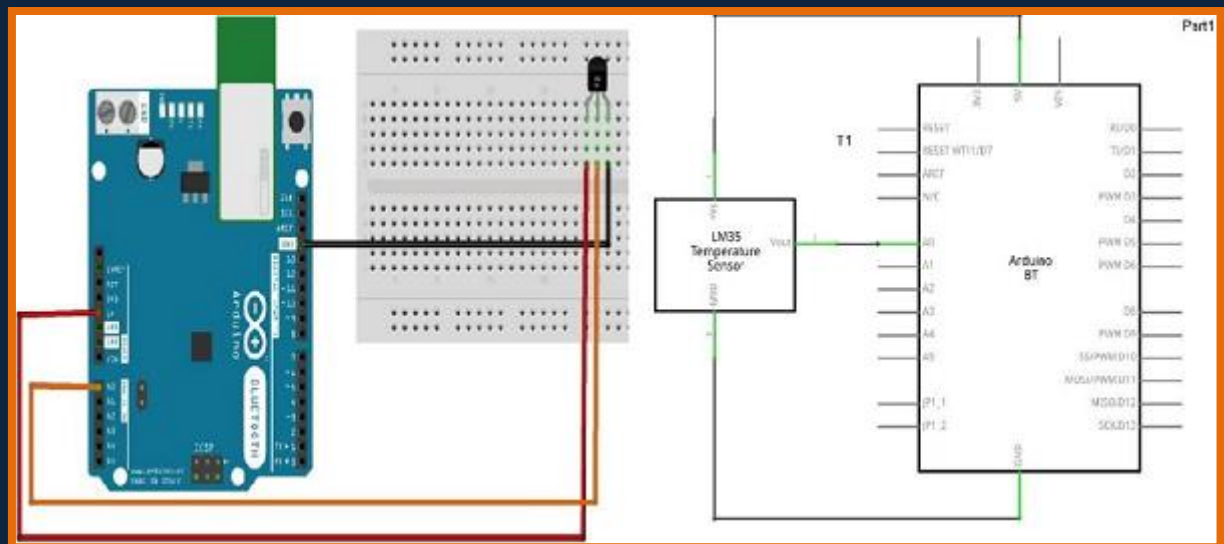
Resultado

Você verá a temperatura e a umidade no monitor da porta serial, que são atualizadas a cada 2 segundos.

Arduino - Sensor de temperatura

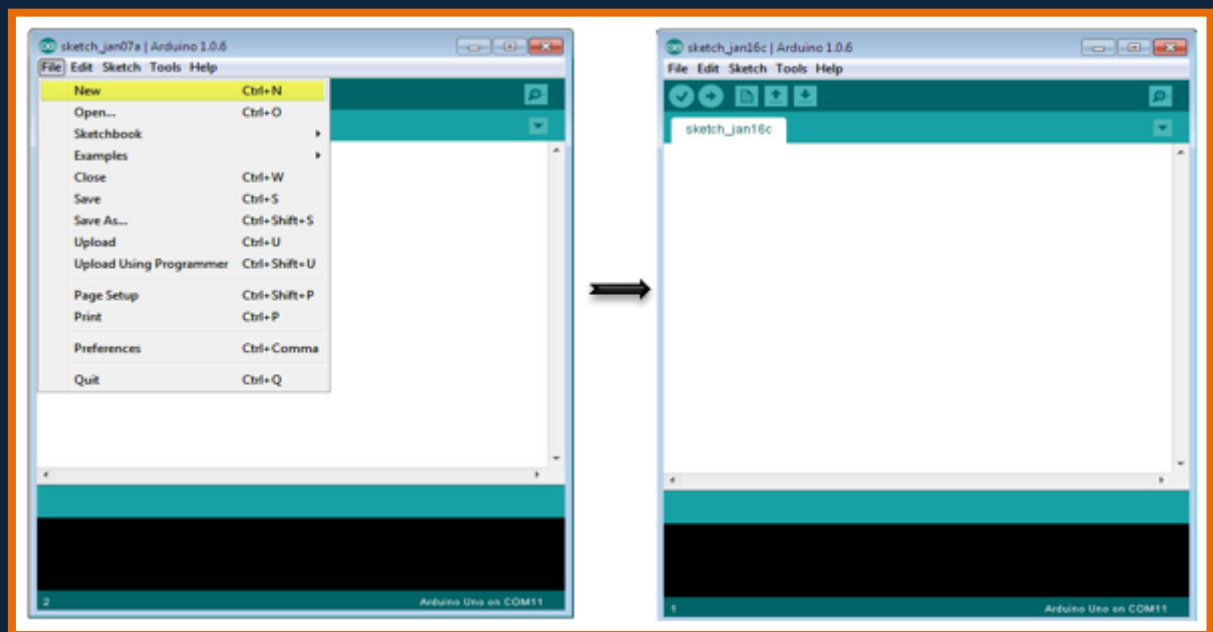
Os sensores de temperatura da série LM35 são dispositivos de temperatura de circuito integrado de precisão com uma tensão de saída linearmente proporcional à temperatura centígrada.

O dispositivo LM35 tem uma vantagem sobre os sensores de temperatura lineares calibrados em Kelvin, pois o usuário não precisa subtrair uma grande tensão constante da saída para obter uma escala Centígrada conveniente. O dispositivo LM35 não requer nenhuma calibração ou ajuste externo para fornecer precisões típicas de $\pm 0.1^\circ \text{C}$ à temperatura ambiente e $\pm 0.1^\circ \text{C}$ em uma faixa de temperatura completa de -55°C a 150°C .



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

```
float temp;
int tempPin = 0;

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

```

void loop() {
  temp = analogRead(tempPin);
  // read analog volt from sensor and save to variable temp
  temp = temp * 0.48828125;
  // convert the analog volt to its temperature equivalent
  Serial.print("TEMPERATURE = ");
  Serial.print(temp); // display temperature value
  Serial.print("*C");
  Serial.println();
  delay(1000); // update sensor reading each one second
}

```

Código a Nota

O sensor LM35 possui três terminais - V_s , V_{out} e GND. Vamos conectar o sensor da seguinte maneira -

- Conecte o $+V_s$ de + 5V em sua placa Arduino.
- Conecte V_{out} ao Analog0 ou A0 na placa Arduino.
- Conecte o GND ao GND no Arduino.

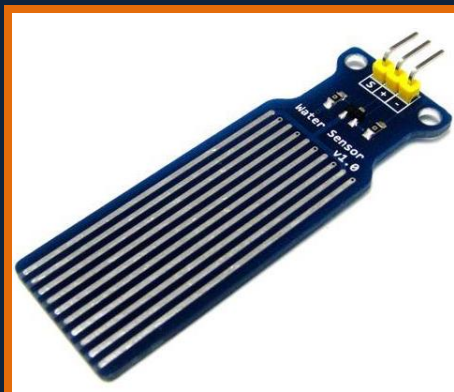
O conversor analógico para digital (ADC) converte valores analógicos em uma aproximação digital com base na fórmula $ADC \text{ Value} = \text{sample} * 1024 / \text{tensão de referência (+ 5v)}$. Portanto, com uma referência de +5 volts, a aproximação digital será igual à tensão de entrada * 205.

Resultado

Você verá a temperatura exibida no monitor da porta serial, que é atualizada a cada segundo.

Arduino - Detector / Sensor de Água

O tijolo do sensor de água foi projetado para a detecção de água, que pode ser amplamente utilizada na detecção de chuvas, nível de água e até vazamentos de líquidos.



Conectar um sensor de água a um Arduino é uma ótima maneira de detectar vazamentos, derramamentos, inundações, chuvas, etc. Ele pode ser usado para detectar a presença, o nível, o volume e / ou a ausência de água. Embora isso possa ser usado para lembrá-lo de regar suas plantas, existe um sensor Grove melhor para isso. O sensor possui uma variedade de traços expostos, que indicam BAIXO quando a água é detectada.

Neste capítulo, conectaremos o sensor de água ao Digital Pin 8 no Arduino e incluiremos o LED muito útil para ajudar a identificar quando o sensor de água entra em contato com uma fonte de água.

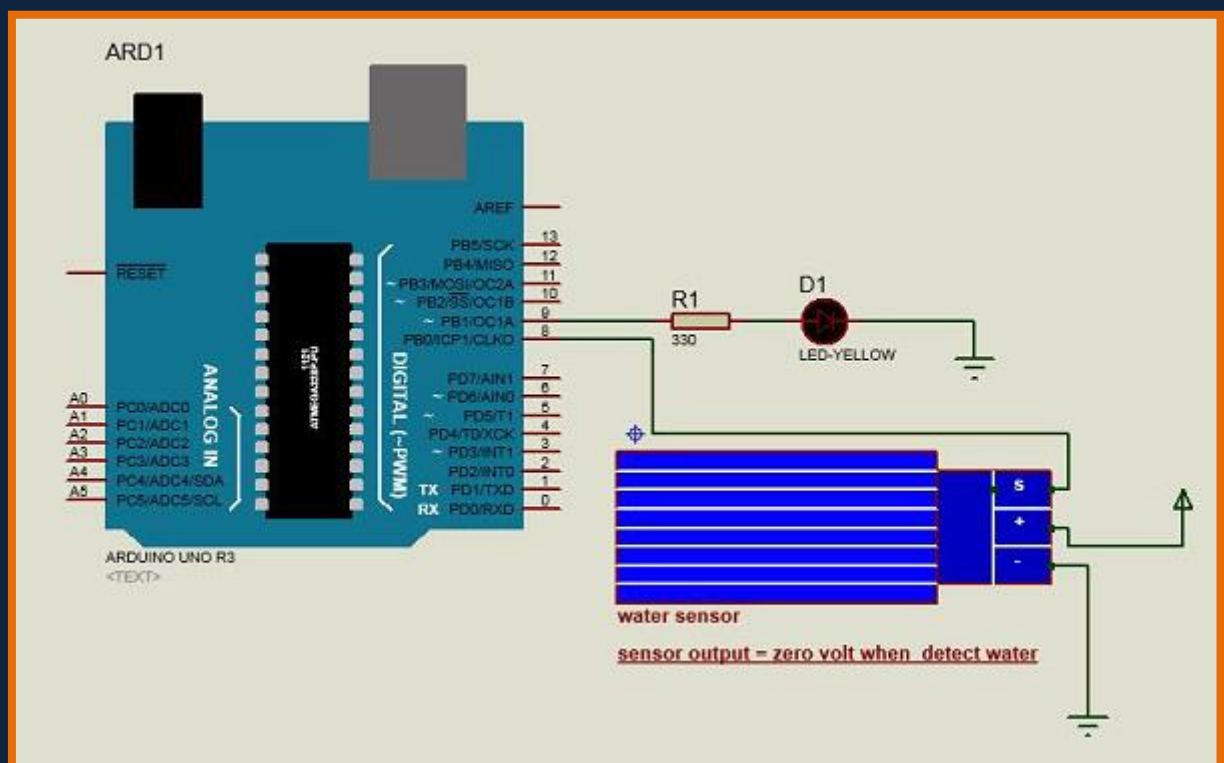
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × Breadboard
- 1 × Arduino Uno R3
- 1 × sensor de água
- 1 × led
- Resistor de 1 × 330 ohm

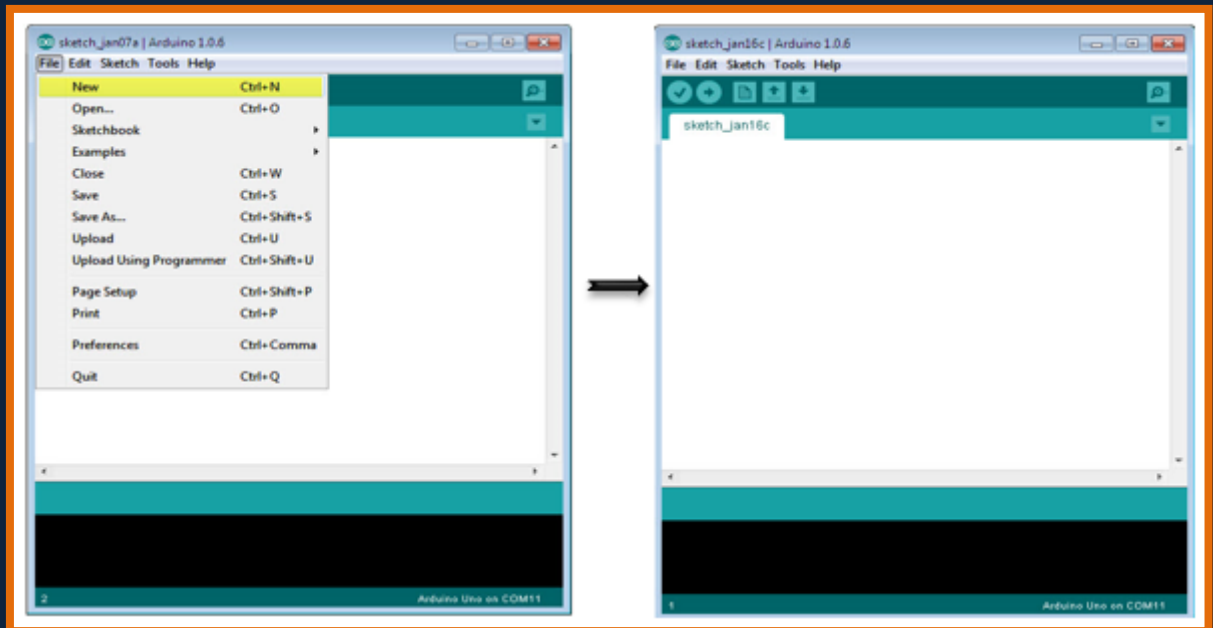
Procedimento

Siga o diagrama do circuito e conecte os componentes na tábua de pão, como mostra a imagem abaixo.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

```
#define Grove_Water_Sensor 8 // Attach Water sensor to Arduino Digital Pin 8
#define LED 9 // Attach an LED to Digital Pin 9 (or use onboard LED)

void setup() {
  pinMode(Grove_Water_Sensor, INPUT); // The Water Sensor is an Input
  pinMode(LED, OUTPUT); // The LED is an Output
}

void loop() {
  /* The water sensor will switch LOW when water is detected.
  Get the Arduino to illuminate the LED and activate the buzzer
  when water is detected, and switch both off when no water is present */
  if( digitalRead(Grove_Water_Sensor) == LOW) {
    digitalWrite(LED,HIGH);
  }else {
    digitalWrite(LED,LOW);
  }
}
```

Código a Nota

O sensor de água possui três terminais - S, V_{out} (+) e GND (-). Conecte o sensor da seguinte maneira -

- Conecte o + V_s de + 5V em sua placa Arduino.
- Conecte S ao pino digital número 8 na placa Arduino.
- Conecte o GND ao GND no Arduino.
- Conecte o LED ao pino digital número 9 na placa Arduino.

Quando o sensor detecta água, o pino 8 no Arduino fica BAIXO e o LED no Arduino acende.

Resultado

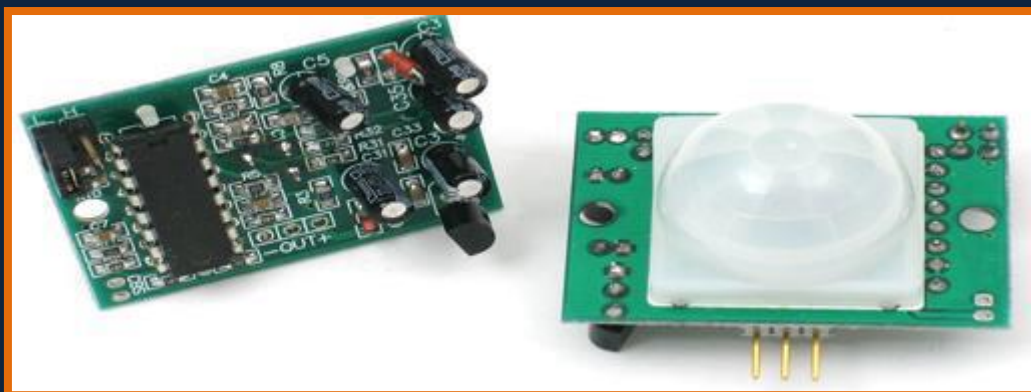
Você verá o LED de indicação acender quando o sensor detectar água.

Arduino - Sensor PIR

Os sensores PIR permitem sentir o movimento. Eles são usados para detectar se um humano se moveu dentro ou fora do alcance do sensor. Eles são comumente encontrados em aparelhos e gadgets usados em casa ou para empresas. Eles são freqüentemente chamados de sensores PIR, "Infravermelho Passivo", "Piroelétrico" ou "Movimento de IR".

A seguir estão as vantagens dos sensores PIR -

- Tamanho pequeno
- Ampla gama de lentes
- Fácil interface
- Barato
- De baixa potência
- Fácil de usar
- Não se desgasta

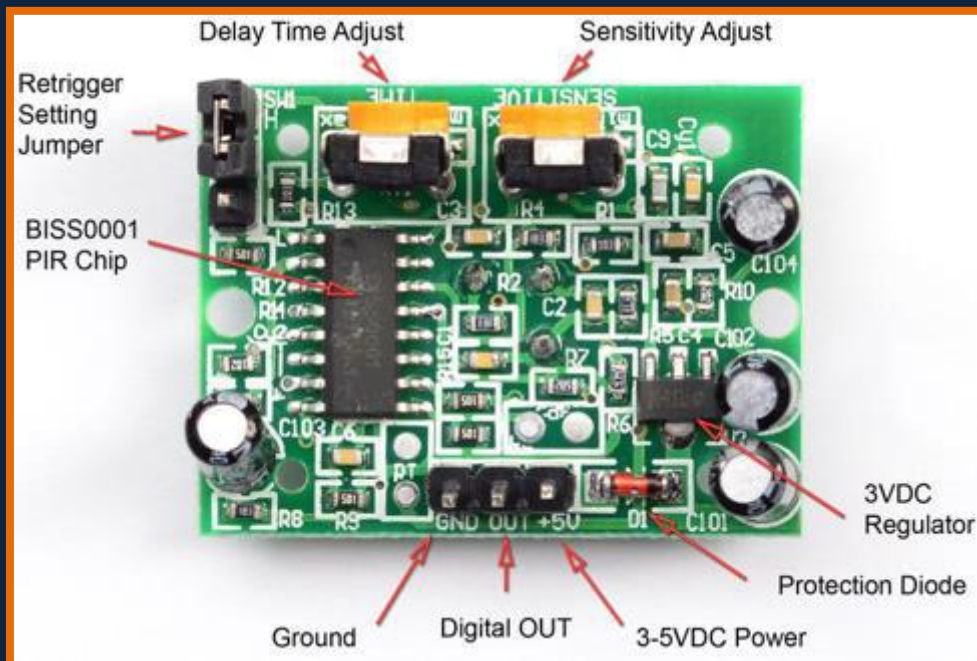


Os PIRs são feitos de sensores piroelétricos, uma lata redonda de metal com um cristal retangular no centro, que pode detectar níveis de radiação infravermelha. Tudo emite radiação de baixo nível e, quanto mais quente é, mais radiação é emitida. O sensor em um detector de movimento é dividido em duas metades. Isso é para detectar

movimento (alteração) e não os níveis médios de IR. As duas metades estão conectadas para que se cancelem. Se metade vê mais ou menos radiação infravermelha que a outra, a saída oscila alta ou baixa.



Os PIRs têm configurações ajustáveis e um cabeçalho instalado nos blocos de aterramento / saída / alimentação de 3 pinos.



Para muitos projetos ou produtos básicos que precisam detectar quando uma pessoa sai ou entra na área, os sensores PIR são ótimos. Observe que os PIRs não informam o número de pessoas ao redor ou a proximidade com o sensor. As lentes costumam ser fixadas a uma certa distância e, às vezes, são acionadas pelos animais domésticos.

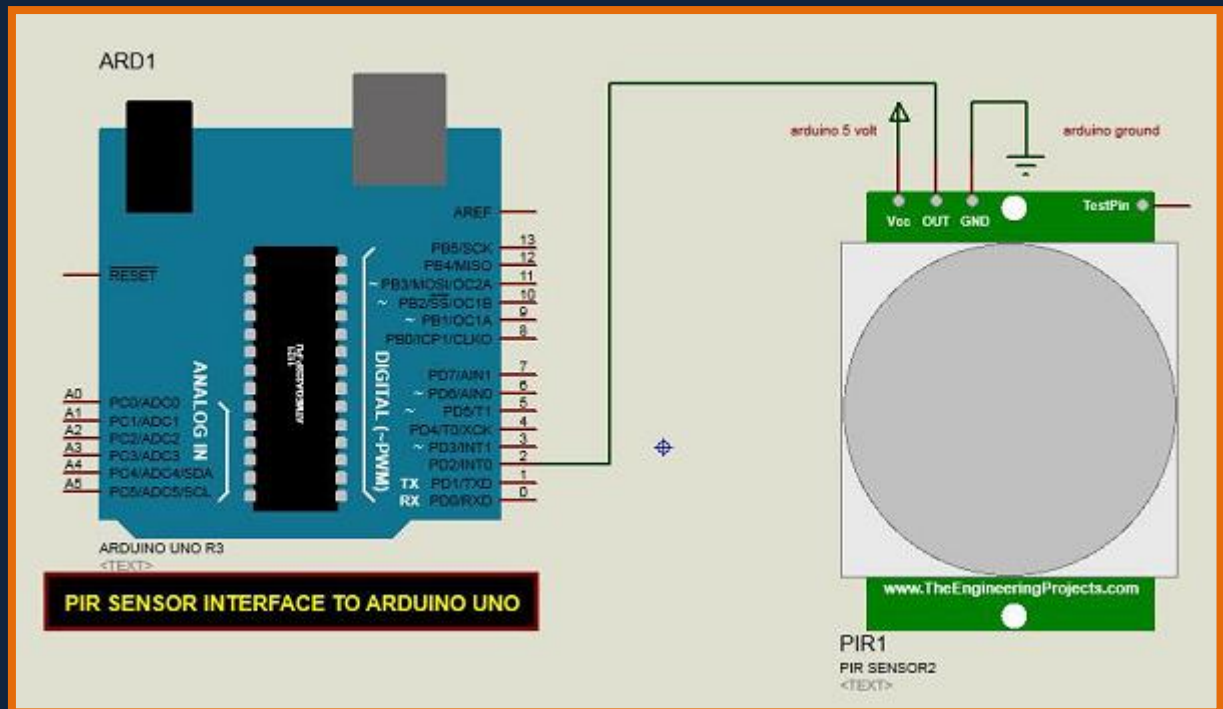
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × Breadboard
- 1 × Arduino Uno R3
- 1 × sensor PIR (MQ3)

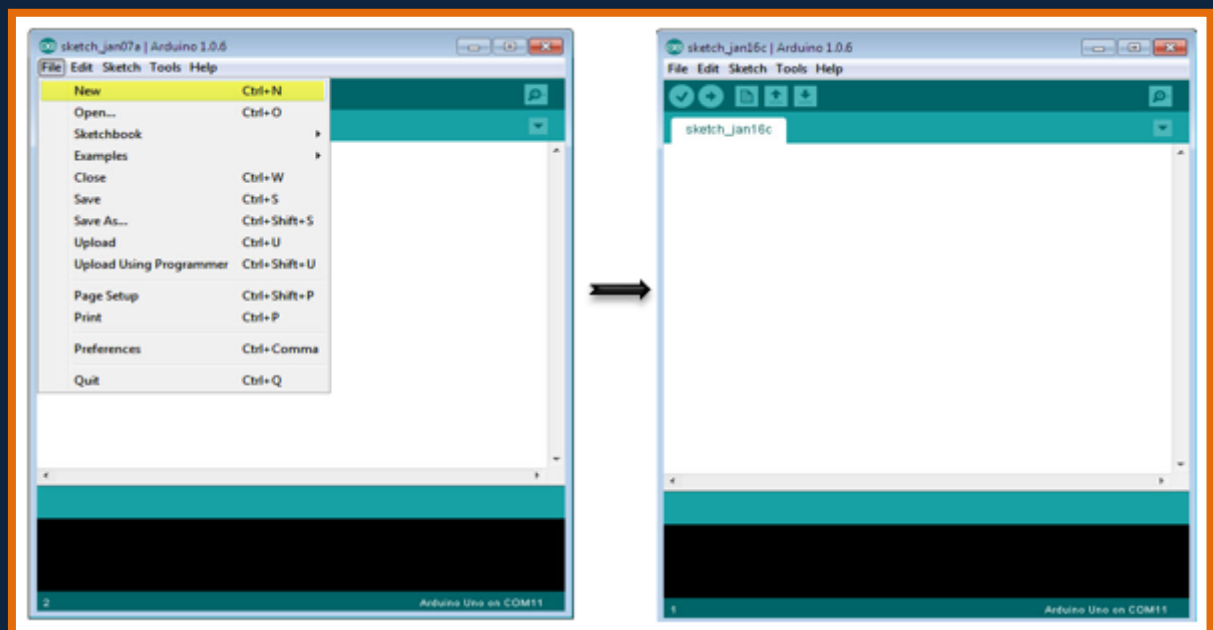
Procedimento

Siga o diagrama do circuito e faça as conexões conforme mostrado na imagem abaixo.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

```
#define pirPin 2
int calibrationTime = 30;
long unsigned int lowIn;
long unsigned int pause = 5000;
boolean lockLow = true;
boolean takeLowTime;
int PIRValue = 0;

void setup() {
  Serial.begin(9600);
  pinMode(pirPin, INPUT);
}

void loop() {
  PIRSensor();
}

void PIRSensor() {
  if(digitalRead(pirPin) == HIGH) {
    if(lockLow) {
      PIRValue = 1;
      lockLow = false;
      Serial.println("Motion detected.");
      delay(50);
    }
    takeLowTime = true;
  }
  if(digitalRead(pirPin) == LOW) {
    if(takeLowTime){
      lowIn = millis();takeLowTime = false;
    }
    if(!lockLow && millis() - lowIn > pause) {
      PIRValue = 0;
      lockLow = true;
      Serial.println("Motion ended.");
      delay(50);
    }
  }
}
```

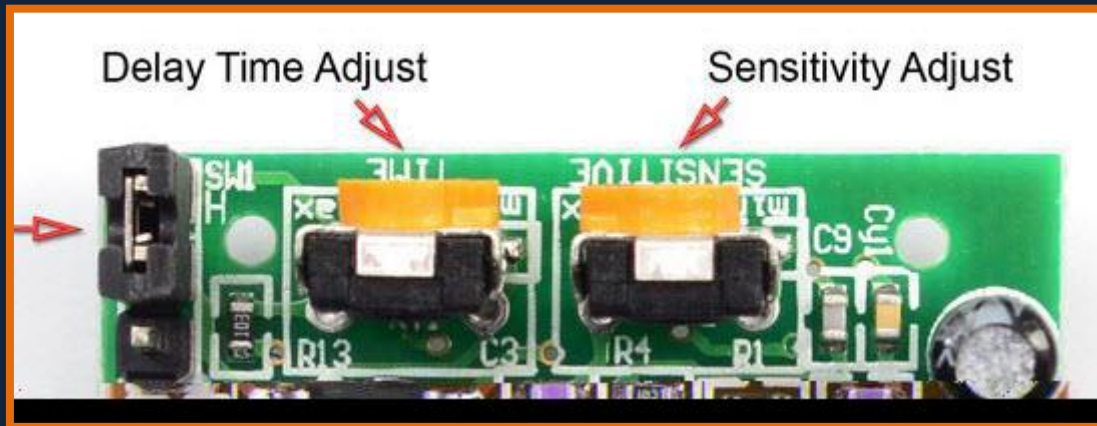
Código a Nota

O sensor PIR possui três terminais - V_{cc} , OUT e GND. Conecte o sensor da seguinte maneira -

- Conecte o + V_{cc} a + 5v na placa Arduino.
- Conecte OUT ao pino digital 2 na placa Arduino.

- Conecte o GND ao GND no Arduino.

Você pode ajustar a sensibilidade do sensor e o tempo de atraso por meio de dois resistores variáveis localizados na parte inferior da placa do sensor.



Depois que o sensor detecta qualquer movimento, o Arduino envia uma mensagem pela porta serial dizendo que um movimento é detectado. O movimento do sensor PIR atrasará um certo tempo para verificar se há um novo movimento. Se não houver movimento detectado, o Arduino enviará uma nova mensagem informando que o movimento foi encerrado.

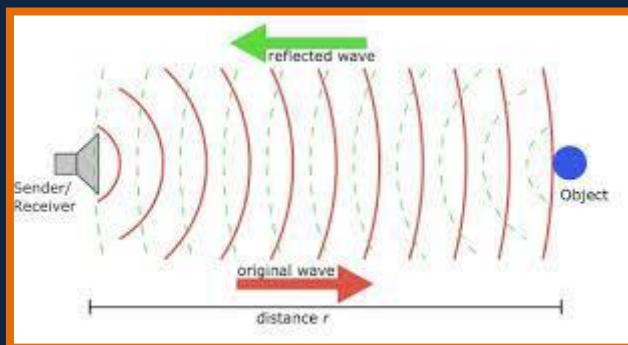
Resultado

Você verá uma mensagem na sua porta serial se um movimento for detectado e outra mensagem quando o movimento for interrompido.

Arduino - Sensor Ultrassônico

O sensor ultrassônico HC-SR04 usa o SONAR para determinar a distância de um objeto, exatamente como os morcegos. Oferece excelente detecção de faixa sem contato com alta precisão e leituras estáveis em um pacote fácil de usar de 2 cm a 400 cm ou 1 "a 13 pés.

A operação não é afetada pela luz solar ou material preto, embora acusticamente, materiais macios, como tecidos, possam ser difíceis de detectar. Ele vem completo com transmissor ultrassônico e módulo receptor.



Especificações técnicas

- Fonte de alimentação - + 5V DC
- Corrente quieta - <2mA
- Corrente de trabalho - 15mA
- Ângulo efetivo - <15 °
- Distância - 2cm - 400 cm / 1 " - 13 pés
- Resolução - 0,3 cm
- Ângulo de medição - 30 graus

Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × Breadboard

Código Arduino

```
const int pingPin = 7; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 6; // Echo Pin of Ultrasonic Sensor

void setup() {
  Serial.begin(9600); // Starting Serial Terminal
}

void loop() {
  long duration, inches, cm;
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();
  delay(100);
}

long microsecondsToInches(long microseconds) {
  return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds) {
  return microseconds / 29 / 2;
}
```

Código a Nota

O sensor ultrassônico possui quatro terminais - + 5V, Trigger, Echo e GND conectados da seguinte maneira -

- Conecte o pino + 5V a + 5v na sua placa Arduino.
- Conecte o Trigger ao pino digital 7 na sua placa Arduino.
- Conecte o Echo ao pino digital 6 da sua placa Arduino.
- Conecte o GND ao GND no Arduino.

Em nosso programa, exibimos a distância medida pelo sensor em polegadas e cm através da porta serial.

Resultado

Você verá a distância medida pelo sensor em polegadas e cm no monitor serial do Arduino.

Arduino - Interruptor de conexão

Botões ou interruptores conectam dois terminais abertos em um circuito. Este exemplo acende o LED no pino 2 quando você pressiona o botão de pressão conectado ao pino 8.

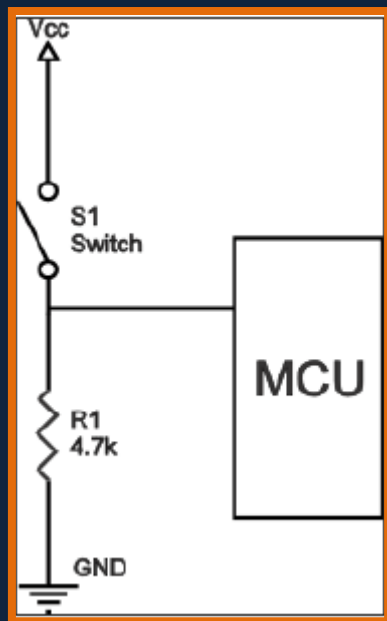


Resistor pull-down

Os resistores pull-down são usados em circuitos lógicos eletrônicos para garantir que as entradas do Arduino se estabeleçam nos níveis lógicos esperados se os dispositivos externos estiverem desconectados ou estiverem com alta impedância. Como nada está conectado a um pino de entrada, isso não significa que seja um zero lógico. Os resistores de tração estão conectados entre o terra e o pino apropriado no dispositivo.

Um exemplo de um resistor pull-down em um circuito digital é mostrado na figura a seguir. Um interruptor de botão está conectado entre a tensão de alimentação e um pino do microcontrolador. Nesse circuito, quando o interruptor está fechado, a entrada do microcontrolador tem um valor alto lógico, mas quando o interruptor está aberto, o resistor de tração puxa a tensão de entrada para o chão (valor zero lógico), impedindo um estado indefinido na entrada.

O resistor de pull-down deve ter uma resistência maior do que a impedância do circuito lógico, caso contrário, pode diminuir demais a tensão e a tensão de entrada no pino permanecerá em um valor baixo lógico constante, independentemente da posição do interruptor.



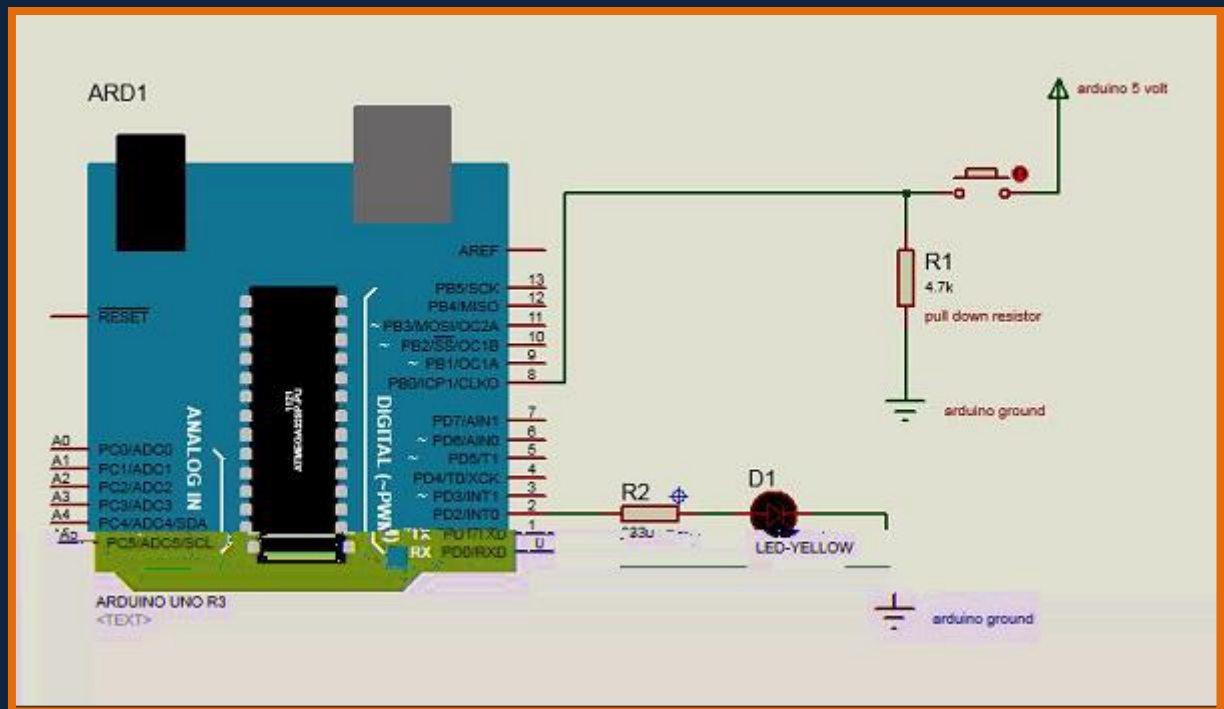
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × placa UNO Arduino
- Resistor de 1 × 330 ohm
- Resistor de 1 × 4.7K ohm (puxe para baixo)
- 1 × LED

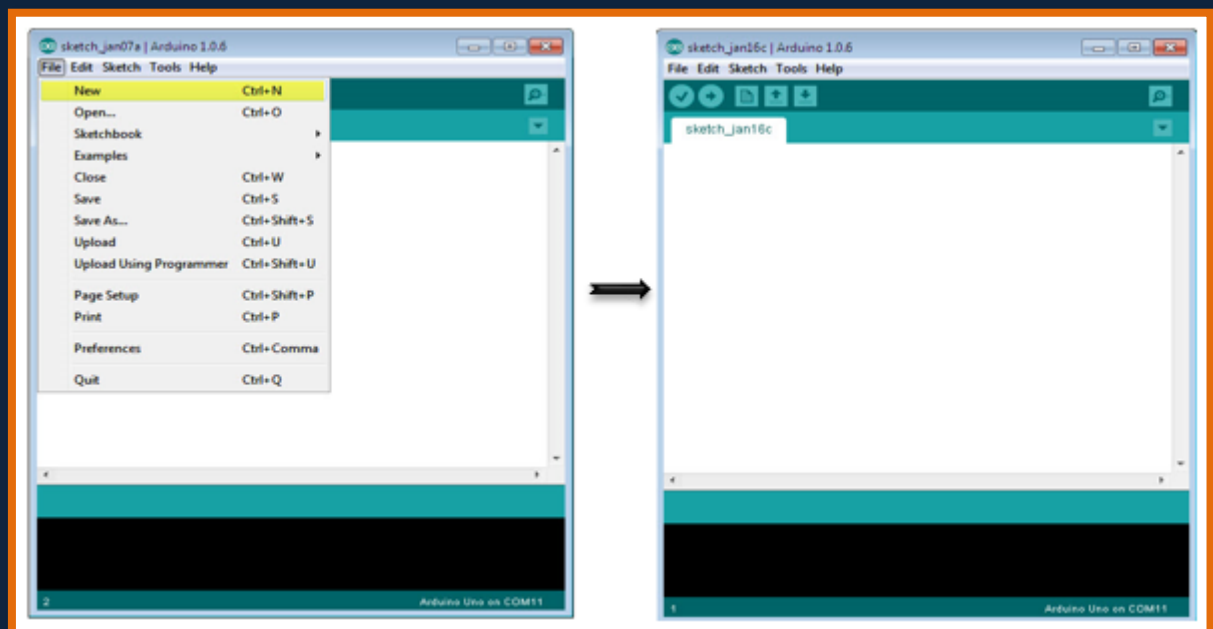
Procedimento

Siga o diagrama do circuito e faça as conexões conforme mostrado na imagem abaixo.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 8; // the number of the pushbutton pin
```

```

const int ledPin = 2; // the number of the LED pin
// variables will change:
int buttonState = 0; // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);
  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}

```

Código a Nota

Quando o interruptor está aberto (o botão não está pressionado), não há conexão entre os dois terminais do botão; portanto, o pino é conectado ao terra (através do resistor de pull-down) e lemos LOW. Quando o interruptor é fechado (o botão é pressionado), ele faz uma conexão entre seus dois terminais, conectando o pino a 5 volts, para que possamos ler um ALTO.

Resultado

O LED acende quando o botão é pressionado e apaga quando é liberado.

Arduino - Motor de passo

Um motor de passo ou motor de passo é um motor síncrono e sem escova, que divide uma rotação completa em várias etapas. Ao contrário de um motor CC sem escova, que gira continuamente quando uma tensão CC fixa é aplicada a ele, um motor de passo gira em ângulos de passo discretos.

Portanto, os motores de passo são fabricados com etapas por rotação de 12, 24, 72, 144, 180 e 200, resultando em ângulos de inclinação de 30, 15, 5, 2,5, 2 e 1,8 graus por etapa. O motor de passo pode ser controlado com ou sem feedback.

Imagine um motor em um avião RC. O motor gira muito rápido em uma direção ou outra. Você pode variar a velocidade com a quantidade de energia fornecida ao motor, mas não pode pedir à hélice que pare em uma posição específica.

Agora imagine uma impressora. Existem muitas partes móveis dentro de uma impressora, incluindo motores. Um desses motores atua como alimentação de papel, girando os rolos que movem o pedaço de papel à medida que a tinta está sendo impressa. Esse motor precisa poder mover o papel a uma distância exata para poder imprimir a próxima linha de texto ou a próxima linha de uma imagem.

Há outro motor conectado a uma haste rosqueada que move a cabeça de impressão para frente e para trás. Novamente, essa haste rosqueada precisa ser movida uma quantidade exata para imprimir uma letra após a outra. É aqui que os motores de passo são úteis.

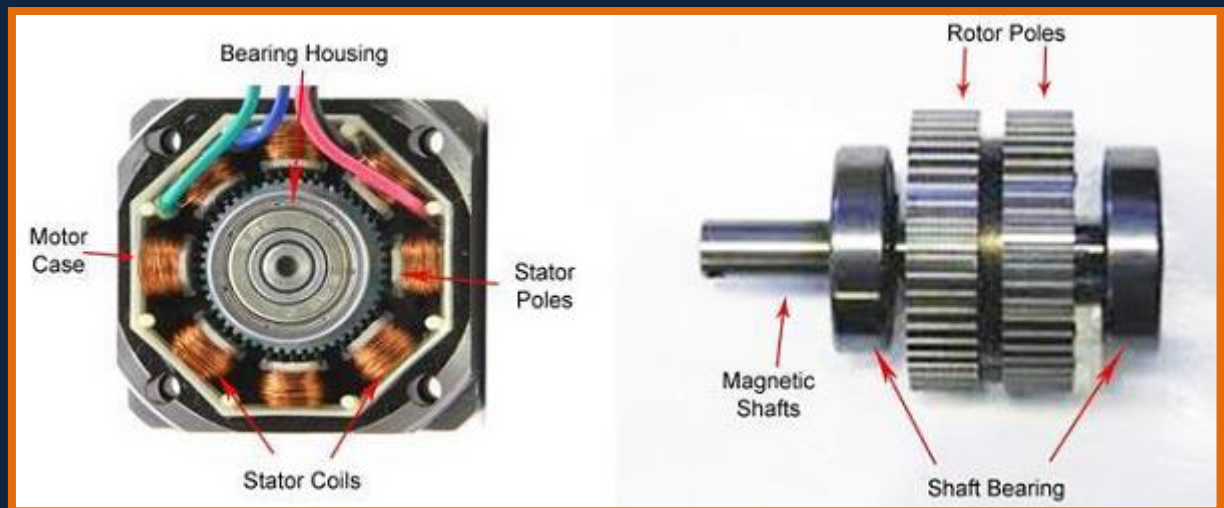


Como funciona um motor de passo?

Um motor CC regular gira apenas na direção, enquanto um motor de passo pode girar em incrementos precisos.

Os motores de passo podem girar uma quantidade exata de graus (ou etapas), conforme desejado. Isso fornece controle total sobre o motor, permitindo movê-lo para um local exato e manter essa posição. Isso é feito alimentando as bobinas dentro do motor por períodos muito curtos. A desvantagem é que você precisa ligar o motor o tempo todo para mantê-lo na posição que deseja.

Tudo o que você precisa saber por enquanto é que, para mover um motor de passo, você diz para ele mover um certo número de etapas em uma direção ou outra, e diz a velocidade na qual pisar nessa direção. Existem inúmeras variedades de motores de passo. Os métodos descritos aqui podem ser usados para inferir como usar outros motores e drivers que não são mencionados neste tutorial. No entanto, é sempre recomendável que você consulte as fichas técnicas e os guias dos motores e drivers específicos para os modelos que você possui.



Componentes Necessários

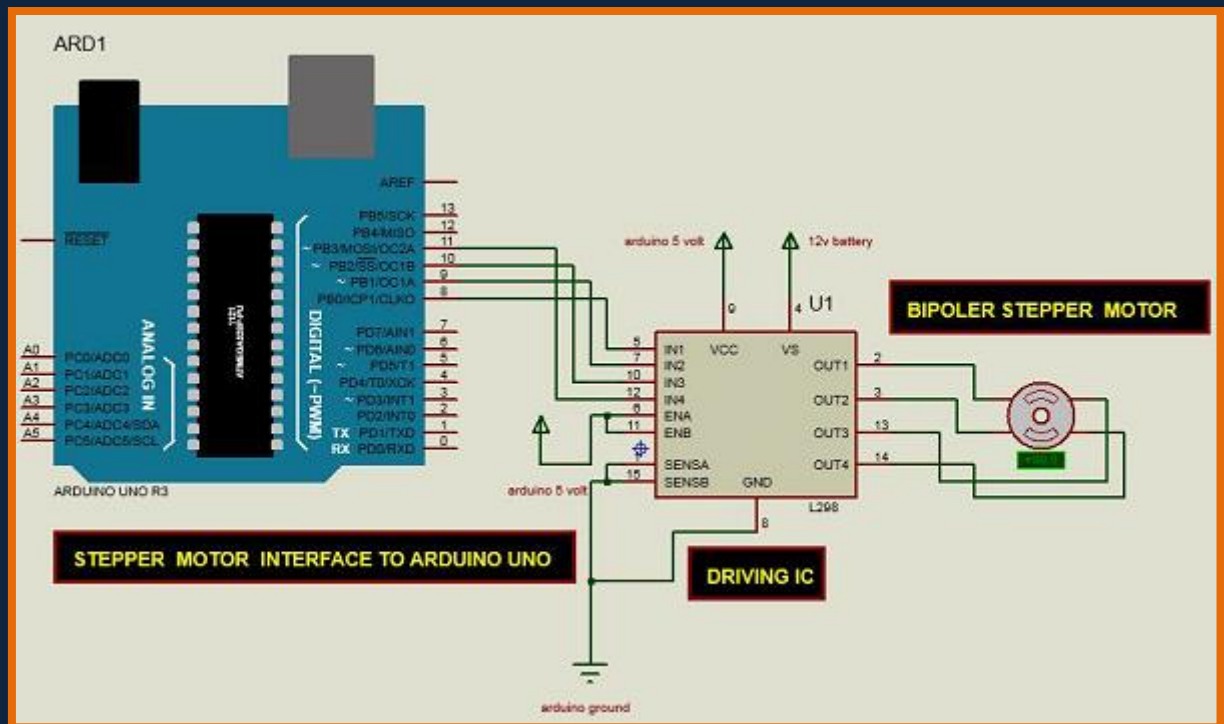
Você precisará dos seguintes componentes -

- 1 × placa UNO Arduino
- 1 × pequeno motor de passo bipolar, como mostra a imagem abaixo
- 1 × LM298 IC de condução



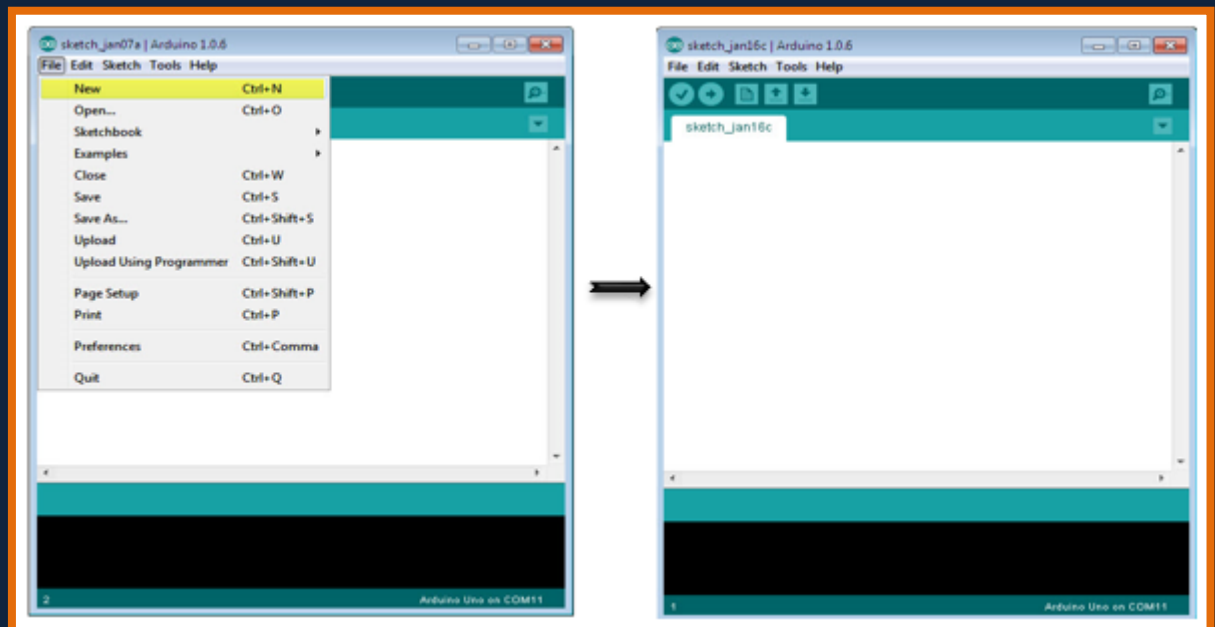
Procedimento

Siga o diagrama do circuito e faça as conexões conforme mostrado na imagem abaixo.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

```
/* Stepper Motor Control */
```

```

#include <Stepper.h>
const int stepsPerRevolution = 90;
// change this to fit the number of steps per revolution
// for your motor
// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);

void setup() {
  // set the speed at 60 rpm:
  myStepper.setSpeed(5);
  // initialize the serial port:
  Serial.begin(9600);
}

void loop() {
  // step one revolution in one direction:
  Serial.println("clockwise");
  myStepper.step(stepsPerRevolution);
  delay(500);
  // step one revolution in the other direction:
  Serial.println("counterclockwise");
  myStepper.step(-stepsPerRevolution);
  delay(500);
}

```

Código a Nota

Este programa aciona um motor de passo unipolar ou bipolar. O motor está conectado aos pinos digitais 8 a 11 do Arduino.

Resultado

O motor levará uma rotação em uma direção, depois uma rotação na outra direção.

Arduino - Servo Motor

Um servomotor é um pequeno dispositivo que possui um eixo de saída. Este eixo pode ser posicionado em posições angulares específicas enviando ao servo um sinal codificado. Enquanto o sinal codificado existir na linha de entrada, o servo manterá a posição angular do eixo. Se o sinal codificado mudar, a posição angular do eixo mudará. Na prática, os servos são usados em aviões controlados por rádio para posicionar superfícies de controle como elevadores e lemes. Eles também são usados em carros controlados por rádio, fantoches e, claro, robôs.



Servos são extremamente úteis em robótica. Os motores são pequenos, possuem circuitos de controle integrados e são extremamente poderosos para seu tamanho. Um servo padrão como o Futaba S-148 possui torque de 42 onças / polegadas, o que é forte para seu tamanho. Também consome energia proporcional à carga mecânica. Um servo com carga leve, portanto, não consome muita energia.

As entranhas de um servomotor são mostradas na figura a seguir. Você pode ver o circuito de controle, o motor, um conjunto de engrenagens e o estojo. Você também pode ver os três fios que se conectam ao mundo exterior. Um é para energia (+ 5 volts), terra, e o fio branco é o fio de controle.



Trabalho de um servo motor

O servomotor possui alguns circuitos de controle e um potenciômetro (um resistor variável, também conhecido como pot), conectado ao eixo de saída. Na figura acima, o pote pode ser visto no lado direito da placa de circuito. Este pote permite que o circuito de controle monitore o ângulo de corrente do servo motor.

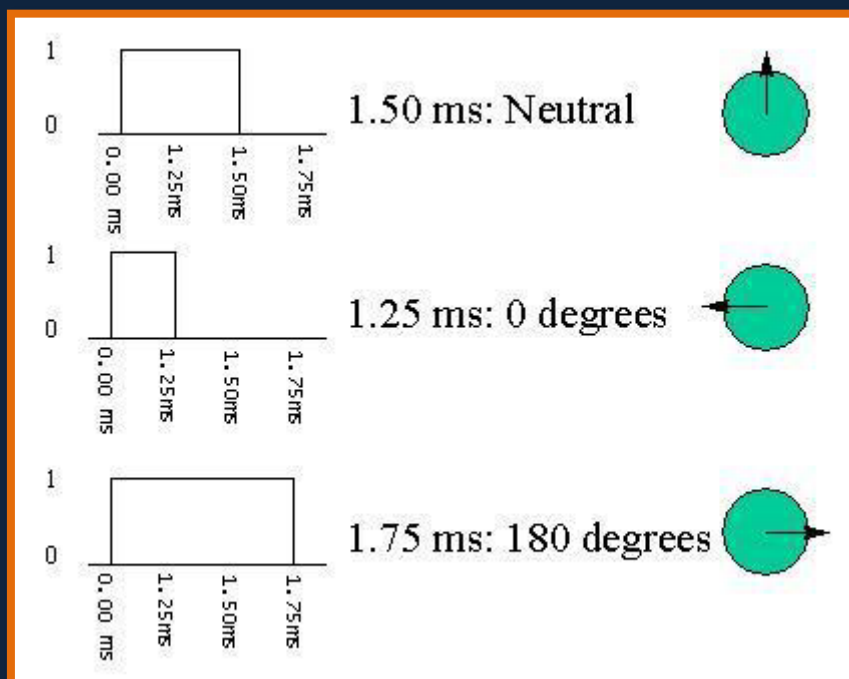
Se o eixo estiver no ângulo correto, o motor será desligado. Se o circuito achar que o ângulo não está correto, ele girará o motor até que esteja no ângulo desejado. O eixo de saída do servo é capaz de viajar em algum lugar em torno de 180 graus. Geralmente, ele está em algum lugar na faixa de 210 graus, no entanto, varia dependendo do

fabricante. Um servo normal é usado para controlar um movimento angular de 0 a 180 graus. Ele não é mecanicamente capaz de girar mais longe devido a uma parada mecânica construída na engrenagem de saída principal.

A potência aplicada ao motor é proporcional à distância que ele precisa percorrer. Portanto, se o eixo precisar girar uma grande distância, o motor funcionará a toda velocidade. Se precisar girar apenas uma pequena quantidade, o motor funcionará a uma velocidade mais lenta. Isso é chamado de **controle proporcional**.

Como você comunica o ângulo em que o servo deve girar?

O fio de controle é usado para comunicar o ângulo. O ângulo é determinado pela duração de um pulso aplicado ao fio de controle. Isso é chamado de **modulação por código de pulso**. O servo espera ver um pulso a cada 20 milissegundos (0,02 segundos). A duração do pulso determinará a que distância o motor gira. Um pulso de 1,5 milissegundos, por exemplo, fará o motor girar para a posição de 90 graus (geralmente chamada de posição neutra). Se o pulso for menor que 1,5 milissegundos, o motor girará o eixo para mais perto de 0 graus. Se o pulso for maior que 1,5 milissegundos, o eixo se aproximará de 180 graus.



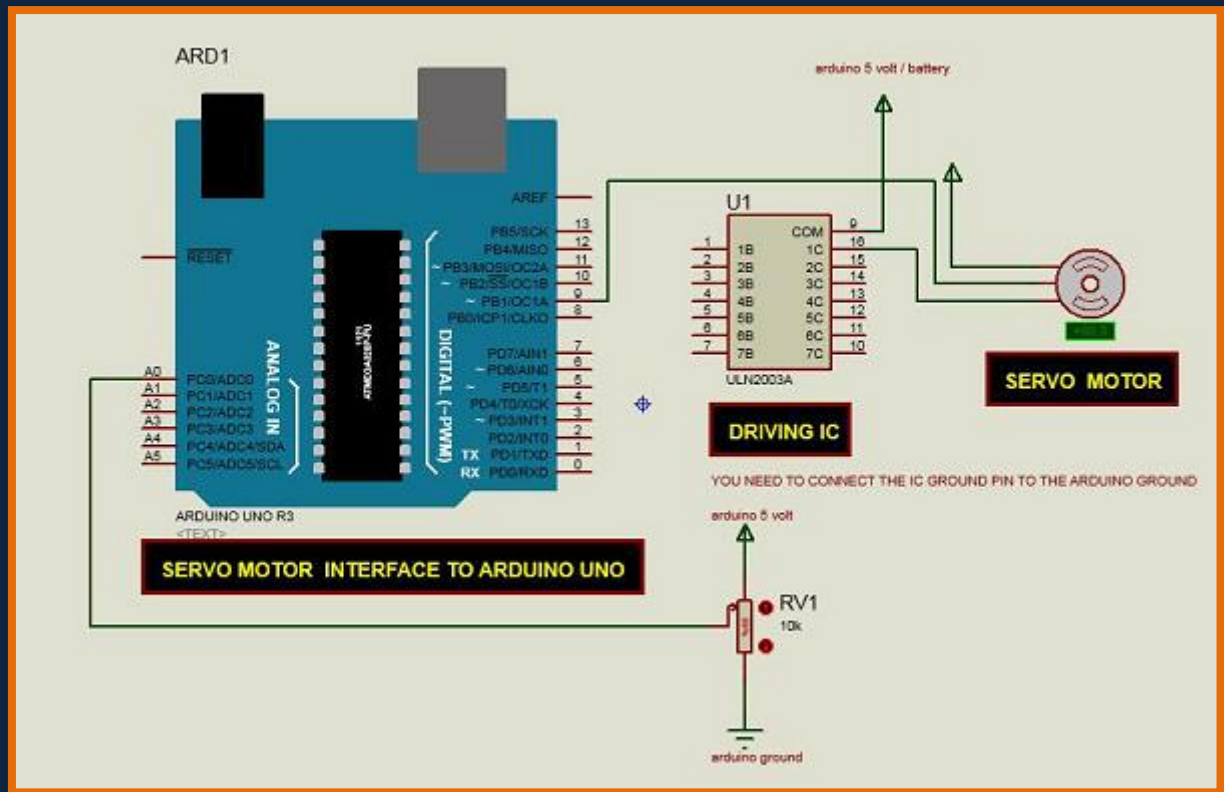
Componentes Necessários

Você precisará dos seguintes componentes -

- 1 × placa UNO Arduino
- 1 × servo motor
- 1 × ULN2003 IC de condução
- Resistor de 1 × 10 K Ω

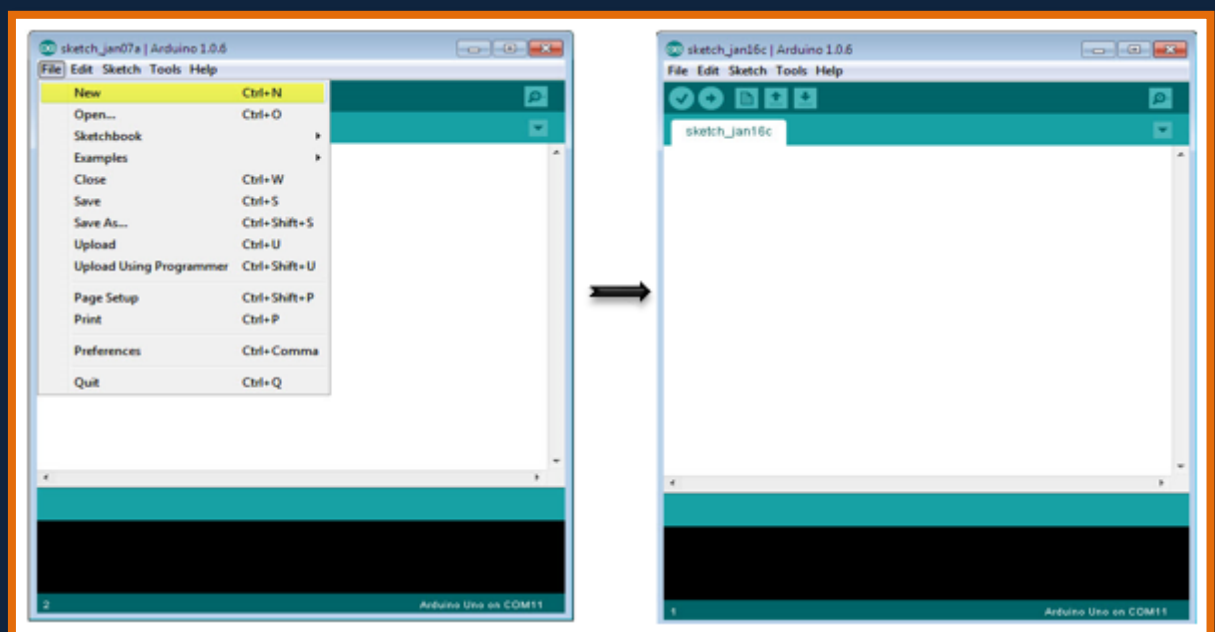
Procedimento

Siga o diagrama do circuito e faça as conexões conforme mostrado na imagem abaixo.



Esboço

Abra o software Arduino IDE no seu computador. A codificação na linguagem Arduino controlará seu circuito. Abra um novo arquivo de esboço clicando em Novo.



Código Arduino

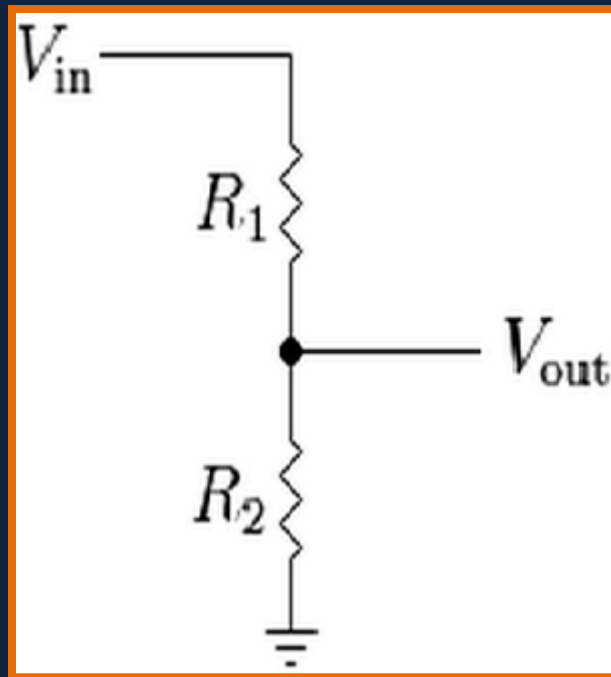
```
/* Controlling a servo position using a potentiometer (variable resistor) */  
  
#include <Servo.h>  
Servo myservo; // create servo object to control a servo  
int potpin = 0; // analog pin used to connect the potentiometer  
int val; // variable to read the value from the analog pin  
  
void setup() {  
  myservo.attach(9); // attaches the servo on pin 9 to the servo object  
}  
  
void loop() {  
  val = analogRead(potpin);  
  // reads the value of the potentiometer (value between 0 and 1023)  
  val = map(val, 0, 1023, 0, 180);  
  // scale it to use it with the servo (value between 0 and 180)  
  myservo.write(val); // sets the servo position according to the scaled value  
  delay(15);  
}
```

Código a Nota

Os servomotores têm três terminais - potência, terra e sinal. O fio de energia normalmente é vermelho e deve ser conectado ao pino de 5V no Arduino. O fio terra geralmente é preto ou marrom e deve ser conectado a um terminal do ULN2003 IC (10-16). Para proteger sua placa Arduino contra danos, você precisará de um driver IC para fazer isso. Aqui usamos o ULN2003 IC para acionar o servo motor. O pino de sinal é normalmente amarelo ou laranja e deve ser conectado ao pino número 9 do Arduino.

Conectando o potenciômetro

Um divisor de tensão / divisor de potencial são resistores em um circuito em série que escalam a tensão de saída para uma proporção específica da tensão de entrada aplicada. A seguir está o diagrama do circuito -



$$V_{out} = (V_{in} \cdot R_2) / (R_1 + R_2)$$

V_{out} é o potencial de saída, que depende da tensão de entrada aplicada (V_{in}) e dos resistores (R_1 e R_2) da série. Isso significa que a corrente que flui através de R_1 também irá fluir através R_2 sem ser dividida. Na equação acima, como o valor de R_2 mudanças, o V_{out} escalas em conformidade com respeito à tensão de entrada, V_{in} .

Normalmente, um potenciômetro é um divisor de potencial, que pode dimensionar a tensão de saída do circuito com base no valor do resistor variável, que é dimensionado usando o botão. Possui três pinos: GND, Sinal e + 5V, conforme mostrado no diagrama abaixo -



Resultado

Ao alterar a posição NOP da panela, o servo motor altera seu ângulo.

Considerações Finais

Espero ter lhe ajudado com esses pequenos tutoriais de projetos com arduino e introdução a robótica . Isso vai lhe ser útil em futuros projetos seja acadêmico ou profissional .

Seja você mesmo. “Nunca se compare com ninguém neste mundo. ...
Bill Gates