15

Programación y robótica

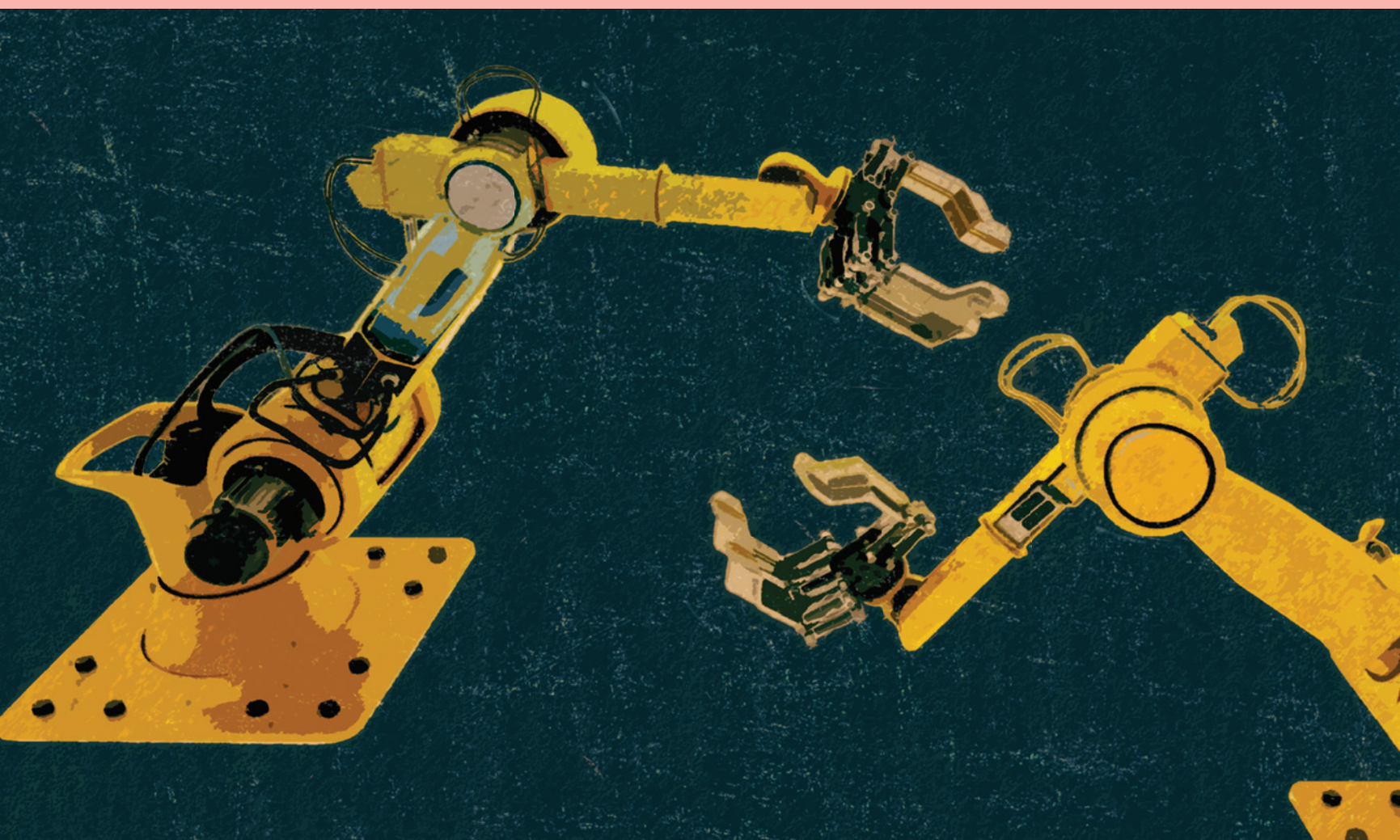
- Software de programación
- Elementos físicos
- Elementos de un programa

«Programación» y «robótica» son conceptos que van de la mano en el estudio de la tecnología actual. Para comprender el concepto de «robot», resulta muy útil comparar su funcionamiento con el de un ser humano.

Así pues, el cuerpo humano dispone de una unidad central, que es el cerebro, que se encarga de procesar la información que recibe de los órganos sensoriales, como ojos u oídos, y que envía órdenes a otros elementos, como músculos o cuerdas vocales, para generar una respuesta determinada. Un robot sustituye el cerebro humano por un dispositivo microcontrolador, los órganos sensoriales por sensores de diversa índole y los elementos actuadores por servomotores, timbres, ledes, etc. Hasta aquí la parte física de un robot (hardware).

Sin embargo, todos estos elementos no serían más que un amasijo de cables y piezas metálicas incapaces de realizar acción alguna de no ser por la existencia de una serie de órdenes o comandos, almacenados en la microcontroladora, a los que llamamos «programa» (software).

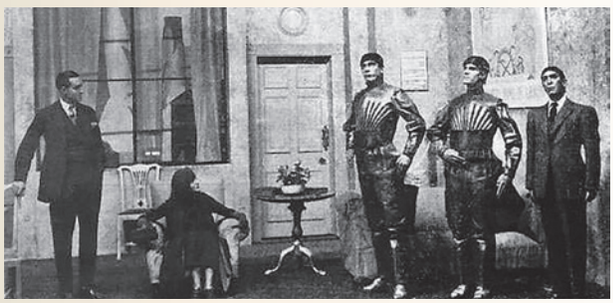
Por todo ello, en esta unidad aprenderemos, por un lado, a conectar los sensores y actuadores a la microcontroladora y, por otro, a diseñar un programa que gobierne todos estos elementos físicos.



CURIOSIDADES

Etimología de la palabra «robot»

El término «robot» fue utilizado por primera vez en 1920 por el dramaturgo checoslovaco Karel Čapek en su obra *R. U. R. (Rossum's Universal Robots, Robots Universales Rossum). R. U. R.* cuenta la historia de un empresario que construye una fábrica en una isla perdida en medio del océano donde dedicarse a la construcción de máquinas diseñadas a imagen y semejanza de los seres humanos con el objetivo de utilizarlas como mano de obra barata. Otro personaje dota a los robots de alma y, finalmente, estos se rebelan y declaran la guerra a la humanidad. Esta obra se ha llevado en numerosas ocasiones al teatro.



¿Quién inventó este concepto?

Sin embargo, no fue Karel Čapek quien inventó la palabra. En una breve carta escrita a la editorial del *Diccionario Oxford*, Karel atribuye a su hermano Josef la creación del término. Al parecer, la idea inicial de Karel era bautizar a sus máquinas como «*laboři*» (del latín *labor*, 'trabajo') pero cambió de idea tras pedir consejo a su hermano, que le sugirió designar sus máquinas como «*roboti*». La palabra «*robota*» significa, literalmente, 'trabajo' o 'labor' y, figuradamente, 'trabajo duro' en checo y en muchas lenguas eslavas.

El futuro ya está aquí

Los robots y otras tecnologías están transformando la sociedad tal y como la conocemos. Al oír la palabra «robot» dibujamos en nuestra cabeza máquinas de aspecto humano capaces de andar y hablar como nosotros. Imaginamos un futuro en el que los robots limpiarán nuestra casa, cocinarán nuestra comida y conducirán nuestro coche. La realidad es que ese futuro ya está aquí: aspiradores automatizados que memorizan la forma de nuestros salones, robots de cocina que realizan el 80 % del trabajo y nos ofrecen consejos, teléfonos móviles con los que conversamos y a los que ordenamos con nuestra voz diferentes tareas e, incluso, coches que conducen por nosotros. Este último es el caso del coche autónomo de Google. La combinación de un computador, diferentes sensores y automatismos ha permitido crear un coche que conduce por sí solo: reconoce los carriles, las señales de tráfico y los semáforos; sabe que llega a un cruce; ve a los otros vehículos, ciclistas y peatones; controla la distancia de seguridad con el vehículo que va delante y toma las decisiones pertinentes para no sufrir ningún percance. O sea, lo mismo que haría un conductor responsable y en plenas facultades.

Google Car ya ha recorrido cientos de miles de kilómetros, tanto en ciudad como en carretera. El estado de Nevada (EE. UU.) ya ha legislado sobre este tipo de vehículos, y es, por ahora, el único lugar por donde se puede circular con uno de ellos.

Actividades

1> La evolución de la tecnología conlleva muchos beneficios, pero también provoca inquietud. A menudo, vemos en películas de ficción como los robots llegan a desarrollar sentimientos y terminan por rebelarse contra los humanos. Sin llegar tan lejos, el temor a que los robots evolucionen lo suficiente como para sustituir a las personas en puestos de trabajo es una realidad. Discute con tu compañero o compañera sobre este tema y responde a las siguientes preguntas:

a) ¿Crees que los robots pueden suponer un peligro para la humanidad?

b) Identifica en tu entorno algún puesto de trabajo en el que un humano haya sido sustituido por un robot.

c) Enumera una serie de puntos a favor y en contra de la implantación de robots en el mundo laboral.

d) Enumera una serie de puntos a favor y en contra de la utilización de Google Car.

1. Software de programación

Un **programa** es una secuencia de órdenes que permiten al ordenador la realización de las acciones o tareas correspondientes a dichas órdenes. Los programadores o desarrolladores diseñan sus programas utilizando un lenguaje entendible que da como resultado el **código fuente**. Para que un ordenador pueda entender estas órdenes o instrucciones, el programa debe ser traducido al lenguaje del ordenador, esto es, el **lenguaje máquina** o **código binario**.

Un **software de programación** es el conjunto de herramientas que los desarrolladores utilizan para crear, depurar, traducir y mantener sus programas y aplicaciones.

¿Sabías que...?

El vocabulario informático incluye a menudo acrónimos procedentes de expresiones en inglés. Por ejemplo:
IDE: *Integrated Development Environment*, que significa 'entornos de desarrollo integrado'.
GUI: *Graphical User Interface*, cuyo significado en español es 'interfaz gráfica de usuario'.

Algunas de estas herramientas son:

- **Editores de texto.** Se emplean para crear o modificar el código fuente.
- **Compiladores.** Traducen el código fuente al lenguaje del ordenador para que este pueda comprender las instrucciones que recibe y actuar basándose en ellas.
- **Intérpretes.** Traducen y ejecutan el programa al instante.
- **Depuradores.** Sirven para controlar el desarrollo del programa. Así el programador puede realizar un seguimiento del código que se compila y ejecuta.
- **Entornos de desarrollo integrado (IDE).** Agrupan las anteriores herramientas en un entorno visual que facilita su manejo, de forma que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar, cargar el programa para su ejecución, etc. Habitualmente, cuentan con una avanzada interfaz gráfica de usuario (GUI).

1.1. Programación de placas controladoras

Existe una gran variedad de IDE para la creación de programas con los que hacer funcionar los diferentes tipos de placas controladoras. Algunos ejemplos son los siguientes:

- **IDE de Arduino**. Este programa permite editar y depurar el código. Usa un lenguaje propio basado en el lenguaje de programación de **Processing**. Contiene un compilador y un software de comunicación que, mediante conexión USB, permite cargar el programa diseñado en la memoria del controlador. Puede utilizarse con todas las placas controladoras compatibles con Arduino, como, por ejemplo, las placas de BQ.
- **Crumble**. Este software, diseñado expresamente para trabajar con el controlador Crumble, fue creado por **Redfern Electronics** y es un entorno de programación gráfico inspirado en **Scratch**. Resulta muy sencillo de usar y permite realizar programas de una forma rápida y eficaz.

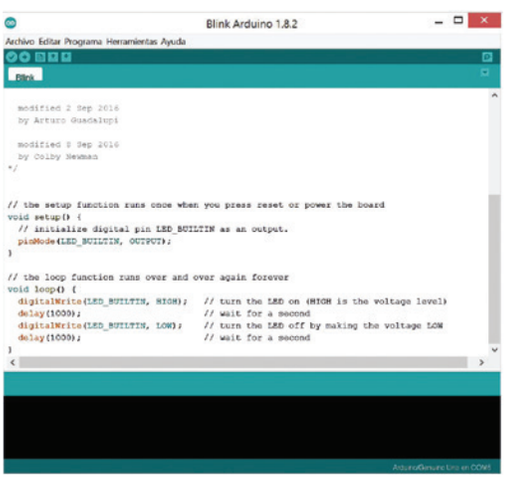


Fig. 15.1. Interfaz de Arduino.

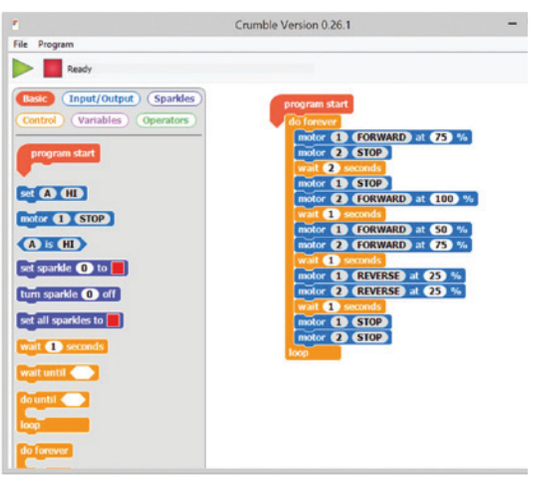


Fig. 15.2. Interfaz de Crumble.

- **S4A (Scratch for Arduino)**. Es una modificación de Scratch que, a través de bloques, permite diseñar programas para Arduino. La placa debe estar conectada mediante un conector USB al ordenador, de manera que el programa se ejecuta sin necesidad de cargarlo en la placa y, a la vez, se alimenta a través de esta conexión.
- **Ardublock**. Se trata de un entorno de programación visual por bloques para Arduino que se instala como una extensión del IDE de Arduino y facilita la tarea de redactar las instrucciones.
- **Bitbloq**. Está diseñado para programar placas BQ Zum y es compatible con Arduino. Puede descargarse o utilizarse a través de su plataforma on-line. Es similar a Scratch, ya que permite programar por bloques.
- **Processing**. Es un lenguaje de programación *open source* (de código abierto), de libre descarga y multiplataforma. Está pensado para crear programas con imágenes, animaciones e interacciones entre objetos.

En Internet

En estas direcciones podrás descargar los IDE que se citan en el texto.

- Crumble
https://redfernelectronics.co.uk/
- IDE de Arduino
https://www.arduino.cc/
- S4A
http://s4a.cat/
- Ardublock
http://blog.ardublock.com/
- Bitbloq
http://bitbloq.bq.com/
- Processing
https://www.processing.org/

1.2. Diagramas de flujo

Un **diagrama de flujo**, o *flow chart*, es la representación gráfica de un proceso o algoritmo.

En un diagrama de flujo, cada instrucción puede ser representada por un símbolo que contiene una breve descripción de la etapa del proceso. Los símbolos gráficos del flujo del proceso están unidos entre sí mediante flechas que indican el orden de ejecución.

Los diagramas de flujo utilizan una simbología normalizada, esto es, unas formas que deben significar siempre una acción específica. Los símbolos más utilizados son los siguientes:

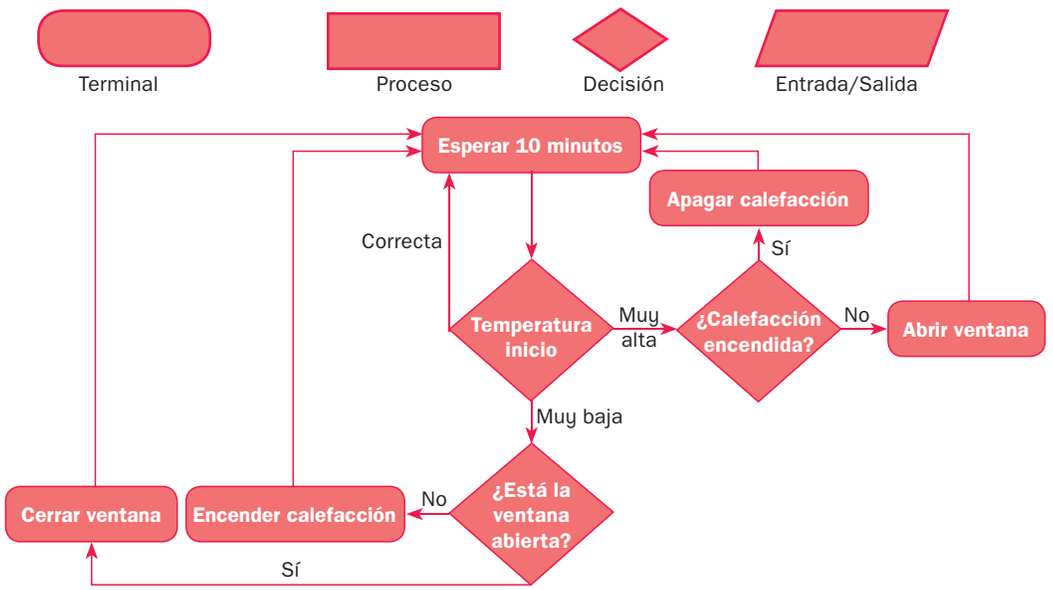


Fig. 15.3. Diagrama de flujo que representa el control de temperatura de un termostato.

Antes de empezar a programar y entrar en los detalles del lenguaje de programación, es muy recomendable dibujar un diagrama de flujo, como si de un boceto se tratara. Los diagramas de flujo permiten hacerse una idea de las acciones necesarias y de su relación con las demás, así como identificar la existencia de bucles repetitivos, el número de pasos del proceso u otras operaciones que el programa pueda requerir.

En Internet

Para crear tus propios diagramas de flujo con el ordenador, puedes trabajar con procesadores de texto, como Word o Writer, que ofrecen la posibilidad de dibujar flechas y simbología propia de estos utilizando la herramienta integrada de dibujo *Formas*. También podemos encontrar en Internet páginas que permiten diseñar diagramas on-line, como, por ejemplo, **SmartDraw**:
https://www.smartdraw.com/

Actividades

1> Dibuja un diagrama de flujo que represente:

a) El proceso del sistema de iluminación que funciona de manera que la luz solo se enciende durante unos segundos al detectar la presencia de una persona.

b) El funcionamiento de un toldo inteligente cuyo objetivo es evitar la luz solar directa durante las horas más calurosas (más de 20 °C).

2. Elementos físicos

La **robótica** es la rama de la tecnología que estudia el diseño y la construcción de máquinas capaces de desempeñar una tarea de manera automática. Si esta máquina es capaz de realizar únicamente una tarea o una serie de tareas determinadas que no pueden ser modificadas, se habla de **autómata.** Este sería el caso de un termostato o una tostadora. Pero si además esta máquina puede ser reprogramada para ejecutar otras tareas, se considera que es un **robot.**

Un **robot** es una máquina automática y programable que es capaz de captar información, procesarla y actuar en función de la decisión tomada.

Hoy en día la frontera entre robot y autómata no está clara, ya que los autómatas han evolucionado mucho y pueden ser programables.

Los componentes de un robot se pueden clasificar en tres grandes grupos:

- Las placas controladoras.
- Los sensores.
- Los actuadores.

¿Sabías que...?

La primera placa Arduino fue inventada en el año 2005 por un estudiante del instituto IVRAE (Instituto de Diseño Interactivo de Ivrea), de la provincia italiana de Turín.
Este estudiante, llamado **Massimo Banzi,** tenía el objetivo de cubrir una necesidad de aprendizaje de los estudiantes de computación y electrónica del mismo instituto ya que, por aquel entonces, adquirir una placa microcontroladora era relativamente caro. Banzi pretendía, además, evitar la quiebra de su escuela gracias a las ganancias que produciría vendiendo sus placas dentro del campus al accesible precio de un euro por unidad.

2.1. Placas controladoras

La **placa controladora** es el dispositivo que permite gobernar los sensores y actuadores de un robot.

Principalmente, una placa controladora se compone de un microprocesador y puertos de entrada y salida. Utilizando el software de programación elegido, en el ordenador se diseñan programas que la placa controladora ejecutará.

Las dimensiones de las placas presentan la ventaja de tener un tamaño reducido. Se pueden alimentar a través de un cable USB conectado al ordenador o mediante alimentación externa.

Hoy en día se puede encontrar en el mercado muchas versiones y marcas de placas controladoras que ofrecen una gran variedad de prestaciones con el objetivo de adaptarse a las necesidades de cada proyecto. Existen placas de diversos tamaños, con mayor o menor memoria de almacenamiento, con diferente número de entradas y salidas digitales o analógicas o, incluso, con conexión Bluetooth. Por otro lado, existe la posibilidad de adquirir tarjetas de expansión, también llamadas «shields», que se pueden acoplar a placas más básicas dotándolas así de características extras.

Una de las placas más utilizadas en educación es Arduino, que fue la primera marca italiana fabricante de placas microcontroladoras.

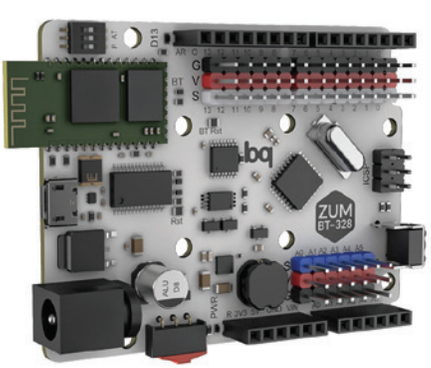


Fig. 15.4. Placa BQ Zum.

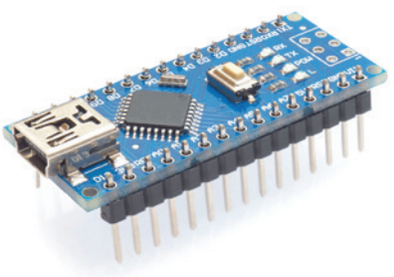


Fig. 15.5. Arduino Nano.

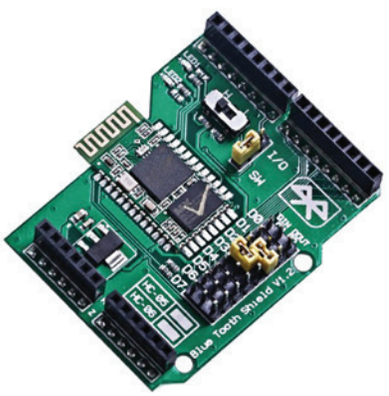
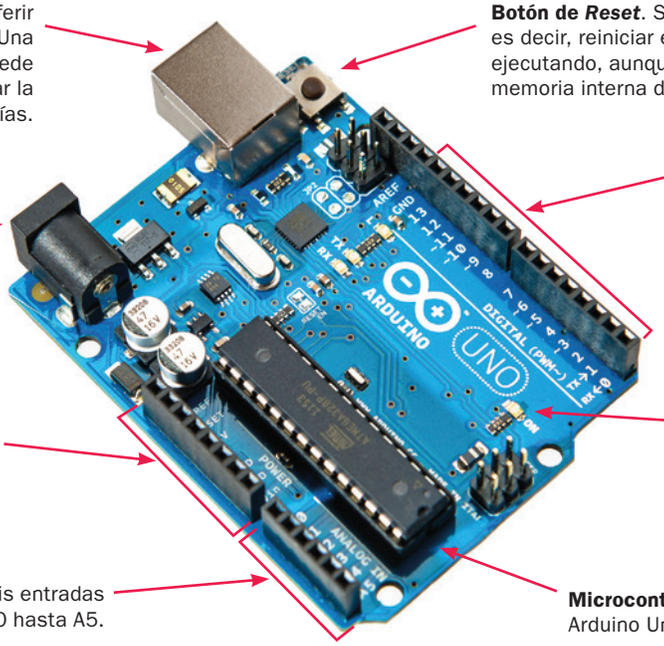


Fig. 15.6. Arduino Shield Bluetooth SHD18.

A. Placa Arduino Uno

Las principales partes de una placa Arduino Uno son las siguientes:



La corriente máxima de salida de cada uno de los pines de Arduino es de 40 mA y 200 mA para la suma total de los pines. Este es un dato importante a tener en cuenta a la hora de alimentar los diferentes componentes que se utilicen en el circuito.

Para dar salida a voltajes intermedios, se utilizan pines digitales etiquetados con el símbolo «~» o el acrónimo «PWM» («modulación por anchura de pulso», por sus siglas en inglés). Con estos pines se puede, por ejemplo, iluminar un led a media potencia.

Los componentes utilizados deberán conectarse a las entradas digitales o analógicas en función del rango de valores que se quiera enviar a un componente o recibir de él:

- Los pines digitales envían o reciben dos únicos valores, que son 0 y 5 voltios, y que corresponden a los valores lógicos «LOW», 'bajo' y «HIGH», 'alto', respectivamente. Con ellos podemos encender o apagar un led (actuando como pin de salida) o comprobar si un botón externo está encendido o apagado (actuando como pin de entrada).
- Los pines analógicos pueden recibir un rango continuo de voltaje entre 0 y 5 voltios. Actúan solo como pines de entrada.

Dado que, a menudo, los pines disponibles en una placa Arduino son insuficientes y que algunos componentes necesitan conectarse en serie con resistencias y el espacio de la placa es limitado, es muy habitual utilizar una placa de conexiones, también llamada **protoboard.**

Una placa protoboard permite ampliar las posibilidades de la placa controladora y el circuito electrónico que se quiera construir. Esta placa está formada por una serie de filas y columnas de pines conectados entre sí en su interior para disponer los elementos que el circuito requiera, en serie o en paralelo. Para ello se emplean los **jumpers,** cables de diferente longitud provistos de terminaciones conectoras macho para su conexión.

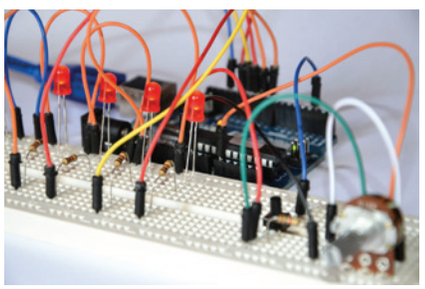


Fig. 15.7. Placa protoboard conectada a placa microcontroladora mediante jumpers.

Actividades

2> Dibuja una placa protoboard y conecta:

a) Tres resistencias en serie.

b) Tres resistencias en paralelo.

c) Dos resistencias en paralelo conectadas en serie con una tercera resistencia.

B. FPGA libres

Una **FPGA** o **matriz de puertas programables** (del inglés *Field Programmable Gate Array*) es un dispositivo lógico programable, es decir, un chip cuyas puertas lógicas se pueden programar. Así pues, la interconexión y la funcionalidad de los bloques lógicos pueden ser configuradas, de manera que no se modifica una capa de software, como pasa en Arduino, sino el hardware del dispositivo.

El llamado formato *bitstream* de una FPGA específica define cómo están conectados los elementos internos de una FPGA y cómo interactúan entre ellos. Las empresas que fabrican y venden FPGA no suelen hacer público este formato para proteger su producto y asegurar las ventas; sin embargo, este hecho supone a su vez una barrera a su acceso y desarrollo para muchos usuarios. Así pues, las FPGA se consideraban dispositivos privativos o cerrados, ya que, para trabajar con ellas, resultaba necesario utilizar las herramientas que proveía una empresa en particular.

Sin embargo, en diciembre de 2015, el desarrollador **Clifford Wolf**, presentó el resultado de su proyecto **Icestorm,** una serie de herramientas libres *(open source toolchain)* para el desarrollo de una FPGA modelo iCE40. El trabajo de tres años llevado a cabo por el equipo de Wolf liberó el diseño y el método de programación de la placa iCE40 y abrió la veda de programación de las FPGA. Desde entonces se puede hablar de FPGA libres. Este hito ha supuesto toda una revolución en el mundo de la robótica, ya que las FPGA se utilizan en multitud de campos, que van desde la industria de fabricación mecanizada hasta la industria aeroespacial, pasando por la industria de impresoras 3D o las aplicaciones de robótica simple.

Como consecuencia de estos avances, ha sido posible crear placas con FPGA libres y programables utilizando únicamente software libre. Este es el caso de la placa IceZUM Alhambra, desarrollada originalmente en BQlabs y diseñada por **Eladio Delgado**, en colaboración con **Juan González.** El software libre Icestudio, creación de **Jesús Arroyo,** permite, además, programar esta placa de manera directa y gráfica, facilitando así a los estudiantes la introducción a la electrónica digital.

@ *En Internet*

http://www.iearobotics.com

Esta es la wiki de Juan González; en ella puedes encontrar multitud de actividades, información ampliada y videotutoriales creados por él mismo.

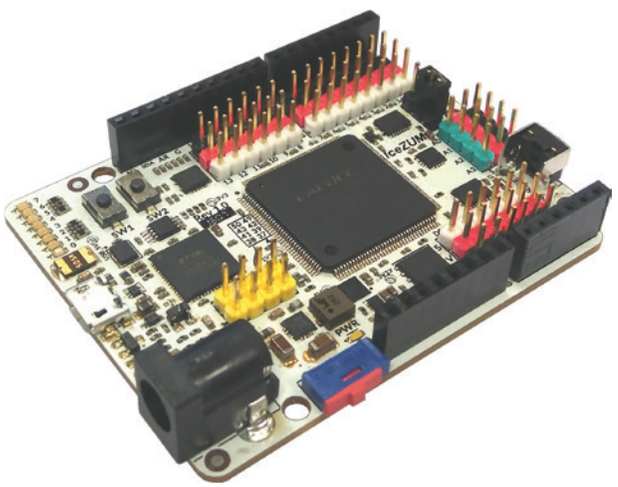


Fig. 15.8. Placa IceZUM Alhambra.

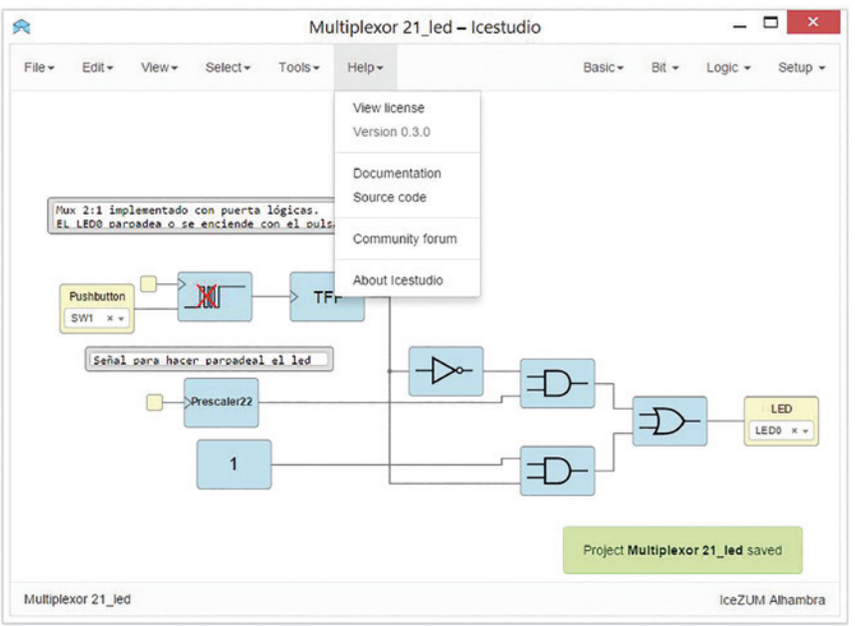


Fig. 15.9. Software Icestudio.

Actividades

3> Accede a la wiki de Juan González y busca información sobre «Mini-PI». Enumera sus componentes, describe su utilidad y explica qué características la diferencian de la placa controladora BQ Zum.

2.2. Sensores

Un **sensor** es un dispositivo que permite captar información del entorno. Los sensores son los órganos sensoriales de un robot.

Los sensores pueden ser muy variados: de temperatura, de humedad, de movimiento, de presencia, de intensidad luminosa, de distancia, de presión, de inclinación, etc.

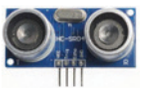
Tipo de sensor	Características	Figura
Sensor PIR *(passive infrared)* o de movimiento	Es sensible al cambio en la radiación de infrarrojos de objetos o personas.	
Sensor LDR *(light dependent resistor)*	Varía su resistencia interna en función de la cantidad de luz.	
Sensor de ultrasonidos	Calcula la distancia a la que está un objeto mediante ondas de ultrasonidos.	
Sensor de infrarrojos	Diferencia entre blanco y negro en función de la luz reflejada por el objeto.	
Sensor NTC *(negative temperature coefficient)* o de temperatura	Varía su resistencia interna en función de la temperatura.	
Sensor de sonidos	Detecta sonidos. También puede emitir sonidos actuando como un timbre o zumbador.	
Sensor de humedad	Detecta la cantidad de humedad existente en el aire del medio en que se encuentra.	
Acelerómetro o sensor de inclinación	Registra la orientación de un objeto, golpes o vibraciones, aceleraciones y la variación de gravedad.	
Pulsador	Son botones con los que iniciar o parar una acción.	
Potenciómetro	Es una resistencia variable que se puede modificar al hacer girar el mando de que viene provisto.	

Tabla 15.1. Tipos de sensores.

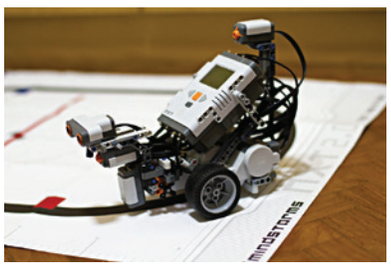
Estos sensores se conectan a una placa protoboard. Algunos fabricantes como BQ integran los sensores a un pequeño soporte al que también van soldados los cables necesarios.

Actividades

4> ¿Cuáles crees que son los sensores necesarios para construir un robot siguelíneas?

¿Sabías que...?

El **robot siguelíneas** es uno de los robots más sencillos de construir y su comportamiento resulta muy sorprendente. Este robot, provisto de motores y ruedas, se coloca sobre una superficie blanca con una línea dibujada de color negro (o viceversa) y, gracias a los sensores de que dispone, es capaz de detectar si está encima de la línea o no. En función de esta información, hace que los motores giren en un sentido u otro con el objetivo de corregir la trayectoria y nunca salirse del camino indicado por la línea.



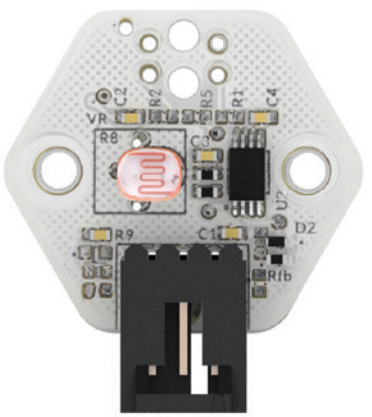


Fig. 15.10. Sensor LDR de BQ.

2.3. Actuadores

Un **actuador** es un dispositivo capaz de transformar la energía eléctrica en la activación de un proceso con la finalidad de generar luz, sonido, etc. Los actuadores reciben órdenes del controlador y actúan en consecuencia.

Los actuadores más utilizados son motores, indicadores luminosos (diodos), zumbadores, pantallas LCD o displays numéricos. La mayoría pueden alimentarse con la corriente máxima de salida de los pines digitales de Arduino, que es de 40 mA, pero conviene recordar que la corriente máxima total de la placa es de 200 mA.

Por otro lado, algunos actuadores requieren *drivers* (controladores) adicionales.

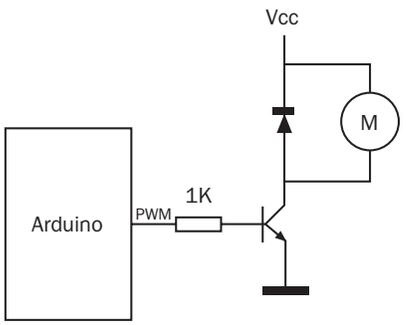


Fig. 15.11. Circuito con amplificador para alimentar un motor de CC con placa Arduino.

Tipo de actuador	Características	Figura
Los **diodos LED** *(light emitting diode)*	Transforman la electricidad en luz. Tienen polaridad, por lo que hay que prestar atención al conectarlos. De sus dos patas, la más larga es el ánodo, o terminal positivo, y la más corta, el cátodo, o terminal negativo. Se deben conectar en serie con una resistencia de aproximadamente 220 Ω para que puedan soportar la corriente.	
Motores de corriente continua	Transforman la energía eléctrica en mecánica, de manera que el motor girará en un sentido u otro en función de la polaridad. En el caso de la placa Arduino, requieren una corriente superior a 40 mA para funcionar, por lo que será necesario acoplar una «shield» o tarjeta de expansión para amplificar la salida de Arduino con un transistor o conectar un relé que accione el motor.	
Servomotores	Son motores CC de precisión que contienen una reductora y permiten que gire de 0° a 180° tomando cualquier posición intermedia. Por ejemplo, para que un motor abra y cierre una compuerta, se puede programar de manera que para abrirla vaya de la posición 0° a la posición 100° y para cerrarla, de 100° a 0°.	
Zumbadores o *buzzers*	Son dispositivos piezoeléctricos que emiten sonido. Se puede controlar el tiempo de duración del sonido y también el tono que reproducen.	

Tabla 15.2. Tipos de actuadores.

Actividades

5> Completa la tabla siguiente indicando a qué entradas y salidas conectarías los siguientes sensores y actuadores:

LED		Servomotor	
Sensor de temperatura NTC		Zumbador o buzzer	
Sensor LDR		Sensor de ultrasonidos	
Sensor de infrarrojos		Motor CC	

A. Divisores de tensión

Un **divisor de tensión o de voltaje** es un circuito cuyos componentes eléctricos están configurados de manera que la tensión o voltaje de una fuente se reparte entre una o más resistencias conectadas en serie.

Los divisores de tensión tienen gran cantidad de aplicaciones y se encuentran entre los circuitos más utilizados en electricidad y electrónica.

Para calcular la caída de voltaje en cada una de las resistencias del circuito que se muestra en la figura, se aplica la ley de Ohm y otras fórmulas básicas de electricidad.

- Se calcula el valor de la resistencia total o equivalente:

$$R_T = R_1 + R_2$$

- Se aplica la ley de Ohm para obtener las expresiones de V_{in} y V_{out} que corresponden a la caída de tensión en R_1 y R_2 respectivamente:

$$V_{out} = I \cdot R_2$$

$$V_{in} = I \cdot R_T = I \cdot (R_1 + R_2), \text{ de donde } I = V_{in}/(R_1 + R_2)$$

  donde I es la corriente que circula por el circuito, V_{out}, el voltaje que se quiere encontrar, y R_2, el valor de la resistencia que provoca la caída de tensión.

- Se obtiene la fórmula final:

$$V_{out} = V_{in} \cdot R_2/(R_1 + R_2)$$

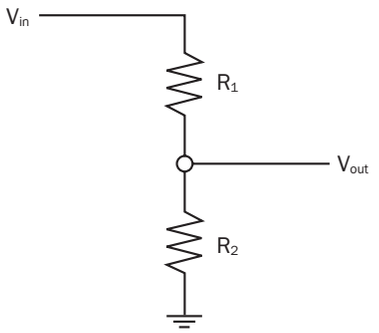


Fig. 15.12. Esquema de un divisor de voltaje.

Dado que para una placa Arduino es muy sencillo medir voltajes, gracias a su convertidor analógico-digital, el divisor de voltaje es ampliamente utilizado para facilitar la medición de datos obtenidos desde ciertos sensores, que son, en realidad, resistencias cuyo valor varía de acuerdo con ciertas magnitudes, como la luz, el ruido, etc. Esto se consigue agregando en serie a la resistencia variable (sensor) otra resistencia fija, formando así un divisor de voltaje. A medida que la resistencia variable cambia su valor, el valor del voltaje de salida V_{out} irá también variando, permitiendo al microcontrolador establecer lecturas según el voltaje recibido.

Este es el caso del ejemplo mostrado en la figura 15.13, en el que la resistencia del LDR disminuye con el aumento de luminosidad. Al hacerlo, el punto en el cual se mide el voltaje aumenta.

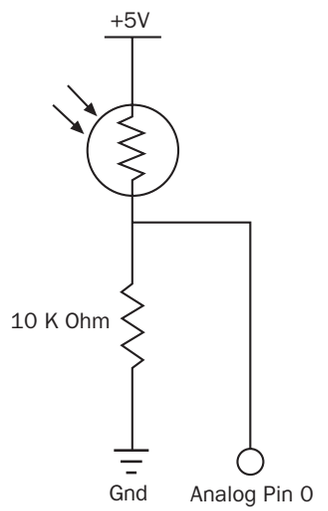


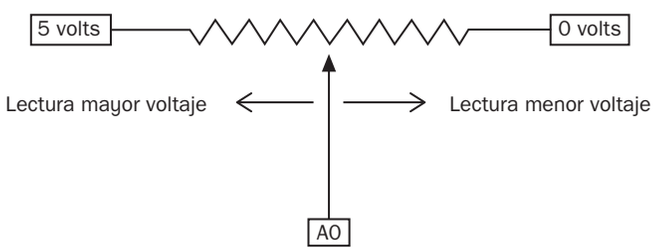
Fig. 15.13. Esquema de un divisor de voltaje para un sensor LDR.

Ejemplo 1

Veamos como ejemplo lo que ocurre en un potenciómetro. Este elemento consiste en una resistencia y un contacto móvil que divide la resistencia en dos más pequeñas y, a medida que varía su posición, modifica el valor resistivo de cada una de ellas.

En la figura siguiente variaremos la lectura de un potenciómetro cuyos pines extremos están polarizados a 5 V. Por lo tanto, la lectura que hagamos desde el pin A0 tendrá un valor entre 0 y 5 V, dependiendo de si el contacto se desliza a la izquierda o a la derecha.

Las lecturas que hagamos en las entradas analógicas darán valores entre 0 y 1023, correspondiendo el valor 0 a una lectura de 0 V y el valor 1023 a una lectura de 5 V. Por ejemplo, una lectura de 512 corresponde a 2,5 V.



3. Elementos de un programa

Son los términos y símbolos utilizados para redactar los programas. Resulta muy útil imaginar cada uno de ellos como una pieza de puzle que encaja con el resto; de hecho, así se representan en programas como Scratch, S4A o Bitbloq, que utilizan bloques de diferentes colores en función de su utilidad.

En los apartados que siguen se exponen los elementos más relevantes.

3.1. Tipos de datos

Los **tipos de datos** o **variables** son las unidades de información que el programa procesará. Para que el programa sepa cómo tratar cada variable, el nombre de la variable irá precedido por una abreviatura que determinará su tipo y el rango de valores que puede almacenar. A este identificador le sigue un valor, que es el contenido de ese dato o variable.

Algunos tipos de datos son:

- **Numéricos:** trabajan con magnitudes numéricas de números reales, enteros, decimales, etc.
- **Caracteres** o **cadenas:** representan caracteres, como números, letras o símbolos. Las cadenas son caracteres concatenados con los que dar forma a palabras o frases.
- **Booleanos:** solo admiten dos valores: verdadero/falso, sí/no, 1/0, etc.
- **Punteros:** señalan la dirección de memoria de otra variable.
- **Tablas:** están compuestas por filas y columnas.
- **Listas** o **colas:** están compuestas por elementos lineales enlazados.
- **Árboles** o **grafos:** están compuestos por elementos no lineales enlazados.
- **Ficheros de bases de datos:** son archivos que contienen numerosos registros.

Así pues, por un lado, existen las **variables,** que, como su propio nombre indica, pueden cambiar su valor durante la ejecución del programa, y, por otro, las **constantes,** cuyo valor permanece fijo todo el tiempo.

Tipo	Descripción	Ejemplo
byte	Almacenan un número entero entre 0 y 255. Codificado en un octeto (1 byte).	*byte minumero = 129;*
int	Integer = Entero Almacenan un número entero entre 32767 y –32767. Codificado en dos octetos (2 bytes).	*int unentero = 28927;*
long	Almacenan un número entero entre 2147483647 y –2147483647. Codificado en cuatro octetos (4 bytes).	*long unenterolargo = 67876;*
float	Almacenan un número real, es decir, con decimales.	*float numerodecimal = 3,56;*
boolean	Almacenan una variable booleana que puede tomar solamente dos valores: 1 o 0.	*boolean mibooleano = true;*
char	Un carácter ASCII almacenado en 8 bits (1 byte). Permite almacenar caracteres como valores numéricos (su código ASCII asociado). El código ASCII para el carácter 'a' es 97; si se le añaden tres se obtiene el código ASCII del carácter 'd'.	*char micaracter = 'a';* *char micaracter = 97;*
array	Sirve para almacenar colecciones de diferentes tipos de datos.	*int misnumeros = {1, 2, 3, 4};*
string	Es un array de tipo char.	*char micadena = "hola";*

Tabla 15.3. Tipos de datos.

¿Sabías que...?

El código ASCII es el código fuente del texto desarrollado en un lenguaje de programación. Gracias a este código es posible representar caracteres alfanuméricos y facilitar la comunicación entre los diferentes dispositivos digitales. A continuación, puedes ver la conversión de algunos caracteres a código binario utilizando el citado código.

Código ASCII

```
0 – 0 0 1 1 0 0 0 0
1 – 0 0 1 1 0 0 0 1
2 – 0 0 1 1 0 0 1 0
3 – 0 0 1 1 0 0 1 1
4 – 0 0 1 1 0 1 0 0
5 – 0 0 1 1 0 1 0 1
6 – 0 0 1 1 0 1 1 0
7 – 0 0 1 1 0 1 1 0
8 – 0 0 1 1 1 0 0 0
9 – 0 0 1 1 1 0 0 1

A – 0 0 1 1 0 0 0 1
B – 0 0 1 1 0 0 1 0
C – 0 0 1 1 0 0 1 1
D – 0 0 1 1 0 1 0 0
E – 0 0 1 1 0 1 0 1
F – 0 0 1 1 0 1 1 0
G – 0 0 1 1 0 1 1 0
H – 0 0 1 1 1 0 0 0
```

3.2. Operadores

Los **operadores** son los símbolos que permiten realizar operaciones aritméticas, lógicas o relacionales entre los datos con que se trabaja. Existen distintos tipos de operadores:

- Los **operadores aritméticos** sirven para trabajar con términos numéricos como si de operaciones matemáticas se tratara. En ellos es importante prestar atención al uso de paréntesis y así asegurar que el resultado obtenido es el deseado. Por ejemplo:

  IF (cantidad de luz <= 300) **THEN** (acción encender led)

  Por otro lado, existen operadores cuya función es aumentar o disminuir en una unidad cierto valor almacenado en una variable para así conseguir estructuras que en programación se denominan **contadores**. Por ejemplo:

  WHILE (variable tiempo <30) **DO** (variable tiempo ++)

- Los **operadores lógicos** o **booleanos** permiten conectar varias propiedades de manera que deban cumplirse ambas (AND), solo una de ellas o alguna de ellas (OR) o ninguna de las condiciones descritas (NOT). Por ejemplo, para que se cumplan dos condiciones a la vez se escribe:

  IF (condición 1) **AND** (condición 2) **THEN DO** (acción)

- Los **operadores relacionales** se utilizan para comparar dos valores y podemos determinar si se quiere que los dos valores sean iguales, distintos, mayores, menores, mayores que o iguales que una determinada variable o constante.

Recuerda

Conviene tener cuidado de no causar *overflow* con las operaciones, es decir, debemos evitar que la magnitud del resultado de una operación sea mayor a la que el tipo de variable asociado pueda contener. Por ejemplo, el número más grande que puede tolerar un entero en Arduino va de -32768 a 32767, por lo tanto, si intento multiplicar 60 × 1000, produciré un error por overflow en el programa.

Símbolo	Descripción	Tipo de operación
=	Asigna un valor.	Aritmética (de asignación)
+, -, *, /	Adición, sustracción, multiplicación, división.	Aritmética
%	Módulo. Devuelve el resto de una división.	Aritmética
++, –	Incrementa, decrementa el valor de una variable.	Aritmética
+=, -=, *=, /=	x += y es equivalente a x = x + y.	Relacional
==, !=, <>	Igual, distinto, distinto.	Relacional
<, <=, >=, >	Menor, menor o igual, mayor o igual, mayor.	Relacional
¡, NOT	Negación.	Booleana
&&, AND	Producto lógico o conjunción.	Booleana
||, OR	Suma lógica o disyunción.	Booleana

Tabla 15.4. Tipos de operadores.

Actividades

6> Utiliza los símbolos de los operadores para escribir la expresión que correspondería a las siguientes operaciones:

a) Igualar una variable a 5.

b) Hacer que una variable sea menor que 7 y mayor o igual que 12.

c) Hacer que una variable sea mayor o igual que 8 o menor o igual que 1.

d) Definir una variable de tipo número entero llamada «pasos» y asignarle el valor 53.

e) Definir una variable para almacenar la palabra «Sacapuntas».

3.3. Funciones y programación estructurada

La **programación estructurada** se basa en una metodología de programación de refinamientos sucesivos. Se plantea una operación como un todo y se divide en segmentos de menor complejidad. La programación estructurada hace posible que, combinando esquemas sencillos, se pueda llegar a construir sistemas amplios y complejos, pero de fácil entendimiento, pues evita códigos largos y repetitivos.

Para la representación gráfica de la programación estructurada se utilizan **diagramas de flujo,** o *flow charts*, que representan el programa con sus entradas, procesos y salidas.

La programación estructurada propone segregar los procesos en estructuras lo más simples posible con la ayuda de las siguientes instrucciones de control:

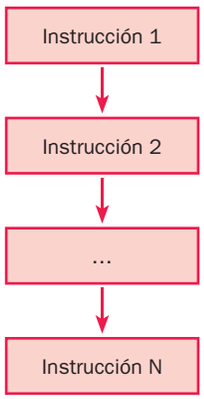


Fig. 15.14. Estructura secuencial.

- **Secuenciales:** son bloques de instrucciones sucesivas que se ejecutan de forma ordenada y seguida.
- **Condicionales** o **selectivas:** son instrucciones que permiten definir condiciones. Dependiendo de si una condición se cumple o no, el programa seguirá por un camino u otro. Por ejemplo:

If (condición) **then** (instrucción 1) **else** (instrucción 2)

que podría traducirse como «**Si** se cumple la condición, ejecuta la instrucción 1; **en caso contrario,** ejecuta la instrucción 2».

Cuando en una estructura condicional solo una de las dos respuestas da lugar a una serie de acciones, se trata de una estructura condicional simple. En cambio, si ocurre que se concatenan varias estructuras condicionales y de una condición surgen varios caminos o respuestas con una serie de acciones correspondientes a cada una de las respuestas posibles, se obtienen las llamadas estructuras condicionales múltiples.

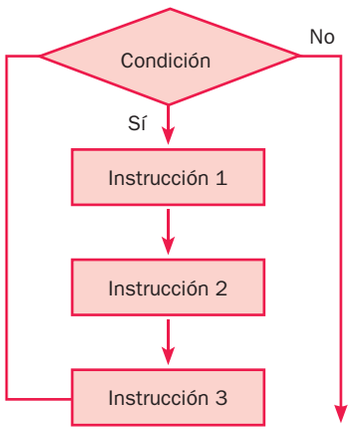


Fig. 15.15. Estructura repetitiva.

- **Repetitivas** o **de iteración:** son instrucciones que se repiten un número limitado de veces o hasta que se cumple una condición definida. Se utilizan los bucles **for** y **while**. Son bucles (en inglés, *loop*), porque estas instrucciones hacen que lo que contienen se repita una y otra vez. Por ejemplo:

While (condición) **do** (instrucción)

que significa «**Mientras** se cumpla esta condición, ejecuta **(haz)** esta instrucción».

A continuación, se muestran los diagrames de flujo correspondientes a estructuras condicionales simples y múltiples.

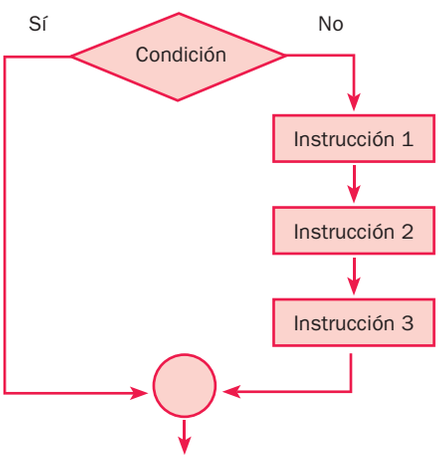


Fig. 15.16. Estructura condicional simple.

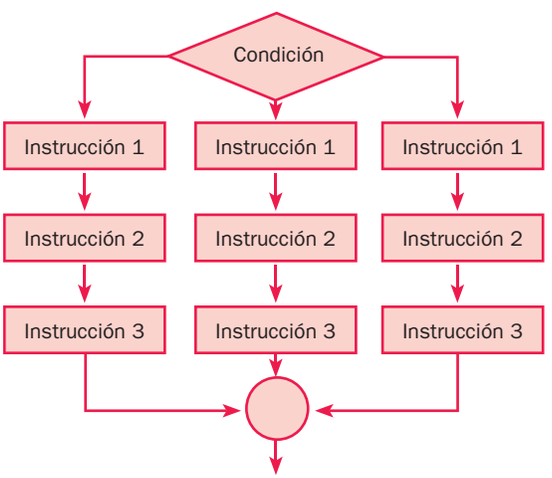


Fig. 15.17. Estructura condicional múltiple.

A. Elementos de control

Son las estructuras que permiten dar forma al programa, así como modificar el flujo de ejecución de las instrucciones que contiene. Combinando los elementos de control se pueden conseguir diferentes y diversos comportamientos, como ejecutar un grupo de sentencias mientras se cumpla una condición o hasta que se cumpla una condición determinada, ejecutar un grupo de sentencias en función del valor de una variable, etc.

Tipo	Descripción	Ejemplo

if	Se utiliza para probar si una determinada condición se ha alcanzado. Si es así, se ejecutarán una serie de operaciones. Si la condición no se cumple, el programa saltará y no ejecutará las operaciones.	`if (alguna Variable > 50)`
		`{`
		`  // do una operación`
		`}`

if... else	«Si esto no se cumple, haz esto otro». Se diferencia del «if» en que, en caso de que no se cumpla la condición, el programa no salta, sino que ejecuta otra serie de operaciones.	`if (valor_pin3 < 500)`
		`{`
		`  // do OperaciónA`
		`}`
		`else if (pinFiveInput >= 1000)`
		`{`
		`  // do OperaciónB`
		`}`
		`else`
		`  // do OperaciónC`
		`}`

for	Se utiliza cuando se requiere que una serie de acciones se repitan un número determinado de veces. Para ello se necesita una variable índice, una condición y un incrementador. En el ejemplo se consigue que un led parpadee tres veces tras pulsar un botón.	`int pulsador = 12;`
		`int led= 4;`
		`int i;`
		
		`void setup()`
		`{`
		`pinMode(pulsador, INPUT);`
		`pinMode(led, OUTPUT);`
		`}`
		
		`void loop()`
		`{`
		`  if (digitalRead(pulsador) == HIGH)`
		`  {`
		`   for (int i = 0; i<=2; i++)`
		`   {`
		`    digitalWrite(led, HIGH);`
		`    delay(1000);`
		`    digitalWrite(led, LOW);`
		`    delay(1000);`
		`   }`
		`  }`
		`else`
		`  {`
		`   digitalWrite(led, LOW)`
		`  }`
		`  }`
		
		// Se definen dos variables, una para el pulsador y otra para el led; se llamarán pulsador y led, respectivamente; «i» será el índice utilizado en las repeticiones y, de momento, no se le asigna ningún valor.
		
		// Se definen si dichas variables serán de entrada o salida. El pin conectado al pulsador será de entrada y el del led será de salida.
		
		//Al pulsar el botón, el pin digital recibirá el valor HIGH y ejecutará todas las operaciones contenidas en el bucle **for**.
		
		//Se asigna a la variable «i» el valor 0 y se establece la condición de que siempre que el valor de «i» sea menor que 2 el programa hará dos cosas:
		1) Aumentará el valor de la variable «i» en uno (i++)
		2) Ejecutará el código entre llaves "{..}". Así pues, las acciones descritas después ejecutarán el encendiendo del led durante un segundo y el apagado, durante otro segundo.
		
		//En caso de que el pin del pulsador no reciba la entrada HIGH, es decir, que no se haya pulsado el botón, el LED se mantendrá apagado.

while	Es un bucle que se ejecutará continuamente mientras se cumpla la expresión entre paréntesis. La variable de prueba deberá cambiar para que el programa salga del bucle.	`while(expresion){`
		`  // acciones`
		`}`

do... while	Funciona como el bucle while, con la salvedad de que la condición se prueba al final, por lo que el bucle se ejecutará al menos una vez.	`var = 0;`
		`while(var < 200){`
		`  // ejecuta una acción 200 veces`
		`  var++;`
		`}`

Tabla 15.5. Elementos de control.

3.4. Librerías

El IDE de Arduino puede utilizar funciones propias y funciones importadas. Las funciones importadas son aquellas que el programa no reconoce por defecto. Para que sea capaz de hacerlo se tienen que instalar librerías adicionales. Estas librerías permiten utilizar funciones específicas para manejar elementos como, por ejemplo, un servomotor. En la actualidad su instalación es sencilla.

Las librerías son fragmentos de código creados por terceros que se pueden incluir en el programa. Existen infinidad de librerías a disposición de los usuarios. Todas ellas son *open source* (de código abierto) y, por ello, se puede utilizar su código.

Las librerías, normalmente, incluyen una serie de archivos comprimidos en un archivo .zip o dentro de un directorio en el que se encuentra:

- Archivo de extensión .cpp, que contiene el código fuente de la librería.
- Archivo de extensión .h, que contiene las propiedades de las funciones de la librería.
- Archivo «Keywords.txt», que contiene las palabras clave que se resaltan en el IDE.
- Archivo «readme», con información adicional e instrucciones de cómo usar las funciones.
- Directorio «examples», con varios programas ejemplo que ayudan a entender cómo usar la librería (opcional).

Fig. 15.18. Cabecera de un programa con la librería servo.

A. ¿Cómo instalar librerías?

La manera más sencilla de instalar librerías es utilizar el gestor de librerías del IDE de Arduino disponible a partir de su versión 1.6.2. Esta herramienta se puede utilizar accediendo a:

Programa → Incluir Librería → Gestionar Librerías

Desde aquí se pueden ver las librerías instaladas, buscar entre las librerías disponibles, instalar nuevas librerías y también actualizarlas. Si se clica sobre la librería «Servo», por ejemplo, aparecerá automáticamente en la cabecera del programa la función «include» con el nombre de la librería instalada.

También se puede importar librerías descargadas previamente como un archivo .zip accediendo a:

Programa → Incluir Librería → Añadir Librería .ZIP

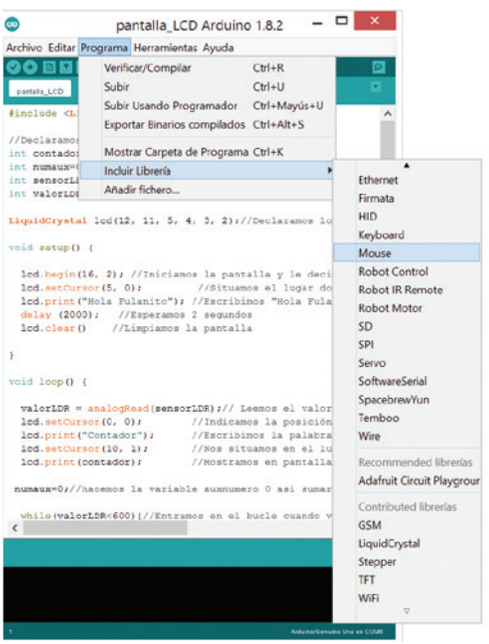


Fig. 15.19. Opción *Gestionar Librería*.

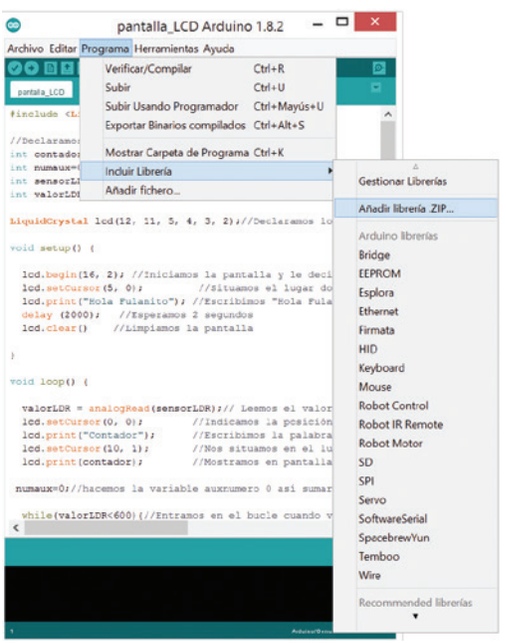


Fig. 15.20. Opción *Incluir Librería*.

Práctica 1. Instalación del IDE de Arduino y conexión de la placa Arduino Uno al ordenador

La primera vez que conectemos la placa Arduino a nuestro ordenador, deberemos instalar el IDE para que la placa sea reconocida y funcione correctamente. Estos son los pasos a seguir:

1. Descargar el IDE de Arduino desde la sección de descargas de www.arduino.cc e instalarlo.
2. Conectar la placa vía USB y seleccionar la opción «Instalar automáticamente el software» o esperar a que se instale automáticamente.
3. Abrir el IDE de Arduino e indicar qué placa vamos a utilizar y el puerto al que está conectada. Sabemos a qué puerto está conectada porque es el que se indica automáticamente al acceder a Herramientas > Puerto COM. Una vez seleccionado el puerto correcto, aparecerá un símbolo de *check* a su lado y podremos leer en la parte inferior derecha de la ventana el tipo de placa y número de puerto seleccionados.

Cada vez que abramos el programa y conectemos la placa, debemos comprobar que el puerto seleccionado es el correcto.

Botones y áreas principales de la interfaz del IDE de Arduino

Menú principal con acceso a las pestañas *Archivo, Editar, Programa, Herramientas* y *Ayuda*.

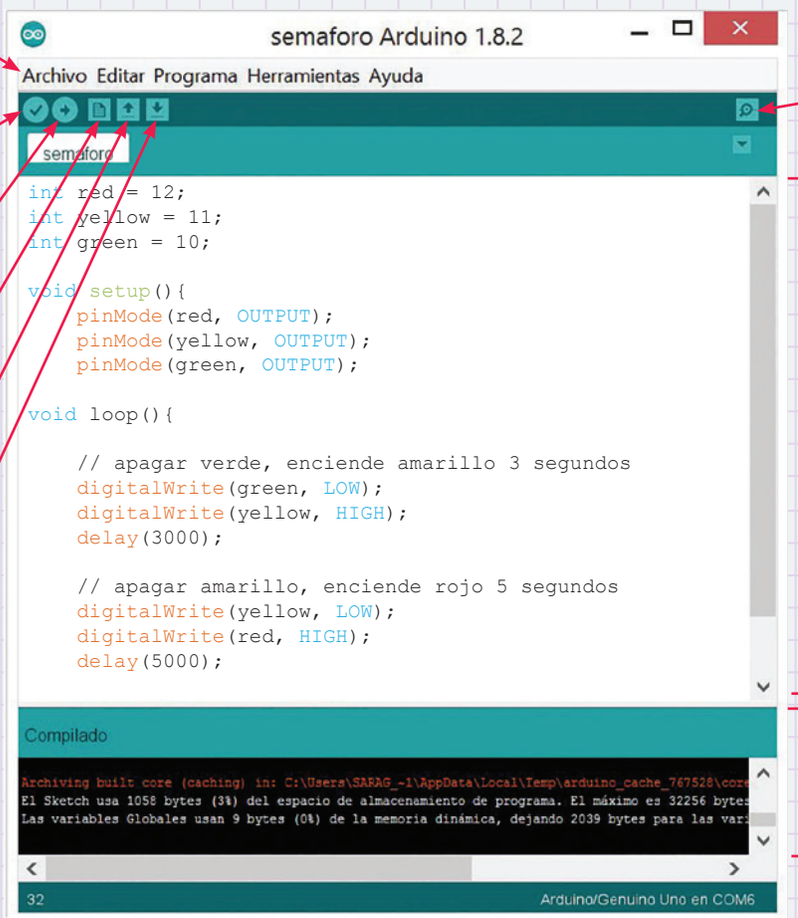
Botón *Verificar*. Comprueba si existe algún error de sintaxis.

Botón *Subir*. Al pulsarlo, el programa se compilará y seguidamente se cargará en la placa.

Botón *Nuevo*. Para comenzar un nuevo programa.

Botón *Abrir*. Para abrir un programa guardado anteriormente.

Botón *Salvar*. Para salvar el programa con el que se está trabajando.



Botón *Puerto serie*. Abrirá una ventana que mostrará datos recibidos de la placa.

Área del *Editor de texto*. Con fondo blanco, es el espacio donde se escriben las instrucciones que dan forma al programa.

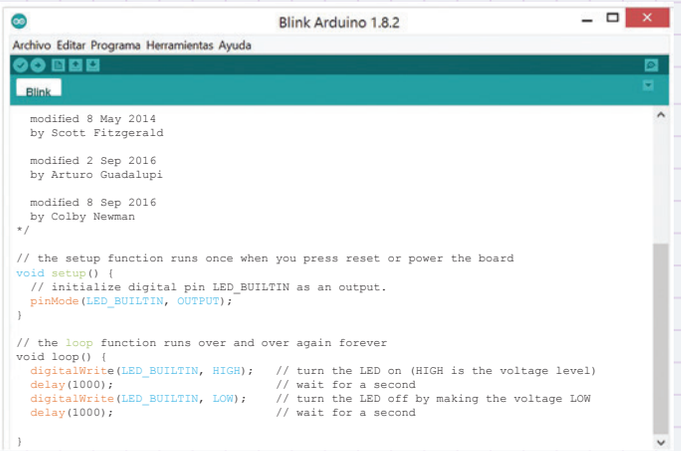
Área de *Consola*. Con fondo negro, es el espacio que el programa utiliza para mostrar mensajes que informan de si el programa está bien escrito o hay algún error, de si la placa está bien conectada, etc.

Actividades

7> Analiza los símbolos utilizados en la redacción del código que se muestra en la imagen de esta página (punto y coma, paréntesis, barra diagonal, llave…) y describe cuál crees que es su cometido y sus características de sintaxis. De la misma manera, se puede apreciar que el programa asigna diferentes colores a palabras específicas. Investiga en Internet y explica el uso de este código de colores.

Práctica 2. Compilar y cargar un programa en la placa

Para compilar y cargar el primer programa, abriremos uno ya escrito desde la base de datos de ejemplos del propio IDE de Arduino.

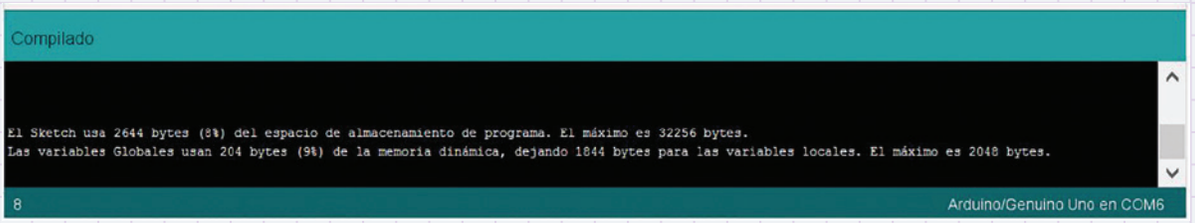


1. Accede a *Archivo* > *Ejemplos* > *0.1 Basics* > *Blink*

   Se cargará un programa que hace parpadear un led.

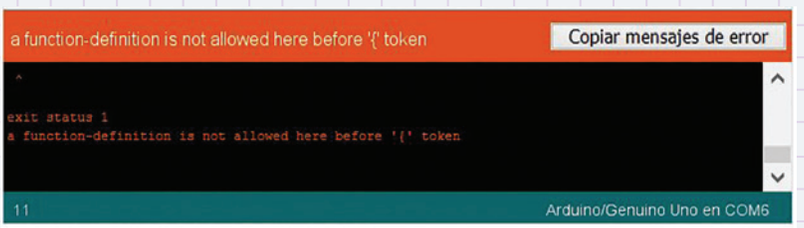
2. Pulsa el botón *Verificar.*

   Una barra de progreso de color verde avanza mientras el IDE comprueba las instrucciones. Como el programa no tiene ningún error de sintaxis, aparecerá en la consola un mensaje en texto blanco que indicará que todo está bien y la palabra «Compilado» en la cabecera de la consola.



Programa correctamente compilado.

En caso de que algo esté mal escrito, el texto del mensaje de la consola aparecerá en naranja y el cursor se colocará en el punto donde se ha detectado el error.

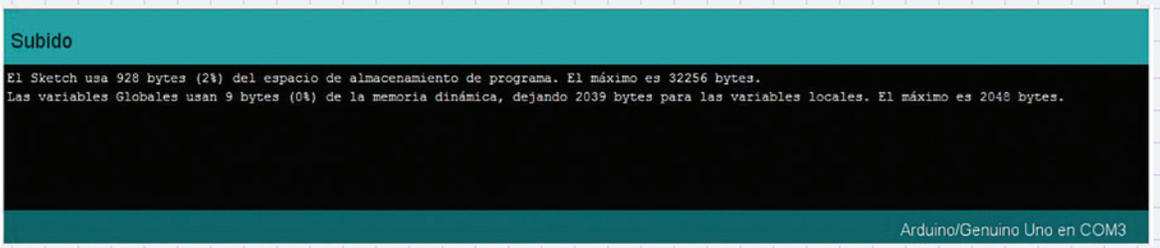


Programa con error de sintaxis.

3. Pulsa el botón *Subir.*

   Una barra de progreso indica que el programa se está cargando en la placa. Si la placa y el puerto están bien configurados, aparecerá el mensaje «Subido» en la cabecera de la consola y el led integrado en la placa Arduino (señalado con una L mayúscula) parpadeará con una frecuencia de un segundo.

   El led se enciende durante un segundo y se mantiene apagado durante otro segundo porque entre los paréntesis de la función delay(1000) está escrito el valor 1000, que son 1000 milisegundos, es decir, un segundo. Si cambias este valor a 2000, por ejemplo, los tiempos de encendido y apagado cambian a dos segundos.



Programa correctamente cargado a la placa.

Práctica 3. El semáforo para coches

En esta práctica diseñaremos una secuencia de ledes. Para ello, analizaremos el programa «Button» al que podemos acceder desde *Archivo > Ejemplos > 0.2 Digital > Button.*

Aprovecharemos este programa para diferenciar las tres partes principales de que consta un programa en el IDE Arduino: cabecera, setup y loop.

1. En la **cabecera** se definen las constantes y variables que requiere el programa.
2. La función de configuración **void setup** contiene las instrucciones de configuración de la placa, como, por ejemplo, qué pin es de entrada o de salida, a qué velocidad funcionará el puerto serie, etc. Es la primera función en ejecutarse y solo lo hace una vez.
3. La función **bucle,** o **void loop,** contiene el código que se ejecutará cíclicamente e infinitas veces (lectura de entradas, activación de salidas, etc.). Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

Cabecera / Setup / Loop

```
const int buttonPin = 2;     // Asignamos el pin2 al pulsador.
const int ledPin =  13;      // Asignamos el pin13 al LED.
int buttonState = 0;         // Declaramos la variable "buttonState"
                             // que leerá el estado del pulsador.

void setup() {
  pinMode(ledPin, OUTPUT);
  // Inicializamos el pin del LED como salida.
  pinMode(buttonPin, INPUT);
}

void loop() {
  // Leemos el estado de valor del pulsador:
  buttonState = digitalRead(buttonPin);

  // Comprueba si el pulsador está siendo activado.
  // Si es así, el valor de buttonState es HIGH:
  if (buttonState == HIGH) {
    // Enciende el LED:
    digitalWrite(ledPin, HIGH);
  } else {
    // sino es así apaga el LED:
    digitalWrite(ledPin, LOW);
  }
}
```

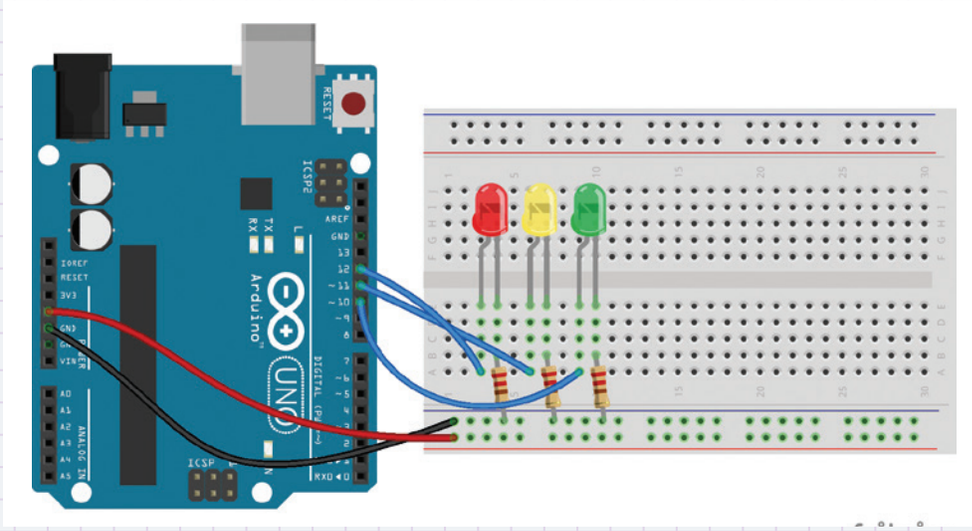
Otras consideraciones importantes a las que prestar atención a la hora de escribir el programa son las siguientes:

- Cada función delimita su contenido con dos llaves, una de apertura y otra de cierre «{ }». Si falta una de ellas, al compilar se recibe un error de sintaxis.
- Después de declarar una variable o definir una acción se añade un punto y coma «;».
- Si se quiere añadir un texto que sirva de guía, pero que no forma parte del código, este deberá estar precedido de dos barras inclinadas «//».

Crea un programa que simule la progresión de luces de un semáforo para coches. Se necesitarán tres ledes: uno de color rojo, otro amarillo y uno verde, además de tres resistencias de 220 Ω.

Realiza las siguientes conexiones con la placa desconectada del ordenador:

1. Alimentaremos el circuito con dos cables: uno rojo, que irá desde el pin de 5 V a la hilera inferior de la protoboard, y otro negro, que irá del pin GND (*ground,* o tierra) a la hilera superior.
2. Conectaremos con *jumpers* o cables los tres ledes con la pata más largahacia la izquierda y, en la misma columna de pines que esta pata, tres cables dirigidos a los pines digitales 10, 11 y 12.
3. Conectaremos con las tres resistencias la pata más corta de los ledes a la hilera superior de la banda de alimentación de la protoboard.



Actividad resuelta

Abre el IDE de Arduino e intenta diseñar el código para hacer funcionar este circuito fijándote en las funciones utilizadas en el ejemplo «Button». Cuando hayas terminado, verifica que todo esté correcto compilándolo y, si es así, conecta el cable USB a la placa y carga el programa en ella.

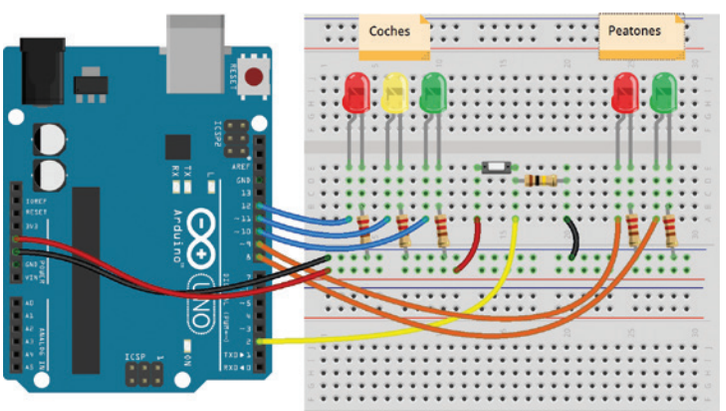
Solución

```
int red = 12;
int yellow = 11;
int green = 10;

void setup(){
    pinMode(red, OUTPUT);
    pinMode(yellow, OUTPUT);
    pinMode(green, OUTPUT);
}

void loop(){

    // apagar verde, enciende amarillo 3 segundos.
    digitalWrite(green, LOW);
    digitalWrite(yellow, HIGH);
    delay(3000);

    // apagar amarillo, enciende rojo 5 segundos.
    digitalWrite(yellow, LOW);
    digitalWrite(red, HIGH);
    delay(5000);

    // rojo y amarillo se enciende a la vez durante 2 segundos.
    digitalWrite(yellow, HIGH);
    delay(2000);

    // apaga rojo y amarillo, enciende verde durante 3 segundos.
    digitalWrite(yellow, LOW);
    digitalWrite(red, LOW);
    digitalWrite(green, HIGH);
    delay(3000);
}
```

Actividades

8> Rediseña el circuito y el programa del semáforo añadiendo dos ledes que representen un semáforo para peatones. Si quieres hacerlo todavía más preciso y real, haz que el led verde de peatones parpadee antes de pasar a rojo. Es tan solo una cuestión de jugar con los tiempos. Analiza el programa que has creado y dibuja el diagrama de flujo que correspondería utilizando la simbología adecuada.

9> Escribe el código necesario para que una fila de cuatro ledes se encienda y apague de manera consecutiva como si de un ecualizador de sonido se tratara.



Práctica 4. El semáforo para coches y peatones con S4A

S4A (Scratch for Arduino) es un «intérprete», es decir, que no es necesario cargar el programa en la placa cada vez que introducimos una modificación, sino que se actualiza y ejecuta constantemente, siempre que el ordenador esté conectado a la placa. Además, utiliza bloques para el diseño del programa, lo que facilita mucho su manejo.

Puedes descargar el programa desde su página oficial en: http://s4a.cat/index_es.html

Para trabajar con S4A es necesario instalar un *firmware* en nuestra placa, que es el programa que hará posible la conexión permanente entre la placa y el programa S4A.

Instalación de firmware para S4A

Sigue los pasos que a continuación indicamos:

1. Descarga el firmware «S4AFirmware16.ino» de S4A desde la sección *Descargas* de la web oficial. Fíjate en qué carpeta se descarga, pues vas a necesitar acceder a ella en el paso siguiente.

   NOTA: En caso de que al pulsar en la palabra «aquí» el firmware no se descargue automáticamente, haz clic derecho con tu ratón sobre la palabra y selecciona *Guardar como.*

2. Desde el IDE de Arduino ve *a Archivo > Abrir* y abre el firmware. Aparecerá entonces un mensaje advirtiendo que la carpeta de instalación se va crear y mover. Acepta clicando en «OK».
3. Asegúrate de que tu placa está correctamente conectada y pulsa el botón *Subir* del IDE de Arduino para cargar el firmware en la placa.
4. Inicia el programa S4A. Puede que aparezca el mensaje «Buscando placa», pero debería desaparecer en unos segundos. A partir de ese momento ya puedes empezar a programar.

La apariencia de S4A es muy similar a la de Scratch, ya que es una modificación suya. Su manejo es muy intuitivo: tan solo tienes que seleccionar las funciones y variables que vayas a utilizar y arrastrarlas al área central donde darás forma al programa.

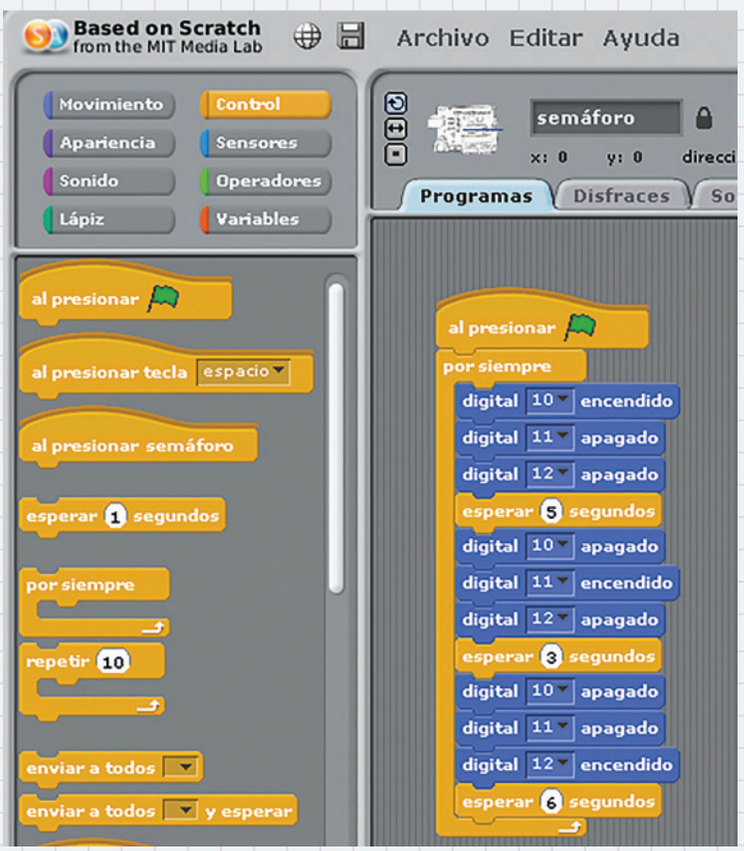
En la imagen puedes ver la estructura básica del programa que haría funcionar el semáforo para coches.

Actividad

Amplía la capacidad de este programa añadiendo las funciones necesarias para hacer funcionar un semáforo para coches y otro para peatones de manera coordinada.

Las funciones que necesitarás para ello son estas:

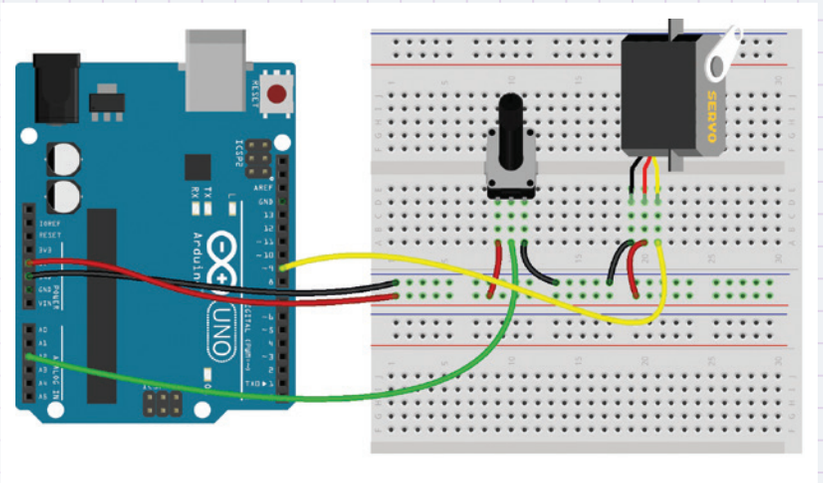
- **al presionar,** para iniciar la ejecución del programa,
- **por siempre**, para que lo contenido en él se repita constantemente,
- **esperar *n* segundos,** para controlar los tiempos. Es similar a la función **delay** del IDE de Arduino.
- **digital *x* apagado/encendido,** para apagar y encender los ledes conectados a los diferentes pines. *x* es el número de pin cuyo elemento conectado queremos encender o apagar.



Práctica 5. Abrir y cerrar una compuerta con un potenciómetro

Un *potenciómetro* es una resistencia variable. Para cambiar la resistencia hay que girar el mando del que viene provisto. Este elemento debe conectarse a una entrada analógica. Las lecturas de las entradas analógicas dan valores entre 0 y 1023, donde el valor 0 corresponde a una lectura de 0 V y el valor 1023, a una lectura de 5 V. Así pues, una lectura de 512 corresponde a un voltaje de 2,5 V.

En esta práctica transformaremos la señal variable del potenciómetro en un número de grados para hacer girar la compuerta de manera solidaria al giro del mando del potenciómetro. La compuerta obtendrá el movimiento de un servomotor.



Estos son los pasos que debes seguir para escribir el programa:

1. Incluir la librería para el servomotor. Puedes seguir los pasos explicados en el apartado 3.4.
2. Definir tres variables de tipo *integer:*
   - Una almacenará la posición del potenciómetro cuyos valores van de 0 a 1023. Puedes llamarla «**pot**».
   - En otra, que llamamos «**val**», almacenamos el valor de la entrada analógica.
   - En la tercera, que denominamos «**ang**», almacenamos el valor de lectura extrapolando la posición del potenciómetro a grados, es decir, de 0 a 180°.
3. En el **void setup** indicamos que el servo está conectado al pin 9 escribiendo: Servouno.attach(9);
4. En el **void loop** el programa leerá la posición del potenciómetro con: val = analogRead(pot);
5. Asignamos a la variable *val* la lectura y la traducimos a grados con la función *map*: val = map(val, 0, 1023, 0, 180);
6. Igualamos el valor resultante a la variable *ang* para que el servomotor tome la posición correspondiente con: Servouno. write(ang);
7. Para que el servo tenga tiempo para posicionarse y funcione sin saltos, añadiremos un pequeño tiempo de espera al final del proceso con la función: delay(15);

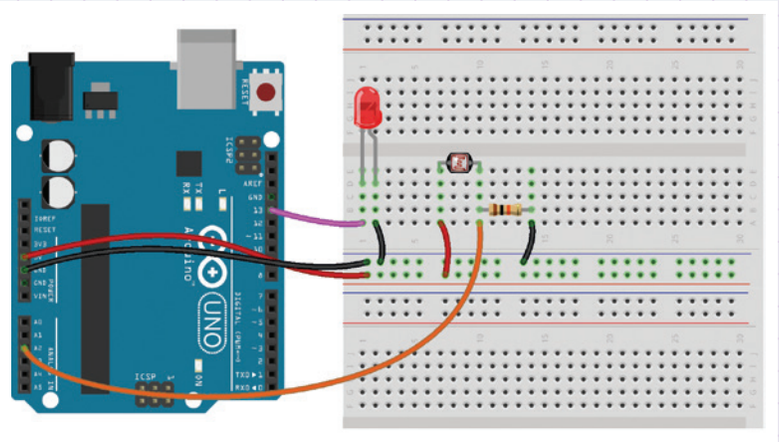
Solución

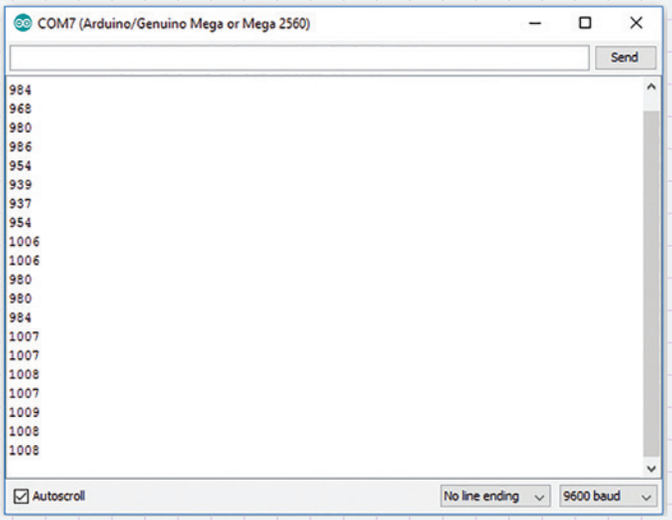
```
#include <Servo.h>

Servo Servouno; //Declaramos que vamos a controlar un servo y lo llamamos Servouno.

int pot = 0;  //Asignamos la variable para el potenciómetro.
int val;      //Variable para almacenar la entrada análoga.
int ang;      //Variable para almacenar valor de ángulo.

void setup()
{
Servouno.attach(9); //Definimos que el servo irá conectado al pin 9.
}

void loop()
{
val = analogRead(pot);            //Lee el valor del potenciómetro (entre 0 y 1023).
ang = map(val, 0, 1023, 0, 180);  //Traduce la lectura análoga (0, 1023) a grados
                                  //(0°, 180°) y asigna el valor a la variable ángulo.
Servouno.write(ang);              //Lleva el servo a la posición definida.
delay(15);                        //Espera un pequeño tiempo antes de volver a leer el valor.
}
```

Práctica 6. Luz nocturna

En esta práctica vamos a construir un circuito que encenderá una luz solo cuando sea de noche y, para ello, se requiere un led, un LDR o sensor de luz y una resistencia de 10 kΩ.

El circuito debe montarse como se muestra en la figura. Fíjate en que el LDR esté en serie con la resistencia de 10 kΩ.





Ventana del monitor serie con los valores recibidos por el sensor LDR.

Un LDR *(Light Dependent Resistor)* es una resistencia cuyo valor es inversamente proporcional a la cantidad de luz que incide sobre su superficie. Es decir, cuanto mayor sea la intensidad de la luz, menor será su resistencia y cuanta menos luz incida, mayor será su resistencia.

En esta práctica también se va a utilizar la función **Serial.println,** que permite visualizar la cantidad de luz que está recibiendo el LDR y mostrarlo en pantalla. Recuerda que para visualizar el monitor serie en el IDE de Arduino que mostrará los valores recibidos del LDR se debe pulsar el botón *Puerto serie.*

Veamos qué debe contener el programa correspondiente.

1. Define tres variables, *led, LDR y valorLDR,* e indica a qué pin estará conectado cada elemento.
2. En el **void setup,** define si los pines son de entrada o salida. Indica que utilizarás el monitor serie e inícialo escribiendo Serial.begin(9600).
3. En el **void loop** redactamos las siguientes acciones:
   - Apagar el led con la función **digitalWrite** (elemento, estado HIGH/LOW).
   - Leer el pin del LDR y asignar el valor recibido a la variable *valorLDR.*
   - Mostrar en el monitor serie el valorLDR: Serial.println(valorLDR).

   Si quieres que en la pantalla del monitor serie el valor leído aparezca precedido del texto "ValorLDR:" o cualquier otro texto, añade la función Serial.print("texto que desees mostrar").
   - Escribir un condicional que encienda el led solo cuando el sensor de luz reciba un valor superior a 900, valor que corresponde a una cantidad de luz baja.

Solución

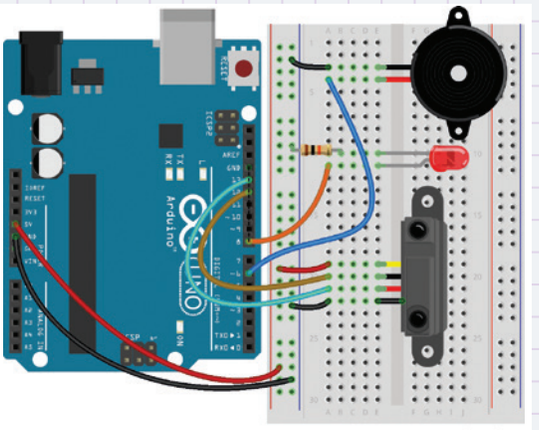
```
int led = 13;
int LDR = A2;
int valorLDR=0;

void setup() {

  pinMode(led, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  digitalWrite(led, LOW);
  valorLDR= analogRead(LDR);
  Serial.print("valorLDR:");
  Serial.println(valorLDR);

  if (valorLDR>900){
    digitalWrite(led, HIGH);
  }
  else{
        digitalWrite(led, LOW);
  }
}
```

Práctica 7. Sistema de ayuda al aparcamiento

En esta práctica reproduciremos lo que ocurre en los coches provistos de un sistema de ayuda al aparcamiento.

Para su montaje, se requiere de un sensor de ultrasonidos, un zumbador y un led. Dispondremos estos elementos en la placa protoboard como se muestra en la figura. Pon especial atención al conectar los terminales del sensor de ultrasonidos. Fíjate en que cada uno de los pines tiene un cometido distinto que viene indicado de la manera siguiente: VCC, TRIG, ECHO y GND.

Se utiliza el sensor de ultrasonidos para calcular la distancia existente entre el sensor y un objeto. Para conseguirlo, el sensor envía una onda ultrasónica a través del disparador o *Trigger* que rebota en el objeto e incide en el receptor del sensor *(Echo).* Se calcula la distancia según el tiempo que ha tardado en volver la onda. Es un cálculo sencillo que se integra en el programa.



Para construir el programa seguiremos los siguientes pasos:

1. Define los pines a los que vas a conectar el led, el zumbador y los dos terminales del sensor ultrasónico (Trigger y Echo).
2. Declara tres variables tipo *integer* para calcular la distancia:
   - *tiemporesp:* almacena el tiempo que tarda el sonido en rebotar.
   - *distancia:* contiene el resultado del cálculo de la distancia correspondiente al tiempo de respuesta.
   - *pausa:* su valor es proporcional a la distancia. Se emplea para que el zumbador emita pitidos y el led parpadee con mayor o menor frecuencia en función de la distancia.
3. En el void setup inicia el monitor serie y define si los sensores son de entrada o salida. Como el Trigger es un emisor, este será de salida, mientras que el Echo, al ser un receptor, será de entrada.
4. En el void loop, emite un sonido desde Trigger, esperamos dos microsegundos y recibiremos el sonido rebotado midiendo el tiempo que transcurre con las funciones siguientes: digitalWrite(trigPin, LOW); delayMicroseconds(2); digitalWrite(trigPin, HIGH); digitalWrite(trigPin, LOW); delayMicroseconds(10);
5. Como la velocidad del sonido es de 343 m/s, se requiere 1/343 = 0,00291 segundos para recorrer un metro, que, expresado en microsegundos por centímetro, son 29,1 μs/cm. La distancia a la que encuentra el objeto es la mitad de este valor porque el *tiemporesp* mide el tiempo que tarda el pulso en ir y volver.
6. Escribe las funciones condicionales necesarias para que el programa haga lo siguiente:
   - Si la distancia a la que se encuentra el objeto es mayor o igual que 50 cm (alcance máximo del sensor) o menor o igual que 1 cm, en el monitor serie se mostrará «Objeto fuera de alcance». En caso contrario, el monitor serie mostrará la distancia a la que se encuentra el objeto.
   - Si la distancia a la que se encuentra el objeto es menor que 10 cm, el zumbador emitirá pitidos y el led parpadeará. La frecuencia a la que ambos funcionarán será proporcional a la distancia, de manera que pausa = distancia × 100.

Solución

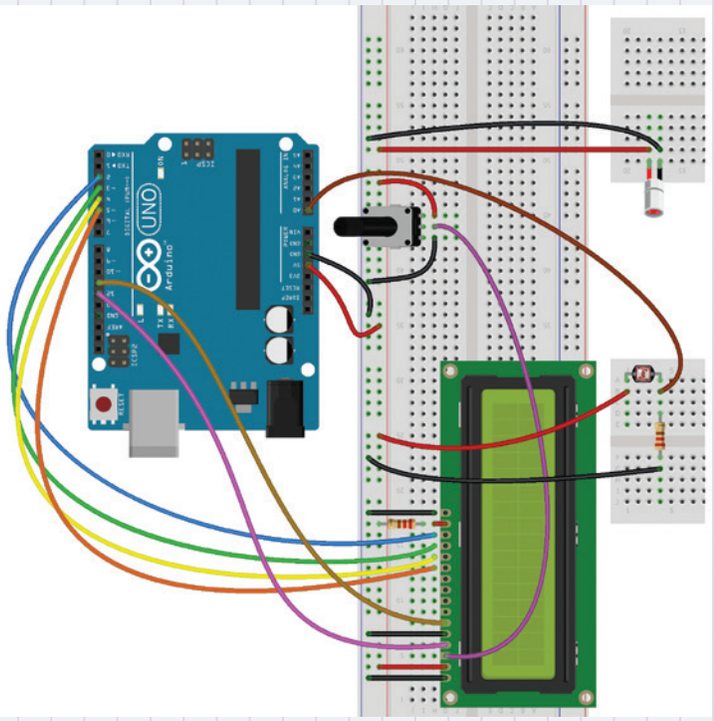
```
int trigPin = 12;
int echoPin = 13;
int zumbador = 6;
int led = 8;
int tiemporesp, distancia, pausa;


void setup() {
Serial.begin(9600);
pinMode(trigPin,OUTPUT);
pinMode(echoPin,INPUT);
pinMode(zumbador, OUTPUT);
pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  tiemporesp = pulseIn(echoPin,HIGH);
  distancia = tiemporesp / 2 / 29.1 ;

  if(distancia >= 50 || distancia <= 1){

  }
  else {
    Serial.print(distancia);
    Serial.println(" cm");
  }
  if(distancia < 10){
    pausa = distancia * 100;
    digitalWrite(zumbador, HIGH);
    digitalWrite(led, HIGH);
    delay(pausa);
    digitalWrite(zumbador, LOW);
    digitalWrite(led, LOW);
    delay(pausa);
  }
}
```

Práctica 8. Contador con sensor LDR y pantalla LCD

En esta práctica vamos a diseñar un sistema que cuente las personas que han entrado en una habitación. Para ello se necesita un diodo láser emisor de luz, un sensor de luz LDR, que recibirá la luz del láser, y una pantalla LCD, que mostrará la cantidad de veces que un objeto ha atravesado el haz de luz de la barrera.

El diodo láser y el sensor LDR deberán estar enfrentados, de modo que la luz del láser incida directamente sobre el LDR. El LDR tomará un valor de resistencia cuando reciba luz y, cada vez que este haz de luz sea interrumpido por algún elemento, el valor de su resistencia variará. Esta variación será detectada por el sistema y sumará una unidad al valor almacenado y mostrado en la pantalla LCD.

En la imagen de la placa puedes ver cómo conectar los diferentes elementos.



Para diseñar el programa puedes seguir estos pasos:

1. Añade la librería que permitirá utilizar las funciones propias de la pantalla LCD: #include <LiquidCrystal.h>
2. Define dos variables, *contador* y *numaux,* que serán números enteros y servirán para almacenar los valores necesarios para contar el número de veces que un objeto atraviesa el haz de luz.

   Define otras dos variables: *sensorLDR,* que recibirá la información del LDR, y *valorLDR,* que almacenará el valor de LDR.
3. Indica a qué entradas están conectados los pines de la pantalla LCD.
4. Para escribir el bloque setup investiga un poco y consigue que la pantalla se inicie y muestre durante dos segundos el texto «Hola, Fulanito» (siendo «Fulanito» tu nombre). Para ello puedes utilizar las siguientes funciones:

   lcd.begin (n.º de caracteres, n.º de filas), lcd.setCursor (x, y), lcd.print ("texto"), delay y lcd.clear.
5. Finalmente, en el bloque void loop, escribe las funciones necesarias para que cada vez que el valor recibido por la LDR sea menor que 600, es decir, que el haz de luz se haya interrumpido, la variable *contador* sume una unidad más y se muestre en la pantalla LCD.

Solución

```
#include <LiquidCrystal.h> // Incluiye libreria

//Declaramos variables
int contador=0;
int numaux=0;
int sensorLDR = A0;
int valorLDR = 0;

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
//Declara los pines que conectan la pantalla.

void setup() {
  lcd.begin(16, 2); //Inicia la pantalla.
  lcd.setCursor(5, 0);  //Sitúa cursor en origen.
  lcd.print("Hola, Fulanito"); //Escribe el texto.
  delay (2000);   //Espera 2 segundos.
  lcd.clear();    //Limpia la pantalla.
}

void loop() {

  valorLDR = analogRead(sensorLDR);
  // Lee.
  lcd.setCursor(0, 0);    //Posiciona en (0,0).
  lcd.print("Contador");  //Escribe.
  lcd.setCursor(10, 1);   //Posiciona en (10,1).
  lcd.print(contador);
  //Muestra en pantalla la variable contador.

  numaux=0;  //Hace la variable auxnumero=0
    while(valorLDR<600){  //Entramos en el bucle
    valorLDR = analogRead(sensorLDR);  // Lee
    if(numaux==0){  // Entra en el if si numaux=0.
    contador++;     //Aumentamos en una unidad
    numaux=1;       //Para que solo aumente
      }
    }
}
```

Práctica 9. Programación orientada a objetos con Processing

La **programación orientada a objetos** se basa en la idea de crear unidades independientes que contengan los valores y datos que van a manejar y las funciones con las que manipularlos.

Para comprender cómo funciona este tipo de programación, deben definirse los siguientes conceptos:

- **Objetos y clases**.

  Los objetos son las entidades con que se trabaja y la clase *(class)* la definición de esos objetos. Podemos imaginar la clase como un molde y los objetos, como cada copia particular generada mediante dicho molde.

  Para la definición de un objeto simple se utiliza, en primer lugar, el nombre de la clase y, a continuación, las variables internas o «propiedades» del objeto. Para crear copias del objeto, se utiliza una función inicial, que tendrá el nombre de la clase, y se asignarán los valores iniciales a sus propiedades.

```
class Ball { // Definimos la clase a la que llamaremos Ball.
int x,y;      // Definimos los dos parámetros x e y de la clase Ball.

Ball () {     // Creamos una copia del objeto Ball y damos valor a x e y.
x=10;
y=20;
}
}
```

- **Métodos:** son las acciones que pueden realizar los objetos, como, por ejemplo, dibujar formas, realizar cálculos con los valores que los definen, trasladarse por la pantalla, etc.

Una de las ventajas que ofrece Processing al trabajar con programación orientada a objetos es que se pueden utilizar matrices. Su uso permite crear copias de un objeto de manera sencilla, cada uno con cualidades diferentes: posición, tamaño, número de patas, velocidad, color, etc.

Ejemplo 1

En el código siguiente se muestra un ejemplo diseñado para dibujar 15 bolas diferentes que se comportarán de manera diversa.

```
Ball A [];
int numBalls;
void setup () {
size (200,200);
smooth();
numBalls=15;
A= new Ball[numBalls];
for (int i = 0; i<numBalls ; i++){
A[i] = new Ball( int(random(100) ) , int( random(100) ) );
}
}
void draw () {
fill(0,0,100,1);
rect(0,0,width,height);
for (int i = 0; i<numBalls ; i++){
A[i].calcula(); // o avanza, o...
A[i].dibuja();
}
}
```

Ejemplo 2

En este ejemplo se expone el código para dibujar una elipse con la función «dibuja» y otro que hará moverse esa elipse con la función «avanza».

```
class Ball {
int x,y;

Ball ()   {
x=10;
y=20;
}

void dibuja ()  {
ellipse(x,y,12,12);
}

void avanza ()   {
x = x+1;
}

}
```

En la web **www.processing.org** puedes encontrar muchos tutoriales y ejemplos de programas, así como descargar el programa de manera libre.

Autoevaluación

1. La herramienta de programación utilizada para traducir el código fuente al ordenador es:

a) El editor de texto.

b) El intérprete.

c) El depurador.

2. ¿Cuál de estos programas no permite programar por bloques?

a) Bitbloq.

b) IDE de Arduino.

c) A4A.

3. En un diagrama de flujo, los procesos en los que se debe tomar una decisión se representan con:

a) Un círculo.

b) Un rectángulo.

c) Un rombo.

4. ¿Cuál de estos elementos no se encuentra integrado en la placa Arduino Uno?

a) Microprocesador.

b) Led de encendido.

c) Sensor LDR.

5. ¿Cómo se llaman los elementos que permiten aumentar las características de una placa controladora?

a) Shields.

b) Actuadores.

c) Conectores USB.

6. ¿De qué manera se pueden alimentar la mayoría de las placas controladoras?

a) Vía cable USB.

b) Alimentación externa.

c) Ambos.

7. ¿Cuál es la corriente máxima de salida de cada uno de los pines de una placa Arduino Uno?

a) 20 mA.

b) 40 mA.

c) 200 mA.

8. Los pines capaces de enviar o recibir dos únicos valores correspondientes a 0 y 5 V son pines:

a) De potencia.

b) Analógicos.

c) Digitales.

9. Para montar un circuito con varios ledes conectados en serie con resistencias, se precisa de una:

a) Placa protoboard.

b) Entrada analógica.

c) Tarjeta de expansión.

10. ¿Cuál de estos sensores es sensible a la radiación de infrarrojos?

a) LDR.

b) PIR.

c) NTC.

11. ¿Cuál de estos elementos puede comportarse como sensor o actuador?

a) Sensor de sonidos.

b) Sensor de infrarrojos.

c) Acelerómetro.

12. ¿Cuál de estos actuadores requiere alimentación adicional para conectarse a una placa Arduino Uno?

a) Led.

b) Motor CC.

c) Servomotor.

13. ¿Qué tipo de dato puede almacenar el número -199?

a) Byte.

b) Long.

c) Booleano.

14. ¿Cuál de estas funciones es un bucle?

a) Serialprintln.

b) For.

c) If.

15. Cuál de estas estructuras de programación incluye de manera intrínseca el establecimiento de una condición?

a) Secuenciales.

b) Condicionales.

c) Repetitivas.

16. ¿Cuál de estos elementos no puede ser utilizado para abrir y cerrar una compuerta?

a) Potenciómetro

b) LDR.

c) LED.

Soluciones: 1b, 2b, 3c, 4c, 5a, 6c, 7b, 8c, 9a, 10b, 11a, 12b, 13b, 14b, 15b y 16c.

Actividades finales

Para repasar

1. Elabora una lista de las herramientas de programación estudiadas e indica su utilidad.

2. ¿Qué tipo de herramienta es S4A y por qué?

3. Explica el significado de los siguientes acrónimos: IDE, GUI, PIR, LDR, LED, NTC y PWM.

4. Dibuja una placa controladora e indica las partes más importantes y su utilidad.

5. ¿Incluye una placa Arduino Uno conexión Bluetooth? ¿Es posible dotar a una placa Arduino Uno de conexión Bluetooth? ¿Cómo?

6. ¿Cuántos pines digitales y analógicos incluye una placa Arduino Uno?

7. ¿Cuáles son los pines que permiten transmitir la salida de una señal analógica?

8. Enumera todos los elementos físicos necesarios para simular un semáforo de coches.

9. Haz una lista de los sensores y actuadores estudiados en esta unidad e indica su utilidad.

10. ¿Qué tipo de variable consideras más adecuada para definir los siguientes valores?

Valor	Tipo
–255	
201	
257	
1, 3, 5, 7, 9	
5.67	
True	
3100	
«f»	
sacapuntas	

11. ¿Qué es la programación estructurada y para qué sirve?

12. ¿Qué entendemos por estructuras de control?

13. ¿Qué elemento de control necesita utilizar un incrementador?

14. Explica con tus propias palabras qué contienen las secciones *cabecera, void setup* y *void loop* de un programa escrito en la IDE de Arduino.

15. ¿Cuál es la manera más sencilla de instalar una librería?

Para afianzar

16. Explica en qué se diferencian las señales digitales de las analógicas.

17. Indica a qué tipo de pin (digital o analógico) deben conectarse los terminales de los siguientes elementos y si serán de entrada (E), salida (S) o ambos: LED, LDR, sensor de proximidad ultrasónico, potenciómetro, servomotor y zumbador.

18. Indica la utilidad de las siguientes funciones:

- pinMode
- digitalRead
- digitalWrite
- delay
- map
- Serial.begin
- include
- analogRead
- Serial.print
- Serial.println
- define
- pulseIn

19. Escribe un programa capaz de contar el número de personas que entran y salen de una habitación, con una puerta de entrada y otra de salida, de manera que, al llegar a un máximo de 30, cierre la de entrada y permanezca cerrada hasta que el número de personas dentro de la habitación sea inferior o igual a 30.

20. Interpreta y describe qué es lo que ocurre en estos programas:

a)

```
const int potenciometroPin = A0;
const int ledPin = 13;
const int umbral = 400;

void setup() {
  pinMode(ledPin, OUTPUT);

  Serial.begin(9600);
}

void loop() {
  int analogValue = analogRead(analogPin);

if (analogValue > umbral) {
  digitalWrite(ledPin, HIGH);

}

else    {
  digitalWrite(ledPin, LOW);
}

Serial.println(analogValue);
delay(1);
```

b)

```
const int ledPin = 13;
const int knockSensor = A0;
const int threshold = 100;
int sensorReading = 0;
int ledState = LOW;

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

void loop() {
  sensorReading = analogRead(knockSensor);
  if (sensorReading >= threshold) {
    ledState = !ledState;
    digitalWrite(ledPin, ledState);
    Serial.println("Knock!");
  }
  delay(100);

 }
```

c)

```
#include <Servo.h>
#include <Arduino.h>
Servo barrapaso;

int ledverderd = 13;
int ledamarmar = 12;
int ledrojoj = 11;

void setup(){
pinMode(ledverderd, OUTPUT);
pinMode(ledamarmar, OUTPUT);
pinMode(ledrojoj, OUTPUT);
barrapaso.attach(9);
}

void loop(){
digitalWrite(ledroj, HIGH);
barrapaso.write(10);
delay (3000);
digitalWrite(ledroj, LOW);
digitalWrite(ledamar, HIGH);
delay(1000);
digitalWrite(ledamar, LOW);
delay(1000);
digitalWrite(ledamar, HIGH);
delay(1000);
digitalWrite(ledamar, LOW);
delay(1000);
digitalWrite(ledamar, HIGH);
delay(1000);
digitalWrite(ledamar, LOW);
delay(1000);
digitalWrite(ledverd, HIGH);
barrapaso.write(100);
delay(5000);
digitalWrite(ledverd, LOW);
}
```

Para profundizar

21. Existen muchos programas para simular circuitos; Fritzing, en particular, resulta muy adecuado para el diseño de circuitos con placas Arduino. Su manejo es muy intuitivo y puedes descargarlo en http://fritzing.org/download/.

A continuación, te proponemos que, para los proyectos propuestos, realices lo siguiente:

a) Dibuja el diagrama de flujo correspondiente al proceso propuesto.

b) Escribe una lista de los componentes electrónicos necesarios para construirlo.

c) Representa el circuito con todos sus componentes conectados utilizando el programa Fritzing.

d) Escribe el programa en el IDE de Arduino, cárgalo en la placa y comprueba si funciona.

1. Monitorización de temperatura y humedad.

En este proyecto debes utilizar la placa Arduino para mostrar en el monitor serie del IDE de Arduino la temperatura y la humedad del ambiente.

2. Compuerta que se abre al llamar tres veces.

Utiliza un actuador que sea capaz de abrir y cerrar una pequeña puerta, de manera que, cuando reciba un sonido tres veces seguidas, se abra, espere unos segundos y, después, se cierre. Si quieres completar esta práctica, puedes añadir un sensor que detecte si la persona u objeto ha atravesado la puerta para proceder a cerrarla.

3. Sistema de seguridad de puerta de frigorífico.

Reproduce un sistema automático que detecte si la puerta de un frigorífico está abierta o no, de manera que se encienda una luz cuando está abierta y que, si se mantiene, abierta más de 15 segundos, suene un pitido intermitente hasta cerrarse.

4. Pantalla LCD indicadora de grados.

Diseña un sistema que muestre en una pantalla LCD la posición en grados de un mando que gire en ambas direcciones. Un LCD *(Liquid Crystal Display)* es una pantalla formada por un número de píxeles de color o monocromos colocados delante de una fuente de luz o reflectora.

Será necesario importar una librería específica para la pantalla LCD. Recuerda que entre sus archivos podrás encontrar funciones específicas e instrucciones para utilizarlas. Las funciones que necesitarás son las siguientes: **lcd.begin** para inicializar el display, **lcd.setCursor** para posicionar el cursor en una coordenada determinada (lo lógico es comenzar en x = 0, y = 1) y **lcd.print** o **lcd.println** para mostrar en el display números enteros.