UNIVERSIDAD NACIONAL DE JAÉN

COMISIÓN ORGANIZADORA

FACULTAD DE INGENIERÍA

PROGRAMACIÓN DE INGENIERÍA

Manual de Programación Arduino enfoque ingenieril

Autor:

Mg. Fuentes Maza Frans

Jaén, octubre 2024.

PRESENTACIÓN

Este manual está diseñado como una guía introductoria a la programación con Arduino, una plataforma de código abierto que permite desarrollar proyectos electrónicos de manera sencilla y accesible. Con este manual, aprenderás a programar tu placa Arduino para interactuar con diferentes sensores, actuadores y otros dispositivos, proporcionando una base sólida para crear proyectos de automatización, robótica, y mucho más.

Arduino combina hardware y software, y su entorno de desarrollo (IDE) facilita escribir, compilar y cargar código en las placas. Gracias a su naturaleza amigable, es ideal para principiantes en la electrónica y programación, así como para desarrolladores más avanzados que buscan realizar proyectos interactivos.

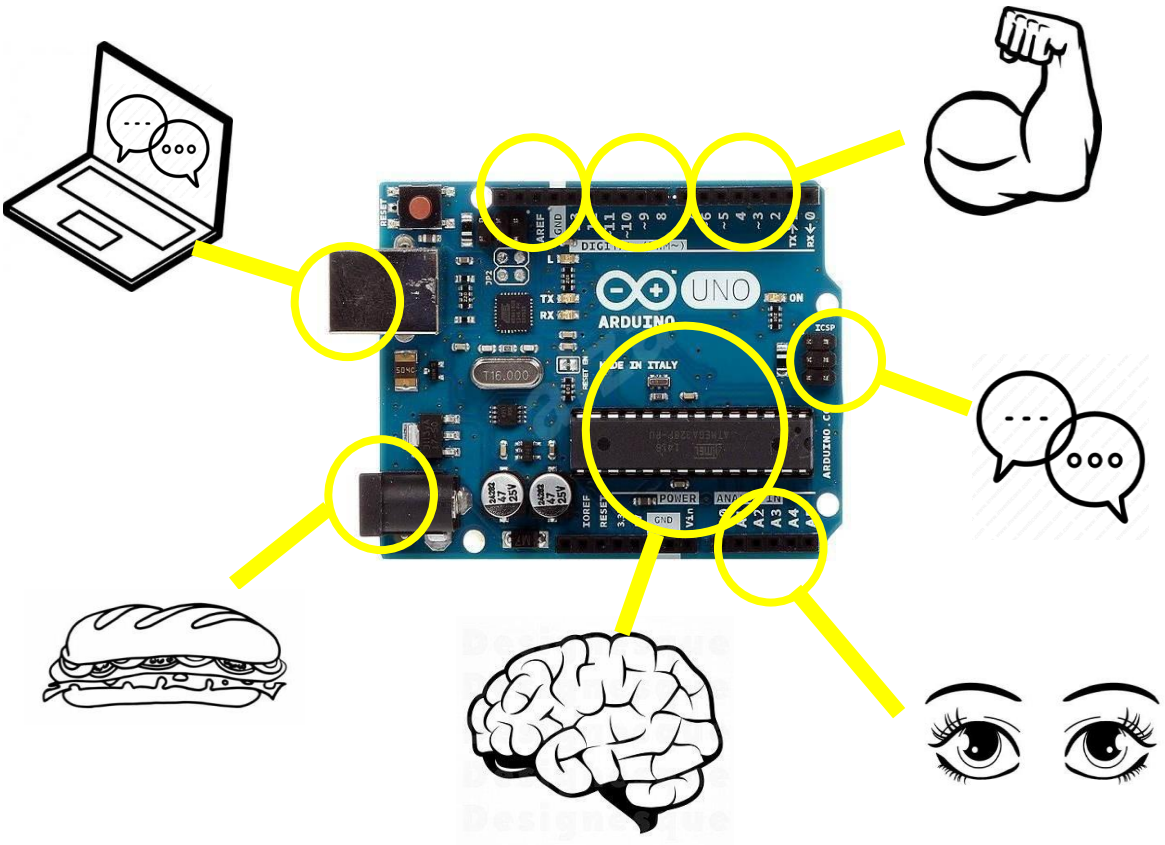
Atte. El Autor

INTRODUCCIÓN

Arduino es una plataforma de hardware y software de código abierto que facilita la creación de proyectos electrónicos interactivos. Se basa en una placa de microcontrolador y en una plataforma de desarrollo de código que permite cargar programas en el microcontrolador.

Las placas de Arduino están diseñadas para ser fáciles de usar, incluso para principiantes en electrónica y programación. Arduino permite controlar sensores, motores, luces, pantallas, entre otros dispositivos, para crear una amplia variedad de proyectos, desde simples luces parpadeantes hasta robots complejos.

Figura 1 Placa Arduino



INDICE

I. ARDUINO

Arduino permite el prototipado electrónico de código de acceso abierto que combina el hardware y software, permitiendo a usuarios de diferentes niveles crear proyectos interactivos. Su simplicidad y versatilidad lo han convertido en una herramienta popular en educación e industria. Fue creado en 2005 por un grupo de ingenieros en Italia, Arduino ha evolucionado a lo largo de los años. Desde su lanzamiento, ha dado lugar a una amplia gama de modelos de placas y una comunidad global activa, contribuyendo a la popularización del desarrollo de sistemas embebidos.

Figura 2 Placa Arduino

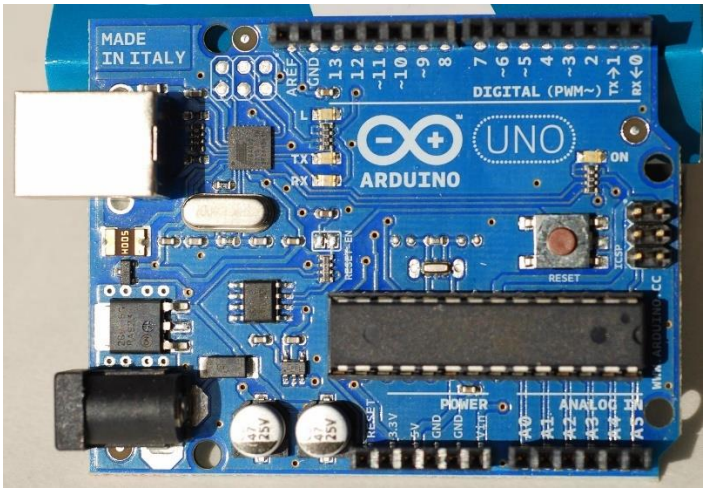


Figura 3 Símbolo Lenguaje Programación Arduino

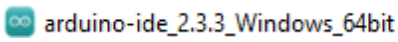
1. Instalación de ARDUINO

a) Desde el sitio web e Arduino: https://www.arduino.cc/en/software , se descarga la versión a instalar

Figura 4 Sitio web de descarga

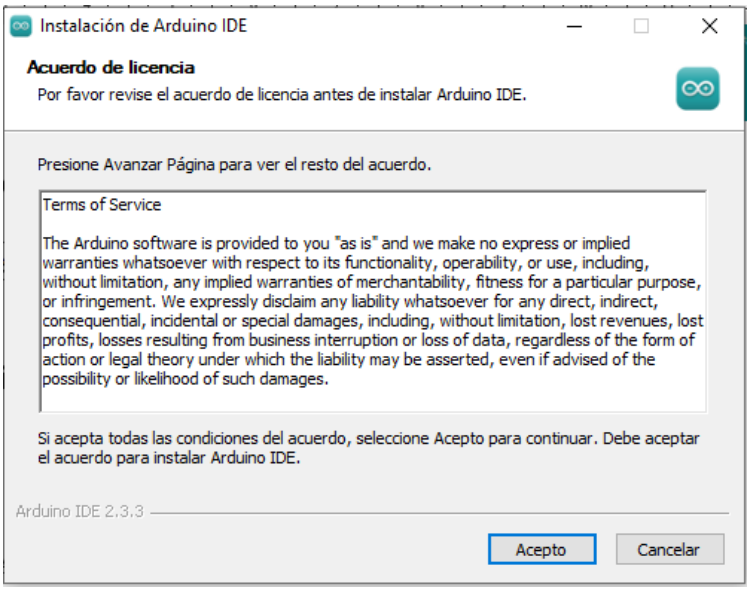
b) Doble clic sobre el Archivo descargado

Figura 5 archivo de instalación



c) Dar clic en siguiente hasta finalizar la instalación

Figura 6 proceso de instalación



d) En el escritorio debe estar el acceso directo de Arduino

e) Doble clic para abrir

2. Página de inicio ARDUINO

Figura 7 pantalla de inicio

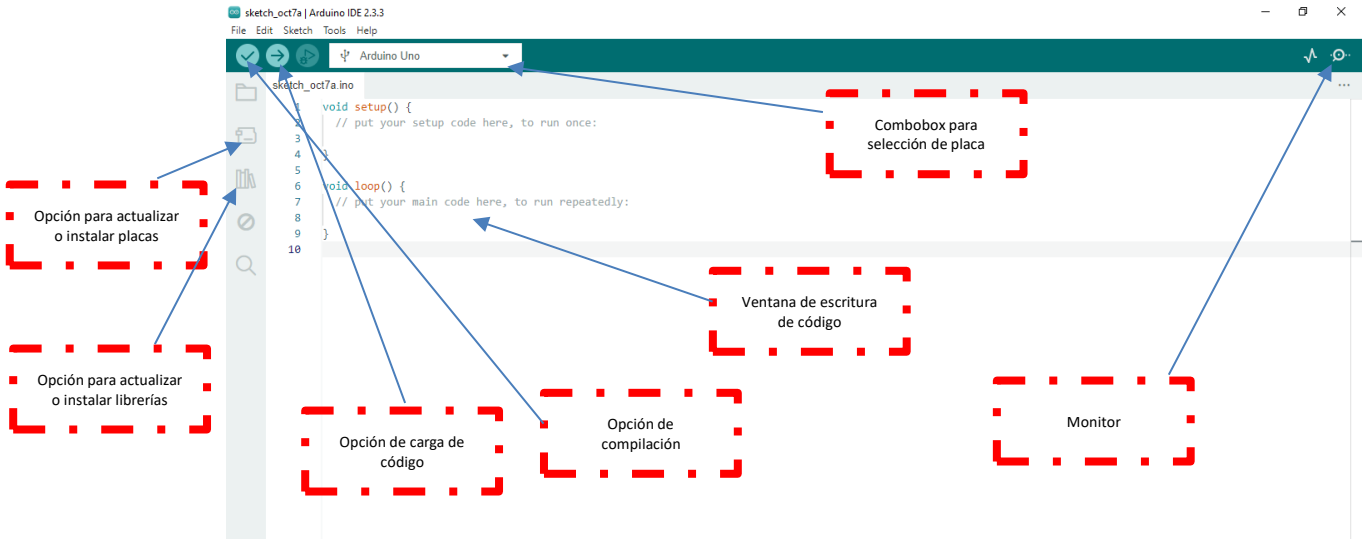
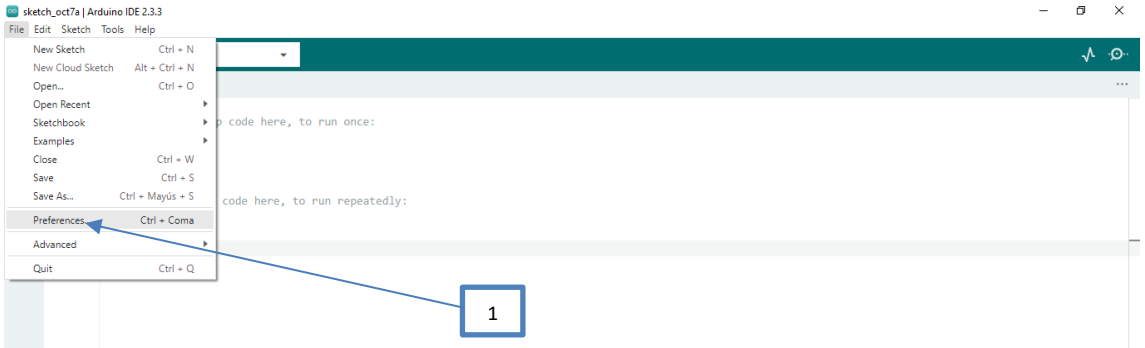
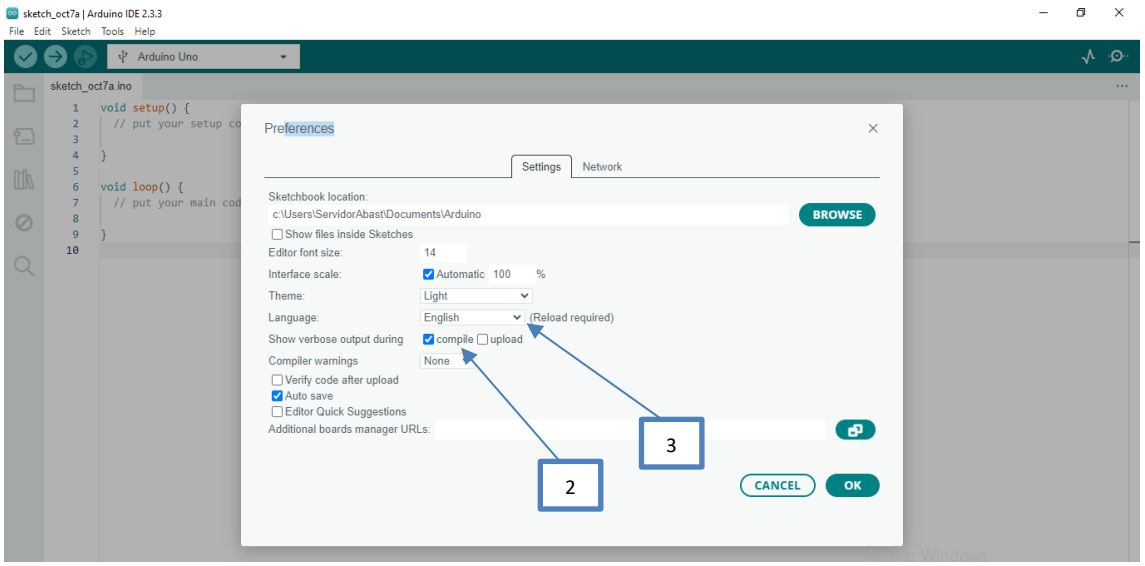


Figura 8 configuración compilación y carga de código

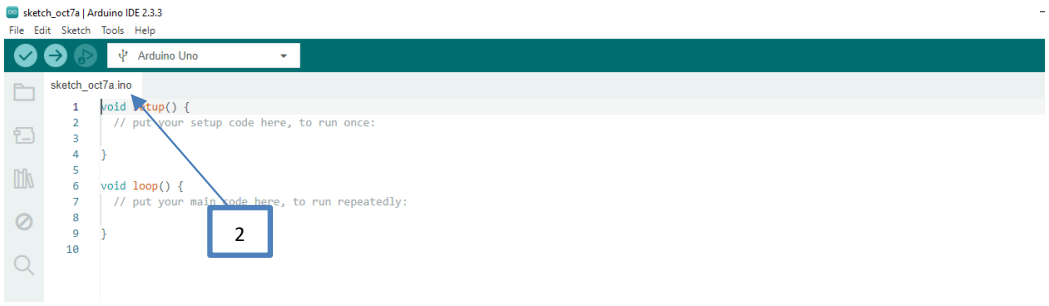




3. Archivos *.INO

Los archivos de Arduino son de extensión *.ino

Figura 9 extensión de archivos



4. Tipos de Arduino

Figura 10 P. Arduino Mega

Figura 11 P. Arduino UNO

Figura 12 P. Arduino NANO

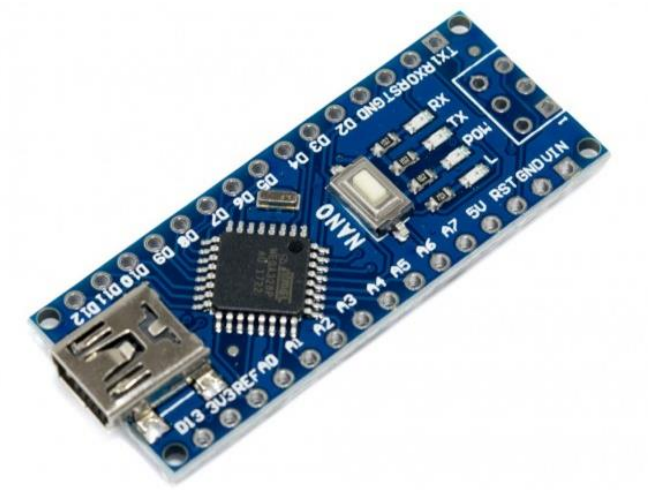


Figura 13 P. Arduino NANO

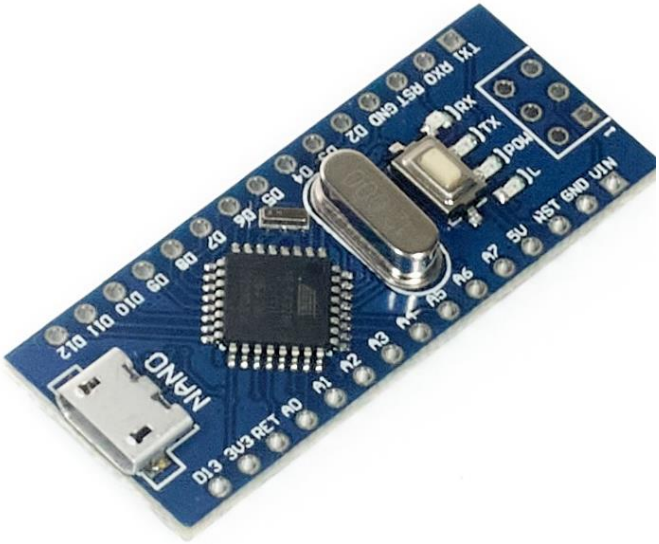


Figura 14 P. Arduino YPAD

5. Fundamentos de programación

4.1. Distribución de pines de Arduino

Figura 15 Distribución de pines

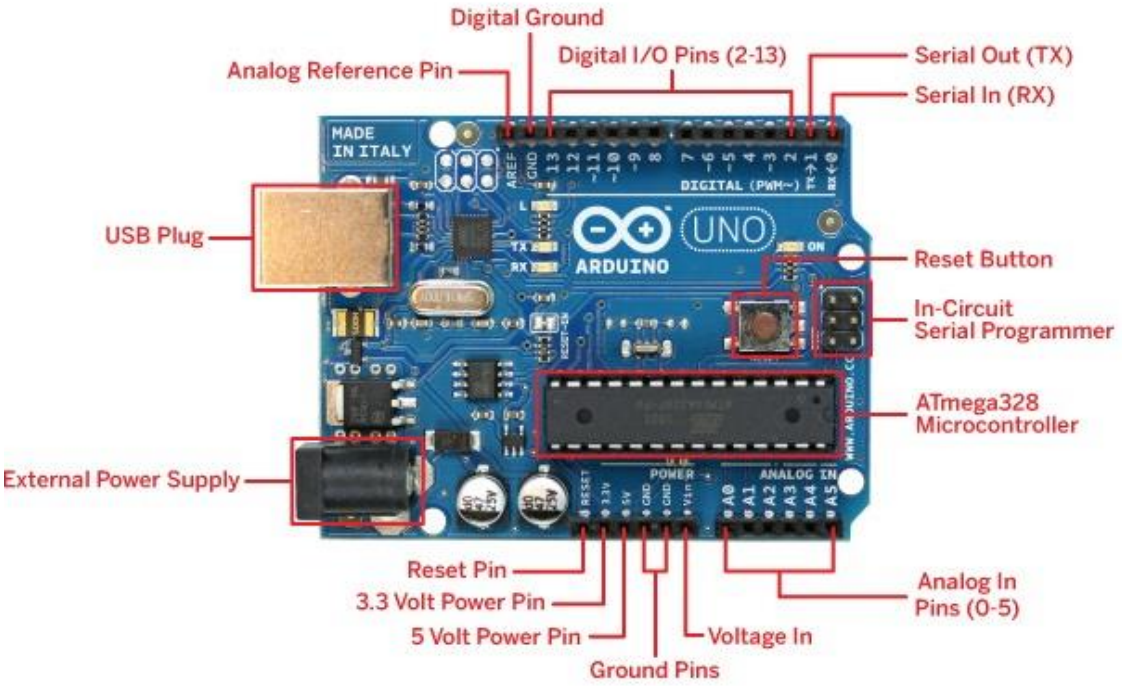
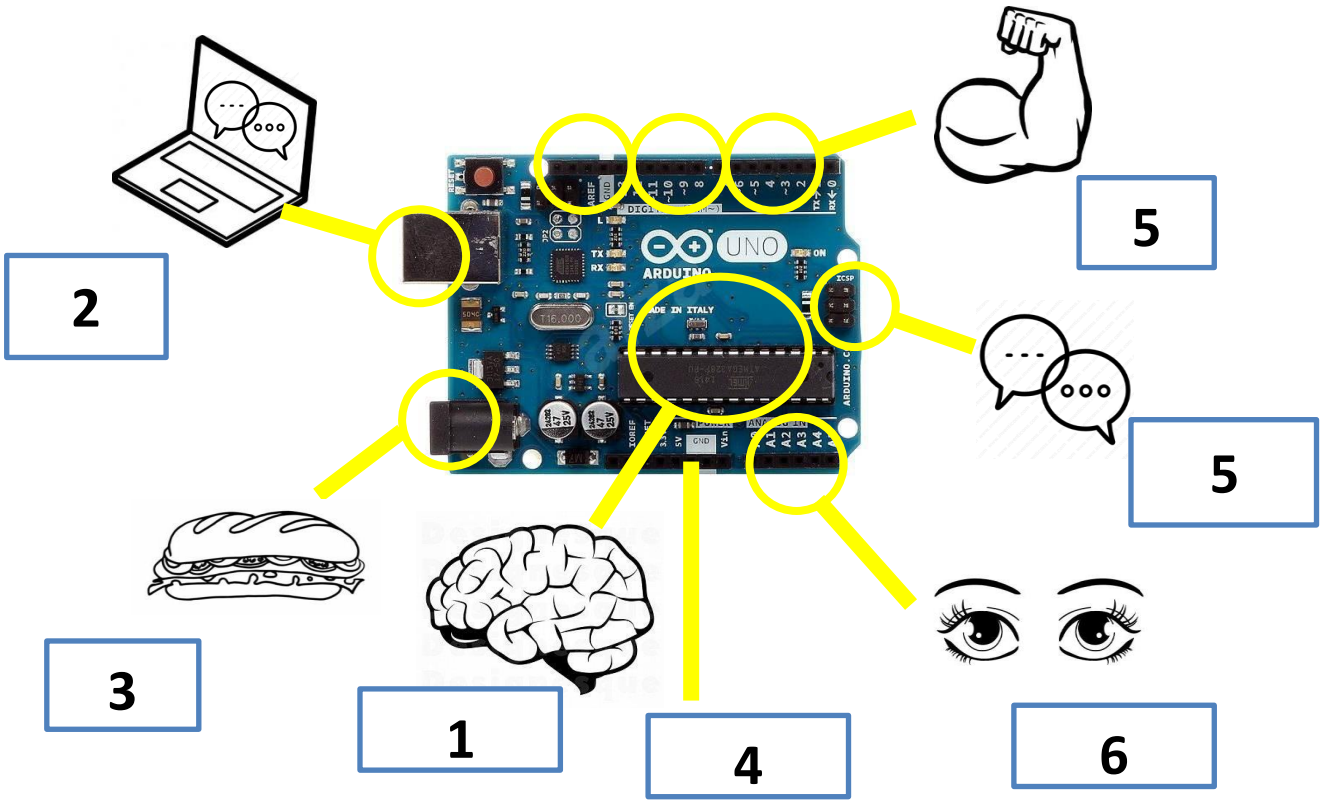


Figura 15 Distribución de pines y características



1. **Procesador Atmel Memoria**

2. **Conexión USB – puerto serie Alimentación 5v baja potencia Fusible 500 mA**

   - Podemos brindar voltaje al Arduino por medio de un conector USB cuando se usan cargas menores y no se necesitan voltajes superiores a 5 voltios.
   - El conector de alimentación es la opción más segura para suministrar energía al Arduino, además del USB.

3. **Alimentación de potencia externa 7 a 12 V**

   - Abastece a 12 Volts: I = 2 / (12-5) = 2 / 7 = 285mA
   - Abastece a 9 Volts: I = 2 / (9-5) = 2/4 = 500mA
   - Abastece a 7 Volts: I = 2 / (7-5) = 2/2 = 1A

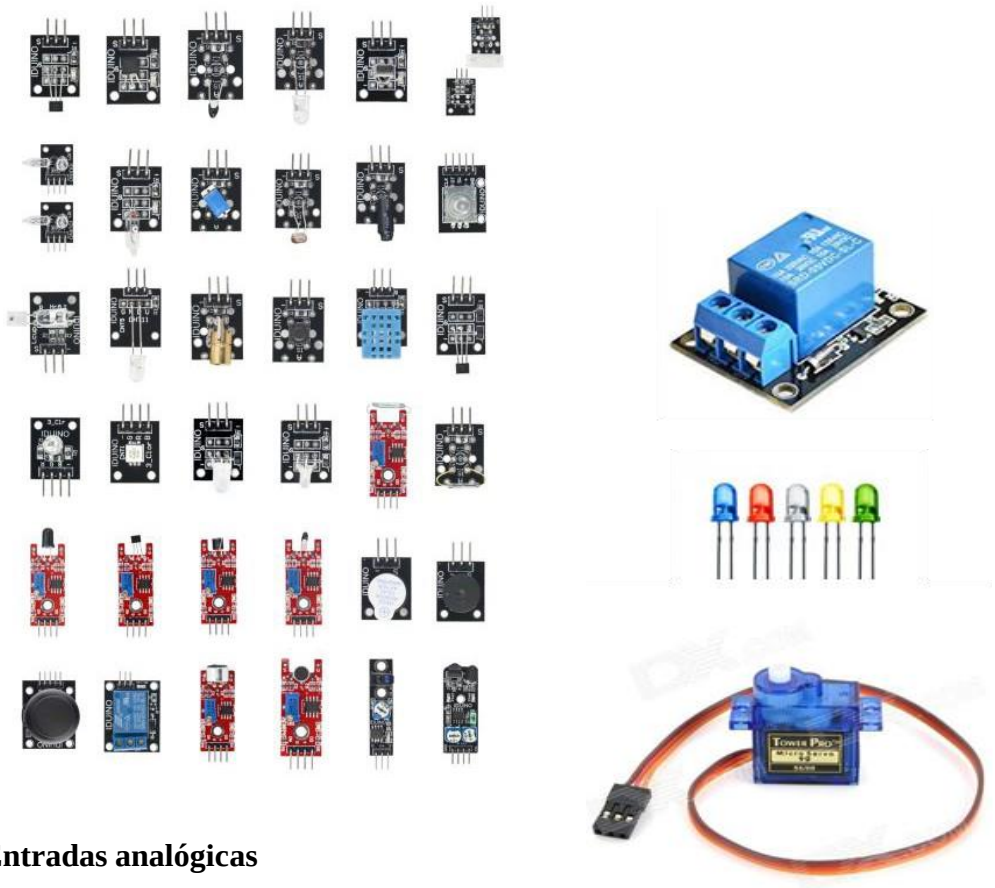
4. **Alimentación pins Vin y 5V + GND**

   - Permite conectar una fuente de alimentación externa que oscile entre 6 y 12 voltios directamente a la entrada del regulador de la placa Arduino.
   - Si se aplica voltaje directamente al pin VIN, no se debe conectar un voltaje al mismo tiempo en el jack de alimentación.

5. Entradas / Salidas digitales

- Configurables como entrada o como salida
- Las salidas PWM pueden regular la potencia de lo que tenemos conectado de acuerdo a la frecuencia de pulsos eléctricos de salida.
- Tienen ruido eléctrico (necesitan estar puestas a GND o a +5V).
- Pins 0 y 1 usados para la comunicación USB (Serie)
- En 3.3V la intensidad máxima es 50 mA. En 5V todo sumado no sobrepasa 200 mA. Una salida individual no sobrepasa 100 mA.
- El Arduino en reposo consumo 65 mA.

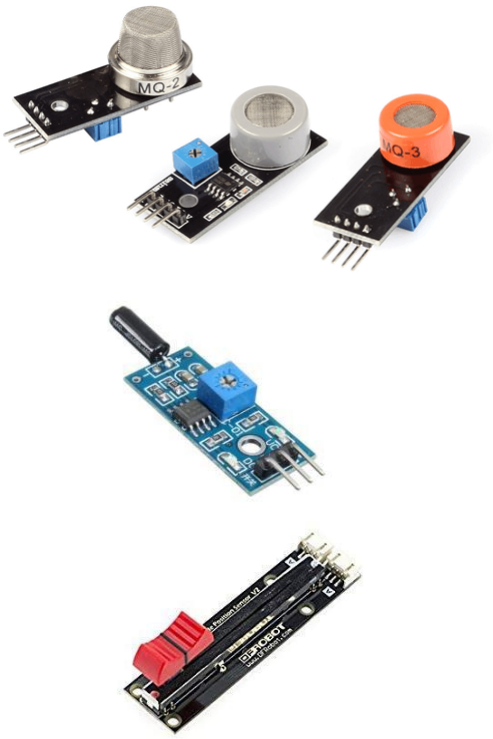
Figura 16 Sensores digitales entrada/ Salidas Digitales



6. Entradas analógicas

- Asume valores entre el rango de 0 y 1. En nuestro caso necesita voltajes de 0 a 5V.
- Conversor analógico digital en la placa.
- En Arduino Uno tiene rango de valores se representa entre 0 y 1023

Figura 16 Dispositivos Analógicos



4.2. Estructura de un programa

Está compuesta por dos partes:

```
void setup()
{
Instrucciones;
}
void loop()
{
Instrucciones;
}
```

En este contexto, setup() se encarga de realizar la configuración inicial, mientras que loop() contiene el código que se ejecutará de manera cíclica (de ahí proviene el término "loop" o "bucle"). Ambas funciones son esenciales para que el programa funcione correctamente (Ruiz Gutiérrez, 2007).

void setup()

```
{
pinMode(pin, OUTPUT); // configura el 'pin' como salida
}
```

void loop()

```
{
digitalWrite(pin, HIGH); // pone en uno (on, 5v) el ´pin´
delay(1000); // espera un segundo (1000 ms)
digitalWrite(pin, LOW); // pone en cero (off, 0v.) el ´pin´
delay(1000);
}
```

4.3. Funciones

Una función es un bloque de código que posee un nombre y un conjunto de instrucciones que se ejecutan al invocarla. Las funciones setup() y loop() son ejemplos de esto. Además, se pueden crear funciones personalizadas para realizar tareas repetitivas, lo que ayuda a simplificar el código y reducir el tamaño del programa (Ruiz Gutiérrez, 2007).

```
type nombreFunción(parámetros)
{
Instrucciones;
}
```

```
int delayVal()
{
int t; // crea una variable temporal 't'
t= analogRead(pot); // lee el valor del potenciómetro
t/= 4; // convierte 0-1023 a 0-255
return t; // devuelve el valor final
}
```

OJO:

- Las llaves se utilizan para marcar el inicio y el final de un bloque de instrucciones.
- El punto y coma “;” se emplea para delimitar las instrucciones en el lenguaje de programación de Arduino.

4.4. Variables

Una variable es la representación abstracta de un valor, y que se puede representar por una letra, palabra o combinación de letras y números.

```
int variableA = 0;
variableA = analogRead(2);// la variableA recoge el valor analógico del PIN2
```

OJO:

Todas las variables deben declararse antes de ser utilizadas. Para declarar una variable, primero se debe especificar su tipo, como int (entero), long (largo), float (coma flotante), entre otros.

4.5. Operadores y Asignaciones

Operadores

x == y // x es igual a y

x != y // x no es igual a y

x < y // x es menor que y

x > y // x es mayor que y

Asignaciones

x ++ // igual que x = x + 1, o incrementar x en + 1

x -- // igual que x = x - 1, o decrementar x en -1

x += y // igual que x = x + y, o incrementa x en +y

x -= y // igual que x = x - y, o decrementar x en -y

x *= y // igual que x = x * y, o multiplicar x por y

x /= y // igual que x = x / y, o dividir x por y

Operadores Lógicos

Logical AND:

if (x > 0 && x < 5) // cierto sólo si las dos expresiones son ciertas

Logical OR:

if (x > 0 || y > 0) // cierto si una cualquiera de las expresiones

es cierta

Logical NOT:

if (!x > 0) // cierto solo si la expresión es falsa

4.6. Subfunciones principales

Digitales:

pinMode(): se utiliza para definir si el pin será usado como entrada (INPUT) o salida (OUTPUT). Esta función debe estar dentro de la función principal voidsetup().

digitalWrite(): se utiliza para determinar un valor lógico por un pin , es decir, poner un valor + (1) o un valor – (0).

digitalRead(): hace lectura desde un pin especifico, un 1 o un 0.

Analógicas:

analogRead(): toma lectura analógica entre 0 y 5 voltios y posteriormente se realiza la conversión análoga digital A/D.

analogWrite(): Transfiere un valor analógico entre 0 y 5 voltios. A un pin especifico de la tarjeta.

TIEMPO:

delay(): genera una pausa de tiempo determinada en milisegundos

delayMicroseconds():genera una pausa de tiempo determinada en microsegundos.

Ejemplo básico de encendido y apagado de un Led por medio de la programación en ARDUINO UNO.

```
int led = 13;  // definición de variable
void setup() {  // definición de entradas y salidas
pinMode(led, OUTPUT); // led será salida
}
void loop() { // definición de función infinita.
digitalWrite(led, HIGH);   //trasladar un valor alto a la variable led
delay(1000);// esperar por 1000 milisegundos o 1 segundo
digitalWrite(led, LOW); //trasladar un valor bajo a la variable led
delay(1000); // esperar por 1000 milisegundos o 1 segundo
}
```

4.7. Estructuras de Control

a) IF:

if es un instrucción donde se busca verificar el valor verdadero de una determina instrucción.

```
if (unaVariable ?? valor)
{
```

```
ejecutaInstrucciones;
}
```

b) if… else

Viene a ser una estructura que se ejecuta para cumplir una condición verdadera o falsa

```
if (inputPin == HIGH) // si el valor de la entrada inputPin es alto
{
instruccionesA; //ejecuta si se cumple la condición
}
else
{
instruccionesB;
```

Ejemplo

```
if (inputPin < 500)
{
instruccionesA; // ejecuta las operaciones A
}
else if (inputPin >= 1000)
{
instruccionesB; // ejecuta las operacione B
}
else
{
instruccionesC; // ejecuta las operaciones C
}
```

c) FOR:

La estructura for se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces hasta cumplir una condición.

```
for (inicialización; condición; expresión)
{
Ejecuta Instrucciones;
}
```

Ejemplo

```
for (int i=0; i<20; i++) // declara i, prueba que es menor que
20, incrementa i en 1
{
digitalWrite(13, HIGH); // envía un 1 al pin 13
delay(250); // espera ¼ seg.
digitalWrite(13, LOW); // envía un 0 al pin 13
delay(250); // espera ¼ de seg.
}
```

d) WHILE:

Un bucle de tipo while se ejecuta de forma continua mientras la expresión dentro de los paréntesis en la cabecera del bucle sea verdadera.

```
while (condición)
{
Ejecutar Sentencias;
}
```

Ejemplo

```
While (variable < 500) //
{
Acciones;
variable++; // aumenta la variable en 1
}
```

e) Do while

El bucle do while opera de manera similar al bucle while, pero la diferencia es que la condición se evalúa al final del bucle (Ruiz Gutiérrez, 2007).

```
do
{
Instrucciones;
} while (condición);
```

Ejemplo

```
do
{
x = leeSensor();
delay(50);
} while (x < 100);
```

6. Ejercicios resueltos

1. En este ejercicio el LED está conectado al pin 13 y se enciende y apaga ("parpadea") cada segundo. En este caso, la resistencia en serie con el LED puede omitirse, ya que el pin 13 de Arduino ya incorpora esta resistencia en la placa (Ruiz Gutiérrez, 2007).

   Solución

```
int led13 = 13;
void setup() // configura el pin de salida
{
pinMode(led13, OUTPUT); // configura el pin 13 como
salida
}
void loop() // inicia el bucle del programa
{
digitalWrite(led13, HIGH); // activa el LED
delay(1000); // espera 1 segundo
digitalWrite(led13, LOW); // desactiva el LED
```

```
delay(1000); // espera 1 segundo
}
```

2. En este ejemplo se lee un simple switch o pulsador conectado a PIN2. Cuando el interruptor está cerrado el pin de entrada se lee ALTO y encenderá un LED colocado en el PIN13 (Ruiz Gutiérrez, 2007).

   Solución

```
int led13 = 13; // pin 13 asignado para el LED de salida
int inPin2 = 2; // pin 2 asignado para el pulsador
void setup() // Configura entradas y salidas
{
pinMode(led13, OUTPUT); // declara LED como salida
pinMode(inPin2, INPUT); // declara pulsador como entrada
}
void loop()
{
if (digitalRead(inPin2) == HIGH)
{
digitalWrite(led13, HIGH); // enciende el LED
delay(1000); // espera 1 segundo
digitalWrite(led13, LOW); // apaga el LED
}
}
```

3. El siguiente ejemplo muestra como el transistor MOSFET conmuta 5 veces cada segundo.

   Solución

```
int outPin4 = 4;
void setup()
{
pinMode(outPin4, OUTPUT);
}
void loop()
{
for (int i=0; i<=4; i++)
{
digitalWrite(outPin4, HIGH);
delay(300); // espera 1/4 segundo
digitalWrite(outPin4, LOW); //
delay(300); //
}
delay(1100); // espera
}
```

4. El siguiente ejemplo lentamente hace que el LED se ilumine y se apague haciendo uso de dos bucles (Ruiz Gutiérrez, 2007).

   Solución

```
int ledPin9 = 9;
void setup(){}
void loop()
{
for (int i=0; i<=270; i++) //   i aumenta
{
analogWrite(ledPin9, i);
delay(1000); //  pausa por 1 segundo
}
for (int i=270; i>=0; i--) //  i decrementa
{
analogWrite(ledPin9, i);
delay(100); //    pausa
}
}
```

5. El siguiente ejemplo utiliza un potenciómetro para controlar un el tiempo de parpadeo de un LED (Ruiz Gutiérrez, 2007).

Solución

```
int potPin0 = 0;
int ledPin12 = 12;
void setup()
{
pinMode(ledPin12, OUTPUT);
}
void loop()
{
digitalWrite(ledPin13, HIGH);
delay(analogRead(potPin0));
digitalWrite(ledPin12, LOW); // pone ledPin en off
delay(analogRead(potPin0)); // detiene la ejecución un tiempo “potPin”
}
```

6. Utilizamos una función para leer el valor analógico y establecer un tiempo de retardo. Este tiempo controla el brillo de un diodo LED conectado en la salida (Ruiz Gutiérrez, 2007).

Solución

```
int ledPin9 = 9;
int analogPin0 = 0;
pin 0
void setup(){}
void loop()
{
for (int i=0; i<=258; i++) // aumenta de valor de i
```

```
{
analogWrite(ledPin9, i); //
delay(delayVal()); // pausa un tiempo
}
for (int i=258; i>=0; i--) // decrementa  i
{
analogWrite(ledPin9, i);
delay(delayVal());
}
}
int delayVal() //  almacena el tiempo de retardo
{
int v;
v = analogRead(analogPin0);
v /= 8; // transforma el valor de 0-1024 a 0-128
return v;
}
```

7. Este ejemplo utiliza la función servoPulse para mover el servo de 10º a 170 º (Ruiz Gutiérrez, 2007).

Solución

```
int servoPin2 = 2;
int myAngle1; // ángulo de  0-180
int pulseWidth; // anchura del pulso para la función servoPulse
void setup()
{
pinMode(servoPin2, OUTPUT);
}
void servoPulse(int servoPin2, int myAngle)
{
pulseWidth = (myAngle1 * 10) + 600; // hay un retardo
digitalWrite(servoPin2, HIGH); // activa el servo
delayMicroseconds(pulseWidth); // pausa
digitalWrite(servoPin2, LOW); // desactiva el servo
delay(20);
}
void loop()
{
// el servo inicia su recorrido en 10º y gira hasta 170º
for (myAngle1=10; myAngle1<=170; myAngle1++)
{
servoPulse(servoPin2, myAngle1);
}
```

```
for (myAngle1=170; myAngle1>=10; myAngle1--)
{
servoPulse(servoPin2, myAngle1);
}
}
```

7. Referencias bibliográficas

Arduino. (07 de 10 de 2024). *Arduino*. Obtenido de https://www.arduino.cc/reference/es/

Ruiz Gutiérrez, J. M. (01 de 08 de 2007). Arduino: Manual de Programación . USA.