Sesion 2 –

Primer Programa

2.1 Objetivos

- Fijar algunas ideas básicas sobre programación.
- Comprender la estructura de un programa Arduino (Sketch).
- Definir las estructuras de bloques.
- Las primeras instrucciones.

2.2 Material requerido.

Arduino Uno o similar.

Un cable USB adecuado al conector de tu Arduino.

Un PC con el entorno de Arduino correctamente instalado y configurado.

2.3 Algunas ideas básicas sobre programación.

Un **programa de ordenador** es básicamente el equivalente a una *receta de cocina*… pero destinado a un público distinto.

Mientras que las personas somos razonablemente buenas *interpretando* las instrucciones, generalmente vagas, de una receta de cocina, cuando programamos quien debe entendernos es un ordenador que espera instrucciones precisas respecto a lo que debe hacer y que además carece por completo de la imaginación o capacidad de improvisación humana.

Por ello se desarrollan los lenguajes de ordenador, para dar instrucciones a una máquina de forma:

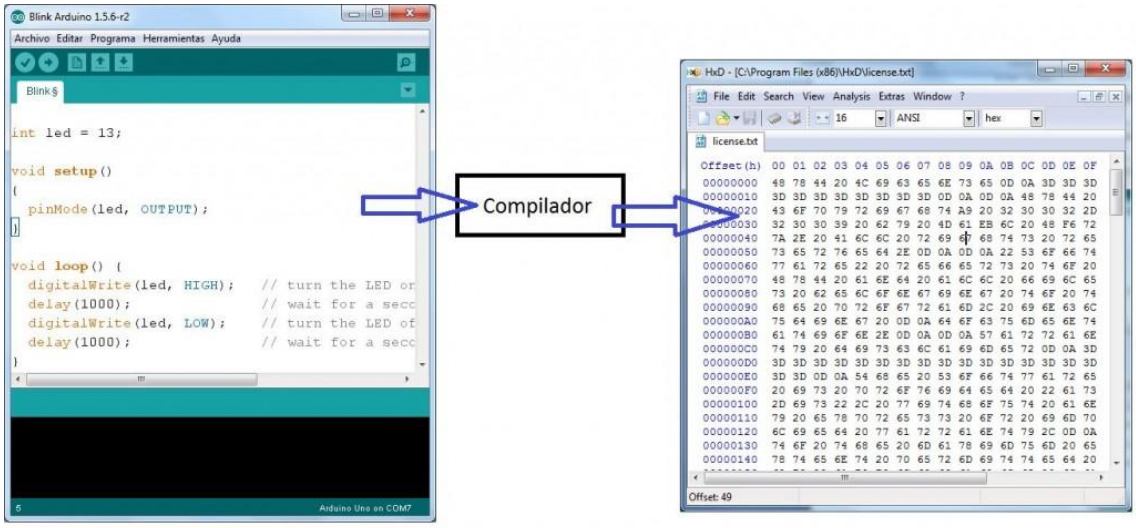
- Precisa: Sin ambigüedades inherentes a la comunicación humana.
- Univoca: Solo se puede interpretar de una manera.
- Concisa: Preferiblemente ordenes cortas.

El IDE de Arduino se programa en una variante de C++ , que es un lenguaje muy extendido por sus características, aunque no es un lenguaje sencillo. C++, que fija reglas estrictas de cómo escribir estas instrucciones.

Un programa es una serie de instrucciones que se ejecutan en secuencia (salvo que indiquemos expresamente condiciones precisas en las que esta secuencia se altera).

Un programa interno comprueba que la sintaxis de nuestro programa es acorde a la norma de C++, y si hay cualquier cosa que no le convence dará un error y finalizará la comprobación obligándonos a revisar lo que hemos escrito.

Cuando el comprobador acepta nuestro programa, invoca otro programa que traduce lo que hemos escrito a instrucciones comprensibles para el procesador de nuestro Arduino. A este nuevo programa se le llama compilador.



Esto es lo que nosotros entendemos.

Esto es lo que entiende el procesador.

El compilador convierte nuestras instrucciones (código fuente) en instrucciones del procesador (código ejecutable).

2.4 Estructura de un programa Arduino.

Un programa o sketch de Arduino consiste en dos secciones o funciones básicas:

- *Setup: Sus instrucciones se ejecutan solo una vez, cuando se arranca el programa al encender Arduino o cuando pulsamos el botón de reset. Generalmente incluye definiciones e inicializaciones de ahí su nombre.*
- *Loop: Sus instrucciones se van ejecutando en secuencia hasta el final…. Y cuando acaba, vuelve a empezar desde el principio haciendo un ciclo sin fin.*

Cuando abrimos el **IDE de Arduino** (o hacemos [Menú]\Archivo\nuevo) él nos escribe ya estas dos funciones (en color cobre):

Nótese que el principio de cada función es indicado por la apertura de llave " { " y el fin de la misma corresponde al símbolo de cerrar llaves " } ".

De hecho el conjunto de instrucciones contenidas entre una apertura y cierre de llaves se llama **bloque** y es de capital importancia a la hora de que nuestro **Arduino** interprete de una u otra manera las instrucciones que le damos.

Es imperativo que a cada apertura de una llave corresponda un cierre de llave. En sucesivos capítulos ampliaremos este concepto.

Por ahora resaltar las líneas que aparecen dentro de los bloques principales:

```
// put your setup code here, to run once
// put your main code here, to run repeatedly
```

Cualquier cosa que escribamos precedido por " // " son comentarios, y serán ignorados. Es decir podemos dejarnos mensajes dentro del código,(que de otro modo darían errores). El compilador ignorará cualquier cosa entre // y el fin de línea.

2.5 Primeras instrucciones en Arduino C++.

Parece obligado en el mundo **Arduino**, que el primer programa que hagamos sea el blinking LED, y está bien porque ilustra algunas ideas interesantes en cuanto a sus posibilidades:

- La capacidad de Arduino para interactuar con el mundo externo. Algo bastante inusitado para quienes estén acostumbrados a la informática tradicional, donde la potencia de cálculo ha crecido de forma espectacular, pero sigue siendo imposible (o casi), influir en el mundo exterior.
- La sencillez del entorno de trabajo. En contraposición a un sistema tradicional de editor / compilador / linker.

Arduino puede relacionarse de diferentes maneras con el mundo que le rodea, Empezaremos por los pines digitales que pueden usarse como:

- Entradas: Para leer información digital del mundo exterior.
- Salidas: Para activar una señal al mundo exterior.

Arduino dispone de 14 pines que pueden ser usados de este modo, numerados del 0 al 13:

En la sesión anterior cargamos un programa de ejemplo que hacia parpadear un LED en la placa con una cadencia definida. Veamos como programar esto.

Pediremos a Arduino que active su pin 13 como de salida digital y después encenderemos y apagaremos esta señal lo que hará que el LED que tiene conectado de serie se encienda o apague al ritmo que marquemos.

Para indicar al sistema que deseamos usar el pin 13 como salida digital utilizamos la instrucción:

```
pinMode ( 13, OUTPUT ) ;
```

El primer parámetro indica el pin a usar y “OUTPUT” es para usarlo como salida, y también podría usarse el valor “INPUT” para indicar que vamos a leer de este pin.

Estas definiciones se harán solo una vez al principio, en la función setup(). La nuestra quedará, con una única instrucción que declara que vamos a usar el pin 13 como salida digital:

```
void setup()
   {
      // initialize the digital pin as an output
      pinMode( 13, OUTPUT) ;
   }
```

- Es importante fijarse en que a pesar de ser una única instrucción, hemos delimitado el bloque de esta función mediante abrir y cerrar llaves.
- Obsérvese que la instrucción finaliza en “ ;” . C++ obliga a acabar las instrucciones con un punto y coma que delimite la orden. Si se omite generará un error.

Para encender el LED usaremos la instrucción:

```
digitalWrite( 13 ,  HIGH) ;
```

Y otra instrucción similar que le ordena apagarlo:

```
digitalWrite( 13 ,  LOW) ;
```

El 13 indica el pin a utilizar y HIGH, LOW indican el valor que deseamos poner en esa salida, que en Arduino corresponden a 5V para HIGH y 0V para LOW.

- Si en la función loop() escribiéramos estas dos instrucciones seguidas, Arduino cambiaría estos valores tan deprisa que no percibiríamos cambios, así que necesitamos frenarle un poco para que podamos percibir el cambio.

Para hacer este retraso de, digamos, un segundo, utilizaremos:

```
delay(1000) ;          //  delay(n) “congela” Arduino n milisegundos
```

Por tanto para programar una luz que se enciende y se apaga, tendríamos que generar una secuencia de órdenes (*Como en una receta e cocina*) que hicieran:

1. Informar a Arduino de que vamos a utilizar el pin13 para escribir valores(en el Setup).
2. Encender el LED : Poner valor alto (5V) en dicho pin.
3. Esperar un segundo.
4. Apagar el LED: Poner valor bajo (0V) en dicho pin.
5. Volver a esperar un segundo.
    - *Si omitiéramos este segundo retraso, apagaría la luz y volvería a empezar encontrándose la orden de volver a encender. No apreciaríamos que se había apagado.(No espero que me creáis. Comprobadlo).*
    - *El procesador de Arduino UNO es muy lento desde el punto de vista electrónico, pero es capaz de conmutar la luz(pasar de encendido a apagado y vuelta a encender) unas 15.000 veces por segundo*.

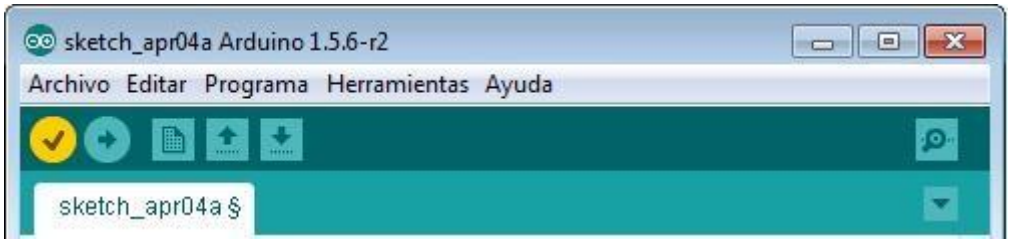
El primer concepto que tenéis que fijar, es que los ordenadores procesan las ordenes en secuencia, una instrucción después de otra y en el orden en que se las dais. *Nuestro programa instruye al ordenador para que ejecute esas instrucciones y fija el orden en el que se ejecutan.*

La forma de escribir un programa en Arduino C++ que haga lo anteriormente descrito es algo parecido a esto

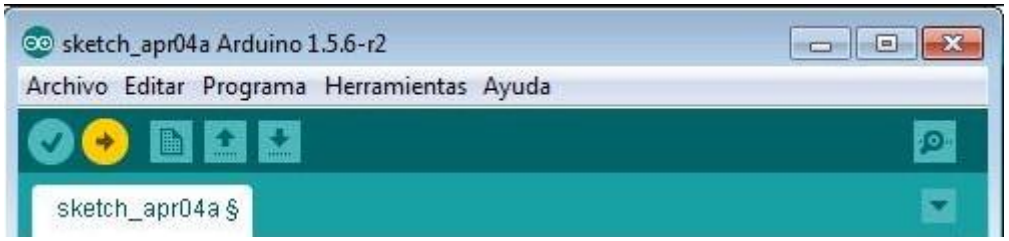
```
void setup()
   {
         pinMode( 13 , OUTPUT); // Usaremos el pin 13 como salida
   }


void loop()
   {
     digitalWrite(13 , HIGH); // Enciende el LED
     delay(1000); // Esperar un segundo
     digitalWrite(13 , LOW); // Apagar el LED
     delay(1000); // Esperar otro segundo
  }
```

- Nótese el sangrado de las líneas para destacar los bloques de código. Esto se considera buena práctica y os lo recomendamos encarecidamente, porque facilita mucho la comprensión del programa.
- Cuando os equivoquéis (y creadme, os vais a equivocar) el sangrado ayuda,y mucho, a visualizar el programa.
- Solo hay dos tipos de programadores. Los que se equivocan y los que se van a equivocar

Solo nos falta ya, comprobar si hay errores y para ello pulsamos el icono en amarillo:



Si todo va bien,(si no hay errores en rojo) podemos compilar y volcar con la siguiente flecha, En caso contrario (y creedme que os pasará con frecuencia) habrá que revisar los posibles errores y corregirlos. Volveremos sobre esto en el futuro.



La flecha en amarillo volcara nuestro programa al Arduino y podremos comprobar que la luz del pin 13 parpadea con un retraso de un segundo entre encendido y apagado.

- Sugerencia: Si modificamos los valores del delay, modificaremos la cadencia del parpadeo.
- Nota: Esto no funcionara con ningún otro Pin del Arduino UNO, porque solo el 13 tiene un LED conectado.

2.6 Resumen de la sesión

En esta sesión hemos aprendido varias cosas importantes:

- El concepto clave de un programa, como secuencia de instrucciones que se ejecuta en el orden marcado.
- Hay dos funciones básicas en todo programa Arduino: Setup() y Loop()..
- Para delimitar un bloque de instrucciones usamos apertura y cierre de llaves..
- Todas las instrucciones acaban en punto y coma (Aunque hay excepciones)…
- Podemos usar comentarios usando // .
- Hemos aprendido algunas instrucciones iniciales del Arduino C++.