Capítulo 01

Introducción

Arduino cuenta con una serie de entradas y salidas digitales y analógicas programables, que son la base de esta placa de desarrollo. Es fundamental conocerlas para comenzar a programarla.

Entradas analógicas y digitales / 9
Pines digitales / 10
Pines analógicos / 22
Otros pines / 29
Actividades / 30
- **Test de autoevaluación**
- **Ejercicios prácticos**

Entradas analógicas y digitales

Arduino es una plataforma de hardware gratuita, creada por **David Cuartielles** y **Massimo Banzi**, que se basa en una placa de circuito con microcontrolador y un entorno de desarrollo. Su principal objetivo reside en promover el uso de la electrónica en proyectos multidisciplinarios para entusiastas y expertos.

Si te alejas un poco de esta definición, extraerás algunas ideas muy interesantes. Por un lado, Arduino es también un **sistema de procesamiento**, un microcontrolador y una placa; además, integra un entorno de desarrollo y es una plataforma de hardware de código abierto. Por otro lado, de forma simplificada, se puede mencionar que es una plataforma de hardware de código abierto, cuyo funcionamiento se basa en una placa con entradas y salidas (analógicas y digitales), y el entorno de desarrollo incorpora todo lo necesario para crear tus propios programas

En **Arduino UNO**, hay entradas y salidas analógicas (6) y digitales (16). Por una parte, las magnitudes analógicas se suelen utilizar para dispositivos de entrada y permiten leer una serie de valores. Por otra parte, la entrada o salida de un pin digital puede ser **0V** o **5V**, lo que indica que el pin está en un nivel bajo (LOW) o en un nivel alto (HIGH), respectivamente.

En todas las placas, los pines son multifunción, es decir, tienen una función u otra según la configuración.

En cuanto al hardware, Arduino integra un microcontrolador que permite programar utilizando lenguajes de alto nivel. Es el elemento que realiza procesos matemáticos y lógicos, y gestiona los recursos de cada componente externo que se conecta a la placa base.

Esta placa de desarrollo contiene una serie de entradas analógicas y digitales, por lo que es posible conectar diferentes sensores y otras placas de desarrollo o blindajes. Todo esto permite agregar nuevas funciones sin

tener que cambiar el diseño original de la placa de circuito.
Un elemento importante en el hardware Arduino es su puerto de entrada/salida, que se puede conectar a una computadora para integrar el trabajo con el software.

Pines digitales

Estos son los pines que el usuario puede activar (colocar voltaje) o desactivar (quitar voltaje). Es similar a escribir 0 y 1, porque ya se ha dicho 0V y 5V, respectivamente.
El pin 0 (entrada serie RX) y el pin 1 (salida serie TX) –puertos serie– se utilizan para la comunicación entre dispositivos. La característica principal de los puertos serie es que envían información bit a bit, un bit a la vez. La versión avanzada de este puerto será un puerto paralelo, lo que te permitirá enviar información en paralelo.
En el caso específico de Arduino UNO, el puerto 0 (RX) será el puerto serie de entrada, y el puerto 1 (TX) será el puerto serie de salida.
Los pines 2 y 3 permiten interrumpir la **operación de bucle**. Las características de este bucle se desarrollan más adelante en otro apartado, pero ten en cuenta que se ejecuta continuamente en el código Arduino. Imagina que quieres leer un sensor cada segundo. No es necesario escribir código varias veces, ya que, en Arduino, existe un bloque de código que puede repetir todo lo que contiene en forma indefinida. Estos pines detienen con precisión el bucle y fuerzan la ejecución del código asignado a cada pin.
Los pines 3, 5, 6, 9, 10 y 11, marcados con el símbolo ~, son entradas y salidas especiales porque, aunque son digitales y se pueden usar como están, también es posible usarlos para ancho de pulso **PWM** (modulación de ancho de pulso). Esta tecnología permite transmitir señales o energía a dispositivos con señales cuadradas. Como ya se ha dicho, los pines digitales solo sirven a dos estados, bajo (0V) y alto (5V). Si cambias los estados de alto y bajo nivel controlando el tiempo de alto nivel y el tiempo de bajo nivel, obtendrás una onda cuadrada, en la que el nivel alto (5V) será el ancho del pulso. La relación entre estos dos tiempos se denomina **ciclo de trabajo**, se expresa como porcentaje (%) e indica el momento en que la señal es alta y el momento en que la señal es baja.
En Arduino, hay una función que permite usar estos puertos, llamada `AnalogWrite`. No puedes usar ningún rango de valores, esta función solo admite valores entre `0` y `255` (8 bits). Por lo tanto, si se toma el ejemplo anterior del ciclo de trabajo, es posible definir una tabla en la que dirá el valor correspondiente para introducir en la función `AnalogWrite` y la salida en voltios.

CICLO TRABAJO (%)	ANALOGWRITE (0 – 255)	SALIDA (VOLTIOS)
0	0	0
25	64	1.25
50	127	2.5
75	191	3.75
100	255	5

Los pines 10, 11, 12 y 13 se utilizan para conectar varios dispositivos juntos, así como varios Arduinos. Son **pines SPI** (*Serial Peripheral Device Interface*), porque utilizan con precisión este estándar de comunicación definido para relacionarse con el circuito integrado a través del bus de comunicación y la comunicación en serie. El pin 10 es **SS** (selección de esclavo), que es el puerto esclavo (**esclavo**). Permite que el esclavo sea seleccionado del maestro (**maestro**), o permite que el maestro active al esclavo (si es un esclavo). El pin 11 es **MOSI** (salida maestra y entrada esclava). Se utiliza como entrada de datos de la estación maestra y salida de datos de la estación esclava. El pin 12 es **MISO** (*Master Input Slave Output*) y, a diferencia del pin 11, permite al host ingresar datos y al esclavo

enviar datos. Finalmente, el pin 13 **SCK** (seleccionar reloj) se usa como señal de reloj. Este es un pulso, como se ve en el pin PWM, que marca la sincronización entre los dispositivos conectados.
Una **señal digital** es un tipo de señal generada por alguna clase de fenómeno electromagnético en el que cada signo que codifica el contenido de ella puede ser analizado en término de algunas magnitudes que representan valores discretos, en lugar de valores dentro de un cierto rango. Por ejemplo, el interruptor de la luz solo puede tomar dos valores o estados: abierto o cerrado; o la misma lámpara: encendida o apagada.
En Arduino y en las placas compatibles, para tratar las entradas y salidas digitales deberás usar las siguientes funciones:

- **`pinMode()`**: configura en el pin especificado si se va a comportar como una entrada o una salida.
- **`digitalWrite()`**: escribe un valor HIGH o LOW en el pin digital especificado. Si el pin está configurado como OUTPUT, pone el voltaje correspondiente en el pin seleccionado. Si el pin está configurado como INPUT, habilita o deshabilita la resistencia interna de pull up del correspondiente pin.
- **`digitalRead()`**: lee el valor del pin correspondiente como HIGH o LOW.

Terminales	Números	Descripción
Serial	0 RX y 1 TX	Se utilizan para recibir (RX) y transmitir (TX) datos serie TTL.
Interruptores externos	2 y 3	Se pueden utilizar estos terminales para disparar una interrupción con un valor bajo, un pulso de subida o bajada, y también un cambio de valor.
PWM	3, 5, 6, 9, 10 y 11	Entrega salidas PWM de 8 bits; para esto se utiliza la función analogWrite(). Con el microchip Atmega8, estas salidas estarán en los pines 9, 10 y 11.
Reset BT	7	En Arduino BT, se encuentra conectado a la línea de reset para el módulo Bluetooth.
SPI	10, 11, 12, 13	Se trata de terminales que soportan comunicación SPI.
LED	13	En algunas placas, un LED se encuentra conectado al pin 13.

Antes de seguir puedes conocer las bases de la programación para Arduino en nuestros e-books Arduino IDE y Programa tu Arduino.
Tanto en la red como en la documentación oficial de esta placa de desarrollo, se encuentran muchos ejemplos. En el siguiente el led conectado al pin 13 parpadeará cada dos segundos:

```
int ledPin = 13;
   void setup(){
   pinMode(ledPin, OUTPUT);
}
void loop(){
   digitalWrite(ledPin, HIGH);
   delay(1000);
   digitalWrite(ledPin, LOW);
   delay(2000);
}
```

En el código que verás a continuación, el LED conectado al pin 13 parpadeará en un intervalo de tiempo variable que depende del número de veces que se ejecuta el programa, para ello se utiliza la función **loop**:

```
int ledPin = 13;
int n = 0;
void setup(){
   pinMode(ledPin, OUTPUT);
}
void loop(){
   digitalWrite(ledPin, HIGH);
   delay(1000);
   digitalWrite(ledPin, LOW);
   n++;
   delay(delayVal(n));
}

int delayVal(int f){
   return f*100;
}
```

A continuación, se lee el valor de un interruptor conectado en el pin 2; cuando está cerrado en el pin de entrada, habrá un estado alto (HIGH) y, por lo tanto, verás que se enciende el led:

```
int ledPin = 13;
int inPin = 2;
void setup() {
   pinMode(ledPin, OUTPUT);
   pinMode(inPin, INPUT);
}
void loop() {
   if (digitalRead(inPin) == HIGH){
      digitalWrite(ledPin, HIGH);
      delay(1000);
      digitalWrite(ledPin, LOW);
      delay(1000);
   }
}
```

En el siguiente código de ejemplo, se va aumentando y disminuyendo el brillo del pin 9 mediante PWM:

```
int ledPin = 9;
void setup(){}
void loop() {
   for (int i=0; i<=255; i++){
      analogWrite(ledPin, i);
      delay(100);
   }
   for (int i=255; i>=0; i--) {
      analogWrite(ledPin, i);
      delay(100);
   }
}
```

Ahora se integra el uso de un potenciómetro para controlar la frecuencia del parpadeo de un led:

```
int potPin = 0;
int ledPin = 13;
void setup() {
   pinMode(ledPin, OUTPUT);
}
void loop() {
   digitalWrite(ledPin, HIGH);
   delay(analogRead(potPin));
   digitalWrite(ledPin, LOW);
   delay(analogRead(potPin));
}
```

Recuerda que una señal digital solo puede cambiar entre dos valores, a los que llamarás **-Vcc** y **+Vcc**. Una salida digital es un dispositivo que permite cambiar su voltaje a uno de estos dos valores mediante programación, lo que hace que puedas manipular el entorno.
Por lo general, en Arduino, los voltajes -Vcc y +Vcc corresponden a 0V (GND) y 5V, respectivamente. Sin embargo, algunos modelos de Arduino funcionan a 3.3V, como algunas placas basadas en procesadores Mini, Nano y ARM, como Arduino Due.
Todos los pines digitales de Arduino se pueden usar como salidas digitales (de ahí el nombre E/S, entrada y salida), pero debe tenerse en cuenta que los pines analógicos también se pueden utilizar como entradas y salidas digitales.
El número exacto de salidas digitales depende del modelo de placa de circuito que estés usando. Arduino Uno y Nano tienen 22 pines, que se pueden usar como salidas digitales; Arduino Mini tiene 20 pines, y puede haber hasta 70 números en el Mega modelo Salida (Figura 1.1.).

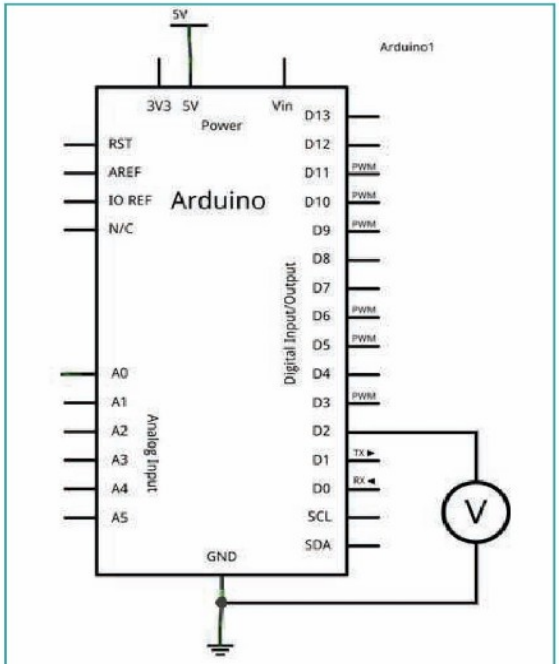

Figura 1.1.

Para probar todo lo mencionado hasta este momento, no necesitas realizar conexiones complejas en tu Arduino. Puedes conectar tu placa de desarrollo a la computadora y cargar el siguiente código, que está basado en el archivo de ejemplo **Blink**, y que se encarga de encender y apagar una salida digital:

```
const int pin = 2;

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

```
  pinMode(pin, OUTPUT);
}

void loop(){
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(1000);
}
```

Ahora bien, el siguiente código recibe un carácter a través del puerto serie para encender o apagar una señal digital desde la PC. A través del puerto serie, puedes enviar un carácter. Si escribes **0**, la salida se apaga, y si escribes **1**, se enciende.

```
const int pin = 2;
int option;

void setup(){
  Serial.begin(9600);
  pinMode(pin, OUTPUT);
}

void loop(){
  //si existe información pendiente
  if (Serial.available()>0){
    //leeemos la opcion
    char option = Serial.read();
    if (option == '0' )
    {
         digitalWrite(pin, LOW);
    }
    else if (option == '1' )
    {
         digitalWrite(pin, HIGH;
    }
    delay(200);
  }
}
```

Otro ejemplo sencillo permitirá que un led se encienda durante dos segundos para luego modificar su intensidad entre los valores máximo y mínimo en forma gradual.
Deberás insertar el led en la protoboard utilizando una resistencia de unos 220-470 Ohmios. Con valores menores, se corre el riesgo de dañarlo, y valores mayores atenúan demasiado el brillo (Figura 1.2.).
Ahora observa el código adecuado. Primero debes declarar las variables:

```
int pinsalida = 10;
int velocidad = 100;
int pwm;
```

Si modificas el valor de la variable **velocidad**, lograrás que el led tarde más o menos tiempo en completar los ciclos de cambio.
En **setup** harás que el led se encienda y, luego, se apague.

```
void setup()
```

```
{
    pinMode(pinsalida,OUTPUT);
    digitalWrite(pinsalida,HIGH);
    delay(2000);
    pinMode(pinsalida,LOW);
    delay(1000);
}
```

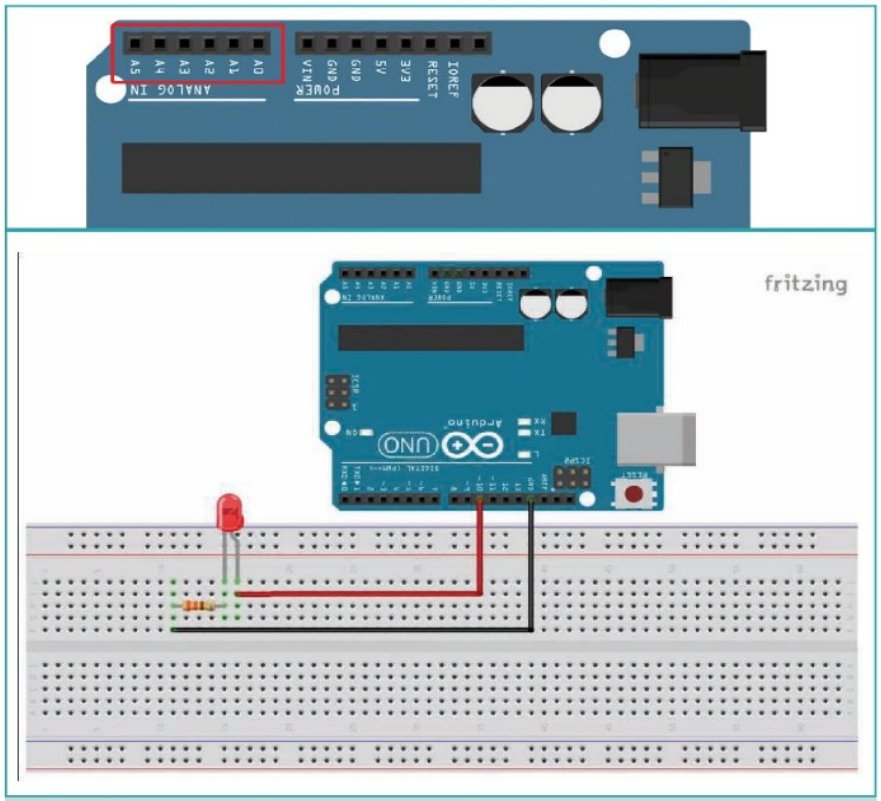

Figura 1.2.

En **loop** harás uso del PWM. Escribe un bucle **for** que dará a la variable **pwm** un valor inicial de **0** y lo irá incrementando en una unidad hasta que alcance el valor máximo de **255**.

```
void loop()
{
    for(pwm=0;pwm<256;pwm++)
    {
        analogWrite(pinsalida,pwm);
        delay(1000/velocidad);
    }
    for(pwm=255;pwm>-1;pwm--)
    {
        analogWrite(pinsalida,pwm);
        delay(1000/velocidad);
    }
}
```

Ahora el código completo:

```
int pinsalida = 10;
int velocidad = 100;
int pwm;

void setup()
{
    pinMode(pinsalida,OUTPUT);
    digitalWrite(pinsalida,HIGH);
    delay(2000);
```

```
    pinMode(pinsalida,LOW);
    delay(1000);
}

void loop()
{
    for(pwm=0;pwm<256;pwm++)
    {
        analogWrite(pinsalida,pwm);
        delay(1000/velocidad);
    }
    for(pwm=255;pwm>-1;pwm--)
    {
        analogWrite(pinsalida,pwm);
        delay(1000/velocidad);
    }
}
```

Pines analógicos

En Arduino UNO, hay 6 entradas analógicas con A al frente. Aunque se pueden utilizar como salidas, el uso más común es leer datos de dispositivos analógicos. Posee una resolución de 10 bits, lo que significa que tienes 1024 valores diferentes, es decir, puedes leer el rango de voltaje de 0V a 5V y detectar un cambio de voltaje de 0.004V (5/1024).
Los pines 4 y 5 admiten la comunicación I2C (circuito integrado), que es muy similar al bus de comunicación SPI y se utiliza para conectar varios dispositivos. En este tipo de comunicación, cada dispositivo tiene una dirección única, y cada uno puede actuar como maestro o esclavo. El pin 4 es SDA (línea de datos en serie), que se utiliza para la transmisión de datos en serie. El pin 5 es SCL (línea de reloj en serie), que proporciona una señal de reloj para mantener sincronizados todos los dispositivos conectados.
Una señal eléctrica analógica es una señal cuyo voltaje o valor de voltaje cambia continuamente y puede tomar cualquier valor. En el caso de la corriente alterna, la señal analógica aumenta su valor con un signo positivo (+) en un medio ciclo y luego disminuye con un signo negativo (-) en el siguiente medio ciclo.
La señal digital obtenida de la señal analógica tiene dos propiedades básicas:

- **Valores**: el valor de voltios define 0 y 1. En tu caso, es tecnología TTL, Arduino Uno es (0-5V), ESP8266 es (0-3.3V).
- **Resolución analógica**: se utilizan símbolos digitales para indicar el número de bits de la señal analógica.

En Arduino, para tratar las entradas y salidas analógicas deberás utilizar las siguientes funciones:

- `analogReference()`: configura la referencia de voltaje usada para la entrada analógica.
- `analogRead()`: lee el valor del pin analógico especificado.
- `analogWrite()`: escribe un valor analógico (onda PWM) al pin especificado. No en todos los pines digitales se puede aplicar PWM.

En Arduino, hay 6 pines para serigrafía de entrada analógica, los números son de A0 a A5. Estos pines se utilizan solo como entradas analógicas. La entrada analógica toma un valor entre 0 y 1023 (Figura 1.3.).

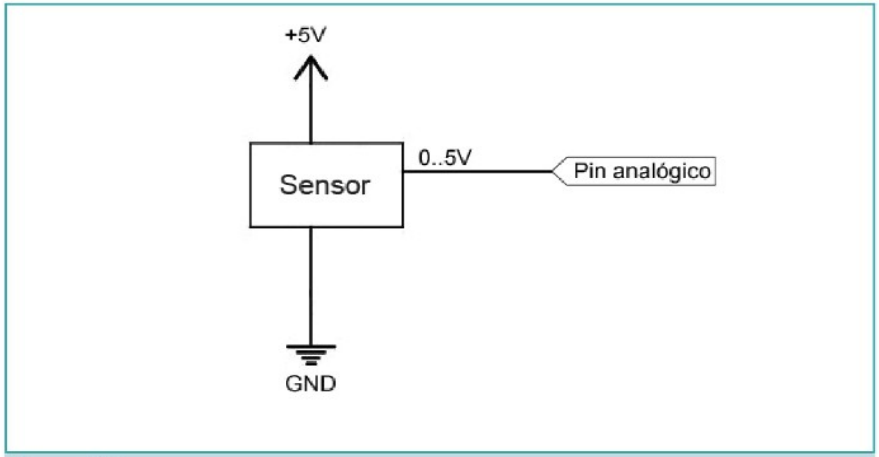

Figura 1.3.

Es importante recordar que no puedes exceder el límite de voltaje permitido, es decir, si se aplica un voltaje mayor a 5 voltios a la entrada digital, la placa Arduino se quemará. Lo mismo ocurre con los voltajes negativos, por lo que debes asegurarte de que el voltaje aplicado a la entrada digital esté entre 0 y 5 voltios.
A continuación verás algunos ejemplos de código para hacer uso de las entradas analógicas en Arduino. Si dispones de un sensor analógico que proporciona una señal analógica entre 0V a 5V, el esquema de conexión es similar al utilizado para realizar una lectura digital.

Figura 1.4. Potenciómetro.

En el siguiente código se realiza la lectura mediante **AnalogRead()** y almacena el valor entregado:

```
const int sensorPin = A0;
int sensorValue;

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

void loop()
{
    sensorValue = analogRead(sensorPin);


    if (sensorValue > 512)
    {
        Serial.println("Mayor que 2,5V");
    }
    else
    {
        Serial.println("Menor que 2,5V");
    }
    delay(1000);
}
```

Es importante considerar que el valor devuelto por **AnalogRead()** se codifica como un número entero de 0 a 1023. Para convertirlo en un valor de tensión puedes usar lo siguiente: