UNIT 12: COMMUNICATIONS

AIMS

Show you the principles which will enable your Arduino to communicate with any kind of device or peripheral including PCs, GPS receivers, computer monitors, GSM or GPRS mobile phone systems, Ethernet networking interfaces, Wi-Fi, Bluetooth and others. A controller's ability to communicate is its window to the world.

In this unit you're going to study the basic principles of serial communication as well as the functions you need to transmit and receive information. We'll also be looking at other general purpose functions so you can expand your knowledge of the Arduino language. Maybe you'll be able to use them for future projects and exercises.

Let me tell you right now that whilst there's quite a lot of theory in this unit, you'll be able to use it all to create more and more powerful and complex programs.

THEORY SECTION

PRACTICE MATERIALS

-Lap top or desk top computer

-Arduino IDE work environment; this should include the supplementary material already installed and configured.

-Arduino UNO controller board

-A USB cable

TABLE OF CONTENTS

THEORY SECTION

1. INTRODUCTION

Since ancient times humans, and other species too I suppose, have felt the need to communicate with their fellow beings. This is how we pass on and receive ideas, information, thoughts, knowledge, feelings, news and many other things. Communication usually involves two or more people unless of course you like talking to yourself and listening to the sound of your own voice.

We use a variety of formats to make communication or conversation more interesting and worthwhile: the spoken word, the written word and sign language. These days of course modern technologies like radio, television, Internet and others greatly expand the variety of formats at our disposal and enhance the quality of our communication.

On the other hand, we've invented a series of basic rules and conventions to ensure the effectiveness of our communication and avoid dialogues at cross purposes and shouting matches. In general, whilst one person speaks and communicates the information, the others listen and receive it.

Computers in general, and Arduino in particular, are also able to transmit and receive information. An Arduino can communicate with PCs, GPS receivers, computer monitors, speech recognition systems, mobile phones or, in other words, a wide range of devices.

Can you imagine what this means? Guided by your program, an Arduino, for instance, can gather information from the surrounding environment using different types of sensors (light, temperature, air pressure, humidity, speed, etc…), pushbuttons, presence detectors, switches and lots, lots more.

It can then transmit this information to a computer, a PC for example. Properly programmed the computer can then record the information, process it, display it, print it out or, in other words, do just about anything with it.

Likewise, a PC can transmit a series of codes or orders to the Arduino. When the Arduino receives these orders it encodes them and converts them into tasks such as turning a light globe, a LED light, a motor or a relay on or off. These orders can generate PWM signals, a range of frequencies, sounds, and even trigger a number of different internal processes in the controller.

In conclusion, the channels of communication open a window to the world. Arduino isn't just a simple board that executes a given program. Arduino's also able to "talk" to other apparatus or computer rigs that may be bigger and more powerful than it and share information with them.

2. SERIAL COMMUNICATION

When we connect Arduino to any other system, we're talking digital communication. The information is always transmitted as binary code comprised of "bits" or levels "1" and "0". The minimum amount of information that can be transferred is a byte or a group of 8 bits.

People have different means of communicating with each other: by speaking or writing to each other or by using sign language. In the same way, digital systems also have different ways of transferring information. The most widely used, which is also available in Arduino, is called UART serial communication.

Just two lines or wires are used: the Tx is for data transmission and the Rx for its reception. The Arduino in Figure 1 data on its Tx line; this is received by the PC on the Rx line. The same applies in the opposite direction: the PC transmits data on its Tx line whilst the Arduino receives it on its Rx line. Transmission at one end is reception at the other and vice versa.

A transfer of data may comprise tens, hundreds or thousands of bytes but it must comprise at least one. You already know that a byte is made up of de 8 bits and each one of them may be level “1” or “0”. The bits “travel around” one after the other on the Tx and/or Rx lines.

The time, or “**t**”, each bit stays on the line (be it the Tx or the Rx) determines how fast the communication is. For instance, if a bit stays on the Tx line for a second, we say that the speed of the communication is one bit per second. We call this a “Baud”: **1 Baud = 1 bit / second**. A byte would therefore take 8 seconds to be transferred. If you multiply these eight seconds by the number of bytes you have to transfer you’ll get the total amount of time you need to complete the transfer.

Arduino is able to perform transfers from 300 bauds up to 115,200 bauds or, in other words, 115,200 bits per second which is approximately 14,000 bytes per second (115200 / 8). A bit is transmitted and received every 8.6 µS (1 / 115,200).

You’ll be doing the exercises on communication in the practice area at a speed of 9,600 bauds. Most of the devices you’ll be communicating with can handle this sure and reliable speed; a PC is one example.

The Figure 1 shows the pins the Arduino UNO uses for serial communication.

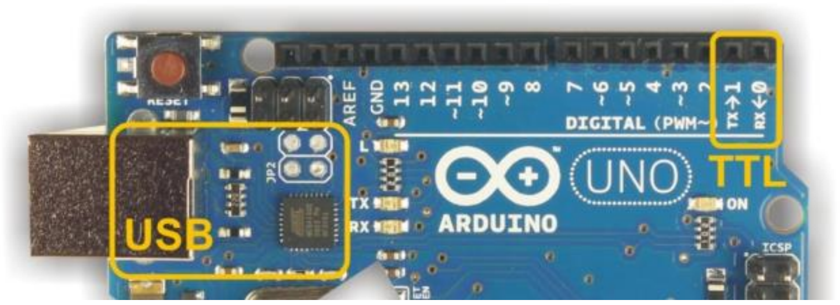


Figure 1

You can configure any digital pin as a general input or output; 0 and 1 are no exception. We haven’t used them as such up until now because we’ve always kept them for use as reception (Rx) and transmission (Tx) lines respectively. You can hook them up directly with any device you want to communicate with that takes 5 V signals known as “TTL levels”.

These same pins, 0 and 1, go to electronic circuits on the Arduino board. The circuits adapt them to USB specifications using the corresponding connector. You’ve actually already been using them. Remember that each time you upload a program it’s because there’s communication between the Arduino and the PC. The PC transfers all the instructions that make up the program.

To sum up, you can use the 0 (Rx) and 1 (Tx) pins, or their equivalent on the USB connector, to convey your own communication; this is controlled by your program.

3. ESTABLISHING COMMUNICATION

Let's get down to business then. From now on, you'll have to configure and enable communication capability for any program you create which requires them. To do this, the Arduino programming language has two very simple functions which are easy to use.

A. THE Serial.begin() FUNCTION

Enable the UART serial communication and establish the speed in bauds. This function triggers the configuration of pins 0 and 1 as reception (Rx) and transmission (Tx) lines respectively. They may not be used as digital input or output pins from now on.

Syntax:

Serial.begin(bauds, SERIAL_NPS);

bauds: Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.

*SERIAL_**NPS***: This is optional and makes it possible to configure character encoding:

N=Nº of bits per character: 5, 6, 7 or 8 (default value)

P=Parity: E= even, O= odd N=sin parity (default value)

S=Nº of stop bits at the end of the character: 2 or 1 (default value)

Example:

Serial.begin(9600);	*//Communication at 9600 with the character 8N1*
Serial.begin(300,SERIAL_5E2);	*//300 bauds with the character 5E2*

B. THE Serial.end() FUNCTION

This function disables serial communication allowing the 0 and 1 pins to be used for general digital input and output again.

Syntax:

Serial.end();

Example:

Serial.end();	*//Disables serial communication*

4. TRANSMITTING DATA

Once you've enabled serial communication you can start creating programs which use Arduino to transmit information to the outside world – a PC for instance. All you have to do is use the right functions. They're the ones we're going to study now.

A. THE Serial.print() FUNCTION

This is the function you're going to use to print data to the serial port; it may have a number of different formats. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are printed as a single character. Characters and strings are printed as is enclosed in single quotation marks (') or double ones (") respectively.

Syntax 1:

Serial.print(value);

value: the value to print - any data type.

Examples:

```
Serial.begin(9600)              //enables communication at 9600 bauds (8N1)
int A = 78;
float PI = 3.141592;
Serial.print(A*2);              //gives "156"
Serial.print(PI);               //gives "3.14"
Serial.print('A');              //gives "A"
Serial.print("Hello world");    //gives "Hello world"
```

Syntax 2:

Serial.print(value,format);

value: the value to print - any data type.

format: specifies the number base for integral data types or number of decimal places for floating point types.

Example:

```
Serial.begin(9600)              //opens the serialport at 9600 bps (8N1)

int A = 78; float PI = 3.141592;

Serial.print(A,BYTE);           //prints “N” (the value 78 equals the character N)

Serial.print(A,DEC);            //prints “78” (as an ASSCII-encoded decimal or base 10) default value

Serial.print(A,HEX);            //prints “4E” (as an ASSCII-encoded hexadecimal or base 16)

Serial.print(A,BIN);            //prints “01001110” (as an ASSCII-encoded binary or base 2)

Serial.print(A,OCT);            //prints “116” (as an ASSCII-encoded octal or base 8)

Serial.print(PI,0);             //prints “3” (without decimals)

Serial.print(PI,2);             //prints “3.14” (with 2 decimals). Default value

Serial.print(PI,5);             //prints “3.14159” (con 5 decimals)
```

B. THE Serial.println() FUNCTION

This function is very similar to Serial.print(). The difference is that after printing the data requested to the serial port as human-readable ASCII text it is followed by a “carriage return” character (\r or 0x0D) and a “newline” character (\n o 0x0A). The receiver usually uses these codes to detect the end of a line of data.

Syntax:

Serial.println(value);

Serialprintln(value, format);

value: the value to print - any data type.

format: specifies the number base for integral data types or number of decimal places for floating point types.

5. RECEIVING DATA

Once you've enabled and established communication, as well as transmitting data your Arduino can also receive characters sent to it by a PC, another Arduino, or any other device that you connect it to.

Arduino can execute programs at the same time as it receives characters. All it does is automatically save each byte it receives in an internal RAM memory called the "serial receive buffer". Have a look at the Figure 2:

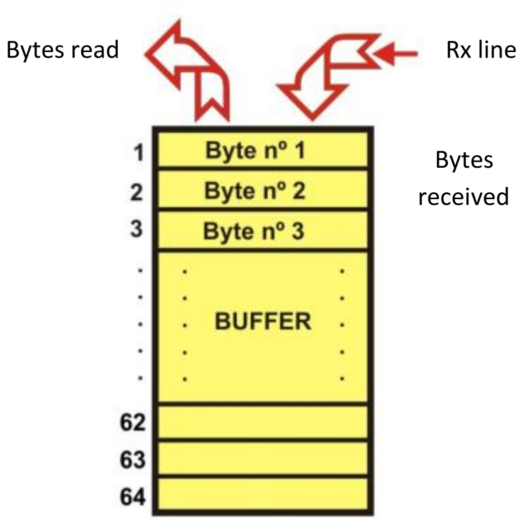


Figure 2

Think of the buffer as a kind of warehouse where you can store characters or bytes as they come in. It can take up to 64 bytes. If it receives more than 64 without having read or expelled the ones that were there before, it overflows and starts losing the bytes that arrived first. The Arduino programming language includes a number of functions that make it possible to know whether or not bytes have been received or, in other words, whether or not there are any stored in the serial receive buffer. These functions also make it possible to read the bytes in order of arrival.

Have a look at the Figure 3. A laptop is connected to our Arduino UNO board. The laptop transmits the characters or bytes that comprise the "HOLA" string. The controller receives the characters or bytes and stores them in the internal receive buffer in the RAM memory.

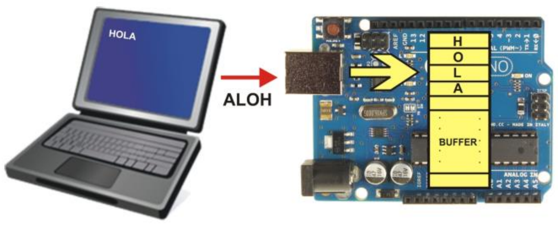


Figure 3

A. THE Serial.available() FUNCTION

This is a very important function. It returns the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes). If it returns a value of 0 it means that it hasn't received any bytes and that the receive buffer is empty. The maximum value it can return is 64.

Syntax:

Serial.available();

Example:

```
byte A;

A = Serial.available();            //"A" loads a 4 assuming "Hello world" has been received.

…

while(! Serial.available());       //pauses until it receives another byte
```

B. THE Serial.read() FUNCTION

Reads serial data coming into the buffer in the order it arrives. If it hasn't received anything it returns a value of -1.

Syntax:

Serial.read();

Example:

```
char A;

A = Serial.read();                 //Reads the first character received (H) in the buffer

A = Serial.read();                 //Reads the following character received (o) in the buffer

A = Serial.read();                 //Reads the following character received (l) in the buffer

A = Serial.read();                 // Reads the following character received (a) in the buffer
```

C. THE Serial.parseInt() FUNCTION

parseInt() returns the first valid (long) integer number from the serial buffer. The function only reads integers and plus or minus signs. The number must be between -32768 y + 32767.

Syntax:

Serial.parseint();

Example:

int N;	*// int type variable (2 bytes)*
N = Serial.parseInt();	*// "N" loads the integer (int) received*

D. THE Serial.parsefloat() FUNCTION

Serial.parseFloat() returns the first valid floating point number from the Serial buffer. The function only reads integers and plus or minus. parseFloat() is terminated by the first character that is not a floating point number. No other characters are accepted.

Syntax:

Serial.parsefloat();

Examples:

float N;	*//floating point variable (4 bytes)*
N = Serial.parsefloat();	*//"N" loads the floating point number (float) received*

The functions for transmitting and receiving data that we've looked at in the preceding pages are the most important ones; they're also the most frequently used ones in serial communication. You're going to use them to perform the following suggested exercises. All the same, just remember there are a few more functions. Have a look at the following website, study the functions there and then try and use them whenever you think they're appropriate: www.arduino.cc.

6. OTHER GENERAL PURPOSE FUNCTIONS

I'd like to introduce a few general purpose functions to further increase your knowledge of the Arduno programming language. You'll be able to use them and the ones you already know to do all the exercises I'll be suggesting in this unit; they'll open doors to the creation of new programs, projects and applications. The only limit is your own imagination!

A. THE pulseIn() FUNCTION

By now you know more than enough to generate PWM output signals. You did it in the previous unit. But you should also be aware that there are peripherals and sensors that give a PWM input signal in response to certain procedures. Have a look at the Figure 4:

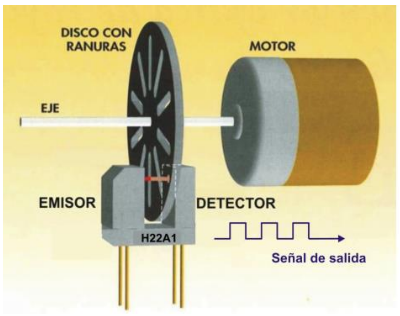


Figure 4

The shaft of the motor is connected to a slotted disc. The H21A1 phototransistor optical interrupter switch generates a level “1” each time one of the slots on the disc goes past it. That’s how we get a PWM digital signal. The length of the level “1” or duty cycle depends on the size of the slots and the level “0” on the distance between them. If you manage to measure the length of the level “1”, or the level “0”, and you know the number of slots, you can calculate how fast the disc is turning. But...how can you measure a PWM input signal? Answer: by using the **pulseIn() function**.

This function reads a pulse (either level “1” or level “0”) on any of the input pins. For instance, if the value is “1”, **pulseIn()** waits for the pin to go to “1”, starts timing, then waits for the pin to go to “0” and then stops timing. It returns the length of the pulse as an unsigned long integer value in microseconds between 0 µS and 4.294.976.295 µS (2^{32}) which is equivalent to about 70 minutes.

Syntax:

pulseIn(pin, level, time);

pin:	*the number of the input pin you want to read the pulse on. (int)*
value:	*type of pulse to read: either the level “1” signal or the level “0” signal. (int)*
timeout (optional):	*the number of microseconds to wait for the pulse to be completed: the function returns 0 if no complete pulse was received within the timeout. Default is 1,000,000 µS or one second (unsigned long).*

Example:

```
unsigned long A;

A = pulseIn(4,1,2.500.000);        //Measures a level “1” pulse on pin 4. Pauses

                                   //for 2.5” until it’s completed. The result is saved

                                   //in the “A” variable
```

B. THE min() AND max() FUNCTIONS

Apart from just adding, subtracting, multiplying and dividing, Arduino can perform quite a few more mathematical and trigonometric functions. Let’s have a look at them:

The min() and max() functions compare two numbers and return the minimum and maximum value of both respectively.

Syntax:

min(A,B); max(A,B);

A: the first number, any data type

B: the second number, any data type

Example:

```
int A = 23;
int B = 15;
int C;
C = min(A,B);                  //C = 15
C = max(A,B);                  //C = 23
```

C. THE abs() FUNCTION

Computes the absolute value of a number.

Syntax:

abs(n);

n: the number

Example:

```
int A = -13;
int C;
C = abs(A);                    //C = 13
```

"The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein."

D. THE pow() FUNCTION

Calculates the value of a number raised to a power. Pow() can be used to raise a number to a fractional power. This is useful for generating exponential mapping of values or curves.

Syntax:

pow(base,exponent);

base: the number (*float*)

exponent: the power to which the base is raised (*float*)

Example:

float A = 23.00;

float B = 3.00;

float C;

C = pow(A,B); *//C = 12167.00 (23^3)*

E. THE sqrt() FUNCTION

Calculates the square root of a number.

Syntax:

sqrt(n);

n: the number, any data type

Example:

int C;

C = sqrt(9); *//C = 3*

F. THE sin(), cos(), tan() FUNCTIONS

These functions enable you to calculate the sine, cosine and tangent of an angle expressed in radians. Remember this:

$$\mathbf{360^{º} = 2\pi\, rad; 180^{º} = \pi\, rad; 1^{º} = 0.01745322925}$$

Syntax:

sin(n); cos(n); tan(n);

n: this is the value of the angle in radians

Example:

```
int angulo = 36;                       //Angle in degrees

float radians = angle * 0.017453;      // Angle in radians

float C;

C = sen(Radians);                      //C = 0.5877

C = cos(Radians);                      //C = 0.8090

C = tan(Radians);                      //C = 0.7265
```

PRACTICE SECTION

The Arduino board will be using serial communication to transmit and receive data in all the exercises that you'll be doing in this unit. But just who will it be communicating with? I trust that it's clear by now that it can communicate with any kind of device. But probably the most convenient device is the PC you use to write your programs.

Yes, that's right: for the following exercises your PC will be receiving the data the Arduino board transmits and the Arduino board will be receiving the data your PC transmits. You'll be using serial communication between the Arduino board and the PC.

Your PC needs a "communications program" to transmit and receive data. There are lots of programs like this: Hyper Terminal (included in Windows), Docklight, Procomm, Smartcomm and many others. The Arduino environment even includes a built-in serial monitor, or simple communications program, which you can use to communicate with an Arduino board. And that's exactly what we're going to use. All you need to do to enable it is press the magnifying glass icon, as shown in the Figure 5.

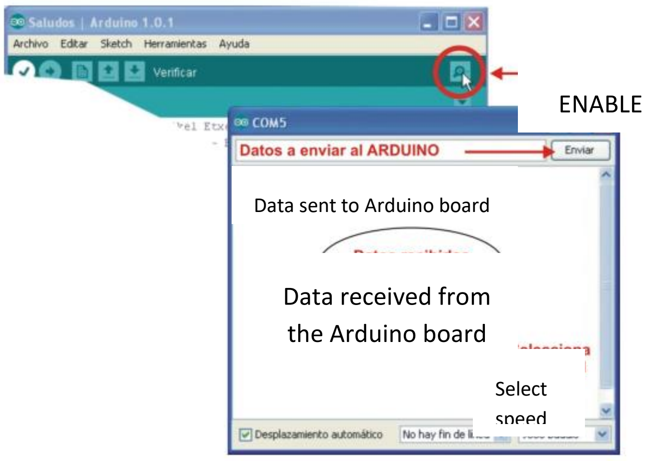

Saludos | Arduino 1.0.1
Archivo Editar Sketch Herramientas Ayuda
Verificar
ENABLE
COM5
Datos a enviar al ARDUINO
Enviar
Data sent to Arduino board
Data received from the Arduino board
Select speed
Desplazamiento automático
No hay fin de lí

Figure 5

You write the data that you want to send in the field at the top (e.g. to the Arduino board). When you press the "send" button the data the PC transmits the data. The data you receive will appear in the white box at the bottom (e.g. from the Arduino board). You can also select the transmission speed. This will be 9600 bauds or bits per sec in the exercises we'll be doing.

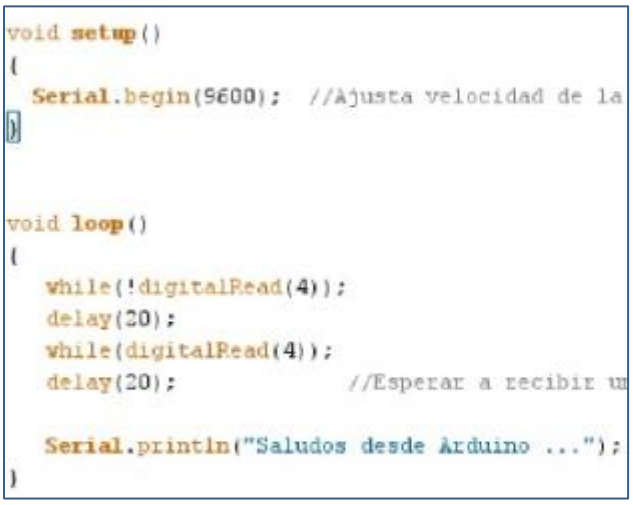
7. EXAMPLE 1: Saying hello

This is a very simple exercise on data transmission. Each time you press the pushbutton on the development board connected to pin 4 the Arduino board responds by transmitting a greeting. Have a look at Figure 6 and you'll see that it's quite simple.

The Serial.begin() function in setup() sets the communication speed.

Once a pulse on pin 4 is detected we use the Serial.Println() function to transmit the string we've indicated.

Once you've compiled and recorded the program on the Arduino UNO board, don't forget to enable the "Serial Monitor" program using the magnifying glass icon. Each time you press pushbutton 4 you'll receive a greeting message from the Arduino board.

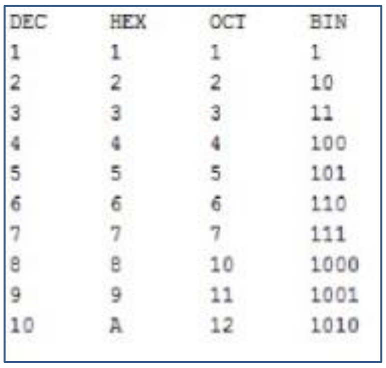


```
void setup()
{
  Serial.begin(9600);   //Ajusta velocidad de la
}


void loop()
{
   while(!digitalRead(4));
   delay(20);
   while(digitalRead(4));
   delay(20);              //Esperar a recibir u

   Serial.println("Saludos desde Arduino ...");
}
```

Figure 6

8. EXAMPLE 2: Numbering systems

This exercise offers a great opportunity to revise the different numbering systems. Each time you press pushbutton 4, the Arduino board transmits a list of 10 consecutive integers expressed in the following formats: decimal (base 10), hexadecimal (base 16), octal (base 8) and binary (base 2). Figure 7 would be the result:



```
DEC     HEX     OCT     BIN
1       1       1       1
2       2       2       10
3       3       3       11
4       4       4       100
5       5       5       101
6       6       6       110
7       7       7       111
8       8       10      1000
9       9       11      1001
10      A       12      1010
```

Figure 7

9. EXAMPLE 3: Calculator

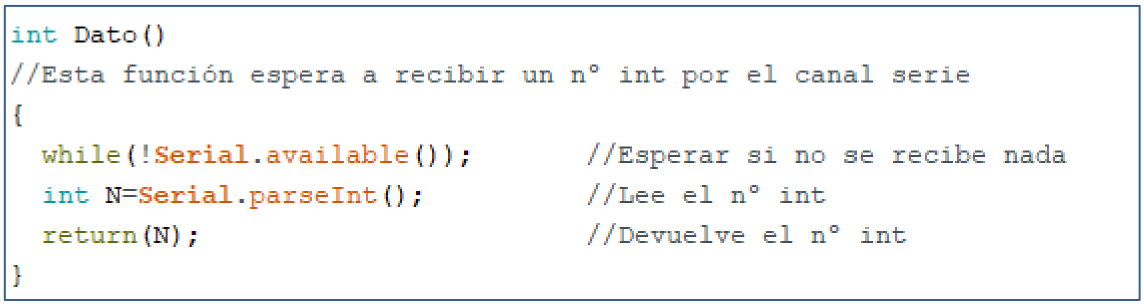
Now we're going to simulate a calculator capable of performing the four basic arithmetical operations between two integers: addition, subtraction, multiplication and division. There are basically two interesting aspects to the exercise:

- The communication is two-way. The Arduino board receives the two numbers and the sign of the operation to be performed (+, -, *, y, /) from the PC. The Arduino board subsequently performs the operation and retransmits the result to the PC.
- You're going to create and use a function for the first time. A function is nothing more than a series of tasks that can be executed a number of times. Instead of writing them the exact number of times you need them, you just write them once and then execute them as many times as you need to. The program in this exercise has to receive the two numbers to perform the operation with; in order to receive each one of these numbers it has to perform a series of tasks:

1. Wait until it receives data by serial transmission.
2. Read the value received (the integer).
3. Save it and return it in a variable.

Why do we have to repeat these three tasks for every number we want to receive? Imagine instead of having to wait to receive two numbers you had to wait to receive six. Does that mean you'd have to write those three tasks six times? And what about if you had to receive twenty numbers? Would you have to write those three tasks another twenty times? The answer, quite obviously, is no. There has to be some kind of technique for avoiding all this repetition; this technique involves creating your own functions. They're also called "subroutines".

Pay close attention to the extract in the Figure 8

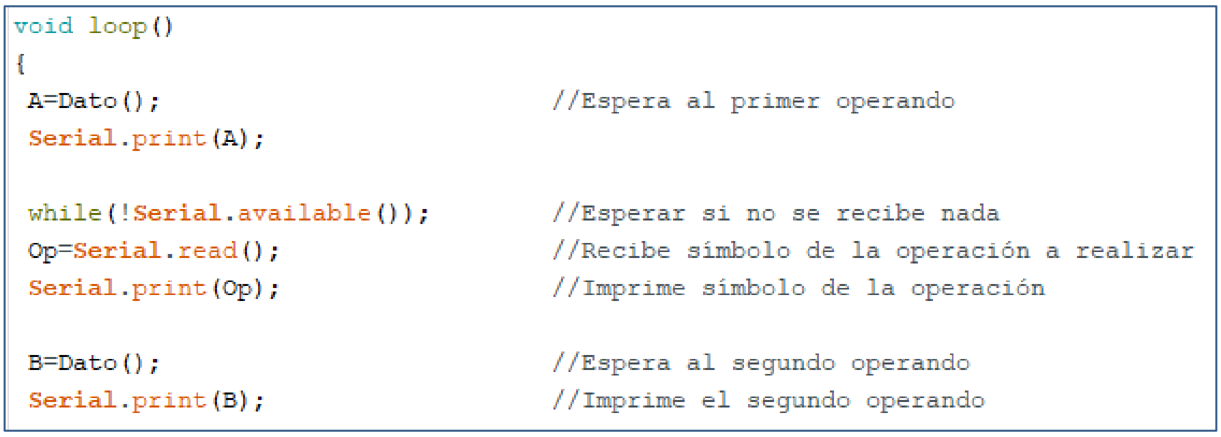


```
int Dato()
//Esta función espera a recibir un n° int por el canal serie
{
  while(!Serial.available());       //Esperar si no se recibe nada
  int N=Serial.parseInt();          //Lee el n° int
  return(N);                        //Devuelve el n° int
}
```

Figure 8

We've created a function called **Dato()** which in turn is made up of three other functions or tasks: **while(), Serial.parseint()** and **return().** Each time your program calls up and uses the **Dato()** function, these three internal functions are also executed: **(while(!Serial.available())**, which waits until data is received by serial transmission; **(int N=Serial.parseint())**, which reads the integer that's just been received; and **(return(N))**, which returns the number to your program.

Now your program will be able to execute all the tasks in the **Dato()** function as many times as you want without you having to write them each time. All you have to do is call up the **Dato()** function that contains them. Have a close look at **¡Error! No se encuentra el origen de la referencia.** extract from the same program:



```
void loop()
{
 A=Dato();                           //Espera al primer operando
 Serial.print(A);

 while(!Serial.available());         //Esperar si no se recibe nada
 Op=Serial.read();                   //Recibe símbolo de la operación a realizar
 Serial.print(Op);                   //Imprime símbolo de la operación

 B=Dato();                           //Espera al segundo operando
 Serial.print(B);                    //Imprime el segundo operando
```

Figure 9

In the main body of the **loop()** program, you'll see the first call to our **A=Dato()** function. The function has to wait until it receives an integer by serial communication. It then reads it and returns it having saved it onto the "A" variable. Next it retransmits it.

Then it waits to receive the symbol of the arithmetic operation that you want it to perform (+, -, *, /), it reads it and saves it onto the "Op" variable: Next it retransmits it.

Finally it calls up the same function again, **B= Data().** Once again it waits until it receives an integer, reads it and returns it, having saved it onto the "B" variable this time. It retransmits it.

Our program has already received the A number, the operation sign and the "B" number. It's got everything it needs to make the calculation. Have a look at some of the operations performed. Make certain you remember to open the "*Serial Monitor*" using the magnifying glass icon!

Here are some recommendations and advice for designing and making your own functions:

- ✓ It's a good idea to define and write your functions at the beginning of the program, even before the main void loop() function.
- ✓ You can't use a function without previously defining it. If you don't, it shows error.
- ✓ When assigning a name to a function, remember it has to always begin with a letter; there can't be any blank spaces either.
- ✓ All the functions contained in the function you create have to be enclosed in square brackets ({…}).
- ✓ If the function returns an integer, as in this example, you have to declare it as such: int Data().
- ✓ The return() function terminates your function and may or may not return the value you indicate. return(N) returns the value N in the example.
- ✓ A function can also have input parameters which it has to use to operate. These parameters are "passed on" from the main program that calls up the function

- ✓ One function can “call up” another.
- ✓ Once the execution of a function has been terminated, we go back to the main program or the function that called it up.
- ✓ The main program or a function can use and “call up” another function and execute all the instructions it contains as many times as may be necessary.

10. MATHEMATICS

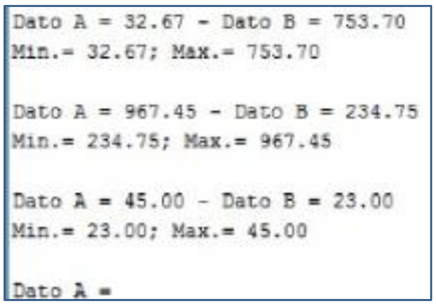
What I’m going to do now is give you some simple examples you can use to check how some of the mathematical and trigonometric functions you studied in the theory area work. All of them are based on two-way serial communication.

The Arduino board receives the data from the PC, performs the calculation requested and retransmits the result to the PC.

A. EXAMPLE 1: Min Max

By using the **min()** and **max()** functions the program obtains the minimum and maximum values of the two numbers you send it from the PC. Once the Arduino board has obtained the result it retransmits it back to the PC. As you can see, the program is fairly “interactive”: messages inform the user exactly which values to insert and when.

You’ll notice too that the **Dato()** function has been modified as well. On this occasion it waits until a floating-point number, a number that has a decimal point, has been inserted.

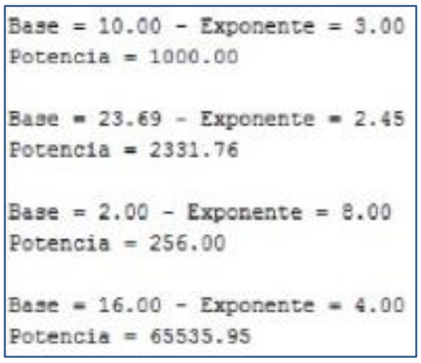


```
Dato A = 32.67 - Dato B = 753.70
Min.= 32.67; Max.= 753.70

Dato A = 967.45 - Dato B = 234.75
Min.= 234.75; Max.= 967.45

Dato A = 45.00 - Dato B = 23.00
Min.= 23.00; Max.= 45.00

Dato A =
```

Figure 10

B. EXAMPLE 2: Power

Need to calculate the value of a number raised to a power? Here's your chance. You transmit the base of the number and the exponent from the PC. The Arduino board will work it out and send the answer back to the computer. Here are a few results.



```
Base = 10.00 - Exponente = 3.00
Potencia = 1000.00

Base = 23.69 - Exponente = 2.45
Potencia = 2331.76

Base = 2.00 - Exponente = 8.00
Potencia = 256.00

Base = 16.00 - Exponente = 4.00
Potencia = 65535.95
```

Figure 11

The example may help you to convert base numbers into decimals.

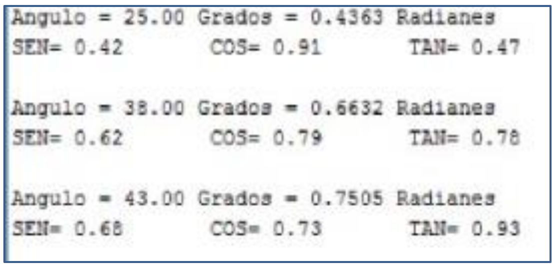
C. EXAMPLE 3: Square root

Just in case you need to work out the square root of a number, here's an example. You'll notice that in this exercise you'll be using float-point numbers to five decimal places.

D. EXAMPLE 4: Trigonometry

We're going to finish off this series of mathematical calculations with a final exercise. It involves working out the sine, cosine and the tangent of an angle expressed in degrees. The program takes care of converting it into radians and works out the results using the sin(), cos() and tan() functions you saw earlier.

The exercise may not seem terribly complicated but it probably uses more Flash program memory than any of the ones you've used up until now – more than 5,100 bytes.



```
Angulo = 25.00 Grados = 0.4363 Radianes
SEN= 0.42        COS= 0.91        TAN= 0.47

Angulo = 38.00 Grados = 0.6632 Radianes
SEN= 0.62        COS= 0.79        TAN= 0.78

Angulo = 43.00 Grados = 0.7505 Radianes
SEN= 0.68        COS= 0.73        TAN= 0.93
```

Figure 12

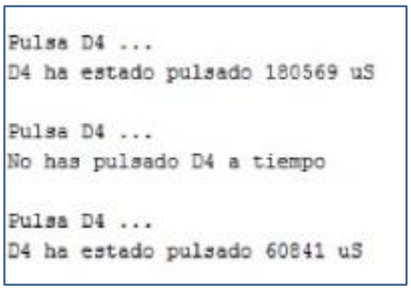
11. EXAMPLE 4: Measuring pulses

That's enough of numbers and calculations; we're going to work with peripherals now. This exercise will show you how to measure the time the pushbutton connected to pin 4 stays on level "1".

You'll be using the **pulseIn()** function. It returns the time the pin stays on level "1" or level "0" in µS. If it goes over the previously established time limit without detecting one or other of these levels there's an overflow or "Time Out". The time limit established was 10 seconds. The function returns a value of 0 in this case. The result of the measurement is transmitted to the PC by serial communication.

There are a few different measurements in the Figure 13.

- ✓ The pushbutton was at level "1" for 180569 µS or 0.180 seconds in the first one.
- ✓ No pulse was detected within the established ten second time limit in the second measurement.
- ✓ The pushbutton remained enabled for 60841 µS or 0.060 seconds in the third one.



```
Pulsa D4 ...
D4 ha estado pulsado 180569 uS

Pulsa D4 ...
No has pulsado D4 a tiempo

Pulsa D4 ...
D4 ha estado pulsado 60841 uS
```

Figure 13

When you try out this exercise, remember that pushbuttons have a "rebound effect"; we talked about it earlier on.

12. EXAMPLE 5: Remote monitoring

Here's something even more interesting. This exercise can certainly have an important practical application; you're sure to find one. The idea is to monitor the state of four pushbuttons.

The Arduino board reads the state of the buttons and transmits the results to the PC. Remember that instead of pushbuttons you could read the state of some sensors or detectors and monitor the signals on a PC, a printer or a screen or on lots of other devices.

The idea is pretty simple. Have a look at the following extract from the program.

It reads pin 12. If the pin's on "1" it transmits the character '1'. If it's not, it transmits the character '0'. It explores pins 8, 7 and 4 in the same way.

13. EXAMPLE 6: Remote control

Of course you can also monitor the state of the output peripherals by remote control. You're going to switch the white, green, amber and red LEDs on and off from the PC. We've invented four commands to do it: "w" for the white LED, "g" for the green one, "a" for the amber one and the "r" for the red one. When the Arduino board receives one of these the commands the corresponding LED changes state.

Have a close look at the program and record it onto the Arduino UNO board. Use the "Serial Monitor" to key in and transmit the four available commands ('w', 'g', 'a' and 'r'). Notice that the corresponding LED changes state. If it's off it'll go on and vice versa.

This exercise will also give you the opportunity to experiment with the **switch() / case** function; we haven't used it a great deal up to now.

14. EXAMPLE 7: PWM Control

What if you could regulate the brightness of a LED light from a PC? This is just what the following exercise enables you to do. By using the "Serial Monitor" from the Arduino IDE you'll be able to control the voltage of the PWM signal that goes to the white LED connected to pin 6 from the PC.

The Arduino board receives the necessary voltage, calculates the length of the duty cycle and generates the appropriate PWM signal. The figure above shows some of the voltages transmitted. Note the variation in the brightness of the LED for each voltage.

15. EXAMPLE 8: Remote Servo Control

And what about moving a servomotor shaft by remote control? Well, here's a good example. You use the PC to indicate the number of degrees you want the shaft to turn as you can see from the figure on the left. Make sure the shaft turns the number of degrees you indicated.

REFERENCES

BOOKS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

WEB SITES

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com