Sesión 6: Programando el cerebro del robot

En esta sesión aprenderemos cómo los robots reciben y ejecutan instrucciones mediante la programación. Exploraremos el microcontrolador Arduino UNO, entenderemos cómo se comunica con el entorno a través de sus pines, y descubriremos cómo los algoritmos son la base del pensamiento computacional para resolver problemas reales en nuestra comunidad.

01

El cerebro del robot

Qué es un microcontrolador y cómo controla cada acción del robot.

02

Arduino UNO

La plataforma de hardware y software de código abierto más usada en robótica educativa.

03

Pines y comunicación

Cómo Arduino recibe y envía información mediante pines de entrada y salida.

04

Algoritmos y programación

Cómo organizar instrucciones lógicas para que el robot resuelva problemas reales.

¿Quién controla un robot?

El cerebro humano vs. el cerebro del robot

Así como el cerebro humano coordina nuestros movimientos, pensamientos y decisiones, los robots necesitan un sistema que controle cada una de sus acciones. Este sistema recibe información, la analiza y decide qué hacer en cada momento.

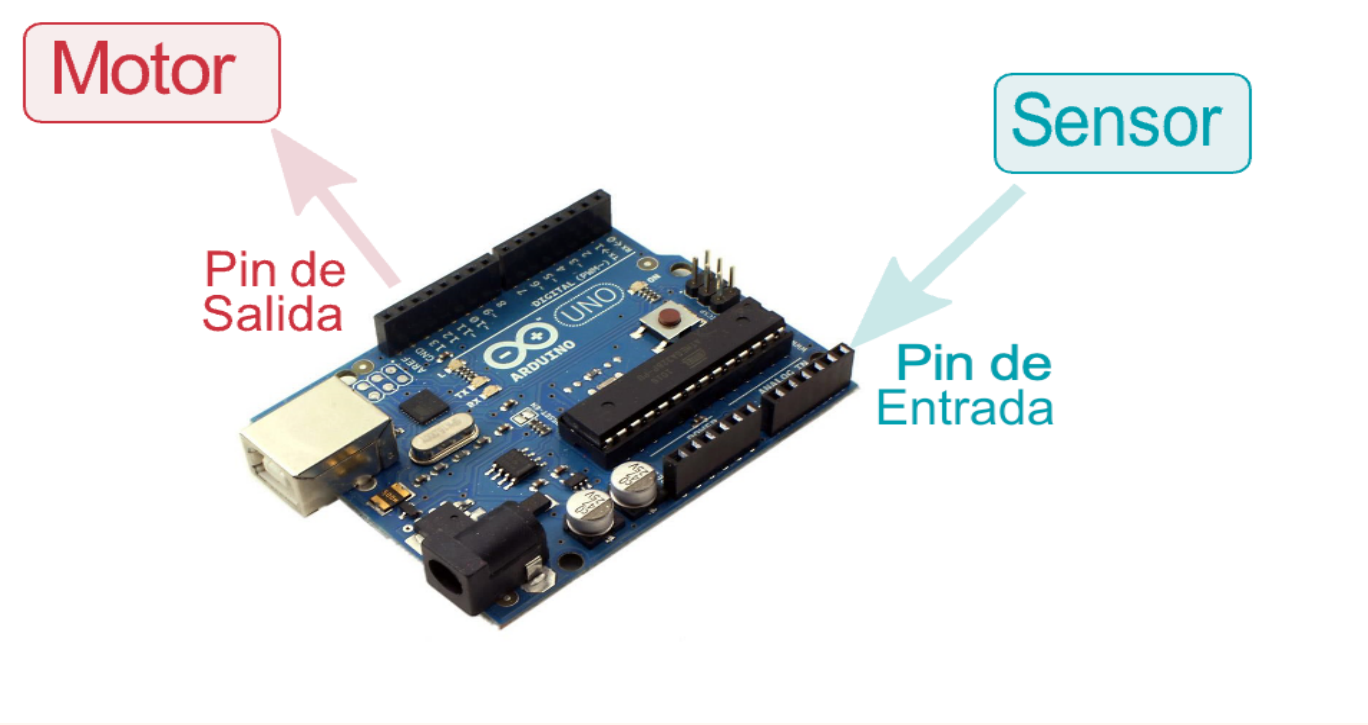
En robótica educativa, ese "cerebro" es el **microcontrolador**: un pequeño dispositivo electrónico capaz de ejecutar miles de instrucciones en muy poco tiempo. Sin un programa que le indique qué hacer, un robot no puede realizar ninguna tarea.

¿Qué puede hacer un microcontrolador?

Un microcontrolador es un circuito integrado diseñado para controlar dispositivos electrónicos. Aunque su tamaño es muy pequeño, tiene la capacidad de recibir información, procesarla y enviar órdenes a diferentes componentes del robot. Gracias a él, un robot puede:

- Leer sensores y detectar obstáculos.
- Encender luces LED y emitir sonidos.
- Controlar motores y tomar decisiones sencillas.

En otras palabras, el microcontrolador coordina el funcionamiento de todos los elementos que forman parte del robot.



Arduino: una plataforma para crear

Uno de los microcontroladores más utilizados en robótica educativa es **Arduino UNO**. Arduino es una plataforma de hardware y software de **código abierto** que permite diseñar y programar proyectos tecnológicos de manera sencilla. Su facilidad de uso la ha convertido en una herramienta muy popular en escuelas, universidades y centros de investigación de todo el mundo. La principal ventaja de Arduino es que permite experimentar, crear y aprender mediante proyectos reales con impacto en la comunidad.

Sistemas de riego automático

Detectan la humedad del suelo y activan el riego solo cuando la planta lo necesita.

Semáforos inteligentes

Controlan el flujo vehicular y peatonal de forma automatizada y eficiente.

Robots móviles

Navegan de forma autónoma, detectan obstáculos y realizan tareas programadas.

Estaciones meteorológicas

Miden temperatura, humedad y otros datos climáticos en tiempo real.

Automatización del hogar

Controlan luces, puertas y electrodomésticos de manera inteligente.

Medidores de humedad

Analizan las condiciones del suelo para optimizar el uso del agua en cultivos.

¿Cómo se comunica Arduino con el mundo?

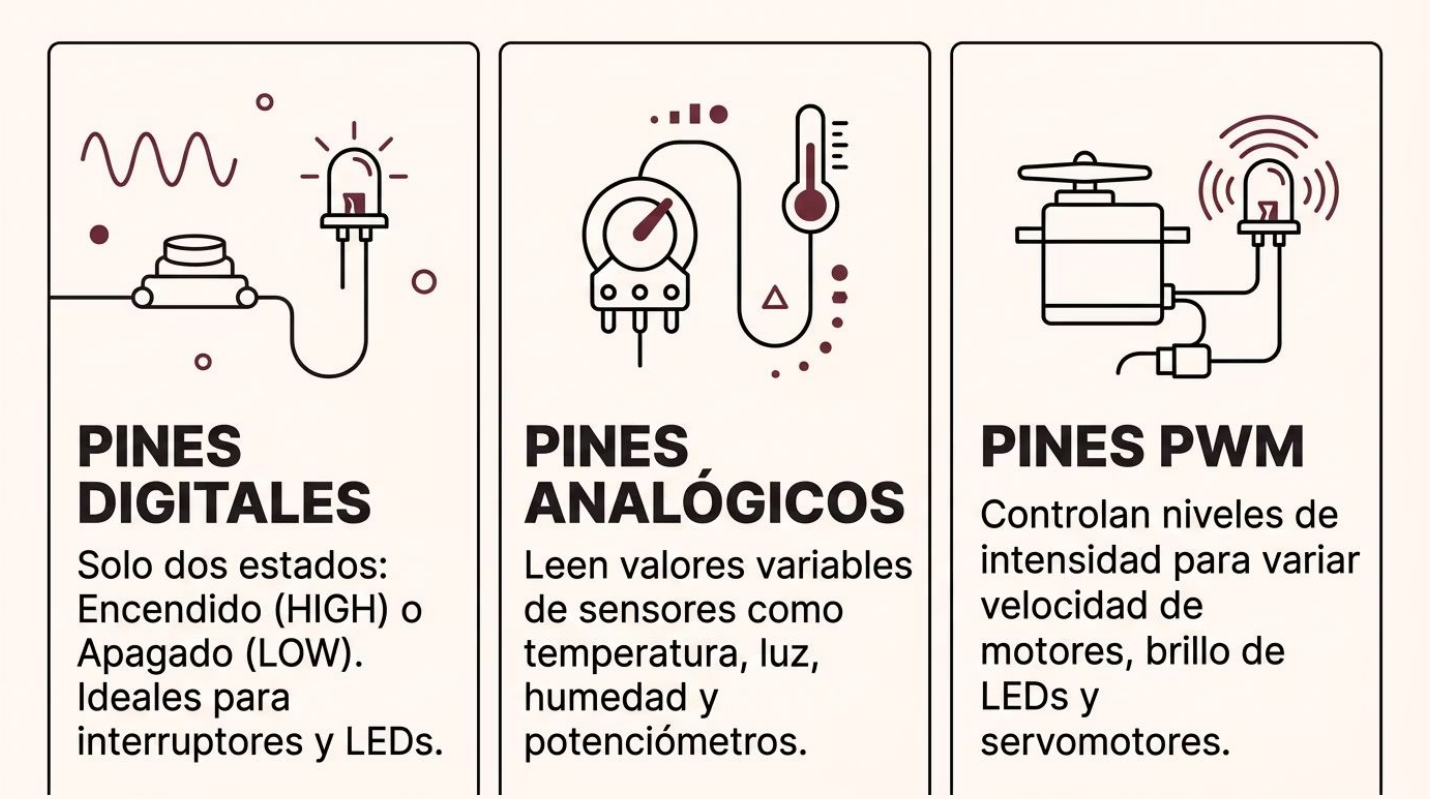
Para que un robot pueda interactuar con su entorno necesita recibir y enviar información. Arduino lo hace mediante pequeños conectores llamados **pines**. Cada pin cumple una función específica y permite conectar sensores, motores, luces LED, botones y otros dispositivos electrónicos. Los pines funcionan de dos maneras fundamentales: como entradas o como salidas.

Pines de Entrada (INPUT)

Reciben información del entorno para que Arduino tome decisiones. Ejemplos: botones, sensores de luz, temperatura y distancia.

Pines de Salida (OUTPUT)

Envían órdenes hacia otros dispositivos. Ejemplos: encender un LED, activar un motor, emitir un sonido o mover un servomotor.



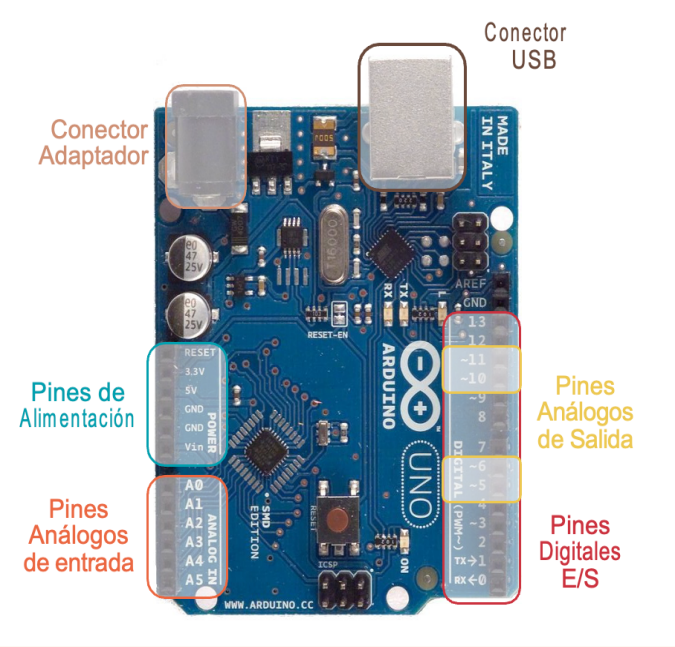
¿Cómo piensa un robot? Algoritmos y lenguajes de programación

Aunque los robots parecen inteligentes, en realidad ejecutan instrucciones que una persona programó previamente. Antes de construir un programa debemos organizar las acciones paso a paso. A esa secuencia ordenada de instrucciones la llamamos **algoritmo**. Un algoritmo puede compararse con una receta de cocina: cada instrucción debe aparecer en el orden correcto para que el robot funcione adecuadamente.

Ejemplo de algoritmo: robot en un cultivo

1. Encender el robot.
2. Leer el sensor de distancia.
3. Si encuentra un obstáculo, detenerse.
4. Girar hacia la derecha.
5. Continuar avanzando.

> Lo más importante no es memorizar comandos, sino aprender a pensar de manera lógica para diseñar soluciones a problemas reales.



Los lenguajes de programación

Los robots no entienden el idioma que hablamos diariamente. Por esta razón utilizamos **lenguajes de programación**, que traducen nuestras instrucciones a un lenguaje que el robot puede interpretar. Arduino utiliza principalmente el lenguaje **C++**, aunque en robótica educativa también existen plataformas de programación por bloques que facilitan el aprendizaje de quienes están iniciando.

Programar para transformar la comunidad

La programación no consiste únicamente en escribir código. También implica observar problemas, analizar información y diseñar soluciones útiles. Con Arduino es posible desarrollar sistemas automáticos de riego, monitoreo de agua en tanques, alimentadores para animales, estaciones climáticas escolares y clasificadores de semillas. Cada programa representa una oportunidad para utilizar la tecnología con un propósito social.

Actividad práctica y reflexión final

Antes de programar un robot, primero debemos aprender a organizar instrucciones. Imagina que vas a programar un robot para regar automáticamente un cultivo: escribe un algoritmo con los pasos que debe seguir. Luego reflexiona sobre cómo programarías un robot para encender una luz cuando oscurece, abrir una puerta al detectar una persona, o regar una planta únicamente cuando el suelo esté seco.

1

¿Cuál es la función del microcontrolador?

Recibe información del entorno, la procesa y envía órdenes a los componentes del robot para coordinar todas sus acciones.

2

¿Qué diferencia hay entre pines de entrada y salida?

Los pines de entrada reciben información del entorno (sensores, botones), mientras que los de salida envían órdenes a dispositivos (LEDs, motores).

3

¿Qué es un algoritmo?

Una secuencia ordenada de instrucciones que el robot debe seguir paso a paso para realizar una tarea correctamente.

4

¿Cómo podría Arduino solucionar un problema de tu comunidad?

Diseñando proyectos como sistemas de riego, estaciones climáticas o alimentadores automáticos que beneficien a las personas.

Lo que aprendí en esta sesión

La programación es una habilidad que puede aprender cualquier persona cuando cuenta con oportunidades, acompañamiento y práctica constante. En robótica educativa, niñas y niños participan en igualdad de condiciones, aportando ideas, diseñando algoritmos y construyendo circuitos. **Las grandes ideas nacen cuando todas las voces tienen la oportunidad de participar.**

☐ Microcontrolador

Puedo explicar qué es y cuál es su función en un robot.

☐ Arduino UNO

Reconozco sus funciones principales y los tipos de pines que utiliza.

☐ Algoritmos

Comprendo qué es un algoritmo y puedo diseñar instrucciones lógicas para resolver un problema.

☐ Impacto social

Valoro la programación como herramienta para crear soluciones tecnológicas que beneficien a mi comunidad.

Iniciando la Programación

Luego de la instalación debemos abrir el programa y nos aparecerá la siguiente interfaz. Dentro de la parte blanca de la interfaz es donde escribiremos las instrucciones que le vamos a dar al robot.

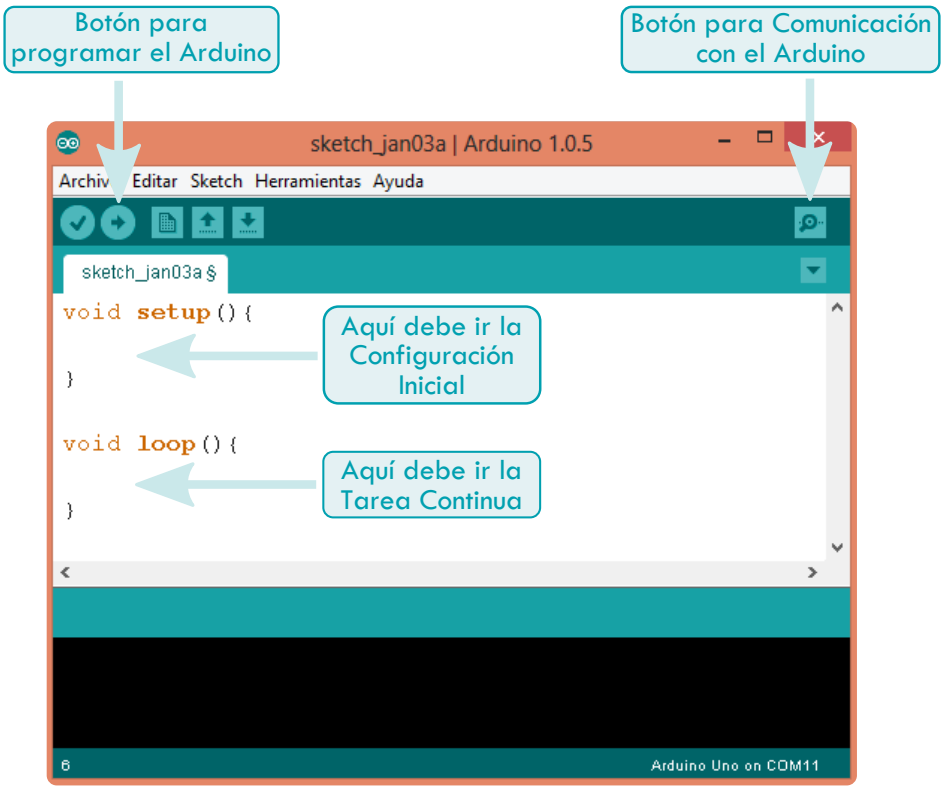


Figura 1.4: Interfaz de Programación Arduino

¿Que es `void setup() {}` ?

Por el momento diremos `void setup() {}` es una palabra clave del lenguaje de programación que sirve para que dentro de las llaves ({}) escribamos las ordenes de inicialización del Arduino. Estas órdenes solo se ejecutan una vez después de haber encendido el robot.

¿Que es `void setup() {}` ?

También es una palabra del lenguaje de programación que sirve para que dentro las llaves ({}) se escriban las ordenes que queramos que el Arduino realice repetidamente hasta ser apagado o reseteado.

Primer Programa: Encender un LED

Ya es hora de que escribamos nuestro primer programa para la tarjeta Arduino, así que empezaremos por lo mas sencillo que es encender el LED que incluye nuestra tarjeta (conectado

al pin número 13).

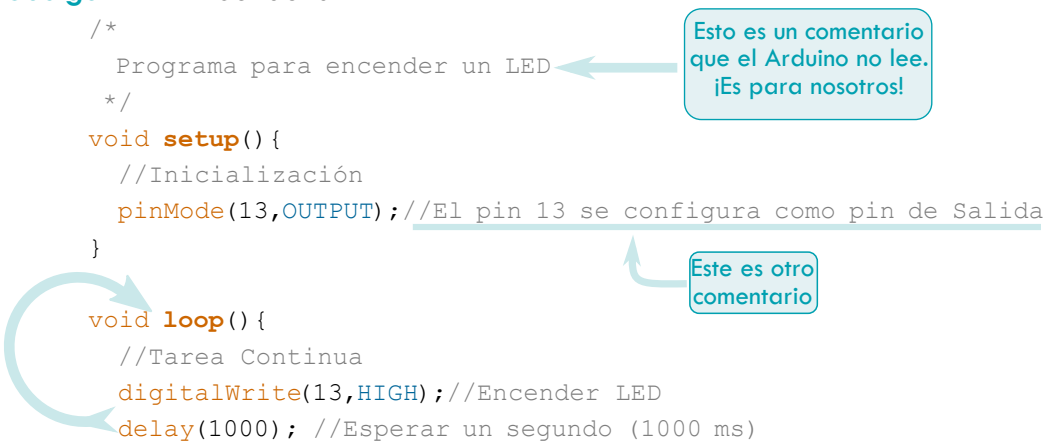
Lo primero que haremos será definir el modo de trabajo del pin 13 como un pin de salida en la configuración inicial y luego en la tarea continua lo encenderemos. Para hacer esto debemos escribir el codigo que se muestra a continuación (Código 1.1).

Código 1.1 — Encender un LED

```
/*
  Programa para encender un LED
 */
void setup(){
  //Inicialización
  pinMode(13,OUTPUT);//El pin 13 se configura como pin de Salida
}

void loop(){
  //Tarea Continua
  digitalWrite(13,HIGH);//Encender LED
  delay(1000); //Esperar un segundo (1000 ms)
  digitalWrite(13,LOW); //Apagar LED
  delay(1000); //Esperar un segundo (1000 ms)

}
```



Al inicio del codigo vemos un slash y un asterisco (`/*`). Esti sirve para dar inicio a comentarios, los cuales no son interpretados por el Arduino, y nos sirven a los programadores para escribir detalles de nuestro codigo. Para finalizar nuestro comentario se usan los caracteres `*/`.

Luego vemos las palabras claves `void setup()`, que como habiamos dicho sirve para escribir dentro de las llaves la configuración inicial. En la linea siguiente vamos dos slash seguidos (`//`). Estos sirven también para hacer comentarios, pero solo en un renglón o línea.

Dendro de las las llaves vemos la instrucción `pinMode(13,OUTPUT)`, la cual configura el pin número 13 como pin de salida. Esto se hace en la configuración inicial para que desde el inicio del programa el pin quede definido como pin de salida. Esta instrucción va seguida de un comentario (`//`) que explica que hace dicha instrucción.

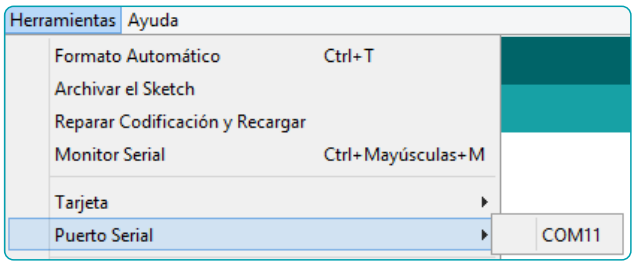
Siguiendo el código encontramos `void loop()` y dentro de las llaves ({}) encontramos la tarea continua que en esta compuesto de cuatro instrucciones. La primera es `digitalWrite(13,HIGH)`, que enciende el led dandole un nivel ALTO (`HIGH`) al pin 13. La segunda es `delay(1000)` que genera un retardo de un segundo, en el cual el led se mantiene encendido. La tercera instrucción es `digitalWrite(13,HIGH)` la cual apaga el led. Finalmente la cuarta instrucción es otro retardo de un segundo para que el led permanezca encendido otro segundo.

Cuando el arduino termina de ejecutar estas instrucción vuelve al inicio de la tarea continua de modo que vuelva a ejecutar `digitalWrite(13,HIGH)`, encendiendo el led, y luego el retardo, y así hasta recorrer toda la tarea continua. Esto se hace infinitamente hasta que la tarjeta sea apagada o reseteada.

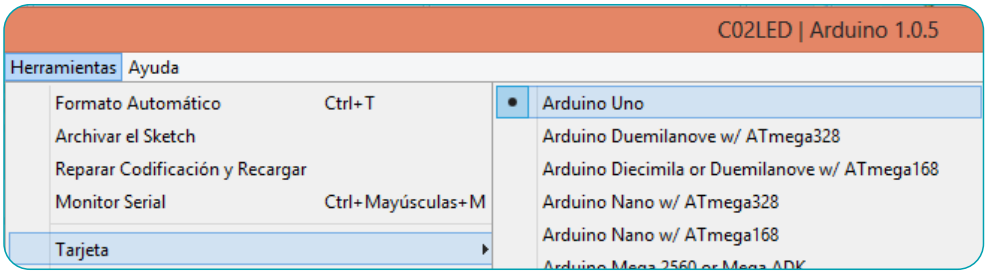
Enviando el programa al Arduino por primera vez

Ahora que entendimos el programa para encender y apagar el led vamos cargar el programa en la tarjeta Arduino siguiendo estos pasos:

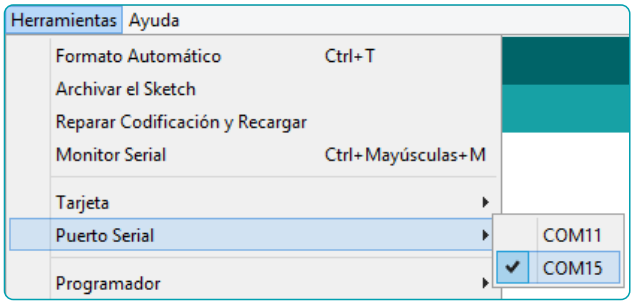
- Antes de conectar la tarjeta, determinemos que puertos de comunicación se encuentran ya ocupados para no seleccionarlos cuando conectemos la tarjeta.



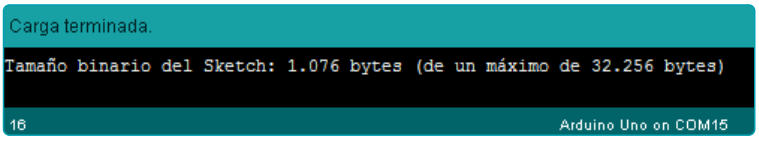
- Conectemos la Tarjeta Arduino al Computador mediante el Cable USB.
- Verifiquemos que esté seleccionada correctamente nuestra tarjeta. (**Arduino UNO**). Vamos a **Herramientas->Tarjeta->Arduino UNO**[2]



- Verifiquemos que el computador reconozca que se ha conectadola tarjeta llendo a **Herramientas->Puerto** miramos cual es el nuevo puerto COM que aparece y lo seleccionamos.



- Ahora podemos dar click en el botón de programar  que se encuentra arriba a la izquierda. Si todo funciona bien en la parte de abajo veremos :



[2] **Tools->Board->Arduino UNO** para las versiones en inglés.

Instrucciónes Básicas

En arduino existen un conjunto básica de instrucciónes veremos algunas de ellas en la Tabla 1.1. Podemos encontrar muchas mas en `http://arduino.cc/es/Reference/HomePage`

Instrucción	Descripción
`pinMode(nPin,Mod)`	`nPin` indica el numero un pin que queramos configurar. `Mod` indica el modo de Trabajo, si es pin de entrada en vez `Mod` escribimos `INPUT` y como pin de Salida escribimos `OUTPUT`.
Ejemplo:	
`pinMode(13,OUTPUT)`	Así se configura el pin número 13 como pin de salida.
`digitalWrite(nPin,Valor)`	Escribe un `Valor` `HIGH` (1) o `LOW` (0) en el pin indicado por `nPin`.
Ejemplo:	
`digitalWrite(2,HIGH)`	En este ejemplo se le da un valor de `HIGH` al pin 2.
`analogWrite(nPin,Valor);`	Escribe un `Valor` entre 0 y 255 en el pin indicado por `nPin`.
Ejemplo:	
`analogWrite(3,150);`	En este ejemplo se le da un valor de `150` al pin 3.
`digitalRead(nPin);`	Lee el valor digital del pin indicado por `npin`. Esta instrucción nos dice si el pin se encuentra en `HIGH` o en `LOW` `nPin`.
Ejemplo:	
`digitalRead(8);`	En este ejemplo se lee el valor del pin 8.
`analogRead(A0);`	Lee el valor analogo del pin indicado por `npin`. Esta instrucción entrega un valor entre 0 y 1024 de acuerdo al voltaje del pin.
Ejemplo:	
`analogRead(A0);`	En este ejemplo se lee el valor del pin `A0`.
`dalay(milisecs);`	Es un retardo que detiene la ejecución del programa por el numero de milisegundos indicados por `milisecs`.
Ejemplo:	
`delay(1000);`	En este ejemplo se hace un ratardo de 1 segundo (1000 milisegundos).
`Serial.begin(9600);`	Sirve para iniciar una comunicación entre el arduino y otros dispositivos como el Computador. El valor de 9600 es la velocidad de la comunicación.
`int Var= 1;`	Crea una variable llamada `Var` y se le da un valor (en este caso "1"). Todas las partes en las que aparezca `Var` tomaran el valor asignado.
Ejemplo:	
`int Val=2;`	En este ejemplo se crea una variable llamada `Val` y se le asigna un valor de 2.

Tabla 1.1: Table caption

Abriendo un ejemplo

Ademas de las referencias de las instrucciones, la misma interfaz de Arduino contiene numerosos ejemplos. Para abrirlos debemos ir a **Archivo->Ejemplos**. Para abrir un ejemplo que tambien prende y apaga el led podemor ir a **Archivo->Ejemplos->Basics->Blink**.

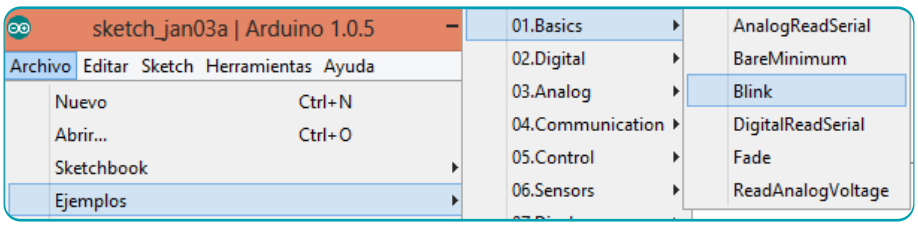


Figura 1.5: Pasos para Abrir un Ejemplo

Variables

¿Cuál es tu nombre?, ¿Cuál es el nombre de tus padres?, ¿Cuántos años tienes?, ¿Cuantos años tienen tus padres? Todas estas respuestas de seguro las sabemos, ya que al igual que un robot tenemos la capacidad de recordar gracias a nuestra memoria. Recordamos el nombre y edades de nuestros conocidos entre otras cosas. Los robots también recuerdan este tipo de información, y la guardan en un espacio de su memoria. Una variable es donde guardamos cierta información. Un ejemplo de variable es mi edad que es 16. Otro ejemplo es mi nombre que es Camilo. En el lenguaje de programación C++ usaremos los siguientes tipos de variables:

Tipo	Codigo	Uso
Entero	`int`	Sirven para almacenar números.
Carácter	`char`	Sirven para almacenar letras.
Palabras o Cadenas de Caracteres	`String`	Sirve para almacenar palabas o incluso Frases.

Ahora veamos un ejemplo:

Código 1.2 — Uso de variables

```
// Declaracion de Variables
String MiNombre="Camilo";
int Edad=16;
//-----------------------
void setup(){
//Iniciar Serial
Serial.begin(9600);
}
void loop(){
 Serial.println("**************");
  Serial.println("Mi Nombre es:");
  Serial.println(MiNombre);
  Serial.println("Mi Edad es");
  Serial.println(Edad);
  delay(10000);
}
```

En este ejemplo en las primeras dos líneas (renglones) podemos ver que se usan una variable tipo `String` para el nombre. Esta variable la llamamos `Minombre` y le asignamos el valor de "Camilo"; también usamos una variable tipo `int` llamada `Edad` a la que le asignamos el valor de `16`.

El resto de las líneas de código se usan para poder visualizar en pantalla las variables a través del puerto serial. [3]

Luego de correr el programa ( ). Vamos a dar click en la Lupa ( ) que aparece en la parte superior derecha.) y nos aparecera la ventana de comunicación serial en la cual se pueden visualizar los datos enviados desde el Arduino.

Podemos cambiar la edad y el nombre y volver a cargar el programa. La idea es jugar con el programa. ¡¡¡¡Eso es lo que haremos la mayor de parte del tiempo con el Arduino!!!

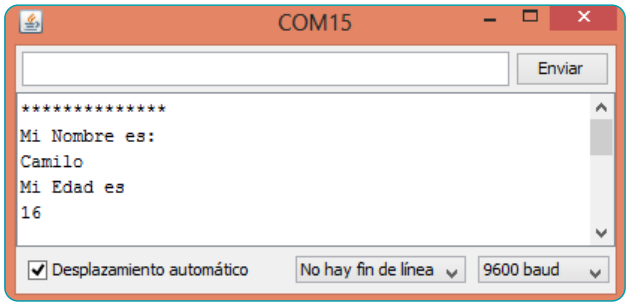


Figura 1.6: Monitor Serial

Tomando decisiones con Arduino

Existen básicamente dos formas en que un robot toma decisiones, una es haciéndose una pregunta de respuesta afirmativa y negativa. Y la otra forma es realizar una tarea de acuerdo al valor de una variable. Veamos esto más a fondo.

Condicionales

Constantemente tomamos decisiones, decidimos si llevar chaqueta, o mejor usar una camisa fresca. Estas decisiones están "condicionadas" por alguno o algunos factores. Por ejemplo la ropa que usamos está condicionada por el clima, y de acuerdo a este tomamos decisiones como las siguientes:

Si está haciendo frío, **entonces** me pongo la chaqueta, **si no** uso una camisa.

Si está lloviendo, **entonces** llevo paraguas, **si no** dejo el paraguas en casa.

Dentro de un algoritmo, este tipo de decisiones las llamamos condicionales. Vemos que en estas primero tenemos una pregunta (**SI**), y de acuerdo a su respuesta se realiza una u otra acción (**entonces**, **sino**). En el lenguaje de programación las condiciones se escriben de la siguiente forma:

[3] El puerto serial es un puerto de comunicación entre dispositivos o robots.

PREGUNTA	SIMBOLO	EJEMPLO	RESPUESTA
Mayor	>	(5>5)	FALSO
Mayor Igual	>=	(4>=4)	VERDADERO
Menor	<	(11<10)	FALSO
Menor Igual	<=	(5<=3)	FALSE
Igual	==	(10==10)	VERDADERO
Diferente	!=	(10!=10)	FALSO

Tabla 1.2: Algunas Relaciones permitidas en C++

Código 1.3 — Condicional

```
if (/*Condición*/) /*Entonces*/{
//Tareas a realizar si verdadero
}
} else //sino{
//Tareas a realizar si falso
}
```

El condicional pueden ser utilizado tanto en la configuración inicial como en la tarea continua, pero será mas común usarlos en la segunda. Veamos un ejemplo en el código 1.4.

Código 1.4 — Condicional Mayoría de Edad

```
int Edad;
//-----------------------
void setup(){
  Serial.begin(9600);
}
void loop(){
  Edad=18;
  if (Edad<18)
  {
  Serial.println("Soy menor de Edad");
  }else{
  Serial.println("Soy mayor de Edad");
  }
  delay(10000);
}
```

Se declara una variable pero no se asigna un valor a esta

Aquí se hace la asignacion. Esta se puede hacer dentro de la configuración inicial o en tarea continua

Puedes correr este codigo y mirar que ocurre al cambiar la edad.

¿Cuales preguntas o condiciones podemos usar?

En los lenguajes de preguntas las preguntas que se hacen esta limitadas a relaciones matemáticas o de igualdad en las que la respuesta solo puede ser **FALSA** o **VERDADERA**. Es decir que al arduino (o incluso a los computadores) no se les hacen preguntas directas sino que las preguntas deben ser formuladas mediante ciertas relaciones matemáticas permitidas por el lenguaje. En el lenguaje C++ se permiten hacer las relaciones de tabla 1.2.

Relación "y"

Alguna vez te han dicho: "Si te portas bien y llegas temprano a casa entonces te daremos más dinero". Esto significa que para que te den más dinero además de "portarte bien", que es la primera condición, tienes que llegar temprano. Usando la relación "y" se tienen que cumplir las

CONDICIÓN 1	RESP	CONDICIÓN 2	RESP	CONDICIÓN	RESPUESTA
(2<=1)	F	(1<1)	F	((2<=1)&&(2<=1))	FALSO
(2==1)	F	(10<=10)	V	((2==1)&&(10<=10))	FALSO
(1<=1)	V	(4!=5)	F	((1<1)&&(4!=4))	FALSO
(4<=3)	V	(1<100)	V	((4<=3)&&(1<100))	VERDADERO
(10<10)	F	(6<7)	V	((10<10)&&(6<7))	FALSO
(11>=10)	V	(3!=4)	V	((11>=10)&&(3!=4))	VERDADERO

Tabla 1.3: Condiciones usando "&&"

CONDICIÓN 1	RESP	CONDICIÓN 2	RESP	CONDICIÓN	RESPUESTA
(2<=1)	F	(1<1)	F	((2<=1)||(2<=1))	FALSO
(2==1)	F	(10<=10)	V	((2==1)||(10<=10))	VERDADERO
(1<=1)	V	(4!=4)	F	((1<1)||(4!=4))	VERDADERO
(4<=3)	V	(1<100)	V	((4<=3)||(1<100))	VERDADERO
(10<10)	F	(6<7)	V	((10<10)||(6<7))	VERDADERO
(10>=11)	F	(3!=3)	F	((11>=10)||(3!=4))	FALSO

Tabla 1.4: Condiciones usando ||

2 condiciones. En el lenguaje C++ para expresar el “y” se usa el símbolo &&. Veamos ejemplos en la tabla 1.3.

Relación “o”

Otras veces te han dicho: “Si te portas bien o llegas temprano a casa entonces te daremos más dinero”. En este caso para que te den más dinero podrías hacer lo siguiente:

- Solo portarte bien.
- Solo llegar temprano a casa.
- Ambas cosas.

De la única forma que no te darían dinero es que no hicieras ninguna de las dos cosas. En el lenguaje C++ expresamos el “o” con el símbolo ||.Veamos Ejemplos en la Tabla 1.4

Selección

Muchas veces la toma decisiones basadas condiciones de respuesta verdadera y falsa no es suficiente, por ejemplo podemos tener diferentes actividades dependiendo del día de semana, ese caso diríamos esto:

En caso que sea **LUNES**:	Practico Futbol.
En caso que sea **MARTES**:	Practico Baloncesto.
En caso que sea **MIERCOLES**:	Practico Ajedrez.
En caso que sea **JUEVES**:	Estudio Ingles.
En caso que sea **VIERNES**:	Estudio Música.
El resto de los días:	Descanso.

Estructuras SWITCH en C++

En el lenguaje de programación C++, existen este tipo de selección de acciones de acuerdo a valor de una variable. Esta variable puede representar una letra o un número o incluso palabras, aunque Generalmente usaremos números. En el Códico 1.5 se ve la forma general de hacer una estructura de casos en C++.

Código 1.5 — Esctrucutura de Casos

```
switch(Variable)
  {
    case  VALOR1:
    //Tarea 1
    break;
    case  VALOR2:
    //Tarea 2
    break;
    case  VALOR3:
    //Tarea 3
    break;
    default:
    //Tarea por defecto
  }
```

Código 1.6 — Ejemplo de Casos con los Días de la semana

```
enum dias {LUNES,MARTES,MIERCOLES,JUEVES,VIERNES,SABADO,DOMINGO};
dias Dia=LUNES;

void setup()
{
  Serial.begin(9600);
}

void loop(){
  switch(Dia)
  {
    case  LUNES:
    Serial.println("Hoy practico Futbol");
    break;
    case  MARTES:
    Serial.println("Hoy Practico Baloncesto");
    break;
    case  MIERCOLES:
    Serial.println("Hoy Practico Ajedrez");
    break;
    case  JUEVES:
    Serial.println("Hoy Estudio Ingles");
    break;
    case  VIERNES:
    Serial.println("Hoy Estudio Música");
    break;
    default:
    Serial.println("Hoy Descanso");
   }
  delay(10000);
}
```

En el Código 1.6 se puede ver que en la primera lína aparece las palabras claves `enum dias {LUNES,MARTES,MIERCOLES....}`, las cuales son una forma decirle al Arduino cuales son los días de la semana[4]. Luego en la segunda línea vemos la instrucción `dias Dia=LUNES`, la cual crea una variable (tipo dias), que nos permite almacenar en ella días de la semana. Luego en la tare continua `void loop` encontramos una estrucutura de casos en la que dependiendo del día de la semana se nos muestra un mensaje. Para correr el codigo debemos dar click en el icono de cargar () y luego en el icono de monitor serial ().

Repitiendo Tareas: Tipos de Ciclos

Muchas veces en nuestros programas vamos a necesitar que se realicen repetidamente ciertas tareas. El numero de repeticiones puede ser constante, o indefinido de acuerdo a ciertas condiciones. Imaginemonos un programa que mientras no reciba una tecla cualquiera, envia mensajes pidiendo que se presione alguna tecla. Otro caso sería que quisieramos hacer un programa que nos envíe 10 mensajes.

Para hacer estas repeticiones se utilizan los cilos. Existen dos grandes clases de estos: El primero se relaciona con el ejemplo de esperar una tecla mientras la condición de Teclar sin presionar se cumpla, este se conoce como ciclo **MIENTRAS**. El segundo caso en el que se ejecutan tareas un numero determinado de veces se conoce como ciclo **PARA**. Estos ciclos se explican a continuación.

CICLO PARA (FOR en Inglés):

Para entender mejor este ciclo veamos su forma de escribirlo en C++.

Código 1.7 — Ciclo PARA (FOR)

```
for(i=0;i<10;i++)
{
//Tareas a Ejecutar
}
```

Este tipo de ciclo ejecuta las tareas escritas dentro de las llaves repetidamente. Cada vez que ejecuta las tareas aumenta el valor de la variable `i` mientras que se cumpla la condición i<10. En este caso la acción entre las llaves se ejecuta 10 veces ya que la acción se ejecuta cuando `i=0` (Una vez),`i=1` (Dos veces),`i=2` (Tres veces),`i=3` (Cuatro veces),`i=4` (Cinco veces),`i=5` (Seis veces),`i=6` (Siete veces),`i=7` (8 Veces),`i=8` (9 veces),`i=9` (Diez veces), pero cuando `i=10` ya no hay ejecución ya que en este caso `i` no es estrictamente menor que `10`.

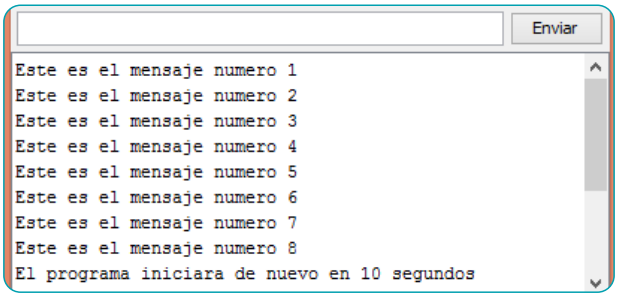


Figura 1.7: Mensajes Repetidos usando `for`

Código 1.8 — Ejemplo FOR

```
void setup(){
  Serial.begin(9600);
}

void loop(){

  for(int i=1;i<9;i++){
      Serial.print("Este es el mensaje numero ");
      Serial.println(i);
   }

  Serial.println("El programa iniciara de nuevo en 10 segundos");
  delay(10000);
}
```

En el ejemplo del ciclo `FOR` se puede ver que a diferencia del ejemplo inicial el primer valor de `i` es `1`. En este caso el mensaje `."Este es el Mensaje Numero"` se imprime 8 veces, que corresponden a `i=1` la primera vez hasta `i=8` la octava vez. Cuando `i=9` ya no hay ejecución y se sigue con el resto del programa, que este caso es consiste en enviar un mensaje anunciando el reinicio del programa en diez segundos. (Mirar Figura 1.7).

CICLO MIENTRAS (WHILE en Inglés):

Obeservemos cuidadosamente su codificación:

Código 1.9 — Ciclo Mientras (WHILE)

```
while(/*Condicion*/)
{
//Tareas a Ejecutar
}
```

Este ciclo ejecuta las tareas descritas en las llaves mientras se cumpla la condición. Si no cumple dicha condición simplemente el programa seguirá derecho sin ejecutar estas líneas de código.

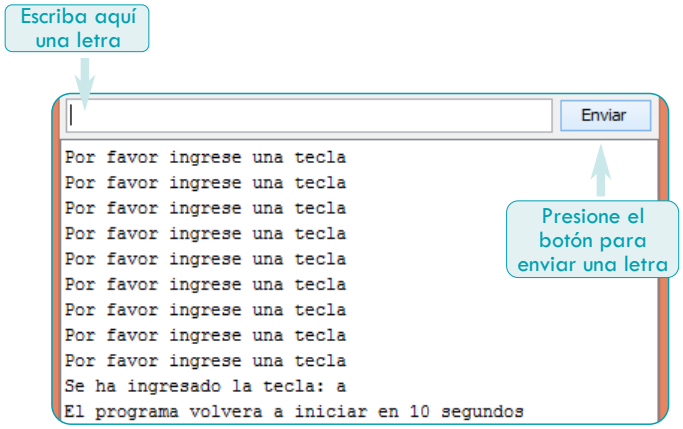


Figura 1.8: Mensajes Repetidos usando `while`

Código 1.10 — Ciclo Mientras (WHILE)

```
char recibido;

void setup(){

  Serial.begin(9600); //Iniciar Comunicación

}

void loop(){

  while(Serial.available()==0){
   //Mientras NO se ingrese un valor se escribe un mensaje
   Serial.println("Por favor ingrese una tecla");
   delay(1000);
    }

  recibido=Serial.read();

  Serial.print("Se ha ingresado la tecla: ");
  Serial.println(recibido);
  Serial.println("El programa volvera a iniciar en 10 segundos");
  delay(10000);
}
```

En el codigo la función `Serial.available()` nos dice el numero de datos disponibles enviados desde el computador al arduino, de manera que si no hay datos nos da un valor de "0". Cuando los datos son leidos usando la función `Serial.read()` el número de datos disponibles vuelve a ser cero hasta que se reciban mas datos.

Luego de correr el ejemplo y abrir el Monitor serial Figura 1.8, vamos a ver que cada segundo llega un mensaje pidiendo que se envíe un letra, o se presione una tecla. Esto ocurre porque la condición se le impuso al ciclo `while` es que `Serial.available()==0` lo cual ocurre siempre que no se envíen datos desde el computador al arduino. Luego se enviamos una letra el numeró de datos disponibles que arroja la función `Serial.available()` ya no será

cero entonces no se ejecutan las instrucción dentro del ciclo `while` continuando así con el resto del programa.

Diagramas de Flujo

SIMBOLO	NOMBRE	USO
	Terminador	Se encuentra al inicio o fin de un proceso. Dentro de él se encuentra información o acciones para iniciar el proceso.
	Proceso	Actividades llevadas a cabo durante el proceso. Puede tener muchas entradas pero una sola salida.
	Entrada y Salida	En su interior están los datos para realizar una actividad.
	Decisión	Punto donde se toma una decisión de “SI” o “NO” dependiendo de la condición interna
	Flechas	Muestra la dirección del flujo de la actividad o proceso.

Tabla 1.5: Símbolos Diagrama de Flujo

Un diagrama de flujo es una representación gráfica de un algoritmo. Nos permite visualizar el algoritmo y probarlo de manera fácil siguiendo el flujo que indiquen las flechas. Es una forma grafica de cómo piensa el robot y el arduino, como se pregunta sobre los cambios de factores externos como sensores y apartir de eso tomar ciertas decisiones. Todo programador antes de sentarse a escribir código, debería plasmar el diagrama de flujo para tener como guía: La lógica, preguntas o condiciones que llevara la inteligencia de su robot. Para poder construir nuestro primer diagrama de flujo debemos conocer los símbolos más usados en su construcción, los cuales se pueden observar en en la Tabla 1.5

Ahora veamos un ejemplo programa en general para un robot, como se muestra en el Diagrama 1.1. Este diagrama consiste en un inicio (Configuración Incial) luego del cual se ejecuta una actividad que recibe ciertos datos para trabajar, luego de esa actividad se toma una decisión cuya respuesta puede desencadenar el fin del programa, o continuar con las actividades del programa.

Los diagramas de flujo se pueden aplicar a cualquier tipo procesos, pueden representar procesos tan simples como controlar el llenado de un tanque (Mirar Diagrama 1.2) hasta procesos tan complejos como el entrenamiento de una Red Neuronal (para ese diagrama aun no estamos preparados).

Diagrama de Flujo 1.1 — Diagrama General

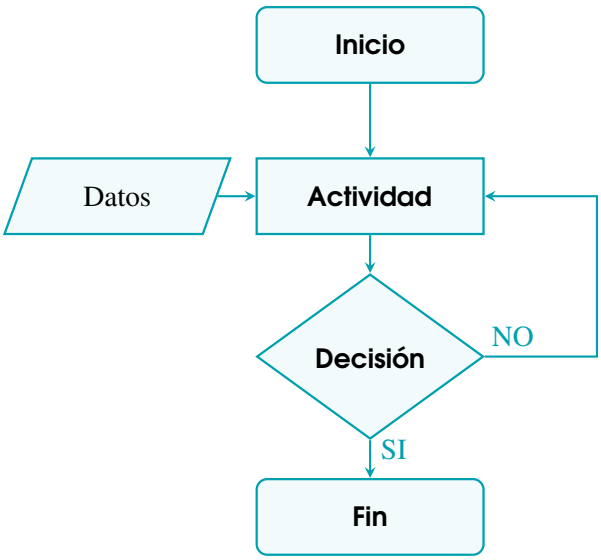


Diagrama de Flujo 1.2 — LLenando un Tanque