



UNIT 1: FIRST PROGRAMS

AIMS

To create some initial programs that enable you to work quickly and easily with digital input and output (I/O). Now's the time to give it a go!

We'll be having a look at programs written in the Arduino programming language so you can see what they look like and then write one yourself. It's a sophisticated language with its own rules, statements and syntax. First we'll have a look at the basic statements you need to use digital input and output;

THEORY SECTION

- WHAT A PROGRAM LOOKS LIKE
 - Comments section
 - Variable and function section
 - Configuration task section
 - Program main body section
 - Messages section
- PROGRAM STATEMENTS
 - set up() function
 - loop() function
 - pinMode() function
 - digitalRead() function
 - digitalWrite() function
- LOGIC OPERATORS
 - NOT Operator
 - AND Operator
 - OR Operator
 - Combining operators

PRACTICE SECTION

- EXAMPLE 1: Illuminating 1 volt LEDs
- EXAMPLE 2: Illumination of 2 volt LEDs



- EXAMPLE 3: Illumination of 3 volt LEDs
- EXAMPLE 4: Reverse illumination of a LED
- EXAMPLE 5: Two button control
- EXAMPLE 6: Individual work

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	4
1. WHAT A PROGRAM LOOKS LIKE	4
A. COMMENTARY SECTION	4
B. VARIABLE AND FUNCTION DECLARATION SECTION	5
C. CONFIGURATION TASK SECTION	6
D. SECTION FOR THE MAIN BODY OF THE PROGRAM	6
E. MESSAGES SECTION	7
2. PROGRAM STATEMENTS	7
A. THE SETUP() FUNCTION	7
B. THE PINMODE() FUNCTION	8
C. THE DIGITALREAD() FUNCTION	9
D. THE DIGITALWRITE() FUNCTION	9
3. LOGICAL OPERATORS	10
A. THE NOT OPERATOR	10
B. THE AND OPERATOR	10
C. THE OR OPERATOR	11
D. COMBINING OPERATORS	11
PRACTICE SECTION	12
4. EXAMPLE 1: ILLUMINATING A 1 VOLT LED	12
5. EXAMPLE 2: ILLUMINATING A 2 VOLT LED	13
6. EXAMPLE 3: ILLUMINATING A 3 VOLT LED	14
7. EXAMPLE 4: REVERSE ILLUMINATION OF A LED	15
8. EXAMPLE 5: TWO BUTTON CONTROL	15
9. EXAMPLE 6: INDIVIDUAL WORK	16
REFERENCES	17

THEORY SECTION

1. WHAT A PROGRAM LOOKS LIKE

```
EXAMPLE_4_1 Arduino 1.8.2
Archivo Editor Programa Herramientas Ayuda

EXAMPLE_4_1.g
/*
 * OPENIN - Open Source Applications in Industrial Automation
 * 2016-2019
 *
 * EXAMPLE_4_1: - Illuminating a 1 volt LED
 */

//Declaration of variables
int Valor; //Variable value
int Pulsador = 4; //INPUT pin
int Led_Blanco = 6; //OUTPUT pin

// Initial Configuration Sentences
void setup()
{
  pinMode(Pulsador, INPUT); //The button is configured as an INPUT
  pinMode(Led_Blanco, OUTPUT); //The led is configured as an OUTPUT
}

void loop()
{
  Valor=digitalRead(Pulsador); //It reads button state
  digitalWrite(Led_Blanco,Valor); //It shows in the led
}

Guardado
11 Arduino/Genuino Uno en COM1
```

A. COMMENTARY SECTION

All programs should begin by supplying certain information. In this case, for example, this would be the name of the course, the date, the author, the company etc (Figure 1). It would also be interesting to read an explanation of what the program consists of or what it does.

This information is called the “Header Comments”. You can include all the comments you want wherever you want as long as they appear between the symbols “/*” and “*/”. Have a close look at the example above.

You can also include simpler one line comments. In this case just put the symbol “//” before them; these comments are very common.

```
EXAMPLE_4_1
/*      OPENIN - Open Source Applications in Industrial Automation
          2016-2019

      EXAMPLE_4_1:  : Illuminating a 1 volt LED
*/
```

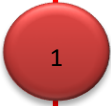


Figure 1

Try not to be lazy. Get used to writing comments about your programs. You might need them yourself in the future to remind you of what you did, how you did it or why you did it.

B. VARIABLE AND FUNCTION DECLARATION SECTION

I recommend you use this second section to declare variables and functions(Figure 2). You don't know what they are yet but, don't worry, you'll learn about them a bit later on. Think of a variable as a kind of box or receptacle you assign a name and possibly a value to. You save it on the controller RAM memory and you can use it later on as many times as you like.

A function, on the other hand, comprises a number of other statements, instructions or functions. You give this group a name too and you can also use it as many times as you need to.

```
//Declaration of variables
int Valor;           //Variable value
int Pulsador = 4;   //INPUT pin
int Led_Blanco = 6; //OUTPUT pin
```



Figure 2

Three variables have been created in the example in the above figure: “Value”, “Button” and “White_LED”. The digital input value will be saved in the first one. The second variable, “Button”, is given a value of 4 which corresponds to the number of the input pin the button is attached to; we have to read the state of the button. We've given a value of 6 to the “White_LED” variable. This corresponds to the output pin the LED has been attached to. The white LED is the one on the protoboard.

Generally speaking, both the variables and the functions must be declared BEFORE being used in the program.

C. CONFIGURATION TASK SECTION

Normally programs written in Arduino language begin by executing a few configuration instructions or functions. These determine which pins are to be used as inputs and which as outputs.

```
// Initial Configuration Sentences
void setup()
{
  pinMode(Pulsador, INPUT);    //The button is configured as an INPUT
  pinMode(Led_Blanco, OUTPUT); //The led is configured as an OUTPUT
}
```

3

Figure 3

The “Button” variable pin in figure 2, which was previously given a value of 4, is configured as an INPUT. The “White_LED” variable pin, or pin 6, is configured as an OUTPUT.

Generally speaking, the configuration statements or functions are only executed once when you RESET the system or connect it to a power source.

D. SECTION FOR THE MAIN BODY OF THE PROGRAM

You have to write all the instructions, statements and functions that make up your program in this section. There are only two functions in the example we’re dealing with at the moment but bear in mind that a program may contain hundreds or even thousands.

```
void loop()
{
  Valor=digitalRead(Pulsador); //It reads button state
  digitalWrite(Led_Blanco,Valor); //It shows in the led
}
```

4

Figure 4

The first function, “Valor=digitalRead(Pulsador);”, reads the digital state of the button connected to pin 4, the one that was previously configured as an input. The button’s state is saved in the “Value” variable in the RAM memory.

The second function, “digitalWrite(Led_Blanco,Valor);”, writes the contents of the “Valor” variable on pin 6; this was read in the previous function. Remember that this pin, number 6, was previously configured as an output.

The controller executes all the functions that make up the main body of the program as fast as possible. It executes them constantly and indefinitely from first to last.

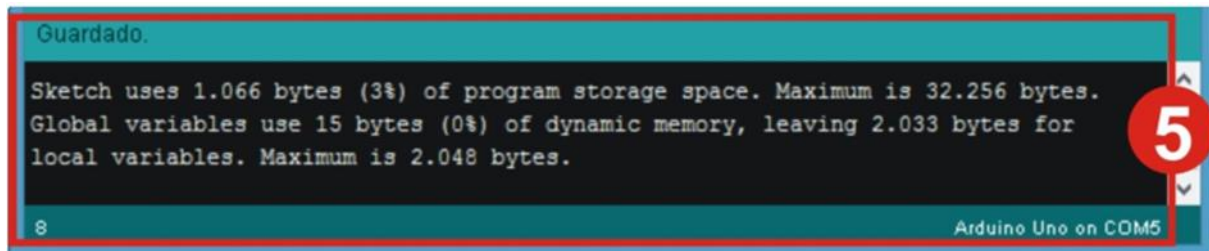
Important: When declaring the variables and configuration and main program functions, make sure they ALWAYS end with this symbol: “;”.



E. MESSAGES SECTION

In this final section, the Arduino IDE work environment will display a series of messages:

it'll tell you whether you're saving, compiling or recording a program on the controller's memory. It'll also let you know if there have been compilation errors, the type of error and where they have occurred.



```
Guardado.
Sketch uses 1.066 bytes (3%) of program storage space. Maximum is 32.256 bytes.
Global variables use 15 bytes (0%) of dynamic memory, leaving 2.033 bytes for
local variables. Maximum is 2.048 bytes.
```

Figure 5

This exercise has worked out perfectly. We're told that the program takes up a total of 1,066 of the available 32,256 bytes of FLASH memory. 15 of the 2,048 available bytes of RAM memory have been used for data (because of the variables that we created). So, there are 2,033 bytes left.

Remember: “Compile” a program means to translate the one you've written using complex Arduino language into binary code (also known as machine language); this is what is actually recorded on the controller's FLASH memory. As far as you're concerned, it's a simple, automatic process.

2. PROGRAM STATEMENTS

Now you've seen what a program looks like – in theory anyway. You've seen that it's made up of a number of instructions or statements also called “FUNCTIONS”; a program may contain thousands of these.

Each programming language has its own functions, its own syntax and its own programming rules. We're going to have a look at the Arduino programming language; we've already used five different instructions or functions. Now let's have a closer look at them.

A. THE SETUP() FUNCTION

This function and all the other functions contained in it are executed when the system is reset. This happens every time you push the RESET button or power up the system.

There are usually a number of other functions within this one: input and output pin configuration, certain variable settings, libraries etc. In general terms, you can put whatever statements and functions you like in the setup() function as long as they are enclosed by curly braces: “{...}”. You always have to create a setup() function even if there's nothing in it: just leave the braces empty.

The important thing to remember is that all the functions contained in the setup() function are only executed once: when you reset.



Syntax:

```
Void setup()  
  
{  
  
.....}
```

Example

```
Void setup()  
  
{  
  
pinMode(INPUT, Button);           //The button pin is configured as an input  
pinMode(White_LED,OUTPUT);       //The button pin is configured as an output  
  
}
```

A program's main loop() function may comprise tens, hundreds or thousands of other functions. The above example comprises only two. The loop() function may also be empty without any functions enclosed in curly braces:{}. But that doesn't make any difference: you always have to have one.

All the other functions enclosed in the loop() function curly braces are executed constantly from first to last one after the other until you disconnect the system.

B. THE PINMODE() FUNCTION

Configure one of the Arduino controller's pins as an input or an output. It usually comes at the beginning of a program and is included in the **setup()** function.

Syntax:

```
pinMode;
```

pin: This is the number of the pin we're going to configure; it may be between 0 and 13 in Arduino UNO.

Mode: This establishes whether it functions as an INPUT or OUTPUT

Examples

```
int Button=4;
```

```
int White_LED=6;
```

```
pinMode(White_LED, OUTPUT);           //Pin 6 is an output
```

```
pinMode(9 OUTPUT);                   //Pin 9 is an output
```

```
pinMode(Button, INPUT);              //Pin 4 is an input
```

```
pinMode(12, INPUT);                  //Pin 12 is an input
```




All the digital pins are automatically configured as inputs when you reset the system.

C. THE DIGITALREAD() FUNCTION

This function reads and returns the binary logic state ("1" or "0", "HIGH" or "LOW") of any of the Arduino controller pins.

Syntax:

digitalRead(pin);

pin: This displays the number of the pin we're going to read; it may be between 0 and 13 in Arduino UNO.

Examples

int Button=4;

Value=DigitalRead(Button); *//Reads the state of pin 4 and saves it in "Value"*

X=digitalRead(8); *//Reads the state of pin 8 and saves it in "X"*

D. THE DIGITALWRITE() FUNCTION

Writes or sets a binary value ("1" or "0", "HIGH" or "LOW") through an output pin.

Syntax:

digitalWrite(pin, value);

pin: This gives us the number of the pin we're going to set the value by; it may be between 0 and 13 in Arduino UNO.

Value: Indicates the value to be set ("1" or "0", "HIGH" or "LOW").

Examples

int White_LED=6;

DigitalWrite(White_LED,Value); *//Sets the contents of the "Value" ("1" or "0")through pin 6*

DigitalWrite(11, LOW); *//Sets level "0" through output pin 11*

DigitalWrite(7,1); *//Sets level "1" through output pin 7*



3. LOGICAL OPERATORS

We can't really go any further without talking about three kinds of operators that are capable of relating different sorts of functions, operations, expressions etc. to each other. They're known as "logical operators" and they're used a lot in digital systems. I'll try and give you a simple explanation of them without using too many technical terms.

A. THE NOT OPERATOR

This operator expresses a negation (NOT) and is represented by an exclamation mark (!). It's easier to understand with an example. Have a look at this function which I'm sure you'll recognize:

Value=digitalRead(Button);

The "Value" variable is equivalent to level "1" when the "Button" is also on "1". If this is not the case the "Value" is equivalent to level "0". Now have a look at the following function:

Value = ! digitalRead(Button);

The "Value" variable is equivalent to level "1" when the "Button" is read and it is NOT on level "1", or, in other words, it's on level "0". Otherwise, if the button is on level "1", the "Value" variable will be level "0". Have a look at the example below and reply:

Value = ! digitalRead(12);

"Value" equals level "1" or "true" if: _____

"Value" equals level "0" or "false" if: _____

B. THE AND OPERATOR

This operation generates a level "1", also known as "true", when ALL the elements that you relate to each other are also at level "1" or are true. It's represented by these symbols: "&&". Have a look at the example:

Value =digitalRead(4) && digitalRead(12);

The "Value" variable is equivalent to level "1", "true", when inputs 4 AND 12 are read and both are also at level "1". If any one of the inputs is at level "0" the result for the "Value" variable is also level "0" or "false. Have a look at the following example below and reply:

Value =digitalRead(4) && digitalRead(8) && digitalRead(12);

"Value" equals level "1" or "true" if: _____

"Value" equals level "0" or "false" if: _____



C. THE OR OPERATOR

This operation generates a level “1”, also known as “true”, when ANY of the elements you relate to each other is also at level “1” or is true. It’s represented by these symbols: “||”. Have a look at the example:

```
Value =digitalRead(4) || digitalRead(12);
```

The “Value” variable is equivalent to level “1”, “true”, when inputs 4 OR 12 OR both are at level “1”. If all of the inputs are at level “0” the result for the “Value” variable is also level “0” or “false. Have a look at the following example below and reply:

```
Value =digitalRead(4) || digitalRead(8) || digitalRead(12);
```

“Value” equals level “1” or “true” if: _____

“Value” equals level “0” or “false” if: _____

D. COMBINING OPERATORS

You can of course combine various logic operators in the same function. You have to use parentheses to establish the order they have to be assessed and calculated. Have a look at the example below:

```
Value = ! digitalRead(4) && digitalRead(7);
```

The “Value” variable is equivalent to level “1” or “true” when input 4 is **NOT** on level “1” **BUT** input 7 is. . Have a look at the following example below and reply:

```
Value = (digitalRead(8) && ! digitalRead(12)) || digitalRead(4);
```

“Value” equals level “1” or “true” if: _____

“Value” equals level “0” or “false” if: _____

PRACTICE SECTION

4. EXAMPLE 1: Illuminating a 1 volt led

This is the same example you studied in the Theory Area so I'm not going to bore you by telling how it works again. I just wanted to remind you that what it does is read the state of the button connected to pin 4 and register this in the white LED connected to pin 6 on the controller.

- **Compilation and Recording**

As you can see from the Figure 6, all you have to do is press the button marked 1. The Arduino IDE environment compiles your program for you by translating it into machine or binary code; the whole process is quite fast and entirely automatic. It's not actually important to know how it happens.

If the program is properly written and there are no mistakes, you'll see some messages like the ones in window 2. You've seen know them before: they tell you how much space the program will take up in the controller's FLASH memory as well as how much RAM memory is consumed by the variables you've used. As well as that, if the program has been properly written, it'll be recorded immediately in the controller's FLASH memory.

If there are any mistakes they'll appear in the same window; you'll see the kinds of mistakes you've made and where they are in the program. Just correct them and then keep compiling the program.



Figure 6



- **Checking**

Even if the Arduino IDE environment doesn't inform you of any errors when you've finished compilation it doesn't necessarily mean the program's going to work properly. The environment only detects syntactical errors: a function that's not written properly or that doesn't exist, variables that haven't been previously declared, data that's not right etc.

The acid test comes when you check that the controller does exactly what you programmed it to do. This example is very simple, all you have to do is push the D4 button: if the white LED attached to pin D6 lights up everything is okay; if it isn't, it'll go out.

One last check: when you record the FLASH memory with a program, it stays there indefinitely even if you turn the power off. The programs only get deleted when you record another one. And here's another detail: once a program is recorded, the Arduino board no longer relies on the PC in any way whatsoever. And there's a very easy way to check all this: take out the USB cable from the Arduino UNO board and plug in an ordinary power lead instead or connect up a battery like in the figure. You'll soon see if everything works just like it did before.

Maybe you think that all this doesn't amount to much and that you don't need a controller to illuminate a LED with a button. You're wrong. You've taken the first step: if you've been able to understand the concept of a program and its functions, I can assure you that you'll be able to do much more complicated and interesting things. You just have to take things little by little.

5. EXAMPLE 2: Illuminating a 2 volt led

Here's another version of the previous program. It controls white LED connected to pin 6 with the button connected to pin 4 on the same board. Have a look at the figure and compare it to the previous example. The sole difference is that we haven't used variables to define which pins the button and the white lead are connected to.

The pin reference numbers 4 and 6 appear in the functions themselves: `pinMode()`, `digitalRead()`, `digitalWrite()`. In this case, the pin numbers are said to be expressed as "constants" not "variables".

Just as you did with the previous example, compile and record the program; after that make sure it works properly.

- **Conclusions**

Using variables or constants to define a pin is basically up to you. For instance, if the button is going to be permanently connected to pin 4, use this number as a constant just as we did in this example; you'll save on RAM memory.

On the other hand, if there's a chance that this button, or any other peripheral, may be subject to changes in controller pin connections, it's better to define this connection as a variable: as "`intButton=4;`" for example. This way, if the button gets connected to pin 12 in the future, the only thing you'll have to change will be the declaration of the variable: "`intButton=12;`". You won't have to modify the program functions that use this button.



6. EXAMPLE 3: Illuminating a 3 volt led

Here's yet another and final version of the program for controlling a LED with a button.

The first thing to do is compile the program and record it on the controller; make sure it works properly.

- **Conclusions**

If you study the program carefully you'll see what the differences are between this example and EXAMPLE 1 and EXAMPLE 2. We can draw the following conclusions:

- ✓ No variable has been used; this means a saving on RAM memory. In a lot of cases using variables and constants depends more on what exactly you're interested in doing.
- ✓ The pins to be used as digital inputs don't need to be configured using `pinMode()`. All the pins are automatically configured as inputs.
- ✓ The result of a function can be used as the input for another function without going through the intermediate variable "Value" as in the other examples. Have a look at the following expression:

`digitalWrite(6,digitalRead(4));`

There's one function inside another. The most internal function is the one that's executed first: `digitalRead(4)`. The result is passed on to the following function `digitalWrite(...)`;

- ✓ A single program can be written in a number of different ways. Right now just worry about whether or not the program works properly; that's the most important thing. In time you'll be able to reduce the amount of FLASH program memory and RAM data memory used and this will make it more "EFFICIENT".

You can do a very simple check yourself. Compile and record EXAMPLE 1, EXAMPLE 2 and EXAMPLE 3 again. Look at how much FLASH program memory RAM variable memory you use. You get this information from the Arduino IDE environment every time you finish recording a program; it appears at the bottom. Complete the following table:

EXAMPLE	FLASH Program Memory	RAM Data Memory
EXAMPLE 1		
EXAMPLE 2		
EXAMPLE 3		

7. EXAMPLE 4: Reverse illumination of a led

The idea is to use the NOT logic operator and make a program to illuminate the white LED connected to pin 6 whenever the button connected to pin 4 is NOT on.

The main body of the program is contained in the loop() function and has been highlighted in yellow. It's very like the previous example. The only difference is that the NO function, represented by the symbol "!", inverts the reading of the button connected to pin 4.

This example (Figure 7) could simulate a machine with a motor that's always running unless an emergency button is pushed.



```
void setup()
{
  pinMode(6, OUTPUT); //la patilla 6 del led blanco se configura
}

void loop()
{
  digitalWrite(6, !digitalRead(4)); //con el estado invertido del pin
}

Sketch uses 1,044 bytes (3%) of program storage space. Maximum is 32,256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2,039 bytes for local variables. Maximum is 2,048 bytes.
```

Figure 7

8. EXAMPLE 5: Two button control

Imagine you want to control an alarm with two buttons that are a long way apart. The alarm should go off when you push one OR other of the buttons (OR both). The alarm is simulated by the red LED connected to pin 11. The two buttons are connected to pins 4 and 12.

Here's the final program; have a close look at Figure 8.



```
void setup()
{
  pinMode(11, OUTPUT); //la patilla 11 del led rojo se configura
}

void loop()
{
  digitalWrite(11, digitalRead(4) || digitalRead(12)); //Pulsar 0 o 1
}

Sketch uses 1,034 bytes (3%) of program storage space. Maximum is 32,256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2,039 bytes for local variables. Maximum is 2,048 bytes.
```

Figure 8



9. EXAMPLE 6: Individual work

Use EXAMPLE 6 as a program model or “template”. Make the necessary modifications to the model to do the exercises I’ll be setting you. I hope you use everything we’ve looked at up to date, learn something and, above all, have a good time. As we learn more and more about programming, you’ll realize that the controller is capable of so much more than just illuminating a simple LED...and this is just the start!

Exercise	Description
1	The amber LED should go on when buttons 7 and 12 are pushed.
2	The green LED goes on when buttons 12 and 8 are pushed but not button 4.
3	The white LED goes on when button 4 is pushed but not button 7. The red LED goes on when buttons 8 or 12 are pushed.
4	The green LED goes on when button 4 is pushed but not button 7, and when button 8 is pushed but not button 12.
5	The red LED goes on when buttons 4, 7 and 8 are pushed or, on the other hand, button 12 is.

Remember the following: you must write down every modification you make in the Arduino IDE environment, record it on the controller and make sure it works according to the specifications for each exercise.

Complete the following table.

Exercise	Function	Your program's instructions
1	setup()	
	loop()	
2	setup()	
	loop()	
3	setup()	
	loop()	
4	setup()	
	loop()	
5	setup()	
	loop()	



REFERENCES

BOOKS

- [1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
- [2]. **Practical ARDUINO**, Jonathan Oxe & 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>