02

INGREDIENTES

INTERFACE DE NAVE ESPACIAL

SU PLACA ARDUINO VA A PROTAGONIZAR UNA PELÍCULA DE CIENCIA FICCIÓN

Descubra: entrada y salida digital, su primer programa, las variables

Tiempo: **45 MINUTOS**

Nivel: **bajo**

Proyecto en el que se basa: **1**

Ahora que tiene los fundamentos de la electricidad bajo control, es el momento de pasar a controlar cosas con su Arduino. En este proyecto, va a construir algo que podría ser una interface de una nave espacial de una película de ciencia ficción de los años 70. Va a montar un panel de control con un pulsador y luces que se encienden cuando presiona el pulsador. Puede decidir que indican las luces "Activar hiper-velocidad" o "¡Disparar los rayos laser!". Un diodo LED verde permanecerá encendido hasta que pulse el botón. Cuando Arduino reciba la señal del botón pulsado, la luz verde se apaga y se encienden otras dos luces que comienzan a parpadear.

Los terminales o pins digitales de Arduino solo pueden tener dos estados: cuando hay voltaje en un pin de entrada, y cuando no lo hay. Este tipo de entrada es normalmente llamada digital (o algunas veces binaria, por tener dos estados). Estos estados se refieren comúnmente como **HIGH (alto)** y **LOW (bajo)**. **HIGH** es lo mismo que decir "¡aquí hay tensión!" y **LOW** indica "¡no hay tensión en este pin!". Cuando pone un pin de **SALIDA (OUTPUT)** en estado **HIGH** utilizando el comando llamado **digitalWrite()**, está activando-lo. Si mide el voltaje entre este pin y masa, obtendrá una tensión de 5 voltios. Cuando pone un pin de **SALIDA (OUTPUT)** en estado **LOW**, está apagando-lo.

Los pins digitales de Arduino pueden trabajar como entradas o como salidas. En su código, los configurará dependiendo de cual sea su función dentro del circuito. Cuando los pins se configuran como salidas, entonces podrá encender componentes como los diodos LEDs. Si se configuran como entradas, podrá verificar si un pulsador está siendo presionado o no. Ya que los pins 0 y 1 son usados para comunicación con el ordenador, es mejor comenzar con el pin 2.

MONTANDO EL CIRCUITO

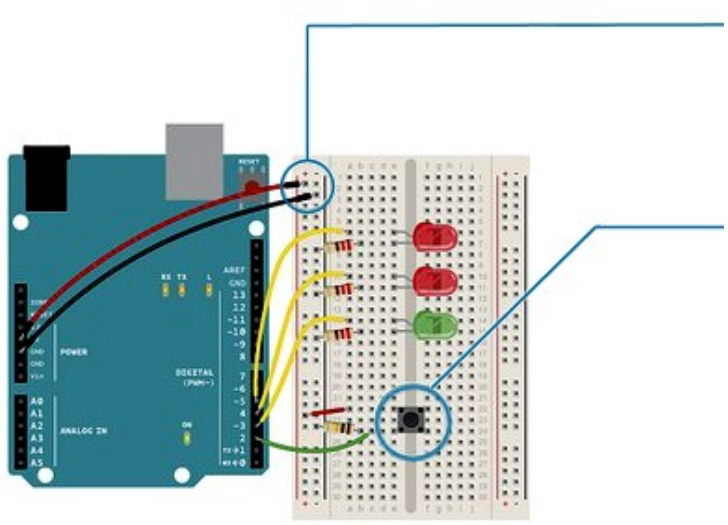
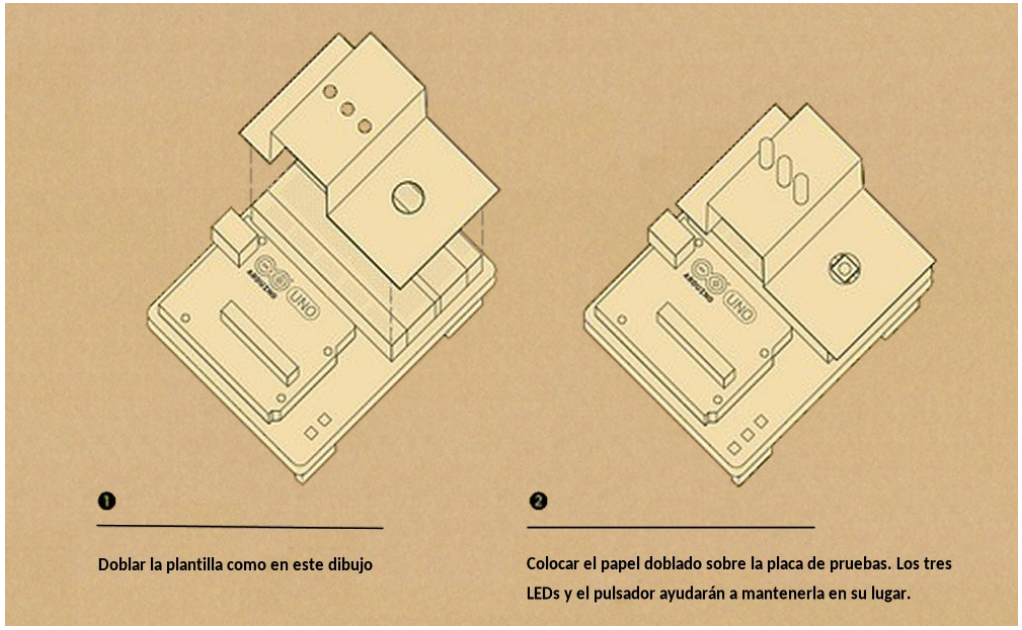


Figura 1

1. Conectar la placa de pruebas a las conexiones de 5V y masa de Arduino, igual que en el proyecto anterior. Colocar los dos diodos LED rojos y el LED verde sobre la placa de pruebas. Conectar el cátodo (patilla corta) de cada LED masa a través de un resistencia de 220 ohmios. Conectar el ánodo (patilla larga) del LED verde al pin 3 de Arduino. Conectar los ánodos de los LEDs rojos a los pins 4 y 5 respectivamente.

2. Colocar el pulsador sobre la placa de pruebas como hizo en el proyecto anterior. Conectar un extremo a la alimentación, y el otro terminal del pulsador al 2 de Arduino. También necesita añadir una resistencia de 10K ohmios desde masa al pin del interruptor que va conectado a Arduino. Esta resistencia de puesta a cero conecta el pin a masa cuando el pulsador está abierto, así que Arduino lee LOW cuando no hay tensión en ese pin del pulsador.

Puede cubrir la placa de pruebas con una plantilla suministrada en el kit. O puede decorarla para fabricar su propio sistema de control de la nave. Las luces que se encienden y se apagan no nos dicen nada, pero cuando se colocan en un panel de control y se les coloca una etiquetas estos sí que sabemos para que valen. ¿Que quiere que indique la luz verde?, ¿Qué significa el parpadeo de los LEDs rojos?, ¡usted decide!.



EL CÓDIGO

Algunas notas antes de comenzar

Cada programa de Arduino tiene dos funciones básicas principales. Las funciones son partes de un programa de ordenador que ejecuta instrucciones específicas. Las funciones tienen un único nombre, y son "llamadas" cuando se necesitan. Estas dos funciones principales en un programa de Arduino son llamadas con **setup()** y **loop()**, la cuales necesitan ser declaradas, esto quiere decir que es necesario indicarle a Arduino lo que estas funciones harán. En la siguiente página y en su parte superior dentro del primer cuadro se puede ver como hay que declarar estas funciones **setup()** y **loop ()**. Lo primero que hay que hacer antes de meterse en la parte principal del programa es crear una variable. Las variables son nombres que se utilizan para guardar información dentro de la memoria de Arduino. Los valores de las variables pueden cambiar dependiendo de la instrucciones que contenga el programa. El nombre de las variables deben de ser una descripción de tipo de información que contienen. Por ejemplo, una variable llamada **SwitchState** (estado del pulsador) le dice lo que está guardando: el estado de un pulsador. Por otra parte, una variable llamada "**x**" no dice mucho acerca del tipo de información que guarda.

Vamos a comenzar a programar

Para crear una variable, es necesario declarar de que *tipo* se trata. Una variable de *tipo* **int** guardará un numero entero (también llamado integer); eso significa que almacena cualquier número sin decimales. Cuando se declara una variable, normalmente también se le asigna a la vez un valor inicial. Las declaraciones de la variables siempre deben de finalizar con un punto y coma (;).

Configurando como funciona un pin

La función **setup()** solo se ejecuta una vez, cuando Arduino recibe la alimentación para funcionar. Esta función define un bloque, el cual se abre con una llave "{" y se cierra con otra llave "}", en donde se escriben las instrucciones que configuran los pins digitales de Arduino como entradas o como salidas, usando para ello una función llamada **pindMode()**. Los pins conectados a los LEDs serán **OUTPUT**s y el pin de un pulsador será una **INPUT**.

Crear la función loop

La función **loop()** se ejecuta continuamente después de que la función **setup()** se haya completado. A través de las instrucciones que se incluyen dentro del bloque después de la función **loop()** es donde se comprobará el voltaje de las entradas y si las salidas están activadas o desactivadas. Para verificar el nivel de voltaje de una entrada digital, se utiliza la función **digitalRead()** la cual comprueba el voltaje en el pin elegido. Para saber que pin debe de verificar **digitalRead()** espera un *argumento*.
Los argumentos son información que se le pasa a las funciones diciéndoles como deben de realizar su trabajo. Por ejemplo, **digitalRead()** necesita un argumento: que pin debe verificar. En el programa de Interface de la Nave Espacial, **digitalRead()** va a verificar el estado del pin 2 para después almacenar el valor dentro de la variable switchState.
Si hay voltaje en el pin 2 cuando **digitalRead()** es llamada, entonces la variable **switchState** almacena un valor **HIGH** (o 1). Si no hay voltaje en el pin 2, **switchState** almacenará el valor **LOW** (o 0).


```
void setup (){
}

void loop (){
}
```

{ Las llaves }

Cualquier código que escriba entre llaves será ejecutado cuando la función es llamada

```
1 int switchState = 0;
```

```
2 void setup (){
3   pinMode(3, OUTPUT);
4   pinMode(4, OUTPUT);
5   pinMode(5, OUTPUT);
6   pinMode(2, INPUT);
7 }
```

Mayúsculas y minúsculas

Poner atención a la hora de escribir en mayúsculas y minúsculas dentro del código. Por ejemplo, **pinMode** es el nombre de una instrucción, pero pinmode producirá un error.

```
8 void loop (){
9   switchState = digitalRead(2);
10  // Esto es un comentario
```

Comentarios

Si alguna vez quiere usar el lenguaje natural dentro del programa, puede escribir un comentario.

Los comentarios son notas que se dejan para recordar lo hace el programa, además el microcontrolador ignora estos comentarios. Para añadir un comentario escribir antes dos líneas inclinadas // y a continuación escribir lo que se desee anotar. El microcontrolador ignorará cualquier texto escrito después de estas dos líneas.

La instrucción if

En la siguiente página en la parte superior derecha se utiliza la palabra "if" para verificar el estado de algo (en este caso el estado del pulsador, en qué posición encuentra). Un estamento **if()** en programación compara dos cosas, y determina si la comparación es verdadera o falsa. Dependiendo de este resultado se lleva a cabo una acción u otra. Cuando en programación se comparan dos cosas hay que usar dos signos de igual ==. Si solo se usa un signo de igual, simplemente se le asigna un valor a una variable en lugar de compararla con algo.

Construyendo su nave espacial

digitalWrite() es la instrucción que permite poner +5V o 0V en un pin de salida. **digitalWrite()** tiene dos argumentos: el número del pin a controlar y que valor se coloca en ese pin, HIGH o LOW. Si quiere apagar los LEDs rojos y encender el LED verde dentro del estamento **if()** el código debería ser como el que se muestra en la siguiente página en la parte superior donde aparecen las tres instrucciones **digitalWrite()**. Con las dos hojas juntas puede verse esta parte del código justo a la derecha de este párrafo.

Si se ejecuta el programa ahora, las luces cambiarán cuando se presione el pulsador. Eso esta bien, pero es posible añadir un poco mas de complejidad al programa para mejorar el efecto final.

En las instrucciones anteriores se le indica a Arduino lo que tiene que hacer cuando el pulsador está abierto. También hay que indicarle que hacer cuando el pulsador está cerrado. El estamento i**f()** puede usar opcional mente el componente **else** que permite hacer otra cosa si la condición inicial no se cumple. En este caso, como se acaba de comprobar a través del código si el pulsador esta **LOW**, hay que escribir nuevas instrucciones para la condición **HIGH** dentro del bloque de la instrucción **else**.

Para conseguir que los LEDs rojos parpadeen cuando el pulsador este presionado es necesario encenderlas y a pagarlas dentro del bloque de la instrucción **else** que se acaba de escribir. Para hacer esto, cambiar el código tal y como se puede ver en el recuadro de la derecha (parte inferior de la hoja siguiente).

Ahora el programa hará que los LEDs rojos parpadeen cuando el pulsador está presionado.

Después de colocar los LEDs a un cierto estado, es necesario que Arduino realice una pausa antes de cambiarlos de nuevo a su estado anterior. Si esta pausa no se realiza los diodos parpadearan demasiado rápido y parecerán que alumbran menos, no se vera el parpadeo. Esto sucede porque Arduino ejecuta esta parte del programa **(loop)** miles de veces por segundo, y los LEDs se encienden y se apagan tan rápido que no podemos percibirlo, dando la sensación de que no se apagan, solo que disminuyen su luminosidad. La instrucción **delay()** permite que Arduino deje de ejecutar cualquier cosa que este haciendo durante un periodo de tiempo. Dentro del argumento de la instrucción **delay()** se establece el número de mili segundos que Arduino estará parado antes de ejecutar la siguiente parte del código. Hay 1000 mili segundos en un segundo. **delay(250)** producirá una pausa de un cuarto de segundo.


```
11   if (switchState == LOW) {
12   // el pulsador no está presionado
13     digitalWrite(3, HIGH); // LED verde
14     digitalWrite(4, LOW); // LED rojo
15     digitalWrite(5, LOW); // LED rojo
16   }
```

```
17   else { // el pulsador está presionado
18       digitalWrite(3, LOW);
19       digitalWrite(4, LOW);
20       digitalWrite(5, HIGH);
21       delay(250); // en pausa un cuarto de segundo
22       // cambiar el estado de los LEDs
23       digitalWrite(4, HIGH);
24       digitalWrite(5, LOW);
25       delay(250); // en pausa un cuarto de segundo
26   }
27 } // Volver al comienzo de la instrucción loop
```

Puede ser útil escribir el flujo del programa en pseudo código: es una forma que describe lo que tiene que hacer el programa en lenguaje plano, pero de una forma que hace mas fácil escribir el código final a partir de este lenguaje plano. En este caso se va a comprobar si **switchState** está **HIGH** (si el pulsador está presionado) o no lo está. Si el pulsador está presionado se apagará el LED verde y los rojos se encenderán. En el recuadro de la izquierda pueden verse las instrucciones de este pseudo código.

Si el switchState esta LOW:
  encender el LED verde
  apagar los LEDs rojos

Si el switchState esta HIGH:
  apagar el LED verde
  encender los LEDs rojos

COMO SE UTILIZA

Una vez que Arduino esta programado, el LED verde debe de verse encendido. Cuando presione el pulsador, los LEDs rojos comenzarán a parpadear, y el LED verde se apagará. Intenta cambiar el tiempo de las dos instrucciones **delay()**; observa que le pasa a los LEDs y como la respuesta del sistema ha cambiado al variar la velocidad del parpadeo. Cuando en el programa se llama a la instrucción **delay()** hace que todo deje de funcionar durante un tiempo. No se puede leer la posición del pulsador hasta que el tiempo establecido dentro de la instrucción **delay()** haya finalizado. Los retrasos dentro de un programa son a menudo útiles, pero hay que fijarse al realizar un proyecto que su uso no es innecesario y que no interfiere el funcionamiento de dicho proyecto.

¿Cómo podría conseguir que los diodos LEDs comiencen a parpadear cuando el programa comienza? ¿Cómo puedo hacer una interface para mis aventuras intelesterales mas grande, o mas compleja con LEDs y pulsadores

Cuando se inicia la creación de una interface para un proyecto, pensar acerca de las expectativas de las personas mientras lo están usándolo. Cuando presionen el pulsador, ¿querrán que ocurra algo de inmediato? ¿Debe de existir un tiempo de retraso entre la pulsación y lo que Arduino hace?. Intenta ponerte en el lugar de las personas que lo van a usar en el momento de realizar el diseño, y ver si cumple con las expectativas de lo que se espera del proyecto.

En este proyecto, ha creado su primer programa para Arduino para controlar el comportamiento de algunos LEDs a través de un pulsador. Ha usado variables, una instrucción if()...else, y funciones para leer el estado de una entrada y controlar salidas.

CÓMO LEER EL CÓDIGO DE COLORES DE LAS RESISTENCIAS

Los valores de las resistencias se indican mediante bandas de colores, según un código que se desarrollo en 1920, cuando erá muy difícil escribir números en objetos tan pequeños.

Cada color se corresponde con un número, como se puede ver en la tabla inferior. Cada resistencia tiene entre 4 o 5 bandas. En las resistencias con 4 bandas, las dos primeras bandas indican los dos primeros dígitos del valor de la resistencia, mientras que la tercera banda de color indica el número de ceros que sigue a los dos primeros valores (técnicamente esta tercera banda representa potencias de diez). La última banda especifica la tolerancia: en el ejemplo inferior, se puede leer un valor de resistencia de 10K, y este valor puede variar en más o menos un 5% según esta tolerancia