

Sensores y Actuadores

Universidad Autónoma de Nuevo León (UANL)

Introducción a Arduino, parte 2



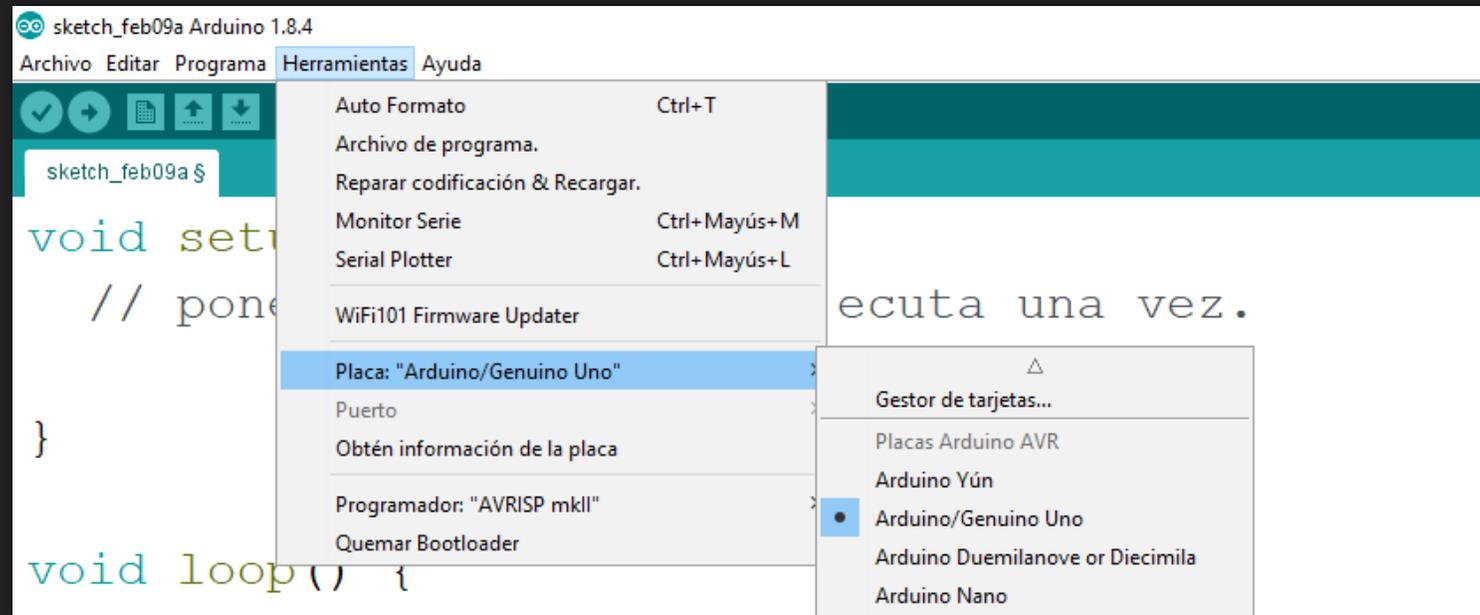
Estructuras principales

*La doble diagonal es para poner comentarios o apuntes de programador

sketch_feb09a §

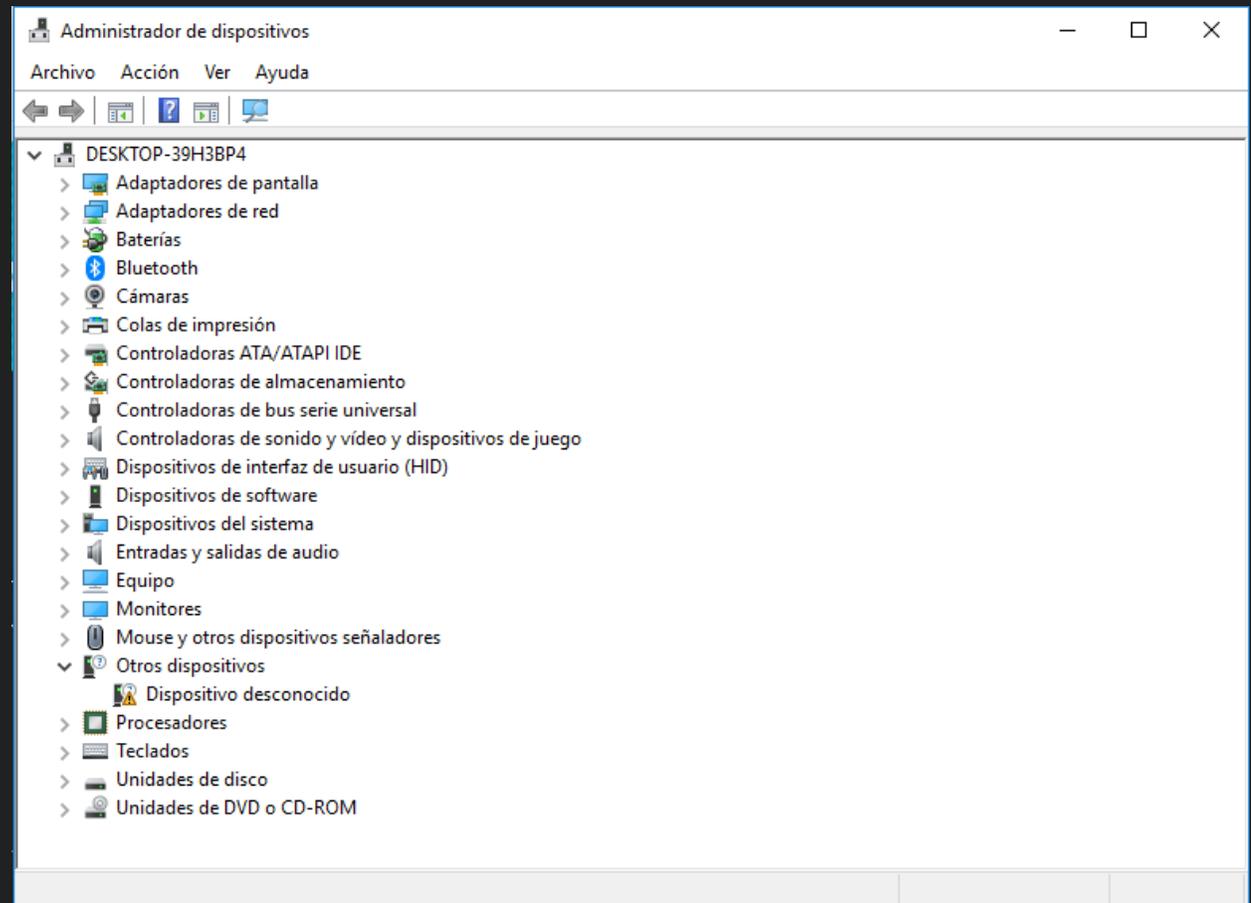
```
void setup() {  
    // poner código que se ejecuta una vez.  
}  
  
void loop() {  
    // poner código que se ejecuta repetidas veces.  
}
```

¿Cuál placa conectaste?



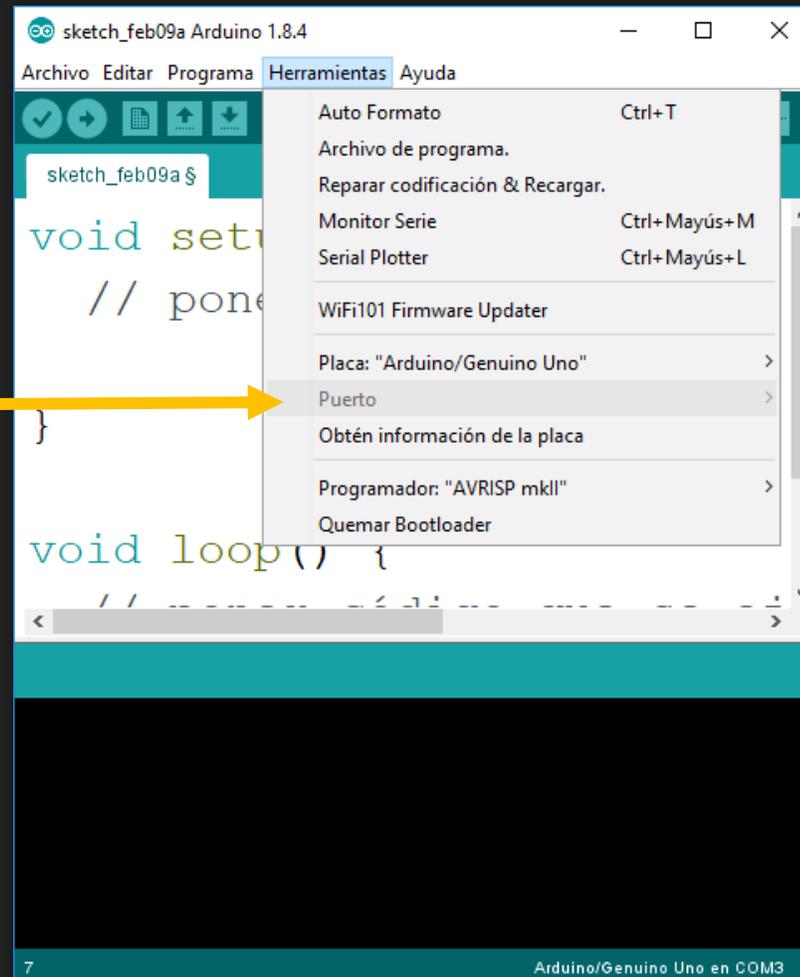
¿En qué puerto?

- Inicio + R
- Escribir devmgmt.msc
- Ver dispositivos conectados



¿En qué puerto?

Seleccionar el puerto donde está conectado el Arduino



Tipos de errores comunes al programar

- De sintaxis (escribir mal una función).
- De operación (hacer una operación inválida).
- De lógica (el programa se ejecuta exitosamente pero no hace lo que el programador desea).

Arduino Cheat Sheet

Arduino Programming Cheat Sheet

Primary source: Arduino Language Reference
<http://arduino.cc/en/Reference/>

Structure & Flow

Basic Program Structure

```
void setup() {  
  // Runs once when sketch starts  
}  
void loop() {  
  // Runs repeatedly  
}
```

Control Structures

```
if (x < 5) { ... } else { ... }  
while (x < 5) { ... }  
for (int i = 0; i < 10; i++) { ... }  
break; // Exit a loop immediately  
continue; // Go to next iteration  
switch (var) {  
  case 1:  
    ...  
    break;  
  case 2:  
    ...  
    break;  
  default:  
    ...  
}  
return x; // x must match return type  
return; // For void return type
```

Function Definitions

```
<ret. type> <name>(<params>) { ... }  
e.g. int double(int x) {return x*2;}
```

Operators

General Operators

= assignment
+ add - subtract
* multiply / divide
% modulo
== equal to != not equal to
< less than > greater than
<= less than or equal to
>= greater than or equal to
&& and || or
! not

Compound Operators

++ increment
-- decrement
+= compound addition
-= compound subtraction
*= compound multiplication
/= compound division
&= compound bitwise and
|= compound bitwise or

Bitwise Operators

& bitwise and | bitwise or
< bitwise xor ~ bitwise not
<< shift left >> shift right

Pointer Access

& reference: get a pointer
* dereference: follow a pointer

Built-in Functions

Pin Input/Output

Digital I/O - pins 0-13 A0-A5
pinMode(pin, [INPUT, OUTPUT, INPUT_PULLUP])
int digitalRead(pin)
digitalWrite(pin, [HIGH, LOW])

Analog In - pins A0-A5

int analogRead(pin)
analogReference([DEFAULT, INTERNAL, EXTERNAL])

PWM Out - pins 3 5 6 9 10 11

analogWrite(pin, value)

Advanced I/O

tone(pin, freq_Hz)
tone(pin, freq_Hz, duration_ms)
noTone(pin)
shiftOut(dataPin, clockPin, [MSBFIRST, LSBFIRST], value)
unsigned long pulseIn(pin, [HIGH, LOW])

Time

unsigned long millis() // Overflows at 50 days
unsigned long micros() // Overflows at 70 minutes
delay(msec)
delayMicroseconds(usec)

Math

min(x, y) max(x, y) abs(x)
sin(rad) cos(rad) tan(rad)
sqrt(x) pow(base, exponent)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)

Random Numbers

randomSeed(seed) // long or int
long random(max) // 0 to max-1
long random(min, max)

Bits and Bytes

lowByte(x) highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn) // bitn: 0=LSB 7=MSB

Type Conversions

char(val) byte(val)
int(val) word(val)
long(val) float(val)

External Interrupts

attachInterrupt(interrupt, func, [LOW, CHANGE, RISING, FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()

Libraries

Serial - comm. with PC or via RX/TX
begin(long speed) // Up to 115200
end()
int available() // #bytes available
int read() // -1 if none available
int peek() // Read w/o removing
flush()
print(data) println(data)
write(byte) write(char * string)
write(byte * data, size)
SerialEvent() // Called if data rdy

SoftwareSerial.h - comm. on any pin
SoftwareSerial(rxPin, txPin)
begin(long speed) // Up to 115200
listen() // Only 1 can listen
isListening() // at a time.
read, peek, print, println, write
// Equivalent to Serial library

EEPROM.h - access non-volatile memory
byte read(addr)
write(addr, byte)
EEPROM[index] // Access as array

Servo.h - control servo motors
attach(pin, [min_us, max_us])
write(angle) // 0 to 180
writeMicroseconds(us) // 1000-2000; 1500 is midpoint
int read() // 0 to 180
bool attached()
detach()

Wire.h - I²C communication
begin() // Join a master
begin(addr) // Join a slave @ addr
requestFrom(address, count)
beginTransmission(addr) // Step 1
send(byte) // Step 2
send(char * string)
send(byte * data, size)
endTransmission() // Step 3
int available() // #bytes available
byte receive() // Get next byte
onReceive(handler)
onRequest(handler)

Variables, Arrays, and Data

Data Types

boolean true | false
char -128 - 127, 'a' '\$' etc.
unsigned char 0 - 255
byte 0 - 255
int -32768 - 32767
unsigned int 0 - 65535
word 0 - 65535
long -2147483648 - 2147483647
unsigned long 0 - 4294967295
float -3.4028e+38 - 3.4028e+38
double currently same as float
void i.e., no return value

Strings

```
char str1[8] =  
{ 'A', 'r', 'd', 'u', 'i', 'n', 'o', '\0' };  
// Includes \0 null termination  
char str2[8] =  
{ 'A', 'r', 'd', 'u', 'i', 'n', 'o' };  
// Compiler adds null termination  
char str3[] = "Arduino";  
char str4[8] = "Arduino";
```

Numeric Constants

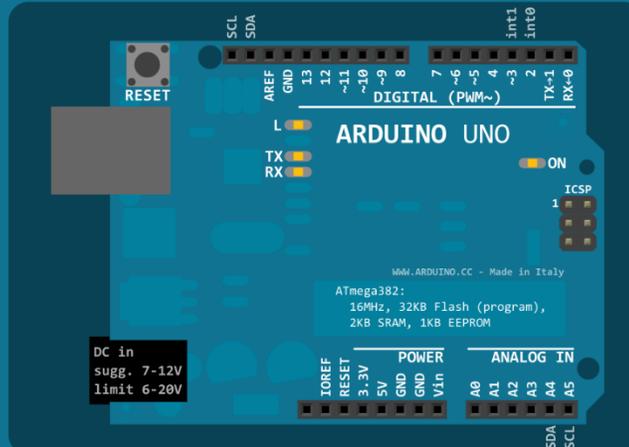
123 decimal
0b01111011 binary
0173 octal - base 8
0x7B hexadecimal - base 16
123U force unsigned
123L force long
123UL force unsigned long
123.0 force floating point
1.23e6 1.23*10⁶ = 1230000

Qualifiers

static persists between calls
volatile in RAM (nice for ISR)
const read-only
PROGMEM in flash

Arrays

```
int myPins[] = {2, 4, 8, 3, 6};  
int myInts[6]; // Array of 6 ints  
myInts[0] = 42; // Assigning first  
// index of myInts  
myInts[6] = 12; // ERROR! Indexes  
// are 0 though 5
```



CC BY SA by Mark Liffiton

Adapted from:
- Original: Gavin Smith
- SVG version: Frederic Dufourg
- Arduino board drawing: Fritzing.org

Funciones básicas

- `pinMode(pin, [INPUT,OUTPUT]);` //INPUT: Arduino recibe datos, OUTPUT: Arduino envía datos
- `digitalWrite(pin, [HIGH,LOW]);` //HIGH: Presencia de voltaje, LOW: Ausencia de voltaje

IMPORTANTE → Cada función termina con punto y coma (;)

Importancia de los retardos

- ¿Cuál es el problema con el siguiente código al momento de encender y apagar el LED?

```
prueba Arduino 1.8.4
Archivo Editar Programa Herramientas Ayuda
prueba $
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  digitalWrite(13, LOW);
}
```

Importancia de los retardos

- ¿Cuál es el problema con el siguiente código?

```
prueba Arduino 1.8.4
Archivo Editar Programa Herramientas Ayuda
prueba $
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  digitalWrite(13, LOW);
}
```

Importancia de los retardos

- Hacen que una función permanezca activa durante un tiempo determinado.

```
prueba Arduino 1.8.4
Archivo Editar Programa Herramientas Ayuda
prueba $
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}
```



Declaración de variables

int led = 13;

Tipo de dato

Nombre

Valor

¿Para que son
útiles las variables?

Importancia de las variables

- Hacen al código más fácil de manejar al momento de hacer cambios. Compare el siguiente programa con el mostrado anteriormente:

```
prueba $
int led = 13;
int retardo = 500;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(retardo);
  digitalWrite(led, LOW);
  delay(retardo);
}
```

Tarea 4 (por equipos)

- Realizar un programa en el cual el led permanezca encendido 1000 milisegundos y apagado 1000 milisegundos. Después, debe permanecer encendido 900 milisegundos y apagado 900 milisegundos. Así sucesivamente hasta llegar a 100 milisegundos encendido y 100 milisegundos apagado. Una vez que se llega a 100 milisegundos empezar de nuevo en 1000 milisegundos.
- Entregar un reporte en el que se especifique como se logró realizar el programa.

Para la siguiente clase...

- Instalar qucs (Quite Universal Circuit Simulator).
- Traer LEDs de diferentes colores y resistencias de 330 ohms o 470 ohms.