UNIVERSIDAD YACHAY TECH

Wireless Network

LAB3: RFID Technology Using an Arduino

Objective

1. Describe the basic principles of radio communications
2. Describe how to set up an Arduino and write your first Arduino code.
3. Understand the technology used in RFID cards and readers
4. Understand the basic hardware and software features of MIFARE Classic RFID cards
5. Learn to use the open source MF522.h library.

Discussion of fundamentals

Arduino an open source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. The open source Arduino software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open source software. With the growing interest in home-made, weekend projects among students and hobbyists alike, Arduino offers an innovative and feasible platform to create projects that promote creativity and technological tinkering.

An Accessible Hardware Platform

An Arduino UNO is shown in Figure 1, where the main component is the ATmega328p microcontroller from Atmel (now MicrochipTM), which is the large chip with 28 legs in the image. It is a controller with 8-bit-wide registers, and operates at a clock frequency of 16 MHz. It has 32 kB RAM memory and 1 kB non-volatile EEPROM memory, which can be used to store persistent data that need to survive tuning off and on the supply voltage. There are three timers on board, which are basically counters that count clock cycles and are programmable to perform some action, once a counter reaches some value.

The UNO interacts with its environment through 13 digital input-output (IO) pins, of which most can be configured to be either input or output, and have software-configurable pull-up resistors. Several of the pins are configurable to support I2C, SPI, and RS-232 communication. Moreover, there are six analog input pins. They measure voltages of up to the supply voltage of 5 V. An alternative internal reference voltage source provides a 1.1V reference. All digital and analog pins are routed to pin headers that are visible on both sides of the Arduino printed circuit board (PCB). Furthermore, the built-in hardware

RS-232 port is connected to an RS-232-to-USB converter that allows communication and programming from a host computer. There is no WiFi, Bluetooth, or Ethernet support on the UNO board, but extension boards, so-called shields, are available for mounting directly onto the pin headers

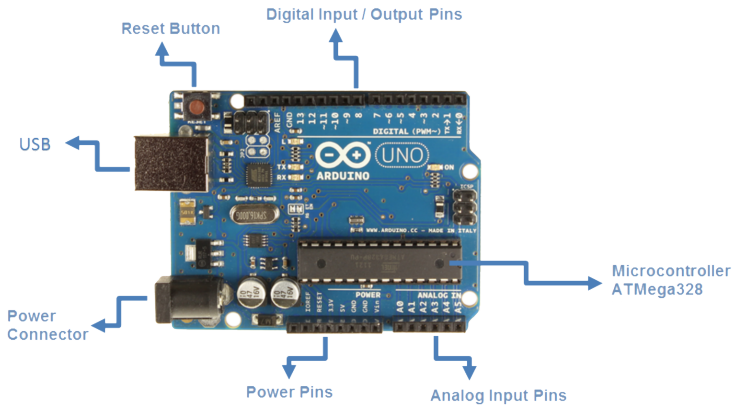


Figure 1: Some parts of Arduino

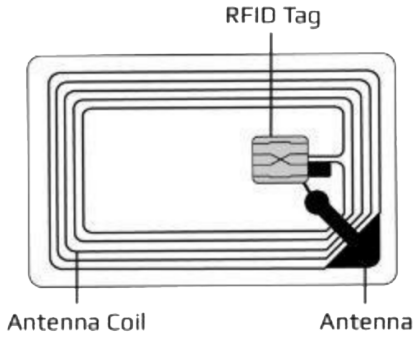
The programming language used for the Arduino is essentially a version of C/C++. The programming environment is the Arduino IDE (integrated development environment). The team that developed it bundled it with many prewritten functions and libraries to simplify the process of writing code to interface with hardware. For example, these libraries take the multiple lines of code required to turn on an LED and simplify them into a single instruction!

Shields

Shields are boards put on top of the Arduino board. They enhance the platforms basic functionality by bringing in extra peripherals and sensors. Many shields come with libraries specifically made for them.

NOTE: Arduino's name honors a king named Arduin who lived in Ivrea at the beginning of the eleventh century. Therefore, the term 'shield' was chosen to 'protect' the king.

Many different shields are available. Arduino is an open platform, which allows makers and manufacturers to produce their own add-ons to the platform at no fee. You can find specific shields for almost any application out there: controlling motors, reading Real Time Clock (RTC) chips, storing data in SD cards, wireless modules (Wifi, Bluetooth) and so on.



RFID Technology

Radio frequency identification is a wireless communication technology that lets computers read the identity of inexpensive electronic tags from a distance, without requiring a battery in the tags. The electromagnetic field that surrounds an RFID antenna can be broken up into two segments near-field and far-field. Typically, near-field is defined as the field around the antenna up to one wavelength (λ) away (approximately up to 35 centimeters). The two segments of the RF field, near-field and far-field, have different energies so they typically require a corresponding antenna type to get the best read range. (The near-field is primarily magnetic in nature, while the far-field has both electric and magnetic components.)

The communications between an RFID tag and an RFID reader (via the antenna) occurs using a process known as electromagnetic coupling. There are two types of coupling inductive and capacitive.

- INDUCTIVE: A near-field antenna uses inductive coupling which means that it uses a magnetic field to energize the RFID tag. A magnetic field is created in the near-field region that allows the RFID readers antenna to energize the tag. The tag then responds by creating a disturbance in the magnetic field that the reader picks up and decodes.
- CAPACITIVE: A far-field antenna uses capacitive coupling (or propagation coupling) to energize the RFID tag. Capacitive coupling occurs when the RFID readers antenna propagates RF energy outward and that energy is used to energize the tag. The tag then sends back a portion of that RF energy to the readers antenna as a response which is known as backscatter.

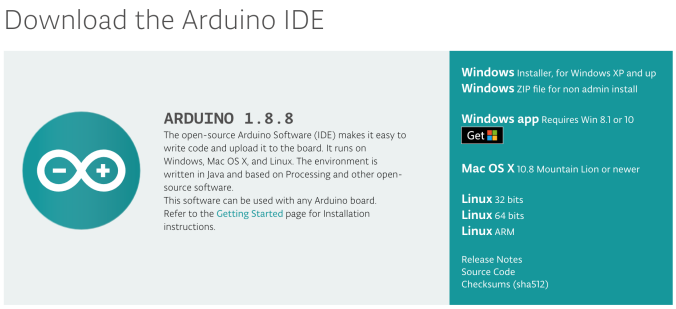
Excersice 1: Getting Started with Arduino

Materials needed

1. 1x Arduino-compatible board such as an Arduino UNO
2. 1x USB cable A to B
3. 2x LEDs
4. 2x 330 resistors
5. A mini breadboard
6. 5x male-to-male jumper wires

Downloading and installing the software

Arduino is open source-oriented. This means all the software is free to use non-commercially. Go to `http://arduino.cc/en/Main/Software` and download the latest version for your specific operating system. If you are using a Mac, make sure you choose the right Java version; similarly on Linux, download the 32-or 64-bit version according to your computer.



Windows

Once you have downloaded the setup file, run it. If it asks for administrator privileges, allow it. Install it in its default location (`C:\Program Files \Arduino` or `C: \Program Files (x86) \Arduino`). Create a new folder in this location and rename it `My Codes` or something where you can conveniently store all your programs. The computer will begin to install the drivers for the Arduino by itself. If it does not succeed, go to the `Device Manager` and check if it has been installed under COM ports.

Linux (Ubuntu 12.04 and above)

Once you have downloaded the latest version of Arduino from the preceding link, install the compiler and the library packages using the following command: `sudo apt-get update && sudo apt-get install arduino arduino-core`. If you are using a different version of Linux, this official Arduino walkthrough at `http://playground.arduino.cc/Learning/Linux` will help you out.

Connecting the Arduino

It is time to hook up the Arduino board. Plug in the respective USB terminals to the USB cable and the tiny LEDs on the Arduino should begin to flash.

Figure 2: Arduino UNO plugged in

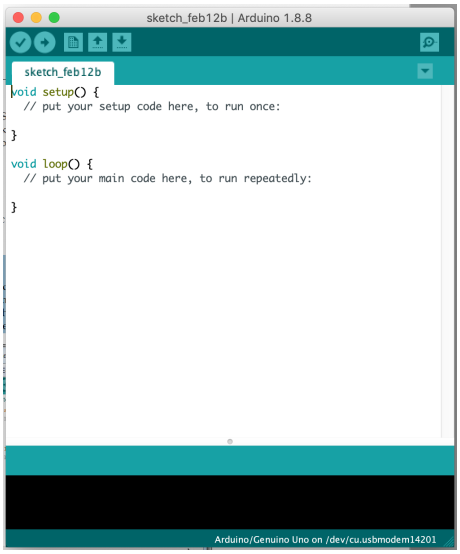


Figure 3: Arduino IDE

If the LEDs didn't turn on, ensure that the USB port on your computer is functioning and make sure the cable isn't faulty. If it still does not light up, there is something wrong with your board and you should get it checked.

The Arduino IDE

The Arduino software, commonly referred to as the Arduino IDE (Integrated Development Environment), is something that you will become really familiar with as you progress through this lab. The IDE for Windows, Mac OS, and Linux is almost identical. Now let's look at some of the highlights of this software. Figure 3 shows the window that you will see when you first start up the IDE. The tick/ check mark verifies that your code's syntax is correct. The arrow pointing right is the button that uploads the code to the board and checks if the code has been changed since the last upload or verification. The magnifying glass is the Serial Monitor. This is used to input text or output debugging statements or sensor values.

All Arduino programmers start by using one of examples presented in `File — Examples`. Even after mastering Arduino, you will still return here to find examples to use. Figure 4 shows the tools that are available in the Arduino IDE. The Board option opens up all the different boards that the software supports.

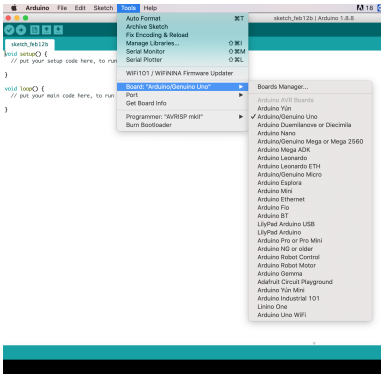


Figure 4: Arduino tools

Writing a simple print statement

The Arduino uses a Serial Monitor for displaying information such as print statements, sensor data, and the like. This is a very powerful tool for debugging long codes.

Hello World

Open up the Arduino IDE and write the following code into a new sketch:

Code

```
void setup() {
Serial.begin(9600);
Serial.println("Hello World!");
}
void loop() {
}
```

Open **Tools** — **Board** and choose **Arduino UNO**, as shown in Figure 5

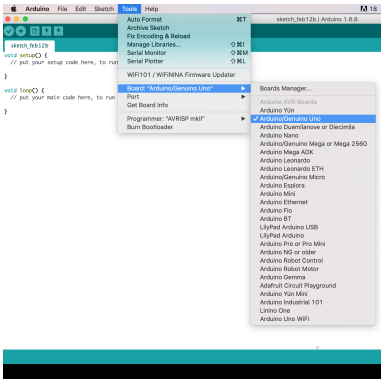


Figure 5: Examples of Arduino

Open **Tools** — **Port** and choose the appropriate port (remember the previous COM xx number? select that), as shown in Figure 6. For Mac and Linux users, once you have connected the Arduino board, going to **Tools** — **Serial Port** will give you a list of ports. The Arduino is typically something like `/dev/tty.usbmodem12345` where 12345 will be different.

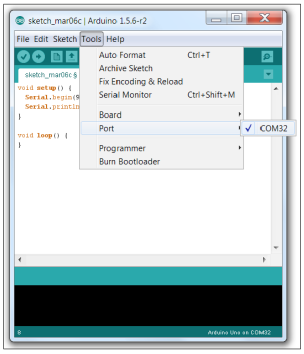


Figure 6: Selecting the Port

Finally, hit the Upload button. If everything is fine, the LEDs on the Arduino should start flickering as the code is uploaded to the Arduino. The code will then have uploaded to the Arduino. To see what you have accomplished, click on the **Serial Monitor** button on the right side and switch the baud rate on the Serial Monitor window to 9600. You should see your message `Hello World!` waiting for you there.

Excersice 2: Using serial communication

Serial communication is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port which is also known as a UART. Serial data transfer is when we transfer data one bit at a time, one right after the other. Information is passed back and forth between the computer and Arduino by, essentially, setting a pin to high or low. Just like we used that technique to turn an LED on and off, we can also send data. One side sets the pin and the other reads it. In this section of the lab, you will see two examples. In the first example, Arduino will send data to the computer using serial communication, while in the second example, by sending a command (serial) from the computer, you can control the functionality of the Arduino board.

Serial write

In this example, the Arduino board will communicate with the computer using the serial port, which can be viewed on your machine using the Serial Monitor. Write the following code to your Arduino editor:

Code

```
void setup() // run once, when the sketch starts
{
Serial.begin(9600); // set up Serial library at 9600 bps
Serial.println("Hello world!"); // prints hello with ending line
break
}
void loop() // run over and over again
{
// do nothing!
}
```

Even if you have nothing in the setup or loop procedures, Arduino requires `void setup()` and `void loop()` to be there. That way it knows you really mean to do nothing, as opposed to forgetting to include them!

`Serial.begin` sets up Arduino with the transfer rate we want, in this case 9600 bits per second. `Serial.println` sends data from Arduino to the computer. Once you compile and upload it to your connected Arduino board, open **Serial Monitor** from the Arduino IDE. You should be able to see the **Hello world!** text being sent from the Arduino board:

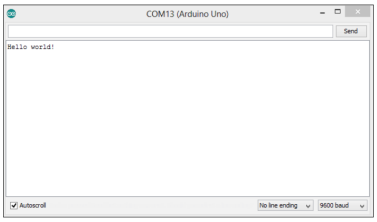


Figure 7: Hello World

Serial read

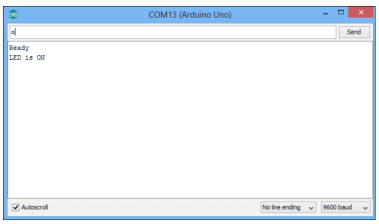
In the previous example, serial library was used to send a command from Arduino to your computer. In this example, you will send a command from the computer, and Arduino will do a certain operation (turn on/off LED) based on the command received:

Code

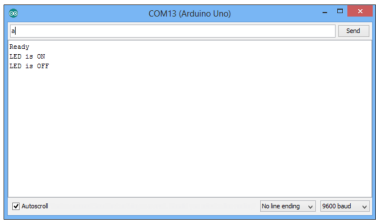
```
int inByte; // Stores incoming command
void setup() {
Serial.begin(9600);
pinMode(13, OUTPUT); // LED pin
Serial.println("Ready"); // Ready to receive commands
}

void loop() {
   if(Serial.available() > 0) { // A byte is ready to receive
      inByte = Serial.read();
      if(inByte == 'o') { // byte is 'o'
        digitalWrite(13, HIGH);
        Serial.println("LED is ON");
      }
      else
      {
       // byte isn't 'o'
       digitalWrite(13, LOW);
       Serial.println("LED is OFF");
      }
   }
}
```

The `inByte` function will store the incoming serial byte. From the previous example, you should be familiar with the commands written in the `setup` function. In the loop function, first you need to know when a byte is available to be read. The `Serial.available()` function returns the number of bytes that are available to be read. If it is greater than 0, `Serial.read()` will read the byte and store it in an inByte variable. Let's say you want to turn on the LED when the letter 'o' is available. For that you will be using the if condition, and you will check whether the received byte is 'o' or not. If it is 'o', turn on the LED by setting pin 13 to HIGH. Arduino will also send an **LED** is **ON** message to the computer, which can be viewed in Serial Monitor:



If it is any other character, then turn off the LED by setting pin 13 to LOW. Arduino will also send an **LED** is **OFF** message to the computer, which can be viewed in Serial Monitor:



The world of LED

Remember the LED that we spoke about in the prerequisites? Let's learn a bit about it before plugging it in, as shown in the following image

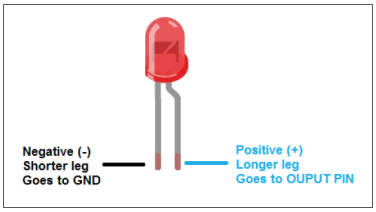


Figure 8: LED basics

The LED, or light-emitting diode, is a quite robust piece of electronics. However, this doesnt mean that you can treat it in any way you want and still hope that it will work as expected. First of all you should take care never to exceed the recommended voltage for said LED. Most LEDs require a resistor to work, but fortunately pin 13 also comes with a built-in resistor because of the attached surface-mounted LED. No other pin on the Arduino has a built-in resistor like pin 13, which is why you should always take care to use resistors whenever theres an LED in your circuit. Secondly, the 5mm LED has two legs, one of which is longer than the other. This longer leg is the positive lead (+) and should be connected to pin 13 in the example; the shorter pin is the negative lead (-) and should be connected to the ground pin (gnd). Another interesting fact about the LED is that if you look closely inside the bulb youll see that the two leads are each connected to a little piece of metal. This metal has a very distinctive look, so you can use it to discern between the two leads.

Plug in the LED such that the longer leg goes into pin 13 and the shorter leg goes into the GND pin, as in the following:

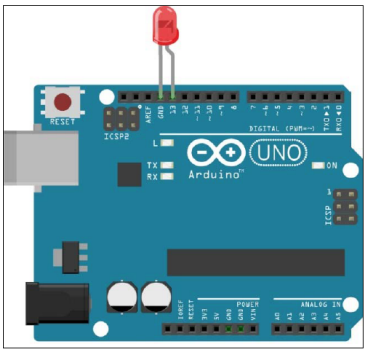


Figure 9: Arduino LED setup (Fritzing)

This diagram is made using software called Fritzing. This software will be used in future projects to make it cleaner to see and easier to understand as compared to a photograph with all the wires running around. Fritzing is open source software which you can learn more about at www.fritzing.org.

Open up a new sketch. Go to **File** — **Examples** — 01. **Basics** — **Blink**. Upload the code. Your LED will start blinking.

Let's see what the code does and what happens when you change it. This is the blink example code that you just used:

Code

```
/*
Blink
Turns on an LED on for one second, then off for one second,
repeatedly.

This example code is in the public domain.
*/


//Pin 13 has an LED connected on most Arduino boards.
//give it a name:
int led = 13;


//the setup routine runs once when you press reset:
void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}


//the loop routine runs over and over again forever:
void loop() {
digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage
level)
delay(1000); // wait for a second
digitalWrite(led, LOW); // turn the LED off by making the voltage
LOW
delay(1000); // wait for a second
}
```

We have three major sections in this code.

Code

```
int led = 13;
```

This line simply stores the numerical PIN value onto a variable called led.

Code

```
void setup() {
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}
```

This is the setup function. Here is where you tell the Arduino what is connected on each used pin. In this case, we tell the Arduino that there is an output device (LED) on pin 13.

Code

```
void loop() {
digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage
level)
delay(1000); // wait for a second
digitalWrite(led, LOW); // turn the LED off by making the voltage
LOW
delay(1000); // wait for a second
}
```

This is the loop function. It tells the Arduino to keep repeating whatever is inside it in a sequence. The `digitalWrite` command is like a switch that can be turned ON (`HIGH`) or OFF (`LOW`). The delay(`1000`) function simply makes the Arduino wait for a second before heading to the next line.

Add another LED, you'd need some additional tools and some changes to the code. This is the setup that you want to create.

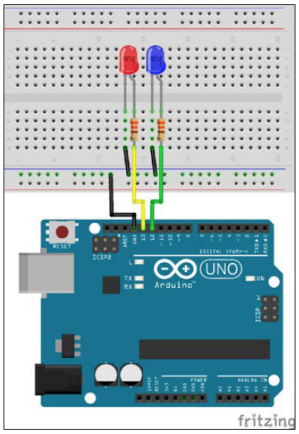


Figure 10: Connecting two LEDs to an Arduino

If this is your first time using a breadboard, take some time to make sure all the connections are in the right place. The colors of the wires don't matter. However, GND is denoted using a black wire and VCC/5V/PWR is denoted with a red wire. The two resistors, each connected in series (acting like a connecting wire itself) with the LEDs, limit the current flowing to the LEDs, making sure they don't blow up. As before, create a new sketch and paste in the following code

Code

```
/*
Double Blink
Turns on and off two LEDs alternatively for one second each
repeatedly.

This example code is in the public domain.
*/

int led1 = 12;
int led2 = 13;

void setup() {
// initialize the digital pins as an output.
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);

// turn off LEDs before loop begins
digitalWrite(led1, LOW); // turn the LED off (LOW is the voltage
level)
digitalWrite(led2, LOW); // turn the LED off (LOW is the voltage
level)
}

//the loop routine runs over and over again forever:
void loop() {
digitalWrite(led1, HIGH); // turn the LED on (HIGH is the voltage
level)
digitalWrite(led2, LOW); // turn the LED off (LOW is the voltage
level)
delay(1000); // wait for a second
digitalWrite(led1, LOW); // turn the LED off (LOW is the voltage
level)
digitalWrite(led2, HIGH); // turn the LED on (HIGH is the voltage
level)
delay(1000); // wait for a second
}
```

Once again, make sure the connections are made properly, especially the

positive LEDs (the longer one to OUTPUT PIN) and the negative (the shorter to the GND) terminals. Save the code as DoubleBlink.ino. Now, if you make any changes to it, you can always retrieve the backup. Upload the code. 3 2 1 And there you have it, an alternating LED blink cycle created purely with the Arduino. You can try changing the delay to see its effects.

Excersice 3: Using MFRC522 RFID with Arduino

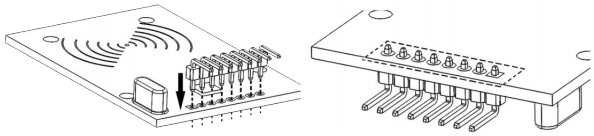
Radio-Frequency Identification (RFID) is the use of radio waves to read and capture information stored on a tag attached to an object. A tag can be read from up to several feet away and does not need to be within direct line-of-sight of the reader to be tracked. This is the advantage over Bar-code. A RFID reader is a device used to gather information from an RFID tag, which is used to track individual objects. Radio waves are used to transfer data from the tag to a reader. A passive tag is an RFID tag that does not contain a battery, the power is supplied by the reader. When radio waves from the reader are encountered by a passive rfid tag, the coiled antenna within the tag forms a magnetic field. The tag draws power from it, energizing the circuits in the tag.

Components Required

1. Hardware: arduino UNO, 1K resistor (two pieces), RC522(RFID reader module), x2 LED(Red,Green), x1 Breadboard and enough JUMPER
2. Software: arduino IDE.

Solder the pins

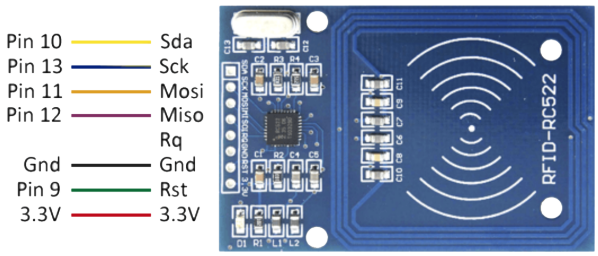
Begin by attaching header pins to the RFID reader. The header pins should be connected such that the exposed headers are on the same side of the board as the screenprinted text and MFRC522 chip. Reversing the board, solder the pins to the exposed pads on the underside.



Connection between the RFID module and Arduino

Now, its time to connect our Arduino with the RFID reader. Refer to the PIN wiring below, as well as the Connection schematic diagram for easy reference.

Double check before you connect your arduino with your computer, the wrong connection could damage your arduino



RC522	Arduino Uno
SDA (SS)	Pin 10
SCK	Pin 13
MOSI	Pin 11
MISO	Pin 12
IRQ	-
GND	GND
RST	Pin 9
3.3V	3.3V

Programming of Module RC522

In order to work the Module in Arduino it is necessary to download its corresponding library, which we will use will be one of the most common ones, the one developed by Miguel Balboa (RFID library). In menu `Tool`, `Manage library` and while updating the list we look for the library RC 522. The library that we install is identified by `RFC522 by github community`

1. Plug your arduino into your computer and select the correct board and com, then verify and upload your sketch
2. Scan your card , your should saw a bunch of numbers pop out
3. For the first program we will use one of the examples that is installed with the library MRFC522, we select `dump info`. It is a program to dump information. When we bring the card to the reader, we will obtain information about it.
4. We will analyze the lines of the code

Questions

1. Analyze the result of the information dump.
2. Identify the UID of the card and the keychain
3. Analyze the technical sheet of the integrated circuit of the card
4. What is the capacity of EEPROM memory?

5. How many sectors and blocks does EEPROM have?
6. In which sector and block of memory is the UID of the card stored?
7. Are there any methods of encrypting the information stored on the card?

Excersice 4

Execute the following code and analyze its operation.

Code

```
#include <SPI.h>                          // includes SPI bus library
#include <MFRC522.h>                      // includes specific library for MFRC522

#define RST_PIN  9                        // constant to reference a reset pin
#define SS_PIN  10                        // constant to reference a pin of slave select

MFRC522 mfrc522(SS_PIN, RST_PIN);         // create mfrc522 object by sending pins: slave select and reset

void setup() {
  Serial.begin(9600);                     // initializes communication by serial monitor at 9600 bps
  SPI.begin();                            // initializes SPI bus
  mfrc522.PCD_Init();                     // initialize reader module
}

void loop() {
  if ( ! mfrc522.PICC_IsNewCardPresent())         // if there is not a card present
    return;                                       // return to the loop waiting for a card

  if ( ! mfrc522.PICC_ReadCardSerial())           // if you can not get card data
    return;                                       // return to the loop waiting for another card

  Serial.print("UID:");                           // presents UID text:
  for (byte i = 0; i < mfrc522.uid.size; i++) {   // loop to check the UID one byte at a time
    if (mfrc522.uid.uidByte[i] < 0x10){           // if the read byte is less than 0x10
      Serial.print(" 0");                         // print white space and number zero
      }
      else{                                       // otherwise
      Serial.print(" ");                          // print a blank
      }
    Serial.print(mfrc522.uid.uidByte[i], HEX);    // prints the UID byte read in hexadecimal
  }
  Serial.println();                               // new line
  mfrc522.PICC_HaltA();                           // stop communication with card
}
```

Excersice 5

In the piece of code below you need to change the UID of the your card and keychain. Execute the following code and analyze its operation.

Code

```
#include <SPI.h>                          // includes SPI bus library
#include <MFRC522.h>                      // includes specific library for MFRC522

#define RST_PIN  9                        // constant to reference a reset pin
#define SS_PIN  10                        // constant to reference a pin slave select

MFRC522 mfrc522(SS_PIN, RST_PIN);         // create mfrc522 object by sending pins : slave select and reset

byte LecturaUID[4];                               // create array to store the read UID
byte Usuario1[4]= {0x90, 0x0E, 0xE4, 0xA4} ;      //  UID card read in the previous exercise
byte Usuario2[4]= {0x06, 0x76, 0x25, 0xD9} ;      // UID keychain  read in previous exercise

void setup() {
  Serial.begin(9600);                     // initializes communication by serial monitor at 9600 bps
  SPI.begin();                            // initializes SPI bus
  mfrc522.PCD_Init();                     // initialize reader module
  Serial.println("Ready");                // Presents the text: Ready
}

void loop() {
  if ( ! mfrc522.PICC_IsNewCardPresent())                 // if there is not a card present
    return;                                               // return to the loop waiting for a card

  if ( ! mfrc522.PICC_ReadCardSerial())                   // if you can not obtain data from the card
    return;                                               // return to the loop waiting for another card

    Serial.print("UID:");                                 // shows the text: UID
    for (byte i = 0; i < mfrc522.uid.size; i++) {         // loop to check the UID one byte at a time
      if (mfrc522.uid.uidByte[i] < 0x10){                 // if the read byte is less than 0x10
        Serial.print(" 0");                               // print white space and number zero
        }
        else{                                             // otherwise
          Serial.print(" ");                              // print a blank
          }
          Serial.print(mfrc522.uid.uidByte[i], HEX);      // print the UID byte read in hexadecimal
          LecturaUID[i]=mfrc522.uid.uidByte[i];           // store in array the UID byte
          }

          Serial.print("\t");                             // print a tab space

          if(comparaUID(LecturaUID, Usuario1))            // call compareUID function with User1
            Serial.println("Welcome User 1");    // if it returns true it presents the text: Welcome
          else if(comparaUID(LecturaUID, Usuario2))       // call compareUID function with User2
            Serial.println("Welcome User 2");    // if it returns true it presents the text: Welcome
           else                                           // otherwise
            Serial.println("Denied access ");             // shows text: denied access

                mfrc522.PICC_HaltA();                     // stop communication with card
}

boolean comparaUID(byte lectura[],byte usuario[])         // compareUID function
{
  for (byte i=0; i < mfrc522.uid.size; i++){              // loop to check the UID, one byte at a time
  if(lectura[i] != usuario[i])                            // if UID byte read is different from user
    return(false);                                        //returns false
  }
  return(true);                                           // if the 4 bytes match, it returns true
}
```

Excersice 6. Hand-in (at the start of next lab)

1. Write code to design and implement an RFID access control system
2. Write a report considering the following format
   (a) Course Title, Lab no, Lab title, your name, and date.
   (b) Section on the lab experiment: Brief description of the lab experiment including the goals and discussion on the theory of operation. Schematics of the circuit (take a screen capture of the schematics). Discussion of the results indicating that the circuit functions properly.

(c) The lab report is an important part of the laboratory. Write it carefully, be clear and well organized using LaTeX. It is the only way to convey that you did the job in the lab.