

ARDUINO



CREA TODO LO QUE TU
IMAGINACIÓN TE PERMITA

ÍNDICE:

0-Sobre el autor y licencia:	0
1-Introducción:.....	0
1.1 - ¿Qué es un microcontrolador?	0
1.2 -¿Qué es Arduino?	4
1.3 Software de desarrollo:.....	9
2-Lenguaje de programación:	14
2.1-.Lenguaje Arduino:.....	16
2.2-Estructura de un programa:	29
2.3-ArduBlock: lenguaje gráfico:	32
3-Ejemplos prácticos.....	34

0-Sobre el autor y licencia:

Isaac es un técnico en electrónica y domótica, escritor de algunos libros técnicos, y profesor de Linux, supercomputación y programación.

Más sobre su trabajo – <http://architecnologia.es/>

Este ebook ha sido registrado bajo licencia *Creative Commons Attribution 4.0*.



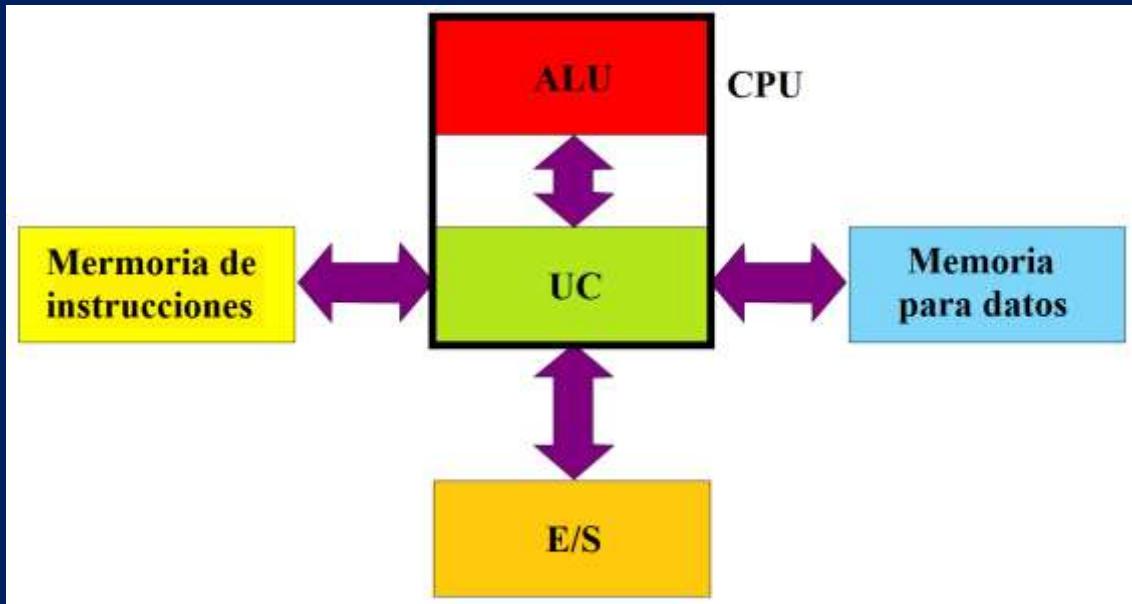
1-Introducción:

Este manual gratuito va dirigido a todos aquellos que quieran comenzar con la plataforma Arduino. Con él obtendréis los conocimientos básicos para empezar a realizar vuestros propios prototipos. El manual trata de ser lo más breve y claro posible para condensar cuanta más información mejor...

1.1 - ¿Qué es un microcontrolador?

Un microcontrolador (μ C), a veces llamado MCU (MicroController Unit), es básicamente un ordenador integrado en un solo chip. Y esto es cierto porque un microcontrolador tiene en su interior las tres unidades básicas de un ordenador, es decir, memoria, CPU o microprocesador y periféricos E/S. Evidentemente el rendimiento y capacidad

de uno de estos chips no se puede comparar al de un computador moderno.



La mayoría de los microcontroladores modernos no solo integran los elementos básicos mencionados anteriormente, sino que pueden llegar a tener muchas más partes: memoria RAM, memorias flash, temporizadores, decodificadores, conversores A/D y D/A, controladores (DMA, USB, Ethernet, PCI, etc.), UARTs (Universal Asynchronous Receiver-Transmitter), ...

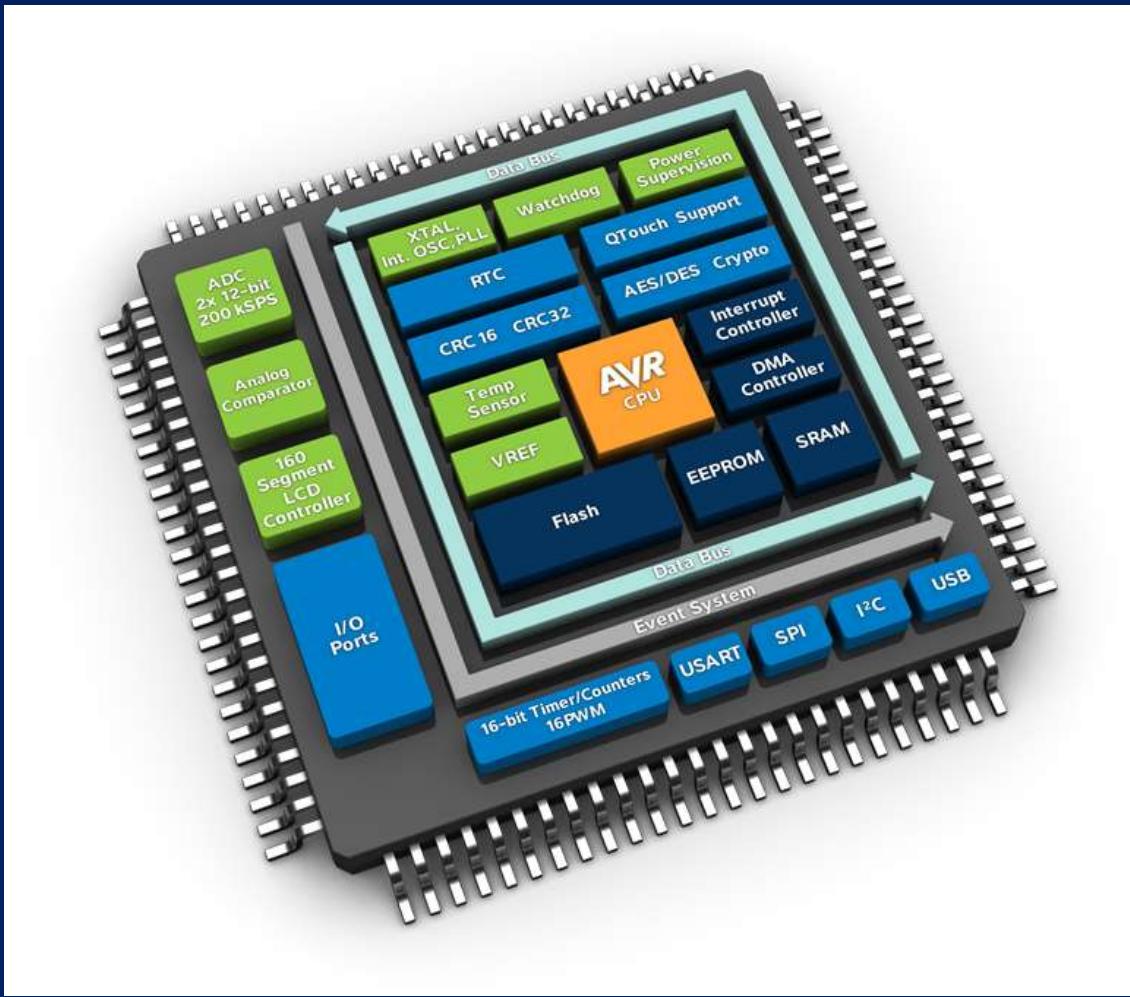
Un microcontrolador puede hacer funciones tan simples como hacer titilar una luz o manejar máquinas y robots complejos. Por su versatilidad están presentes en muchos aparatos cotidianos: ordenadores, electrodomésticos, TVs, reproductores multimedia, automóviles, dispositivos de red, etc., incluso en las máquinas y robots industriales.

Existen multitud de fabricantes y tipos de microcontroladores, algunos históricos como el primero de ellos, el TMS 1000 de Texas Instruments (creado en 1971 por Gary Boone y Michael Cochran). Otros tan populares como la

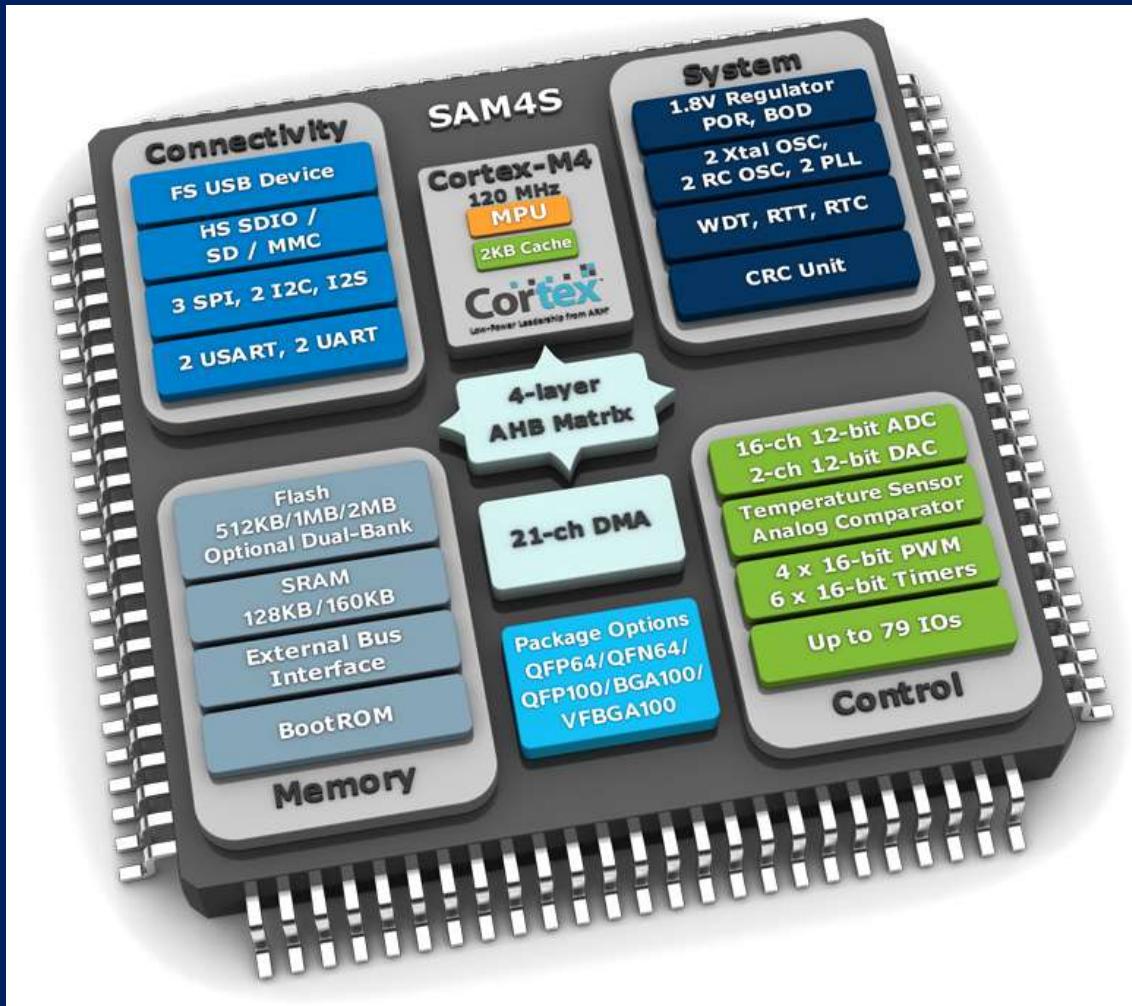
familia PIC de la compañía Microchip. Pero los que más nos interesan son los de la compañía ATmel, ya que son los que integra la plataforma Arduino.

ATmel es un fabricante de semiconductores fundado en 1984 y que comenzó a popularizar sus microcontroladores derivados del 8051 de Intel. ATmel también tiene sus propias arquitecturas de microcontroladores AVR y AVR32, e incluso crea micrонтroladores modernos basados en los populares microprocesadores ARM.

Según la versión de la placa Arduino, el tipo de microcontrolador puede variar, pero los más comunes son los ATmega basados en la arquitectura AVR RISC de 8 bits (un derivado de la arquitectura Harvard creado por estudiantes del Norwegian Institute of Technology y optimizada por ATmel). AVR posee una pipeline que los hace bastante rápidos comparados con otros competidores.



Pero Arduino también integra otras versiones de ATmel (SAM3x Series) que poseen un núcleo ARM Cortex-M3 más potente que los AVR. Los cores ARM Cortex-M son de 32 bits basados en la arquitectura RISC ARMv7-M desarrollada por ARM Holdings.

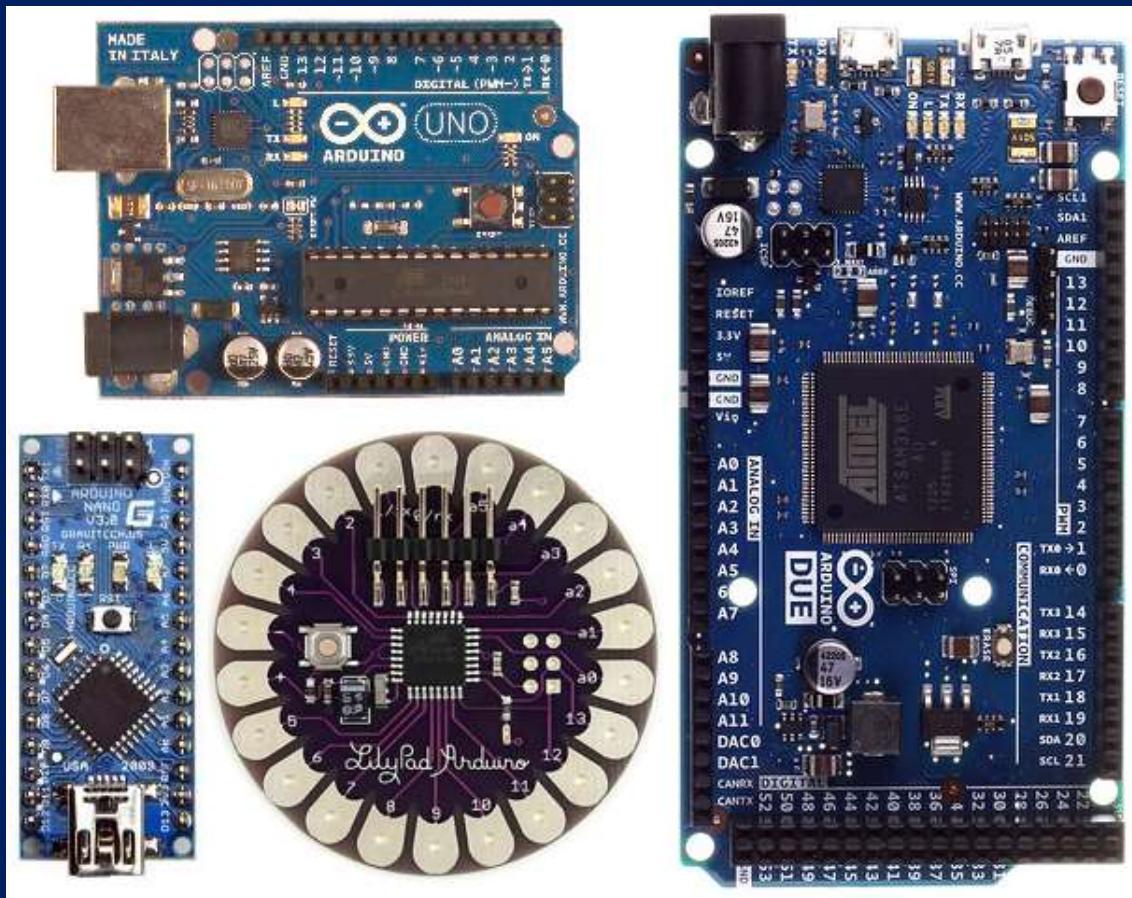


1.2 -¿Qué es Arduino?

Es una plataforma de hardware libre, porque sabrás que no solo existe el software libre, ¿no? Esta placa se basa en uno de los microcontroladores descritos anteriormente y junto con un entorno de desarrollo, que veremos en el punto siguiente, tienes todo lo necesario para crear tus proyectos electrónicos. Algo muy apropiado en la era del DIY (Do It Yourself) o “Hágalo usted mismo”.

***Nota:** para los más curiosos decir que Arduino es una palabra de origen germánico que significa “El que ayuda a los amigos”. No pienses que Arduino viene de “arduo”, porque ya verás que fácil es de emplear...

Como ya sabrán, en el mercado existen diversos productos. Unos son oficiales y otros no oficiales (compatibles de otros fabricantes, por ejemplo SainSmart, Brasuino, Freeduino,...). Lo que varía de uno a otro, a parte del precio, son las características del microcontrolador, el número de patillas digitales, entradas analógicas disponibles, cantidad de memoria,...



Algunas de las versiones oficiales de placas son: Leonardo, Uno, Due, Mega, Nano, LilyPad, Pro, Esplora, Micro, etc. Quizás yo destacaría Uno, Due, Mega y LilyPad.

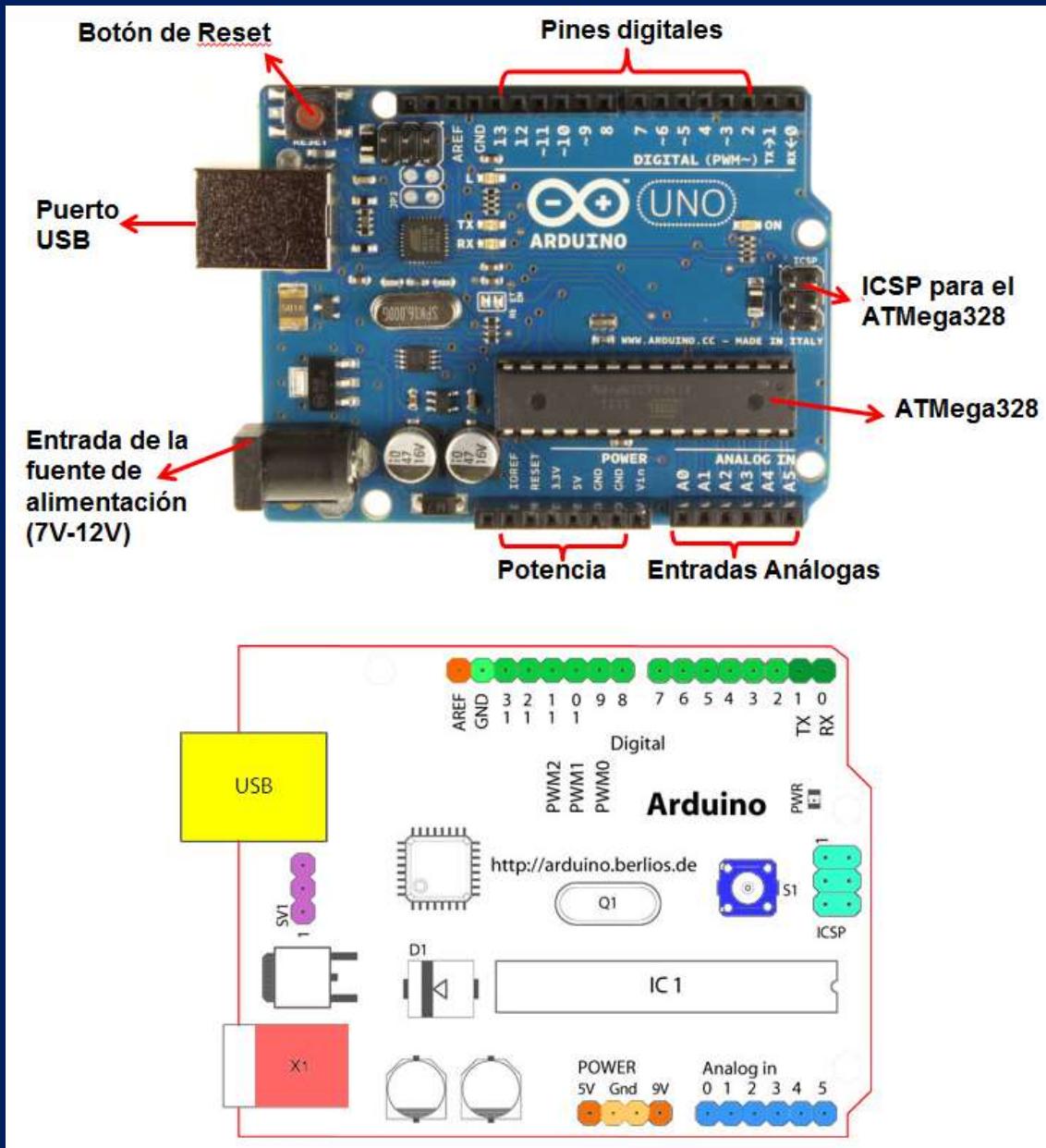
Modelo de placa Arduino	Características
Uno	Con un microcontrolador ATmega328P a 16Mhz, puerto USB, alimentación de 5v, SRAM de 2Kb, 32Kb de flash, 1Kb de

	EEPROM, 14 pines digitales y 6 con PWM, así como 6 salidas analógicas.
Mega	ATmega2560 a 16Mhz, 5v, 4Kb de EEPROM, 8Kb de SRAM y 256Kb de flash. Las entradas digitales son iguales al Due, solo que posee dos más con PWM y 16 analógicas.
Due	Microcontrolador AT91SAM3X8E basado en ARM a 84Mhz, alimentación de 3.3v, flash de 512Kb, SRAM de 96Kb y no contiene EEPROM. Las entradas digitales ascienden a 54, 12 con PWM y las analógicas son 12.
LilyPad	ATmega328V a 8Mhz, 5.5v, 16Kb de flash, 0.5 de EEPROM y 1 de SRAM. Las entradas digitales son 14, 6 con PWM y 6 analógicas. Lo más peculiar es su reducido tamaño y que la placa es flexible.

***Nota:** esta selección que he realizado no quiere decir que sea la mejor, eso lo decides tú, según las necesidades que tengas.

Por otro lado también existen los accesorios o “shields”, que son placas o componentes para complementar la placa Arduino. Entre los shields podemos encontrar dispositivos GPS, placas Ethernet, pantallas LCD, breadboard, tarjeta Arduino Basic I/O, cámaras, etc.

Para rematar este tema, podría describir cuales son las partes de una placa de Arduino, pero es mejor y más práctico verlo en la siguiente ilustración:



Para que funcione necesitamos la placa de Arduino, un cable USB para conectarla al PC, el software de desarrollo, una fuente de alimentación (también puede ser una pila que de el voltaje necesario, aunque si no se desconecta del PC no es necesario) y los periféricos, ya sean analógicos o digitales.

Evidentemente los periféricos digitales irán conectados a las E/S (entradas/salidas) digitales PWM (donde conectaremos los periféricos que se quieran gobernar, como pulsadores, leds, alarmas, motores, bombillas,...), mientras que los analógicos irán conectados a los pines analógicas. Por cierto, PWM es el acrónimo de Pulse Wave Modulation (modulación de onda por pulsos) y todas los pines PWM están marcados con un ~ delante de su numeración.

Fíjate en la imagen anterior, las conexiones POWER o Potencia, son muy importantes. Como dijimos anteriormente, la placa Arduino puede conectarse al USB de nuestro ordenador y alimentarse desde ahí, pero si la quieres utilizar sin el ordenador, puedes emplear una fuente de alimentación conectada a la toma destinada a ello o emplear pilas. Si lo haces mediante baterías, puedes conectar un borne de la pila a V_{in} , que es la entrada de voltaje para alimentar Arduino y el otro borne a GND (tierra). También he de decir, que si la placa se está alimentando por USB o por la toma de la fuente de alimentación, los pines V_{in} y GND pueden ser utilizados también para obtener corriente de ellos.

El resto de pines puede variar según la placa, pero normalmente pueden aportarnos voltajes para manejar nuestros componentes electrónicos utilizados en el proyecto (3v3, 5v,...). Otro pin interesante es AREF, que proporciona un voltaje de referencia para los pines analógicos. Rx es para recibir y Tx para transmitir datos a través de un puerto serie TTL. Reset es capaz de resetear la memoria del microcontrolador, SDA y SCL también están presentes en algunas placas para dar soporte al protocolo I2C/TWI, incluso podemos encontrarnos con placas que tengan conexiones para interrupciones, SPI, etc.

***Nota:** son necesarias unas nociones básicas de electrónica para realizar el montaje de los circuitos adecuadamente. Si eres inexperto, los ejemplos prácticos te aclararán mucho para futuros proyectos.

1.3 Software de desarrollo:

Para que un microcontrolador cumpla las funciones, tan solo hay que crear un programa y colocarlo en la memoria del microcontrolador para que comience a ejecutarlo. Para cerrar este código fuente se necesita un compilador y otros elementos que se integran en un IDE (Integrated Development Environment) o entorno de desarrollo integrado. El entorno de desarrollo integrado gratuito de la plataforma es Arduino IDE, compatible con Linux, Windows y Mac OS X.

También existen otros entornos o complementos que nos pueden ayudar en la tarea. Por ejemplo, para los que no se les de bien lo de aprender un lenguaje de programación, pueden utilizar ArduBlock, un complemento para emplear un lenguaje gráfico que utiliza bloques funcionales para programar Arduino.

***Nota:** existen alternativas a ArduBlock, como MiniBloq, ModKit, Amici, etc. Y otros programas como Fritzing para dibujar circuitos.

Otros prefieren otras herramientas como MyOpenLab para sus proyectos e incluso S4A (Scratch for Arduino). El primero es un entorno orientado al modelado y simulación de sistemas electrónicos, control y robótica. Mientras que S4A es muy

similar a ArduBlock, aunque quizás te interese más si tus proyectos están más orientados a robótica.

Yo aconsejo trabajar con Arduino IDE y, si queréis, instalar el complemento ArduBlock para trabajar de forma gráfica. Para su correcta instalación, descargamos la versión compatible con nuestro sistema en <http://arduino.cc/es/Main/Software>

-Si nuestro sistema es Windows, el archivo será un ejecutable .exe que se instalará fácilmente con unos cuantos clics.

-En el caso de Apple Mac OS X, el fichero tendrá la extensión .dmg típica de los entornos Macintosh. La instalación puede ser tan sencilla como abrir el fichero desde el Finder y arrastrar el ícono del programa al directorio “Aplicaciones” del disco duro. Si lo deseas, puedes utilizar un instalador.

-En el caso de Linux, el fichero a descargar es un tarball de tipo .tgz, que puedes desempaquetar en un directorio. Luego, desde la consola te colocas en dicho directorio con el comando “cd”, por ejemplo: “cd Descargas/arduino-0019” y luego puedes emplear el comando “./arduino” para ejecutarlo, sin necesidad de instalar. Esto te valdría para cualquier distribución, pero algunas como openSuSE o Ubuntu, tienen un paquete de Arduino IDE en sus repositorios, con lo cual puedes instalarlo fácilmente con “YaST” o el “Centro de Software”.

Una vez tengas instalado Arduino IDE, deberemos instalar ArduBlock. Como es un fichero .jar nos servirá en cualquier plataforma, pero previamente debemos tener instalado Java

para que funcione. El fichero de ArduBlock lo puedes descargar en <https://github.com/taweili/ardublock/downloads>

Luego abrimos Arduino IDE y nos dirigimos a “Archivo/Preferencias” y observamos cual es la ubicación de Sketchbook, según el sistema operativo seguimos estos pasos:

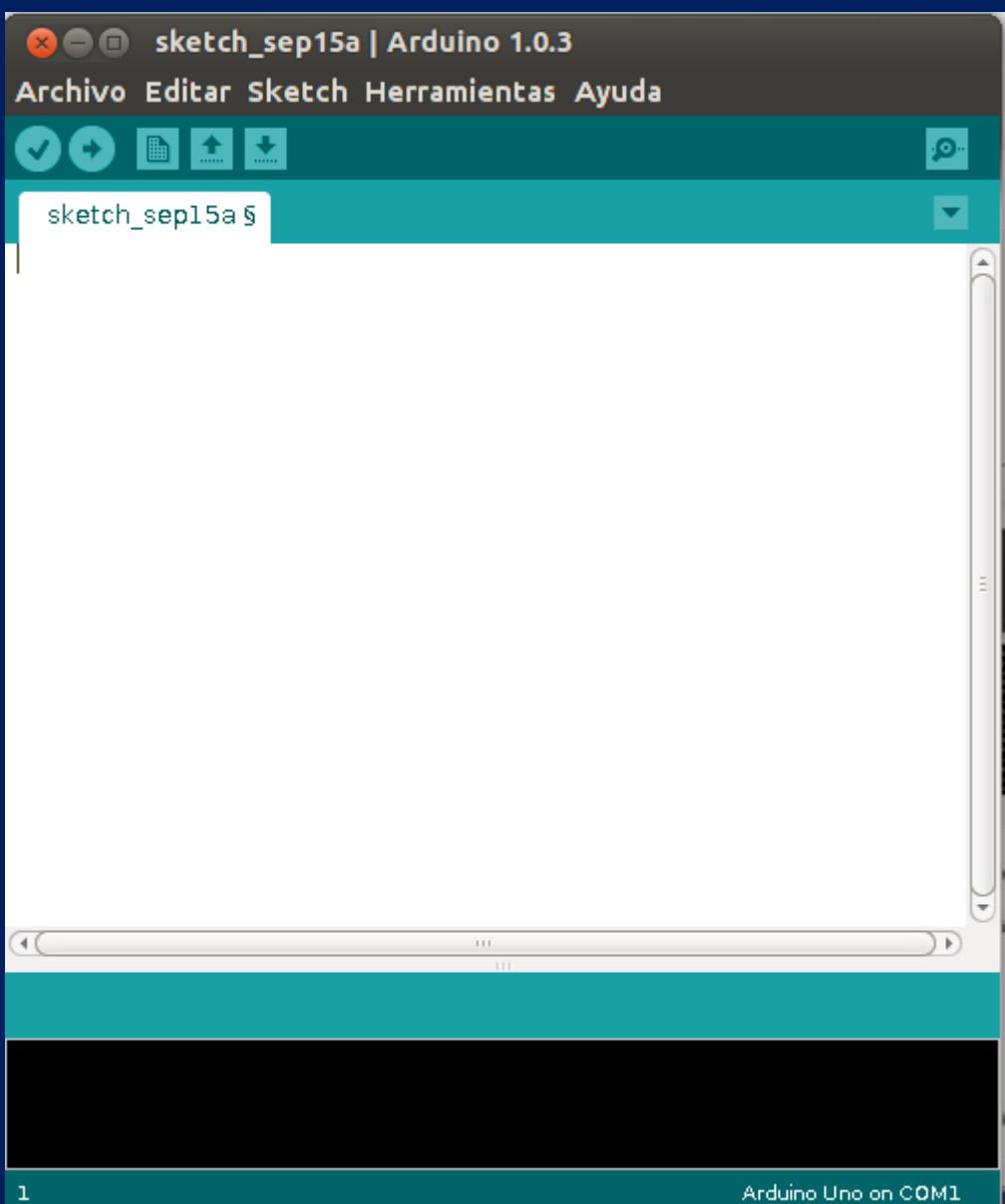
-Para agregarlo en Windows, deberás dirigirte a la dirección *C:\Users\nombre_usuario\Documents\Arduino* y creamos carpetas como una muñeca rusa, una dentro de otra, de tal manera que queden con esta jerarquía *\tools\ArduBlockTool\tool* y pegar allí el archivo .jar que nos hemos descargado.

-En el caso de Mac OS X, nos vamos a */Users/nombre_usuario/Documents/Arduino* y aquí deberemos crear un directorio llamado “*tools*”. Dentro de éste creamos otro llamado “*ArduBlockTool*” y dentro otro llamado “*tool*”. Recuerda que no debes poner las comillas y respetar las mayúsculas y minúsculas. Es decir, la dirección completa debe quedar así: */Users/nombre_usuario/Documents/Arduino/tools/ArduBlockTool/tool* y dentro pegamos el fichero.

-En Linux hacemos algo parecido a lo que se hace en Mac, el fichero .jar debe alojarse en */home/nom_usuario/sketchbook/tools/ArduBlockTool/tool*. Creamos las carpetas oportunas...

Una vez seguidos los pasos anteriores, si vamos a la interfaz gráfica de Arduino IDE y abrimos el menú *Herramientas*, veremos como aparece un nuevo ítem llamado “*ArduBlock*” (si no apareciese, cierra Arduino IDE y vuelve a abrirlo). O sea, que ya podemos trabajar en modo texto o gráfico y esto es todo en cuanto al software, ya que el entorno es bastante

intuitivo y no necesita muchas explicaciones, bueno o así... Quizás destacar que la ventana de Arduino IDE consta de la barra de menú arriba, unos botones de acceso rápido bajo el menú, la zona en blanco es el editor de texto donde se escribe el código, la zona verdosa es el área de mensajes y finalmente existe una zona negra que representa la consola. Por cierto, en el menú *Herramientas*, puedes seleccionar el tipo de placa que utilizas en *Tarjeta*, así como el tipo de *Programador* y *Puerto Serial*. Solo si seleccionas los valores adecuados, nuestro programa o código fuente (sketch) se transfiera correctamente al hardware.



En cuanto a ArduBlock, tampoco hay mucho que comentar, es una ventana con una zona de trabajo en color gris oscuro, donde podremos arrastrar todos los bloques funcionales que están catalogados a la izquierda. Como pueden observar parece un puzzle, así que no todos los bloques encajan con otros (esto evitará errores a los no experimentados). Arriba aparecen unos botones Load (cargar un sketch), Upload (convierte el diagrama de bloques en código en Arduino IDE) o Save (guardar) el proyecto, mientras que en la esquina superior derecha hay un recuadro que representa la zona de trabajo minimizada para poder ir a un punto concreto con un solo movimiento de ratón. Algunos bloques poseen valores que se pueden cambiar, como por ejemplo los de retraso, pines,...



Continuando con el temario, una vez tengamos nuestro código fuente o sketch, ya sea escrito por nosotros o convertido desde ArduBlock, lo siguiente sería pasar el programa a la placa Arduino. Para ello basta con presionar el botón redondo con una flecha hacia la derecha que hay en la ventana de Arduino IDE, éste es el botón para cargar de forma rápida el sketch en nuestra

placa. Claro está, la placa debe estar conectada mediante el cable USB y se debe haber instalado el driver oportuno (FTDI).

Realizado esto, lo que hace el software de desarrollo es verificar (depurado o “debugging”) que el código fuente no tiene errores sintácticos y si los tiene aparecerá un mensaje en la zona de mensajes para informarte de cual es el error para que lo repares antes de cargar el sketch. El siguiente paso sería transformar el código, escrito en lenguaje de alto nivel, al lenguaje máquina entendible por el microcontrolador, esto es lo que se denomina “compilar el código”. Por último se envía el código de bajo nivel y se guarda en la memoria del microcontrolador.

Finalmente, decir que los documentos guardados de Arduino IDE tienen la extensión *.ino, mientras que los de ArduBlock tienen la extensión *.abp (AVS Barcode Profile). El primero es el sustituto del antiguo *.pde para guardar los códigos fuente o sketches y el segundo fue promovido por AVS (Alver Valley Software) para contener barcodes. Así que cuando veas una de estas extensiones, debes saber que pertenecen a proyectos guardados de Arduino.

2-Lenguaje de programación:

Los que estéis familiarizados con la programación de otros microcontroladores como los PICs o las placas Basic Stamp de Parallax, este apartado no os supondrá mayor problema, puesto que ya tenéis unas bases y solo os falta familiarizao con la sintaxis del nuevo lenguaje. A los menos expertos les aconsejo que aprendan bien el lenguaje Arduino o que utilicen herramientas como ArduBlock.

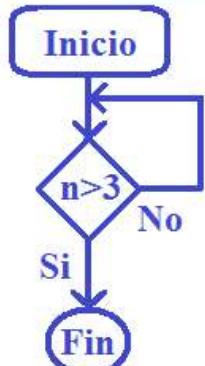
Si no sabes qué es un lenguaje de programación, para no entrar en demasiados detalles teóricos, quédate con que es un

lenguaje artificial con el que se pueden dar instrucciones o diseñar procesos.

Existen lenguajes de alto nivel (Arduino, C, C++, Java, Python,...) y bajo nivel (ASM o esamblador y código máquina). Los de alto nivel son más fáciles y entendibles para los humanos, ya que tienen una estructura sintáctica similar a la que empleamos en nuestro lenguaje. Los de bajo nivel ejercen un control directo sobre el hardware y están estrechamente relacionados con la estructura o arquitectura del mismo, pero son difíciles por ser representados por mnemónicos o código binario. Pero para facilitarnos aun más las cosas, ahora existen los lenguajes gráficos, mucho más intuitivos y fáciles para los humanos al componerse de bloques gráficos en forma de diagramas de flujo que representan el programa.

L. gráfico

L. alto nivel



```
#include <stdio.h>
{
    printf("Hola mundo");
}
```

L. bajo nivel

Ensamblador

```
.model small
.stack
.data
    saludo db "Hola mundo", "$"
.code
main proc
    mov ax, seg saludo
    mov ds, ax
    mov ah, 09
    lea dx, saludo
    int 21h
    mov ax, 4c00h
    int 21h
main endp
end main
```

```
1110011010101010101010101010
101010101010101010101010101010
10101010101010101010101010111
01110010111000000000000000000000111
011001010000111110101010101010
0111010101010101010111101010101
0101010101010101111010000000000
0000000000000000000000000000110000000
```

Código máquina

2.1.-Lenguaje Arduino:

El lenguaje Arduino es un lenguaje de programación de alto nivel similar a otros lenguajes de programación, pero muy sencillo y fácil de aprender (en comparación con otros). Este

el mismo lenguaje sirve para todas las placas de Arduino, así que no te preocupes, no tendrás que aprender un lenguaje diferente según la placa que adquieras.

Este lenguaje se debe traducir en lenguaje máquina para que el microcontrolador lo entienda, es decir, unos y ceros. Tengo que decir, que el código binario tampoco lo entiende el microcontrolador, ya que es un circuito y como tal solo comprende los pulsos eléctricos. Así que un cero será un voltaje bajo y un uno un voltaje alto, así es como los chips trabajan...

El lenguaje nos servirá para escribir un programa o sketch, que no es más que una serie de instrucciones, para ordenarle al chip lo que tiene que hacer.

La mejor forma de aprender un lenguaje es viendo código fuente escrito y escribiendo tus primeros programas basándote en otros. Por eso, recomiendo abrir Arduino IDE, ir al menú *Archivo* y luego ir abriendo todos los ejemplos ya hechos que encontrarás en el ítem *Ejemplos*. Por ejemplo, si accedes a *Basic/Blink*, verás el sketch que se necesita para hacer titilar un LED (se enciende un segundo y se apaga durante otro segundo...).

Ahora bien, ¿de qué se compone el lenguaje Arduino? Pues como cualquier otro lenguaje de programación, se compone de funciones, variables, tipos de datos, operadores, sentencias, entradas y salidas, etc. Si ya has aprendido algún otro lenguaje de programación, esto te sonará a bla bla bla...

Intentaré hacer un resumen del lenguaje Arduino para integrarlo todo y que tengáis una perspectiva global de él. No tratéis de entenderlo nada más verlo, lo iréis aprendiendo poco a poco con la práctica y viendo ejemplos ya escritos por

otros programadores (vuelvo a repetir, es lo mejor para aprender).

Estructura principal	<p><code>void setup()</code> → Estructura de configuración. Todas las sentencias englobadas aquí se repiten solo una vez.</p> <p><code>void loop()</code> → Estructura de bucle principal. Permite ejecutar las sentencias repetidas veces.</p> <p><code>#include</code> → Incluir ficheros de biblioteca</p> <p><code>#define</code> → Definir macros y constantes.</p>
Estructura de control	<p><code>if()</code> → Bucle condicional, si se cumple una función.</p> <p><code>if()...else</code> → Bucle condicional, con una opción que se ejecutan en caso de no cumplirse la función.</p> <p><code>for()</code> → Se lleva a cabo para solo una determinada función.</p> <p><code>switch()...case</code> → En función de una expresión, se seleccionará el caso oportuno a ejecutar.</p> <p><code>while()</code> → Crea un bucle que se ejecuta mientras se cumpla la función. En caso contrario cesa.</p> <p><code>do()...while</code> → Ejecuta una sentencia una o más veces en función de la expresión.</p> <p><code>break</code> → Rompe los bucles para que no se ejecuten una y otra vez.</p> <p><code>continue</code> → Obliga al bucle a ejecutar la siguiente iteración.</p> <p><code>return</code> → Sale de una sentencia.</p> <p><code>goto</code> → Va acompañado de una etiqueta o nombre y saltará allí donde esté dicha etiqueta a lo largo del código fuente.</p>
Elementos sintácticos básicos	<p><code>;</code> → Finaliza una línea o instrucción.</p> <p><code>{}</code> → Encierra un bloque de código.</p> <p><code>//</code> → Comentario de línea única. El texto que lo sucede será ignorado por el compilador.</p> <p><code>/* */</code> → Comentario multilínea. Todo el texto que se encierra entre estos símbolos se trata como un comentario.</p> <p><code>\0</code> → Carácter nulo (null).</p>
Operadores	<p><u>ARITMÉTICOS:</u></p> <p><code>=</code> → Igual o asignación</p> <p><code>+</code> → Suma</p> <p><code>-</code> → Resta</p> <p><code>*</code> → Multiplicación</p>

	<p>/ → División % → Módulo</p> <p><u>COMPARACIÓN:</u> == → Igual que != → Distinto < → Menor que > → Mayor que <= → Menor o igual que >= → Mayor o igual que</p> <p><u>BOOLEANOS:</u> && → AND → OR ! → NOT & → AND entre bits → OR entre bits ^ → XOR entre bits ~ → NOT entre bits << → Desplazamiento de bits a la izquierda >> → Desplazamiento de bits a la derecha</p> <p><u>COMPUESTOS:</u> ++ → Incremento -- → Decremento += → Suma compuesta -= → Resta compuesta *= → Multiplicación compuesta /= → División compuesta</p>
Constantes	<p>HIGH → Valor lógico alto LOW → Valor lógico bajo INPUT → Entrada OUTPUT → Salida true → Verdadero false → Falso</p>
Tipos de datos	<p>boolean → Booleano char → Carácter. Puede ser un carácter entre comillas simples (' ') o su número correspondiente en ASCII. byte → Byte (8 bits) int → Entero unsigned int → Entero sin signo long → Entero largo unsigned long → E. largo sin signo float → Coma flotante double → Coma flotante doble</p>

	<p><i>string</i> → Cadena de texto <i>array</i> → Matriz o arreglo <i>void</i> → Nada <i>word</i> → Palabra</p>
Conversión y otros	<p><i>char()</i> → Conversión carácter <i>byte()</i> → Conversión byte <i>int()</i> → Conversión entero <i>word()</i> → Conversión palabra <i>long()</i> → Conversión largo <i>float()</i> → Conversión coma flotante <i>variable scope</i> → <i>static</i> → Almacenamiento estático <i>volatile</i> → Almacenamiento volátil <i>const</i> → Constante <i>sizeof()</i> → Tamaño</p>
Funciones E/S digitales	<p><i>pinMode(pin, modo)</i> → Configura el pin especificado para que se comporte como una entrada o una salida. <i>digitalWrite(pin, valor)</i> → Activar las salidas digitales. <i>digitalRead(pin)</i> → Leer pin de entrada</p>
Funciones E/S analógicas	<p><i>analogRead(pin)</i> → Leer pin de entrada <i>analogWrite(pin, valor)</i> → Asigna el valor de un pin a la salida PWM. <i>analogReference()</i> → Configura el voltaje de referencia usando por la entrada analógica.</p>
Funciones avanzadas	<p><i>shiftOut(dataPin, clockPin, bitOrder, valor)</i> → Desplaza un byte de datos, bit a bit según el tiempo. <i>pulseIn(pin, valor, timeout)</i> → Lee un pulso en un pin. <i>Tone()</i> → Genera una onda cuadrada de una determinada frecuencia. <i>noTone()</i> → Detiene la generación de una señal cuadrada para dejar de emitir tono. <i>shiftIn()</i> → Igual a shiftout pero de una entrada.</p>
Funciones de tiempo	<p><i>delay(milisegundos)</i> → Insertar un retraso o tiempo de inactividad en ms. <i>delayMicroseconds(microsegundos)</i> → Delay o retraso especificado en microsegundos. <i>millis()</i> → Devuelve el número de milisegundos que lleva la placa Arduino ejecutando el actual programa. Cuando se desborda vuelve a contar desde cero. <i>micros()</i> → Igual al anterior pero en</p>

	<p><i>microsegundos. Sufre el overflow a los 70 min aproximadamente.</i></p>
Funciones matemáticas	<p><i>min(x, y) → Calcula el mínimo entre dos valores.</i></p> <p><i>max(x, y) → Calcula el máximo entre dos valores.</i></p> <p><i>abs(x) → Calcula el valor absoluto de un número.</i></p> <p><i>constrain(x, a, b) → Restringe un número a un rango definido.</i></p> <p><i>map(value, fromLow, fromHigh, toLow, toHigh) → Remapea un número desde un rango hacia otro.</i></p> <p><i>pow(base, exponente) → Resuelve potencia.</i></p> <p><i>sq(x) → Calcula el cuadrado de un número.</i></p> <p><i>sqrt(x) → Calcula la raíz cuadrada.</i></p> <p><i>sin(rad) → Seno.</i></p> <p><i>cos(rad) → Coseno.</i></p> <p><i>tan(rad) → Tangente.</i></p>
Funciones números aleatorios	<p><i>random() → Da valores al azar.</i></p> <p><i>randomSeed(semilla) → Busca numeros aleatorios a partir de una semilla dada.</i></p> <p><i>random(max) → Da valores aleatorios hasta un máximo impuesto.</i></p> <p><i>random(min, max) → Da numeros aleatorios entre dos valores dados.</i></p>
Control de servos	<p><i>servo.attach() → Asocia la variable Servo a un pin.</i></p> <p><i>servo.write() → Escribe un valor en el servo, controlando el eje en consecuencia.</i></p> <p><i>servo.writeMicroseconds() → Escribe un valor en microsegundos en el servo, controlandolo según el valor.</i></p> <p><i>servo.read() → Lee el ángulo actual del servomotor.</i></p> <p><i>servo.detach() → Desasocia la variable servo de su pin.</i></p>
Bits y bytes	<p><i>lowByte() → Extrae el byte de orden inferior (derecha) de una variable.</i></p> <p><i>highByte() → Extrae el byte más significativo de una variable (izquierda).</i></p> <p><i>bitRead() → Lee un bit de un número.</i></p> <p><i>bitWrite() → Escribe un bit en un número.</i></p> <p><i>bitSet() → Pone a uno un bit de una variable numérica.</i></p> <p><i>bitClear() → Limpia (pone a cero) un bit de una</i></p>

	<p><i>variable.</i></p> <p><i>bit()</i> → Calcula el bit especificado. Siendo <i>bit0</i> el primero, <i>bit1</i> el segundo, <i>bit2</i> el tercero,...</p>
Interrupciones	<p><i>attachInterrupt()</i> → Especifica la función que hay que invocar cuando se produce una interrupción externa.</p> <p><i>detachInterrupt()</i> → Apaga la interrupción dadas a una función.</p> <p><i>interrupts()</i> → Activa las interrupciones que se han desactivado.</p> <p><i>noInterrupts()</i> → Desactiva las interrupciones.</p>
Comunicaciones serie	<p><i>Serial.begin(baudios)</i> → Establece la velocidad de datos en baudios (bits por segundo) para la transmisión de datos en serie.</p> <p><i>Serial.available()</i> → Devuelve el número de bytes disponibles para ser leídos por el puerto serie. Es decir, los datos ya recibidos que están en el buffer.</p> <p><i>Serial.end()</i> → Desactiva la comunicación serie, permitiendo a los pines RX y TX ser usados como E/S digitales.</p> <p><i>Serial.parseFloat()</i> → Devuelve el primer coma flotante válido del buffer serie.</p> <p><i>Serial.parseInt()</i> → Devuelve el primer valor entero válido del buffer serie.</p> <p><i>Serial.read()</i> → Lee los datos entrantes del puerto serie.</p> <p><i>Serial.flush()</i> → Vacía el buffer de entrada de datos serie.</p> <p><i>Serial.print(datos)</i> → Lee los datos entrantes por el puerto serie.</p> <p><i>Serial.find()</i> → Lee datos del buffer serie buscando la cadena especificada.</p> <p><i>Serial.findUntil()</i> → Lee los datos de memoria hasta que encuentra una cadena.</p> <p><i>Serial.peek()</i> → Retorna el siguiente byte del puerto serie sin borrar nada.</p> <p><i>Serial.readBytes()</i> → Lee los caracteres del puerto serie dentro del buffer.</p> <p><i>Serial.readBytesUntil()</i> → Igual al anterior, pero dentro de un array.</p> <p><i>Serial.setTimeout()</i> → Pone el máximo en ms de espera para el puerto serie.</p> <p><i>Serial.write()</i> → Escribe los datos binarios en el Puerto serie.</p> <p><i>Serial.serialEvent()</i> → Muestra algún evento</p>

(cambio) ocurrido en el puerto serie.

*Las celdas con tipografía en color naranja pertenecen a componentes estructurales, mientras que las rojas son funciones.

Dando un nuevo paso, te describiré que es eso de las variables, constantes, etc. Lo primero es decirte que, a parte de las funciones vistas en la tabla anterior, tú puedes crear nombres (identificadores) para designar algunos parámetros. Estos identificadores no pueden coincidir con ninguna de las palabras reservadas a funciones u otros elementos propios del lenguaje, ya que el programa las confundiría. ¿Pero para qué sirven? Pues te pueden ayudar a bautizar tus variables y constantes (pueden ser de los tipos de datos vistos en la tabla anterior), para así poderlas llamar en un momento dado. Ahora te explicaré que es eso de variable, constantes y trataré de introducir los arrays, que son un tipo de datos algo peculiar:

-Constante: es un valor/carácter/cadena que tú determinas y no varía, de ahí su nombre. Por ejemplo, una constante podría ser el número π o cualquier otro valor que quieras que no varíe durante la ejecución. Las constantes se pueden introducir de varias maneras, una de ellas es como en el lenguaje C, utilizando “#define nom_const valor” y otra es mediante “const tipo_dato nom_const = valor”, siendo esta última la mejor. Ej:

```
#define PINLED 8  
#define num_pi 3.14  
#define RESPUESTA5 true  
const float pi = 3.14;
```

```
const char LETRA = 'a';  
cons int octal-code = 035;  
const byte mibyte = B00001101;  
cons int hexa-num = 0xAAA;
```

-Variables: son identificadores que pueden tomar cualquier valor a lo largo del programa. Imagínate que estos nombres son como contenedores que guardan cualquier tipo de información durante la ejecución y que ésta puede variar de un momento a otro. Igual que las constantes, pueden ser de los tipos vistos en la tabla anterior (entero, coma flotante, doublé, carácter, ...). Las variables también pueden tener un valor inicial predeterminado, el cual variará durante el proceso. Veamos algunos ejemplos para que lo entiendas mejor:

```
int num1;  
float decim;  
char letraactual;  
byte DATO = B00000011;  
unsigned long primitivo = 385.130.212;  
char micaracter = 65;  
char micaracter = 'A';  
float valor = -64e7;
```

-String y Array: estos tipos de datos los incluyo aquí por lo genuino de cada uno. Vayamos por partes, los string

en realidad se comportan como un array de caracteres y su sintaxis es igual a la de un array (no existe el tipo “string” como tal). Un array (arreglo o matriz) es una colección de datos que se almacenan en la memoria en forma de conjunto. El formato de un array es el siguiente (el primero es la declaración de un array sin inicializar), siendo “n” el número de elementos que integran el array:

tipo_dato nom_array[n];

tipo_dato nom_array[n] = {datos};

Los datos contenidos en un array pueden ser accedidos/introducidos utilizando unos apuntadores que tienen la siguiente sintaxis, siendo “i” la posición del dato en la memoria ocupada por el array (recuerda que la posición primera es la 0, es decir, son “zero indexed”):

nom_array[i]

Veamos algunos ejemplos de arrays, los tres primeros serían ejemplo de un string, los siguientes son otros tipos de arrays y los tres últimos son ejemplos de apuntadores:

ARRAY	RESULTADO EN MEMORIA
<i>char string1[3] = {'a', 'l', 'm'};</i>	<i>a l m</i> <i>I s a a c</i> <i>a r d u i n o</i> <i>-1 2 3 -4 5 -6 7 8 ...</i>
<i>char cadena[5] = "Isaac";</i>	<i>-1 0.04 14.6</i> <i>a l m</i> <i>I s a a c</i>
<i>char texto[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};</i>	
<i>int mathematic[10];</i>	
<i>int num3[] = {-1, 2, 3, -4, 5, -6, 7, 8};</i>	
<i>float matriz[3] = {-1, 0.04, 14.6};</i>	
<i>string1[2]</i>	
<i>cadena[0]</i>	
<i>matematic[7]=8+5;</i>	

Tras esto me gustaría hacer algunos matices y darte algunos consejos. El primero es que recuerdes siempre que los

identificadores de las variables y constantes deben coincidir con los escritos durante el programa para que funcione. Un error muy común es introducirlos mal y por ello generará un error al compilarlo. Tampoco debes descuidar los ; al final de cada sentencia o las { }, otro error muy frecuente...

```
const int ledPin = 13;          //Observa que la constante entera es ledPin
const int botonpin = 8;         //Esta es botonpin

int ledestado = HIGH;           //Fijate que esta variable se llama ledstado

void setup() {
    pinMode(botonpin, INPUT);   //Aquí daria error, puesto que nos hemos comido una n
    pinMode(ledpin, OUTPUT);    //Aquí otro, puesto que la p está en minúscula

    digitalWrite(ledPin, ledestado) //En este caso están correctas, pero olvidamos el ;
}
```

Siguiendo con las aclaraciones, decir que los valores numéricos pueden expresarse en decimal, con signo (-/+), hexadecimal (con el prefijo 0x), octal (con el prefijo 0), o binario (con el prefijo B). Así que no te extrañe ver estos prefijos en los ejemplos anteriores.

Y lo más relevante, la memoria RAM y Flash de las placas Arduino es limitada. Según el modelo puede tener más o menos, esto interfiere en lo complejo que pueda ser nuestro programa. Los datos, números, caracteres, etc., ocupan como mínimo 8 bits (1 byte), algunos como los enteros son 2 bytes, doblé, long y coma flotante hasta 4 bytes. Por eso, ten en cuenta de cuanta memoria dispones y cuantos de estos datos has incluido en tu programa. Ten en cuenta que las constantes se almacenan en la Flash y las variables en la RAM. Por ejemplo, Arduino UNO tiene 32KB de flash ($32 \times 1024 = 32768$ bytes) y una RAM de 2KB ($2 \times 1024 = 2048$ bytes). Y no dispones de toda esa memoria, habría que descontarle las direcciones de flash empleadas por las sentencias de tu programa y la memoria RAM ocupada por el sistema

Arduino. Pero no te preocupes, normalmente no tendrás problemas con la memoria si la complejidad es baja.

Seguimos... Si ya sabes programar en algún lenguaje sabrás que es una biblioteca (library), se trata de código utilizado con frecuencia que se incluye en estas bibliotecas para no tener que rescribirlo, simplemente basta con hacer una referencia a estas bibliotecas. Esto nos ayudará con partes muy recurrentes que ya nos vienen “prediseñadas”. Muchas de las funciones descritas anteriormente se encuentran dentro de las bibliotecas, por ello es necesario saber que bibliotecas existen para hacer las llamadas oportunas y disponer de dichas funciones.

Arduino IDE ya contiene algunas bibliotecas, pero si se desea alguna que no se incluya, habrá que instalarla por nuestra cuenta. Basta con descargarla, descomprimirla. Luego alojaremos el directorio dentro de la ruta *Sketchbook/Libraries*. Más tarde, la localizaremos desde el menú Sketch → Importar biblioteca, en Arduino IDE. Si os fijáis, la carpeta de la biblioteca consta de dos ficheros, uno de cabecera “*.h” y otra de código “*.cpp”.

Por cierto, antes de ver una tabla con las bibliotecas existentes, te diré que se declaran igual que en el lenguaje C, es decir, incluyendo una línea tal que así:

```
#include <nom_lib.h>
```

Por ejemplo: #include <Servo.h>

EEPROM	Contiene funciones para leer y escribir en memorias EEPROM. La función <code>read()</code> y <code>write()</code> pertenecen a ella.
Ethernet	Para conectar a internet la placa. Incluye: <code>begin()</code> ,

	<i>Server(), available(), Client(), connected(), connect(), ...</i>
<i>Firmata</i>	Para comunicación con aplicaciones de equipos informáticos.
<i>LiquidCrystal</i>	Contiene funciones para el control de pantallas Display o LCD. Contiene funciones como LiquidCrystal(), home(), display(), noDisplay(), cursor(), nocursor(), autoscroll(), ...
<i>Servo</i>	Control de servomotores.
<i>SoftwareSerial</i>	Para la comunicación serie de cualquier pin digital.
<i>Stepper</i>	Para controlar los motores de tipo paso a paso (steppers).
<i>Wire</i>	Permite comunicar dos dispositivos I2C/TWI (Two Wire Interface), para enviar y recibir datos.
<i>Matrix</i>	Manipulación de Displays de matrices LED.
<i>Sprite</i>	Control de sprites para animaciones con matrices de LEDs.
<i>Messenger</i>	Procesar mensajes de texto desde la computadora.
<i>NewSoftSerial</i>	Versión actualizada de SoftwareSerial.
<i>OneWire</i>	Control de dispositivos que usan el protocolo One Wire (Dallas Semiconductor).
<i>PS2Keyboard</i>	Lee caracteres de un teclado tipo PS/2.
<i>Simple Messenger System</i>	Retransmisión de mensajes entre Arduino y el PC.
<i>SSerial2Mobile</i>	Permite a Arduino enviar mensajes SMS o emails mediante un gadget conectado a un móvil.
<i>Webduino</i>	Biblioteca de web server extensible, para usarla con el complemento Arduino Ethernet.
<i>X10</i>	Permite enviar mensajes X10 por líneas de corriente AC. Muy útil para domótica.
<i>XBee</i>	Comunicaciones entre XBees en modo API.
<i>SerialControl</i>	Para conectar mediante puertos serie otras placas Arduino y controlarlas todas.
<i>Capacitive Sensing</i>	Puede transformar dos o más pines en sensores capacitivos.
<i>Debounce</i>	Para leer entradas digitales con ruido.
<i>Improved LCD library</i>	Repara bugs de la librería LCD oficial de Arduino.
<i>GLCD</i>	Grafica rutinas de pantallas LCD basados en chipset KS0108 ó compatibles.
<i>LedControl</i>	Para el control de matrices de LEDs o displays de 7 segmentos.
<i>LedDisplay</i>	Control de marquesina de LED de la serie HCMS-29xx.

<i>Tone</i>	Para generar señales de onda cuadrada.
<i>TLC5940</i>	Controlador de PWM de 16 canales y 12 bits.
<i>DateTime</i>	Registro de fecha y hora actual.
<i>Metro</i>	Cronometrar acciones en intervalos regulares.
<i>MsTimer2</i>	Permite utilizar “timer 2 interrupt” para detonar una acción cada N milisegundos.
<i>TextString</i>	Para el manejo de strings o cadenas de texto.
<i>PString</i>	Para imprimir en búfer.
<i>Streaming</i>	Simplifica declaraciones de impresión.

*Las de color amarillo son bibliotecas estándar y las de rojo son bibliotecas contribuidas.

Ahora ya lo sabes casi todo sobre el lenguaje Arduino, probablemente no hayas entendido mucho, si no tienes conocimientos de programación previos. No te preocupes, con los ejemplos se te aclarará todo.

2.2-Estructura de un programa:

Un programa en Arduino debe constar de unas partes fundamentales, como las estructuras de preparación y ejecución, las sentencias, etc. Todo esto lo veremos mejor en los casos prácticos, por eso estudiaremos un código muy simple, el de la siguiente ilustración:

```
/*Ejemplo es un simple programa que sirve para
crear un bucle que genera una frecuencia
de 320Hz en el pin digital 8*/
void setup()      // Esta estructura es la de preparación
{
}

void loop()       //Mientras que esta es la de ejecución
{
  tone(8, 320);
}
```

En la imagen vemos un encabezado a modo de comentario que nos introduce en una breve descripción del sketch. Seguidamente viene la línea de preparación (`void setup()`), con la que se inicia el programa y se ejecutan todas las sentencias que haya en su interior (es decir, entre `{...}`). Luego la línea que marca la ejecución del bucle, cuya función es ejecutar una y otra vez las sentencias que están dentro de ella (`{...}`). Finalmente la sentencia que determina la funcionalidad de este ejemplo concreto, emitir un tono por el pin 8 con una frecuencia de 320 Hz.

Veamos otro ejemplo algo más complejo, en el que llama la atención que antes de la estructura de preparación existen otras líneas de código. Estas son declaraciones, aunque también pueden ser llamadas a bibliotecas, etc. Luego vemos la estructura de preparación (`void setup()`) y entre sus llaves se encuentran dos sentencias que se ejecutan solo una vez, concretamente sirven para configurar los pines como entrada y salida, respectivamente. Tras esto viene la estructura de bucle, dentro de la cual hay una sentencia que lee el estado del pin del botón y pone la variable “estado” con el valor leído. Además existe otra estructura condicional `if-else`, la cual

determina que hacer en caso de estar el botón presionado o no estarlo:

```
// Programa para el control de un LED mediante un botón

const int pinboton = 2;      /*Declaración de la constante entera que marca al pin 2
                                como la entrada donde está conectado el botón*/
const int patillaLED = 13;    /* Declaración para marcar el pin 13 como la salida
                                donde se conecta el LED*/

//Las anteriores declaraciones son constantes, por tanto no varían
//La siguiente si que es variable
int estado = 0;             //Declara que el estado inicial del botón es 0

void setup() {
    //El pin del LED se comportará como salida
    pinMode(patillaLED, OUTPUT);
    //El pin del botón se comportará como entrada
    pinMode(pinboton, INPUT);
}

void loop(){
    //Lee el estado del pin donde está conectado el botón
    estado = digitalRead(pinboton);

    //Condición que determina lo que hacer en caso de que esté pulsado
    if (estado == HIGH) {
        // en este caso activa el LED
        digitalWrite(patillaLED, HIGH);
    }
    else {
        // En caso de no estar presionado, el LED estará apagado
        digitalWrite(patillaLED, LOW);
    }
}
```

Como ejercicio te propongo que seas tú mismo el que describa cada parte de alguno de los ejemplos que contiene Arduino IDE. Verás como los comentarios te ayudan en ello. Por eso, como consejo, estaría bien que escribas comentarios en cada línea de código que tú escribas, esto te ayudará a saber para que sirve y si otro la lee comprenderás lo que has querido hacer. También es interesante poner un comentario al principio para aclarar para que sirve el sketch.

Finalizo con otra explicación importante y es que cualquier código fuente debe cumplir una serie de condiciones para considerarse óptimo, que son:

- La sintaxis debe ser correcta. No debe haber errores.
- Debe estar correctamente escrito, es decir, debe hacer lo que se espera de él de una manera correcta. A veces, aunque la sintaxis es buena, el programa hace cosas raras o inesperadas.
- El código fuente debe ser lo más claro posible, si es necesario añadir comentarios aclarativos. Esto facilitará su lectura y desarrollo posterior.
- La eficiencia es otro punto primordial. Varios códigos pueden estar correctos y realizar adecuadamente la tarea para la que han sido programados, pero puede que algunos no lo hagan de la forma más eficiente (gestionando los recursos al máximo sin desperdiciarlos).

Si eres precavido y meticuloso con el código te resultará más fácil corregir errores, detectar puntos flacos y optimizar tu source code, además de estar orgulloso de haber creado un buen programa.

2.3-ArduBlock: lenguaje gráfico:

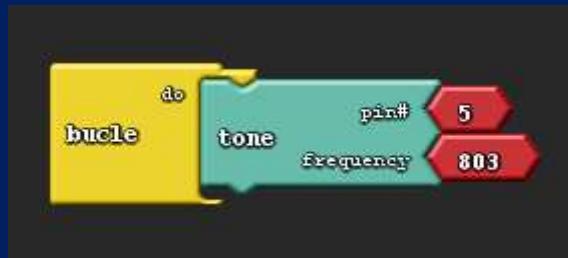
Como vimos en secciones anteriores, la ventana de ArduBlock contiene una serie de fichas catalogadas por categorías en la parte izquierda de la ventana. ArduBlock es muy sencillo e intuitivo, por ello bastará con explicar que las fichas son

bloques funcionales que podemos arrastrar hasta la zona de trabajo para ir creando nuestro sketch de manera gráfica.

Ya dijimos que estos bloques son como fichas de puzzle, no encajan todas con todas, sino que solo algunas de ellas pueden ser enlazadas. Esto evitará enlazar piezas que no se pueden unir por falta de sentido o porque no sea físicamente posible. Así que si sabes hacer puzzles, sabes programar en ArduBlock.

Fichas	Función
<i>Control</i>	<i>Son las fichas que permiten controlar tareas o realizar bucles con otras fichas.</i>
<i>Pins</i>	<i>Orientadas a la gestión de líneas de entrada/salida, ya sean analógicas o digitales.</i>
<i>Número/Constantes</i>	<i>Con ellas podemos establecer variables con las que asignar valores y constantes. Estas fichas pueden ser manipuladas haciendo clic sobre ellas para cambiar sus valores.</i>
<i>Operadores</i>	<i>Nos permiten realizar operaciones lógicas y aritméticas con las fichas anteriores.</i>
<i>Utilidades</i>	<i>Aquí hay fichas muy variadas: contadores, temporizadores, retardos, generadores de números aleatorios, mapear números, imprimir o transmitir a un puerto serie, etc.</i>
<i>TinkerKit Blocks</i>	<i>Aquí encontraremos complementos útiles como LEDs, botones, acelerómetros, sensores, relés, transistores, resistencias, etc. Estos complementos pueden ser muy útiles para realizar tareas con ellos.</i>
<i>DFRobot</i>	<i>Igual al anterior, solo que aquí los sensores y módulos que encontraremos están especialmente dirigidos para la robótica.</i>
<i>Seeedstudio Grove</i>	<i>Finalmente, podemos tener acceso a fichas para joysticks que usaremos en el manejo de algunos proyectos.</i>

Como ejemplo, supongamos que deseamos crear un bucle que emita una frecuencia de 803 Hz por el pin digital número 5, bastaría con hacer lo siguiente:



Si ha cometido algún error o desea deshacerse de alguna ficha, arrástrela hacia la zona negra de la izquierda, bajo las categorías de las fichas y se eliminará. Si haces clic con el botón derecho sobre una ficha, te dará dos opciones, añadir comentario (o eliminarlo si ya lo tiene) y clonarla. Así podrás documentar tus diseños con comentarios y copiar partes que vayas a usar más veces para evitarte volver a ponerlas desde cero...

3-Ejemplos prácticos...

Antes de nada, los parámetros del código fuente escrito y los físicos deben coincidir. Si no es así puede considerarse un foco importante de errores. Por ejemplo, si en el código has puesto que se emitirá un tono de 500Hz en el pin#8 y el zumbador lo has insertado en el pin 6, no funcionará. ¡Presta atención para que la correlación sea la correcta!

Una vez tenemos los conocimientos teóricos básicos, vamos a pasar a la acción que es como verdaderamente se aprende. Existen investigaciones sobre el cerebro y el aprendizaje que han demostrado que recordamos aproximadamente un 10% de lo que leemos, un 26% de lo que escuchamos, un 30% de lo que vemos, un 50% de lo que vemos-escuchamos, un 70% de lo que decimos y un 90% de lo que hacemos. Por tanto, ¿por

qué perder el tiempo leyendo teoría si solo vamos a recordar un 10% pudiendo hacer proyectos y recordar un 90?

A continuación muestro 13 ejemplos prácticos, 8 realizados en Arduino IDE y otros 5 en Ardublock. ¡Disfruta aprendiendo!

####=====@@@@=====####

*--Para no engordar más el documento, facilitarte la edición y prueba, los ejemplos los he incluido en ficheros *.ino y *.abp junto a este PDF--*



####=====@@@@=====####

¡¡¡ Gracias por su atención y le deseo que le haya sido útil este tutorial !!! Espero que muy pronto esté creando sus propios proyectos con Arduino y los comparta...

Recuerde, para más información, puede estar al tanto del blog:

<http://architecnologia.es/>



FIN...