

CRISTINA RODRÍGUEZ SÁNCHEZ - URJC

Arduino Notebook

NOTEBOOK1: Introduction to Digital Design & Arduino

DESCRIPTION OF TASKS

The objective of the laboratory sessions is to introduce you to the design of digital systems based on Arduino (open-hardware) and increase your level of programming. We will learn to connect the Arduino with external input and output peripherals, to obtain a higher level system. If you do all the practices with a desire to learn, you can make your own digital designs.

HARDWARE

In this initiation script we show you examples of codes to learn how to program and how to upload them to an Arduino Mega board, which is what we will use in class.

In addition, in the laboratories you will also have a Grove kit that includes a Manual, which you must follow to complete your practice. We remind you that it is mandatory that you take care of the material and that you respect all its contents. At the end of each practice and at the end of the block of practices of this subject, the material should be complete and in the same way as you found it at the beginning of the subject.

In addition to the plate, in the laboratories you will also have a special USB cable for the Arduino Mega board. This cable is also part of the content of the practices. Please take care of it and do not lose it.

In Fig. 1 you can see a photo of the Arduino Mega board, indicating the notation of each of the input / output legs of the board. Remember what we saw in class about digital and analog outputs and inputs. You can use in the simulator Arduino mega or Arduino Uno. The Arduino Uno board has less pins.

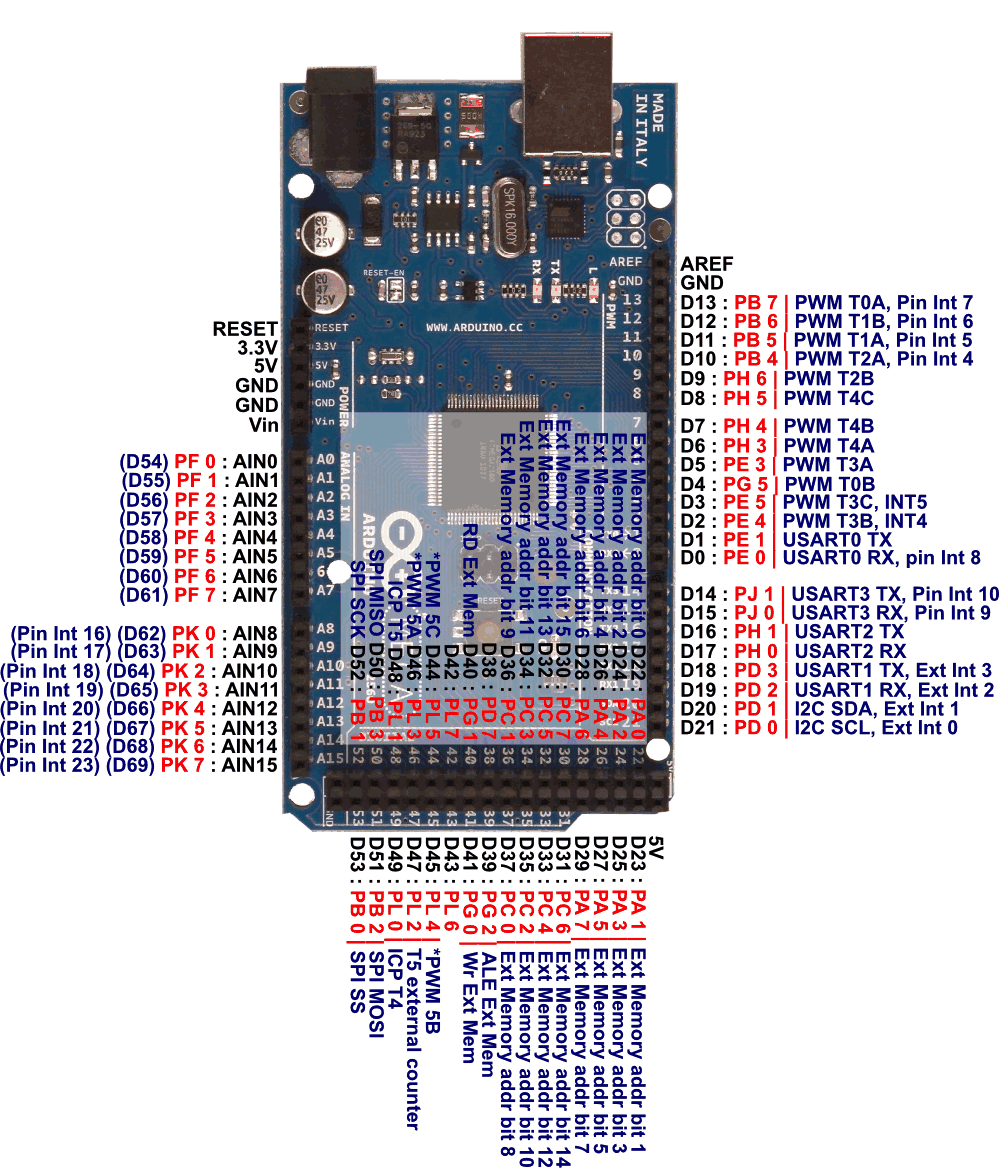


Fig. 1: Picture and pinout of Arduino mega.

CHAPTER ONE

Before starting, remember to review the transparencies of the Arduino seminar and above all do not forget to check the official Arduino website. There you have many guided and documented examples that will help you to follow this script much better. And when you have any questions about Arduino, search the internet. There are many Arduino resources at your fingertips.

Next, we will show you different examples that you should follow to understand how Arduino is programmed and the different structures that you can have.

Note that Arduino is case-sensitive: Be careful when writing variable names or commands.

Let's start to see how a program is made in Arduino.

The language with which our Arduino boards are programmed is a derivative of the Wiring programming language. Its structure and syntax is the same as that of C. In fact, all C libraries and some C ++ libraries can be used with Arduino.

The basic structure of a sketch is very similar to that of any C program, with some differences imposed by the fact that our platform interacts directly with the physical environment that surrounds it.

Thus, the first part of a Sketch will be the declaration of variables, where we will define the global variables of the Sketch, declaring its type and, if we want, its initial value. These variables will be accessible from any part of our Sketch, as opposed to local variables, which are those that are declared within a function or within a control structure (for example, a For loop).

Examples:

|  |  |
| --- | --- |
| 1 2 3 | int pin;    // Create an int variable int numero = 1;   // Create an int variable. Number has value 1. |

We will see the types of variables in depth later.

The second part of a Sketch or program in Arduino is a part that only runs once after the Arduino board is booted. It's the so-called Setup.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | Void Setup() { ... ... ... } |

It is usually used to define the pins we are going to use and declare them as inputs or outputs. That is, on which pins we will connect an LED, a temperature sensor, a servomechanism, etc. When it comes to sending information, for example, to turn on a Led, we will have to connect it to an output pin. While when we want to read some data from abroad we must define the pin as an input pin. For example, to read the measurement of a temperature sensor connected to our board.

The Setup () should be considered as a set of code that only runs once when the Arduino starts up, and serves to define and initialize the different parameters of the board and the code. This will give us incredible flexibility when it comes to programming.

The next important part of a Sketch is the block of code called Loop (loop). This is the main part of our program, and in it we will have to put the code so we want the microcontroller to run. It is called Loop because the microcontroller executes it recursively. That is, this part of the Sketch will be repeated continuously while the Arduino board has power.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | Void Loop() { ... ... ... } |

Not all the code has to be in the Loop. Certain parts can be written outside of this loop: functions. The functions are fragments of separate code that can be called from the Setup or from the Loop, and that we will study in detail later.

The Sketches generated from the Arduino IDE are saved in files with the extension .pde. This comes from the Arduino IDE is an adaptation of the IDE of a language called Processing (which also comes the name Sketch). The pde extension comes from 'Processing Development Environment'.

To handle the information in a program, save the result of operations, etc ... the variables mentioned above are used. A variable can contain different types of data:

* boolean
* char
* byte
* int
* unsigned int
* long (32b)
* unsigned long (int 32b with sign)
* float
* double (float 32b)
* string
* array
* void (null)

Además, también debes recordar que hay determinados caracteres especiales que sirven para diferentes funciones, tal y como se explicó en clase. Su sintaxis es la siguiente:

* ;
* {}
* // (comments in one line)
* /\* \*/ (comments in one line or more

On the other hand, you have different operators to perform arithmetic, comparison or composition operations:

• Arithmetic Operators

- = (assignment)

- + (sum)

 (subtraction)

- (multiplication)

- / (division)

-% (rest)

• Comparative Operators

- == (equal to)

-! = (other than)

- <(less than)

-> (greater than)

- <= (less than or equal to)

-> = (greater than or equal to)

• Boolean operators

- && (Y)

- || (or)

-! (denial)

• Composition Operators

- ++ (increase)

- - (decreases)

- + = (sum composition)

- - = (composition subtracts)

- \* = (multiplication composition)

- / = (division composition)

**Now we are going to do code step by step. You will see that the code is self-documented using comments separated by the character "//".**

**On the first day of Class you will have to go with all the code of this complete script, compiled and simulated with the https://www.tinkercad.com.** An access will be opened in the virtual campus so you can upload the programs made in Arduino (code) and the videos or links to your https://www.tinkercad.com so that the professor can access them. To know what to deliver in this booklet, at the end of each milestone, the section "What should I give to the course teacher?" Is indicated by the instructions indicated in that section.

In order to implement the codes of the practices, and to deliver the simulation of them, you can use the tinkercad program. The tinkercad program allows you to simulate a board in an online environment and add hardware components, upload the code to that simulation board, execute, debug step by step, etc. In the following website you have a video tutorial:

https://www.tinkercad.com

INSTALL ARDUINO ON mac or Linux

(***not neccesary if you will use the simulator***)

If you have a computer with a Linux or Mac operating system, all you have to do is:

1. Connect the Arduino board to the PC.

2. Download the software from https://www.arduino.cc/en/Main/Software

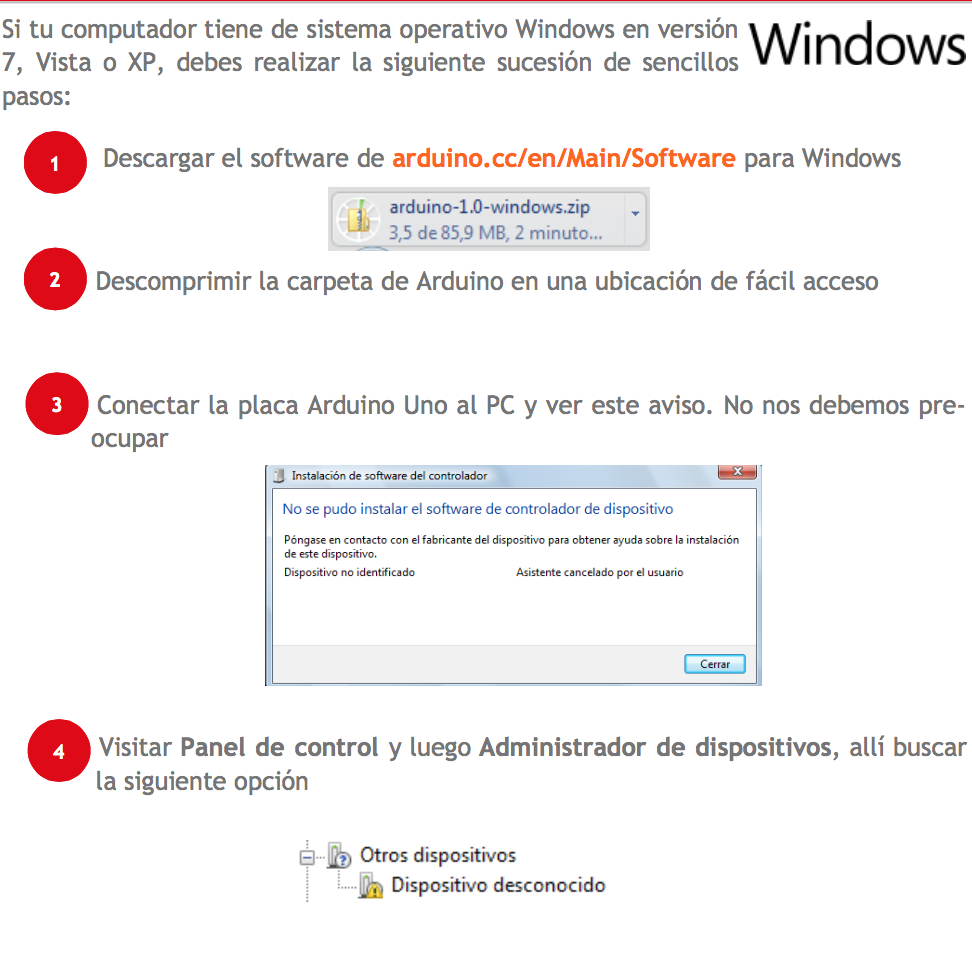
(when downloading the software will ask you if you want to just download it or contribute by making a donation and download it, you can choose any of the two options).

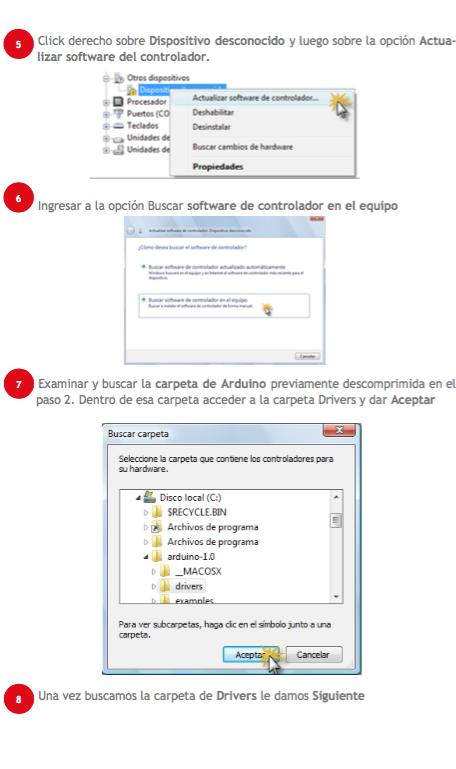
3. Run and start programming.

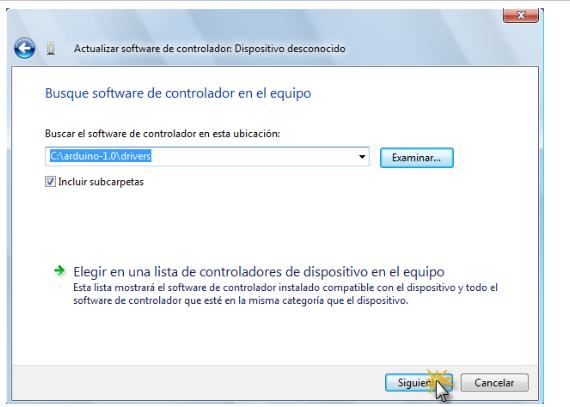
Install Arduino on Windows

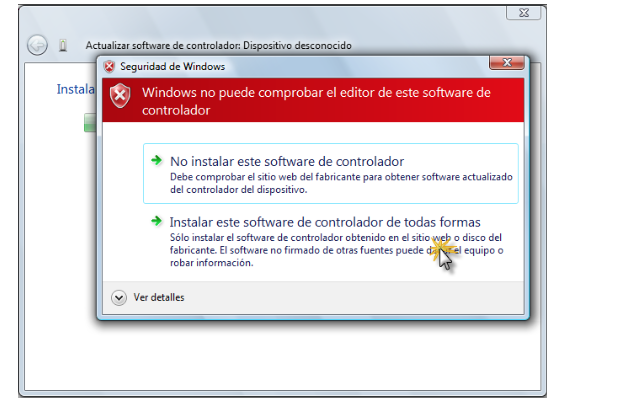
(Note: when downloading the software will ask you if you want to just download or contribute by making a donation and download it, you can choose any of the two options).

You will access instructions in english: https://www.arduino.cc/en/Guide/Windows





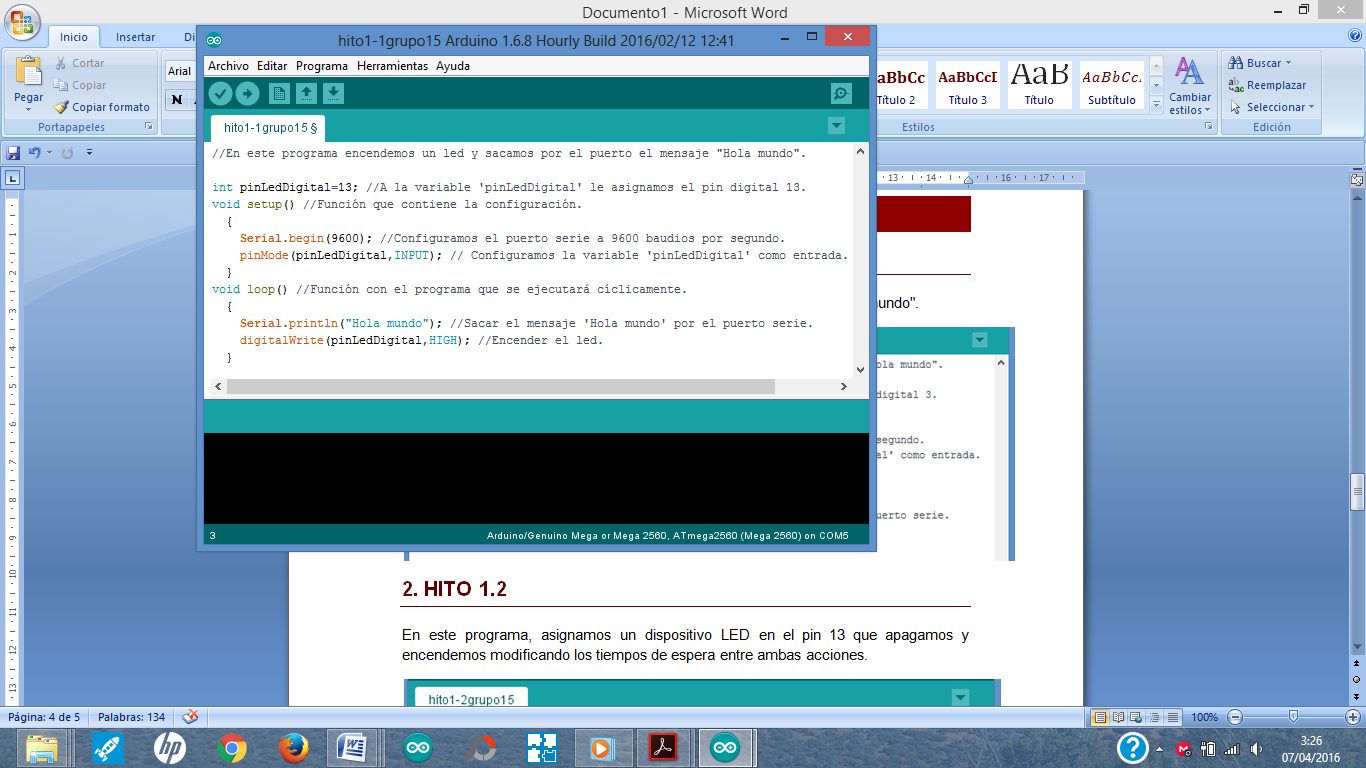




Finally, a window will appear indicating that the driver software has been updated.

1. HITO 1.1

In this program we assign an LED device on pin 13 that we turn off and on, modifying the waiting times between both actions. The LED connected to pin 13 is already integrated in the board, next to the switch. However, you can connect another LED with a resistor to any other digital pin (see Fig. 1). You only have to change the number 13 in the code by the pin where you have connected it.



In this example you will see how to make your first "Hello world" in Arduino. To do this, a pin will be configured in digital mode and as an output to turn on a led. In addition, the serial terminal will be configured to send messages from the Arduino to the computer, and you can see them on the screen of your PC. In this example, we tell you very briefly how to send text strings to the serial port to debug your program. To print the "hello world" message on the screen, use the instruction Serial.println ("Hello world");

On the instruction Serial.println comment that, although we will see in some milestone later in depth, Arduino has a "serial port" that allows to send from Arduino with the USB cable to the PC data and that can be viewed. For this, the Arduino software has a "Serial Monitor" window that if pressed allows you to see everything that is sent with the Serial.println function from the Arduino board to the PC. This type of instructions are very useful to know where your program is going, what values ​​you read from the connected sensors, or to visualize the result of some operation. It basically means "Print on screen". You will see that within the sequence "setup ()" is the instruction Serial.begin (9600), this is so that the data sent by the Serial Monitor of Arduino is sent at 9600 bits per second. Later we will talk about how to modify this speed. At the moment, mark this speed by default.

In addition, in the code, you will also see the instructions to turn on an LED that is previously physically connected to a digital pin. You must use the digitalWrite instruction (see Figure 1 to see what digital pins are available). The sequence of instructions to turn on / off a led is the following: digitalWrite (pin, value), where pin is the number of the pin where the Led is connected and value can be HIGH (turn on) or LOW (turn off). Actually it is as if we were sending the LED a digital 1 when we want to turn on and a digital 0 to turn off. Think what voltage could be measured on that pin when we put a digital 1 via software?

Once you have understood the code, the next thing we will do is download it to the plate. For it:

1) We opened the Arduino environment.

2) Create a new file (File -> new).

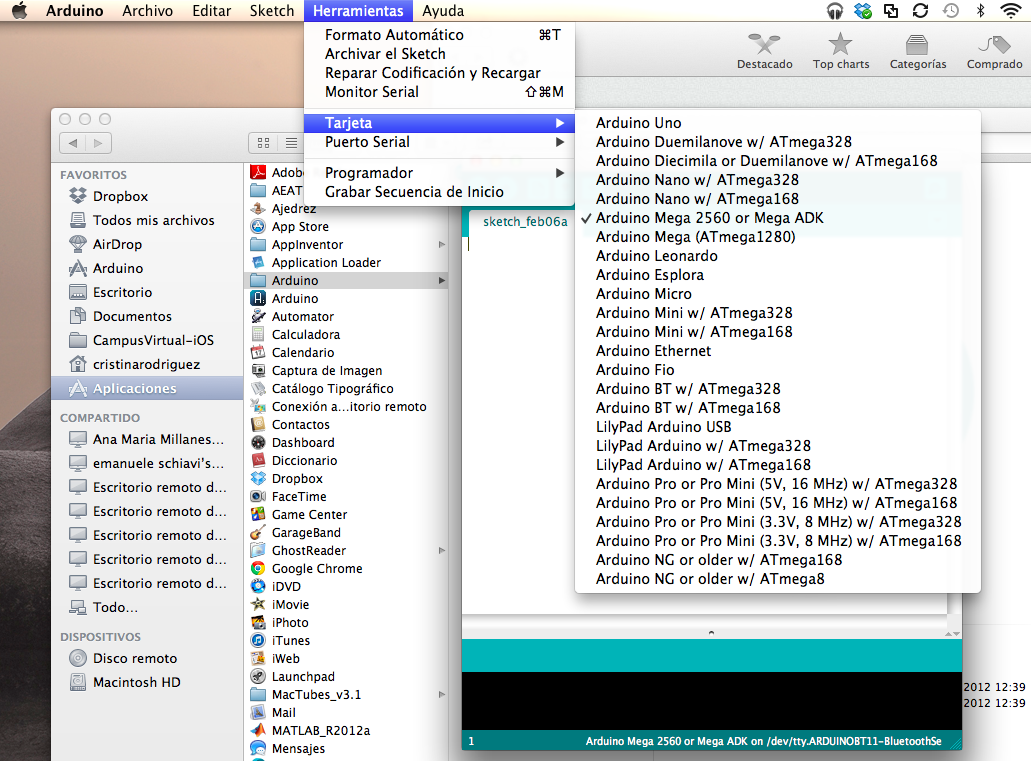
3) We write in that same file that the code of the image of this Milestone 1.1 is opened (it is better to write the code by hand, if you copy / paste from a file, undesired codes may appear that produce errors in Arduino).

a) Note: Right after printing "Hello World", add the instruction → Delay (8000). This will allow you to exit the screen of the Serial Monitor of your Arduino interface the word "Hello World" every 8 seconds. Otherwise, it will appear continuously without any pause.

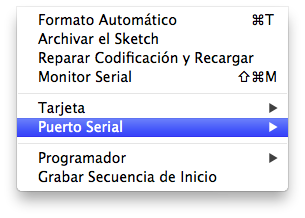
4) We save the file with the name HelloMundo.ino

5) We connect the Arduino MEGA to the computer with the USB cable that comes with the practice kit.

6) In the arduino environment, select → Tools → Card. In the drop-down menu we choose the one called Arduino MEGA as in the following figure.



1) We select the serial cable to control the board. To do this, in the same Tools menu, select "Serial Port" and in the drop-down menu we choose the virtual serial port that was created when the board was connected (see the following figure). How to know? It is usually the last one on the list, but in the control panel of your computer, in the hardware management option you can find the answer.

a) Note: Windows follows the following structure in the name COM1, COM2, COM3 ... In Linux / dev / ttyUSB0 and in Mac /dev/tty.USB0...

1) Click on the Record icon (which is an icon with an arrow facing to the right) on the left side of the interface. We will see that in the lower part of black color a series of messages will appear. When we have finished recording, it will indicate it to us by that same screen.

Note: we can find some error associated with the fact that the card is not well identified, you have not correctly set the port of the serial port or that there is a syntax failure. Syntax errors refer to the fact that you did not write the code correctly. The same environment tells us in which line the error is.

2) To see the famous "Hello world" on the screen of your computer, click on the icon "Monitor Series" that is on the right side of the Arduino environment, top (if you put on the icon without pressing it will come out on your part left what it means and what it is for). We will open a serial monitor (called hyperterminal) and after a few seconds (which takes to start the plate) + the delay time of the code (8000 milliseconds = 8 seconds), a message will appear on the screen "hello world" it will be repeated every 9000 milliseconds.

3) We try to change the delay or add new phrases to be sent by the Serial Monitor to practice this type of simple instructions.

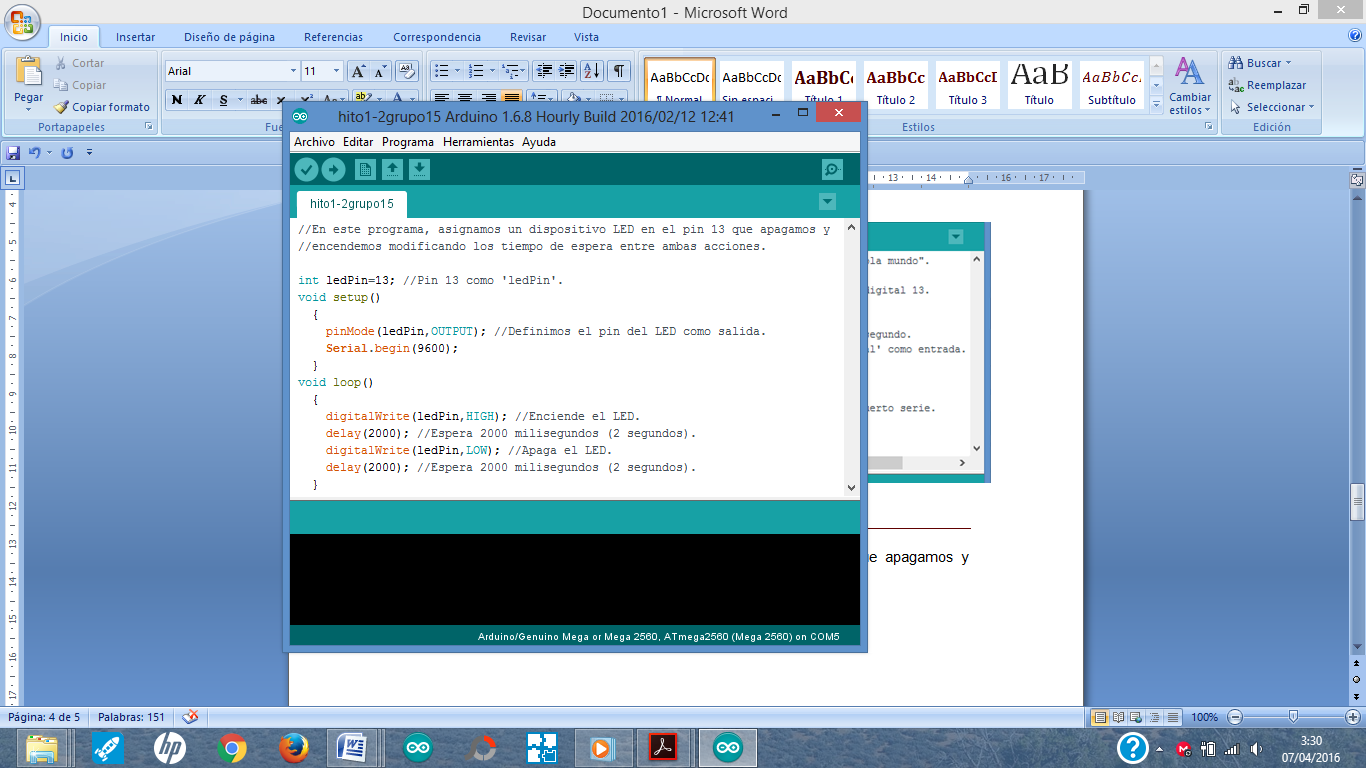
1. HITO 1.1.2

Now to practice in the simulator, look at this simple visual guide:

<https://www.youtube.com/watch?v=5Bn-3K6SXcA>

2. HITO 1.2

In this program we will take a few steps similar to the previous one, only we will turn on and off one LED every X time. To do this, we assign an LED device on pin 13 that we will turn off and on by modifying the waiting times between both actions. For this, we will use the delay command. Its syntax is the following, where time\_ms is a number that represents the time in ms that you want the program to be paused:

delay (time\_ms);

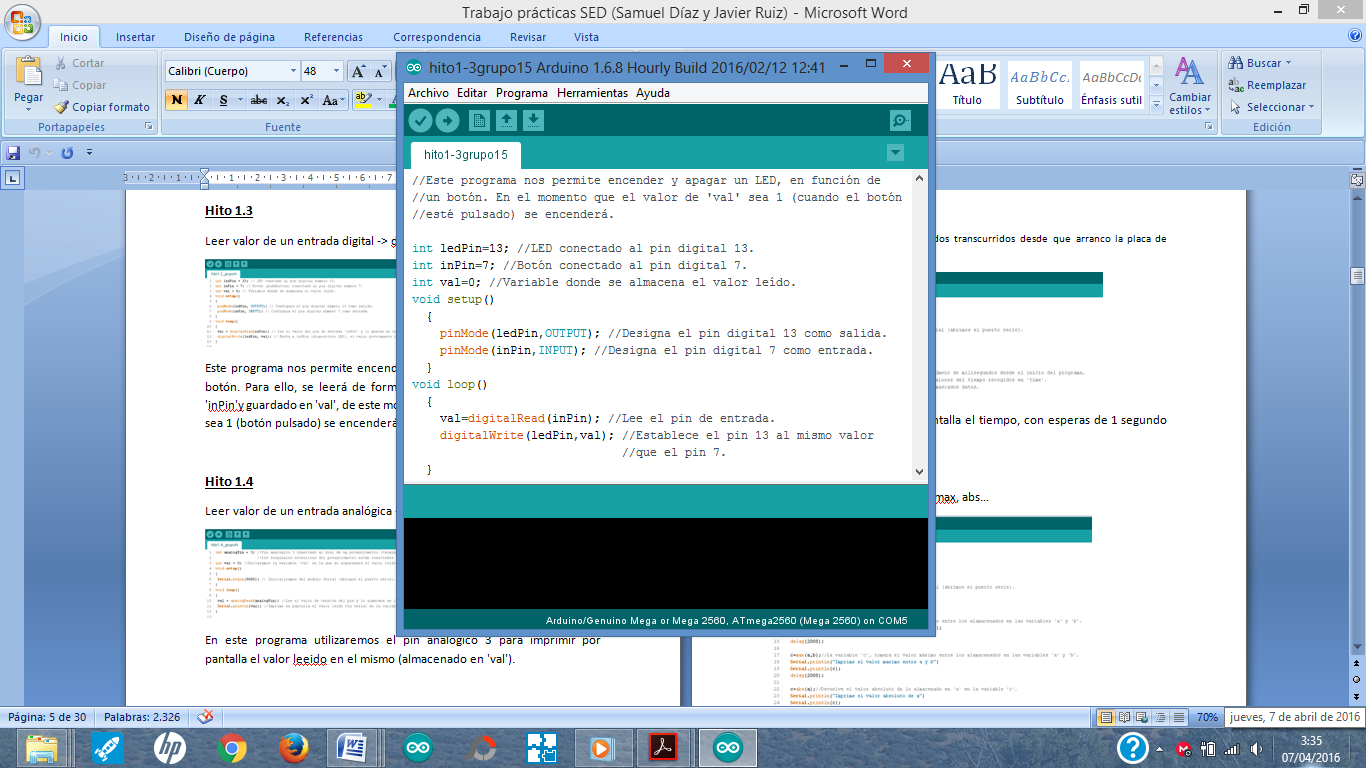
3. HITO 1.3

This program allows us to turn on and off an LED connected to digital pin 13, depending on the value read from a button connected to pin 7 digital.

To make this milestone we use the digitalRead command. The syntax of digitalRead to read the contents of an element that can only have 2 states (for example: on or off), that is connected to a certain pin, and know if it is on or off is the following:

val = digitalRead (pin)

where pin represents the number of the pin to which the component is connected (in this case a button) and val the variable in which the digital value of said button is stored (it can also be a led because it has two states: on or off ). At the moment when the value of 'val' is 1 (when the button is pressed) the Led will light up.



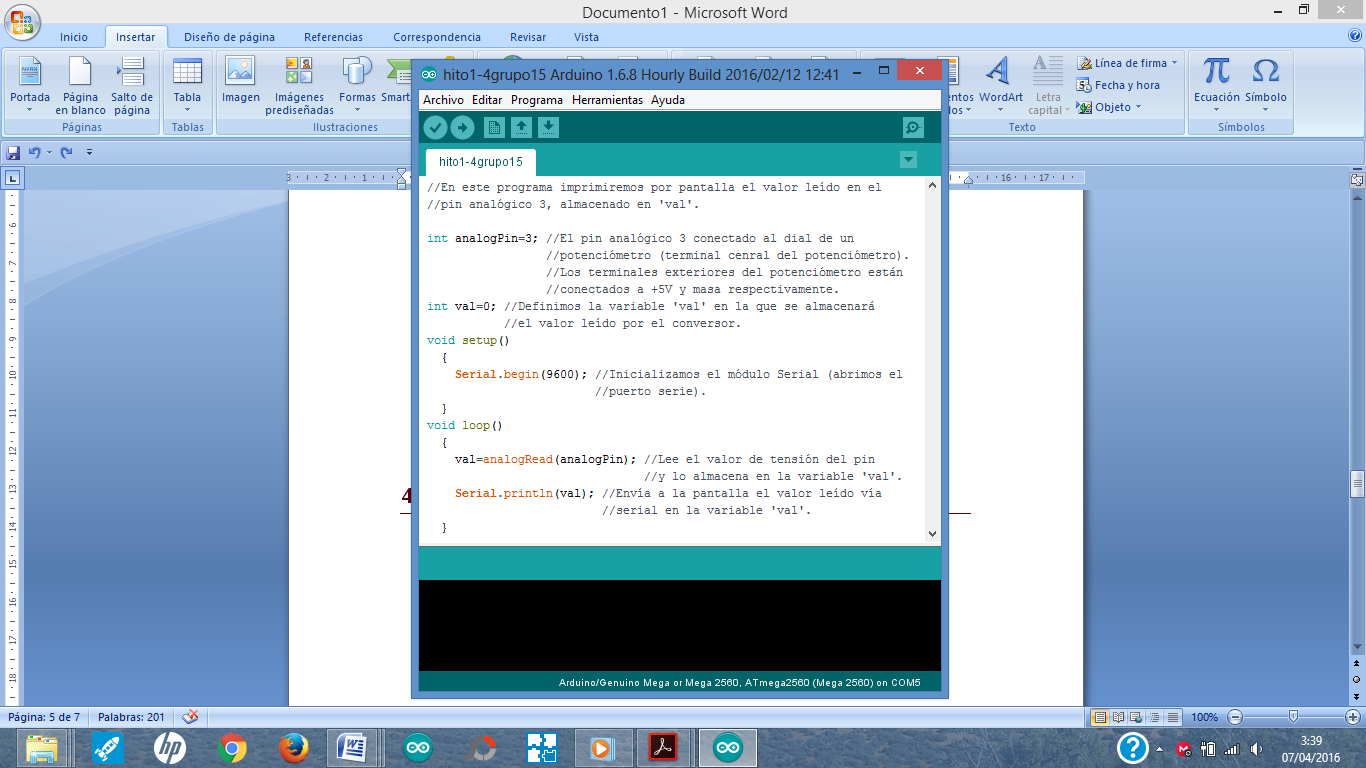
4. HITO 1.4

When we read the digital value you will have seen from the previous example that you will only get 0 or 1, depending on when the button was pressed. In the following program, we will see an example when what we have is a sensor or element connected to an analog pin, its value can oscillate between 0V and 5V (5V if it is fed to 5). Check the arduino website to see the resolution of the conversion.

The "analogRead" command reads the voltage value on the analog pin specified in parentheses. The Arduino MEGA board has 16 channels connected to a 10-bit digital analog converter. This means that it will convert voltages between 0 and 5 volts to an integer between 0 and 1023 (what you'll get in the value field). This provides a resolution in the reading of: 5 volts / 1024 units, that is, 0.0049 volts (4.9 mV) per unit. The input range can be changed using the analogReference () function.

The converter takes approximately 100 microseconds (0.0001 seconds) to read an analog input so it can carry a maximum reading rate of approximately 10,000 readings per second.

The following program prints on the screen the value read on analog pin 3, which will then be stored in 'val', finally, the value of the variable 'val' is printed on the screen.

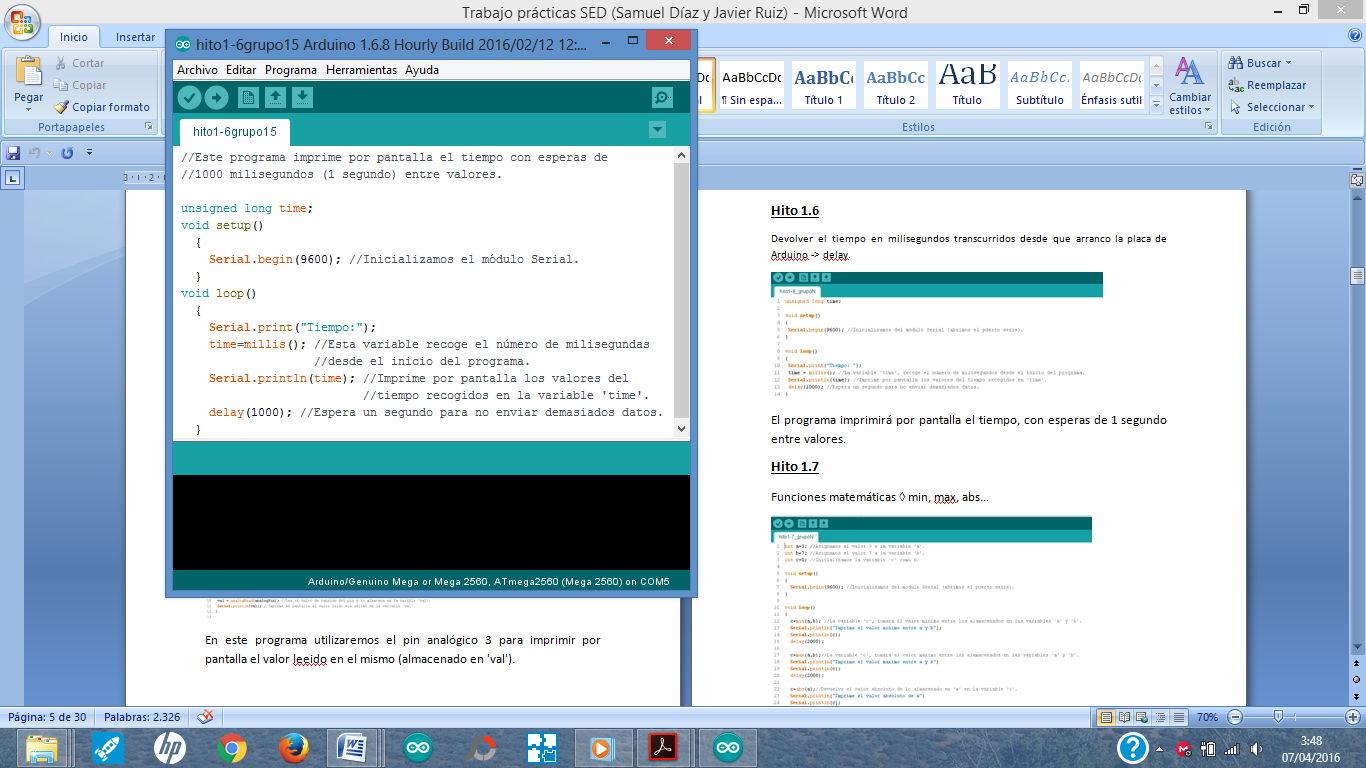


5. HITO 1.5

In this milestone, we will have to change the previous milestone to do the following. If the value of "val" is 0, then a green LED will light and a red LED will be off. If the value of "val" is 1, then a red LED will light and the green LED will be off. If you do not have LEDs of those colors, you can use other colors to perform the same function. Also, make sure that at least 2 seconds pass between each analogRead reading.

6. HITO 1.6

This program prints on screen the time with waits of 1000 milliseconds (1 second) since I start the program with the "millis" command. Then, print your content on the screen.



7. HITO 1.7

Now let's practice with some Arduino math functions, such as those that allow you to calculate minimum, maximum, absolute and square root.

Some mathematical functions and their syntax:

• Returns in variable c the minimum of two values a and b → c = min (a, b).

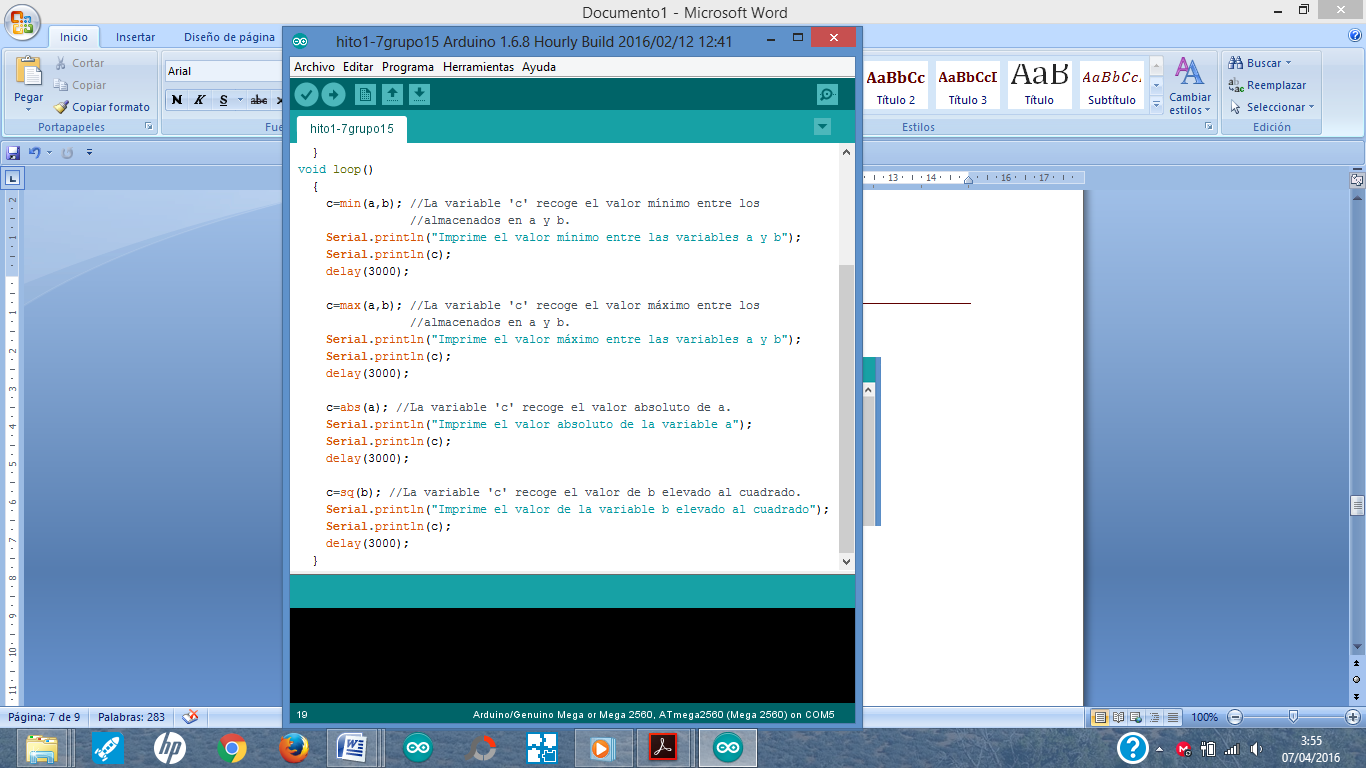
• Returns in variable c the maximum of two values a and b → c = max (a, b).

• Returns in variable c the absolute value of a value → c = abs (var).

• Returns in c the square of a value → c = sq (var).

a, b and var are names of variables that may contain numbers.

The following program prints on the screen the different values of the variable 'c', the minimum and the maximum value between 'a' and 'b', the absolute value of 'a' and the square of 'b'. It prints with waits of three seconds..



8. HITO 1.8

To check if a condition "condition" is met, the conditions commands must be used. When a condition is met, the instructions that are between the keys are executed sequentially. There are two types of conditionals: "If it is done, I do X" and "If it is done, I do X, if I do not do Y". The syntax is the following:

Syntax and description for IF

If (condition)

{

// here instructions

}

Syntax and description for IF ELSE

If (condition)

{

// here instructions yes "condition" is met

}

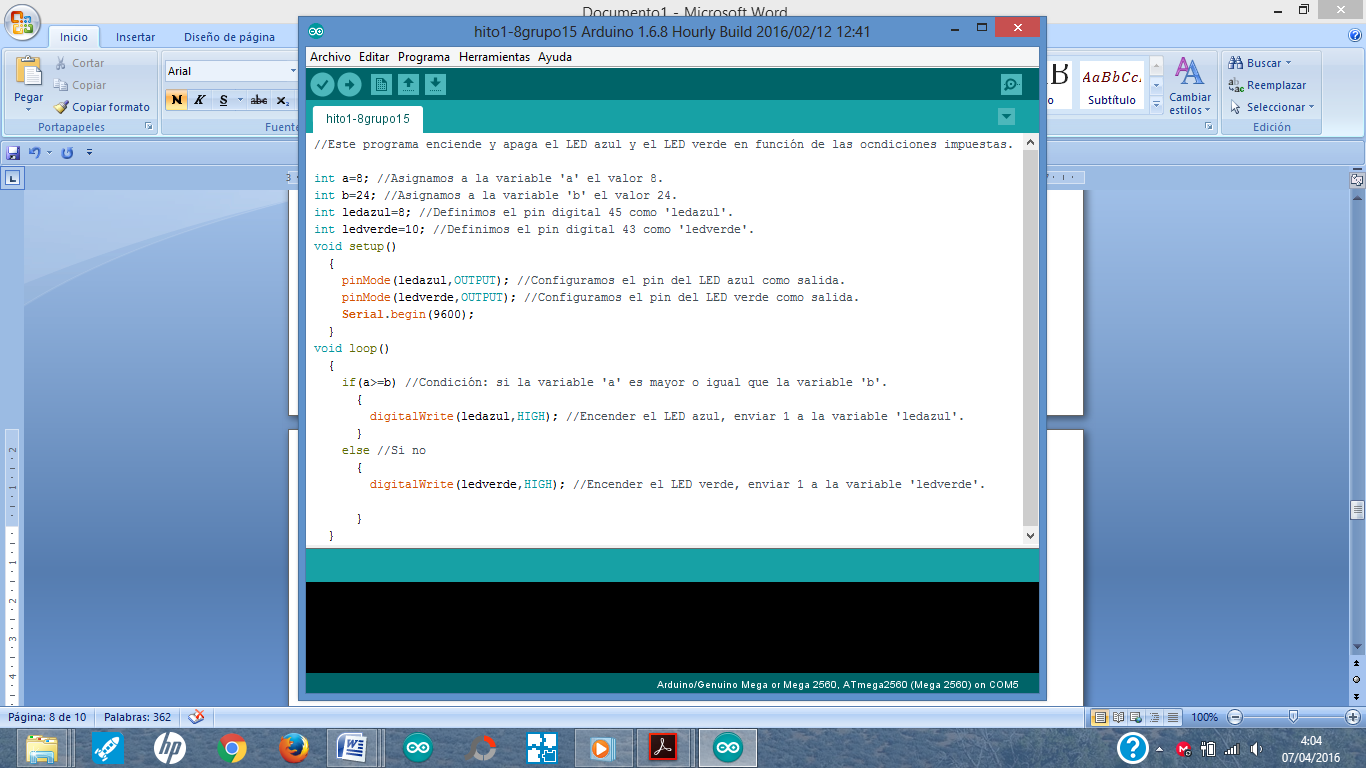
else

{

// here instructions if "condition" is not met

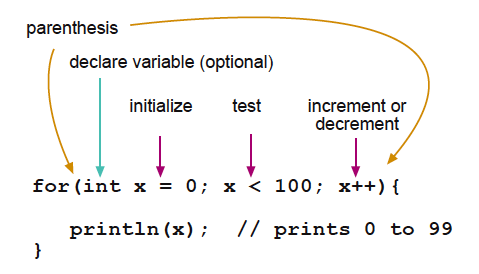
}

To test this type of statement with conditions, the following program turns the blue LED and the green LED on and off depending on the conditions imposed using the conditional instructions → if, else.



9. HITO 1.9

Next, we will see an example to perform instructions with loops. We will see first the instruction or "for" command. The operation and the syntax are described in the following example:



This program increases the value stored in the variable 'PWMpin' from 0 to 255 by means of a for loop.



10. HITO 1.10

The next command to perform conditional loops is while. It is used to perform a set of instructions enclosed between the keys of the while loop, as many times as the condition that exists in parentheses in the while loop. The conditions are generated the same as those of the IF and ELSE explained in milestone 1.8

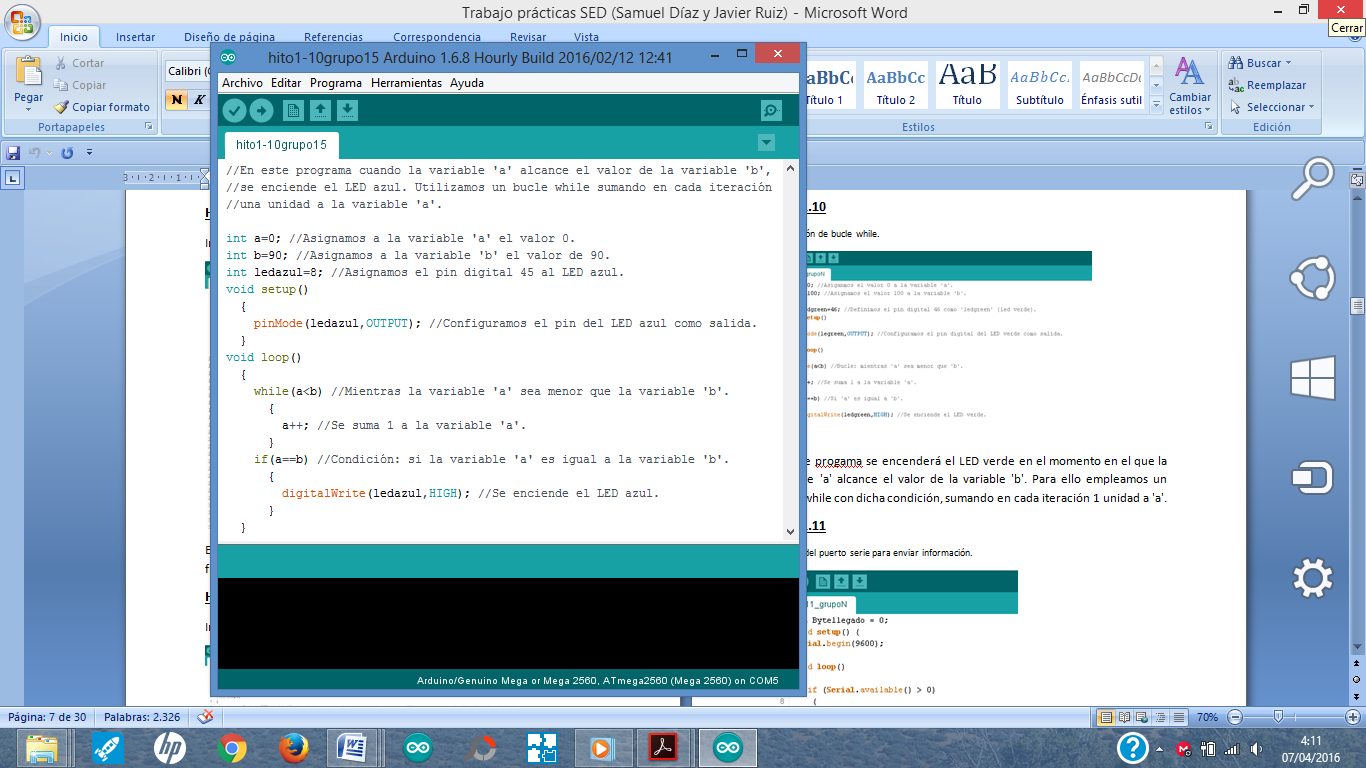
The syntax is the following:

while (expression) {

   // statement (s)

}

To see an example, in the following program when the variable 'a' reaches the value of the variable 'b', the blue LED lights up. For this, we use a while loop adding in each iteration of the loop a unit to the variable 'a'.



11. HITO 1.11

In this program we will see an example to learn a little more about the control and management of the Serial port of Arduino. So far, you have used it to print things on the "Serial Monitor" screen of Arduino.

The serial port serves to receive and send data to the microcontroller through the UART that is connected from the microcontroller to the USB adapter. That way you can see what happens through the Monitor Series (icon that is on the right of the whole with an arrow to the right in your Arduino environment) - WE RECOMMEND THAT YOU REVIEW THE THEORETICAL CONTENT OF THE UART SEEN IN CLASS TO UNDERSTAND THIS PART.

There are several important instructions related to the Serial port:

Instruction Serial.begin (baud): We will start configuring the serial port. Below we show you what the instruction is, it will sound like a class. Open the serial port and set the baud rate for serial data transmission. The typical speed value to communicate with the computer is 9600, although other speeds can be supported. It is always configured in the setup.

void setup ()

{

Serial.begin (9600); // open the serial port

} // setting the speed to 9600 bps

Instruction Serial.println (): Prints the data on the serial port, followed by an automatic carriage return and line feed. This command takes the same form as Serial.print (), but it is easier to read the data in the Monitor Series of the software.

Note: For more information on the different possibilities of Serial.println () and Serial.print (), see the Arduino website.

There are different formats to print data in different formats by the Serial port. Next, we indicate the different options to write by the serial port in different formats.

• Serial.println (b) dumps or sends the value of b as a decimal number in characters in ASCII format followed by "CR" and "LF".

• Serial.println (b, DEC) dumps or sends the value of b as a decimal number in ASCII characters followed by "CR" and "LF".

• Serial.println (b, HEX) dumps or sends the value of b as a hexadecimal number in ASCII characters followed by "CR" and "LF".

• Serial.println (b, OCT) dumps or sends the value of b as an Octal number in ASCII characters followed by "CR" and "LF".

• Serial.println (b, BIN) dumps or sends the value of b as a binary number in ASCII characters followed by "CR" and "LF".

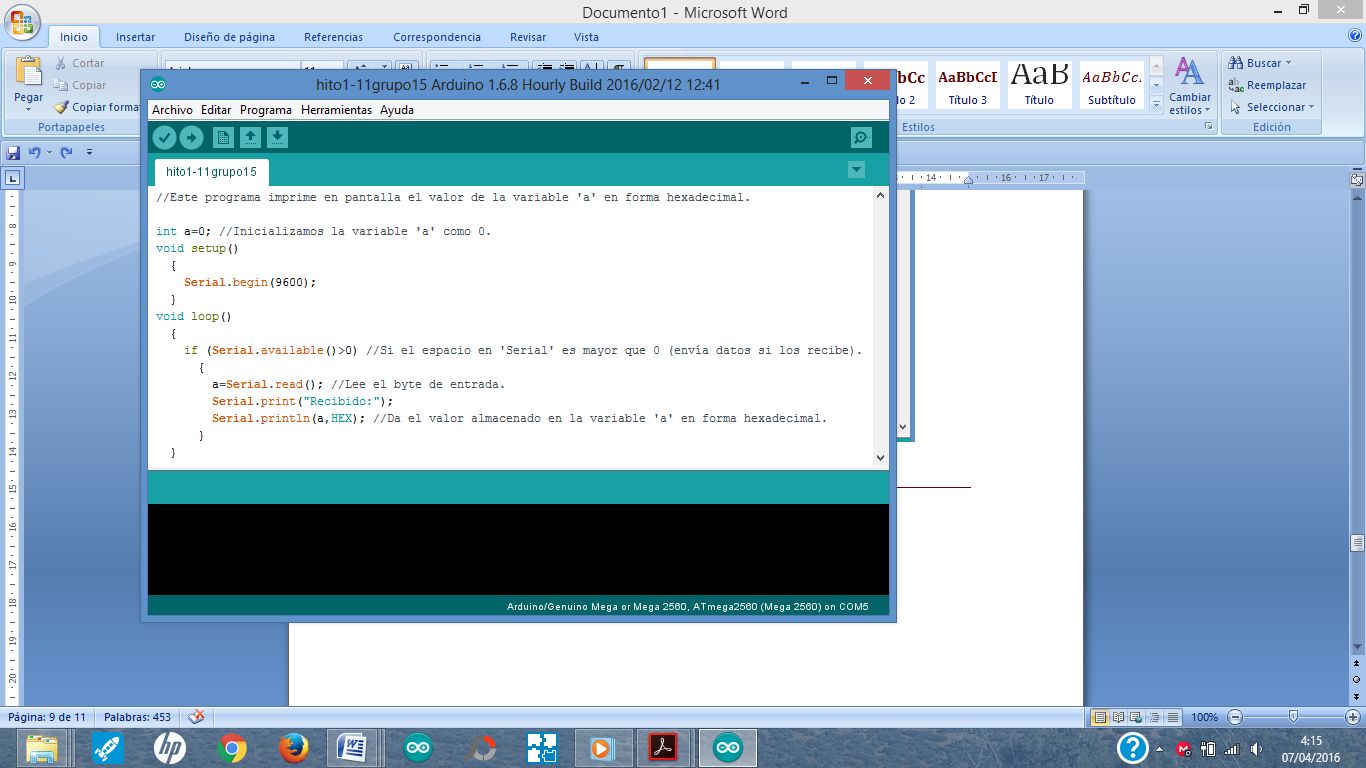
• Serial.print (b, BYTE) dumps or sends the value of b as a byte followed by "CR" and "LF".

• Serial.println (str) dumps or sends the string as an ASCII string followed by "CR" and "LF".

• Serial.println () is used to print an empty line break, without characters. Its operation is equivalent to the printNewline () statement.

Well, the following program we will see an example to read also from Arduino the data that you enter by the PC with the keyboard and visualize it in a format. In particular, the following program example prints the value contained in the variable 'a' in hexadecimal form on the screen.

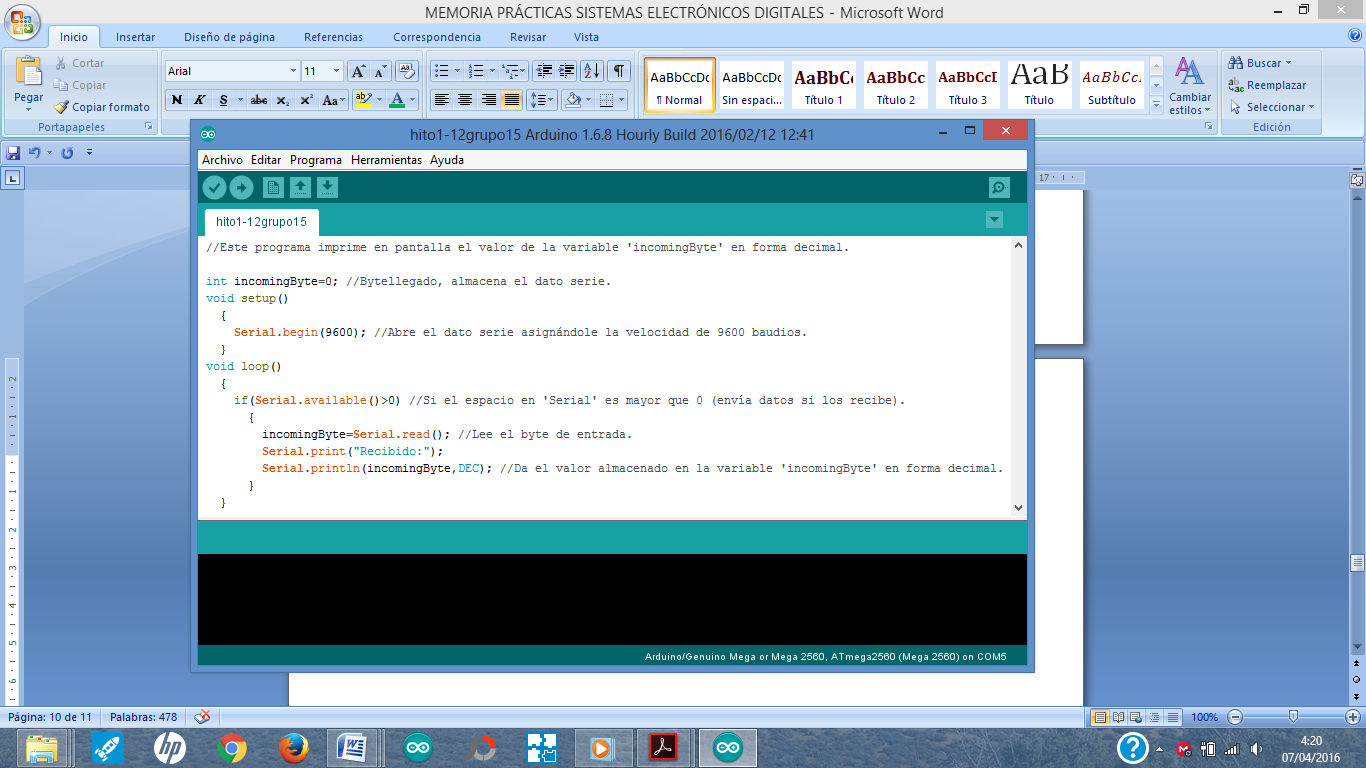
.



12. HITO 1.12

This program prints on the screen the value of the variable 'incomingByte' in decimal form.

Decimal form of an 'incomingByte' value → Serial.println (incomingByte, DEC).

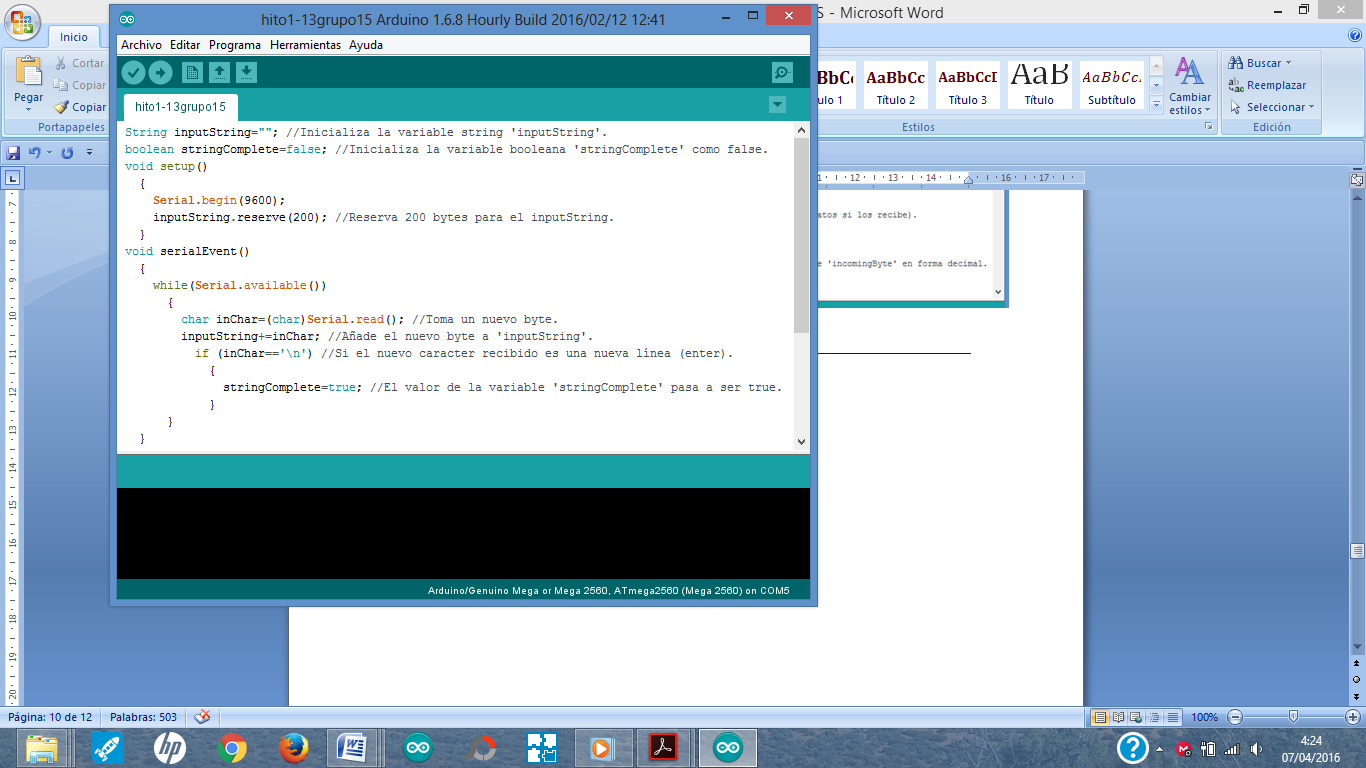
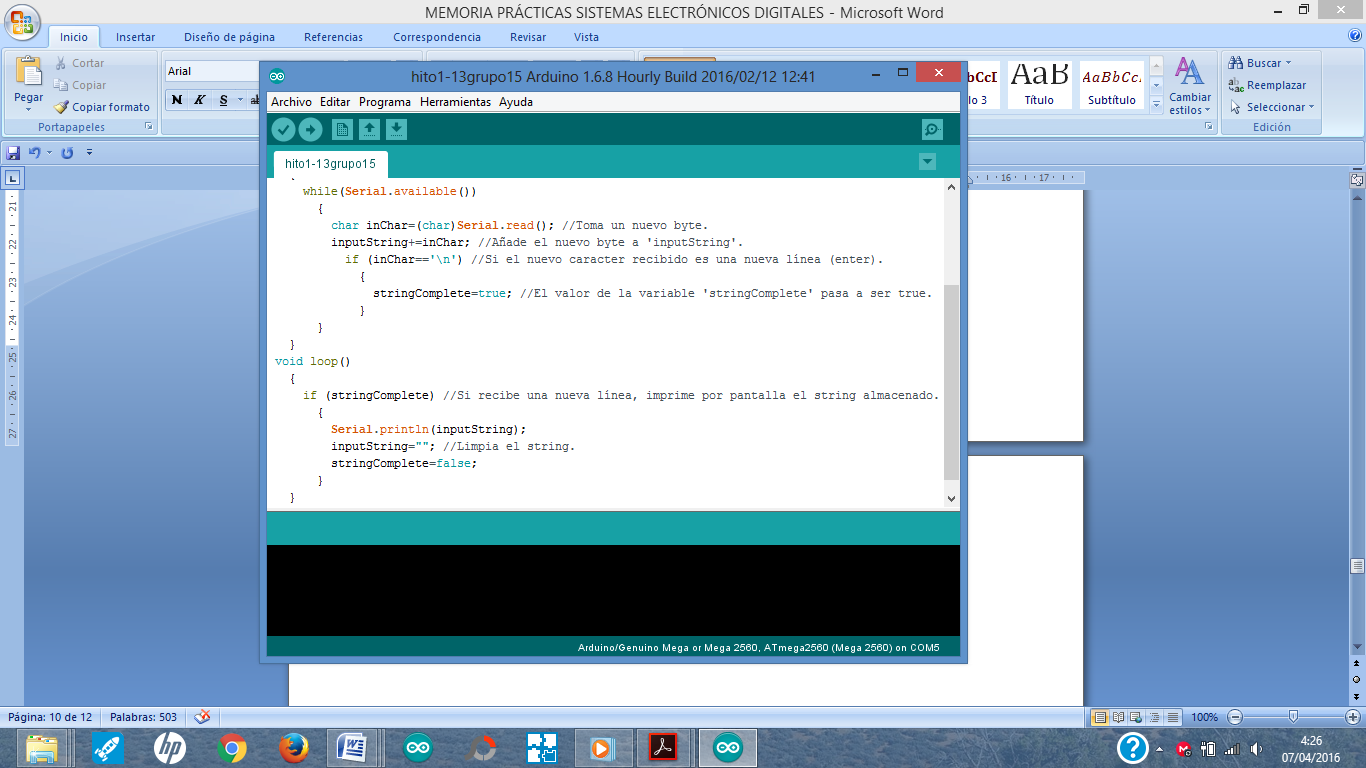


13. HITO 1.13

In the following program we will see an example for the control of the serial port to receive or read information. Specifically, in this example the port is configured to 9600 baud, then an interrupt is configured that will attend an event of the serial port.

An interruption is like an instruction that remains in the background, waiting for something to happen, and when it occurs then a set of instructions is executed. In particular, the interruption is called SerialEvent (), which is an interruption that is made by default in Arduino, that is, you do not have to configure it. But what you must do is create the content of the body of the instruction, what goes in between the keys. What goes between keys is what will be executed every time the interruption occurs, in this case, every time there is an event in the serial port (eg, that there are data pending reading). Then, within this interruption it enters a loop while there is data to be read by the serial port. When there is no more data to read, there is a carriage return, and therefore, the phrase received by the serial port is complete and ready to be processed. An example would be to receive by the serial port "Hello this is an example", once it reaches the word example the last character "or" will be followed by a line break that is what allows to identify the end of the chain. In this case, the Boolean variable stringComplete will be set to true. Also, if you notice, there is a Serial.read () instruction that allows you to read from byte to byte, and each byte is saved in the inChar variable. Well, what happens is that first of the phrase "Hello this is an example" will read the 'H', then the 'o', then the 'l', and so on until the end of the sentence (or string) .

In this case, the loop () is in a permanent wait state until a complete character string has been read, that is, until stringComplete has become true. Then, once it is set to true, it prints on the screen the content read by the serial port, which has previously been stored from the serialEvent function in the inputString variable.



**When is something displayed on the screen?**

**Through the command Serial.printl () the value of the variable indicated in parentheses is printed on the arduino screen.**

**What does the example use to concatenate strings?**

**By means of the inputString + -inChar command, the entered characters are added until they stop indicating a new string, by means of another variable, such as "enter", which means that the chain ends.**

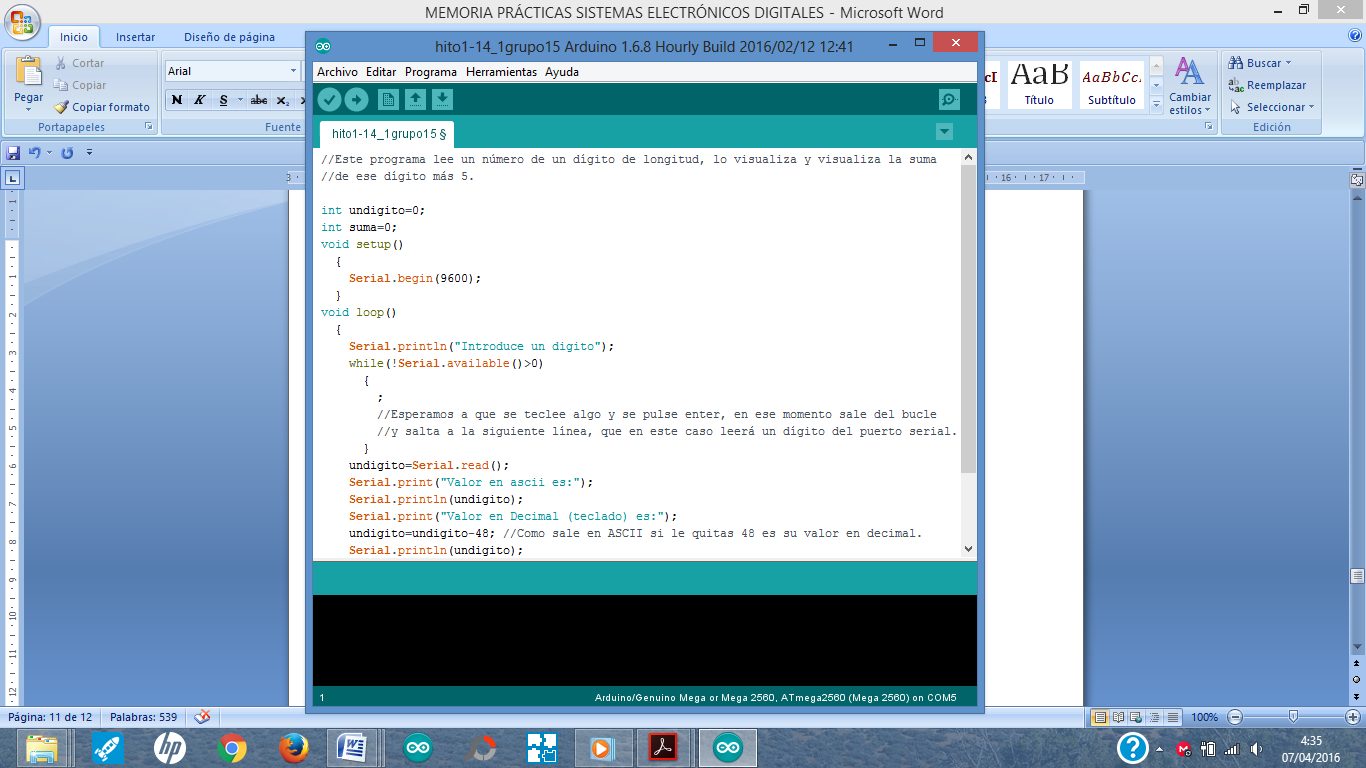
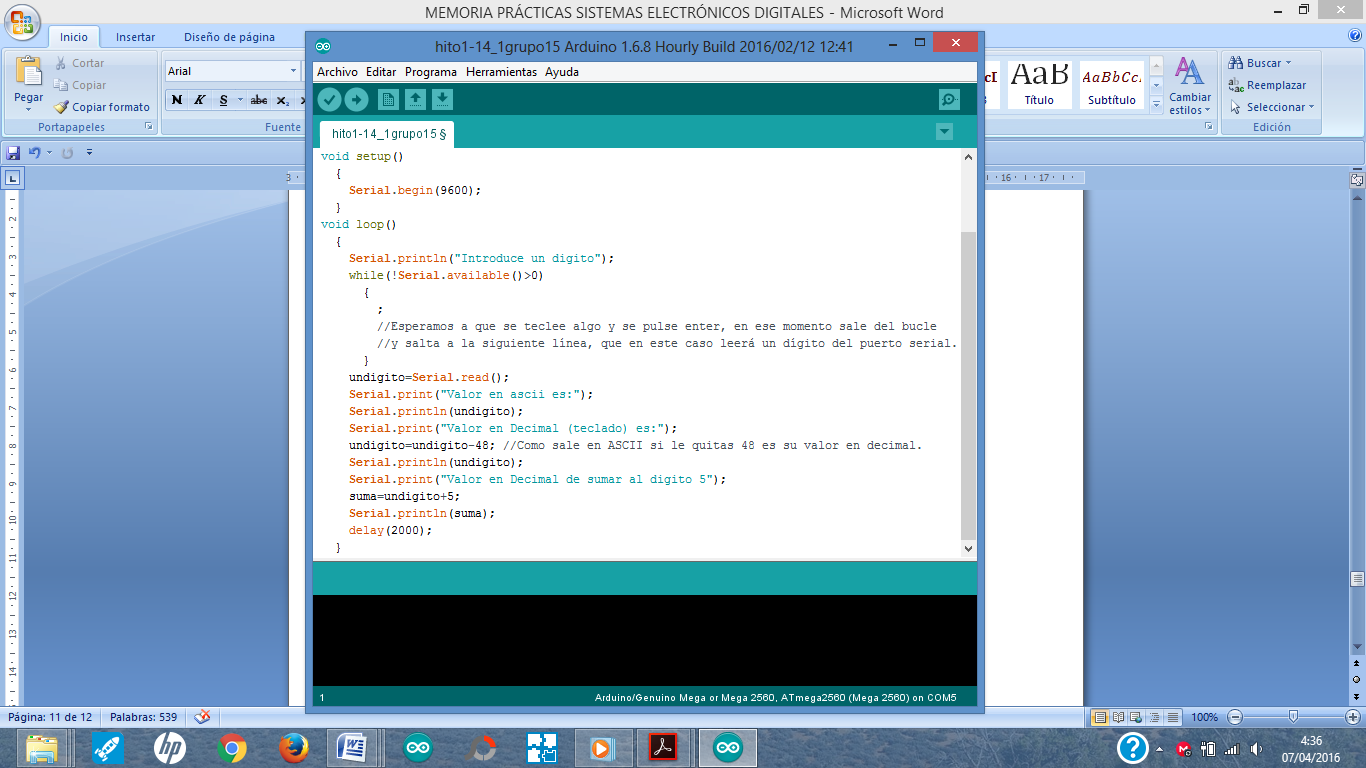
**The inChar variable refers to the different characters that are introduced in Arduino and that are received by the serial port.**

14. HITO 1.14

**14.1. HITO 1.14.1**

This program reads a one-digit length number, prints its value on the screen, and finally prints the sum of that digit plus five on the screen. This program is based on the example of Milestone 1.12. In addition, it allows to visualize the keyboard in Ascii format and in its decimal value. If you consult an Ascii table, you will see that the number in Ascii that corresponds to a digit is not the same as decimal. The difference is 48, so the operation you see in the example is performed. But this example only applies to numbers that contain a digit, that is, from 0 to 9.

.

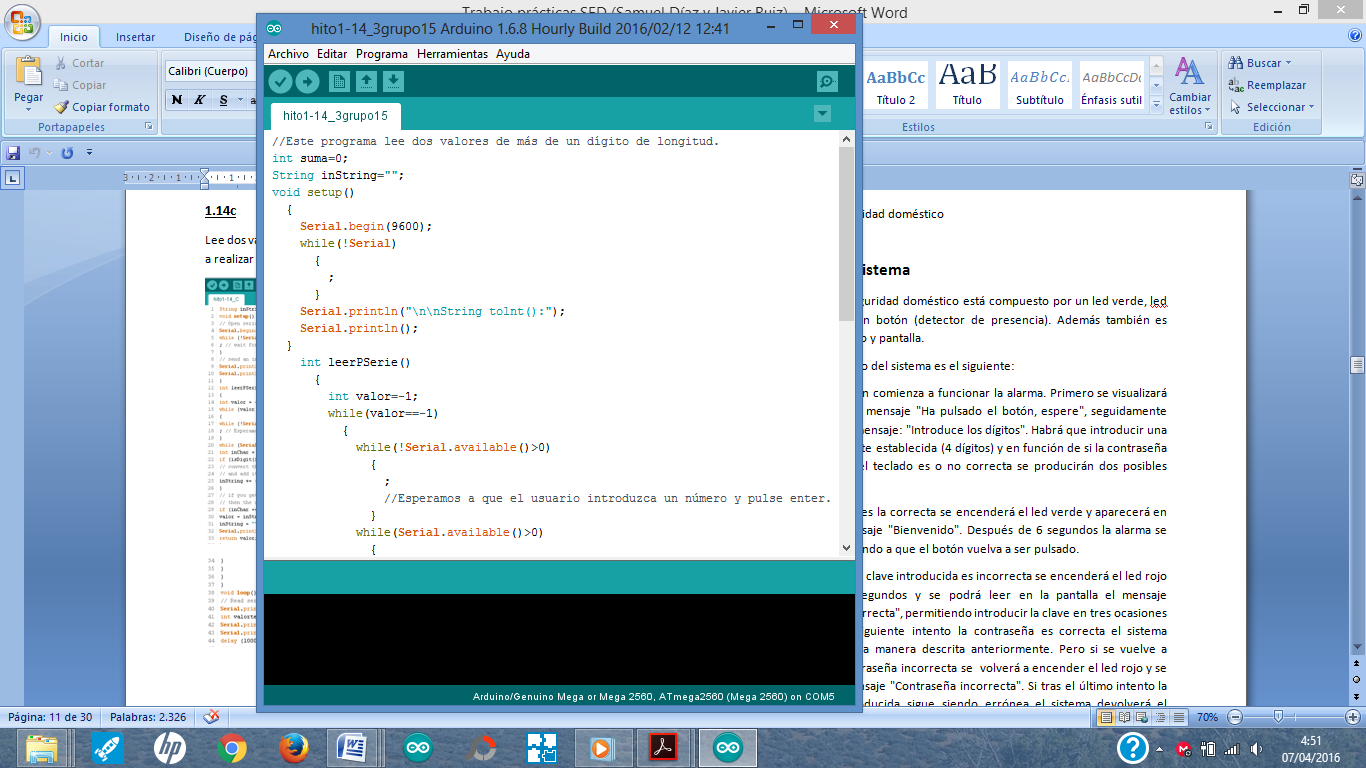
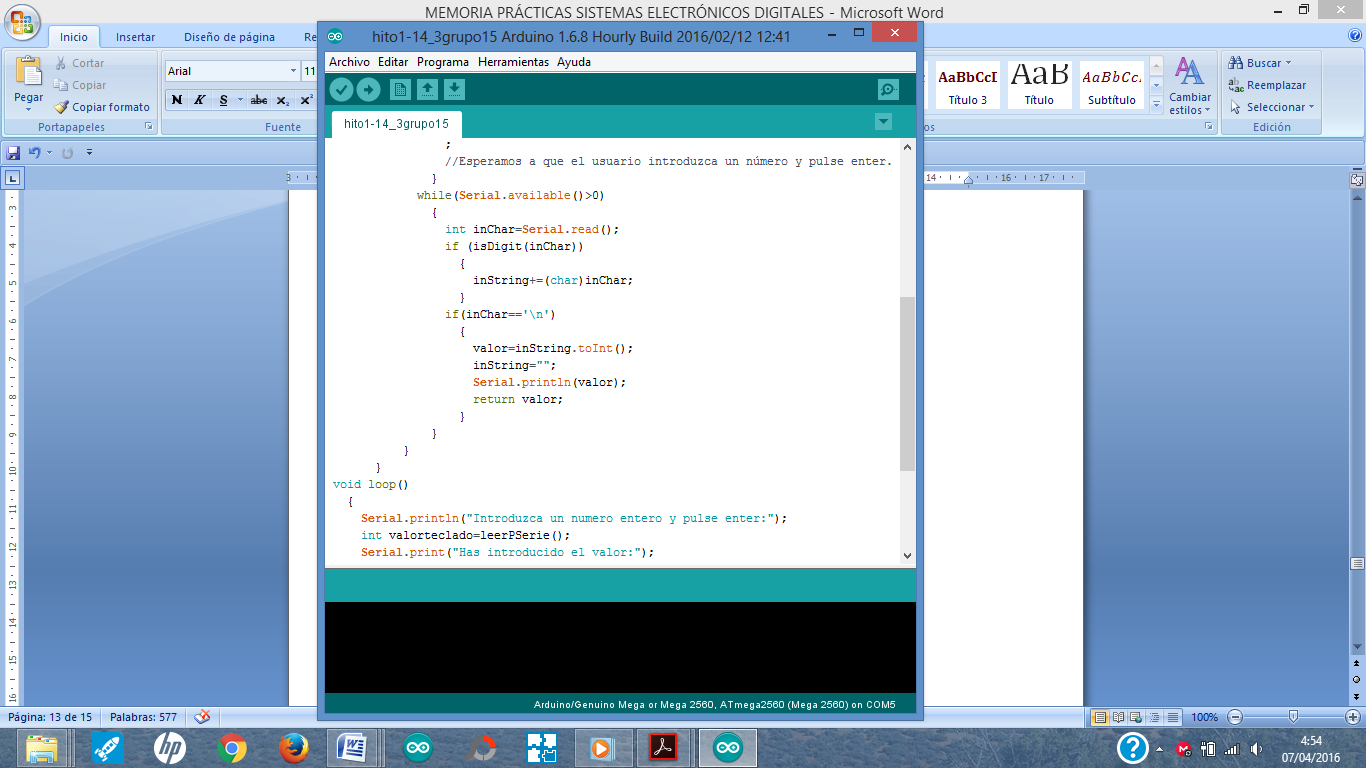


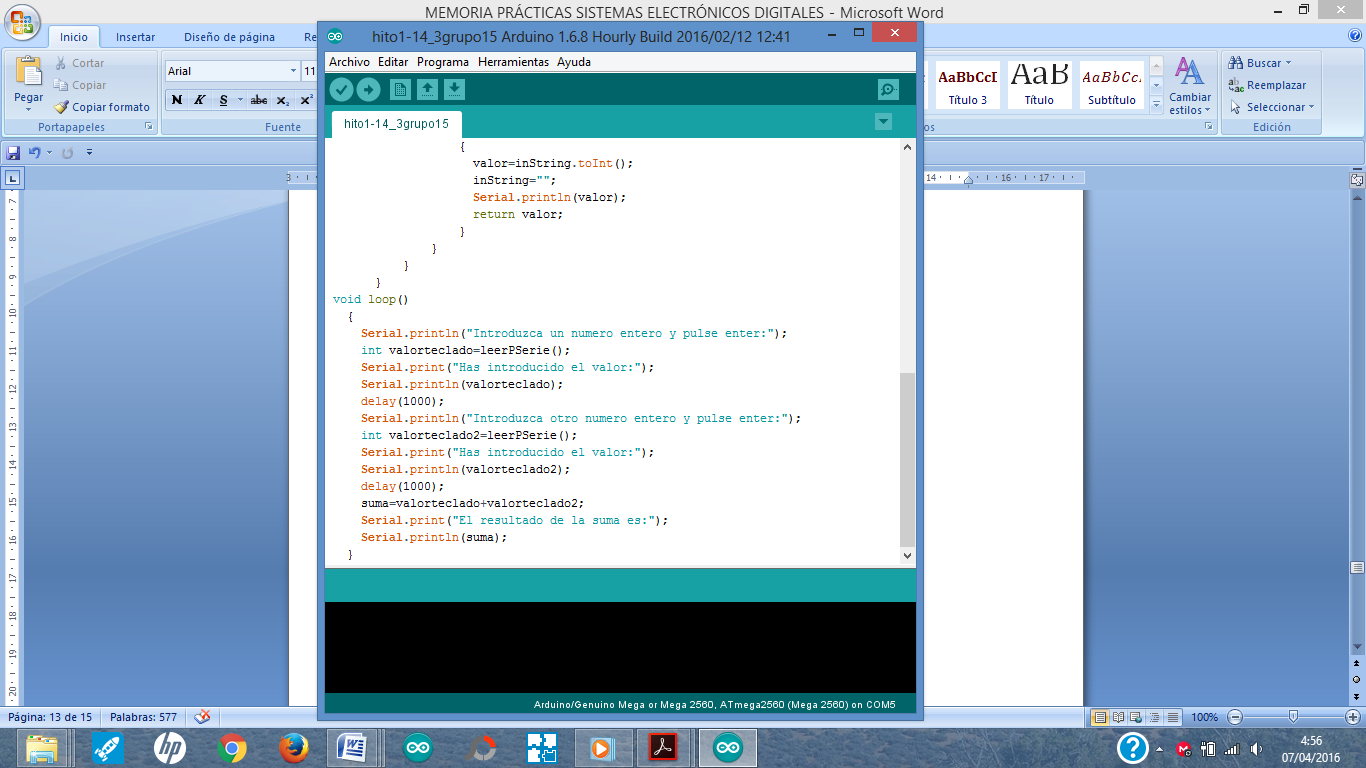
**14.2. HITO 1.14.2**

Try to make an example where instead of reading a digit you read two different numbers of a single digit and visualize on the screen the result of the sum of both, the multiplication and subtraction of both.

**14.3. HITO 1.14.3**

The next program reads two numbers more than one digit in length and makes a sum of both. Try modifying the example to make multiplication and subtraction.





**15. HITO 1.15**

In this milestone we will use "procedures" and "functions". In programming, a function is a group of instructions with a particular objective and which is executed when called from another function or procedure. A function can be called multiple times and even call itself (recurring function).

Functions can receive data from the outside when called through the parameters and must deliver a result. They differ from the procedures because they do not return a result. In general, functions must have a unique name in the field in order to be called, a type of result data, a list of input parameters and their code.

Syntax of a procedure:

void procedure\_name (input\_list\_list)

{

 // Here would go the instructions

}

where:

- procedure\_name must be a unique name throughout the program, and not be named like any other "reserved" procedure of the libraries for Arduino.

- input\_list\_list are the types of input variables that will be used within the instructions within the procedure. Its syntax is:

➢ Type\_variable1 name\_variable1, Type\_variable2 name\_variable2

Below, we show you an example of use. They are usually put before the setup () procedure but after the declaration of the global variables of your program:

void add (int a, int b)

{

int c;

c = a + b;

Serial.print ("The result of the sum is");

Serial.println (c);

}

At this moment, to be able to use it from any part of your program, for example, inside the loop would be to include this:

... // inside the loop

int n1, n2;

n1 = 5;

n2 = 6;

add (n1, n2);

...

at this moment, when you run your program through the "Serial Monitor" it would appear "The result of the sum is 11");

Now, let's see what a function is like. A function returns a value, so let's now return the value of the sum in another variable. The goal is to be able to use that value for another calculation or whatever you want to do in your program that is useful.

Syntax of a function:

Output\_type function\_name (input\_list\_list)

{

 // Here would go the instructions

}

where:

- Output\_type: Is the value of the type of the variable that contains the output result of the function.

- function\_name must be a unique name throughout the program, and not be named like any other "reserved" library procedure for Arduino.

- input\_list\_list are the types of input variables that will be used within the instructions within the procedure. Its syntax is:

➢ Type\_variable1 name\_variable1, Type\_variable2 name\_variable2

Below, we show you an example of use. They are usually put before the setup () procedure but after the declaration of the global variables of your program:

int add (int a, int b)

{

int c;

c = a + b;

return c;

}

At this moment, to be able to use it from any part of your program, for example, inside the loop would be to include this:

... // inside the loop

int n1, n2, c;

n1 = 5;

n2 = 6;

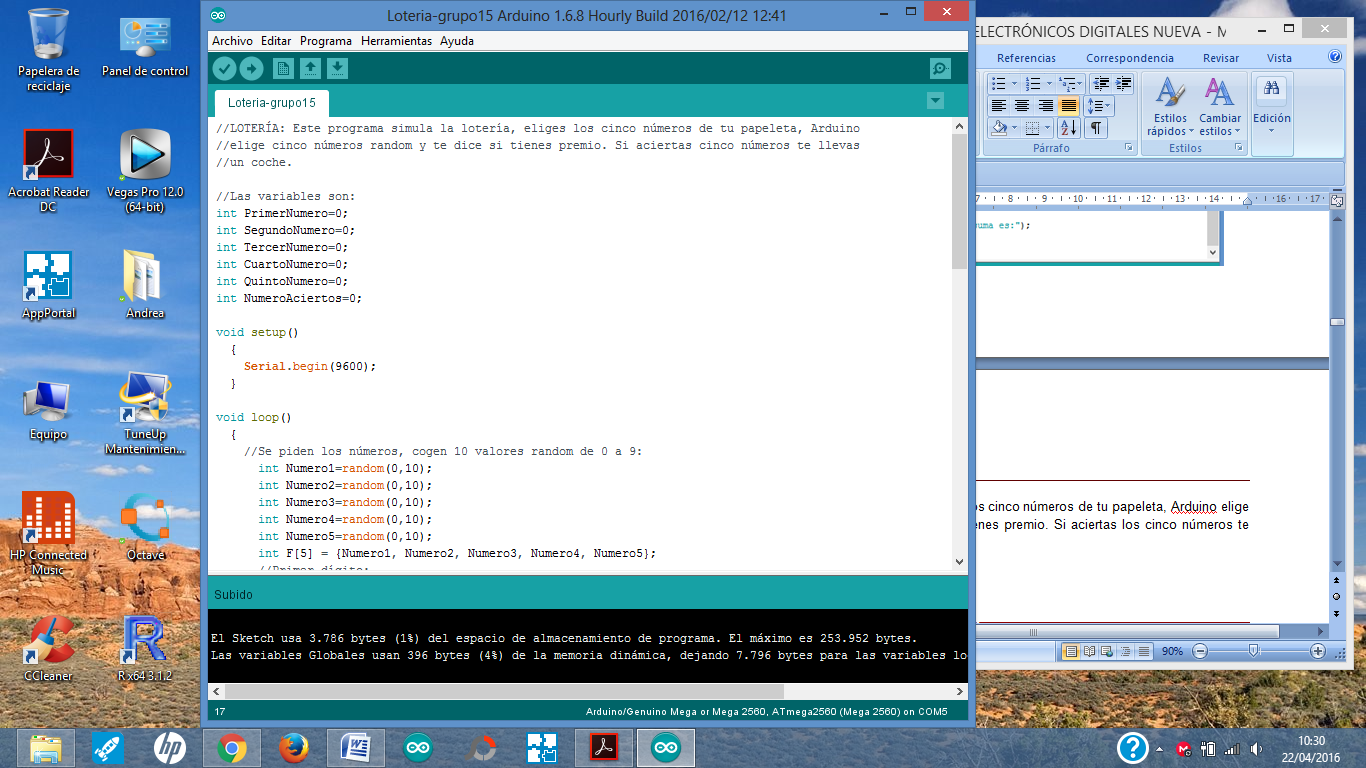
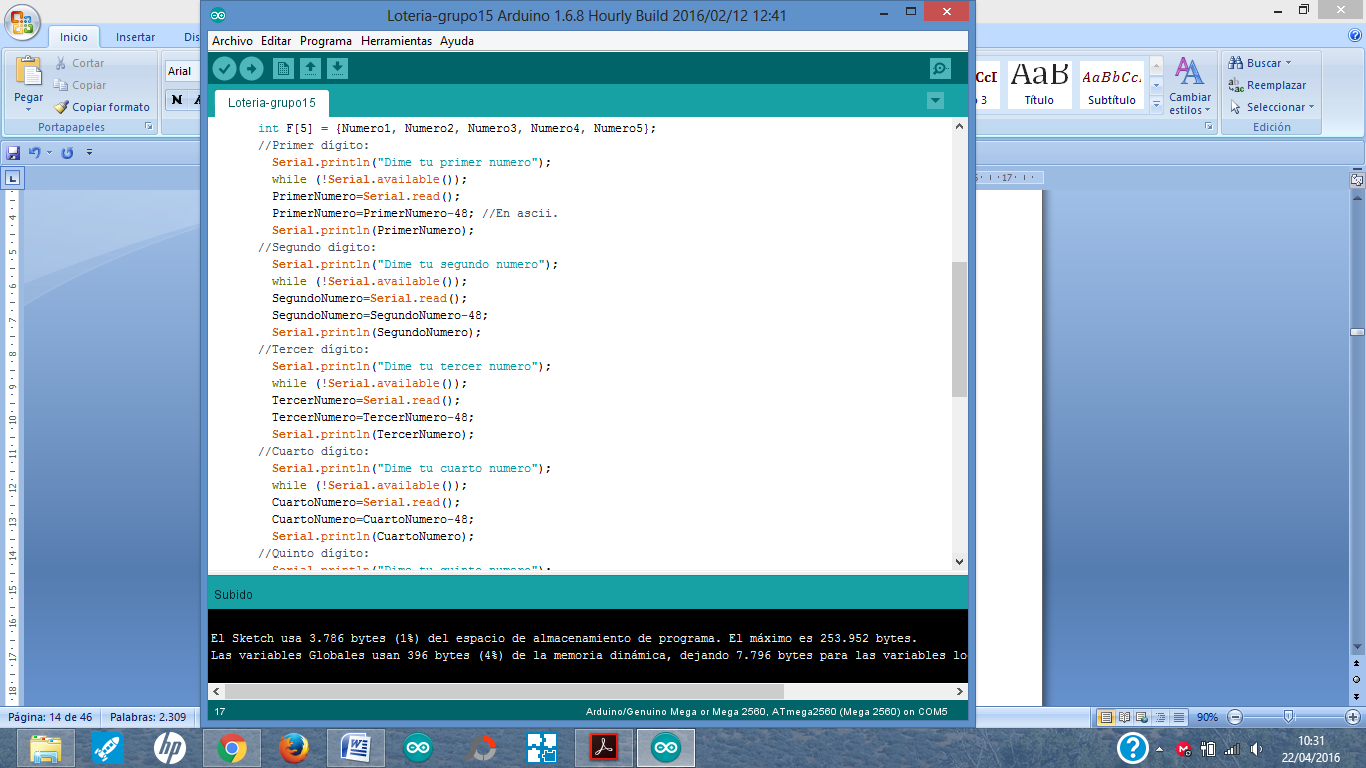
c = add (n1, n2);

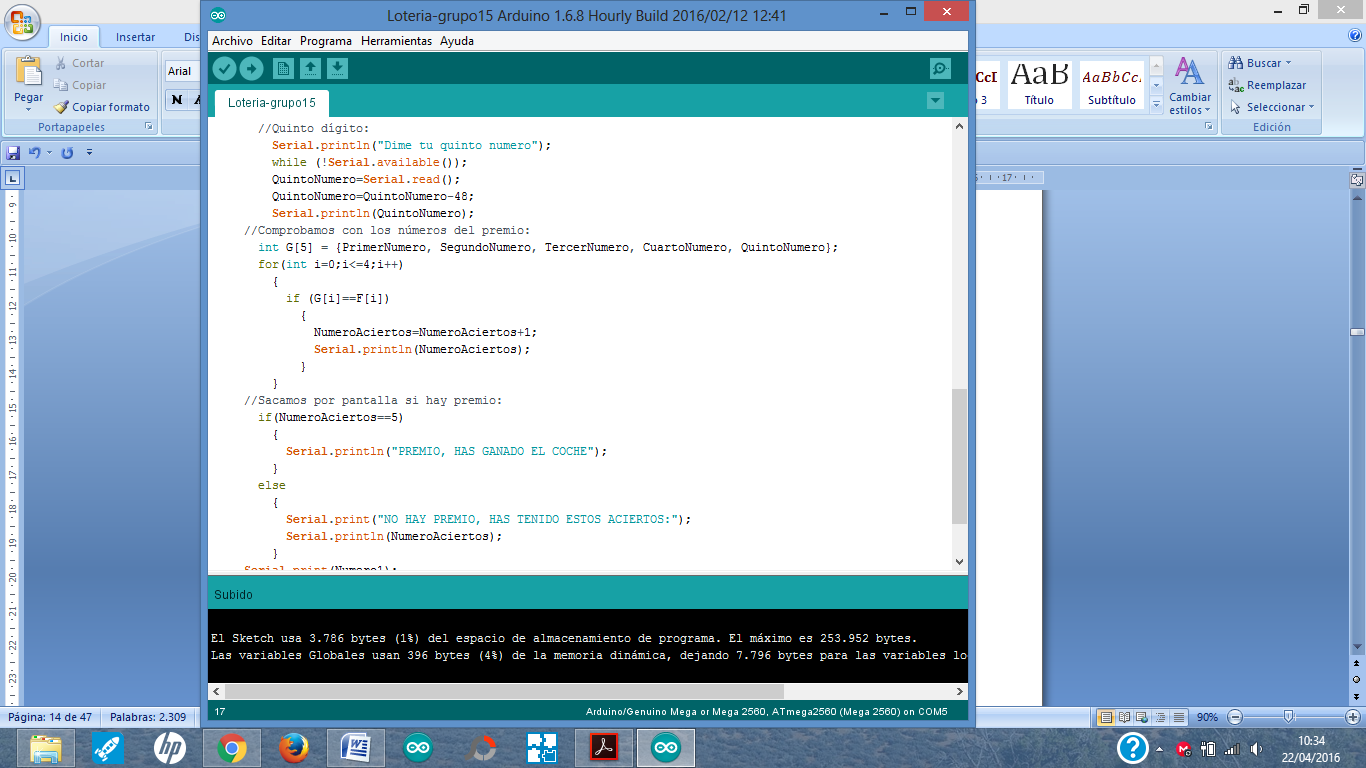
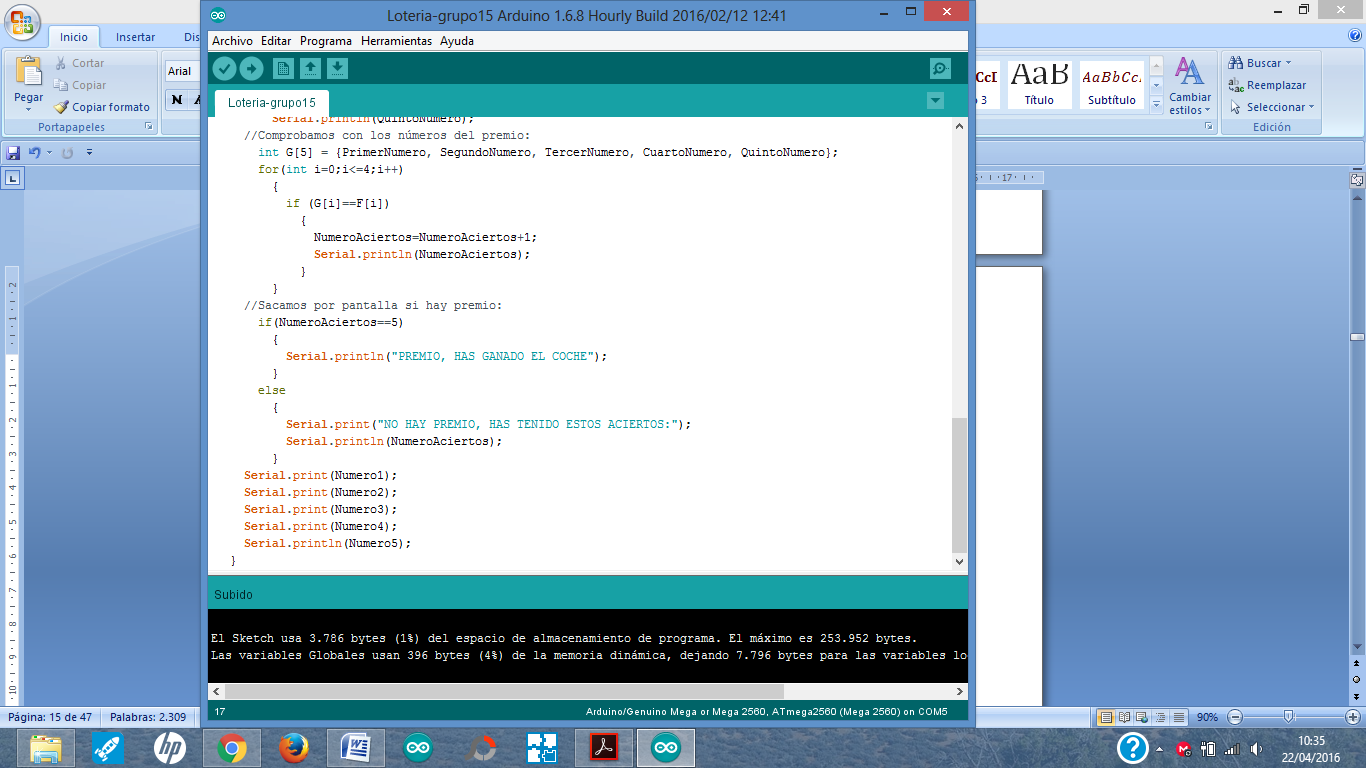
... // From here the value 11 can be used to perform other instructions.

at this moment, when you run your program through the "Serial Monitor" it would appear "The result of the sum is 11");

16. LOTTERY

This program is an example of simulation of a lottery program. In the program, you choose the five numbers of your ballot, then the same program with the random instruction chooses five times a random number until it has 5 random numbers, and the numbers remain hidden for the moment, they are not displayed. The next thing he does is ask the user to enter five numbers on the keyboard of the "Serial Monitor" to see if he guesses the 5 numbers randomly generated by the program. If you hit them you have a prize, and if not, then you see a message on the screen that you do not have a prize and you can not take a car ;-).





**VÍDEO DEL PROGRAMA**

<https://www.youtube.com/watch?v=pR5V5UoFG7Q>