PLACA ARDUÍNO
E SUAS FUNCIONALIDADES

ARTE, CIÊNCIA E TECNOLOGIA

sesc

Desenvolvido por: Paulo Losse - PLV Education

PROCESSADOR VS CONTROLADOR

Um microprocessador, conhecido também somente por processador, é um componente mais voltado para cálculos matemáticos e lógicos, sendo especialmente veloz e eficiente neste tipo de trabalho. Presente em computadores e celulares, tornou-se muito popular. Para seu correto funcionamento, precisamos de componentes auxiliares, como memória RAM, HD e outros.

Já os microcontroladores possuem, em geral, a unidade lógica programável (equivalente ao processador), a memória de uso volátil (equivalente a RAM) e uma memória voltada para armazenamento mais duradouro (equivalente ao HD) tudo num mesmo chip, o que reduz o tamanho e peso do sistema como um todo, facilitando a utilização em projetos como robótica e automação.

ARDUÍNO

O arduíno é uma plataforma eletrônica de software e hardware livre, que tem por objetivo ser fácil de aprender e usar, tanto do ponto de vista de montagem dos circuitos elétricos quanto da programação.

Funciona basicamente como o cérebro do nosso sistema. Na placa temos alguns pinos que podem ser usados como entradas ou saídas, através dos quais podemos ler sinais elétricos vindos de sensores (como um sensor de distância, por exemplo) ou controlar atuadores (como motores ou um LED). Através da programação que fazemos, o microcontrolador na placa sabe como interpretar os sinais dos sensores e o que fazer com eles (acionar um alarme, ligar um motor, fechar uma porta, etc).

Existem diversas versões do arduíno, com diferentes características; você pode verificá-las no site oficia: www.arduino.cc/en/main/boards. A versão que estaremos vendo aqui é o Arduíno UNO, uma das mais famosas. O Uno, como o chamaremos aqui, trabalha com 5V, possui 14 pinos de I/O, dos quais 6 possuem capacidade para PWM e 6 entradas analógicas.

O INÍCIO

Vamos começar decifrando essa sopa de letrinhas da última frase:

O primeiro conceito importante que precisamos ter em mente é a diferença entre um sinal analógico e um digital. Um sinal digital é representado simplesmente por 0 **ou** 1. Ou é 0 ou é 1, ou está desligado ou está ligado, não existe uma opção "no meio". Este tipo de sinal é o que os controladores e processadores entendem. O nível 0, também conhecido como nível baixo (low) é criado na prática com 0V e o nível 1, também conhecido como nível alto (high) é criado com 5V no caso do Uno.

Já um sinal analógico varia **entre** 0 e alguma tensão (que no caso do Uno não pode ultrapassar 5V). Um dos exemplos mais clássicos desse tipo de sinal é o som. Na prática, a maioria das grandezas físicas são analógicas: temperatura, pressão, distância, velocidade... Como os microcontroladores só entendem sinais digitais, é preciso fazer uma conversão através de um conversor analógico digital (ADC). O Arduíno Uno já contém um ADC de 10 bits embutido. Ser de 10 bits significa que ele converte qualquer valor que esteja entrando nele (desde que não ultrapasse o limite de 5V) em algo entre 0 e 1023 (pois 210 = 1024).

Na placa do Uno podemos facilmente identificar os pinos de entrada analógica, nomeados como analog input. Já os pinos digitais, podem ser usados como entrada (input) ou saída (output), daí seu nome: **I**nput/**O**utput (I/O).

Não existem saídas analógicas reais, porém alguns pinos possuem capacidade para uma tecnologia chamada PWM que "simula", com algumas limitações, a funcionalidade de uma saída analógica. Esses pinos são na verdade digitais e podem ser programados para funcionar como entrada ou saída digital normal **ou** PWM. Na placa física são identificados por um til (~) ao lado do número da porta.

CONCEITOS BÁSICOS DE ELETRICIDADE

> Eletricidade

Basicamente a eletricidade é o fluxo ordenado de elétrons, daí seu nome.

Toda matéria contém elétrons, porém em alguns materiais esses elétrons fluem com mais facilidade que outros. Os materiais que permitem facilmente a passagem de corrente elétrica são chamados condutores, é o caso dos fios de equipamentos eletrônicos, por exemplo. Já os materiais que não permitem a passagem fácil de corrente são denominados isolantes, como a borracha e o vidro.

> Tensão

A tensão elétrica é a medida da diferença de potencial elétrico entre dois pontos. Simplificadamente, é a "vontade" que os elétrons têm em "correr" do ponto A para o ponto B.

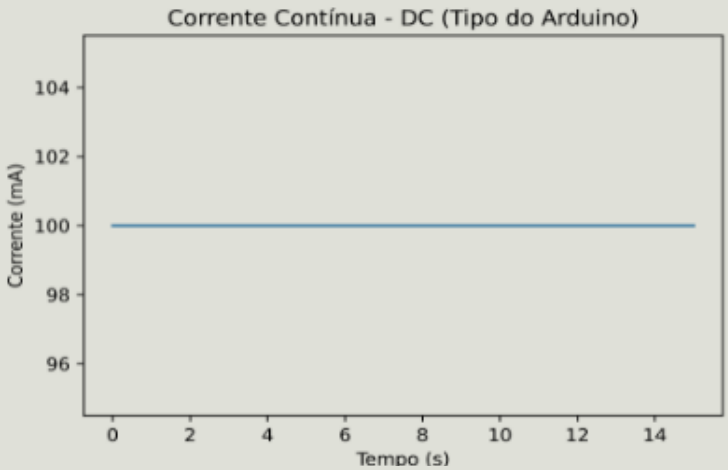
A analogia com a água é muito utilizada para simplificar a "visualização". Imagine uma caixa d'água cheia. Perto da caixa, o fluxo de água é baixo, quanto maior a diferença de altura entre você e a caixa, maior o fluxo. Esta diferença de **altura** é análoga à diferença de potencial elétrico (tensão).

Normalmente é representada pela letra V ou U em circuitos elétricos e fórmulas matemáticas e sua unidade é o Volt (V).

A tensão também é conhecida como "voltagem", porém este termo está errado. O correto é tensão.

>Corrente

A corrente é a medida do **fluxo** de elétrons, ou seja, quantos elétrons passam por determinada área em determinado tempo. Normalmente é representada pela letra I e sua unidade é o Ampere (A).

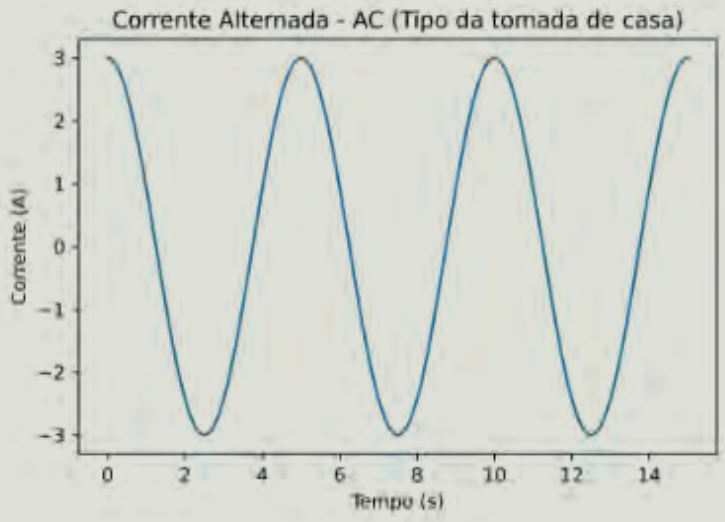


Seguindo nossa analogia com a caixa d'água, o fluxo de água é equiparável com o fluxo de elétrons.

A corrente é muitas vezes chamada de "amperagem", mas este termo também está errado.

Existem dois tipos principais de corrente: Corrente Alternada (AC em inglês ou CA em português) e a Corrente Contínua (DC ou CC).

Como podemos ver no gráfico à direita, a corrente contínua tem seu valor constante ao longo do tempo, daí seu nome. Este é o tipo de corrente que encontramos em pilhas e baterias. É também o tipo de energia que sai das USBs e o tipo que o Arduíno usa.



Já no gráfico da esquerda, vemos a corrente alternada, que altera seu sentido ao longo do tempo. Perceba o valor 0 no meio do eixo vertical do gráfico e valores negativos de corrente abaixo dele. Não existe corrente negativa, isso é apenas uma representação de que ela está fluindo no sentido contrário. Esse é o tipo de energia que temos nas tomadas em nossas casas.

Em geral, o formato de onda da tensão e da corrente são muito semelhantes. Assim sendo, existe também a tensão contínua e a tensão alternada, que seguem o mesmo princípio explicado acima. Uma diferença importante é que dizemos que a tensão muda de polaridade e não de sentido como a corrente (por isso inclusive que não existe positivo e negativo numa tomada).

É comum também algumas pessoas usarem os termos corrente e tensão como sendo a mesma coisa, o que, como vimos, está errado.

> Resistência

Como falamos mais acima, alguns materiais conduzem a corrente elétrica mais facilmente que outros.

A **dificuldade** de circulação de corrente elétrica representa a **resistência** de um material. Ou seja, quanto menor a resistência, mais facilmente a corrente flui.

Normalmente é representada pela letra R e sua unidade é o Ohm (representado pela letra grega ômega: Ω).

Materiais condutores, como o cobre (usado para fabricação de fios) possuem uma resistência baixa. Já materiais isolantes, por sua vez, possuem resistências elevadas.

Em geral, os metais são bons condutores de eletricidade. Também é comum que bons condutores de eletricidade sejam bons condutores de calor e vice-versa. De curiosidade, a Prata é o metal mais condutor que se conhece.

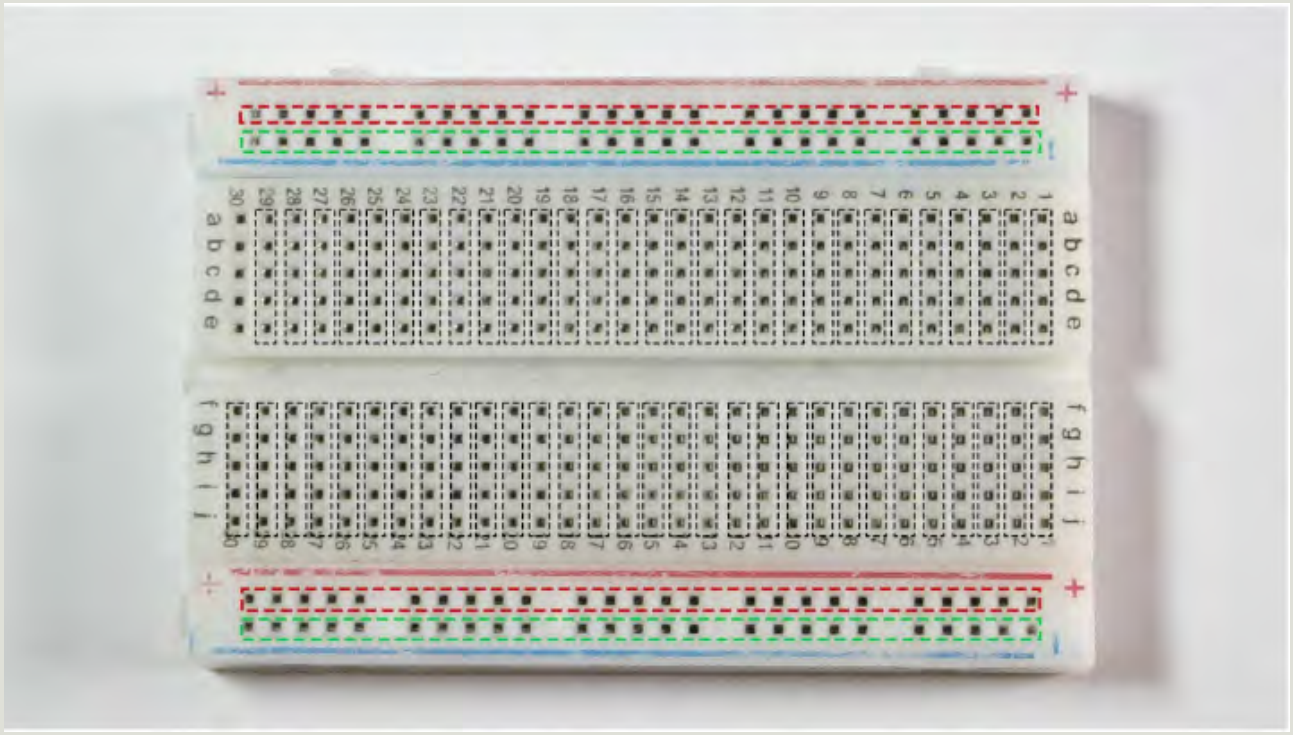
COMPONENTES

> Protoboard

A protoboard, também conhecida como breadboard ou "placa de ensaio" em português, é uma matriz formada por contatos elétricos interconectados de uma forma específica. Seu objetivo é facilitar a montagem, desmontagem e modificação de circuitos e é, por isso, especialmente utilizada durante a fase de projeto e testes.

Todos os projetos de exemplo que veremos aqui mais adiante serão montados com o auxílio de protoboards.

Na imagem abaixo podemos ver um exemplo de protoboard com linhas tracejadas que representam como os contatos são interligados internamente. Ser interligado eletricamente é o equivalente a soldar esses pinos ou ligá-los com um fio.



> Jumpers

Jumpers nada mais são que pedaços de fios com conectores nas pontas que facilitam sua inserção e remoção de uma protoboard, sensores, arduínos entre outros equipamentos. Também serão utilizados em todos projetos mostrados aqui.

> LEDs

LEDs são diodos emissores de luz e são componentes muito simples de serem utilizados. Possuem apenas dois terminais, um (o terminal maior) deve ser ligado na alimentação positiva e o outro na negativa.

Existem LEDs de vários tamanhos, cores e tipos. A maioria dos LEDs comuns funcionam com aproximadamente 2V e 20mA, isso é importante pois o arduíno Uno funciona com 5V e, assim como ligar algo de 110V numa tomada de 220V não é uma boa ideia, ligar um LED diretamente no Uno faria, muito provavelmente, o LED queimar. Resolveremos isso com o uso de resistores.

> Resistor

O resistor é um componente que tem por objetivo aumentar a resistência de um "caminho" elétrico (lembre-se da propriedade resistên-

cia que vimos anteriormente). Tem também apenas dois terminais e, como não tem polaridade, pode ser ligado em qualquer direção, ou seja, os terminais podem ser invertidos sem problemas.

A maioria dos resistores têm faixas coloridas pintadas neles. Essas faixas são um código de cores usado no mundo todo que representa o valor da resistência do resistor.

Aqui estaremos usando os resistores apenas para limitar a corrente que circula pelos LEDs, de forma que estes não queimem ao serem ligados no arduíno. Para isso usaremos resistores de 220Ω, representados normalmente pelas faixas vermelho, vermelho, marrom e dourado.

> Potenciômetro

Potenciômetros são resistores variáveis, ou seja, podemos ajustar seu valor de resistência através de algum tipo de controle. Normalmente este controle é mecânico e feito pela ação humana.

A regulagem de velocidade de ventiladores de teto, por exemplo, é normalmente feita por potenciômetros.

Possuem três pinos e a forma de conexão depende de cada circuito.

> Botão

Existem vários tipos de botão, mas fundamentalmente o botão possui apenas dois terminais que, por padrão, não são conectados entre si. Quando apertamos o botão, um contato é fechado entre estes terminais, ou seja, ele passa a se comportar como um fio, ligando um terminal ao outro. Ao soltar, uma mola (ou componente com efeito mola) faz o botão retornar ao estado original, ou seja, terminais abertos (não conectados).

O nome desse botão descrito é *push button* normalmente aberto.

> Sensor de proximidade ultrassônico

O sensor ultrassônico que estaremos utilizando aqui é o HC-SR04, muito famoso e extremamente adaptado para fácil integração com o arduíno de forma que se tornou já quase um padrão.

Porém é sempre bom lembrar que o que faz um sensor ser ultrassônico é a tecnologia utilizada: o ultrassom. O que faz um sensor ser de proximidade é que ele mede distâncias. Assim sendo, é possível que existam diversos sensores ultrassônicos de proximidade, cada um com suas características e forma de utilização próprias. E isso é verdade também para quaisquer outros sensores.

Um ultrassom é um som de frequência tão elevada que o ouvido humano não é capaz de ouvir. E sim, o princípio de funcionamento deste sensor é o mesmo utilizado por aparelhos de ultrassonografia.

No HC-SR04 nós temos quatro terminais. Dois para alimentação: Vcc (5V) e GND e dois para controle: Trig e Echo. Quando acionamos o pino Trig através do código, o sensor emite uma série de pulsos ultrassônicos que batem no objeto mais próximo e retornam em direção ao sensor (efeito eco, daí o nome do outro pino). Quando o sensor reconhece esse retorno, ativa o pino Echo, que estamos monitorando através do arduíno. Conhecendo a velocidade do som no meio (normalmente o ar) e o tempo que demorou entre acionarmos o Trig e recebermos o retorno no Echo, podemos calcular a distância que o objeto que gerou este eco está do sensor.

> Sensor de presença PIR

Utilizaremos aqui um sensor de presença PIR genérico, próprio para ser usado com arduínos.

Esse sensor possui três terminais, um para alimentação positiva (5V), outro para negativa (GND) e um pino de sinal (signal). Este pino de sinal é o que iremos monitorar pelo arduíno.

Quando o sensor reconhece alguma presença em seu campo de "visão", aciona o pino de sinal, colocando nível lógico 1 (5V) nele e, após um certo tempo, desliga esse pino, retornando para GND. Normalmente a distância de "visão" e o tempo que o pino de sinal fica ativado são reguláveis através de pequenos potenciômetros no sensor.

> Servo Motor

Existem diversos tipos de motores. O servo motor é um tipo de motor especial em que podemos especificar para qual ângulo o motor deve se mover. A maioria dos servo motores tem um giro máximo de 180°.

Estaremos utilizando um micro servo aqui. É sempre importante verificar a tensão de funcionamento de seu motor. Nosso modelo usa 5V e consome uma baixa corrente, podendo ser ligado diretamente ao Uno.

Possui três terminais, um para alimentação positiva (5V no nosso caso), um para negativa (GND) e um de controle, também conhecido como sinal (signal) que é justamente o utilizado para informar ao servo para qual ângulo girar. Esse pino de sinal deve ser ligado a um pino PWM do arduíno.

> Relé

Um relé possui três terminais, o normalmente aberto (NA ou NO em inglês), o normalmente fechado (NF ou NC) e um móvel, conhecido como comum (COM).

Uma mola mantém o terminal móvel conectado ao normalmente fechado, daí seu nome: está com contato elétrico fechado quando não fazemos nada. Através de outros dois terminais, podemos acionar uma bobina que funciona como um eletroímã, atraindo o terminal móvel, que se desconecta do normalmente fechado e se conecta ao normalmente aberto. Quando tiramos energia da bobina, a mola faz com que o terminal comum retorne à posição original (conectado ao normalmente fechado).

Como não há contato elétrico entre a bobina e os terminais controlados, podemos, através de um controle simples com o arduíno, acionar cargas com tensões, correntes e até mesmo tipos de tensões diferentes da do arduíno.

Aqui estaremos utilizando um módulo relé para arduíno, que já vem com o relé e alguns outros componentes auxiliares para nos facilitar. Iremos utilizá-lo para acionar uma lâmpada e uma válvula solenóide, ambas de 127Vac (volts em corrente alternada).

Na hora de selecionar o seu, preste atenção na tensão de funcionamento da bobina, que deve ser de 5V já que esta é a tensão disponível no arduíno Uno. Preste atenção também na máxima tensão e corrente nos terminais controlados, que devem ser superiores às especificações do componente que você deseja controlar, ou seja, o relé tem que aguentar mais do que você irá exigir dele.

> Válvula solenoide

Tem um funcionamento parecido ao do relé. Possui uma bobina que quando acionada atrai um componente da válvula, abrindo-a e permitindo a passagem de fluido. Quando a bobina é desenergizada uma mola retorna os componentes pros lugares originais, fechando a passagem de fluido.

O funcionamento descrito acima é de uma válvula normalmente fechada. Existem também as válvulas normalmente abertas, que funcionam de maneira inversa: fecham quando energizamos a bobina.

Existem diversos tipos de bobinas, algumas DC e outras AC. Preste bastante atenção ao selecionar a sua. Optamos aqui por uma válvula normalmente fechada acionada com 127Vac. Por se tratar de um componente AC e o arduíno (que estaremos usando para controlá-la) ser DC, precisamos utilizar um relé entre eles.

PROGRAMAÇÃO

Através da programação dizemos pro arduíno como interpretar os dados dos sensores e o que queremos fazer com estas informações.

Para programarmos o arduíno vamos precisar baixar um programa chamado IDE do arduino, que pode ser encontrado no site oficial: www.arduino.cc/en/software.

Escolha a versão mais apropriada para seu sistema operacional e depois clique em “JUST DOWNLOAD”.

A linguagem de programação utilizada é própria do arduíno e é baseada em c++. Vale mencionar que é toda em inglês, mas é totalmente possível aprender os comandos sem saber o idioma.

Também é importante destacar que a linguagem é ***case sensitive***, ou seja, faz diferença usar letras maiúsculas ou minúsculas. "Setup" não é a mesma coisa que "setup", por exemplo.

O ponto e vírgula (;) é utilizado para indicar o fim de uma linha de instruções. A **maioria** das linhas **terá** um ponto e vírgula no final, mas isso **não é verdade para todas** as linhas, então preste bastante atenção neste detalhe. O uso incorreto do ponto e vírgula **irá gerar um erro**.

> setup

```
void setup() {


}
```

É uma das funções principais do arduíno e estará presente em todos os seus códigos. Assim como toda função, seu escopo é definido pelas chaves, ou seja, tudo que estiver entre a abertura de chaves e a próxima chave de fechamento faz parte da função.

Essa função é executada somente uma vez quando o código é iniciado.

> loop

```
void loop() {


}
```

É outra função básica do arduíno e também estará presente em todo código.

É executada em repetição (loop) infinita, ou seja, quando o arduíno reconhece que acabou de executar tudo que está dentro desta função, ele simplesmente volta pro início dela e repete tudo.

> Variáveis e seus tipos

```
int leitura_pot = 0;
int controle_led = 0;
```

Variáveis são espaços na memória que servem para armazenarmos alguma informação temporariamente. Essas informações podem ser lidas e modificadas ao longo do código.

Existem diversos tipos de variáveis, cada uma permite armazenar um determinado tipo de informação. As que utilizaremos aqui são:

int -> Armazena valores inteiros

float -> Armazena valores reais (“com vírgula”)

long -> Armazena valores inteiros grandes

bool -> Armazena valores lógicos, ou seja, 0 ou 1 (false ou true, respectivamente)

unsigned long -> Armazena valores inteiros longos e somente positivos. É um tipo mais incomum e será utilizado somente para um propósito específico no último projeto.

Aqui é interessante lembrar que em inglês as casas decimais são separadas por ponto e não por vírgula como no Brasil, ou seja, 3,57 causaria um erro; o correto é 3.57.

> Função pinMode

```
void setup()
{
  pinMode(sensor_pir, INPUT);
  pinMode(rele, OUTPUT);
}
```

A função *pinMode* define se um pino deve ser tratado pelo arduíno como entrada ou saída de informações. Isso é necessário para todos os pinos digitais, pois como vimos estes são pinos I/O, ou seja, podem ser tanto entradas como saídas e precisamos dizer isto ao arduíno.

Normalmente declaramos (usamos) esta função dentro da função *setup*, assim o arduíno sabe como tratar os pinos logo quando é ligado.

Toda função é acompanhada de parênteses () quando é chamada e dentro desses parênteses podemos passar alguns parâmetros, separados por vírgula caso existam mais de um. Algumas funções não requerem nenhum parâmetro, como a *setup* e *loop*, já outras, como a *pinMode*, precisam de alguma informação extra para funcionarem corretamente.

No caso da *pinMode*, os parâmetros que precisamos passar são: sobre qual pino estamos falando e se queremos ele como entrada (INPUT) ou saída (OUTPUT), nessa ordem.

> Função digitalWrite

```
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(13, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Através da função *digitalWrite* conseguimos "escrever" algum valor em algum pino digital, ou seja, conseguimos dizer para este pino virar nível 1, efetivamente colocando 5V nele ou virar 0, colocando 0V (GND).

É importante destacar que para o arduíno "escrever" algo em um pino, este pino deve ser uma saída. Uma entrada só pode ser lida.

Essa função também recebe dois parâmetros: o pino ao qual estamos nos referindo e o valor que desejamos escrever, que pode ser HIGH (nível 1 -> 5V) ou LOW (nível 0 -> GND).

> Função digitalRead

```
void loop()
{
  estado_pir = digitalRead(sensor_pir);
```

Utilizamos a digitalRead quando queremos ler a informação presente em algum pino digital.

Lembrando que para um pino ser lido este precisa estar definido como uma entrada.

Precisamos de apenas um parâmetro: o pino a ser lido.

Essa função nos retorna o valor lido, por isso é comum que seja utilizada sendo atribuída a uma variável, assim a variável em questão passa a ter o valor lido e podemos utilizar este valor posteriormente, para algum cálculo ou alguma tomada de decisão.

> Função analogRead

```
void loop()
{
  leitura_pot = analogRead(A0);
```

A analogRead é usada para lermos um valor de uma porta analógica.

Semelhantemente à função anterior, esta retorna o valor lido, que é normalmente armazenado diretamente numa variável e precisamos de apenas um parâmetro: o pino a ser lido.

> Função analogWrite

```
  analogWrite(6, controle_led);
}
```

Comparando com as funções anteriores, o nome desta pode gerar uma ideia errada. Inicialmente podemos pensar que ela serve para fazer uma escrita em portas analógicas, mas como já mencionamos anteriormente o arduíno Uno **não** possui saídas analógicas.

A função analogWrite é usada para escrevermos um valor **PWM** em um dos pinos **digitais** compatíveis, que são os identificados com um til (~) na placa física.

Precisamos passar como argumento o pino que desejamos controlar (que deve estar configurado como saída) e o valor do PWM que desejamos escrever.

No arduíno Uno os registradores de controle do PWM são de 8 bits, isso significa que podemos escrever valores entre 0 e 255 em um pino PWM. A título de curiosidade, como o pino não é realmente analógico, o que ocorre na prática não é uma regulagem da tensão, mas sim do duty cicle. Na prática, em alguns tipos de cargas conseguimos um comportamento parecido com o que teríamos variando a tensão,

como é o caso do controle de brilho de um LED, por exemplo. Nesses casos, quanto mais próximo do valor 255, mais próximo de 5V estará o pino.

> Função delay

```
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Essa função tem o único propósito de aguardar um determinado tempo. O código fica totalmente parado na linha em que o delay foi chamado, esperando o tempo especificado.

Precisamos passar somente um argumento: o tempo desejado, em **milissegundos**.

> Função millis

```
void loop()
{
  tempo = millis() - ultimo_horario;
```

Esta função retorna o valor em **milissegundos** desde que o arduíno foi ligado. Sabendo o tempo de agora e tendo um tempo anterior armazenado em alguma variável, conseguimos determinar quanto tempo se passou desde a última atualização de tal variável.

Utilizamos esta técnica para criar contadores melhores que um simples delay. Eles são melhores pois o código não fica parado enquanto conta o tempo, assim podemos continuar processando outras informações.

A função millis não requer nenhum parâmetro, porém é importante observar que o retorno dela é no formato unsigned long e, portanto, precisa ser armazenado em uma variável de tipo compatível.

> #define

```
#define rele 5
```

Este comando cria uma espécie de apelido. Normalmente o utilizamos para criar apelidos para os pinos.

Isso serve dois propósitos básicos: facilita a leitura do código por um humano, seja você mesmo no futuro ou outro(a) programador(a) que esteja trabalhando em conjunto e facilita a modificação do código caso seja necessário alterar o circuito elétrico fisicamente, pois podemos simplesmente mudar a referência do apelido para o novo pino e o código está pronto, não precisamos "caçar" no código toda vez que usamos o pino para mudar manualmente.

É muito comum vermos códigos que fazem essa funcionalidade com o uso de variáveis, porém como vimos variáveis são espaços de memória e a memória do arduíno (e de qualquer sistema) é limitada. Como esse valor não será alterado ao longo do código, não há motivos para gastarmos espaço de memória com ele, logo o define é uma melhor opção.

> #import

Este comando é utilizado para importarmos outros códigos para "dentro" do nosso, nos permitindo utilizar definições, funções e métodos diretamente.

Podemos fazer isso com os nossos próprios códigos, a fim de criar vários pequenos códigos no lugar de apenas um gigante e assim facilitar a leitura e manutenção. Podemos também usar códigos dos outros (as famosas bibliotecas) desde que por estes autorizados, evitando a necessidade de reinventar a roda.

Um mesmo código pode ter diversos *imports*.

> if e else

```
if (estado_pir == 1){
  digitalWrite(rele, HIGH);
}
else{
  digitalWrite(rele, LOW);
}
```

If significa "se" e else significa "senão". Estes comandos são utilizados para criarmos condições no código e tomarmos determinada ação **se** algo for verdadeiro e outra ação caso não seja.

Veja a estrutura básica na imagem ao lado. Ao lado do *if*, dentro dos parênteses, passamos a condição que deve ser testada. Os comandos dentro do *if* (cujo escopo é delimitado pelas chaves, assim como nas funções) só são executados caso a condição testada seja verdadeira.

O else não requer uma condição para teste, os comandos dentro dele simplesmente são executados caso a condição testada no *if* anterior seja falsa. Caso você queira que nada seja executado se a condição for falsa, pode simplesmente não colocar o *else*, ele não é obrigatório.

Caso o *if* correspondente tenha sido executado, o else é simplesmente ignorado. O escopo do *else* também é delimitado pelas chaves e todo else está associado ao *if* imediatamente acima dele.

PROJETOS

Em todos os projetos abaixo iremos utilizar o arduíno Uno, a protoboard e jumpers conforme necessário.

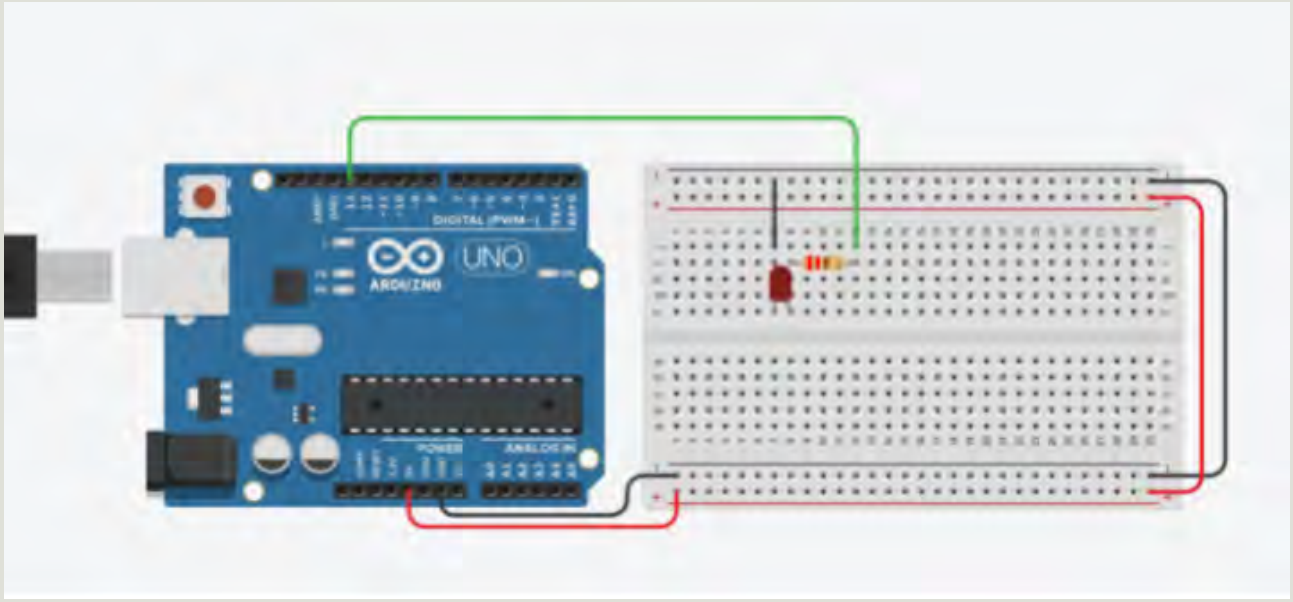
Os projetos com partes mecânicas são apenas um exemplo e devem ser modificados conforme a necessidade.

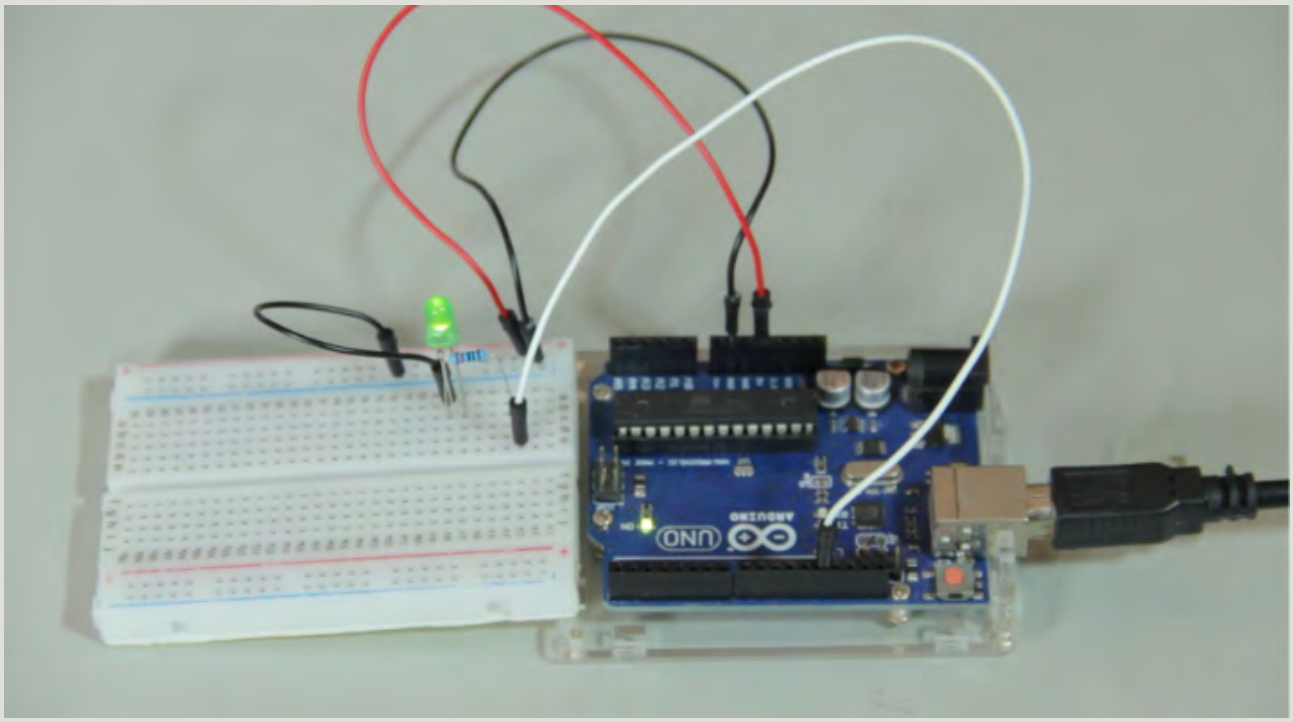
Muito cuidado nos projetos que envolvem eletricidade vinda da tomada. A possibilidade de choque é real e pode ser fatal. Se você não tem certeza do que está fazendo, não faça.

Para demonstrar os circuitos elétricos, estamos utilizando prints do circuito montado num simulador online e gratuito chamado Tinkercad, da empresa Autodesk, que você pode testar aqui www.tinkercad.com. Os projetos dos circuitos e quaisquer comentários são de autoria do mesmo autor desta apostila e não são uma responsabilidade da Autodesk. A reprodução destas imagens é cortesia da Autodesk, Inc. *("Autodesk screen shots reprinted courtesy of Autodesk, Inc.")*.

PISCANDO UM LED

Como nosso primeiro projeto faremos um LED piscar sozinho. Para isso iremos precisar, além dos componentes básicos falados acima (arduíno, protoboard e jumpers), de um LED e de um resistor de 220Ω.





Após a montagem do circuito físico, vamos para o código:

```
1  void setup()
2  {
3    pinMode(13, OUTPUT);
4  }
5
6  void loop()
7  {
8    digitalWrite(13, HIGH);
9    delay(1000);
10   digitalWrite(13, LOW);
11   delay(1000);
12 }
```

Na linha 3 definimos o pino 13, onde ligamos o LED, como saída.

Na linha 8 ligamos o LED colocando nível alto (HIGH) no pino 13. Esperamos 1 segundo (1000 milissegundos) na linha 9, apagamos o LED na linha 10 e esperamos mais 1 segundo na linha 11.

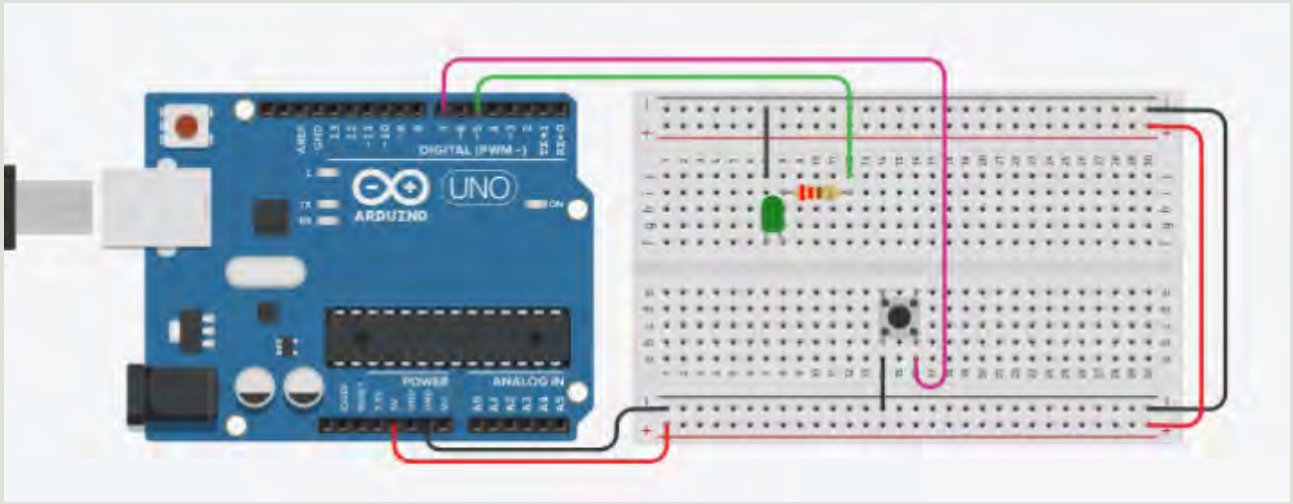
Na linha 12 temos a chave de fechamento da função loop, indicando o fim do escopo desta. Como vimos, a função loop se repete infinitamente, então ao identificar o final na linha 12, o arduino volta pra linha 7.

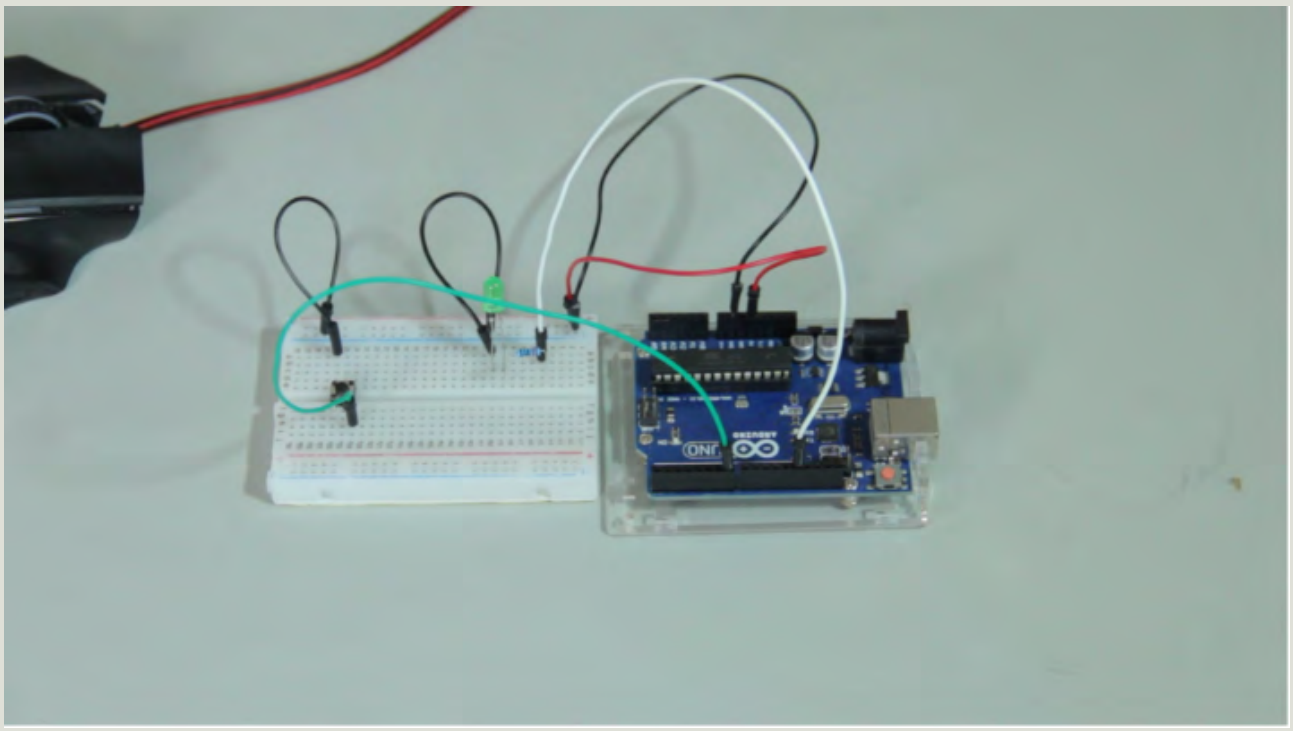
CONTROLE DO LED COM BOTÃO

Agora vamos adicionar um pouco de complexidade controlando o LED com um botão.

O botão só pode estar pressionado ou não pressionado, não existe um "meio termo". Assim sendo, vamos ligá-lo à uma porta digital que deve ser configurada como entrada, já que estaremos entrando com informação no arduino.

Além dos componentes básicos, vamos utilizar um LED, um resistor de 220Ω e um push button.





Agora o código:

```
1  bool leitura_botao = 0;
2
3  void setup()
4  {
5    pinMode(5, OUTPUT);
6    pinMode(7, INPUT_PULLUP);
7  }
8
9  void loop()
10 {
11   leitura_botao = digitalRead(7);
12
13   digitalWrite(5, !leitura_botao);
14 }
```

Na linha 1 estamos definindo a variável chamada “leitura_botao” do tipo bool e inicializando-a com o valor 0. Esta variável irá armazenar o dado se o botão está pressionado ou não.

Na linha 5 definimos o pino 5 como saída e na linha 6 definimos o pino 7 como entrada com *pullup* ativado. Este *pullup* é necessário por estarmos usando um botão e devido a ele, **o arduino irá ler nível 0 quando pressionarmos o botão.**

Na linha 11 fazemos a leitura do pino digital 7 (onde conectamos o botão) e armazenamos o resultado na variável *leitura_botao* que foi criada previamente.

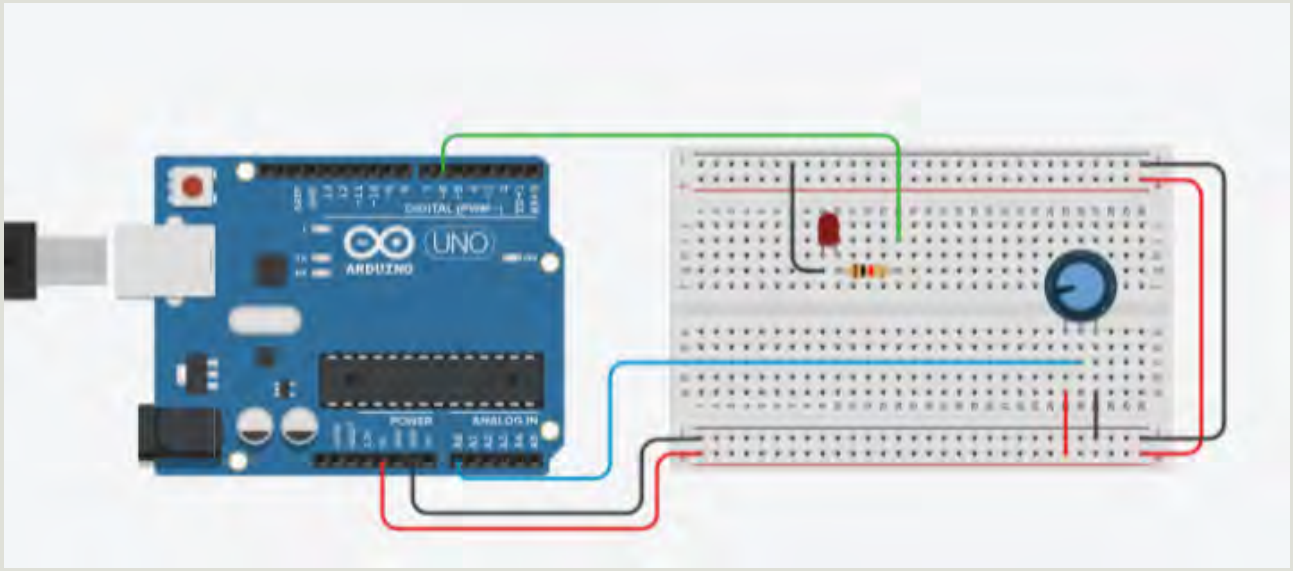
Na linha 13 estamos escrevendo no pino digital 5 (onde está o LED) o inverso do valor da variável *leitura_botao* (sabemos que é o inverso pelo símbolo de exclamação antes do nome da variável). Esse conceito de inverso funciona pois *leitura_botao* é do tipo bool, ou seja, só pode ser 0 ou 1, que representam o nível baixo (LOW) e alto (HIGH), respectivamente. Estamos utilizando o inverso pois como falamos, o arduino irá ler 0 (nível baixo) quando pressionarmos o botão e queremos acender o LED quando pressionarmos o botão. Como o LED acende com nível alto (5V), usamos o inverso.

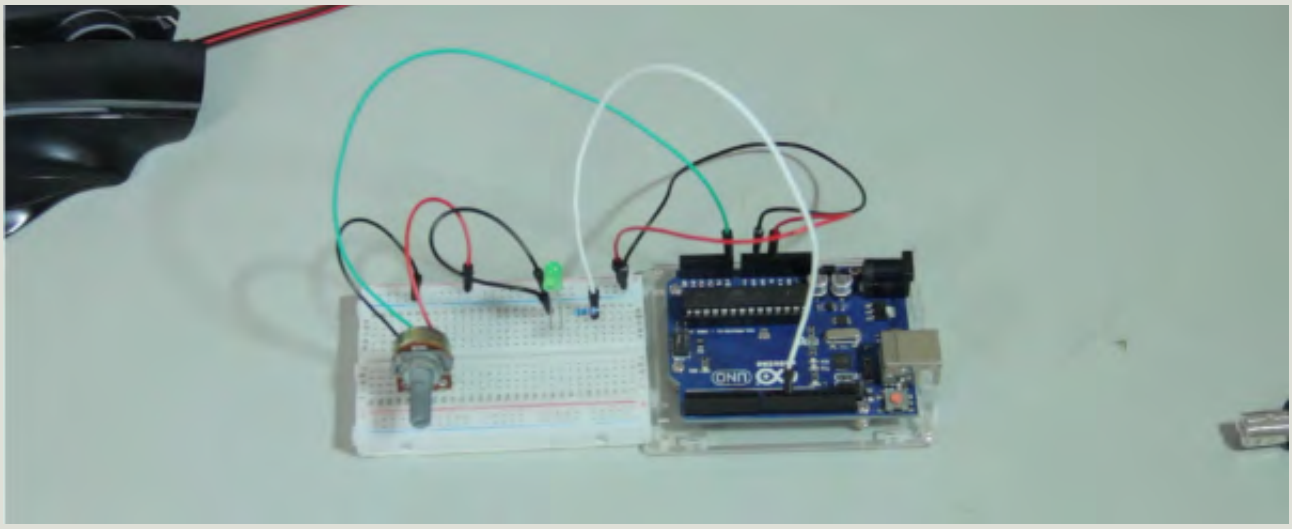
CONTROLE DO LED COM POTENCIÔMETRO

Agora iremos usar nosso primeiro componente analógico, o potenciômetro.

Vamos precisar dos componentes básicos, LED, resistor de 220Ω e um potenciômetro de qualquer valor. Normalmente potenciômetros de 1k e 10k são comuns de serem achados para comprar e irão funcionar bem aqui.

Lembre-se de que, por ser um componente analógico, o potenciômetro deve ser ligado numa porta analógica do arduino e como iremos controlar o brilho do LED, precisamos ligar o LED em um pino PWM (os marcados com til~ na placa).


UNO
ARDUINO
POWER
ANALOG IN



```
1  int leitura_pot = 0;
2  int controle_led = 0;
3
4  void setup()
5  {
6    pinMode(6, OUTPUT);
7  }
8
9  void loop()
10 {
11   leitura_pot = analogRead(A0);
12
13   controle_led = map(leitura_pot, 0, 1023, 0, 255);
14   analogWrite(6, controle_led);
15 }
16
17
18
```

Nas linhas 1 e 2 estamos criando as variáveis que iremos utilizar neste código, ambas do tipo inteiro.

Na linha 6 definimos o pino 6 (onde está o LED) como saída. Como só existem entradas analógicas no arduino, não é obrigatório definirmos o pino do potenciômetro (A0) como entrada.

Na linha 11 estamos efetuando a leitura do pino A0 e armazenando o resultado na variável leitura_pot.

Como comentamos, toda leitura analógica é interpretada pelo arduino Uno como um valor entre 0 e 1023, enquanto pinos PWM, onde ligamos o LED, entendem valores entre 0 e 255. Por esse motivo, é necessário fazer uma equivalência desses valores. Fazemos isto através da função map na linha 13.

Na linha 14 escrevemos no pino 6 (onde está o LED) o valor da variável controle_led, que já está entre 0 e 255 conforme explicado no parágrafo anterior.

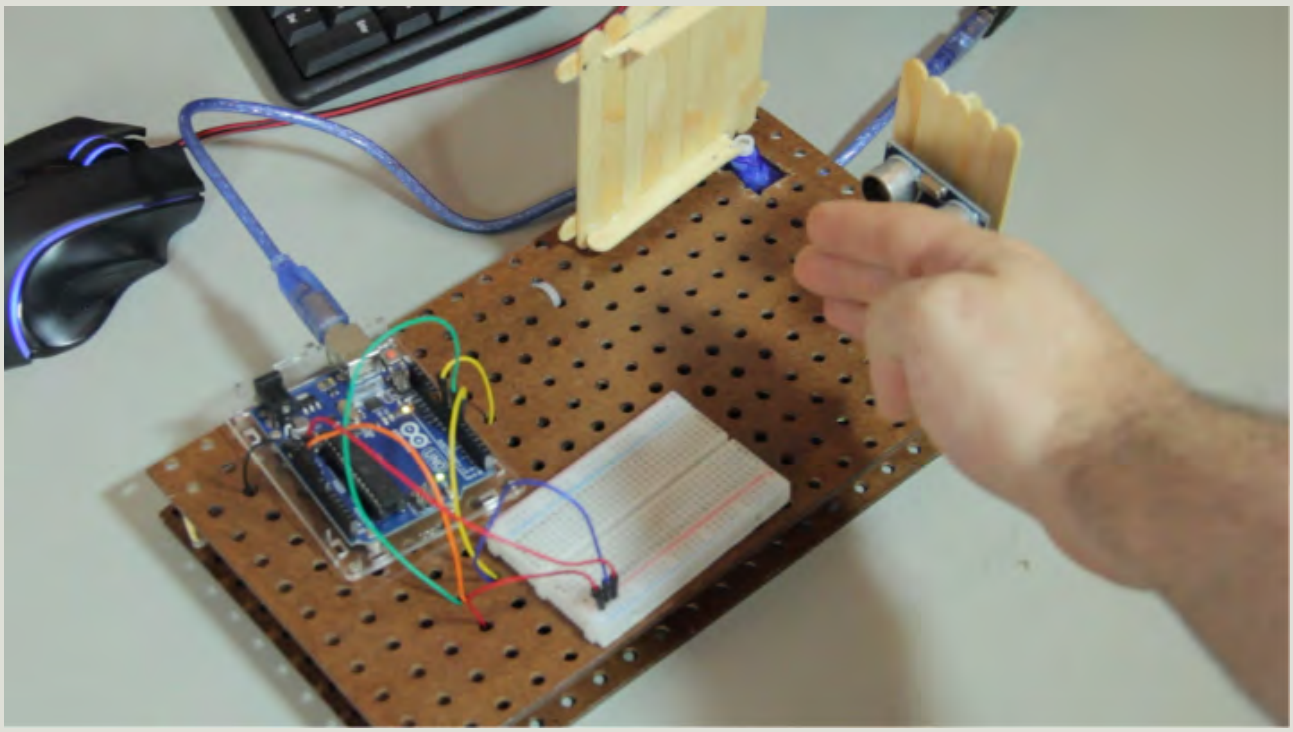
PORTÃO AUTOMÁTICO COM USO DE SERVO MOTOR E SENSOR ULTRASSÔNICO

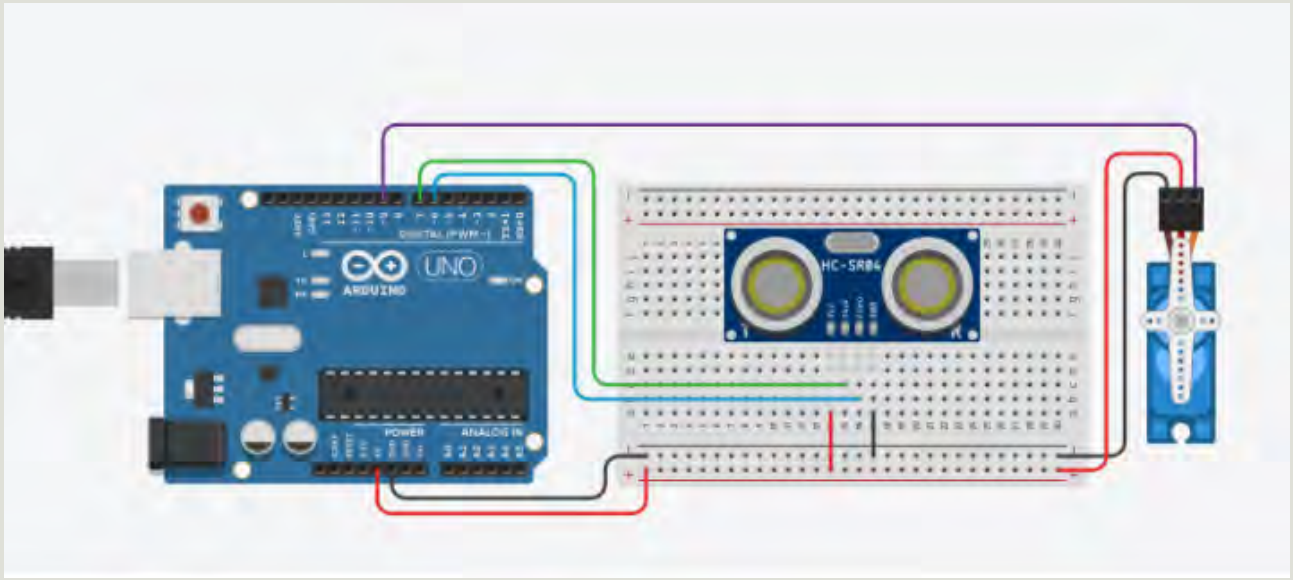
A partir de agora os projetos têm uma parte mecânica maior. Esta parte serve apenas como um exemplo e deve ser modificada de acordo com sua necessidade.

Para o portão automático iremos utilizar os componentes básicos, um micro servo-motor de 5V e o sensor HC-SR04.

Para a estrutura mecânica usamos uma madeira como base (que pode facilmente ser substituída por papelão rígido, por exemplo), palitos de sorvete para o portão, cola quente, fita dupla face, parafusos e um formão para fazer os cortes na madeira (que pode ser um estilete se você usar o papelão). Sua imaginação é o limite aqui. Apenas **lembre-se de tomar os devidos cuidados e usar as proteções necessárias para trabalho com componentes quentes e cortantes.**

Para este projeto vamos precisar também instalar nossa primeira biblioteca. Para isso, clique aqui para acessar o site, clique no botão verde "Code" e em "Download ZIP". Agora vá na IDE do arduino e clique em "Sketch -> Include Library -> Add .ZIP Library...", selecione o arquivo que acabamos de baixar e clique em "Open". Caso sua IDE esteja em português, o caminho é: "Sketch -> Incluir biblioteca -> Adicionar biblioteca .ZIP", selecione o arquivo baixado e clique em "Abrir".




UNO
ARDUINO
HC-SR04

```
1  #include <Ultrasonic.h>
2  #include <Servo.h>
3
4  #define trig 7
5  #define echo 6
6
7  Ultrasonic sensor(trig, echo);
8
9  Servo meu_servo;
10 long micro_segundos = 0;
11 float distancia = 0;
12
13 void setup() {
14   pinMode(trig, OUTPUT);
15   pinMode(echo, INPUT);
16   meu_servo.attach(9);
17   meu_servo.write(82);
18 }
19
20 void loop() {
21   micro_segundos = sensor.timing();
22   distancia = sensor.convert(micro_segundos, Ultrasonic::CM);
23
24   if (distancia < 7){
25     meu_servo.write(0);
26     delay(2000);
27     meu_servo.write(82);
28   }
29
30   delay(50);
31 }
```

Na primeira linha incluímos a biblioteca “Ultrasonic”, que irá nos auxiliar no uso do sensor ultrassônico. Na linha 2 incluímos a biblioteca “Servo” que irá nos auxiliar no uso do servo motor.

Nas linhas 4 e 5 estamos dando apelidos para os pinos de interesse. Em todo lugar do código que escrevermos “trig”, o arduino irá entender “pino 7”. Analogamente, o apelido do pino 6 é “echo”.

Na linha 7 estamos falando para o arduino que teremos um sensor ultrassônico ligados no pino 7 e 6 (lembre-se dos apelidos que acabamos de definir) e que o nome desse sensor ao longo do código será "sensor".

Na linha 9 dizemos que usaremos um servo motor e vamos chamá-lo de "meu_servo".

Na linha 10 criamos a variável que irá armazenar o tempo que o som (do ultrassom) levará entre ser emitido e recebido de volta na forma de eco pelo sensor. Na linha 11 criamos a variável que irá armazenar esse valor convertido para centímetros.

Dentro do setup definimos o pino trig como saída (linha 14) e o pino echo como entrada (linha 15).

Na linha 16 estamos informando que nosso servo motor (meu_servo) será ligado no pino 9. Já na linha 17 estamos mandando este servo se mover para 82°, valor que, de acordo com nossos testes, fechava o portão na nossa montagem mecânica.

No loop a primeira coisa que fazemos é medir o tempo entre a emissão do som e seu retorno através do método "timing" (linha 21). O valor é diretamente armazenado na variável "micro_segundos" que já havia sido criada para esta função.

Na linha 22 convertemos o valor lido acima para CM e armazenamos o resultado em "distancia".

Perceba que nos dois últimos comandos usamos o "sensor", nome que demos ao nosso sensor ultrassônico lá na linha 7. Se você alterar lá em cima, precisa alterar aqui também.

Na linha 24 fazemos um if testando se a distância medida é menor que 7, ou seja, se o objeto está a menos de 7 cm do sensor. Caso esteja, as linhas 25, 26 e 27 serão executadas. Caso não esteja, ou seja, o objeto esteja a mais que 7 cm do sensor, as linhas 25, 26 e 27 serão ignoradas. Perceba que não temos um bloco else, portanto não há uma funcionalidade a ser feita caso a distância seja maior que 7 cm.

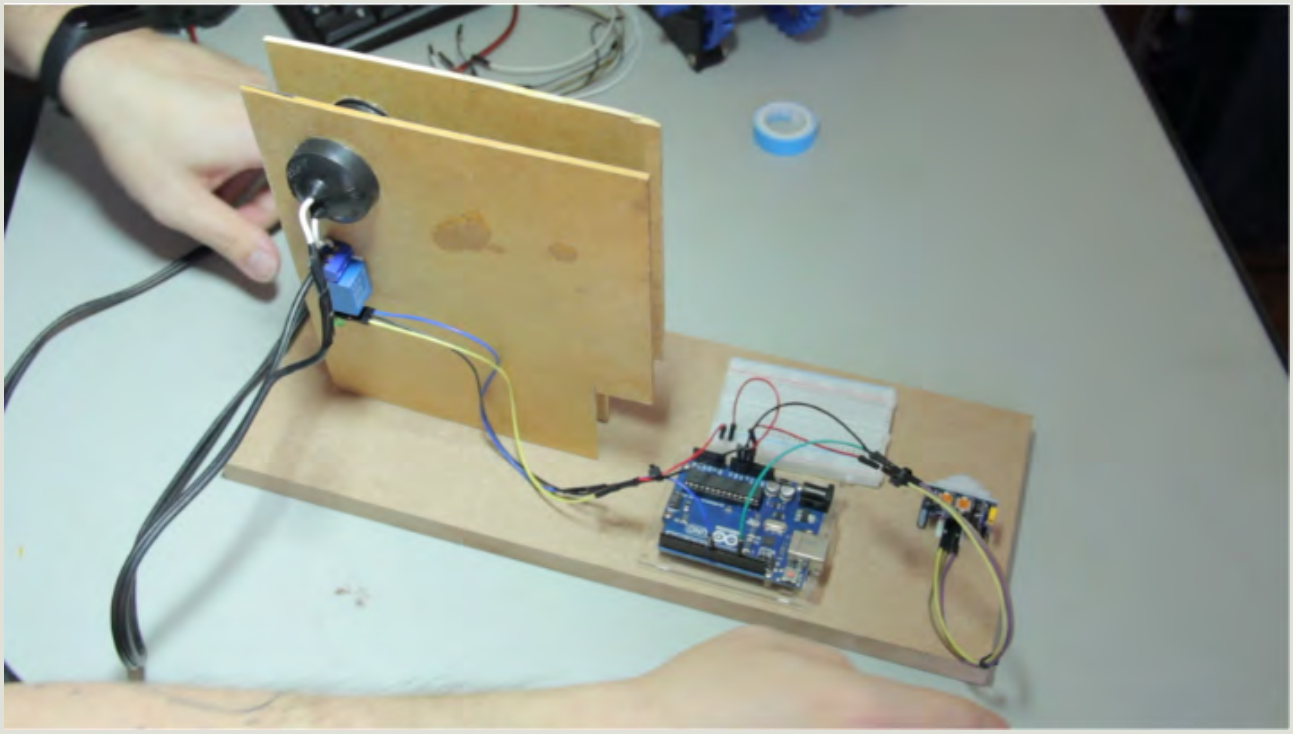
Na linha 25 movemos o servo para 0°, abrindo o portão. Na linha 26 esperamos um tempo de 2 segundos, mantendo o portão aberto. Na linha 27 voltamos o servo para a posição de 82°, fechando o portão.

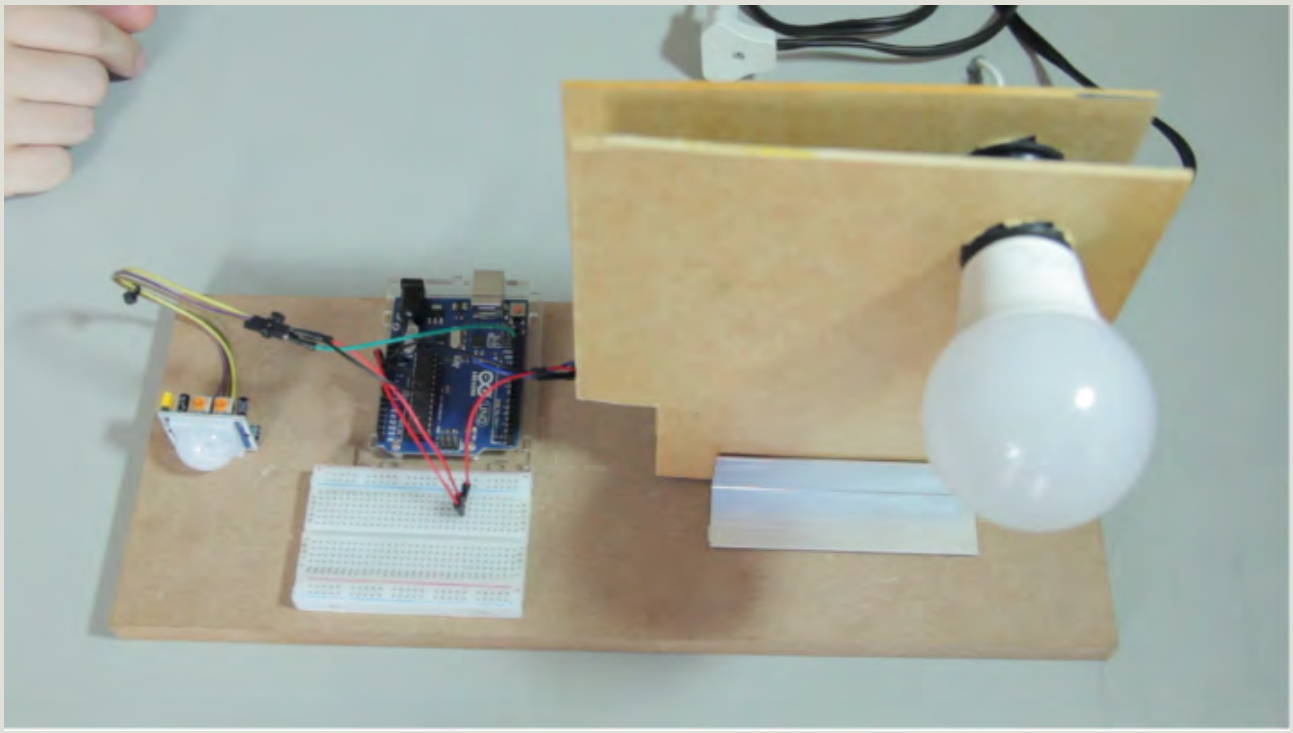
Na linha 30 temos um pequeno *delay* que serve apenas para o arduino não ficar repetindo o loop muito rápido e efetuando centenas de milhares de leituras do sensor ultrassônico. Normalmente as próprias fabricantes dos sensores ultrassônicos recomendam um pequeno *delay* como este pois se você jogar muitos feixes de som em sequência, o eco de um pode influenciar no outro. Teste alguns valores aqui e veja o que funciona melhor para a sua aplicação.

ILUMINAÇÃO AUTOMATIZADA COM USO DE SENSOR DE PRESENÇA PIR E RELÉ

Os materiais necessários, além dos básicos, são: sensor de presença PIR, um módulo relé para arduino, bocal (soquete) para lâmpada, uma lâmpada para testes e um pedaço de fio com uma tomada na ponta.

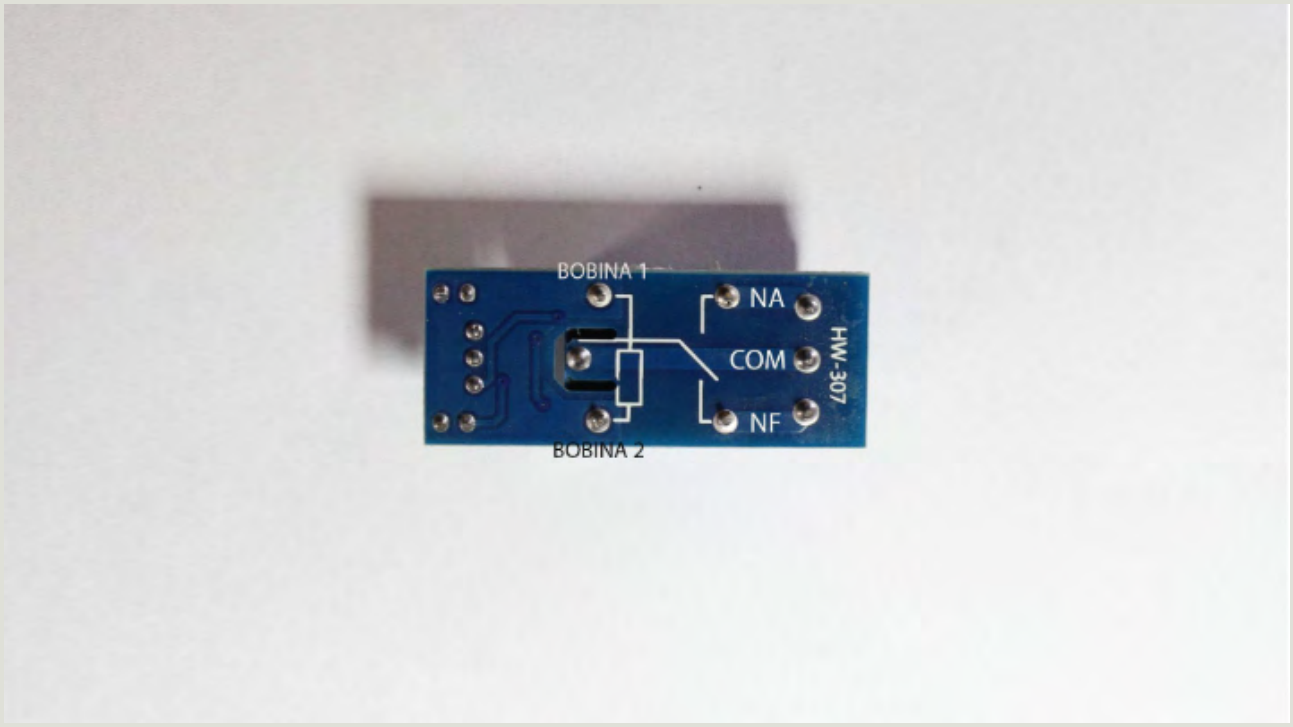
Esse é nosso primeiro projeto usando energia AC. **Lembre-se que tanto 127Vac quanto 220Vac podem causar choques fatais.** Use sempre os equipamentos e proteções necessárias, tome muito cuidado e **se não tiver certeza do que está fazendo, não faça!** Não nos responsabilizamos por nenhum acidente.





Da tomada temos dois fios, um deve ser ligado diretamente em um dos fios do bocal e o outro deve ser conectado ao terminal comum (COM) do módulo relé.

O outro fio do bocal deve ser conectado ao normalmente aberto (NA) do módulo relé.



Devemos conectar o GND do arduino aos GNDs do sensor PIR e do módulo relé, bem como devemos conectar os 5V do arduino aos VCCs do sensor e do módulo relé. Como só temos um pino de 5V no arduino, usamos a protoboard para conectar os dois componentes.

O pino de sinal do sensor PIR bem como o pino de sinal do módulo relé devem ser conectados em pinos digitais do arduino. Aqui usamos o pino 13 e 8, respectivamente.

```
1  #define sensor_pir 13
2  #define rele 8
3
4  bool estado_pir = 0;
5
6  void setup()
7  {
8    pinMode(sensor_pir, INPUT);
9    pinMode(rele, OUTPUT);
10 }
11
12 void loop()
13 {
14   estado_pir = digitalRead(sensor_pir);
15
16   if (estado_pir == 1){
17     digitalWrite(rele, LOW);
18     delay(2000);
19   }
20   else{
21     digitalWrite(rele, HIGH);
22   }
23 }
```

Nas duas primeiras linhas estamos dando apelidos para os pinos de interesse. Em todo lugar do código que escrevermos "sensor_pir", o arduino irá entender "pino 13". Analogamente, o apelido do pino 8 é "rele".

Na linha 4 estamos criando a variável "estado_pir", do tipo bool e inicializando-a com o valor 0. Nela iremos armazenar a leitura do sensor PIR, que representa se temos presença ou não no ambiente.

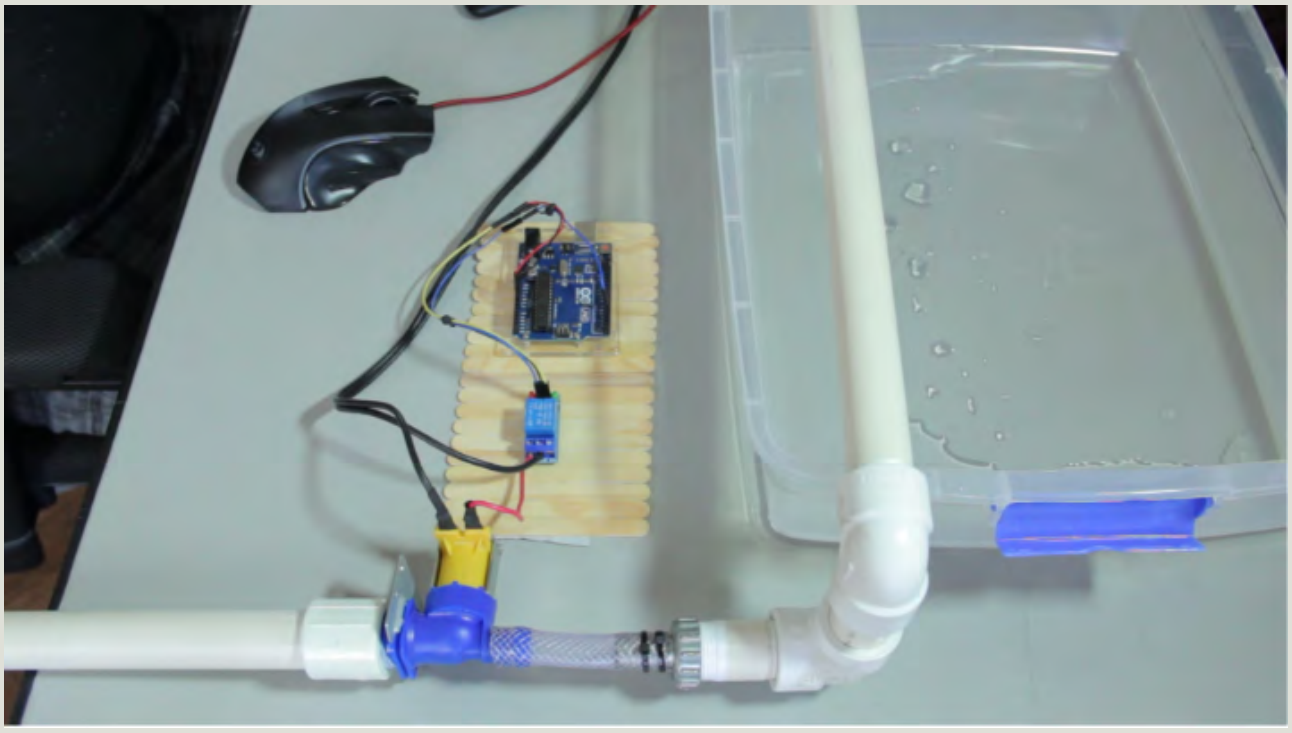
Dentro do setup (linhas 8 e 9) definimos nossas entradas e saídas.

Iniciamos o loop fazendo a leitura do sensor e armazenando o resultado na variável "estado_pir" (linha 14). Em seguida fazemos um if verificando se a variável "estado_pir" é igual (==) a 1, ou seja, se o sensor foi ativado. Caso o sensor esteja ativado (igual a 1), acionamos o relé (que irá acender a lâmpada) através da linha 17. **Perceba que este módulo relé é acionado com nível baixo (LOW).** Em sequência esperamos 2 segundos (linha 18). Neste caso (if executado), o else será pulado, ou seja, as linhas 20, 21 e 22 serão ignoradas pelo arduino.

Caso a condição testada no if seja falsa, ou seja, a variável "estado_pir" seja 0, as linhas 17 e 18 não serão executadas e a linha 21 será. Na linha 21 estamos desligando o relé (e consequentemente a lâmpada).

SISTEMA DE IRRIGAÇÃO COM VÁLVULA SOLENÓIDE

Neste nosso último projeto estaremos utilizando a válvula solenóide para criarmos um sistema de irrigação automatizado.



Escolhemos uma válvula normalmente fechada que é acionada por 127Vac. Por ser acionada com corrente alternada, precisamos de um módulo relé para controlá-la com o arduino.

Assim sendo, o material necessário, além dos itens básicos, são uma válvula solenóide e o módulo relé, que já vimos no último projeto. Vamos precisar também de um fio com uma tomada na ponta para ligarmos a válvula.

Um dos fios da tomada deve ser ligado diretamente em um dos conectores da válvula. O outro fio da tomada deve ser ligado ao pino comum (COM) do módulo relé.

Já o outro conector da válvula deve ser conectado ao pino normalmente aberto (NA) do módulo relé.

Também precisamos conectar ao módulo relé os 5V e GND vindos do arduino. O pino de controle deve ser ligado em alguma porta digital do arduino, usamos aqui a porta 5.

O controle será feito através de um temporizador: a cada hora iremos ativar a irrigação por 10 minutos.

```
#define rele 5

unsigned long ultimo_horario = 0;
unsigned long tempo = 0;

void setup()
{
  pinMode(rele, OUTPUT);
  digitalWrite(rele, HIGH);
  ultimo_horario = millis();
}

void loop()
{
  tempo = millis() - ultimo_horario;

  if (tempo == 3600000){ //Tempo entre irrigações (1h = 36000
    digitalWrite(rele, LOW);
    delay(60000); //Quanto tempo ficará irrigando
    digitalWrite(rele, HIGH);
    ultimo_horario = millis();
  }
```

Na linha 1 definimos que o pino 5 terá o apelido “rele”.

Nas linhas 3 e 4 estamos criando as variáveis que iremos utilizar.

No setup começamos definindo o pino “rele” como saída (linha 8), em seguida mandamos este pino ficar com nível alto que, conforme explicado no último projeto, **desliga** o relé. Fazemos isso aqui para garantir que a válvula fique fechada desde o início do código, evitando molhar algo sem querer. Por último usamos a função millis para armazenar na variável “ultimo_horario” quanto tempo o arduino está ligado.

Na linha 15 estamos dizendo que a variável “tempo” irá receber o valor de millis **agora** menos o valor armazenado em “ultimo_horario”. Lembre-se que a função millis retorna, em milissegundos, quanto tempo se passou desde que o arduino foi ligado até agora. Assim sendo, a variável “tempo” funciona como um contador, já que, conforme o arduino fica ligado, seu valor só aumenta até modificarmos “ultimo_horario”.

Na linha 17 testamos se a variável “tempo” atingiu o valor 3600000, equivalente a 1 hora.

Caso seja, as linhas 18 até 21 serão executadas. Caso não seja, simplesmente não faremos nada (perceba a ausência do else) e o código simplesmente retorna para a linha 15.

Na linha 18 acionamos o relé que por sua vez aciona a válvula solenóide, iniciando a irrigação. Na linha 19 esperamos 10 minutos (60000 milissegundos). Na linha 20 desligamos o relé, parando a irrigação. Por fim, na linha 21 atualizamos o valor da variável “ultimo_horario” para o valor de millis atual. Na prática isso representa o horário em que ativamos a irrigação pela última vez, daí a escolha do nome da variável.