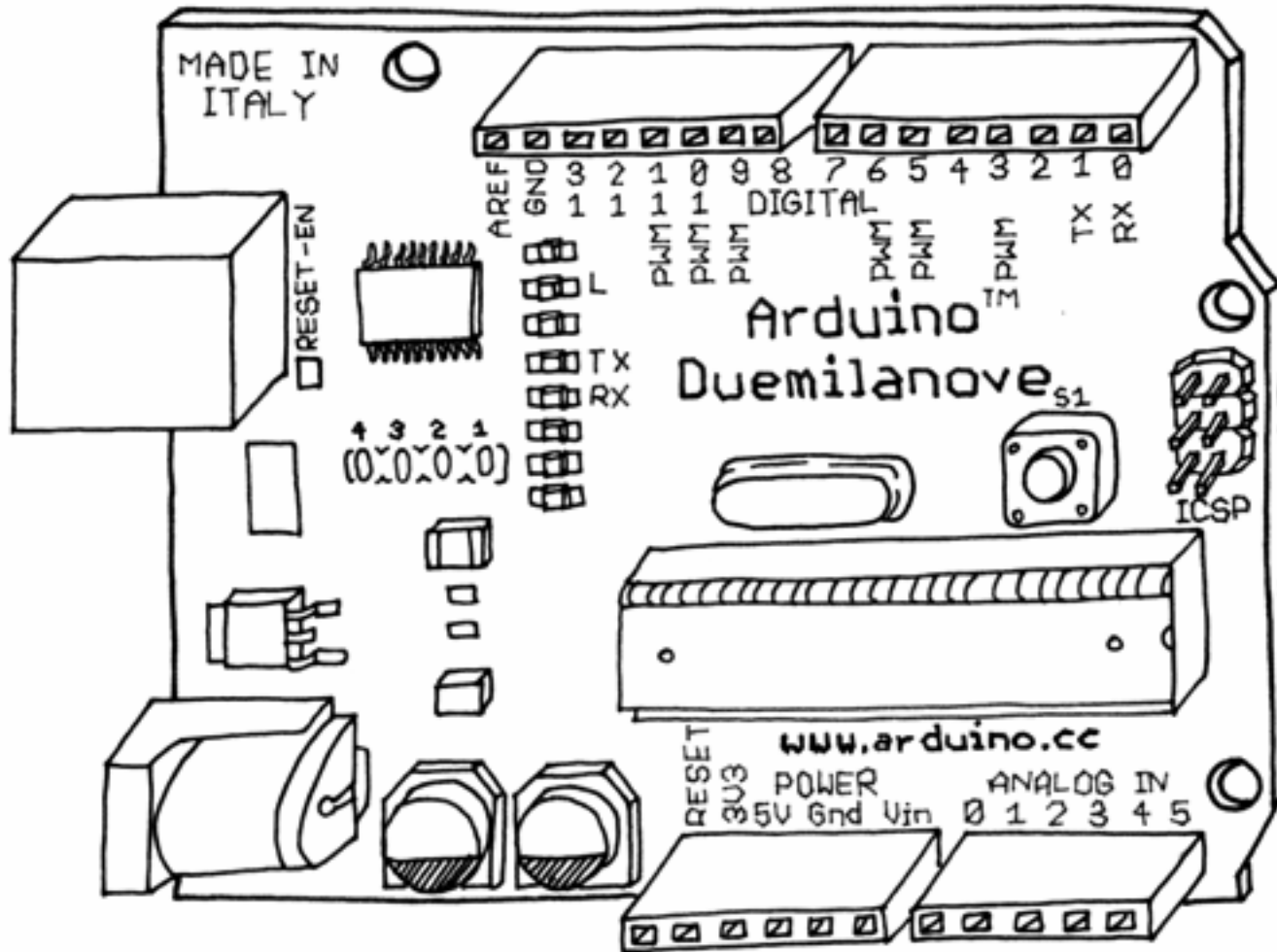


# Arduino



Programación básica

# El lenguaje

- Similar a c/c++
- Sintaxis similar
- Mismos símbolos especiales
  - { } ( ) [ ] ; , “ ‘ + - \* /
- Misma manera de comentar
  - //comenta esta parte
  - /\* comenta esta parte  
y esta otra \*/

# Estructura básica de un programa

- void setup()
  - Esta parte se repite una sola vez
  - Podemos guardar datos de configuración
- void loop()
  - Es el programa principal
  - Se repite una infinidad de veces hasta se indique lo contrario
  - No se puede salir de este ciclo

# Constantes

- Son valores predefinidos dentro del lenguaje, se emplean para hacer mas comprensible la lectura de los programas
  - HIGH | LOW
  - INPUT | OUTPUT
  - true | false

# Variables

- Son localidades de memoria en las cuales se pueden guardar datos
- Una variable debe ser declarada y opcionalmente asignada a un determinado valor
- En la declaración de la variable se indica el tipo de datos que almacenará  
`int minumero;`  
`float pi = 3.141516;`

# Variables

- Pueden ser de diversos tipos
  - Boolean            0 ó 1, HIGH ó LOW, true o false  
    Un solo bit
  - char                todos los ASCII  
    Un byte
  - byte                un número de 0 a 256  
    un byte
  - Int                 desde –32,768 a 32,767  
    dos bytes            unsigned int va de 0 a 65,535
  - long                desde –2,147,483,648 a 2,147,483,647  
    cuatro bytes        unsigned long va de 0 a 4,294,967,295

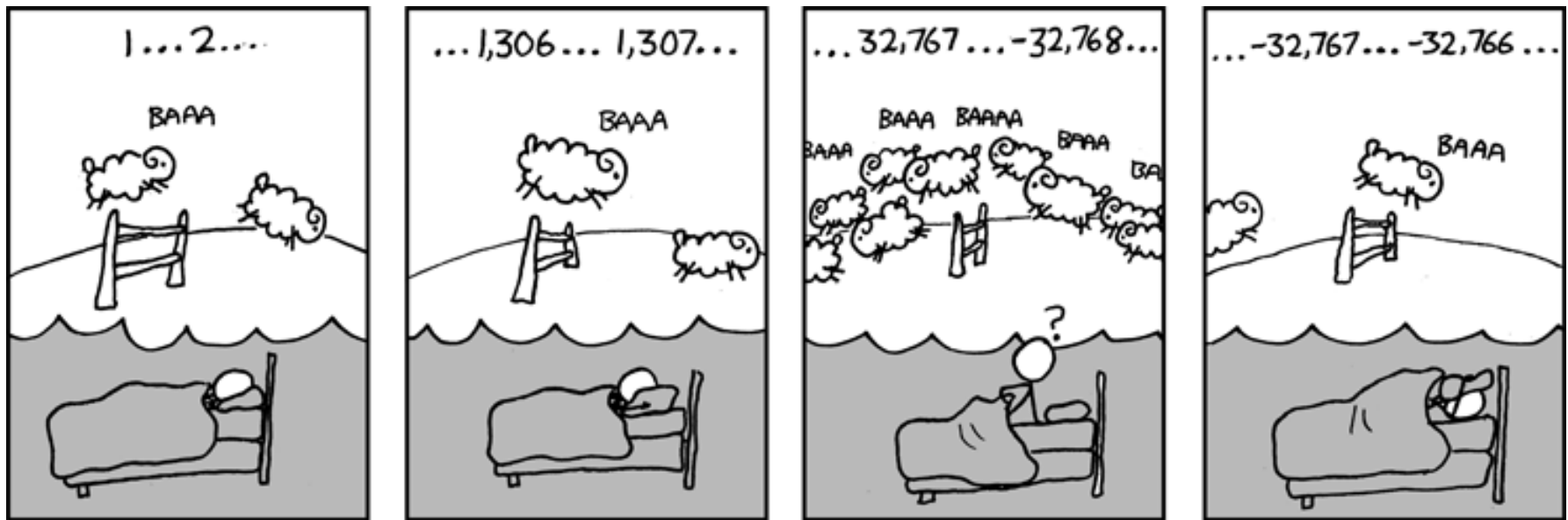
# Variables

- Continuando con la lista
  - float es un número flotante (3.141516)  
consume 4 bytes
  - double número flotante de doble precisión  
Valor máximo de  $1.7976931348623157 \times 10^{308}$
  - string arreglo de chars  
Dependiendo del número de letras es su tamaño
  - array un arreglo de datos del mismo tipo  
`int miarray[] = {dato1, dato2, ..., daton};`  
`Int pines[5] = {2 , 3 , 4 , 5};`

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

# Variables



Si los androides de verdad sueñan con ovejas eléctricas, no hay que olvidar declarar `contOveja` como `unsigned int`

# Estructuras de control

- Son estructuras que manejan el flujo del programa
  - If                      Si condicional  
    if(algunavariabile ?? Algo){ funcion }
  - if...else            Si condicional de lo contrario  
    if(algunavariabile ?? algo) {funcionCondicion }  
    else { funcionCondicionNegada }

# Estructuras de control

- If else if      Si condicional de lo contrario si  
    if(algunaVariable ?? algo) { funcion1 }  
    else if { funcion2 }  
    ...  
    else { funcionN }

# Estructuras de control

– for

```
for(inicializador; condición; incremento)
{ funciónIncrementada }
```

– while

```
while( algunaCondición ) {
    funciónCondicionada }
```

– do... while      similar a while

```
do { funcionCondicionada } while(condicion)
```

– switch case

# Estructuras de control

- break                      sale de la estructura  
    for(i=0;i<10;i++)  
    { if(i ==9) { break;}}
- continue                  continua en la estructura
- return                      usado en funciones  
    Regresa el valor del resultado de alguna  
    función

# Operadores de comparación

- Usados en las sentencias de control

==	igual a
!=	no es igual a
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que



# Operadores aritméticos

- Se pueden crear funciones complejas a partir de ellas

+	suma
-	resta
*	multiplicación
/	división
%	modulo

# Operadores compuestos

- Son combinación de una operación aritmética con una asignación

++	incremento
--	decremento
+=	suma compuesta
-=	resta compuesta
*=	multiplicación compuesta
/=	división compuesta

# Funciones de entrada y salida

- Los pines digitales de Arduino por default están configurados como entradas
  - Como entradas están en un estado de alta impedancia
- Por otro lado el ATMEGA168 y 328 tienen incorporados resistencias de PullUp de  $20k\Omega$ 
  - Se puede acceder a ellos por software
  - Como salidas están en un estado de baja impedancia

# Funciones de entrada y salida

- **pinMode(pin, modo)**
  - El pin es el número de pin físico (0 a 13)
  - Modo es como va a trabajar
    - INPUT
    - OUTPUT

# Funciones de entrada y salida

- **digitalRead(pin)**
  - Lee el estado del pin
  - Si es alto regresa un HIGH
  - Si es bajo regresa un LOW
- **digitalWrite(pin, valor)**
  - Escribe en el pin el valor establecido
    - LOW
    - HIGH

# Funciones de entrada y salida

- Para activar las resistencias de pullup se indica de la siguiente manera

```
pinMode(pin,INPUT);  
digitalWrite(pin,HIGH);
```

Ya que esta definido como una entrada al darle la instrucción digitalWrite un alto activa la resistencia en ese pin

# Funciones de entrada y salida

- **analogRead(pin)**
  - Lee en el puerto analógico valores con el ADC
  - Regresa un valor de 0 a 1024
- **analogWrite(pin, valor)**
  - Escribe un valor de PWM en el pin indicado
    - Valor es el ciclo de trabajo de 0 a 255

# Funciones de entrada y salida

- **analogRead(pin)**
  - Lee en el puerto analógico valores con el ADC
  - Regresa un valor de 0 a 1024
- **analogWrite(pin, valor)**
  - Escribe un valor de PWM en el pin indicado
    - Valor es el ciclo de trabajo de 0 a 255

# Funciones de entrada y salida

- `shiftOut(dataPin, clockPin, bitOrder, value)`
  - Envía datos a un registro serial usa dos pines
    - `dataPin` es por donde se envía los datos
    - `clockPin` aquí se manda una señal de refrescamiento
    - `bitOrder` es el tipo de ordenamiento
      - **MSBFIRST** o **LSBFIRST**
    - `Value` es un byte que se va a mandar
- `unsigned long pulseIn(pin, value)`
  - Mide un pulso en un pin
    - `Value` es el tipo de pulso HIGH o LOW

# Funciones de tiempo

- `unsigned long millis()`
  - Mide el tiempo desde que se inicia el sketch o hasta cuando se resetea millis
- `delay(ms)`
  - Es un atraso se mide en milisegundos
    - Mientras esta activo esta en un NOP
- `delayMicroseconds(us)`
  - Es un atraso se mide en microsegundos
    - Mientras esta activo esta en un NOP

# Funciones matemáticas

- $\min(x, y)$ 
  - Selecciona el valor mínimo entre X ó Y
- $\max(x, y)$ 
  - Selecciona el valor máximo entre X ó Y
- $\text{abs}(x)$ 
  - Regresa el valor absoluto de x
- $\text{constrain}(x, a, b)$ 
  - Rechaza el valor de x si no esta en el rango comprendido entre a y b

# Funciones matemáticas

- `map(value, fromLow, fromHigh, toLow, toHigh)`
  - Mapea a Value de un rango comprendido desde fromLow a fromHigh al rango toLow a toHigh
- `double pow(base, exponent)`
  - Regresa  $\text{base}^{\text{exponent}}$
- `double sqrt(x)`
- `double sin(rad)`
- `double cos(rad)`
- `double tan(rad)`

# Números aleatorios

- `randomSeed(seed)`
  - Asegura una distribución aleatoria de números asignando una semilla nueva
  - Puede usarse con `analogRead` leyendo ruido electromagnético
- `long random(max)` `long random(min, max)`
  - Genera un número aleatorio comprendido desde 0 a max de definirse min el rango va desde min a max

# Comunicación Serial

- `Serial.begin(speed)`
  - Inicializa la comunicación serial
  - El parámetro `speed` indica la velocidad de transmisión
- `Serial.print(data)` `Serial.print(data, encoding)`
  - Imprime un dato en una misma línea
- `Serial.println(data)` `Serial.println(data, encoding)`
  - Imprime un dato en una línea nueva al final retorna el valor `“/n”`
    - En ambos casos `encoding` es como va a ir codificado
      - DEC HEX OCT BIN BYTE

# Comunicación Serial

- `int Serial.available()`
  - Indica si ya se ha establecido la comunicación serial
- `int Serial.read()`
  - Lee byte por byte los datos enviados desde un dispositivo serial
- `Serial.flush()`
  - Limpia el buffer de comunicación serial

# Operaciones con bits

- $\&$  and
- $|$  or
- $\wedge$  xor
- $\sim$  not
- $\ll$  corrimiento a la izquierda
- $\gg$  corrimiento a la derecha

# Funciones de bajo nivel

- sei ( ) ;
  - Habilita las interrupciones
- cei ( ) ;
  - Deshabilita las interrupciones
- sbi ( REGISTER, BIT ) ;
  - Escibe en el registro indicado un bit
- cbi ( REGISTER, BIT ) ;
  - Limpia del registro un bit