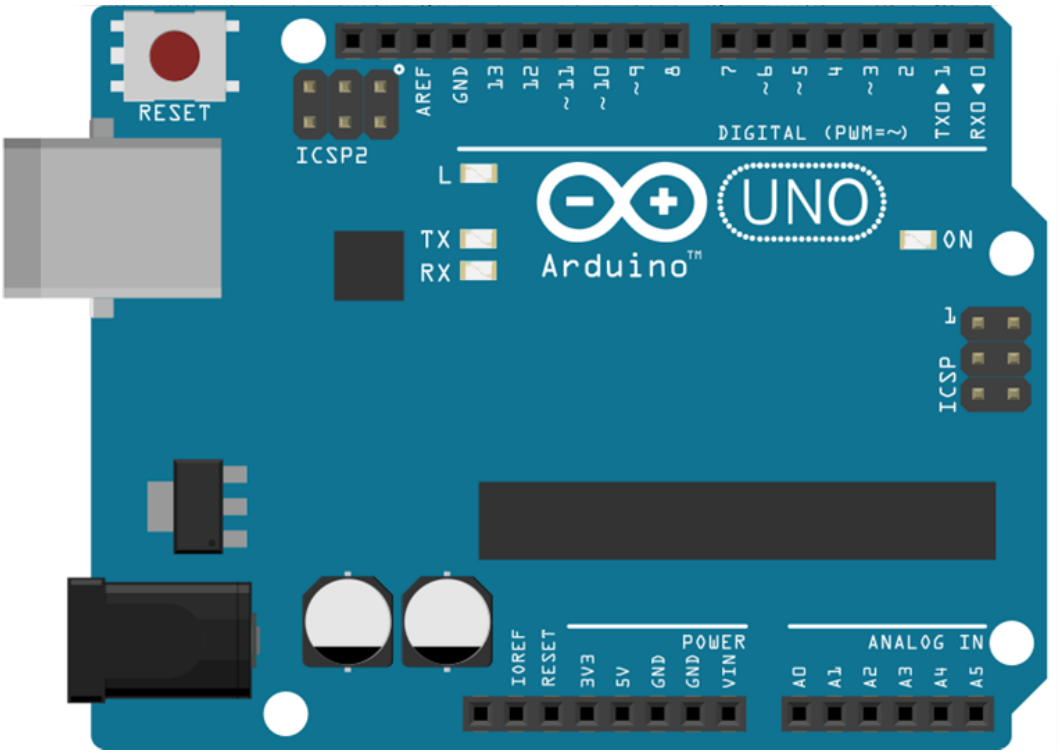
INTRODUÇÃO AO ARDUÍNO



GER

Grupo de Estudos em Robótica

ÍNDICE

Sobre o Grupo

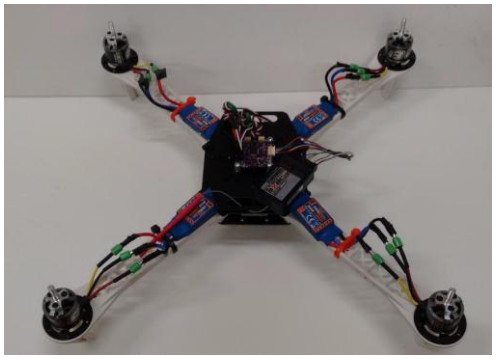
Criado em 2009 dentro do Ramo Estudantil IEEE, o GER, Grupo de Estudos em Robótica, é uma equipe de robótica autônoma da Faculdade de Engenharia Mecânica da Unicamp.

Atualmente o GER conta com 26 membros ativos, alunos da Engenharia de Controle e Automação, Engenharia Elétrica, Ciência e Engenharia da Computação.

O principal objetivo do GER é promover o estudo e aprendizado entre seus membros, em aréas relacionadas à robótica, o grupo estuda assuntos como: aplicação de técnicas de controle, processamento de imagem, visão computacional, simulação, sistemas multi-agente, entre outros.

O GER participa de competições de robótica e simulação, são elas:

- **Competição Brasileira de Robótica:** nas categorias IEEE VSSS (Very Small Soccer Size) e IEEE SEK (Standard Educational Kit)
- **The Freescale Cup:** competição de robôs seguidores de pista.
- **Torneio Juvenil de Robótica:** na categoria Sumô
- **Robocode:** competição de simulação de guerra de robôs. **O GER foi campeão em 2014!**



Além das competições, existem também projetos paralelos, como o desenvolvimento de um Drone e outros projetos pequenos de robótica e automação.

Introdução

O que é?

Arduíno é uma plataforma *open-source* de desenvolvimento que consiste em uma placa com microcontrolador e uma IDE (Integrated Development Environment), o software que permite ao usuário desenvolver o código usado pela placa. O grande diferencial do Arduíno é permitir a programação de um micro controlador usando uma linguagem de alto nível, possibilitando que iniciantes e leigos em eletrônica e programação possam desenvolver projetos relativamente complexos.

História

O projeto iniciou-se na cidade de Ivrea, Itália, em 2005, mais especificamente no Interaction Design Institute Ivrea. A equipe de fundadores era composta por Hernando Barragan, Massimo Banzi, David Cuartielles, Dave Mellis, Gianluca Marino, e Nicholas Zambetti. Na época, os estudantes do instituto utilizavam sistemas de prototipagem considerados caros. Como alternativa a isso, surgiu o Arduíno. Tal nome veio de um bar em Ivrea, onde alguns dos fundadores do projeto se encontravam.

Seu sucesso foi sinalizado com o recebimento de uma menção honrosa na categoria *Comunidades Digitais* em 2006, pela **Prix Ars Electronica**, além da marca de mais de 50.000 placas vendidas até outubro de 2008.

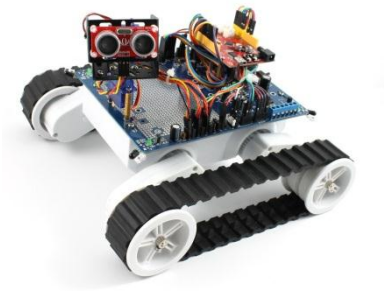
Comunidade

A comunidade que estuda e utiliza Arduíno é bem numerosa e acessível. É possível encontrar uma grande variedade de projetos e tutoriais através de sites, fóruns online e livros.

No próprio site do Arduíno existem tutoriais que mostram diversas funcionalidades da placa e como desenvolver projetos com ela.
(https://www.Arduíno.cc/en/Tutorial/HomePage)

Além disso, com o Arduíno, também é possível utilizar vários tipos de sensores e shields (placas de expansão de hardware) de maneira rápida e simples. Esses dispositivos são extremamente úteis em diversas aplicações eletrônicas.

Exemplos

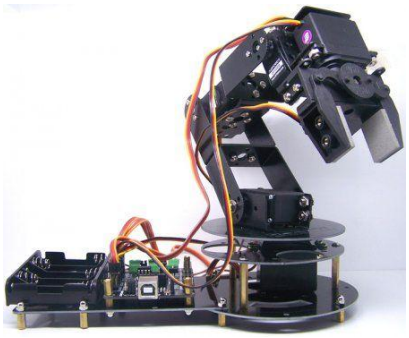


Devido a sua grande comunidade o Arduíno se tornou extremamente popular nos circulos **DIY** (Do it yourself), vários exemplos de projetos podem ser encontrados na internet:

Um exemplo clássico da robótica é o robô seguidor de linha, que utilizando sensores de luz consegue percorrer uma pista desenhada no chão. Um projeto desse tipo utilizando Arduíno pode ser visto em http://goo.gl/ePkyDJ.

O Arduíno também pode ser utilizado em projetos mais complexos, como para transformar uma bateria utilizada em games (como Guitar Hero) em uma bateria eletrônica comum. (Veja o projeto em http://goo.gl/5kZAXK).

Além disso, existem outros projetos interessantes como um braço robótico controlado via Bluetooth (http://goo.gl/jIhZCc) ou até um quadricóptero (http://goo.gl/zSHlyF).



Assim, há uma grande variedade de projetos divertidos e úteis que podem de ser trabalhados com Arduíno. Cada um tem seu nível de complexidade e muitos não são tão fáceis de serem desenvolvidos. Mas o que não se pode negar é que o Arduíno é uma plataforma acessível e poderosa.

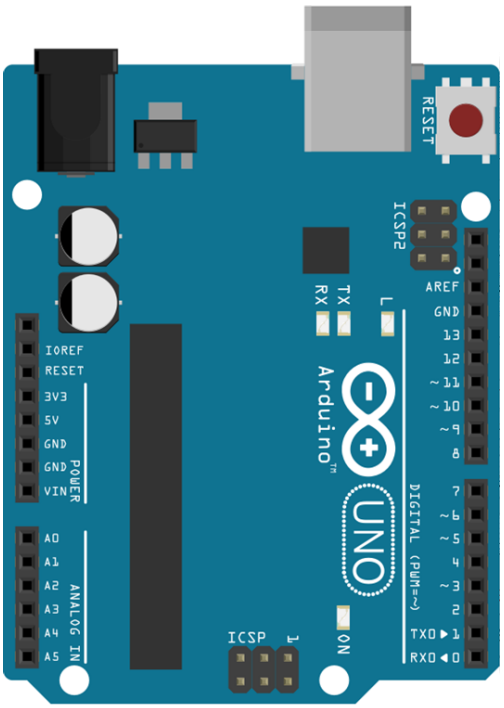
Sites com tutoriais de projetos com Arduíno (em inglês):

- http://playground.Arduíno.cc/Projects/Ideas
- www.instructables.com/id/Arduíno-Projects
- http://duino4projects.com

A placa

Existem diferentes modelos de placas que utilizam a plataforma Arduíno, para este curso utilizaremos o Arduíno Uno, modelo mais simples de básico.

Especificações



Microcontrolador: ATmega328
Tensão de operação: 5V
Tensão de Entrada (recomendado): 7-12V
Tensão de Entrada (limites): 6-20V
Pinos digitais I/O: 14 (6 tem suporte a PWM)
Entradas Analógicas: 6
Corrente nos pinos digitais: 40 mA
Flash Memory: 32 KB (ATmega328)
SRAM: 2 KB
Clock: 16 MHz

O que um iniciante precisa saber dessas informações?

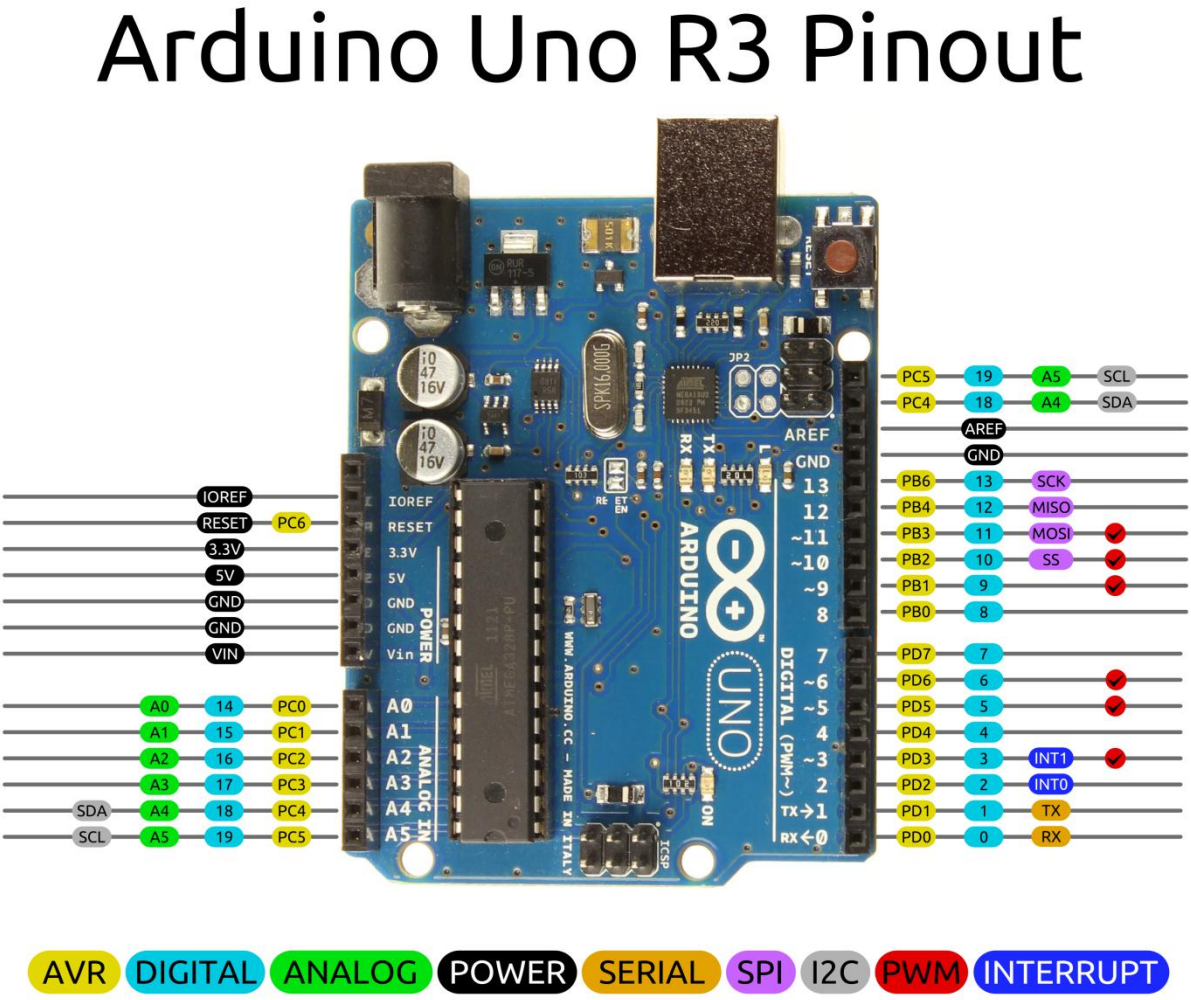
- Que ele não deve conectar o arduino a uma fonte maior que 12V. Um computador já estabelece automaticamente uma tensão segura para a placa a partir do USB, mas cuidado ao conectá-la a uma bateria ou tomada pela entrada de energia, sempre cheque sua voltagem. Tensões maiores podem danificar a placa e o computador em que ela está conectada.
- Pinos analógicos trabalham com tensões que variam de 0V até 5V. Pinos digitais trabalham com o valor de 0V (desligado) ou 5V (ligado). Você aprenderá a programar a placa de modo que o pino escolhido exerça a tensão desejada para realizar uma tarefa (como os exercícios propostos a neste guia).
- Preste atenção na corrente, se você colocou 5V no pino, certifique-se que seu circuito tem uma resistência equivalente de pelo menos 125 ohm (comprove por $V = R * I$).

Portas

Esteja atento a esta imagem para realizar os exercícios propostos.

Perceba no canto esquerdo inferior os pinos analógicos (A0 até A5) e no canto direito os pinos digitais (1 até 12). Cada um tem um respectivo número, o qual você irá se referir em seu código para ativar aquele determinado pino. No canto esquerdo superior têm-se os pinos de tensão fixa (GND é o pino conectado ao terra, ou seja, 0V).

Referências:
https://www.arduino.cc/en/Main/arduinoBoardUno

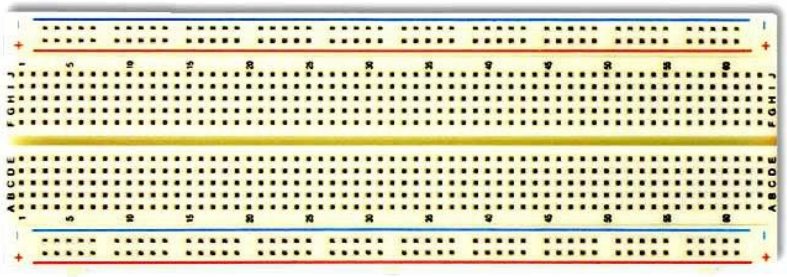


Entradas Analógicas e Sensores

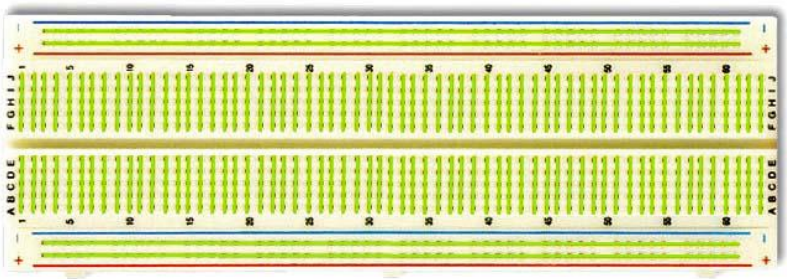
Protoboard

Antes de começar a programar a placa, é necessário aprender a usar uma ferramenta extremamente util na prototipagem de circuitos: a ***Protoboard***.

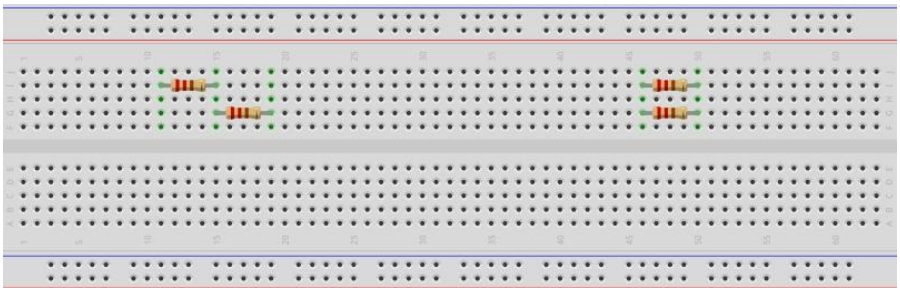
A protoboard é uma placa com vários furos para fazer conexões entre componentes de um circuito elétrico, sendo, normalmente, usada para fazer testes com circuitos pequenos devido à grande facilidade de manuseio. O modelo utilizado pode ser visualizado na imagem abaixo.



A placa é espelhada e tem duas partes: a central, para conexões entre os componentes elétricos, e a periférica, geralmente usada para conectar a alimentação e a terra. Na parte central os furos das colunas são conectados entre si, mas não se conectam com outras linhas. Já na parte periférica os furos das linhas são conectados entre si e não se conectam com os furos das colunas. Como pode ser visto na imagem abaixo.



Para conectar um resistor, por exemplo, basta colocar os seus dois terminais em colunas diferentes. Se uma ligação de dois resistores em série for necessária, basta conectar os terminais de um resistor em colunas diferentes e os terminais do outros resistor em uma das colunas já utilizadas e em outra livre. No caso de uma ligação em paralelo, é necessário ligar as pontas de cada resistor nas mesmas colunas. Uma imagem exemplificando as conexões pode ser vista a seguir.



Projeto 0: *Hello World*

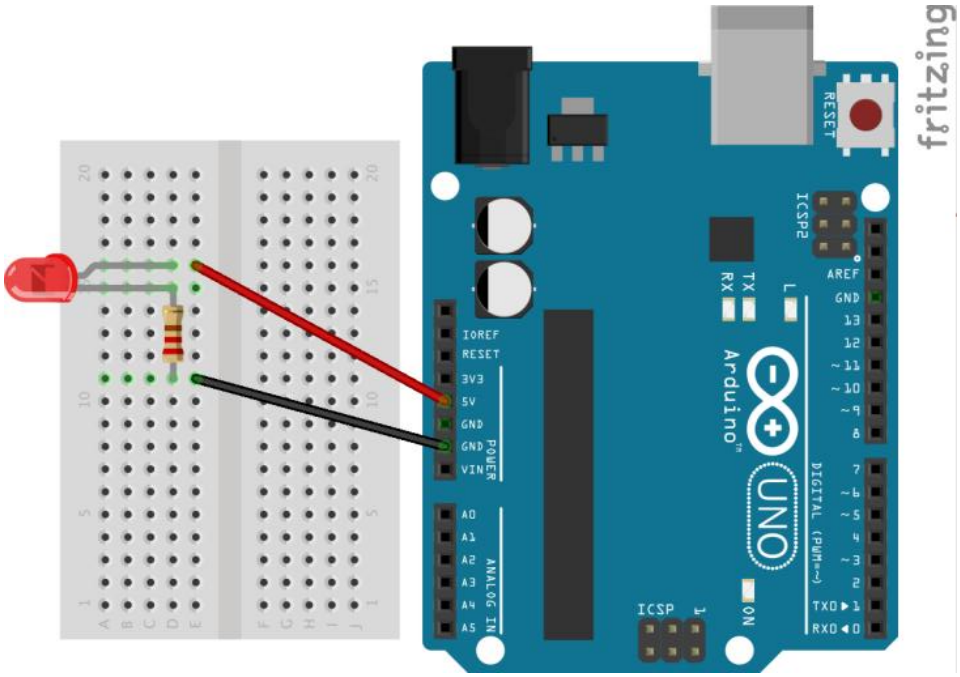
Façamos um pequeno projeto para exercitar o uso da Protoboard e se familiarizar com algumas das portas do Arduíno.

Usaremos as portas de **5V** e **GND** do Arduíno para acender um LED. Estas portas estão localizadas perto da região denominada **Power**. Temos as seguintes portas:

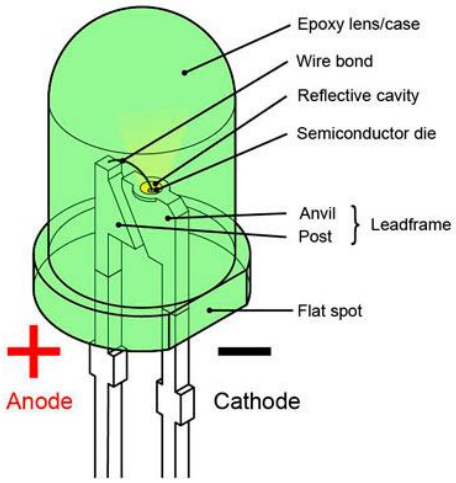
- **5V** : Fornece um sinal constante de 5V
- **3.3V** : Fornece um sinal constante de 3,3V
- **GND** : Fornece um sinal constante de 0V (sinal baixo)
- **Vin** : Usada para alimentar o Arduíno
- **Reset** : Usado para resetar o Arduíno de forma externa
- **IORef** : Fornece a tensão correspondente ao Input/Ouput digital da placa, no caso do Arduíno Uno 5V.

Primeiramente, certifique-se que o Arduíno esteja desligado (desconectado do computador e com nenhum dos seus LEDs acessos), neste projeto especifico isto não fará muita diferença, mas é uma boa pratica (por motivos de segurança) sempre desligar o Arduíno quando for alterar o seu circuito.

Agora, faça o circuito conforme a figura abaixo, e depois, ligue o Arduíno na porta USB do seu computador. ***Não esqueça de colocar o resistor, caso contrário o LED pode queimar.***



Caso o LED não acenda, certifique-se ele esta corretamente conectado. O LED, **L**ight **E**mitting **D**iode, como o próprio nome já diz, é um diodo, ou seja, possui uma polaridade correta. Você deve conectar a alimentação do seu circuito (5V) ao ânodo do LED (caracterizado pela perna maior) e o terra do circuito (GND) ao catódo do LED.

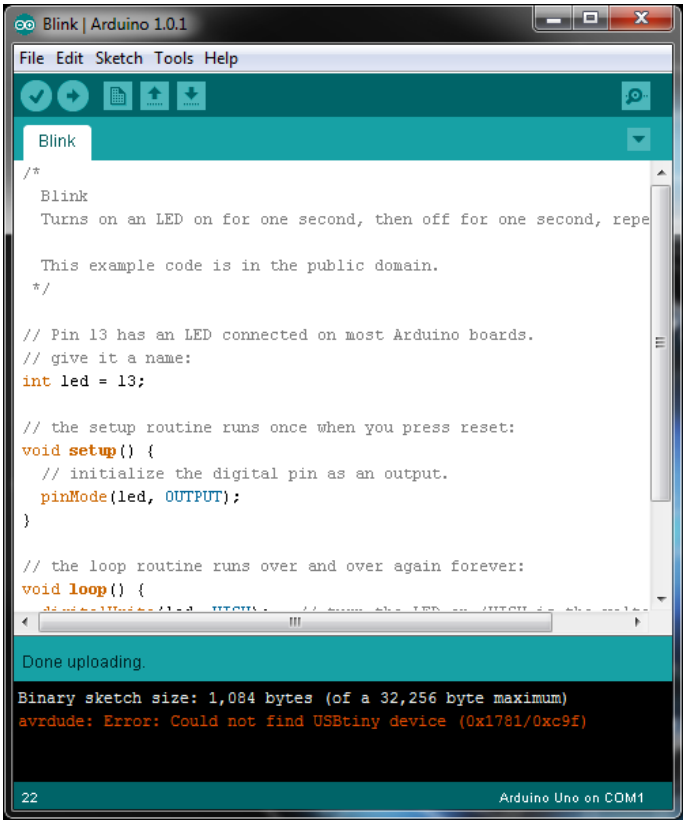


Desafio Adicional:

Usando apenas duas portas do Arduíno (5V e GND), conecte dois LEDs em série, em paralelo com outros dois LEDs em série. Tente fazer este desafio com apenas dois resistores e três jumpers.

Estrutura básica de um código

Uma das facilidades propostas pelo Arduíno é uma IDE simples e fácil de usar. Ela possui vários exemplos prontos de código, um monitor serial e uma ferramenta que simplifica a adição de novas bibliotecas.



Antes de começarmos a programar é necessário que nos certifiquemos que a IDE esta devidamente configurada para a nossa placa e para a entrada USB que estamos utilizando.

Para isso vá em: **Tools > Board** e se certifique de que a opção **Arduino Uno** esteja selecionada. Depois vá em **Tools > Serial Port** e se certifique de que a porta correta está selecionada (depende do seu computador).

Agora podemos começar a programar. Provavelmente o maior atrativo do Arduíno é que ele permite a programação de um micro-controlador com linguagem de alto nível. Neste caso, utilizaremos C++ com uma série de bibliotecas prontas oferecidas. Mais adiante, iremos abordar um pouco a questão de Orientação a Objeto em um projeto do Arduíno.

Um código usado no Arduíno possui uma estrutura básica que sempre deve ser seguida.

Basicamente, todo código deve possuir duas funções:

- **void setup()** : Esta função é chamada apenas uma vez, no inicio da execução do código, ela deve ser usada para setar qualquer componente ou váriavel do código, analogamente à um método construtor de uma classe.
  Obs: A função pode ser chamada pelo usuário posteriormente porém isto não é uma boa prática, por convenção a função setup() só deve ser usada para inicialização.

- **void loop()** : Após a função setup(), o código irá chamar esta função loop *ad infinitum*. É interessante frisar que projetos de eletrônica e robótica normalmente funcionam em cima de um loop infinito como é o caso do Arduíno. Esta função é analóga a função main() de um código em C.

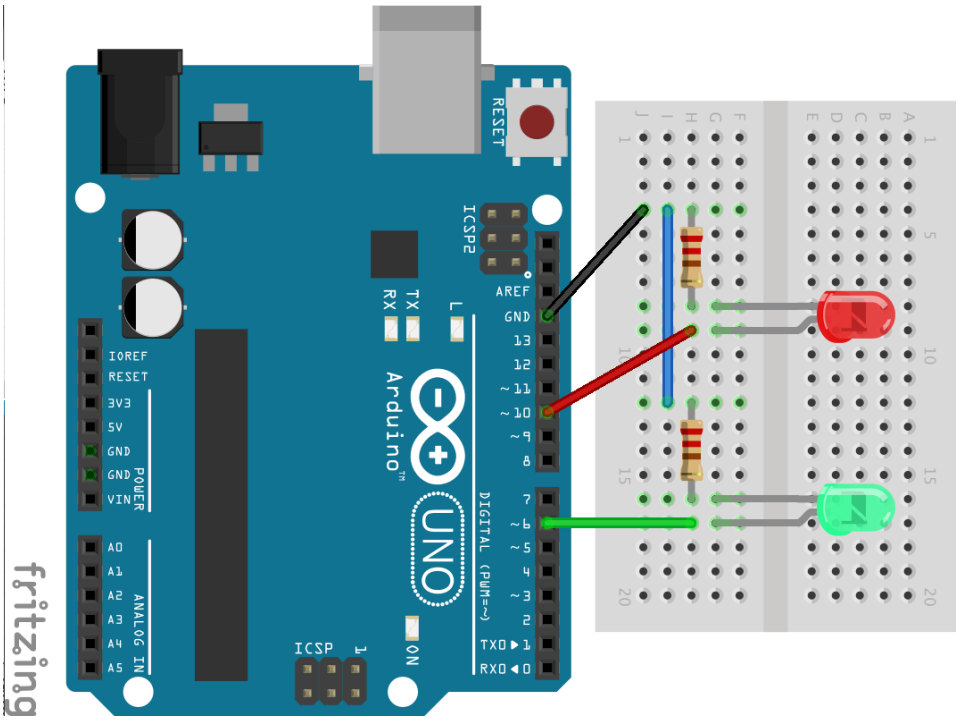
Temos aqui um exemplo de código, no proximo projeto explicaremos o que cada uma dessas funções aqui usadas faz. Este código especifico é um dos exemplos que podem ser acessados na IDE em **File > Examples**.

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

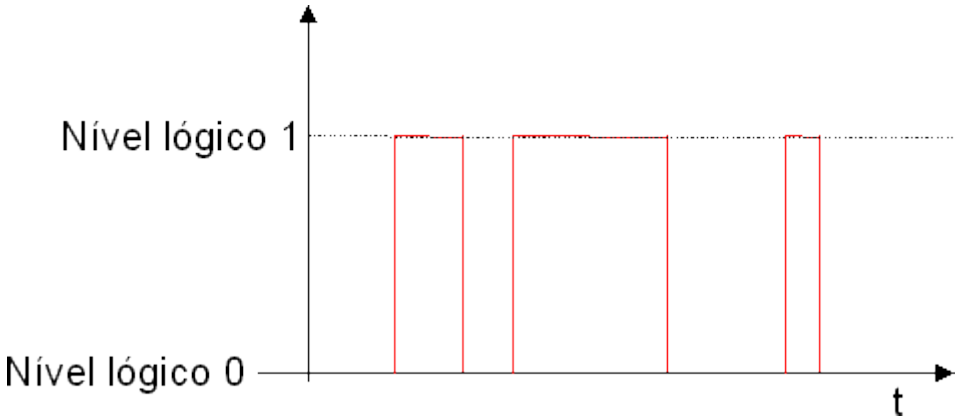
Projeto 1: *Piscar LED*

Iremos escrever um código básico para fazer com que dois LEDs pisquem alternadamente, ou seja, quando um liga o outro desliga.

O circuito para este projeto está na imagem a seguir:



Utilizamos os pinos digitais **10** e **6**. O Arduíno Uno possui 14 pinos que podem ser usados como saídas digitais, eles podem ser usados para emitir um sinal de 5V (HIGH) ou de aproximadamente 0V (LOW).



Perceba que alguns pinos possuem um **~** ao lado do número, isto será abordado mais tarde. Além disso os pinos **0** e **1** possuem ao seu lado as palavras **RXD** e **TXD**, isto é utilizado para comunicações com o protocolo serial, isto não será abordado neste curso porém tente não utilizar estas portas.

Este é o código utilizado:

```
/* Projeto 1, piscar LED */

//Primeiramente criamos duas constantes que definem as nossas portas
int led1 = 10;
int led2 = 6;

void setup() {
  // Iremos setar as portas como output na função setup, que só é chamada uma vez
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);

}

void loop() {
  digitalWrite(led1, HIGH);  // Liga o primeiro LED
  digitalWrite(led2, LOW);   // Desliga o segundo LED
  delay(1000);               // Espera por 1 segundo
  digitalWrite(led2, HIGH);  // Liga o segundo LED
  digitalWrite(led1, LOW);   // Desliga o primeiro LED
  delay(1000);               // Espera por 1 segundo
}
```

Neste código utilizamos três funções importantes, são elas:

- **pinMode**(ledPin, OUTPUT) : Define o pino ledPin, no caso do exemplo os pinos 6 e 10, como OUTPUT (saida)

- **digitalWrite**(ledPin, HIGH/LOW) : Manda um valor alto ou baixo para uma saída digital.

- **delay(1000)** : Pausa a execução do código, nesse caso a pausa dura 1000ms

Desafio Adicional:

Escreva um código que faça um LED piscar com diferentes frequências de forma iterativa, ou seja, o LED começa a piscar lentamente, aumenta a frequência gradualmente (até que o usuário não consiga perceber que está piscando) e diminua a frequência até a frequência inicial.

Desafio Adicional++:

Escreva um código que faça dois LEDs piscarem com diferentes frequências de forma iterativa e alternada, ou seja, um Led começa a piscar lentamente, aumenta a frequência gradualmente (até que o usuário não consiga perceber que está piscando) e diminua a frequência até o estado inicial, o outro LED faz o contrário, começa a piscar de forma rápida e diminui a frequência até ficar constantemente ligado, depois volta a aumentar até a frequência inicial.

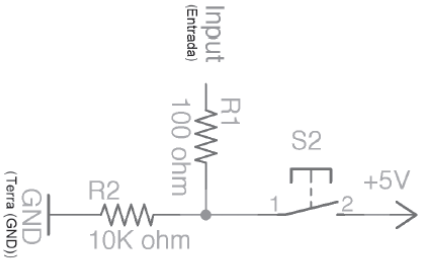
Projeto 2: *Push-Button*

No projeto anterior usamos as portas digitais para emitir um sinal para o nosso circuito, agora iremos emitir um sinal do nosso circuito para o Arduíno.

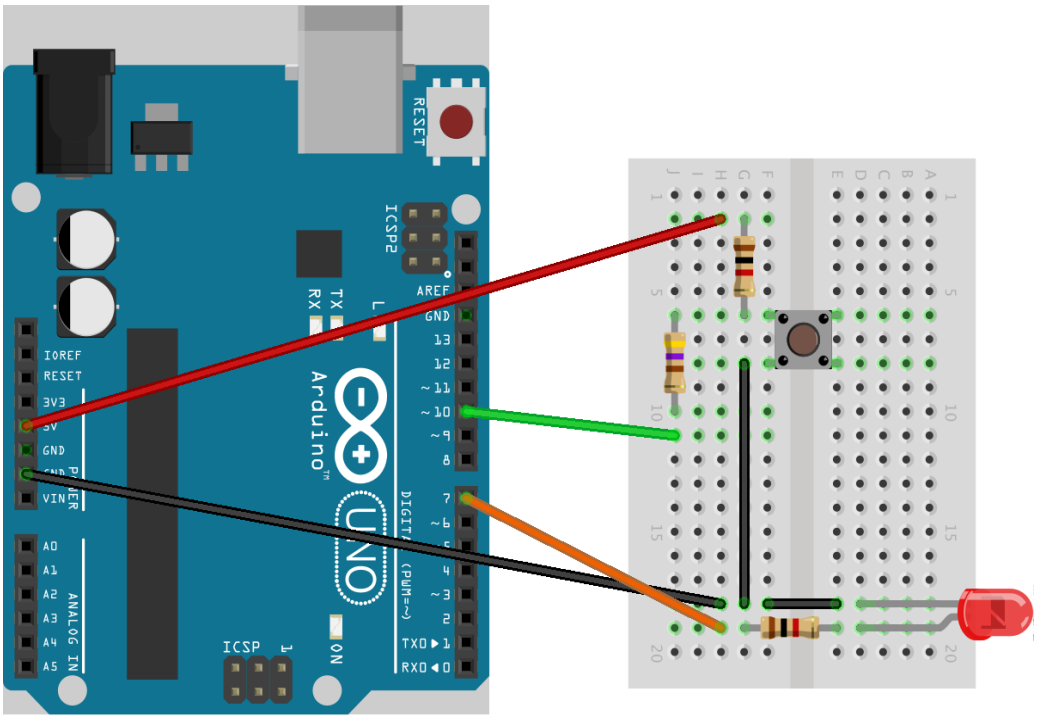
Para que possamos observar o uso de entradas digitais, iremos usar um *push-button*.

Este componente possui um funcionamento bem simples, quando apertado ele fecha uma chave, criando um curto entre dois circuitos isolados, possibilitando que o nosso circuito interaja diretamente com o usuário.

Neste projeto iremos usar um *push-button* para ligar um LED quando apertado uma vez, e desliga-lo quando apertado mais uma vez.



Este é o circuito do projeto, perceba que para se utilizar o *push-button* devemos usar um resistor, isto se deve ao fato de que o botão possui uma pequena resistência interna (bem menor que a do resistor), logo quando apertarmos o botão um curto será criado e a corrente irá deixar de ir para o nosso Arduíno.



Este é o código do projeto:

```
/* Projeto 2, push-button */

//Primeiramente criamos duas constantes que definem as nossas portas
int led = 7;
int button = 10;

//Variavel auxiliar
boolean toogle = false;

void setup() {
  // Iremos setar a porta do LED como output pois é uma saida
  pinMode(led, OUTPUT);

  //Iremos setar a porta do botao como input pois é uma entrada
  pinMode(button, INPUT);

}

void loop() {
  //Checamos o estado de nossa variavel auxiliar,
  //que define se o LED deve estar desligado ou não
  if(toogle)
  {
    digitalWrite(led,HIGH);
  }else
  {
    digitalWrite(led,LOW);
  }

  //Iremos ler o botão, caso ele esteja apertado
  if(digitalRead(button))
  {
    //Alteramos o valor da nossa variavel auxiliar,
    //mostrando que podemos alterar o LED
    toogle = !toogle;

    //Entramos em um loop infinito por segurança, caso o contrario
    //o LED sempre mudará se o usuario ficar com o botão pressionado
    while(digitalRead(button))
    {}
  }
```

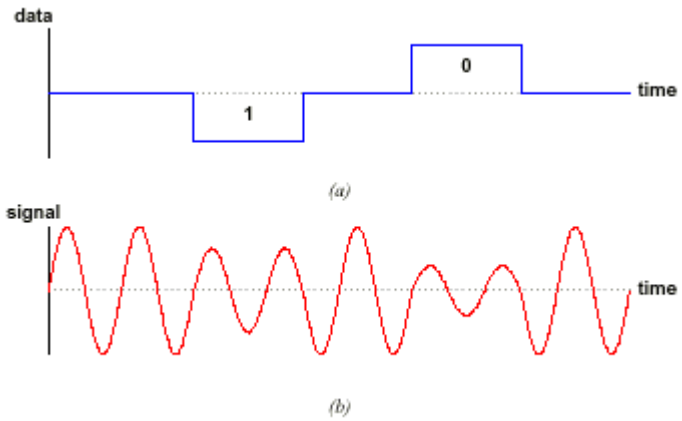
Desafio Adicional:

Inicialmente faça o LED piscar como nos ultimos projetos, depois faça com que cada vez que o usuário apertar o botão a frequência fique mais rápida, quando ela atingir seu valor máximo, faça com que cada “aperto” do botão diminua a frequência. Não esqueça que a frequencia só deve mudar após o usuário ter soltado o botão.

Sinais Analógicos: *PWM*

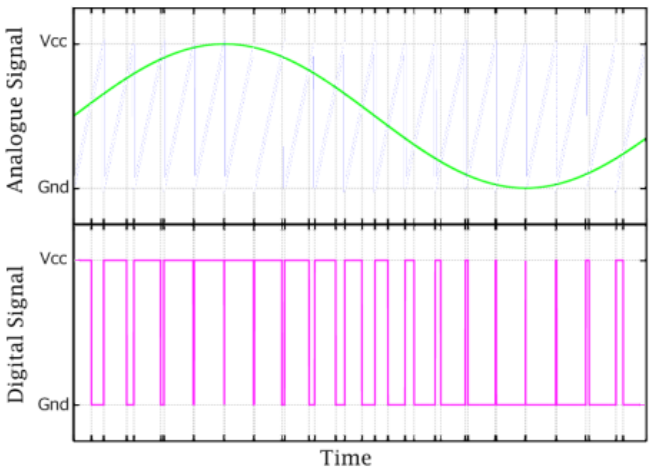
Até agora, todos os projetos lidaram somente com valores digitais, **0V (falso)** e **5V (verdadeiro)**, porém em várias aplicações precisamos trabalhar com um intervalo maior de possiveis valores, para isso precisamos trabalhar com **Sinais Analógicos**, a imagem abaixo mostra a diferença entre os sinais.

Observe que o sinal digital (azul) só admite dois valores, enquanto o sinal analógico (vermelho) permite um número muito maior de valores. O quão preciso são os valores do sinal analógico depende do seu hardware e software. No caso do Arduíno as saídas possuem uma precisão de 8 bits [0,255], enquanto as entradas posuem uma precisão de 10 bits [0,1023].



Porém o Arduíno não possui como emitir sinais verdadeiramente analógicos, porém seus componentes permitem o uso de **PWM** para sanar este problema.

Pulse **W**idth **M**odulation consiste em mandar vários sinais digitais, com diferentes larguras de banda em uma frequencia relativamente alta, de forma que o valor médio do sinal em um dado instante se assemelha a um sinal analógico. A imagem abaixo ilustra o PWM.



Projeto 3: PWM com Motor DC

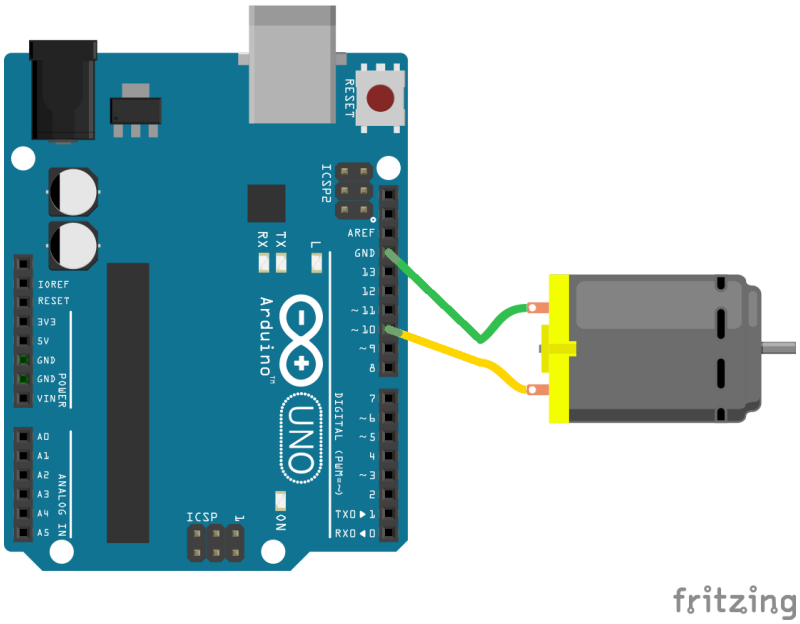
Iremos demonstrar o funcionamento de mais um simples, porém muito útil, componente, o **Motor DC**. Com apenas dois terminais, o motor permite uma integração de um sistema elétrico com um sistema mecânico.

Neste projeto, iremos alterar a intensidade de rotação do motor de forma iterativa.

Os motores utilizados neste curso possuem uma tensão muito baixa, logo não há nenhum risco para o Arduíno, porém lembre-se sempre: **Todo motor é também um gerador**, portanto é recomendado que se coloque um diodo em série com o motor, pois depois que ele for desligado, devido a inércia, ele continuará girando, criando assim uma tensão que pode danificar o seu Arduíno.

Este é o circuito do projeto, observe que diferente do LED, o motor não possui uma polaridade correta, porém caso troquemos as portas de seus terminais podemos observar que o sentido de rotação do motor irá se inverter.

Também observe que o motor esta ligado em uma porta com o símbolo **‘~’** escrito ao lado de seu número, isto denota que a porta é capaz de emitir um sinal PWM, o Arduíno Uno possui 6 portas com esta capacidade (3,5,6,9,10,11).



O código do projeto não apresenta grandes complicações, porém existe uma função nova que ainda não utilizamos.

```
/* Projeto 3 - PWM com Motor DC*/

//Iremos colocar o motor na porta 10
int motorPin = 10;

void setup()
{
  //Setamos a porta do motor como Saida
  pinMode(motorPin, OUTPUT);
}

void loop(){

  //Aumentaremos gradativamente a potencia do motor
  for(int i=0; i<255; i++)
  {
    analogWrite(motorPin, i);
    //Colocaremos um pequeno delay para que o usuario possa perceber a diferença
    delay(10);
  }

  //Diminuiremos gradativamente a potencia do motor
  for(int i=255; i>0; i--)
  {
    analogWrite(motorPin, i);
    //Colocaremos um pequeno delay para que o usuario possa perceber a diferença
    delay(10);
  }

}
```

- **analogWrite(motorPin, i)** : Envia um valor ‘i' para uma saída digital com suporte a PWM. O valor está contido no intervalo [0,255].

Desafio Adicional:

Use um push button para alterar a intensidade de rotação do motor, quanto mais tempo se aperta o botão mais rápido o motor gira. Ao soltar o botão a potência do motor cai gradativamente.

Projeto 4: Servomotores

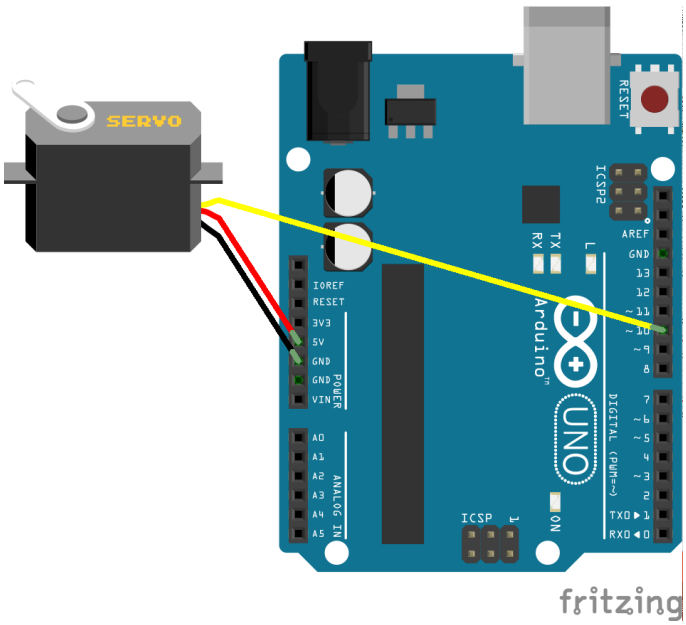
Como discutido anteriormente, o Arduíno possui uma série de bibliotecas prontas que facilitam o uso de certos componentes que normalmente necessitam de um grande volume de código. Isto aliado à **Programação Orientada a Objeto** facilita bastante o desenvolvimento de projetos com poucas linhas de código.

Isto é ilustrado com o uso de um **Servomotor**. Um componente que também faz a ponte entre um sistema elétrico e um sistema mecânico, porém com uma maior precisão. Enquanto o motor DC permite que mudemos a intensidade de rotação, o servomotor permite que giremos um ângulo já definido.

Neste projeto iremos alterar o ângulo de rotação de um servo de maneira iterativa.

Segue o circuito do projeto, note que o servo possui três fios, são eles:

- **VCC:** Fio vermelho, usado para alimentar o servo.
- **GND:** Fio preto ou marrom, é o terra do servo
- **Signal:** Fio normalmente amarelo, é o que iremos usar para mandar o sinal para o servo, note que o sinal deve ser mandado por uma porta com suporte a PWM.



Segue adiante o código utilizado no projeto. Existe uma série de diferenças deste código com o código anterior:

```
#include "Servo.h"

//Usaremos o pino 10 para conectar o servo
#define servoPin 10

//Instancia da classe Servo
Servo servo;

void setup(){
  servo.attach(servoPin); //Metodo de inicializacao de um Servo
}

void loop(){
  //Iteracao do angulo
  for(int i=0; i<=180; i++)
  {
    servo.write(i); //Manda um angulo para o servo, que vai de 0 ate 180
    delay(5);
  }

  for(int i=180; i>=0; i--)
  {
    servo.write(i);
    delay(5);
  }
}
```

Primeiramente, note que utilizamos uma outra biblioteca: **Servo.h**, ela é uma das bibliotecas já inclusas por default na IDE, ela facilita o uso de servosmotores devido a classe **Servo**. Instanciamos um objeto desta classe e utilizamos métodos que diminuem consideravelmente o volume de código, os métodos são:

- **servo.attach() :** método inicializador do Servo
- **servo.write(int a) :** rotaciona o servo em um ângulo "a", contido no intervalo [0,180]

Desafio Adicional:

Use o push button para girar um ângulo fixo do servo (15º) de maneira suave. Quando ele chegar no seu limite (180}), ele deve girar no sentido oposto, ou seja, -15º a cada aperto do botão.

Entradas Analógicas e Sensores

Já aprendemos a mandar um sinal analógico para nosso circuito, veremos agora que o inverso é tão simples quanto.

O Arduíno possui 6 entradas analógicas, na parte da placa denotada por **Analog In**. Como já citado anteriormente, a entrada analógica tem uma precisão de 10 bits, ou seja, assume valores no intervalo [0,1023].

Para que possamos estudar as entradas analógicas iremos usar sensores, componentes que permitem que recolhamos informação do meio que o nosso sistema está inserido e transformar estas informações em dados.

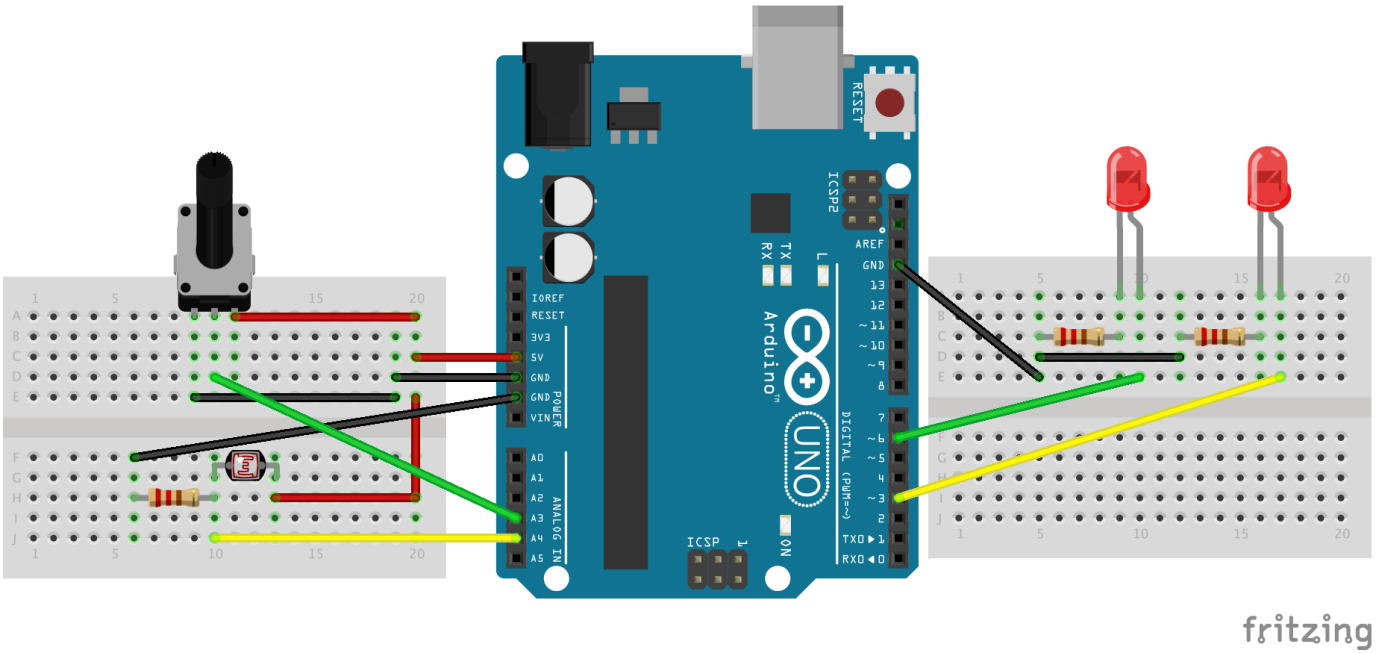
Para os nossos proximos projetos iremos usar três componentes diferentes:

- **Potênciometro:** Não é um sensor propriamente dito, e sim um resistor com resistência variavél.
- **LDR:** Um resistor cuja resistência se altera dependendo da luminosidade
- **Sonar:** Sensor que detecta barreiras fisicas na sua frente, não iremos usar uma entrada analógica.

Projeto 5: Sensores

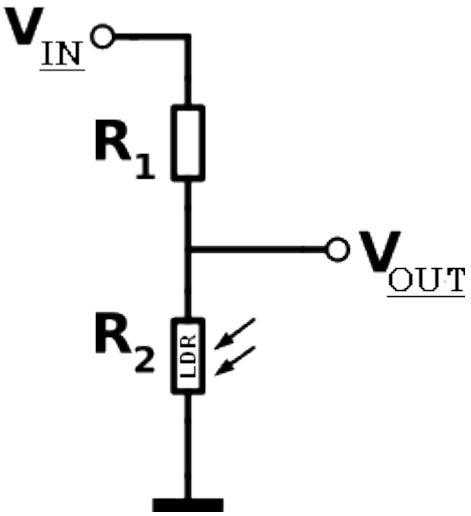
Para este projeto iremos utilizar o **LDR** e o **Potenciômetro**, para alterar a intensidade da luz de um LED, caso queira troque o LED pelo motor DC.

Segue o circuito usado no projeto.



Observe que tanto o LDR quanto o Potenciômetro, estão ligados a uma fonte de 5V, isto se dá porque ambos são sensores resistivos que não deixam de ser **divisores de tensão**, a tensão lida pelas portas do Arduíno é alterada dependendo da resistência de cada componente.

O diagrama a seguir ilustra de forma melhor o funcionamento dessa configuração de sensores e resistores:



O código deste projeto não é complexo:

```
/* Projeto 5, Sensores*/

#define LED1 6
#define LED2 3

#define LDR 4
#define Poten 3

void setup()
{
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);

  //Nao precisamos setar os sensores pois estarão
  //em portas que só devem ser usadas como INPUT
}

void loop()
{

  //Como a precisao do sensor é de 10 bits, iremos dividir por 4
  //para conseguir valores na faixa de [0,255]
  int LDR_value = analogRead(LDR)/4;
  int Poten_value = analogRead(Poten)/4;

  //Escrevemos o valor no LED e esperamos um breve momento entre leituras
  analogWrite(LED1, LDR_value);
  analogWrite(LED2, Poten_value);
  delay(50);

}
```

A única função nova que estamos usando é:

- **analogRead(int a):** Retorna o valor lido pela porta analógica, este valor está contido no intervalo [0,1023].

Observe que ao passar o valor lido pelo sensor para os LEDs, devemos dividir o valor por 4 para “converter” o para 8 bits, perdemos um pouco de precisão mas para fins didáticos isto não causa nenhum tipo de problema ao projeto.

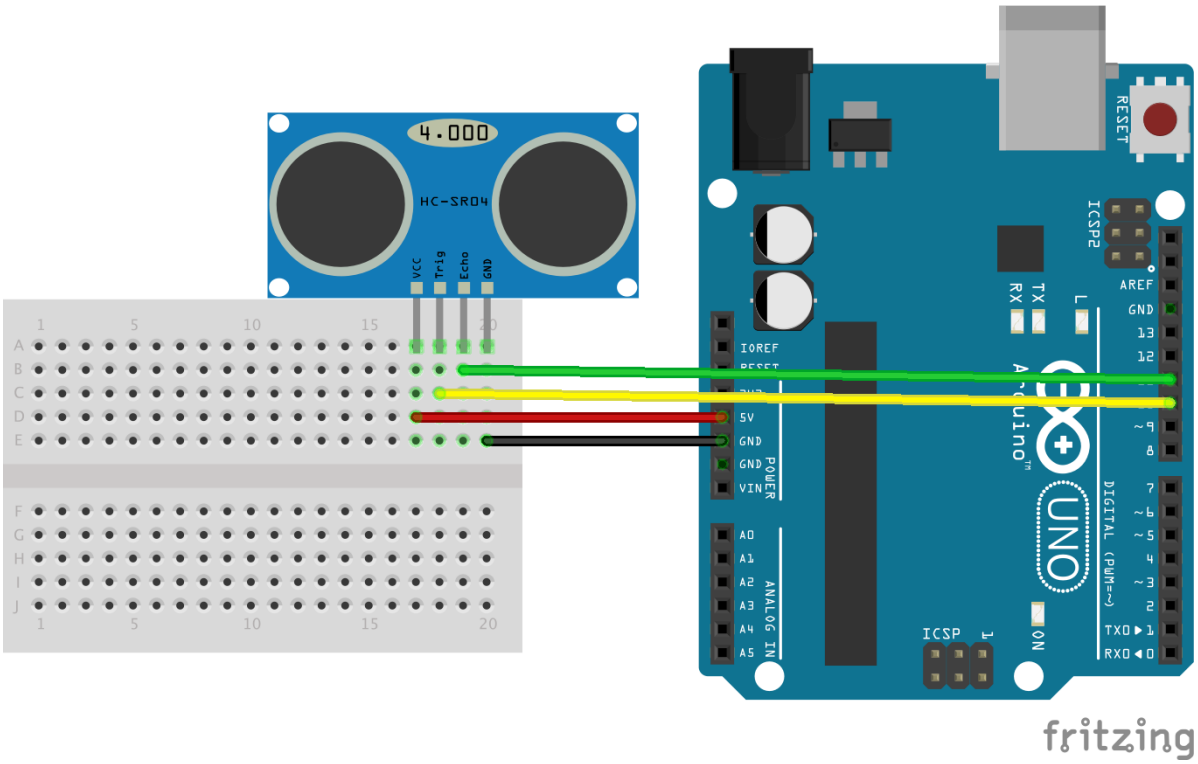
Projeto 6: Mandando e Recebendo Strings

O Arduíno permite o uso de comunicação serial pela sua porta USB. Além disso, métodos de classe da linguagem e o monitor serial da internet facilitam o processo de mandar e receber dados do Arduíno para outros sistemas, como por exemplo um computador.

Neste projeto iremos usar comunicação Serial para “ligar” um sonar e ler quantos centimetros estão sendo lidos pelo sensor.

O circuito é extremamente simples, note que o sensor possui 4 terminais, são eles:

- **VCC :** Alimentação
- **GND :** Terra
- **Echo :** Sinal que iremos mandar para o sensor
- **Trig :** Sinal que iremos ler do sensor



O código do projeto é relativamente diferente do que fizemos anteriormente, porém ainda é relativamente simples. Para facilitar o projeto iremos usar uma nova biblioteca **Ultrasonic.h**, ela deve estar junto com o arquivo desta apostila, caso contrário você pode baixa-la neste link.

Para adicionar a biblioteca vá ,na IDE, em **Sketch > Import Library > Add Library...**, selecione o arquivo .zip. Depois vá em **Sketch > Library > Ultrasonic**.

Com a biblioteca adicionada, podemos escrever o nosso código:

```
/* Projeto 6 - Mandando e Recebendo Strings */

// Inclusão de bibliotecas.
#include <Ultrasonic.h>   // inclui biblioteca de manipulação de servos motores.

#define echoPin 10 //Pino 10 recebe o pulso do echo
#define trigPin 11 //Pino 11 envia o pulso para gerar o echo

//Instancia do nosso sensor
Ultrasonic ultrasonic(11,10);

void setup(){
  Serial.begin(9600); // Inicializa a comunicação serial com baud rate de 9600
  pinMode(trigPin, OUTPUT);  // define o pino triger como saída.
  pinMode(echoPin, INPUT);  // define o pino echo como entrada.
}

void loop(){

  //String de input, guardaremos o buffer do serial
  String input = "";

  //Lemos o a entrada serial, caracter por caracter
  while(Serial.available()>0)
  {
    input += (char) Serial.read();
    delay(5); //Pequeno delay por seguranca
  }

  if(input == "on")
  {
  Serial.print(ultrasonic.Ranging(CM)); //Exibe a medição do sonar em cm
  Serial.println( "cm" );// imprime o centímetro no final
  delay(100); //espere 1 segundo pra calcular novamente por segurança
  }
}
```

Este código apresenta uma série de conceitos novos, são eles:

- **ultrasonic(11,10)** : Assim como instanciamos, agora iremos instanciar um objeto da classe Ultrasonic.
- **Serial.begin(9600)** : Metódo de classe, da classe estática Serial, tal classe possibilita o uso do monitor serial da IDE, este método inicializa a comunicação com um baud-rate de 9600
- **String input = “”** : Criamos um objeto da classe string para facilitar a leitura de caractéres
- **Serial.available()** : Retorna true se o buffer do serial possuir um caractér esperando para ser lido.

- **Serial.read()** : Lê um caractér do serial, note que precisamos fazer um *type casting* para char, pois o método retorna um byte.
- **Serial.print** : Manda uma cadeia de caractéres para ser impressa no nosso terminal.

Desafio Adicional:

Note que eventualmente o sensor nos dá um valor absurdo (algo em torno dos 3000cm), modfique o código para impedir que isso aconteça e que possamos fazer uma leitura mais precisa.

Desafio Adicional++:

Use tudo que aprendeu até agora no curso para controlar três atuadores diferentes (LED, motor DC, servomotor) com três diferentes sensores (LDR, potenciometro, sonar). Porém o usuário deve escolher qual sensor controla cada atuador na execução do código, isto deve ser digitado pelo usuário no teclado, por exemplo:

Caso o usuário digite: *Servo* > *LDR*.
O Servomotor deve girar baseado na leitura do LDR.

Extra: Orientação a Objeto

Como já discutido anteriormente, o Arduíno permite que usemos Orientação a Objeto para facilitar o desenvolvimento de projetos. A sintaxe para a definição de classes é semelhante à do **C++,** observe a definição da classe Led:

```
class Led{

  private:
   byte pino;
   boolean estado;

  public:
   Led(byte p){
     setaPino(p);
   }

   void setaPino(byte pin){
     pino = pin;
     pinMode(pino,OUTPUT);
   }

   void ligaLed(){
     estado = HIGH;
     digitalWrite(pino,estado);
   }

   void desligaLed(){
     estado = LOW;
     digitalWrite(pino,estado);
   }
};
```

Note que podemos alterar qualquer atributo fisico do LED por meio desta classe, diminuindo o volume de código caso tenhamos, por exemplo, vários LEDs no nosso projeto.

Desafio Adicional:

Refaça algum dos projetos anteriores criando uma classe para o componente usado.