E. E. S. T. Nº 5 – TIGRE

ELECTRÓNICA

SISTEMAS DE CONTROL

Arduino

TRABAJO PRÁCTICO Nº 1

TEMA: Introducción a la programación con Arduino

ALUMNO:	**Curso:**
FECHA DE ENTREGA:	
FECHA DE APROBACIÓN:	
CALIFICACIÓN:	FIRMA:

¿Qué es Arduino?

Arduino es una plataforma de **hardware libre**, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

El **hardware** consiste en una placa con un **microcontrolador** Atmel y **puertos** de entrada/salida.

El **software** consiste en un entorno de desarrollo que implementa el lenguaje de programación **Processing/Wiring** y un cargador de arranque (bootloader) que corre en la placa.

Al ser **open-hardware**, tanto su diseño como su distribución son **libres**. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de **proyecto** sin necesidad de adquirir **licencia** alguna.

Algunos de los modelos de Arduino con sus principales características son:

Modelo	µControlador	Flash KiB	EEPROM KiB	SRAM KiB	Entradas-Salidas digitales	de las cuales son PWM	Entradas analógicas
Nano	ATmega328	32	1	2	14	6	8
Uno	ATmega328	32	1	2	14	6	6
Mega2560	ATmega2560	256	4	8	54	14	16
Leonardo	ATmega32U4	32	1	2.5	20	7	12

Arduino UNO rev 3

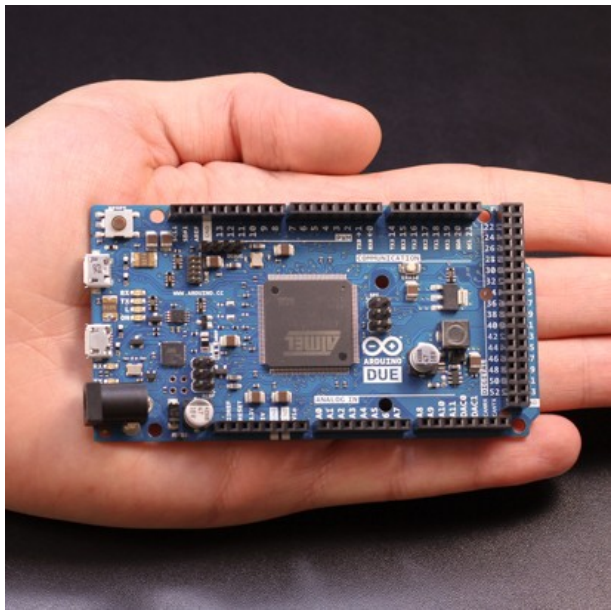
La versión más **conocida**, mejor **documentada** y la más **adecuada** para empezar es la **Arduino Uno**.

La tarjeta está basada en el micro **ATmega328** y funciona a **16 MHz**. Tiene **14 E/S** digitales (**6** de ellas pueden utilizarse como salidas **PWM**), **6 entradas analógicas**, conector **USB**, conector para alimentación externa, conector **ICSP** y botón de **reset**.

La **UNO rev 3** es totalmente **compatible** con versiones anteriores pero incorpora una serie de **mejoras**: nuevo **bootloader** OptiBoot que permite cargar programas a 115 Kbps (antes era a 56 Kbps) y ocupa sólo 512 bytes (antes 1024). El **bootloader** nos permite **transferir** los programas sin necesidad de un programador especial como en el caso de los PIC.

El nuevo chip convertidor USB-Serie es el ATmega16u2 (en lugar de USB FTDI) y permite que al conectar la placa a la PC pueda ser **reconocida** como un dispositivo **USB** común.

¿Por qué Arduino?



Hay muchos otros microcontroladores y plataformas con microcontroladores disponibles para la computación física: los PIC de Microchip, Parallax Basic Stamp, BX-24 de Netmedia, Phidgets, Handyboard del MIT y muchos otros ofrecen funcionalidades similares. Todas estas herramientas organizan el complicado trabajo de programar un microcontrolador en paquetes fáciles de usar. Arduino, además de **simplificar** el proceso de trabajar con microcontroladores, ofrece algunas **ventajas** respecto a otros sistemas a profesores, estudiantes y amateurs:

- **Accesible**. Las placas Arduino son más **baratas** comparadas con otras plataformas de microcontroladores. Todas las versiones pueden adquirirse armadas o puede ser montadas por uno mismo, ya que sus especificaciones técnicas son abiertas.
- **Multi-Plataforma**. El software de Arduino funciona en los sistemas operativos Windows, Macintosh OSX y Linux, en cambio, la mayoría de los entornos para microcontroladores están **limitados** a Windows.
- **Entorno de programación simple y directo.** El entorno de programación de Arduino es fácil de usar para principiantes y lo suficientemente flexible para los usuarios avanzados. Pensando en los profesores, Arduino está basado en el entorno de programación de **Procesing** con lo que el estudiante que aprenda a programar en este entorno se sentirá familiarizado con el entorno de desarrollo Arduino.
- **Software ampliable y de código abierto**. El software Arduino está publicado bajo una licencia **libre** y está preparado para ser **ampliado** por programadores experimentados. El lenguaje puede ampliarse a través de **librerías de C++** y, si se está interesado en profundizar en los detalles técnicos, se puede dar el salto a la programación en el lenguaje **AVR C** en el que está basado. De igual modo se puede añadir directamente código en AVR C en sus programas si así lo desea.
- **Hardware ampliable y de código abierto**. Arduino está basado en los microcontroladores ATMEGA168, ATMEGA328 y ATMEGA1280. Los planos de los módulos están publicados bajo licencia **Creative Commons**, por lo que diseñadores de circuitos con experiencia pueden hacer su **propia** versión del módulo, ampliándolo u optimizándolo. Incluso usuarios relativamente inexpertos pueden construir la versión para la placa de desarrollo para entender cómo funciona y ahorrar algo de dinero.

Comenzando con Arduino (en Windows)

Vamos a explicar cómo conectar la placa Arduino al ordenador y volcar el código del primer programa. Un programa en Arduino se denomina habitualmente **sketch**.

Si tiene **otro** sistema operativo consulte el sitio oficial

http://arduino.cc/en/Guide/HomePage

1) Consiga un Arduino con su cable USB

Asumimos que tiene un **Arduino UNO** pero estas instrucciones sirven también para las versiones Duemilanove, Nano, Arduino Mega 2560 o Diecimila.

2) Descargue el IDE

Descargue la última versión **estable** del entorno de desarrollo desde la página de descargas: http://arduino.cc/en/Main/Software. Elija el fichero con extensión .zip.

Luego **descomprima** el fichero en una carpeta de su preferencia. Asegúrese de mantener la estructura de directorios. Haga doble clic en la carpeta ***arduino-x.xx*** para abrirla. Debería ver una serie de ficheros y carpetas ahí dentro.

3) Conecte la placa

Al **conectar** la placa Arduino a su ordenador usando el cable USB, el LED indicador de la alimentación (identificado como ON en la placa) debería quedar **encendido** a partir de ese momento.

Es muy probable que si la placa es **nueva,** tenga ya **grabado** el sketch ***Blink*** y comience a **parpadear** otro LED identificado con la letra L.

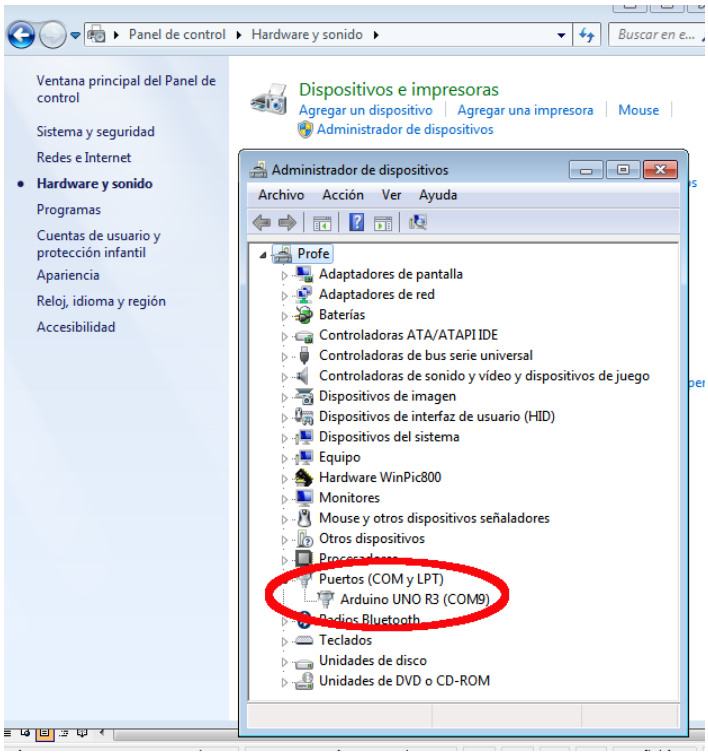
4) Instale los drivers

Que la placa esté conectada no significa que haya **comunicación** con la computadora, simplemente está alimentada eléctricamente a través de ella.

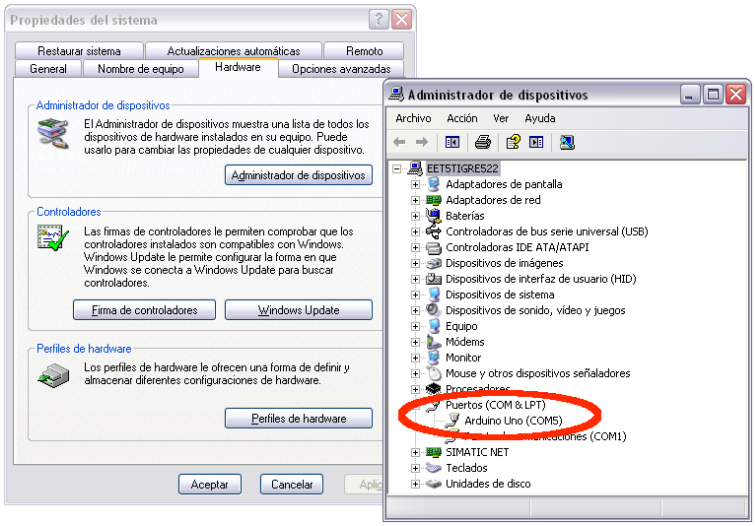
Al conectarla, Windows trata de buscar un controlador adecuado para ella. La mayoría de las veces el proceso falla, así que cancele el mismo y trate de instalar el driver de manera **manual.**

Elija la opción de ***Instalar desde una lista o localización específica (Avanzado)***, navegue hacia el lugar donde descomprimió el fichero del paso 2 y busque en la carpeta ***drivers*** el fichero ***arduino.inf***.

Para comprobar que los *drivers* se han instalado **correctamente** puede hacer:



- En la versión Windows 7 (imagen de la izquierda): abra **Panel de Control – Hardware y sonido - Administrador de Dispositivos**. Busque en la sección de puertos COM qué **puerto** le asignó Windows a su placa Arduino. Este dato es **importante** para el paso 8.
- En la versión Windows XP (imagen de más abajo) haga clic secundario en **Mi PC** luego vaya a **Propiedades – Hardware** y busque en la sección de puertos COM qué **puerto** le asignó Windows a su placa Arduino. Este dato es **importante** para el paso 8.

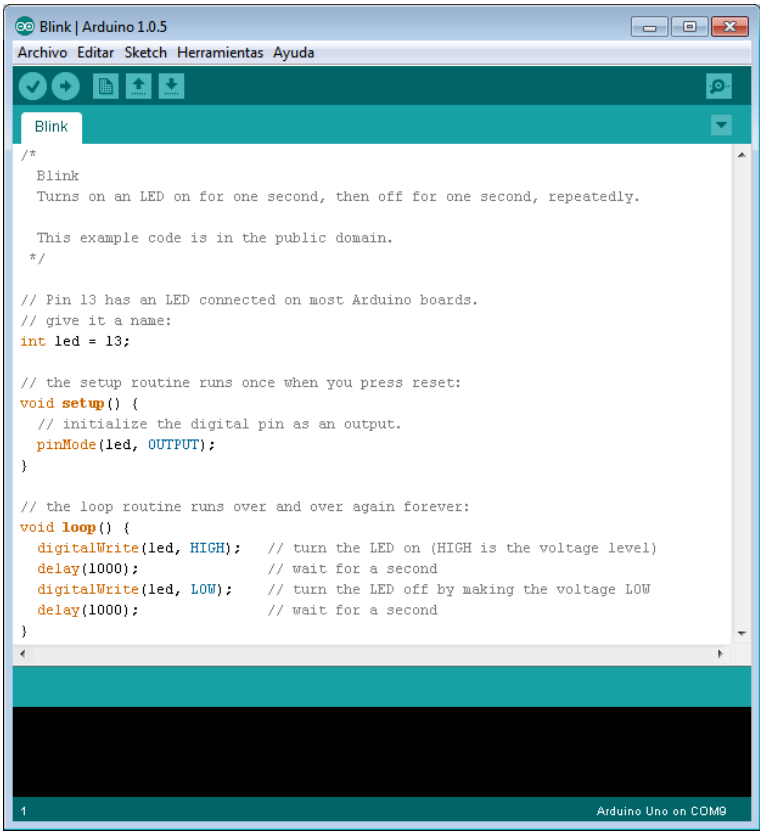


5) Ejecute la aplicación Arduino

Ejecute el fichero ***arduino.exe*** de la carpeta de instalación. Deberá aparecerle una ventana en blanco. Si el idioma de la interfaz no es el correcto, puede ejecutar la combinación de teclas ***Ctrl + ,*** (Control coma) y seleccionar el idioma de su preferencia.

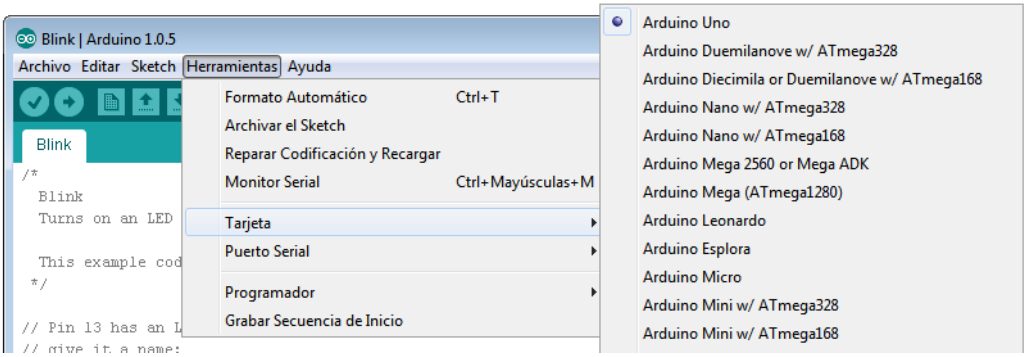
6) Abra el ejemplo Blink

Acceda a ***Archivo – Ejemplos – 01. Basics – Blink***.



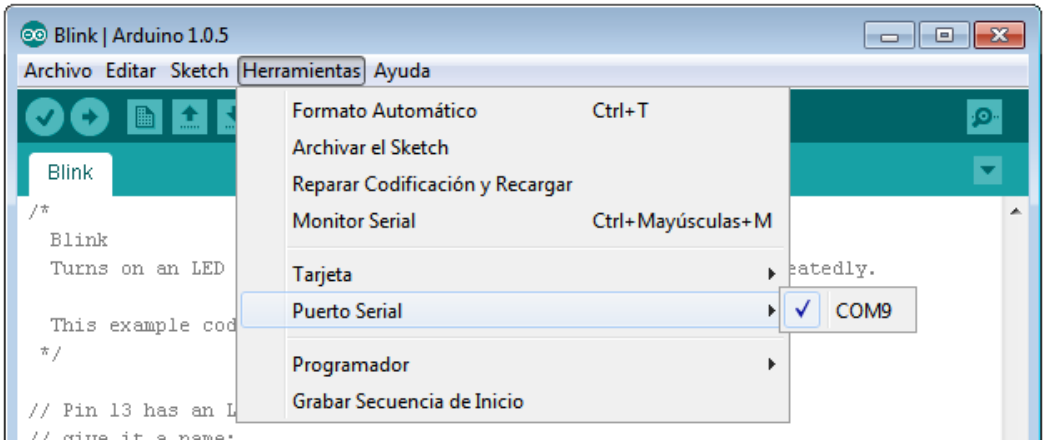
7) Seleccione su placa

Acceda a ***Herramientas – Tarjeta*** y luego elija su placa.



8) Seleccione el puerto serie

Acceda a ***Herramientas – Puerto Serial*** y el mismo debe **coincidir** con el puerto que verificó en el **Paso 4**.

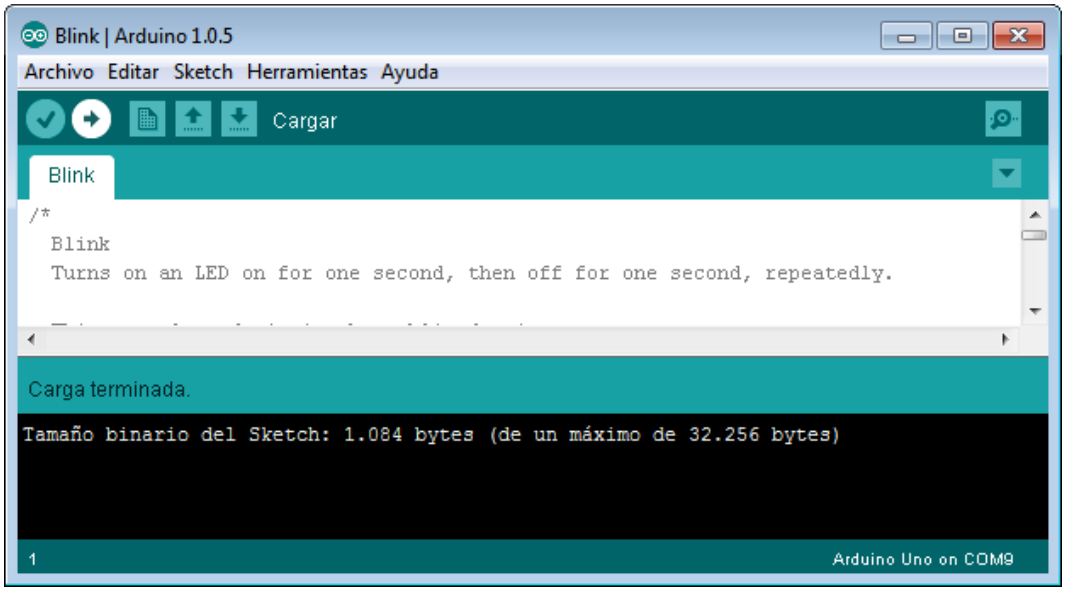


9) Suba el sketch a la placa

Pulse sobre el botón **Cargar** en el entorno Arduino (está identificado con una flecha apuntando hacia la derecha). Espere unos pocos segundos porque primero el programa debe compilarse y luego es transferido a la placa

Luego de unos instantes debería ver **parpadear** los LEDS **RX** y **TX** de la tarjeta, esto indica que se está **transfiriendo** el programa.

Si el volcado del código es **exitoso** verá aparecer el mensaje "**Carga terminada**" en la barra de estado.



Si tuvo alguna dificultad puede consultar en la página de **Arduino Solución de problemas** http://arduino.cc/es/Guide/Troubleshooting

Si todo funcionó correctamente ya está **listo** para **empezar** a trabajar.

Primer proyecto

Vamos hacer el "**Hola Mundo"** de los proyectos con microcontroladores: el clásico encendido y apagado de un **LED**.

UD ya estuvo trabajando con el ejemplo ***Blink*** y lo que nos proponemos a continuación es "**entender**" como funciona y qué **cuidados** hay que tener para conectar los diferentes **componentes** a los pines de la tarjeta.

Un proyecto Arduino consta de dos fases: **diseño** del **hardware** y escritura del **software**.

Diseño del hardware

Es la parte **física** del proyecto: la interconexión de los diferentes elementos y la placa.

Uno de los aspectos que debemos tener en cuenta es la capacidad de **corriente** que tolera cada **pin**. Cada pin de la placa admite una corriente **máxima** de **40 mA**.

Otro aspecto es como vamos a alimentar la tarjeta. Tenemos dos posibilidades:

- Cable de conexión **USB** que tolera como **máximo 500 mA**.
- Fuente de CC **externa** de entre **7 a 12 V**. Aunque soporta un **máximo** de 20 V.

Más allá del tipo de fuente que utilicemos, la placa posee un regulador para que el voltaje con que trabaja el micro sea de **5 V**.

En la parte de la tarjeta que está identificada con la leyenda **POWER**, encontramos los pines de **5 V** y **GND** que nos permite tomar alimentación para usarla en nuestro protoboard.

Conexión de un LED

Como sabemos, si conectamos **directamente** un LED a alguno de los pines corremos el riesgo no solo que quemar el LED sino de **dañar** la placa ya que consumiría más **corriente** de la que tolera ese pin. Por lo tanto hay que **limitar** la corriente del LED mediante una **resistencia**. Recordemos como se calcula:

$$R = \frac{V_{ALIMENTACIÓN} - V_{LED}}{I}$$

En donde $\mathbf{V_{ALIMENTACIÓN}}$ es 5 V. El valor de I de los LED comunes de 5mm es entre 15 y 20mA. $\mathbf{V_{LED}}$ **depende** del **fabricante** y del **color**. En términos generales, pueden considerarse de forma aproximada los siguientes valores de $\mathbf{V_{LED}}$:

Rojo = 1,8 a 2,2 voltios.

Anaranjado = 2,1 a 2,2 voltios.

Amarillo = 2,1 a 2,4 voltios.

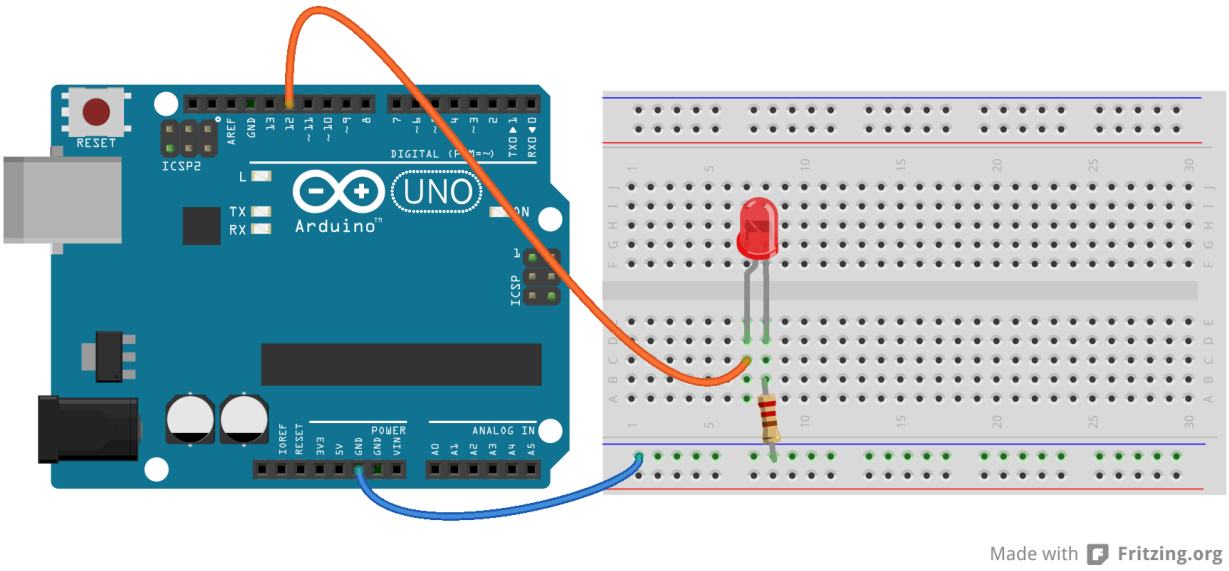
Verde = 2 a 3,5 voltios.

Azul = 3,5 a 3,8 voltios.

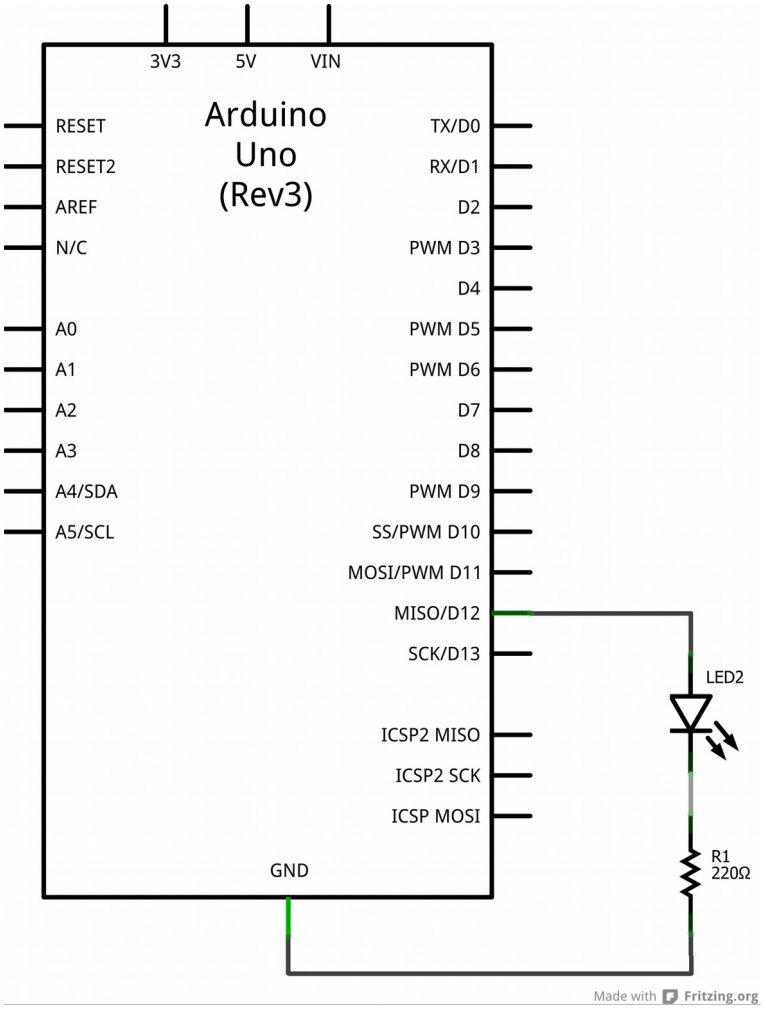
Blanco = 3,6 voltios.

Si vamos a conectar un LED **rojo** y tomamos como referencia los valores **extremos** de V_{LED} e I nos resulta que la resistencia adecuada está entre 150 Ω y 220 Ω.

La imagen siguiente ilustra como debe conectarse un LED rojo al pin 12.



El diagrama **esquemático** del circuito sería este:



Escritura del software

Consiste en **escribir** el programa o **sketch** para nuestro proyecto. El lenguaje utilizado es muy parecido al C y vamos a ir explicando los conceptos a medida que avancemos.

```
1  /*
2    _04_01_Ejemplo
3    Enciende y apaga un LED conectado al PIN 12
4    con un período de un segundo:
5    500 ms encendido y 500 ms apagado.
6    Este código de ejemplo es de dominio público.
7   */
8
9  //Configuración inicial.
10
11 void setup(){
12   pinMode(12, OUTPUT); // Pin 12 como Salida.
13 }
14
15 //Programa principal.
16
17 void loop(){
18   digitalWrite(12, HIGH); // La salida 12 en estado Alto
19   delay(500);             // Demora medio segundo
20   digitalWrite(12, LOW);  // La salida 12 en estado Bajo
21   delay(500);             // Espera medio segundo
22 }
```

Los sketchs de Arduino tienen una **estructura** más o menos rígida:

- La parte de **declaración** de variables o constantes.
- La sección de **inicialización,** identificada mediante **`void setup().`**
- La sección del programa **principal,** identificada mediante **`void loop().`**

Es una buena costumbre **documentar** los programas utilizando los **comentarios.** Ellos no tienen ninguna incidencia en el **código** ya que son **ignorados** por el **compilador** pero sirven para entender el mismo, especialmente si se lo comparte con otros programadores.

Hay **dos** tipos de comentarios:

- Los que ocupan solamente una **sola** línea que comienzan con la secuencia **`//.`**
- Los que pueden ocupar **más** de una línea que deben comenzar con la secuencia **`/*`** y finalizar con la secuencia **`*/`**

Desde la línea **1 a la 7** del código de ejemplo hay un **comentario** que suele ponerse al principio de todo programa que sirve para indicar el **nombre** y **propósito** del mismo, el **autor**, la **fecha** de creación o modificación, el tipo de **licencia**, etc.

En varias partes del código encontrará comentarios de una sola línea colocados en lugares **estratégicos** para poder **comprender** su funcionamiento.

Un programa debería poder entenderlo **cualquier** persona y de eso depende mucho la **calidad** de los comentarios. Acostúmbrese a usarlos especialmente cuando utiliza alguna técnica **extraña** para realizar una tarea o resolver alguna situación y teme olvidar como la realizó.

Inicialización `setup()`

La función **`setup()`** se coloca al **principio** de un programa o sketch. Se emplea para configurar el **estado** de los pines, establecer la **velocidad** de comunicación del puerto serie, inicializar **librerías**, etc.

Esta función se ejecutará una **única** vez después de que se conecte la placa Arduino a la fuente de alimentación o cuando se pulse el botón de **reinicio** (reset) de la placa.

Un bloque de instrucciones dentro de **`setup()`** debe estar encerrado mediante llaves: una al comienzo del bloque **`{`** y otra al final **`}`**.

```
11  void setup(){
12    pinMode(12, OUTPUT); // Pin 12 como Salida.
13  }
```

En nuestro ejemplo, en la línea 12 usamos una sola instrucción: la función **`pinMode`**.

`pinMode()`

Descripción

Configura el pin especificado para comportarse como una **entrada** o una **salida**.

Sintaxis

```
pinMode(pin, modo);
```

Parámetros

pin: el número del pin que se desea configurar.

modo: **INPUT** (Entrada) o **OUTPUT** (Salida).

Pines configurados como salidas

Las palabras **`INPUT`** o **`OUTPUT`** se denominan **constantes** en el lenguaje que utiliza Arduino y deben escribirse **siempre** en **mayúsculas**.

Los pines configurados como **salida** (**`OUTPUT`**) con **`pinMode()`** se dice que están en estado de **baja impedancia**. Esto implica que pueden **proporcionar** corriente o masa a otros circuitos.

Los pines permiten **alimentar** (**proveer** de corriente positiva) o **introducir** (proveer de masa) hasta **40 mA** (miliamperios) de corriente a otros dispositivos.

Esto es muy **útil** para alimentar **LED's** pero **inservible** para controlar la mayoría de los **relés o motores** ya que es necesario añadir circuitos extra para **amplificar** la corriente.

Si **cortocircuitamos** los pines de Arduino o intentamos manejar dispositivos de **alto consumo** de corriente, podemos **dañar** o destruir los transistores de salida del pin o dañar todo el chip ATmega. La mayoría de las veces, esto se traducirá en un pin "muerto" en el microcontrolador, pero el resto seguirá funcionando adecuadamente. Por esta razón, es buena idea, para conectar los pines de salida a otros dispositivos, usar **resistencias** de entre 470 Ω a 1k, a menos que se requiera **mayor** consumo de corriente para una aplicación particular, siempre que se respete el **máximo** admisible de **40mA**.

Programa principal o ciclo `loop()`

Luego de la función **`setup()`** se encuentra la función **`loop()`**.

Cada una de las instrucciones que están dentro del bloque **`loop()`**, se **ejecutan** una tras otra permitiéndole al programa obtener datos del exterior, tomar decisiones, hacer cálculos o responder hacia el exterior con alguna acción. Luego de llegar a la **última** instrucción del bloque, el control del programa **vuelve** a la **primera**.

El programa de ejemplo consta de **cuatro** instrucciones. Es importante destacar que cada una de ellas finaliza con un **punto y coma**.

```
17  void loop(){
18    digitalWrite(12, HIGH); // La salida 12 en estado Alto
19    delay(500);             // Demora medio segundo
20    digitalWrite(12, LOW);  // La salida 12 en estado Bajo
21    delay(500);             // Espera medio segundo
22  }
```

Función digitalWrite()

Dentro de **`loop`** la primera instrucción que se utiliza (línea 18) es **`digitalWrite`**.

`digitalWrite()`

Descripción

Escribe un valor **HIGH** o **LOW** hacia un pin digital.

Sintaxis

```
digitalWrite(pin, valor);
```

Parámetros

pin: el número de pin.

valor: **HIGH** (alto) o **LOW** (bajo).

Si el pin ha sido configurado como **OUTPUT**, su voltaje será de **5V** para **HIGH** o de **0V** (tierra) para **LOW**. Las constantes **HIGH** y **LOW** también siempre van en **mayúsculas**.

Función delay()

Por último, en el ejemplo se utiliza la función **`delay()`**.

`delay()`

Descripción

Pausa el programa por un tiempo especificado en milisegundos.

Sintaxis

```
delay(ms);
```

Parámetros

`ms:` el número de milisegundos que se desea pausar el programa (unsigned long)

ADVERTENCIA

Si bien resulta muy práctico utilizar la función **`delay`**() y en muchos programas es utilizada para retardos cortos, su uso tiene **desventajas** significativas.

Mientras se está ejecutando **`delay`**() la **mayoría** de las operaciones del microcontrolador se **detienen**: no hay lectura de datos, ni cálculos matemáticos ni se pueden cambiar el estado de los pines hasta que **finalice** el tiempo establecido.

Aunque haya otras operaciones que siguen funcionando, como la mantención el estado de los pines, la recepción serie y las interrupciones, los **programadores** expertos suelen **evitar** el uso de **`delay()`** para la sincronización de eventos de más de **10 milisegundos** salvo que el sketch sea muy simple como en este caso.

Como una mejor alternativa existe la función **`millis()`** pero como su uso no es tan intuitivo ni directo como **`delay()`** la dejaremos para más adelante.

Actividades

EJ-0401) Efectúe los **cálculos** necesarios para conectar con seguridad tres LEDs de diferentes colores a tres pines digitales del Arduino. Luego escriba el código correspondiente para simular el comportamiento de un **semáforo** de tránsito.

El funcionamiento deberá tener las siguientes características:

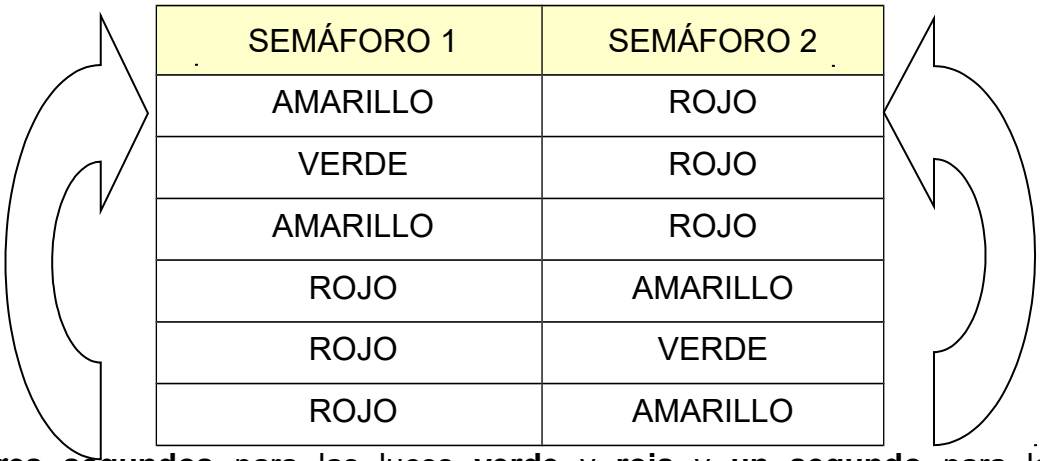
- La **secuencia** que debe seguir el semáforo es:

  Rojo - Amarillo - Verde - Amarillo y volver a comenzar.

- Durante la secuencia las luces **verde** y **roja** deben permanecer encendidas durante **tres** segundos y la **amarilla** durante **un** segundo.

EJ-0402) Basado en el ejercicio anterior haga las **modificaciones** correspondientes en el código para que antes de pasar de verde a amarillo, el LED **verde parpadee** unos instantes, como sucede en algunos semáforos reales. Si es **necesario** modifique la duración de tiempos de encendido de los LEDs para que los cambios sean visibles.

EJ-0403) Se desea operar **dos semáforos** de una intersección de dos calles. La secuencia en este caso deberá ser:



SEMÁFORO 1	SEMÁFORO 2
AMARILLO	ROJO
VERDE	ROJO
AMARILLO	ROJO
ROJO	AMARILLO
ROJO	VERDE
ROJO	AMARILLO

Utilice **tres segundos** para las luces **verde** y **roja** y **un segundo** para las luces **amarillas**.