AULA 12 - IoTA

Programação física

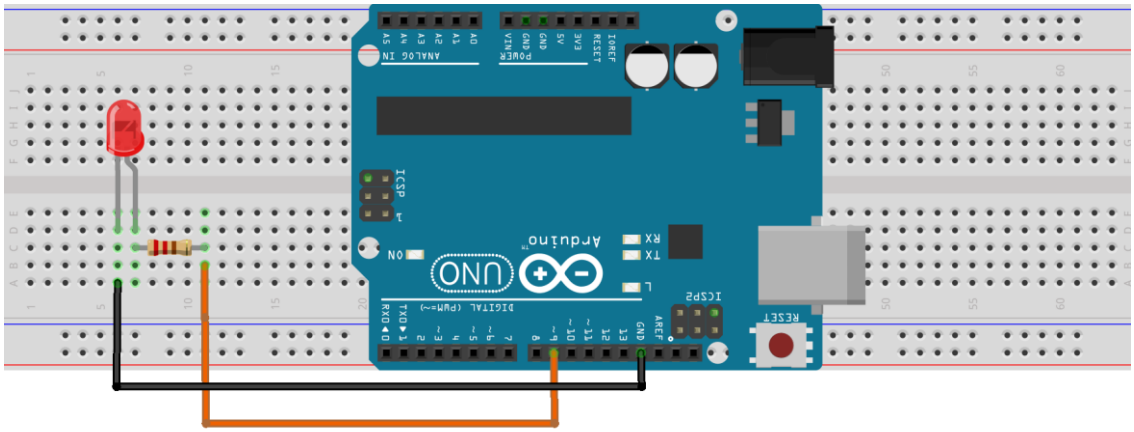
Como controlar a intensidade de um LED com PWM (fade)

https://youtu.be/xO2iHnN5YC4

Fade: Circuito e código

Nas aulas anteriores utilizamos o LED em saídas digitais, que permitiam apenas que o ligássemos ou desligássemos. Agora, o conectaremos uma saída PWM (vamos explicar mais à frente o que significa PWM), para verificar como o seu brilho pode variar.

O circuito apresentado no vídeo é similar ao circuito com LED que você já usou na semana anterior, porém com o LED conectado ao pino 9.



Utilizamos o pino 9 (e não o 13, por exemplo) pois ele pode se comportar como uma saída do tipo PWM (saídas PWM são identificadas com um ~ na frente do seu número).

O código do exemplo apresentado encontra-se abaixo. Faça o upload para sua placa e observe o comportamento do LED.

Se quiser entender com mais detalhes o código utilizado, veja os conteúdos da sessão Indo Além: Conceitos de Programação.

```
/*
 Luminosidade de um LED com variação por meio do PWM
 Um loop aumenta a luminosidade aos poucos e depois outro loop diminui esta luminosidade.
*/
int led = 9;                  // o número do pino ligado ao LED, no caso o 9 e variáveis para guardar valores
int lum;                      // luminosidade do LED
int valor = 5;                // valor para ser reduzido ou aumentado na luminosidade
void setup() {
  pinMode(led, OUTPUT);       // inicialização do pino do LED (pino 9) como saída
}
void loop() {
  for (lum = 0; lum < 255; lum = lum + valor) {         // loop condicional
  // lum inicia com 0 e aumenta com a variável valor até chegar a 255
    analogWrite(led, lum);          // escreve a luminosidade no pino 9, pino do LED
    delay(30);                      // espera 30 milisegundos para ver o efeito de mudança
  }
  for (lum = 255; lum > 0; lum = lum - valor) {         // loop condicional
  // lum inicia com 255 e diminui com a variável valor até chegar a 0
    analogWrite(led, lum);
    delay(30);
  }
}
```

AnalogWrite

Para que você entenda melhor como a função analogWrite funciona, vamos agora testar um exemplo mais simples.
Você se lembra do exemplo Blink, que fazia o LED piscar?

```
void  setup()  {
   pinMode(9,  OUTPUT);
}
void  loop()  {
   digitalWrite(9,  HIGH);
   delay(1000);
   digitalWrite(9,  LOW);
   delay(1000);
}
```

Nele utilizávamos a função digitalWrite(9, HIGH/LOW) para acender ou apagar o LED.
Agora vamos criar o mesmo comportamento utilizando a função analogWrite() . Esta função recebe em seu segundo argumento valores entre 0 e 255. Assim, para acender o LED usaremos analogWrite(9, 255) e para apagá-lo analogWrite(9, 0).
Experimente carregar o código abaixo para a sua placa e observe o comportamento do LED, que deve ser similar ao do exemplo Blink acima.

```
void setup() {
 pinMode(9, OUTPUT);
}
void loop() {
 analogWrite(9, 255);
 delay(1000);
 analogWrite(9, 0);
 delay(1000);
}
```

Agora, faça testes com outros valores entre 0 e 255 na função analogWrite e observe o que acontece.

Simulando PWM

Agora que você já viu como utilizar a função analogWrite para escrever em saídas PWM, vamos refletir um pouco sobre o que está acontecendo com o seu LED para que ele mude seu brilho.
Monte o circuito com o LED na saída 13 e faça novamente o upload do exemplo Blink com a função digitalWrite:

```
void  setup()  {
   pinMode(13,  OUTPUT);
}
void  loop()  {
   digitalWrite(13,  HIGH);
   delay(1000);
   digitalWrite(13,  LOW);
   delay(1000);
}
```

Neste exemplo, o LED fica ligado por 1 segundo e desligado por 1 segundo, certo? Vamos agora fazer alguns experimentos:
Comece mudando o valor que está dentro das duas funções delay() para 100 (ou seja 0,1 seg) e observe o que acontece.

Em seguida, mude o valor de ambos os delay() para 10 (ou seja, 0,01 seg). O que você observa? Seu LED deve estar piscando tão rápido que seus olhos não são capazes de diferenciar os momentos apagados dos acesos, e a sua percepção será a de que o LED está aceso, porém com uma intensidade reduzida.
Você pode fazer o upload de um código que mantém o LED aceso de forma constante para comparar os brilhos.
Agora mude o valor do primeiro delay() para 1, mas mantenha o segundo delay() em 10. O que aconteceu com o brilho do LED?
Faça mais alguns testes, mudando o parâmetro dos delays para valores entre 1 e 10.
Você deve ter notado que quanto menor for o intervalo de tempo no qual o LED fica aceso (ou maior o tempo no qual o LED permanece apagado), mais fraco ele parece.
É exatamente este comportamento que a função analogWrite simula, por meio do uso da modulação de largura de pulsos (PWM). A seguir, vamos mostrar com detalhes como o PWM funciona.

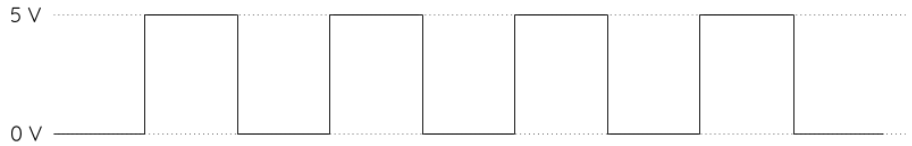
Pulsos PWM

Apesar de utilizarmos uma função chamada de analogWrite para modificar o brilho do LED, o Arduino não possui saídas analógicas de fato, pois ele só é capaz de fornecer valores de tensão de saída de 0 ou 5V.
Já que ele pode apenas estar nestes dois estados, temos dois casos extremos:

Manter a saída sempre em 0 V

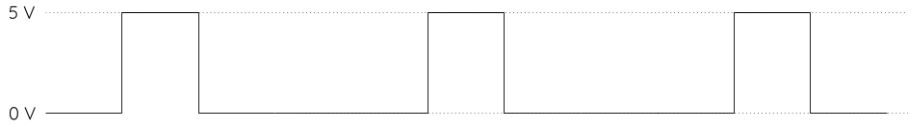
Manter a saída sempre em 5 V

No entanto, sabemos que podemos alternar o valor de saída de 0 para 5V (ou vice versa) em nossos programas, como no exemplo Blink.
Se criarmos um programa no qual o LED fica aceso por 1 segundo e apagado por 1 segundo, teríamos um padrão de pulsos similar à figura abaixo:

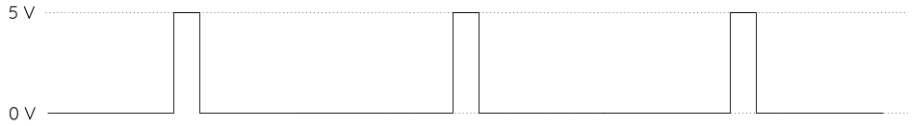


No gráfico acima, podemos observar que não há valor intermediário: ou a tensão está em 5 V, ou em 0 V.
Agora imagine que, ao invés de mantê-lo ligado por 1 segundo e desligado por 1 segundo, reduzíssemos este intervalo para milésimos de segundo, como fizemos na seção anterior. Neste caso, o LED pisca tão rápido que a nossa visão não é capaz de identificar estas mudanças, e enxergamos algo como uma "média" destes valores. Portanto, o LED nos parecerá aceso, porém com uma intensidade luminosa mais baixa.

É este o comportamento que a função analogWrite simula. Com ela o Arduino modula a largura destes pulsos, que serão definidos em valores de 0 a 255. O valor 0 seria equivalente ao que temos no primeiro gráfico, enquanto o valor 255 equivale ao segundo gráfico e o valor 128 ao terceiro gráfico.
Esta modulação de pulsos é chamada de PWM (do inglês pulse width modulation, ou modulação de largura de pulso) e é bastante útil para criarmos estes estados intermediários nas saídas.
Se reduzíssemos ainda mais o intervalo no qual o LED fica ligado, veríamos o LED com uma intensidade luminosa ainda mais reduzida. No gráfico abaixo, ele fica 1/4 do tempo ligado e 3/4 de tempo desligado. Ou seja: 25% do tempo ligado. Na linguagem do Arduino, este comportamento seria traduzido na função analogWrite(9, 64) - já que 64 é aproximadamente 1/4 de 255.



Assim, quanto menor o intervalo de tempo no qual ele permanece ligado, menor será a intensidade luminosa que veremos, e menor o valor do segundo argumento da função analogWrite(). Neste último exemplo abaixo, veríamos o LED com um brilho ainda mais fraco:



Como o sinal elétrico se torna um sinal de áudio

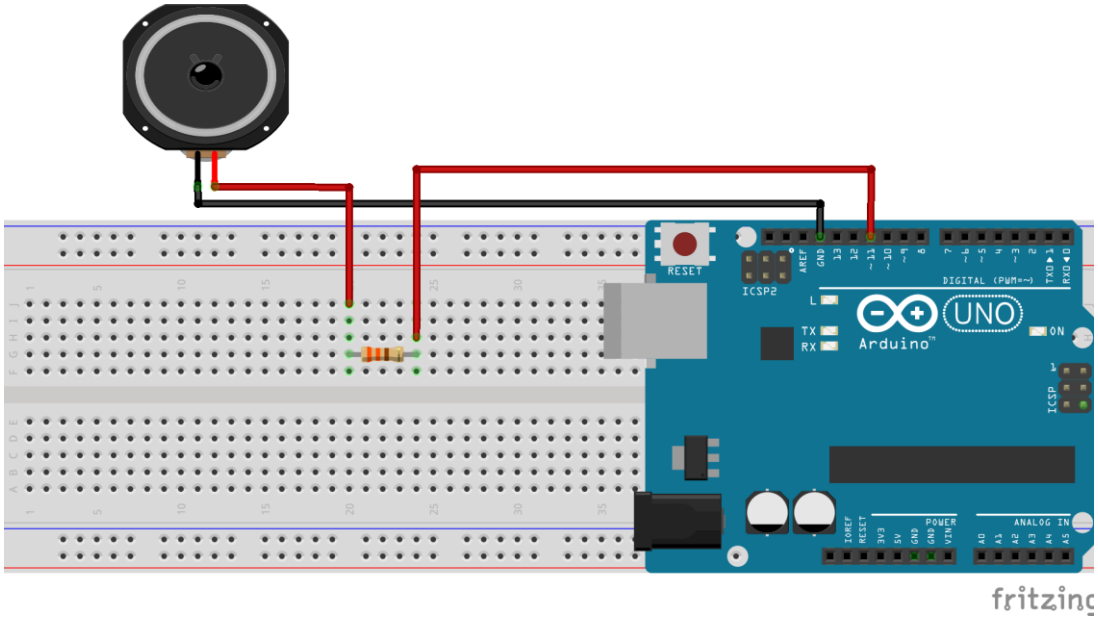
https://youtu.be/Z9LjkeNJNxM

Criando sons Biblioteca TONE

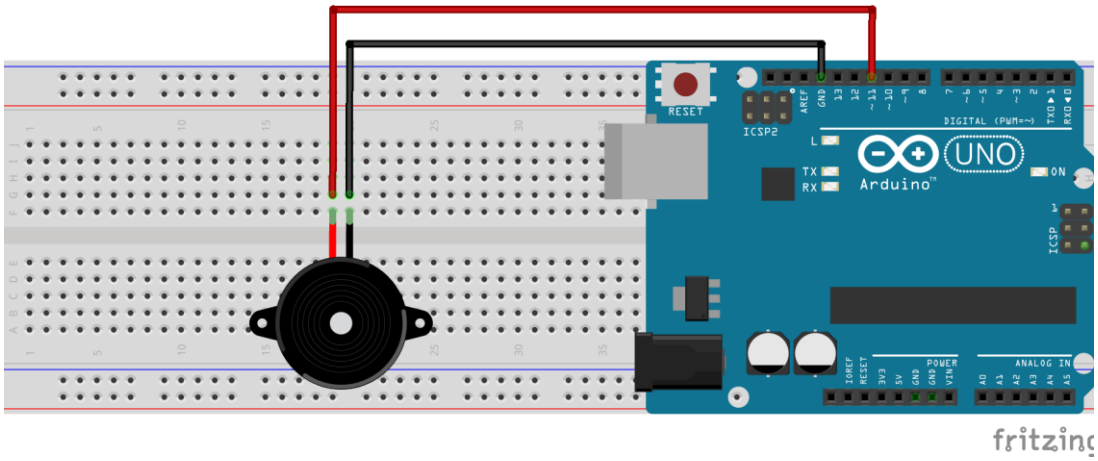
https://youtu.be/OUDzLZUw9bs

Criando sons: circuito e código

Abaixo apresentamos o circuito utilizado no vídeo para criar sons com a Arduino e um alto falante:



Você pode também utilizar um buzzer para esta finalidade. Utilizando um buzzer de 5V, o circuito ficaria assim:



Monte o circuito e faça o upload do código abaixo para a sua placa. Em seguida, abra o Monitor Serial e digite diferentes valores. O valor inserido corresponde à frequência da nota que será tocada.

```
/*
Código p/ emissão de tom conforme frequência informada
*/
int pinoFalante = 11;
void setup() {
  pinMode(pinoFalante, OUTPUT);
  Serial.begin(9600);
  Serial.println("Insira a frequência que quer ouvir: ");
}
void loop() {
  if (Serial.available()) {
  int duracao = 2000;
  int frequencia = Serial.parseInt();
  tone(pinoFalante, frequencia, duracao);
  delay(duracao);
  noTone(pinoFalante);
  }
}
```

Tocando melodias com a função tone

No exemplo anterior, usamos a função tone para tocar sons com a placa Arduino e um alto falante.

Esta função permite que informemos à placa a frequência da nota que queremos tocar, de acordo com a seguinte sintaxe:

- tone(pino, frequência), ou
- tone(pino, frequência, duração)

Nela, os parâmetros esperados são os seguintes:

- pino: o pino ao qual seu alto falante ou buzzer está conectado
- frequência: a frequência da nota em Hertz
- duração: a duração da nota em milissegundos (parâmetro opcional)

Fisicamente, é a frequência da nota que define o seu som. Assim, se você quiser tocar notas específicas utilizando a função tone, terá que saber suas frequências. Na tabela abaixo apresentamos as frequências das notas de uma oitava da escala de Dó maior:

Nota	Frequência (Hz)
Dó - C_4	261.63
Ré - D_4	293.66
Mi - E_4	329.63
Fá - F_4	349.23
Sol - G_4	392.00
Lá - A_4	440.00
Si - B_4	493.88
Dó - C_5	523.25

Mantendo o código e o circuito do exemplo anterior, abra o Monitor Serial e experimente escrever os valores das frequências acima no Monitor para ouvi-las.

Definição das notas em arquivos de cabeçalho

Nos exemplos anteriores havíamos que buscar a frequência de cada nota para poder tocá-la, o que dá bastante trabalho se quisermos escrever uma música longa.
Além disso, caso quiséssemos criar vários sketchs diferentes que toquem melodias precisaríamos definir em cada um deles quais notas correspondem a quais frequências.
Uma forma de facilitar a criação de melodias na Arduino é criar um arquivo de cabeçalho que possui a relação de notas para frequências (similar à tabela apresentada no exemplo anterior).
Esses arquivos, que na linguagem de programação da Arduino usam a extensão .h, são usados sempre que queremos separar certos elementos de um programa que vamos reutilizar em outros programas.
Neste exemplo, vamos criar um arquivo de cabeçalho com a definição das frequências correspondentes às notas de um piano. Em seguida, vamos criar um programa que cria e toca uma melodia, usando os nomes das notas que definimos no arquivo de cabeçalho.
Após a abrir a IDE, crie um nova aba (você pode fazer isso digitando Ctrl + Shift + N ou clicando no triângula no canto superior direito da IDE e selecionando "Nova Aba") e nomeie-a como "alturas.h" (sem as aspas). Cole então nesta nova aba o código abaixo.

```
/*************************************************
 * alturas.h: constantes com as frequencias
 *  correspondentes às alturas de notas musicais
 *  (parte inteira)
 *************************************************/
#define PAUSA    0
#define B0  31
#define C1  33
#define CS1 35
#define D1  37
#define DS1 39
#define E1  41
#define F1  44
#define FS1 46
#define G1  49
#define GS1 52
#define A1  55
#define AS1 58
#define B1  62
#define C2  65
#define CS2 69
#define D2  73
#define DS2 78
#define E2  82
#define F2  87
#define FS2 93
#define G2  98
#define GS2 104
#define A2  110
#define AS2 117
#define B2  123
#define C3  131
#define CS3 139
#define D3  147
#define DS3 156
#define E3  165
```

```
#define F3  175
#define FS3 185
#define G3  196
#define GS3 208
#define A3  220
#define AS3 233
#define B3  247
#define C4  262
#define CS4 277
#define D4  294
#define DS4 311
#define E4  330
#define F4  349
#define FS4 370
#define G4  392
#define GS4 415
#define A4  440
#define AS4 466
#define B4  494
#define C5  523
#define CS5 554
#define D5  587
#define DS5 622
#define E5  659
#define F5  698
#define FS5 740
#define G5  784
#define GS5 831
#define A5  880
#define AS5 932
#define B5  988
#define C6  1047
#define CS6 1109
#define D6  1175
#define DS6 1245
#define E6  1319
#define F6  1397
#define FS6 1480
#define G6  1568
#define GS6 1661
#define A6  1760
#define AS6 1865
#define B6  1976
#define C7  2093
#define CS7 2217
#define D7  2349
#define DS7 2489
#define E7  2637
#define F7  2794
#define FS7 2960
#define G7  3136
#define GS7 3322
#define A7  3520
#define AS7 3729
#define B7  3951
#define C8  4186
#define CS8 4435
#define D8  4699
#define DS8 4978
```

Agora, na outra aba aberta (sem nome e sem código), cole o código abaixo e faça o seu upload para a Arduino (tendo conectado o alto falante ou buzzer ou alto-falante com resistor ao pino 8).
Em seguida você pode modificar as notas que estão em int melodia[] e as durações em int duracoesNotas[] para criar outras melodias. Não se esqueça de fazer o upload do código para sua placa após realizar as alterações.
Observe que não precisamos mais buscar a frequência de cada nota: basta escrever C4, G3, etc. A inclusão da linha #include "alturas.h" faz com que não seja necessário escrever a frequência de cada nota, pois elas são buscadas no arquivo alturas.h a partir dos nomes C4, G3, etc. Notas com acidente são representadas pelo S (sustenido), como CS4, DS4, etc.
Esta notação (C4, G3, etc.) é bem comum em música. Nela, C = Dó, D = Ré, E = Mi, F = Fá, G = Sol, A = Lá e B = Si. O número ao lado corresponde à oitava (ou altura) da nota: mais grave ou mais aguda. O dó central de um piano nessa notação é o C4, por exemplo.

Na seção Indo Além: Conceitos de programação explicamos em mais detalhes como este programa funciona.

```
/*
 Melody
 Conecte um buzzer ou alto falante ao pino 11
 Este exemplo é de domínio público
 http://www.arduino.cc/en/Tutorial/Tone
*/
#include "alturas.h"
// notas da melodia
int melodia[] = {
 E5, E5, PAUSA, E5, PAUSA, C5, E5, PAUSA, G5, PAUSA, PAUSA, PAUSA, G4
};
// duração das notas: 4 = semínima (ou 1/4 seg), 8 = colcheia (ou 1/8 seg), etc.:
int duracaoDasNotas[] = {
```

```
  4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4
};
void setup() {
  // pino no qual está conectado o alto-falante/buzzer
  const int pinoSom = 11;
  // Calcula o tamanho da melodia, tomando o tamanho do vetor que armazena
  // os n números inteiros equivalentes à melodia e dividindo pelo tamanho
  // que um número inteiro ocupa na memória
  int tamanhoDaMelodia = sizeof(melodia)/sizeof(int);
  // itera pelas notas da melodia
  for (int essaNota = 0; essaNota < tamanhoDaMelodia; essaNota++) {
    // para calcular a duração da nota, divida 1 segundo (ou 1000 ms) pelo tipo da nota
    // por ex.: semínima = 1000 / 4, colcheia = 1000/8, etc.
    int duracaoDessaNota = 1000 / duracaoDasNotas[essaNota];
    tone(pinoSom, melodia[essaNota], duracaoDessaNota);
    // para distinguir as notas, defina um intervalo mínimo entre elas
    // a duração da nota + 10% funciona bem (ou 1,1 vezes a nota):
    delay(duracaoDessaNota);
    // parar de tocar a nota
    noTone(pinoSom);
  }
}
void loop() {
  // a melodia não se repete: toca apenas uma vez no setup.
}
```

Servo motor: o que é e como usar

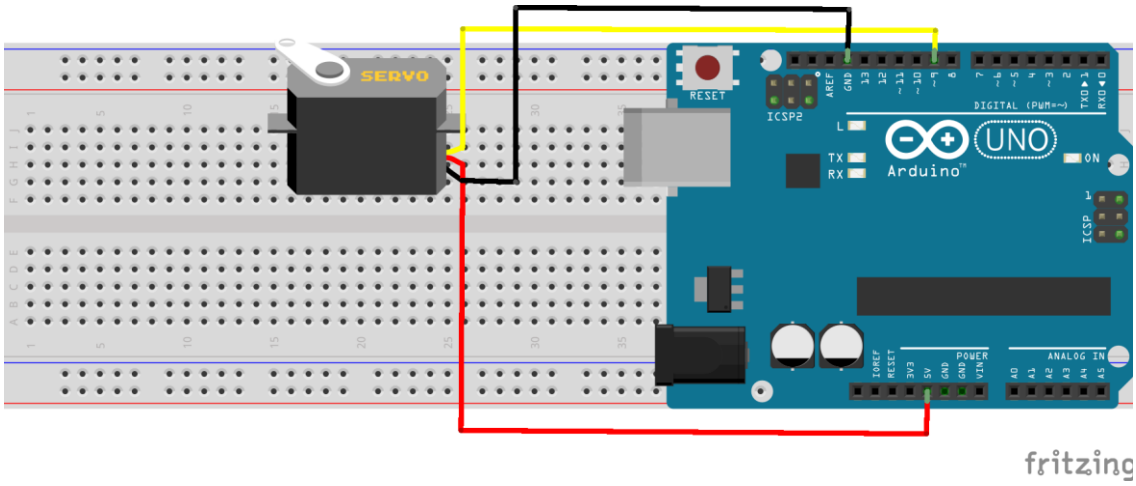
https://youtu.be/ATn8xTQGnWI

Servo motor: circuito e programação

O servo motor é um tipo de motor que permite que indiquemos um ângulo específico para o qual ele deve se mover. Após se mover para o ângulo selecionado, ele permanecerá nesta posição até que receba um comando para ir para outra posição. Em geral, os servo motores possuem um intervalo fixo de rotação que vai de 0 a 180 graus.

Os servo motores costumam ter três pinos para conexão: alimentação (vermelho), sinal (laranja ou branco) e terra (preto ou marrom). Estas são as cores mais comuns, porém alguns fabricantes podem usar outros padrões de cor. Assim, caso o seu motor seja diferente do apresentado, verifique o datasheet (especificação técnica) antes de conectá-lo.

O servo motor que utilizamos neste curso é do tipo micro servo motor 9g. O circuito e o código apresentados no vídeo encontram-se abaixo.



Conecte o servo motor à placa Arduino e faça o upload do código. Em seguida, faça alterações no código para mudar os ângulos e observe os resultados.

```
/*
 Controlando o ângulo de um servo motor
 Em um loop, envia de 2 em 2 segundos os ângulos 180, 90 e 0 graus para o servo motor
*/
// inclui uma biblioteca para controle do servo motor, ou seja, um conjunto de funções específicas para este motor
#include <Servo.h>
//variável para representar o servo motor. É ela quem recebe os comandos neste código para envaiá-las ao servo motor.
Servo myservo;
// variável para guardar o valor do ângulo a ser enviado ao servo motor
int angulo;
// configuração inicial
void setup()
{
   // configura o representante do servo no pino 9
   myservo.attach(9);
}
void loop()
{
   angulo = 180;                    // a variável ângulo recebe o valor 180
   myservo.write(angulo);           // envia o comando "escrever ângulo" para o servo (180 graus)
   delay(2000);                     // espera 2 segundos
   angulo = 90;
   myservo.write(angulo);           // motor vai para o ângulo 90
   delay(2000);
   angulo = 0;
   myservo.write(angulo);           // motor vai para o ângulo 0
   delay(2000);
}
```

Servo motor: outros exemplos

Com o mesmo circuito anterior, faça o upload do exemplo Sweep para sua placa. Este exemplo encontra-se na IDE Arduino em Arquivo > Exemplos > Servo > Sweep.

Observe que no código deste exemplo, é utilizada a função for, sobre a qual falamos na semana anterior. Neste programa, a função for é usada duas vezes: na primeira, faz com que o motor vá para da posição 0° para 180° (de 1° em 1°), e na segunda, que volte gradativamente à posição 0°.

Agora que você já sabe como usar um servo motor, pode testar novas coisas com ele. Para começar, você pode tentar:

Usar um botão para controlar a posição do servo motor (0° ou 180°)

Usar um potenciômetro para controlar a posição do servo motor (de 0° a 180°)

Motor DC: o que é e como usar

https://youtu.be/P6AaVUF7zeI

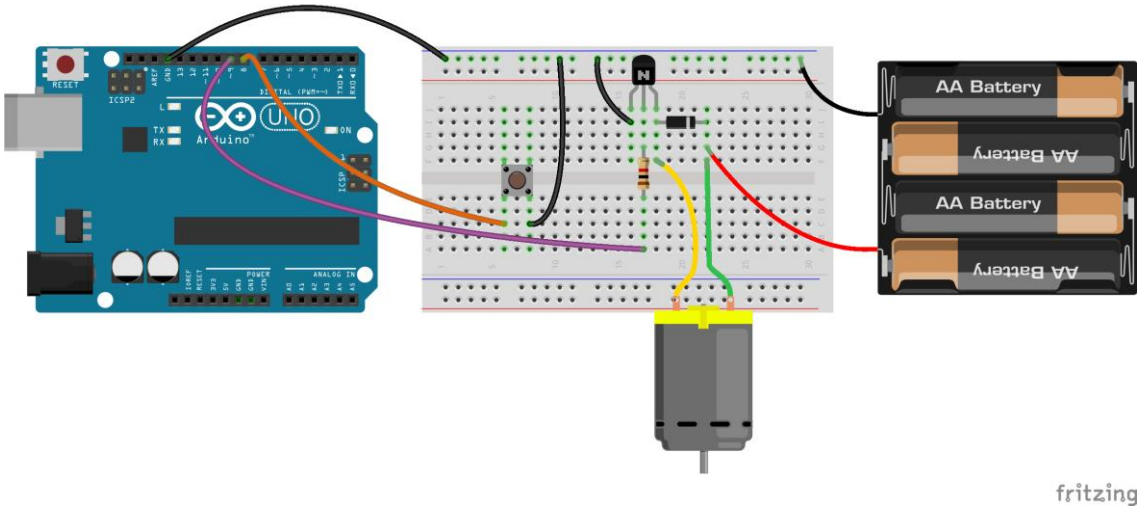
Motor DC: Circuito e código

Como você deve ter notado, o servo motor utilizado até agora podia ser ligado diretamente à placa Arduino, pois possui um consumo de corrente que pode ser fornecido por ela sem danificá-la.

Entretanto, o motor DC (motor de corrente contínua) possui um consumo elevado de corrente e pode danificar a placa se ligado diretamente a ela. Para contornar este problema, adicionamos um circuito com transistor, diodo e bateria externa ao motor, no qual a corrente é fornecida pela bateria e a Arduino apenas é responsável por ativar ou não a alimentação do motor.

Para montar o circuito apresentado nesta aula com motor DC, você precisará de:
1 Protoboard
1 Arduino
1 motor dc 6V
4 pilhas AA com suporte
1 diodo 1N4148
1 transistor 2N2222
1 resistor 1K Ohms
1 chave momentânea (push button)
2 jumpers com garra jacaré
8 jumpers macho-macho

O seu circuito para controle de motor DC com botão deve ficar similar ao abaixo:



Abaixo você encontra o código utilizado.
Observe que ele é bastante similar ao exemplo do botão com LED pois neste caso, também estamos acionando um atuador (motor) em uma saída digital a partir do sinal recebido de uma entrada digital (botão).
Note ainda que o botão não possui pull down, com um dos pinos ligado ao terra (GND) e o outro ao sinal (pino 8). Podemos fazer isso pois definimos na função pinMode(pino_botao, INPUT_PULLUP) que acionaremos no pino 8 o pull up interno da Arduino.
Como estamos usando um pull up (e não um pull down), o outro terminal do botão será ligado ao terra.

```
/*
 Ligando e desligando um motor DC
*/
int pinoMotor = 11;
int pinoBotao = 8;
int estadoBotao = LOW;

void setup() {
   pinMode(pinoMotor, OUTPUT);
   pinMode(pinoBotao, INPUT_PULLUP);
   digitalWrite(pinoMotor, LOW);
}
void loop() {
   estadoBotao = digitalRead(pinoBotao); //leitura do estado do botão
   if (estadoBotao == HIGH) {                    //se o botão estiver apertado, desliga o motor
   digitalWrite(pinoMotor, LOW);
```

```
    }
    else  {                                          //se não estiver apertado, liga o motor
    digitalWrite(pinoMotor,  HIGH);
    }
}
```

Motor DC com PWM

Assim como utilizamos a função analogWrite na semana anterior para criar diferentes intensidades luminosas para o LED, também podemos utilizá-la para controlar a velocidade de rotação de motores DC.

Ao contrário do servo motor, não é possível definir ângulos exatos para o motor DC. No entanto, podemos alterar a sua velocidade de rotação, fazendo com que gire mais rápido ou mais devagar. Para isso, utilizamos pulsos PWM, de maneira similar à utilizada para controle do LED.

Experimente fazer o upload do código abaixo para sua placa, mantendo o circuito da unidade anterior. O que você observa? Tente fazer alterações na programação e entender como elas afetam o comportamento do motor.

```
int motor = 9;          // o número do pino ligado ao motor, no caso o 9
int vel;
int valor = 5;

void setup() {
  pinMode(motor, OUTPUT);
}

void loop() {

  for (vel = 0; vel < 255; vel = vel + valor) {
    analogWrite(motor, vel);
    delay(50);
  }

  for (vel = 255; vel > 0; vel = vel - valor) {
    analogWrite(motor, vel);
    delay(30);
  }
}
```

Repetindo trechos de código: comando "for"

O programa começa definindo a variável lum como zero. Em seguida, temos um loop no qual o brilho do gato assume o valor da variável lum e, a cada repetição, esta variável aumenta 5 unidades. Ou seja: a cada repetição o brilho aumenta em 5. Dentro do loop temos também uma espera de 30 milissegundos para que a mudança seja mais visível. Este aumento de 5 no brilho se repete até que o brilho atinja o valor de 100. Quando isto ocorre, o programa sai do loop e termina.

Experimente variar o valor acrescido na luminosidade e o tempo de espera dentro do loop para ver como estes parâmetros afetam o comportamento do programa.

Em seguida, clique no ícone do Gato 2 e tente entender o código. Clique nos blocos para observar o que acontece.

Agora que você se familiarizou com este tipo de construção, vamos voltar ao exemplo desta semana, no qual mudávamos gradativamente o brilho do LED.

```
int  led  =  9;                  // número do pino no qual o LED está conectado
int  lum;                         // luminosidade do LED
int  valor  =  5;                // valor a ser adicionado ou subtraído da luminosidade
```

```
void setup() {
  pinMode(led, OUTPUT);
}
void loop() {
  for (lum = 0; lum < 255; lum = lum + valor) {
    analogWrite(led, lum);
    delay(30);
  }
  for (lum = 255; lum > 0; lum = lum - valor) {
    analogWrite(led, lum);
    delay(30);
  }
}
```

O programa começa com a definição das variáveis que serão utilizadas e em seguida, na função setup, define-se que o pino 9 (chamado de led na primeira linha do código) será uma saída, ou OUTPUT. Em seguida, na função loop, temos dois comandos for.

O comando for possui os seguintes argumentos:

```
for (inicialização; condição; incremento) {
  // algo que ocorre
  }
```

A inicialização ocorre apenas uma vez, no início. A cada repetição, a condição é testada: se for verdadeira, o "algo que ocorre" é executado e ocorre o incremento; se não for verdadeira, o loop termina.

No nosso caso, temos o seguinte for inicial:

```
for (lum = 255; lum > 0; lum = lum - valor) {
    analogWrite(led, lum);
    delay(30);
}
```

Assim, a luminosidade começa no valor máximo (255), e subtrai-se dela 5 (definido na variável valor) a cada repetição. Enquanto ela for maior que 0 (valor mínimo), a função analogWrite(led, lum) será executada, escrevendo no pino 9 o valor da luminosidade (255, 250, 245, 240, e assim por diante), seguida de uma pausa de 30 milissegundos para que sejamos capazes de enxergar esta mudança ocorrendo.

Em seguida temos outro trecho muito similar, que faz com que o LED aumente sua luminosidade, que começa em 0 e aumenta de 5 em 5.

O comando for é bastante útil pois otimiza um procedimento que poderia demandar muitas linhas de código se fosse escrito de outra forma. Esta estrutura é similar ao exemplo que criamos para o gato - porém, como no Scratch não temos o comando for, utilizamos o bloco repita até que (no qual testamos a condição), e dentro dele fazemos o incremento do valor.

Vetores: uma forma de guardar melodias

Nas unidades anteriores nós aprendemos a usar variáveis para armazenar na memória valores que pretendemos usar mais tarde.

No Sketch Melody da unidade Definição das notas em arquivos de cabeçalho usamos uma nova estrutura de dados que nos permite armazenar uma coleção de variáveis. Essa estrutura chama-se Vetor, ou Array em inglês.

Criando vetores

Há diferentes formas de se criar um vetor. Nesse curso, usaremos a seguinte forma:

<tipo de variável do vetor> <nome do vetor>[] = { <elemento 1>, <elemento 2>, ... };

Voltando ao Sketch, temos o vetor melodia, criado da seguinte maneira:

```
int melodia[] = {
  E5, E5, PAUSA, E5, PAUSA, C5, E5, PAUSA, G5, PAUSA, PAUSA, PAUSA, G4
```

```
};
```
melodia é um vetor que armazena números inteiros (int). Seu primeiro elemento é E5, que podemos ver no arquivo de cabeçalho alturas.h vale 659. Seu segundo elemento também é E5, ou seja, 659. Seu terceiro elemento é PAUSA, que vale 0. E assim por diante.
Normalmente usamos vetores quando queremos agrupar valores que fazem sentido em conjunto, como notas de uma melodia, sequência de leituras de um sensor, cores de pixels de uma linha de uma tela, etc.

Acessando os elementos de um vetor

Para acessar os conteúdos de um vetor, precisamos usar um índice. Imagine o seguinte vetor, que armazena o valor de três pinos da Arduino conectados à LEDs:

```
int pinosLED[] = { 11, 12, 13 };
```

Para ligar o led conectado ao pino 11, poderíamos usar o seguinte comando:

```
digitalWrite(pinosLED[0], HIGH);
```

Para ligar o led conectado ao pino 13, poderíamos usar o seguinte comando:

```
digitalWrite(pinosLED[2], HIGH);
```

Podemos também recuperar o valor de uma posição de um vetor e armazená-lo em outra variável:

```
int primeiroLed;
primeiroLed = pinosLED[0];
```

Um vetor é indexado em zero, o que significa que seu primeiro elemento é acessado com o índice 0, o segundo com o índice 1, e assim por diante. Isso também significa que num vetor de 13 elemento como é a melodia, o último elemento tem índice 12.

Modificando uma posição de um vetor

Para modificar um elemento de um vetor, precisamos saber seu índice. No exemplo acima, suponha que que mudamos o led que estava no pino 11 para o pino 4. Podemos alterar nosso vetor com:

```
pinosLED[0] = 4;
```

Vetores e comando "for"

Vetores são normalmente acessados e manipulados usando comandos "for". Usamos contador de ciclos ou loops (o valor que incrementamos ou decrementamos a cada ciclo) como índice do vetor, acessando assim todos seus elementos.
Por exemplo, para imprimir no monitor serial todas as notas de melodia, poderíamos fazer:

```
int i;
for (i = 0; i < 13; i = i + 1) {
  Serial.println(melodia[i]);
}
```

No trecho acima, 13 é o número de elementos do vetor melodia. Observe que o contador i vai percorrer os valores de 0 até 12, e vamos então imprimir todos os valores que compõe esse vetor.

Calculando o número de elementos de um vetor

No Sketch Melody da unidade Definição das notas em arquivos de cabeçalho usamos outra maneira de calcular o número de elementos do vetor. Ao invés de escrevermos diretamente esse número, usamos o comando sizeof da Arduino, que calcula o tamanho (em bytes) de uma variável ou vetor.

Pegando o tamanho em bytes do vetor de números inteiros melodia e dividindo esse número pelo tamanho em bytes de um número inteiro, temos o número de elementos do vetor!

```
int tamanhoDaMelodia = sizeof(melodia)/sizeof(int);
```

Qual a vantagem de calcular o número de elementos do vetor dessa forma ao invés de o escrevermos diretamente?
Imagine que estamos criando nossa melodia, e adicionamos ou tiramos algumas notas dela. Caso tivéssemos escrito diretamente o número de elementos do vetor no comando for, precisaríamos lembrar de atualizar esse número.
É muito comum esquecermos disso, o que faz com que deixemos de visitar alguns valores do nosso vetor (caso nos esqueçamos de aumentar o número de elementos) ou pior, acessar valores além do nosso vetor (caso nos esqueçamos de diminuir o número de elementos).
É preciso sempre prestar atenção aos índices que estamos usando!

Bibliotecas

Muitas vezes precisamos de códigos bastante complexos para criar determinados programas de Arduino, especialmente quando utilizamos determinados sensores e atuadores que requerem códigos específicos para funcionar.
Bibliotecas são conjuntos de códigos que tornam mais fácil a utilização de sensores e módulos específicos ou que facilitam a criação de determinadas funcionalidades no comportamento da placa. Assim, ao invés de ter que escrever estes códigos longos e específicos, podemos chamá-los de uma biblioteca externa que já os contém.
Diversas bibliotecas já vêm instaladas na IDE, como a biblioteca Servo (para utilizar um servo motor) ou a LiquidCrystal (para utilizar displays de LCD). Mas você pode também utilizar bibliotecas criadas por outras pessoas, disponíveis na Internet (existem centenas!), ou criar a sua própria biblioteca.
Abaixo, apresentamos algumas formas de instalar uma biblioteca:

Usando o gerenciador de bibliotecas

Uma das formas de incluir uma biblioteca é selecioná-la em Sketch > Incluir Biblioteca.
Se a biblioteca desejada não estiver disponível na lista que aparece, você pode instalá-las em Sketch > Incluir Biblioteca > Gerenciar Bibliotecas (recurso disponível da versão 1.6.2 em diante).
No Gerenciador de Bibliotecas você encontrará uma lista de bibliotecas que estão prontas para instalação ou já instaladas. Busque a biblioteca desejada (por exemplo, a biblioteca Tone) e, em seguida, clique em Instalar.

Após concluir a instalação, a biblioteca aparecerá na lista de Sketch > Incluir Biblioteca. Ao selecioná-la, você verá uma nova linha de código no início do seu programa. No caso da biblioteca Tone, teremos:

```
#include <Tone.h>
```

Importando uma biblioteca .zip

Se você quiser incluir alguma biblioteca que encontrou na Internet e que não está disponível na lista acima, você pode instalá-la a partir de um arquivo .zip.
Para isso, faça o download do arquivo .zip correspondente à biblioteca para o seu computador. Em seguida, na IDE vá até Sketch > Incluir Biblioteca > Incluir Biblioteca .ZIP. Selecione o arquivo .zip salvo no seu computador e clique em Abrir.
Por fim, em Sketch > Incluir Biblioteca, selecione a biblioteca que você acabou de adicionar, que aparecerá ao final da lista.

Instalação manual

Você também pode fazer a instalação manual de bibliotecas, adicionando os arquivos da pasta .zip aos diretórios correspondentes da Arduino.
Existem muitas informações disponíveis na Internet sobre como fazer isso. Caso sinta necessidade, você pode buscar estas informações e compartilhá-las no fórum.

Função Map

Imagine que você queira converter o valor de leitura do potenciômetro em um grau correspondente para o ângulo do motor, ou em uma luminosidade para o LED que varie de 0 a 100%.
Os valores de leitura das entradas analógicas vão de 0 a 1023, enquanto os valores de saída PWM variam de 0 a 255 e os ângulos do motor vão de 0 a 180.
Assim, precisamos converter de alguma forma valores de 0 a 1023 em correspondentes de 0 a 255 ou de 0 a180, certo?
Poderíamos fazer isso usando uma regra de 3 e criando uma operação em nosso programa para fazer essa conversão. Seria algo como multiplicar o valor lido na entrada por 255 (ou 180) e depois dividi-lo por 1023.
No entanto, ao invés de termos que fazer estes cálculos, existe uma função que faz esta conversão de maneira automática: a função map.
No caso de uma conversão de valores de leitura que variam de 0 a 1023 para valores de 0 a 255, teríamos:

```
valor2 = map(valor, 0, 1023, 0, 255);
```

De maneira similar, para converter os valores em um intervalo de 0 a 180, teríamos:

```
valor2 = map(valor, 0, 1023, 0, 180);
```

Neste exemplo, função map faz a conversão do que foi lido na variável valor e a salva na variável valor2.
Mas poderíamos também fazer isso utilizando apenas uma variável. Observe o exemplo abaixo:

```
void setup() {}
void loop(){
  int val = analogRead(0);
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

- Começamos fazendo a leitura do sensor A0 e salvamos este valor na variável "val".
- Em seguida, convertemos o valor lido (que vem de uma escala de 0 a 1023) para uma escala de 0 a 255, e o salvamos na variável "val", sobrescrevendo o valor anterior.
- Por fim, enviamos para o pino 9 uma saída PWM correspondente a esta conversão.

Para o caso de um motor, você pode utilizar o exemplo abaixo:

```
#include <Servo.h>
Servo servomotor;
int sensor = A0;
int val;
void setup() {
  servomotor.attach(9);    // attaches the servo on pin 9 to the servo object
}
void loop() {
  val = analogRead(sensor);
  val = map(val, 0, 1023, 0, 180);
  servomotor.write(val);
  delay(15);
}
```

Referências:

- Material integralmente extraído e adaptado do curso **Programação Física**, da plataforma 'Code IoT', criado em parceria com a Samsung e LSI-TEC Escola Politécnica da USP - https://codeiot.org.br/courses/course-v1:LSI-TEC+IOT104+2020_O2/about - Acessado em 01/11/2020.
    - Licença disponível em https://creativecommons.org/licenses/by-nc-nd/4.0/deed.pt_BR - Acessado em 01/11/2020.
    - Plano de aula disponível em https://codeiot.org.br/assets/courseware/v1/fa2b33021964a8f58fbca2b8e4ee677d/asset-v1:LSI-TEC+IOT104+2018_S2+type@asset+block/Plano_de_Aula_Progamacao_Fisica_Semana_3.pdf - Acessado em 01/11/2020.