Enviar números de más de un dígito a Arduino o varios caracteres

Cómo enviar números de más de un dígito a Arduino o palabras enteras sin que la recepción sea de caracter por caracter.

Introducción

Arduino y la comunicación serie: librería Serial

Fuente: http://mindstormsyarduino.blogspot.com.ar/2011/06/arduino-y-la-comunicacion-serie.htm

Todas las placas de Arduino disponen de al menos un puerto serie, que permite la comunicación con un ordenador o con otros dispositivos haciendo uso de esta interfaz. Los pines que se usan para la comunicación por puerto serie vienen marcados en la placa con las siglas RX y TX, que corresponden al pin por el que se enviarán los datos, y al pin por el que se recibirán los datos respectivamente.

El puerto serie sirve para comunicarnos con un ordenador y para poder comunicarnos con determinados sensores. En este epígrafe se expondrán las funciones necesarias para establecer la comunicación e intercambiar datos.. Ejemplo: Modulo Bluetooth.

Conexiones

Como se ha comentado en la introducción de este epígrafe, se van a usar los pines TX y RX de la placa de Arduino.

NOTA: la placa "Arduino Mega 2560", la cual dispone de 8 pines destinados para ello, en lugar de los 2 de los que disponen el resto de placas. Estos pines vienen marcados en la placa y son las siguientes parejas (0,1), (14,15), (16,17), (18,19) y (20,21).

Para un correcto funcionamiento, se debe conectar el pin de recepción de la placa (RX) al pin de transmisión del dispositivo externo, y el pin de transmisión de la placa (TX) al pin de recepción de dicho dispositivo.

Principales funciones

Las principales funciones usadas en la comunicación por puerto serie vienen incluidas en la librería Serial() y son las siguientes:

- **Serial.begin(velocidad).** Establece la velocidad en bits por segundo (baudios) para la transmisión de datos en serie.

El parámetro velocidad se puede configurar con el valor que se desee para comunicarse con un dispositivo externo (hasta un máximo de 115200 baudios), sin embargo si la comunicación por puerto serie va a ser con un ordenador, se deberá elegir una de las siguientes: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200.

- **Serial.end().** Desactiva la comunicación serie, permitiendo que los pines de recepción y transmisión puedan ser usados como pines de entrada / salida digitales. Si se desee reanudar la comunicación serie, se deberá invocar nuevamente Serial.begin().

- **Serial.available().** Devuelve, ecomo entero, el número de bytes (caracteres) disponibles para ser leídos por el puerto serie. Se refiere a datos ya recibidos y disponibles en el buffer de recepción del puerto, cuya capacidad es de 128 bytes. En el caso de Arduino Mega, existen además las funciones Serial1.available(), Serial2.available() y Serial3.available().

- **Serial.read().** Lee los datos entrantes al puerto serie. Devuelve como entero el primer byte disponible recibido por el puerto serie, o -1 si no hay datos disponibles. Al igual que en el caso anterior, para la placa Arduino Mega existen también las funciones Serial1.read(), Serial2.read() y Serial3.read().

- **Serial.flush().** Vacía el buffer de la entrada de datos serie. Esto quiere decir que si se llama a la función Serial.read() o Serial.available(), estas sólo devolverán los datos recibidos después de que se haya realizado esta llamada. No requiere parámetros ni devuelve nada. Para la placa Arduino Mega existen disponibles las llamadas Serial1.flush(), Serial2.flush() y Serial3.flush() además de la principal.

- **Serial.print(valor, formato).** Esta función imprime los datos al puerto serie como texto ASCII. El primer parámetro, valor, es el propio valor que se desea imprimir. Esto se podrá realizar de distintas formas:

Los números son impresos mediante un juego de caracteres ASCII para cada dígito.

Los valores de tipo flotante (float) se imprimen en forma de dígitos ASCII con 2 decimales por defecto, es decir, si ejecutamos Serial.print(1.23456) se imprimirá "1.23".

Los valores de tipo "byte" se enviarán como un único carácter.

Los caracteres y las cadenas se enviarán tal cual.

Además, existe el segundo parámetro es opcional y permite especificar el formato que se desea usar. Así, este parámetro puede tomar el valor BYTE, BIN (valores binarios, en base 2), DEC (valores decimales, en base 10), HEX (valores hexadecimales, en base 16). Si lo que se está imprimiendo son números con coma flotante, este parámetro especificará el número de decimales a usar.

- **Serial.println(valor, formato).** Imprime los datos al puerto serie como texto ASCII seguido de un retorno de carro(ASCII 13 o '\r') y un carácter de avance de línea (ASCII 10 o '\n'). Por lo demás, este comando tiene la misma forma y los mismos parámetros que Serial.print() descrito anteriormente.

- **Serial.write().** Esta función escribe datos binarios en el puerto serie. Estos se enviarán como un byte o como una serie de bytes. Si lo que se desea es envíar los caracteres que representan los números, es mejor usar la función Serial.print() en su lugar. Puede tomar distintos parámetros:

Serial.write(valor) enviará un solo byte.

Serial.write(str) enviará una cadena como una serie de bytes.

Serial.write(buf, len) enviará un array como una serie de bytes. El tamaño del array se indicará en el segundo parámetro.

Ahora vamos al objetivo de este apunte

Cómo enviar números de más de un dígito a Arduino o palabras enteras sin que la recepción sea de caracter por caracter.

Fuente: http://panamahitek.com/enviar-numeros-de-mas-de-un-digito-a-arduino/

A continuación voy a explicar **dos métodos** para leer números o cadenas de caracteres desde el puerto Serie de Arduino. El primer método será por programación pura y dura *al mejor estilo de la vieja escuela.* El segundo método será preciso y conciso, una abreviación del primer método. Lo hago de esta forma debido a que hay personas que consultan este espacio para aprender sobre programación y resolver sus dudas sobre ciertas funciones del Arduino. El primer método, el cual llamaré **Método Buffer** está dirigido a los entusiastas que les gusta ejercitar su lógica y hacer las cosas a su manera. El segundo método, llamado **Método**

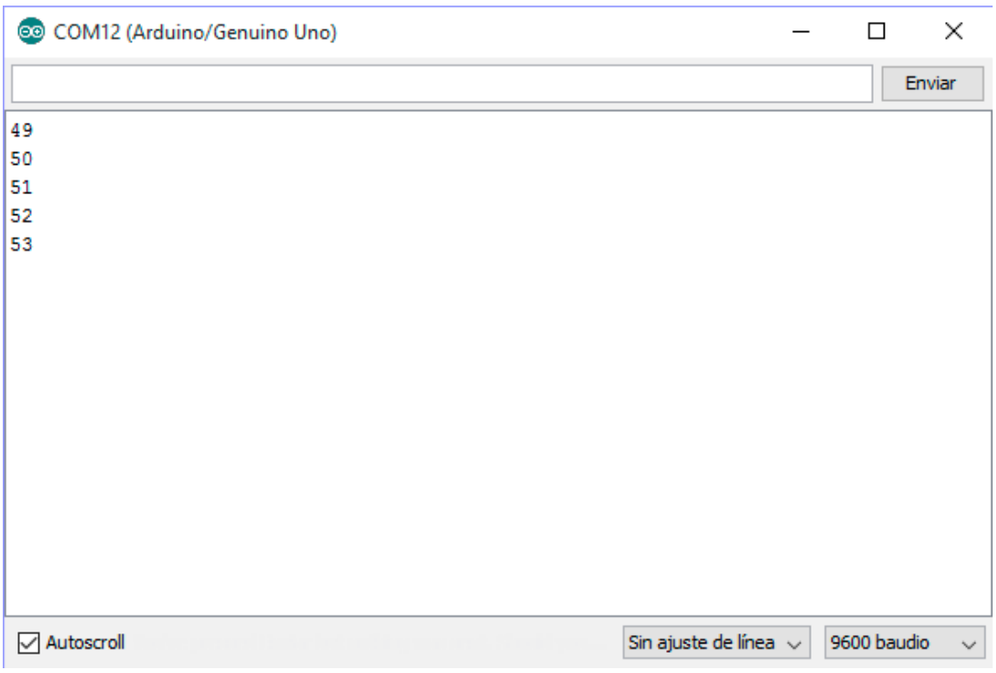
Directo será simplemente una función que el Arduino posee que hará lo mismo que el **Método Buffer** pero mucho más sencillo y simple.

Método Buffer

Resulta que si subimos en nuestro Arduino el siguiente código:

```
void setup() {
  //Iniciamos comunicación con el puerto serie
Serial.begin(9600);

}

void loop() {
/*
 * Evaluamos el momento en el cual recibimos un caracter
 * a través del puerto serie
 */
if (Serial.available()>0){
  //Imprimimos el caracter que recibimos
Serial.println(Serial.read());
}
}
```

Cuando abrimos el Monitor Serie y escribimos la secuencia numérica 1, 2, 3, 4, 5 obtenemos el siguiente resultado:



Lo que se hizo fue:

- 1 (ENTER)
- 2 (ENTER)
- 3 (ENTER)
- 4 (ENTER)
- 5 (ENTER)

En la pantalla se muestra 49, 50, 51, 52 y 53. Estos son los equivalentes de los números del 1 al 5 en la tabla ASCII, como podemos ver en la siguiente imagen:

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad síncr)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

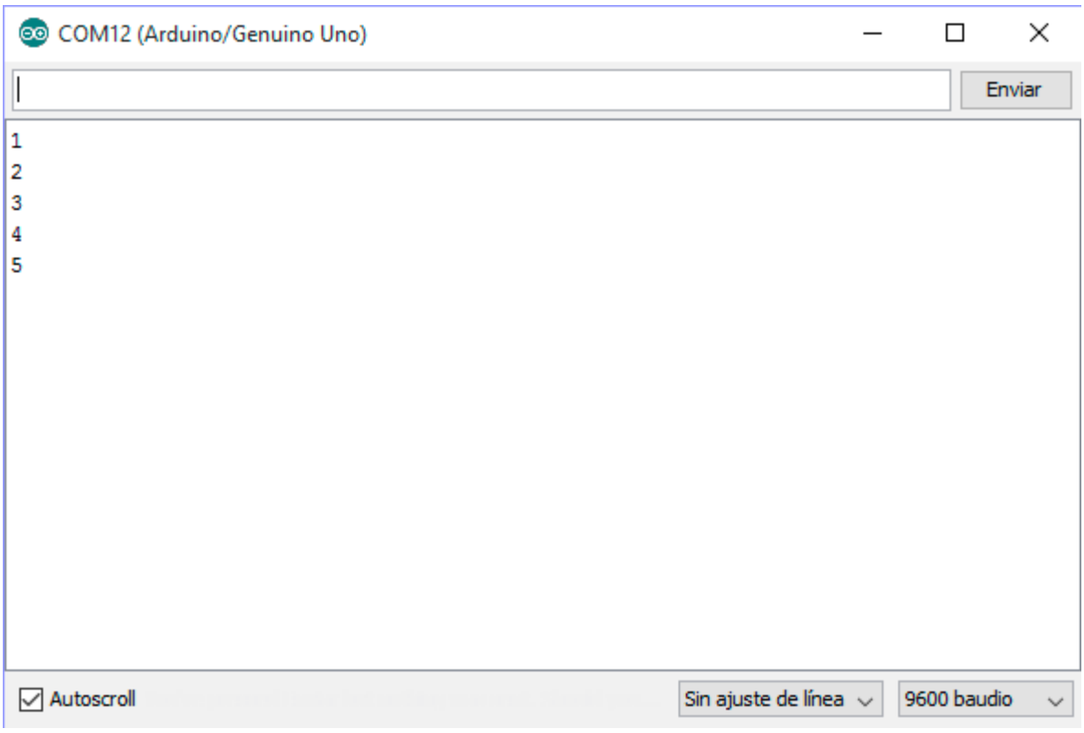
ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

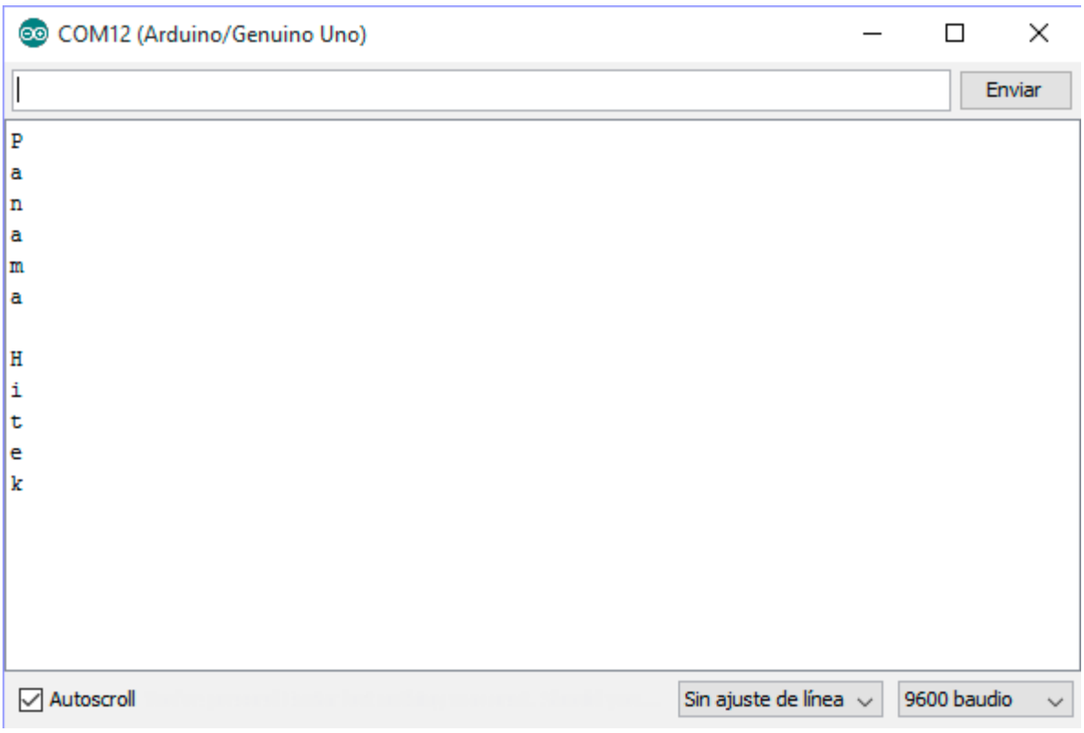
Debo confesar que el lenguaje Arduino me gusta cada día más y es porque siento que a medida que pasa el tiempo y se mejora el propio lenguaje de programación, este se parece más y más a Java, con el cual estoy muy familiarizado. Arduino se derivó de Processing, el cual nació de Java. De hecho el propio software Arduino IDE, donde se programa Arduino, está escrito en Java. Hago este comentario debido a que para corregir el problema de los caracteres en ASCII podemos utilizar un pequeño truco que inicialmente aprendí en Java y que luego me encontré con que en Arduino es posible hacer lo mismo. Una conversión de un entero a caracter utilizando **(char)**. Modifiquemos el código y veamos el resultado de la conversión de número a caracter:

```
void setup() {
  //Iniciamos comunicación con el puerto serie
Serial.begin(9600);

}

void loop() {
/*
 * Evaluamos el momento en el cual recibimos un caracter
 * a través del puerto serie
 */
if (Serial.available()>0){
  //Imprimimos el caracter que recibimos, transformándolo a char
Serial.println((char)Serial.read());
}
}
```

El resultado de la misma operación anterior ahora sera:



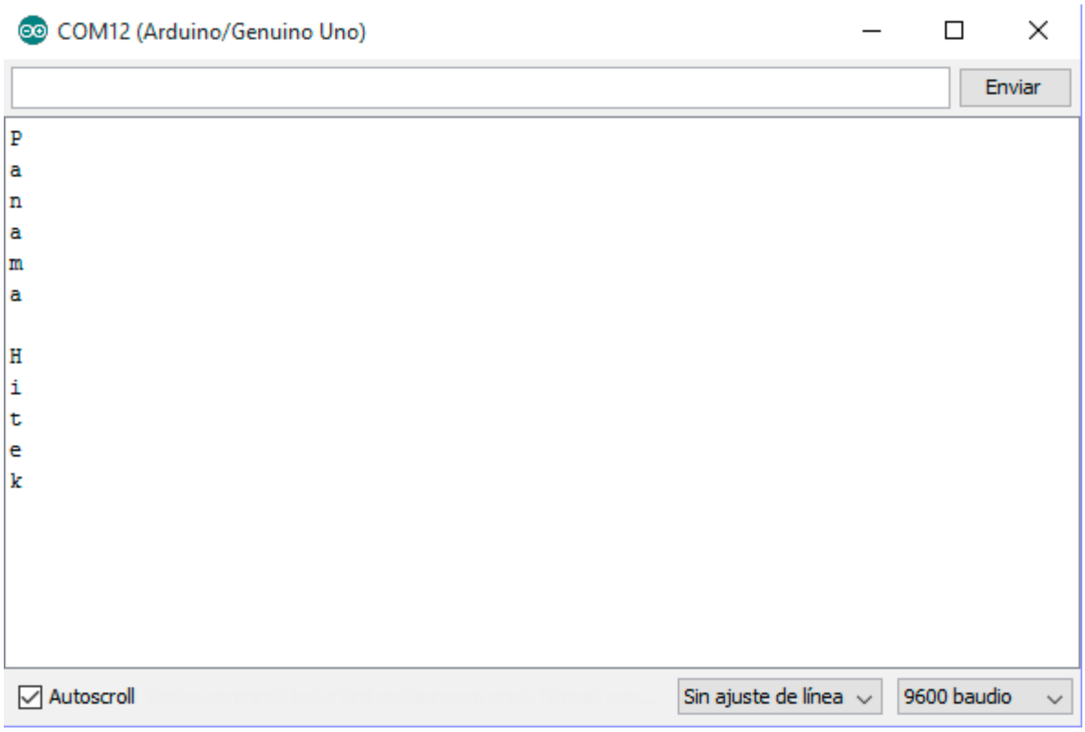
Perfecto. Ahora si nos aparecen los mismos caracteres que escribimos. Pero, ¿que tal si escribimos una palabra o un número de más de un dígito? Escribamos Panama Hitek y veamos que pasa:



Para evitar que esto suceda lo que haremos será crear un buffer que almacene los caracteres que se van recibiendo y luego nos imprima el resultado cuando el buffer acabe de recibir la información.

```
void setup() {
  //Iniciamos comunicación con el puerto serie
  Serial.begin(9600);
}

void loop() {
  /*
   * Evaluamos el momento en el cual recibimos un caracter
   * a través del puerto serie
   */
  if (Serial.available() > 0) {
    //Se crea una variable que servirá como buffer
    String bufferString = "";
    /*
     * Se le indica a Arduino que mientras haya datos
     * disponibles para ser leídos en el puerto serie
     * se mantenga concatenando los caracteres en la
     * variable bufferString
     */
    while (Serial.available() > 0) {
      bufferString += (char)Serial.read();
    }
    //Se imprime el contenido del buffer
    Serial.println(bufferString);
  }
}
```

Cuando intentamos escribir **Panama Hitek** obtendremos:



…

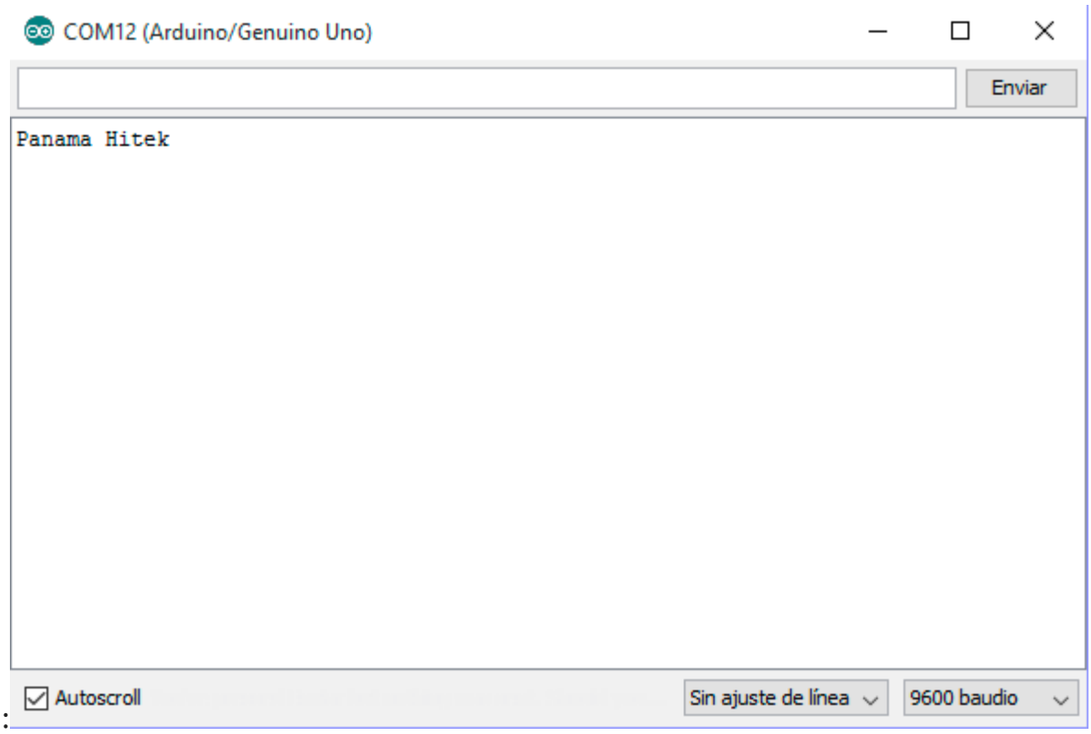
Ocurre lo mismo. Esto se debe a que transcurre muy poco tiempo entre el instante en que el Arduino detecta que hay un caracter disponible para ser leído en el Puerto Serie y la entrada en el while. Esto provoca que el **Serial.available()** del while solo devuelva un valor de 1, lo cual hace que los caracteres se impriman de forma individual. Esto se puede corregir de diferentes formas:

- Aumentando la velocidad de transmisión de datos (los baudios por segundo, el 9600 del **Serial.begin()**)
- Colocando un pequeño delay luego del**if(Serial.avaliable()>0)**
- Colocando un pequeño delay dentro del while.

Esto último será lo que haremos.

```
void setup() {
  //Iniciamos comunicación con el puerto serie
  Serial.begin(9600);
}

void loop() {
  /*
   * Evaluamos el momento en el cual recibimos un caracter
   * a través del puerto serie
   */
  if (Serial.available() > 0) {
    //Delay para favorecer la lectura de caracteres
    delay(20);
    //Se crea una variable que servirá como buffer
    String bufferString = "";
    /*
     * Se le indica a Arduino que mientras haya datos
     * disponibles para ser leídos en el puerto serie
     * se mantenga concatenando los caracteres en la
     * variable bufferString
     */
    while (Serial.available() > 0) {
      bufferString += (char)Serial.read();
    }
    //Se imprime el contenido del buffer
    Serial.println(bufferString);
  }
}
```

Y ahora el resultado será

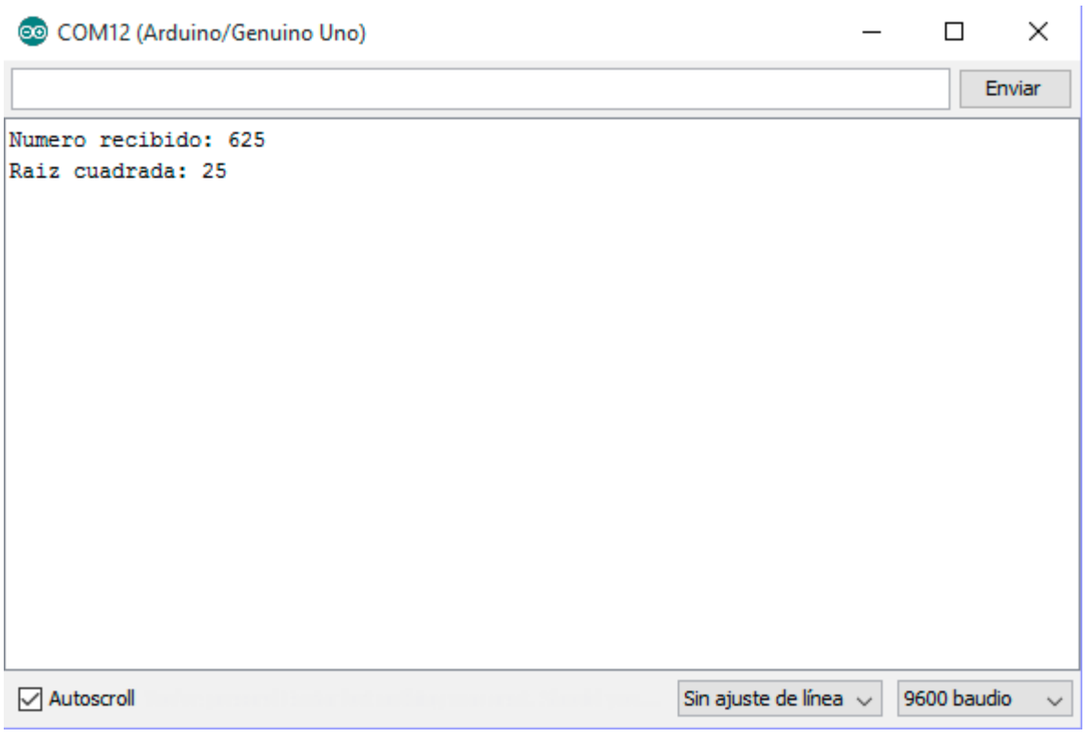


:

Nota: *El delay utilizado es el suficiente para este aplicación en específico pero puede variar según la cantidad de caracteres que se desee leer*. Ahora si queremos enviar números de más de un dígito a Arduino necesitaremos utilizad una conversión de**String** a **int**. Para ello utilizamos el comando **toInt()**:

```
void setup() {
  //Iniciamos comunicación con el puerto serie
  Serial.begin(9600);
}
 void loop() {
  /*
   * Evaluamos el momento en el cual recibimos un caracter
   * a través del puerto serie
   */
  if (Serial.available() > 0) {
    //Delay para favorecer la lectura de caracteres
    delay(20);
    //Se crea una variable que servirá como buffer
    String bufferString = "";
    /*
     * Se le indica a Arduino que mientras haya datos
     * disponibles para ser leídos en el puerto serie
     * se mantenga concatenando los caracteres en la
     * variable bufferString
     */
    while (Serial.available() > 0) {
      bufferString += (char)Serial.read();
    }
    //Se transforma el buffer a un número entero
    int num = bufferString.toInt();
    //Se imprime el número que se recibe
    Serial.print("Numero recibido: ");
    Serial.println(num);
    //Se calcula la raíz cuadrada del número recibido
    num = sqrt(num);
    //Se imprime el resultado
    Serial.print("Raiz cuadrada: ");
    Serial.println(num);
  }
}
```

El código presentado recibe un número cualquiera, lo transforma a número y calcula la raíz cuadrada del mismo y la muestra en pantalla. (Los datos se ingresan por el cuadro de texto del Monitor Serie.



Otro Ejemplo (Este se usa en el programa de Tachos LEDs para recibir ordenes desde la PC) *(Prof: Bolaños D)*

```
//Ejemplo de programa
//Lee algo en el puerto serial y lo almacena en num

int num; // Definida como variable global

void setup() {
  //Iniciamos comunicación con el puerto serie
  Serial.begin(9600);
}
 void loop() {

//Inicio para control desde la PC con programa creado en Builder----

 /*
   * Evaluamos el momento en el cual recibimos un caracter
   * a través del puerto serie
   */
  if (Serial.available()>0) {

  //Delay para favorecer la lectura de caracteres

    delay(22);

    //Se crea una variable que servirá como buffer
    String bufferString = "";

    /*
     * Se le indica a Arduino que mientras haya datos
     * disponibles para ser leídos en el puerto serie
     * se mantenga concatenando los caracteres en la
     * variable bufferString
     */

    while (Serial.available()>0) {
      bufferString += (char)Serial.read();
    }

    num = bufferString.toInt(); //Se transforma el buffer a un número entero
                                //Se carga lo leido en la variable num
                                 //Luego podemos preguntar sobre el valor
                                 // de dicha variable – Por ejemplo
                                 // en Tachos LED su valor selecciona color

}
}
```

NOTA: La variable num en este caso es del tipo entera (int), por lo tanto solo admitira valores numéricos sin decimales (16bits) comprendidos en el rango 32.767 to -32.768.

Si deseamos números mayores puede usar variables del tipo long. El formato de variable numérica de tipo extendido "long" se refiere a números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango -2147483648 a 2147483647.

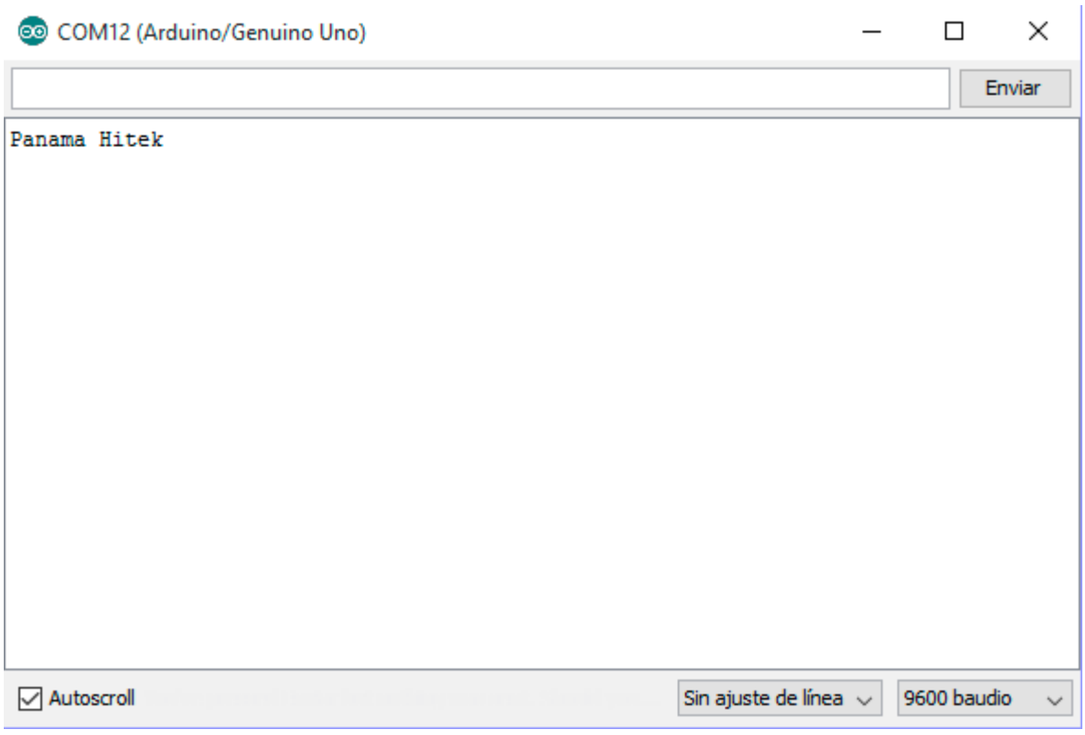
Método Directo

Arduino posee un método llamado **readString()** dentro de la clase Serial que permite leer cadenas de caracteres. De igual forma posee un método llamado setTimeOut() para definir el tiempo que debe esperar el código antes de imprimir la cadena de caracteres y dar por terminada la recepción de paquetes de datos.

El código sería:

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

void loop() {
 if (Serial.available()>0){
 Serial.println(Serial.readString());
 }
}
```

El resultado (el escribir Panama Hitek en el Monitor Serie):

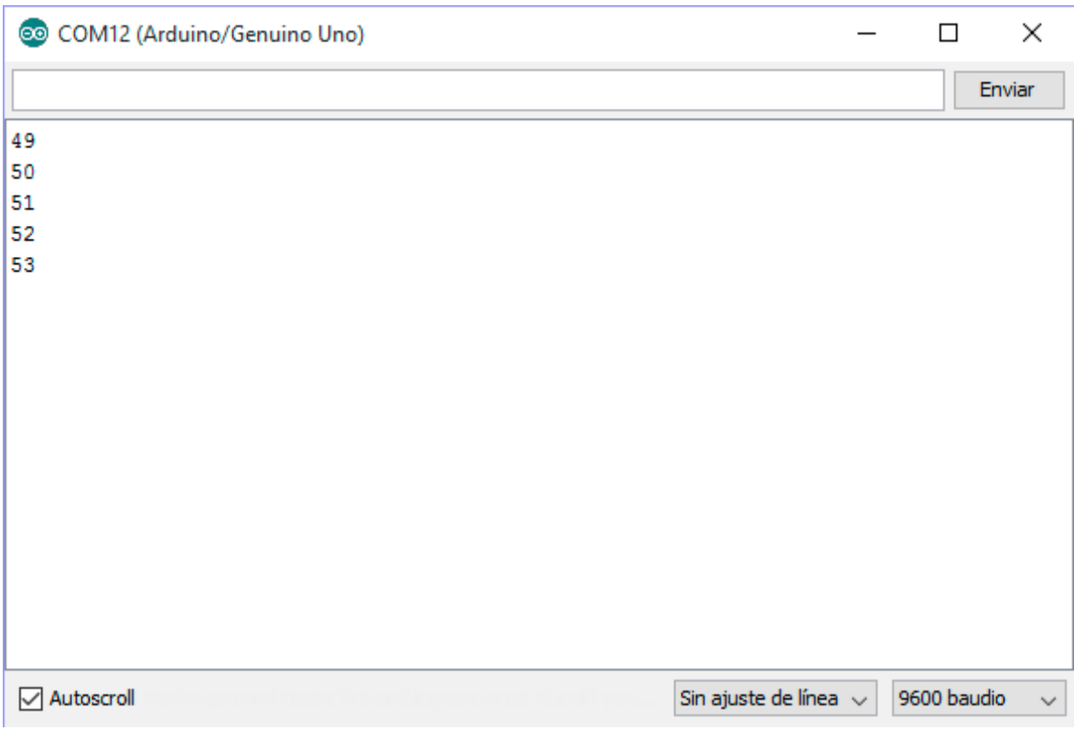


Como podemos observar se da un pequeño retraso en la impresión del mensaje en el Monitor Serie. Esto se debe a que por default se espera 1 segundo para terminar de recibir los datos entrantes. Este tiempo se puede modificar con**Serial.setTimeout()**. El siguiente código incluirá una reducción del timeOut a 100 milisegundos y se repetirá lo mostrado en el **Método Buffer** donde se calcula la raíz cuadrada de un número introducido por teclado.

```
void setup() {
  //Iniciamos comunicación con el puerto serie
  Serial.begin(9600);
  Serial.setTimeout(100);
}
```

```
void loop() {
/*
   * Evaluamos el momento en el cual recibimos un caracter
   * a través del puerto serie
   */
  if (Serial.available() > 0) {
    //Se transforma la lectura a un número entero
    int num = Serial.readString().toInt();
    //Se imprime el número que se recibe
    Serial.print("Numero recibido: ");
    Serial.println(num);
    //Se calcula la raíz cuadrada del número recibido
    num = sqrt(num);
    //Se imprime el resultado
    Serial.print("Raiz cuadrada: ");
    Serial.println(num);
  }
}
```

El resultado:



Esperamos que el concepto haya quedado claro y que la información suministrada sea de su comprensión y utilidad.

Autor : Antony García González

Ingeniero Electromecánico, graduado de la Universidad Tecnológica de Panamá. Miembro fundador de Panama Hitek. Entusiasta de la electrónica y la programación.