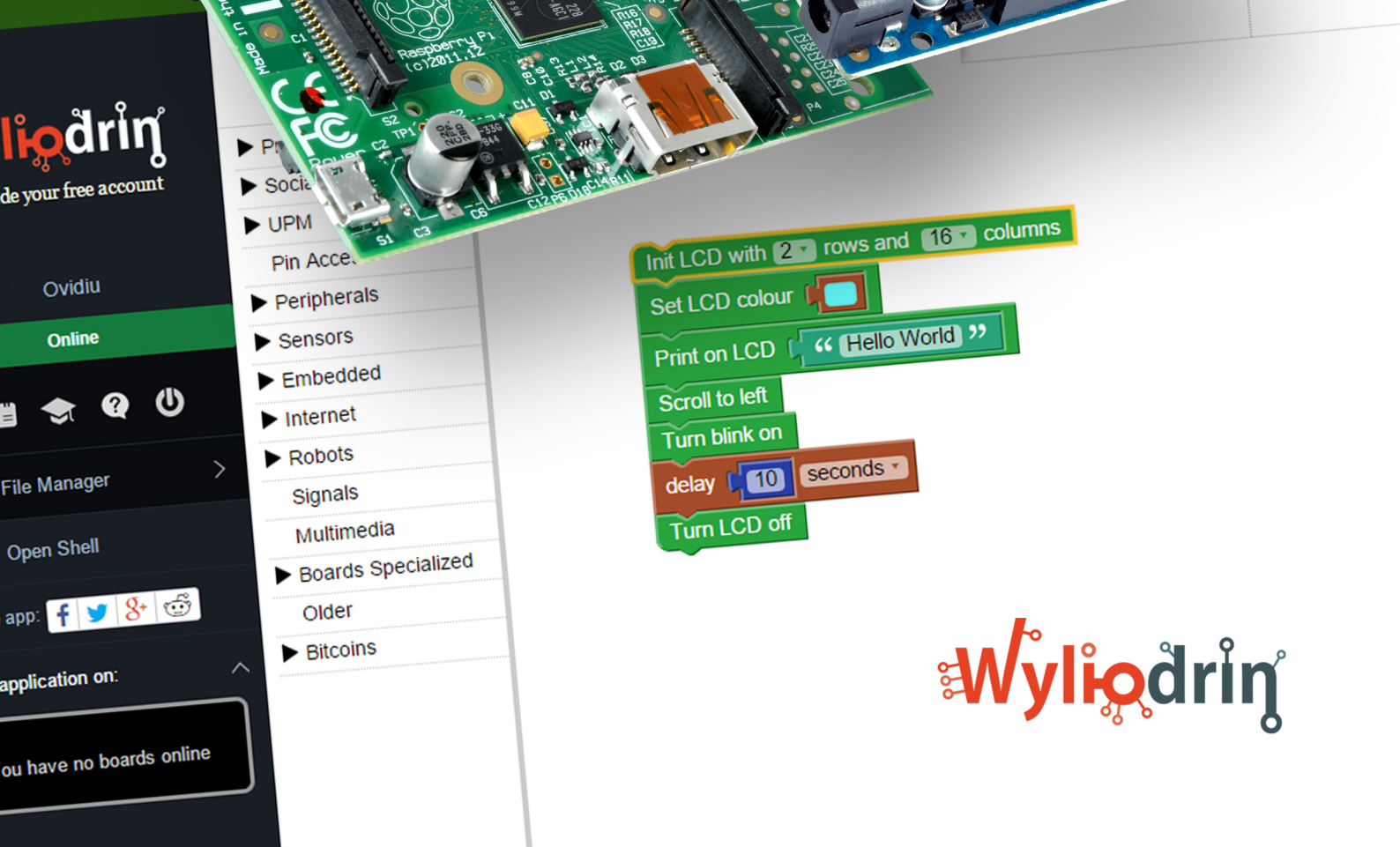
Madalina Tanea Ioana Culic Alexandru Radovici

Getting started guide for Raspberry Pi and Arduino using Wyliodrin



Wyliodrin

Getting Started Guide for Raspberry Pi and Arduino using Wyliodrin

Madalina TANEA, Ioana CULIC, Alexandru RADOVICI

June 1, 2015

We would like to thank Cristian Rusu.

Illustrations by Ovidiu Stoica.

[1] http://creativecommons.org/licenses/by-sa/4.0/

Contents

I Introduction **5**

About Raspberry Pi and Arduino **7**

Introduction to Linux **19**

Introduction to Electronics **27**

Raspberry Pi Setup **35**

Introduction to Wyliodrin **43**

II Tutorials **51**

Radio Station **53**

LED Blink **57**

SOS Morse Code Signaler **63**

LED line **73**

Control LED from Dashboard **95**

Control LED from Twitter **107**

Show Facebook likes on a 7 Segment Display **113**

Connecting Sensors to Arduino **123**

Write "Hello World" on the LCD **141**

Use a button **145**

Light up a multicolor LED **153**

Social Lamp **157**

Smart Parking **169**

III Reference 185

Visual Programming **187**

IV Appendix 209

Resistor Color Code **211**

Manually Deploy a Project On the Raspberry Pi **215**

Raspberry Pi Does Not Appear Online **217**

Part I

Introduction

About Raspberry Pi and Arduino

Raspberry Pi and Arduino are the boards most people use in order to build Internet of Things projects. They are widely used especially due to the small price and the high resistance to current spikes and short-circuits.

A common question people eager to get stared in electronics pose is whether they should get a Raspberry Pi or an Arduino. The answer would be: "it depends on what you want to build". While the Raspberry Pi is powerful and good for processing, is lacks real time processing, Arduino is a real time processor that lacks a lot of memory and processing power. If you want to connect simple sensors and read real time data, use the Arduino. If you need to process a large amount of data or connect to the network, use a Raspberry Pi. Usually you will want to use them together. Acquire data with the Arduino and process is using the Raspberry Pi.

Raspberry Pi

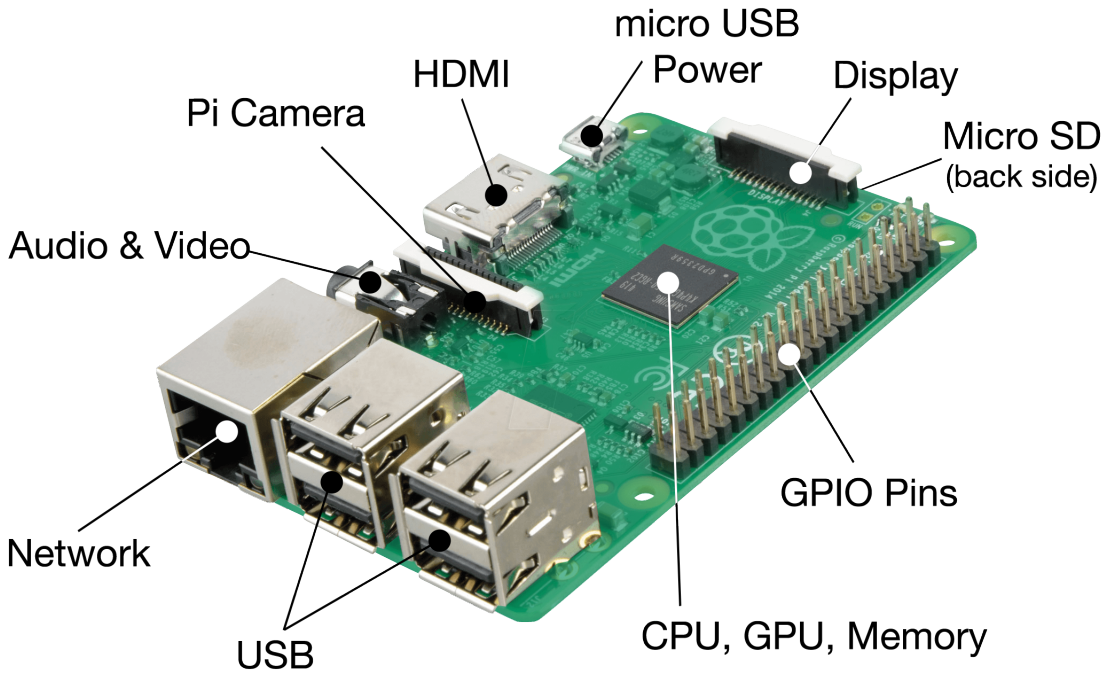


Figure 1: Raspberry Pi 2 model B Board

Main features of the Rapberry Pi

The Raspberry Pi is a very small computer that has the characteristics of the computers people would use 15 years ago.

- ARM 900MHZ processor[1] and 1 GB of RAM;
- HDMI port, an Ethernet port and several USB ports, depending on its version;
- Several pins to use for electronics:

[1]Raspberry Pi 2 Model B

  - two 5V pins;
  - two 3.3V pins;
  - 5 ground pins;
  - 17 data pins;
  - 1 PWM pin (pin 1).

Since it is a computer, the Raspberry Pi runs an Operating System. That means that you can run multiple programs on it and you can run applications that use Internet services. However, this also implies that the application you run on the Raspberry Pi is not real time, thus you cannot estimate when a certain sequence will get executed.

Raspberry Pi Pins Layout

As previously stated, the Raspberry Pi exposes some pins that allow you to connect peripherals to the board. Let's check them out.

The Raspberry Pi (A/B model) has 26 GPIO soldered pins and 5 GPIO pins that you need to solder. These may be used to control electronics connected to the Raspberry Pi. Among the things that can be done are:

- light up LEDs;
- place buttons;
- use relays;
- control motors

It is important to know how these pins can be accessed. For the Raspberry Pi, Wyliodrin uses the WiringPi pins layout described in the figure 2.

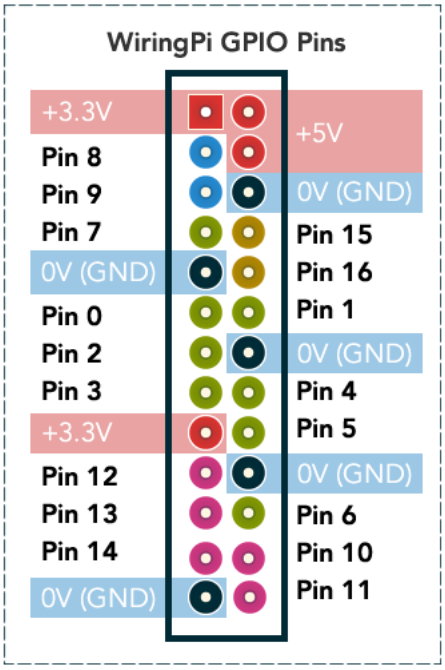


Figure 2: Pins layout Raspberry Pi model A/B

Place the Raspberry Pi with the SD Card facing upwards (figure 3).

Figure 3: Board orientation

The data pins consist of *digital* pins but also pins that support *SPI* or *I2C* communication. You will read more about each of these protocols and how to use them in the following chapters. The Raspberry Pi model B+ has more pins (figure 4).

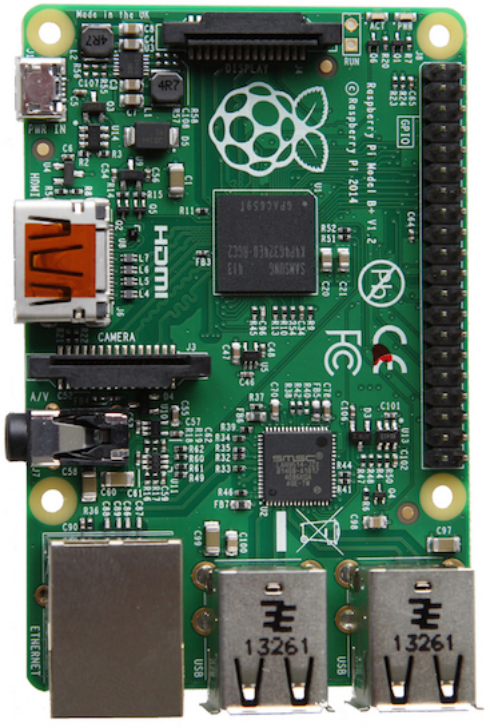


Figure 4: Raspberry Pi B+

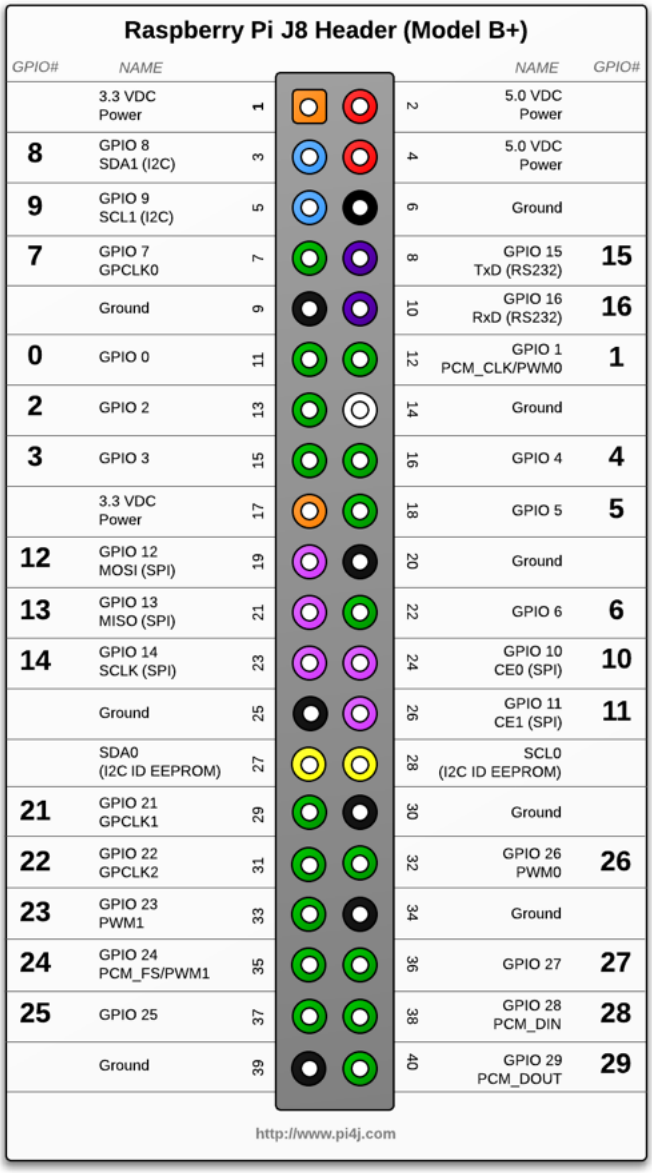


Figure 5: Pins layout Raspberry Pi model B+ and Raspberry Pi 2

Tips & Tricks

There is a visible difference between the Raspberry Pi models B and B+ regarding the pins' layout.

- On the Raspberry Pi model A/B as you start counting from top to bottom, pin 1 provides 3.3V. You can figure out the other pins from there. Even numbers of pins are placed on the right side and the odd numbers on the left.

- The model B+ offers 14 more pins compared to the 26 pin header on the previous Raspberry Pi, including 8 extra GPIO pins, 3 I2C pins and 3 ground pins. You can notice that the first 26 pins from the bottom are the same on both boards. The extra 14 pins are added to the bottom, so any project created on a Raspberry Pi A/B looks the same on a B+ model, you will just have 14 extra pins.

Be advised that pins run at 3.3V. Connecting them to 5 V pins (like the Arduino) might damage the board.

Arduino

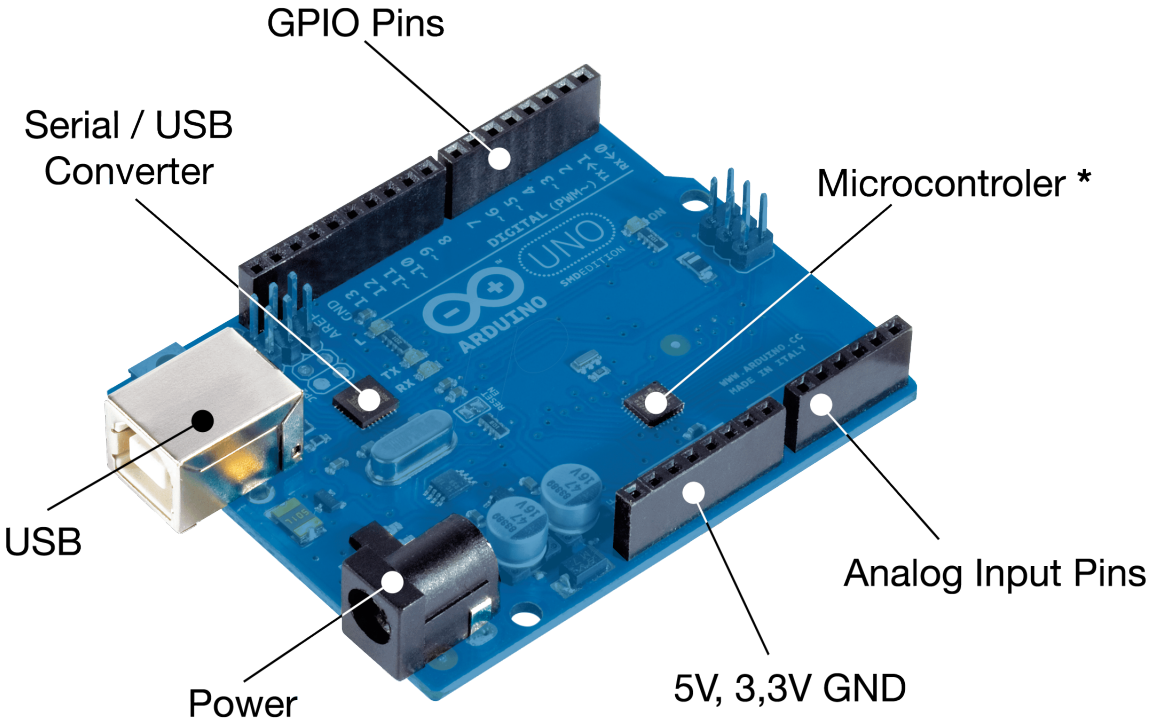


Figure 6: Arduino Board

Unlike the Raspberry Pi, Arduino is a microcontroller board. That means that it does not run an Operating System, what it runs is called Firmware. Basically, it runs only one program. The result is that you can estimate what program sequence gets executed at a certain time.

On the other hand,most of the Arduino boards cannot simply connect to the Internet. However, they expose a higher number and a greater variety of pins.

Depending on the model, Arduino has characteristics around:

- 16 Mhz microcontroller

- 32 KB of program memory
- 2 KB of RAM memory
- USB/serial converter

Program written to the Arduino remain there until they are replaced with another program. Even when powered off, Arduino stores its software.

Arduino Layout

As we said, Arduino is a simple board that executes only one program, but it exposes a large variety of pins. There are multiple Arduino boards on the market, the main differences between them consisting in the processing power and the number of pins. This is why it is difficult to state the exact number of pins a board has and their numbering. However, you can find the following types of pins on any Arduino board:

- 5V pins;
- 3.3V pins;
- data pins;
- analog input pins;
- reset pin (it allows to reset the board);
- AREF (used as reference when reading analog values).

The data pins can be used for *digital input/output* and also for *SPI* and *I2C* communication.

It is easy to identify the pins' number as next to each pin is written its number (1, 2, 3... for data pins and A0, A1... for analog input pins). In figure 7 you

can see an example of an Arduino board with its pins layout.

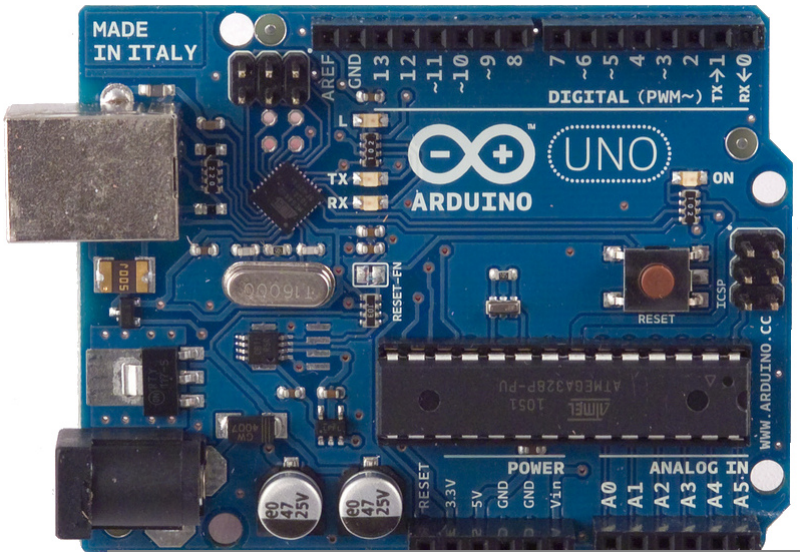


Figure 7: Arduino Uno pins layout

You will get more information on these pins and how to identify them in some future chapters.

Breadboard usage

The breadboard is a kind of solderless electronic circuit building. It basically replaces the cables you otherwise would have to solder in order to connect the pheripherals together.

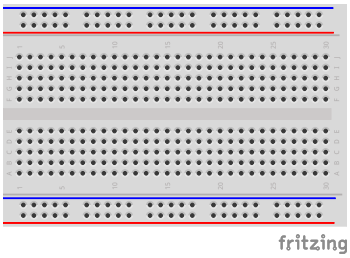
There is a certain number of holes on a breadboard. The common type of breadboard has two areas called strips.

- The bus strips usually are wired to the power supply of the circuit. They are arranged on two rows, one for voltage, usually marked with a red line along the row,the other one for ground, drown in black.

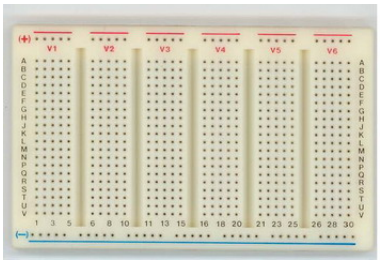
- The socket strips which reunite the components of a circuit. These

have a layout of multiple columns, each consisting of 5 holes.

Beware, each of the rows and each of the columns behave like a single hole, as their holes are connected as a series. In Figure 8a the columns are oriented vertically as opposed to Figure 8b, where the socket strips columns are horizontal.



(a) Vertical socket strips breadboard



(b) Horizontal socket strips breadboard

Figure 8

Conclusion

All in all, you can say that Raspberry Pi works well for applications that require both mobility and an Internet connection. On the other hand the Arduino is the best option for "electronics-only" projects.

However, the best solution to create amazing Internet of Things projects is to combine these two. The Arduino can be connected to the Raspberry Pi so that you have both Internet access and a full rage of different pins.

Introduction to Linux

The Raspberry Pi board usually runs Raspbian as operating system, which is based on the Debian Linux distribution. A Linux distribution is an operating system built around the Linux kernel. The main difference between different distributions is the package manager, basically, the way you install new software.

As any other operating system, Raspbian allows you to control the board via a Shell by using standard Linux commands. Although this sounds intimidating, especially nowadays when the GUI makes everything intuitive and extremely easy, sometimes it is the only viable option.

Further on the book will present the Shell's characteristics together with some of the basic commands and when you might need to use them.

The Shell

The Shell is a window that allows you to interact with the board. It waits for you to enter a command and it executes it. Once you open a Shell, you will see the prompt. That means that everything works fine and the Shell is waiting for your command.

The prompt also offers you some information. First of all, it shows you the user currently logged in. The user's name is that you see before @. It also

username@hostname:~$

Figure 9: Shell prompt

shows the host name of your board.

The most important information the prompt displays is the working directory. That is the directory where you are currently working in. It is displayed right after the colon in the prompt. You will notice that the default working directory is `~`. That is the user's home directory and its equivalent is */home/username*.

Paths

In order to access a certain file or directory, you have to take into account the path to it. There are two different paths you can use: absolute and relative.

In Linux, the directories' structure is like a tree. The root directory is `/` and it contains all the other directories and files.

If you use an absolute path to a file or a directory, that means that you build the path to it starting with the root directory. Thus, you can say that any path that starts with / is an absolute path.

On the other hand, you can use a relative path, which means that you build it starting from the directory you are working in, your working directory. Thus, all the files and directories are relative to it.

When building paths, there are three symbols you should be familiar with:

- . - current directory
- .. -parent directory

- ~ - home directory (/home/username)

pwd

The *pwd* command makes the Shell print the working directory. It is important to know which directory you are working in and sometimes it is difficult to get it from the prompt. So, anytime you feel lost, use *pwd*.

```
pi@raspberry:~$ pwd
/home/pi
```

Figure 10: Example of pwd output

ls

ls makes the Shell print all the files and directories located in the working directory. If you want to see la contents of some other directory, you can pass that directory as an argument to the command. For instance, if you want to print all the files and directories in /, you will write: *ls /* .

cd

You already know that once you open a Shell, the working directory is your home directory. However, you will need to work in other directories too. In order to change the working directory, you will have to use *cd* followed by the directory you want to go to.

For example, if your home directory contains a directory called *homework* and you want to have that as the working directory, you use *cd homework*. You can notice that you used an absolute path. Some other alternatives would

be *cd /home/pi/homework* or *cd ~/homework*. In the last two examples you used an absolute path to refer to *homework* directory.

cat

cat asks the Shell to print the contents of a file. However, it must be clear that you can only see its contents, you cannot modify them. For that you need an editor.

Just like with the *cd* command, *cat* gets as an argument the file it should display.

Example: *cat /etc/passwd*

htop

By using the *htop* command you can see real-time all the processes that run on your board. Once you entered the command you will notice that the prompt does not appear, that is because you cannot enter another command until you are finished with displaying the processes. So, if you want to go back to what you were doing, just hit the *q* key.

For each process displayed, you can see its PID (Process ID), the user who launched the process, how much CPU and memory it is using, the command that started the process and other information.

What you are most interested in is the PID. That is because each process can be identified by its PID and if you want to interact with it, you have to know its process ID.

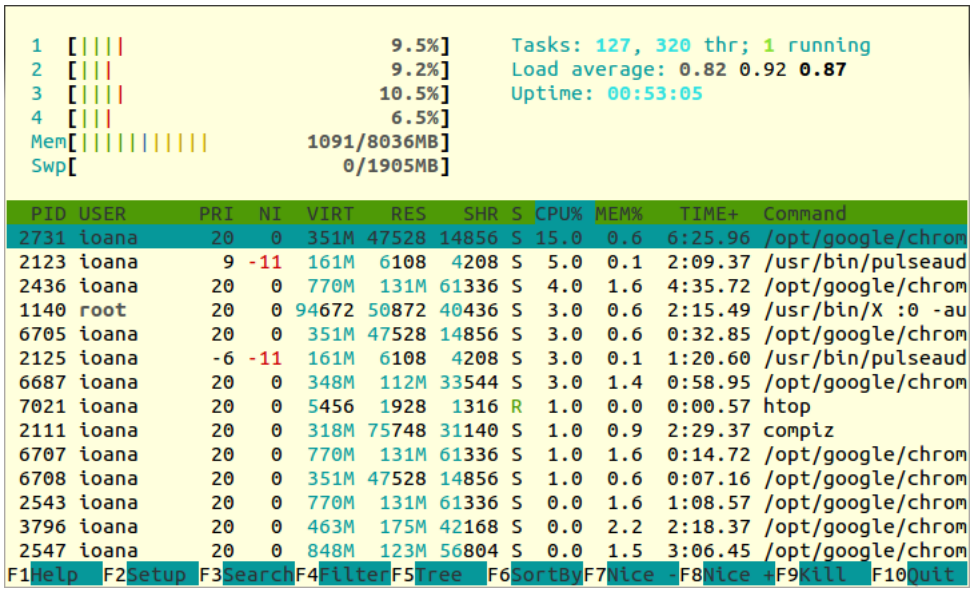


Figure 11: htop output

kill

We know that you can use *htop* to find a process' ID in order to be able to interact with it. *kill* is the command that allows us to interact with another process.

Two processes can interact by using signals. A signal is a number a process sends to another. Both processes know that each number represents an action. you can refer to a signal either by the number or by its name.

```
ioana@ioana:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Figure 12: List of signals

The format of the *kill* command is the following: *kill -signal pid*, where signal is the number representing the action you want to process to do and pid is the process ID.

The two signals you are most interested in are *SIGTERM* (number 15) and *SIGKILL* (number 9).

SIGTERM tells the process to stop its execution. Normally, the process should save all its data and stop running. However, this signal can be ignored by the process. There are times when you cannot kill a process by using SIGTERM.

On the other hand, SIGKILL, kills the process no matter what. The downside is that the process does not have the opportunity to save its data, so killing it like this can result in loss of data. Neverthless, if something happened and your process must be forced to stop, you have to use SIGKILL.

In case the running process has a Shell attached and you can access it, you can simply use a key combination to send the SIGTERM signal to it and make it stop, *Ctrl+C* .

killall

killall has the same effect as *kill*, except that you do not have to know the PID of the process, but its name. Instead of passing the process ID as an argument, you have to pass the process name.

Tips & Tricks

Getting used to working with a Linux Shell is not difficult, especially if you know the following tricks:

- Whenever you are typing a command use the *TAB* key. It will auto complete what you wanted to type, thus eliminating spelling errors. In case there are multiple possibilities, press TAB once more and they will be displayed. If by pressing TAB the command or the argument you want to type is not automatically filled in, it means the command is not valid.

- The most important command you should know is *man*. By using *man* followed by another command name, you have access to that command's manual and you can find how to use it and all it can do.

Introduction to Electronics

You are going to build IoT projects around Raspberry Pi and Arduino. However, these two boards are only a part of the projects, they do all the computing, but you also need I/O devices that you will connect to them. The devices are mainly sensors, LEDs and LCDs. In order to correctly connect the peripherals, you need to be acquainted to basic electronics notions, otherwise, you risk to burn the I/O devices and even the boards.

Ohm's Law

The Ohm's law states that in a circuit the current (I) is directly proportional to the applied voltage (U) and inversely proportional to the resistance (R) of the circuit.

$$I = \frac{U}{R} \tag{1}$$

Kirchhoff's Circuit Laws

Before you state the two laws, you need to understand the following notions:

- junction/node - the place where at least three conducting branches meet
- loop - a closed path, including at least two nodes

Kirchhoff's First Law

Kirchhoff's First Law states that in a node, the sum of the currents is 0.

$$\sum_k i_k = 0 \tag{2}$$

Please keep in mind that currents have directions. Currents incoming have negative values, while currents outgoing have positive values.

Kirchhoff's Second Law

Kirchhoff's Second Law states that the sum of the voltage in a circuit loop is equal to the power source voltage.

$$\sum_k E_k = \sum_k R_k I_k \tag{3}$$

Example:

You have a 3V source and three resistors of different resistance (figure 13). The sum of voltage drops on each of them is equal to the source voltage.

$$R1 + R2 + R3 = VCC1 => 0.25v + 1.25v + 1.5v = 3v \tag{4}$$

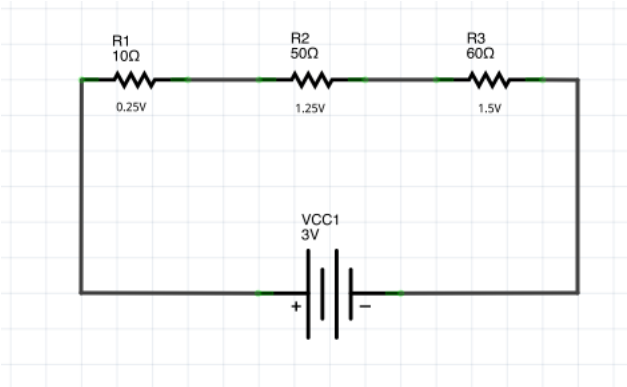


Figure 13: Kirchhoff's second law example

LED

This chapter explains how to correctly connect an LED to a Raspberry Pi or an Arduino.

First of all, you need to know what a diode is.

A diode is an electronic component that has a positive and a negative side and it basically allows the current to flow only in one direction, from positive to negative.

The LED is also a diode. When current is flowing through the LED, it lights up. So in order to light up an LED you need to put the high voltage at the anode and the low voltage at the cathode.

Schematics

Taking into account the theory stated previously, you would build a circuit like the one in figure 14 to light up an LED.

You must take into account that the power source depicted will be replaced in the projects you are going to build with the Raspberry Pi or the Ar-

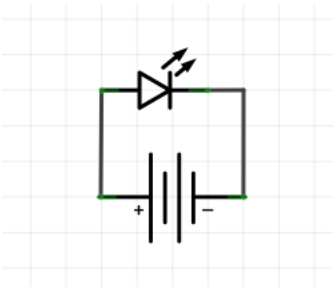


Figure 14: LED schematics example

duino.

There is only one tiny problem with the schematics in figure 14: it is a short circuit. That means there is no resistance to limit the current because the diode does not have any resistance at all. It just allows the current to flow. That can cause big problems (you can damage your Raspberry Pi, for example). To fix this, you need a resistor to limit the current flow (figure 15).

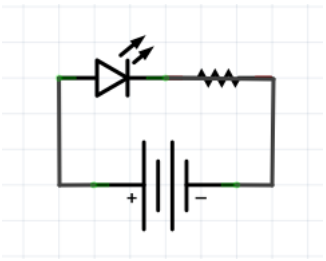


Figure 15: LED correct schematics example

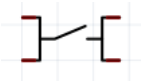
Button

This chapter explains how to correctly connect a button to a Raspberry Pi or an Arduino.

A button, also called a switch, is an electric component that can break an electrical circuit by interrupting the current.

When used in schematics, there are multiple possible symbols to depict it (figure 16).

(a)



(b)

Figure 16: Button symbols

Also, figure 17 depicts an example of circuit that uses a switch.

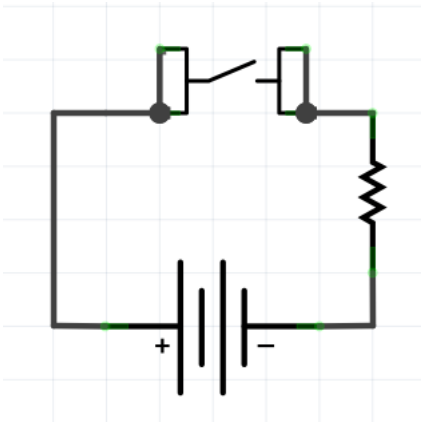


Figure 17: Button example circuit

When the button is pressed, it acts like a wire and it will let the current flow through the circuit. If the button is not pressed, the circuit is interrupted.

When a button is connected to a board, you can tell if the button was pressed by looking at the pin's value.

Let's see how you can connect a button to Raspberry Pi. The first possibility would be the one in figure 18, but it is *wrong*.

Why is it wrong? If the button is pressed, everything works fine. The value of pin would be HIGH and you can say "Yes, the button is definitely pressed".

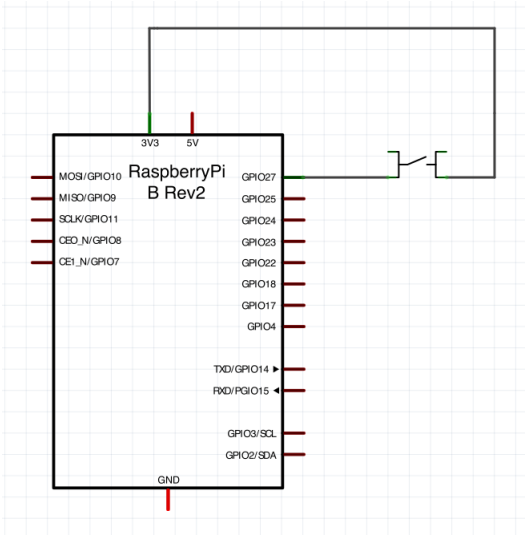


Figure 18: Button incorrectly connected to a Raspberry Pi

But what happens when the button is off? It is important to know that a logic level can be: LOW (or 0), HIGH (or 1) and also UNKNOWN (or high impedance). When the button is not pressed you cannot say for sure what logical level the pin has: it could be 0 as well as 1 because the wire is not connected neither to Ground nor to a power supply.

Let's give it another try (figure 19).

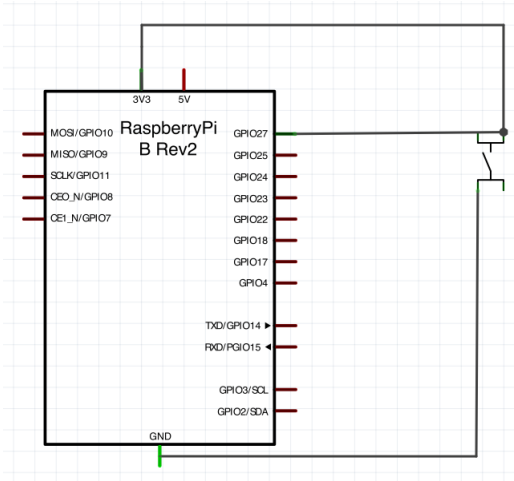


Figure 19: Button incorrectly connected to a Raspberry Pi

Figure 19 is also incorrect. When the switch is off the button's pin value is HIGH. The big problem appears when the button is pressed. It will create a short circuit: the ground is directly connected to VCC which is very bad because you do not have any resistor and the electric current is not limited.

The correct way to connect a button to a board is presented in figure 20.

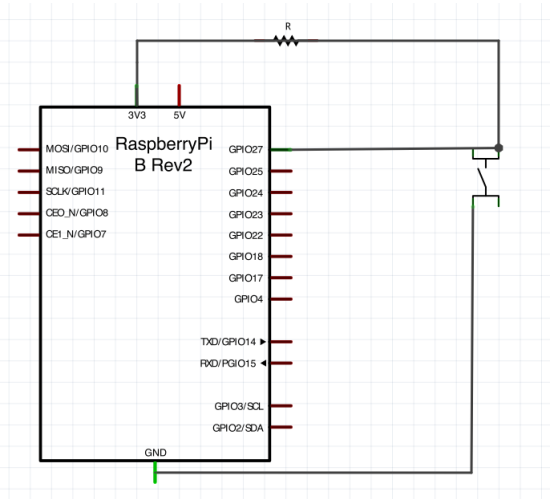


Figure 20: Button correctly connected to a Raspberry Pi

This time you will not have a short circuit because if the button is pressed there is the R resistor. The R resistor is called a *pull up resistor* and the whole system is called a *voltage divider*. If the button is pressed our pin's value will be LOW.

You can also connect the resistor to the Ground. Now you have a pull-down resistor and the pin's value will be HIGH when the button is pressed and low otherwise.

Safety Instructions

Whenever you connect anything to your Raspberry Pi or Arduino you must assure that the board is not powered up. Otherwise you might accidentally create a short-circuit and burn the board.

Only after you assured that everything is correctly connected, you may safely power up the board

Raspberry Pi Setup

For the projects you are going to build you will use a Raspberry Pi together with the Wyliodrin service. Wyliodrin allows us to remotely program and monitor the board.

Let's see how to setup your Raspberry Pi so you can access it via Wyliodrin .

Besides the Raspberry Pi, you will need an SD Card with minimum 4 GB (class 10 is recommended). you also need to assure the Raspberry Pi an Internet connection. You can either use an ethernet cable for a wired connection, or a WiFi dongle.

Setting up the board requires the following steps:

1. Add the board to Wyliodrin ;
2. Download the Wyliodrin Raspberry Pi SD Card Image;
3. Write the image to an SD Card;
4. Download the board configuration file and write it to the SD Card;
5. Insert the SD Card into to Raspberry Pi and connect the board to network.

Add the Board to Wyliodrin

You will go to http://www.wyliodrin.com . After signing in, you will find on the top of the projects page an *Add new board* button (figure 21). you just need to press the button to add the Raspberry Pi to the account.

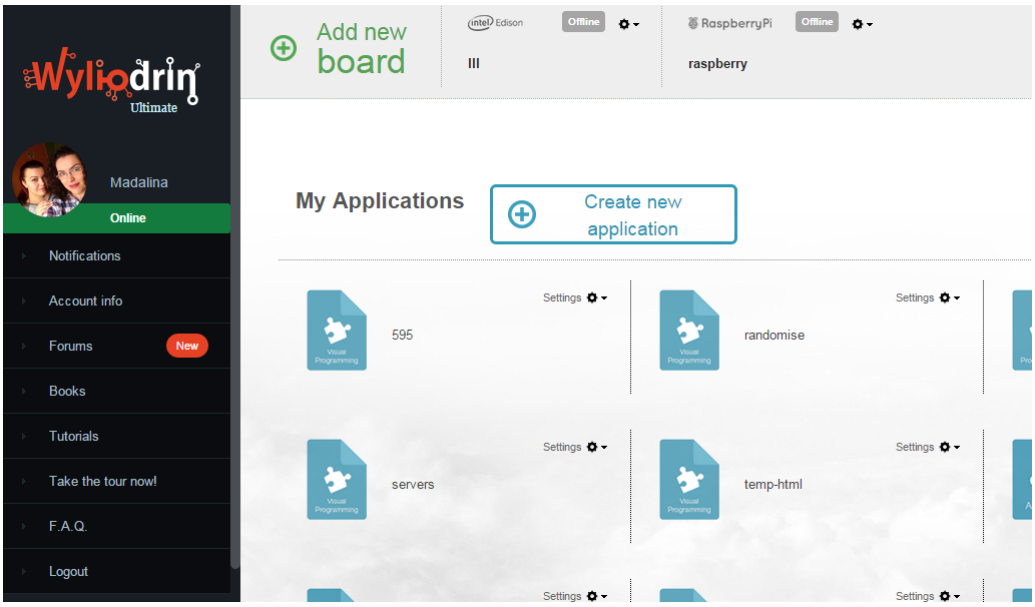


Figure 21: Add new board page

To add the board you have to complete these two steps:

1. Name the board and select the type;
2. Set up the network connection.

Then you press *Next* (figure 22).

On the network connection setup menu (figure 23), you mark *Use Wireless* only if you are going to connect to a Wireless network, otherwise you leave it unmarked. The *Pass Firewalls* option should not be marked, however, if your board does not appear online after completing all the necessary steps, try to reconfigure it and check this option.

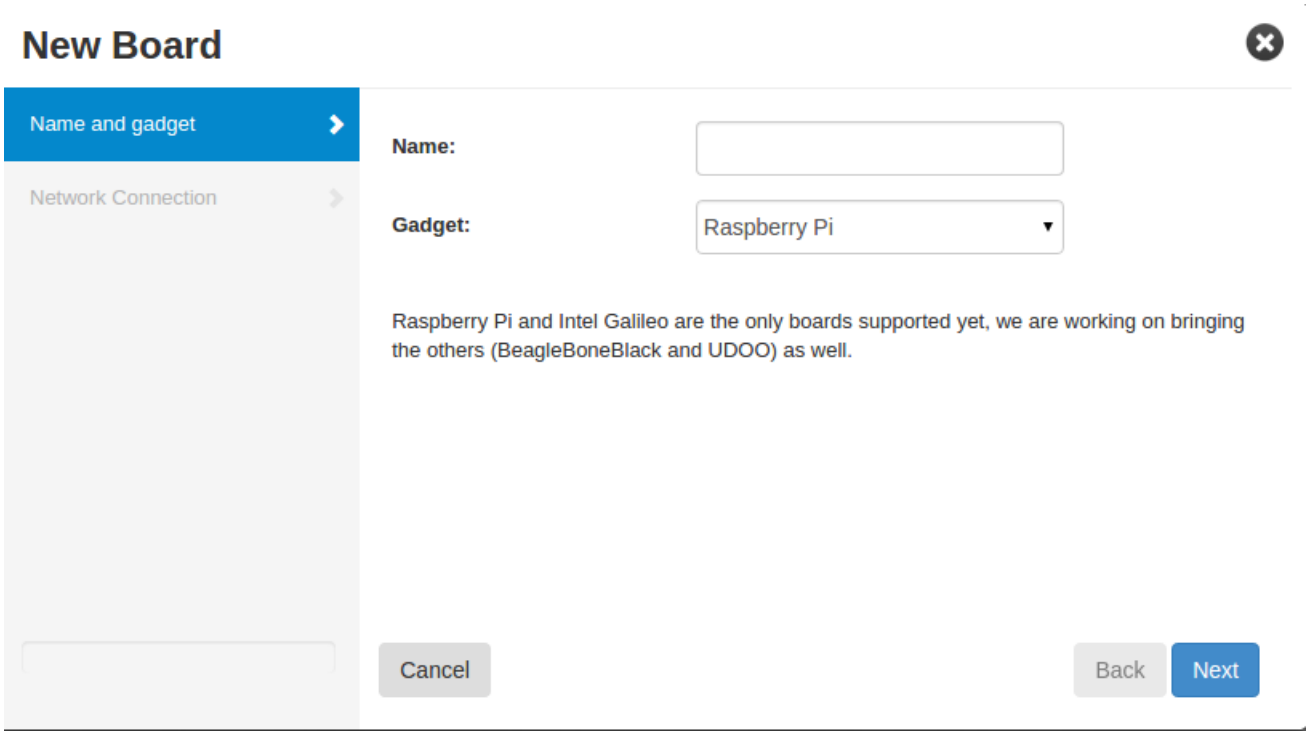


Figure 22: Add board preferences

A tutorial on how to connect the Raspberry Pi will appear.

Download the Wyliodrin Raspberry Pi SD Card Image

You have to download the SD Card image that you are going to use. You can find a link to that image in the window that depicts how to connect the Raspberry Pi to Wyliodrin . Once you click on the link, the image starts downloading.

You need to unzip the archive to obtain the SD Card image.

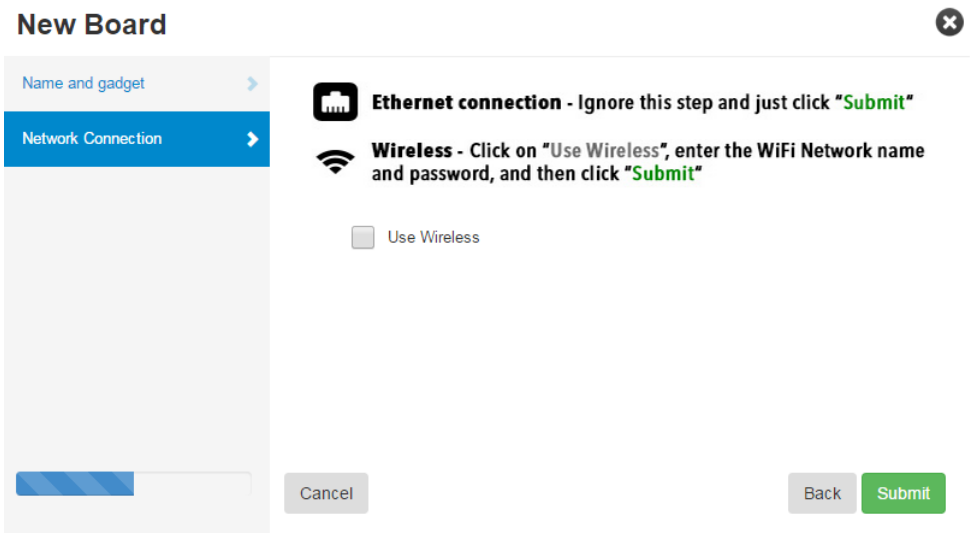


Figure 23: Add network preferences

Write the image to an SD Card

Writing the image to an SD Card is different depending on your system. you are going to present the steps you need to do depending on you computer's operating system.

Windows

1. Insert the SD card into your SD card reader and check what drive letter it was assigned. You can easily see the drive letter (for example E:) by looking in the left column of Windows Explorer. You can use the SD Card slot (if you have one) or a cheap Adapter in a USB slot.

2. Download the Win32DiskImager utility

   (http://sourceforge.net/projects/win32diskimager).

3. Extract the executable from the zip file and run the Win32DiskImager utility; you may need to run the utility as Administrator! Right-click

on the file, and select *Run as Administrator*.

4. Select the unzipped Wyliodrin SD Card Image for the Raspberry Pi.

5. Select the drive letter of the SD card in the device box. Be careful to select the correct drive; if you get the wrong one you can destroy your data on the computer's hard disk! If you are using an SD Card slot in your computer (if you have one) and can't see the drive in the Win32DiskImager window, try using a cheap Adapter in a USB slot.

6. Click Write and wait for the write to complete.

7. Exit the imager and eject the SD card.

Linux

1. Insert the SD card into your SD card reader. The card will appear in */dev* usually under the name of *mmcblk0*.

2. Open a terminal and enter the following command :

```
dd if=raspberrypi_image_file of=/dev/device_name
```

where *raspberrypi_image_file* is replaced with the name of the unzipped Wyliodrin SD Card Image for the Raspberry Pi and */dev/device_name* is replaced with the path to the SD Card, usually it will be */dev/mmcblk0*.

3. Wait until the process has finished.

4. Eject the SD card.

Mac OS

1. Insert the SD Card into the SD Card reader or use a cheap SD Card adapter for your computer.
2. Download PiWriter utility. This will be used for writing the Wyliodrin SD Card Image (https://github.com/Wyliodrin/PiWriter).
3. Run PiWriter. You will be prompted for an administrator user and password. You will need to have administrator right to use PiWriter. If unsure what to do, just type in your password.
4. Follow the instructions on screen
5. When prompted to select a file, select the unzipped Wyliodrin SD Card Image for Raspberry Pi.
6. Wait for the write to complete.
7. Exit the imager and eject the SD card.

Board Configuration File

Once you have written the Wyliodrin SD Card image to the card, you will need to download and copy to the card the Wyliodrin configuration file. The link to the file is also in the tutorial window. If you closed the window, just go to the Wyliodrin Projects page, click the icon on the right next to the boards name and select Configure. Read the instructions on screen, you will find a link to a file called *wyliodrin.json*.

Insert the SD Card into your computer. Once it appears, copy the file *wyliodrin.json* directly on the card. Make sure the file is named exactly *wyliodrin.json*.

Please be aware that each board has a different configuration file. Even if they are all called wyliodrin.json, the content is different for each board that you want to activate.

Connect the Board to Wyliodrin

Eject the card from you computer and insert it into the Raspberry Pi. If it is the case, connect the Raspberry Pi to an Internet cable and plug it in. You will have to wait a little, the board will appear as online on the Wyliodrin Projects page. You have successfully activated the Raspberry Pi and.

Now you can get down to creating the projects we will present further on. Enjoy!

Introduction to Wyliodrin

You are going to build all the projects by using the Wyliodrin platform. Wyliodrin allows remote access to the board. That means that you can write the program, upload it on the board, run it and finally monitor the board by using graphs.

You already got in touch with the platform in the previous chapter, when you configured the Raspberry Pi. Now let's see what other options you have access to.

Projects Page

Once logged in, you can see the projects page (figure 24). On that page you can add other boards by pressing the *Add new board* button and following the instructions described previously.

In the column on the left, there are some options:

- Notifications - shows messages you receive from the Wyliodrin team regrading changes to the platform;
- Account info - displays information on you account(name, email, account type, communication token, number of boards, number of projects);
- Tutorials - on click, it opens another window that shows Wyliodrin 's

wiki page; there you can find all sorts of tutorials and guidance to get started with using the platform;

- Take the tour - makes a quick tour of this page;
- F.A.Q. - opens another window with some of the most common questions answered; we recommend to check that page anytime you encounter a problem, you might find the solution there.

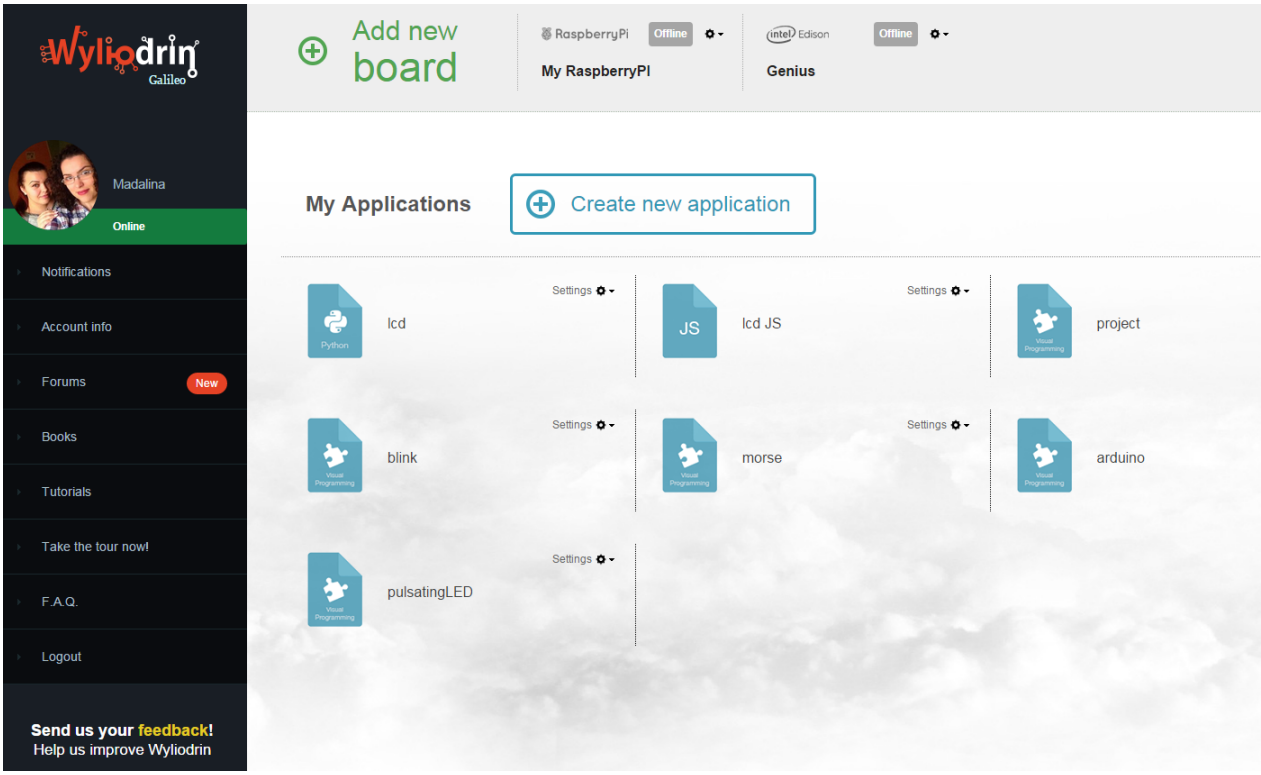


Figure 24: Wyliodrin projects page

Each board has also some options that you can access by pressing the settings wheel on the left of each board:

- Setup Tutorial - opens up the tutorial regarding adding a new board (you can find more information on this in the previous chapter);
- Download wyliodrin.json - allows you to download the board's config-

uration file in case you want to change any settings or you got another SD Card;

- Download SD Card Image - downloads the whole image in case you got another SD Card;
- Offline? - opens another window with solutions in case your board does not appear online;
- Board ID - displays the board's ID and, if enabled, the communication token; you will need this information for projects that make two or more boards communicate, or if you want to communicate with your board from a web or mobile application;
- Disable/Enable Open Messages - Open Messages has to be enabled in order to allow a web or mobile app communicate with your board;
- API Semantics - opens a wiki page that explains how Open Messages and Boards Communication work;
- Configure - allows you to change the board's settings; take notice that once you do that you need to replace the old wyliodrin.json file with a newly downloaded one;
- Extra Libraries - downloads and installs extra libraries on the board; usually the Raspberry Pi Sd Card image already contains all the libraries;
- Update Image - downloads and installs any updates available for the board's SD Card image;
- Update Streams - downloads and installs on the board the environment needed in order to create applications using Streams programming language;
- Shell - opens a shell where you can control the board by using Linux

commands we presented previously;

- Task Manager - displays all the running processes and allows you to kill any process;
- Power off - it powers off the board;
- Remove - it removes the board; this cannot be undone;

There is one more important button: *Create new application*. When you press it, a pop-up shows up where you have to enter information on the new application you want to create (figure 25).

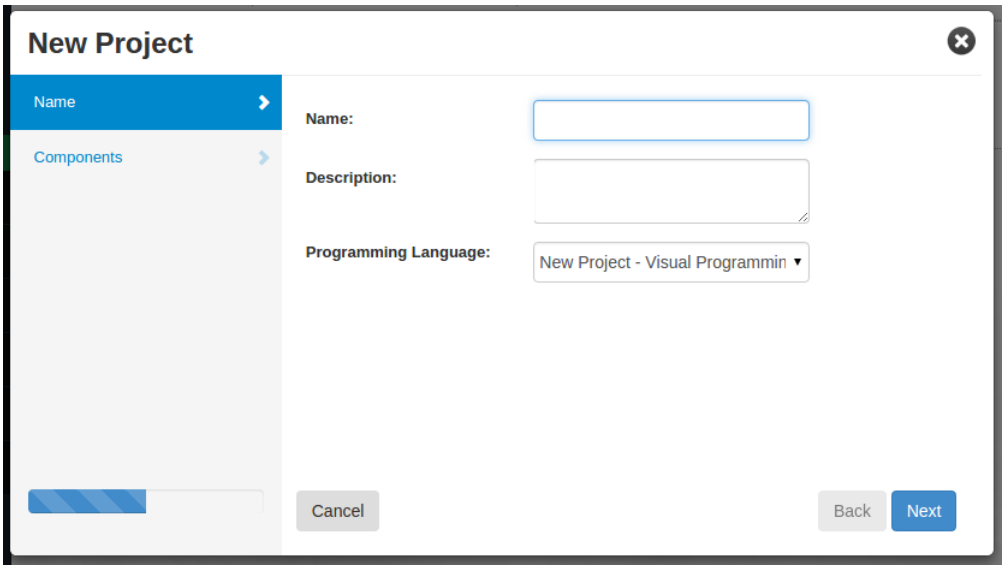


Figure 25: Add a new application

After you enter the name and the description, you have to choose the programming language you are going to use. Then you press next and you can check two more options regarding other external boards that you can connect to the Raspberry Pi. In the following chapters you are going to create projects using Arduino. For those, you will need to check the first option. Otherwise, you just leave them unmarked.

Once you click *Submit*, a new application is created. That means that you

can see it in the applications dashboard.

Application Page

When you click on an application, a new window is created. There you can write the application, you can build the graphs dashboard and finally you can run the application.

In order to run the application, you click on the board's name on the left. If you have multiple online boards, you can run it on each of them.

Application Share

The applications created can be shared with other Wyliodrin users. They can either watch real time while you develop the application, or they can clone the application and continue to work on it separately.

To share the application, you use the buttons on the left side (figure 26). You click on any of them and Wyliodrin will ask if you allow the project to be made public. you answer yes and press the button again to actually share the project. you can either send it via Facebook as a private message.

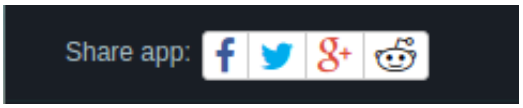


Figure 26: Share application buttons

The ones receiving the link can simply follow the changes applied to the project, or if they are logged in on the Wyliodrin website, can clone the project.

Graphs Dashboard

When you click on the *Dashboard* button on the right-top of the page, you can choose the graphs you want to monitor your board. Basically, you can use that graphs to display data coming from the sensors. The data is sent from the application as a signal, by using some special functions we will talk about later.The graph just receives a signal and displays its value.

To add a new graph, you simply have to choose one from the menu and drag and drop it.

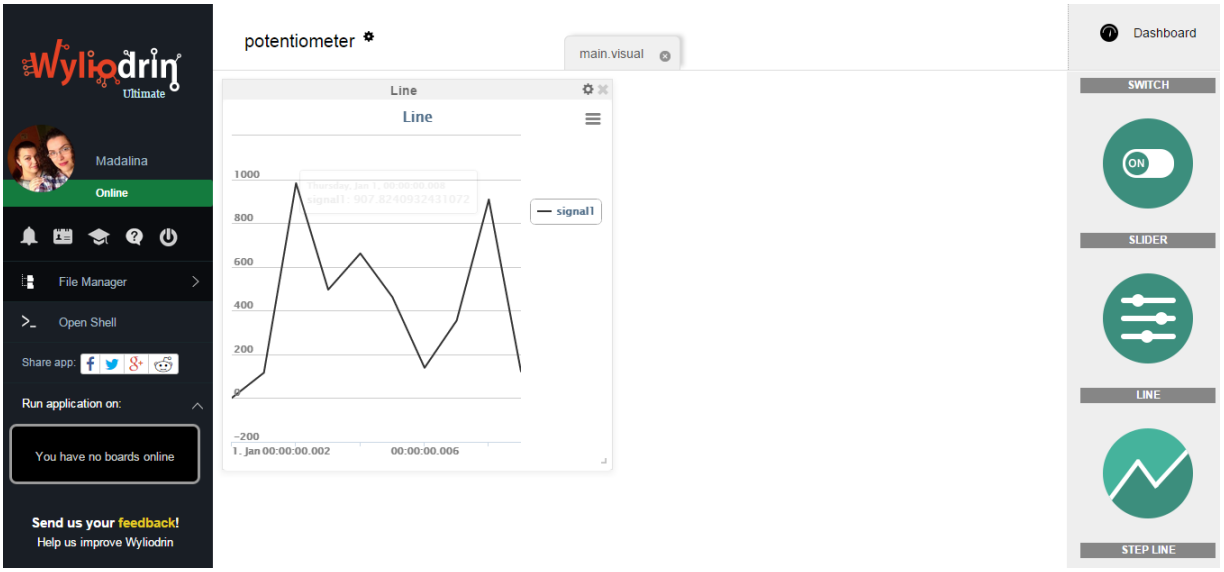


Figure 27: Line graph added to dashboard

To add a certain signal to the graph, you click on the settings wheel on top and a settings pop-up will be displayed (figure 28).

The settings menu allows us to add a new signal, you just type in its name and you can also choose a color. Then, you can change parameters like minimum and maximum axis values, legend or if you want the graph to change its axes as more data is being received or not.

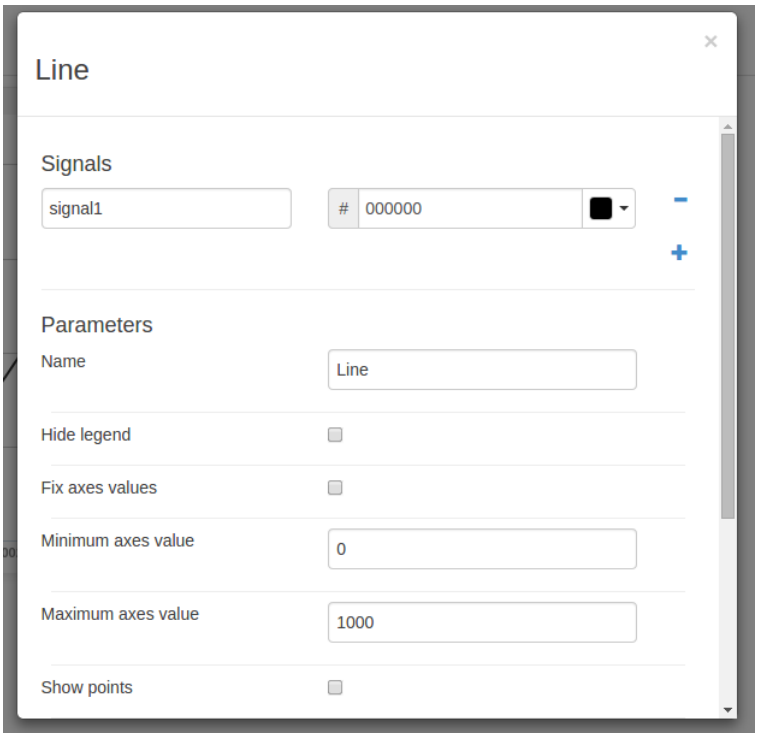


Figure 28: Line settings

Once you run the application that sends that signal to a graph, you will see how the values are being displayed on the graph.

Please pay attention that the name of the signal sent from the application is exactly the same as the signal attached on the graph. Otherwise, at runtime, no data will be displayed on the graph.

Part II

Tutorials

Radio Station

Let's put the Raspberry Pi at use!

In this project you are going to make a radio station out of the Raspberry Pi. Now you don't have to worry about music anymore.

What you need

- One Raspberry Pi connected to Wyliodrin ;
- Headphones.

The Setup

First of all, you need to connect the headphones to the Raspberry Pi. The board has an audio jack where you connect the headphones.

The Code

You will create a new application starting from an example.

After you press the *Add new application* button and name the application,

you select for the programming language *Music - Visual Programming*. This will create an application starting from an example for playing music (figure 29).

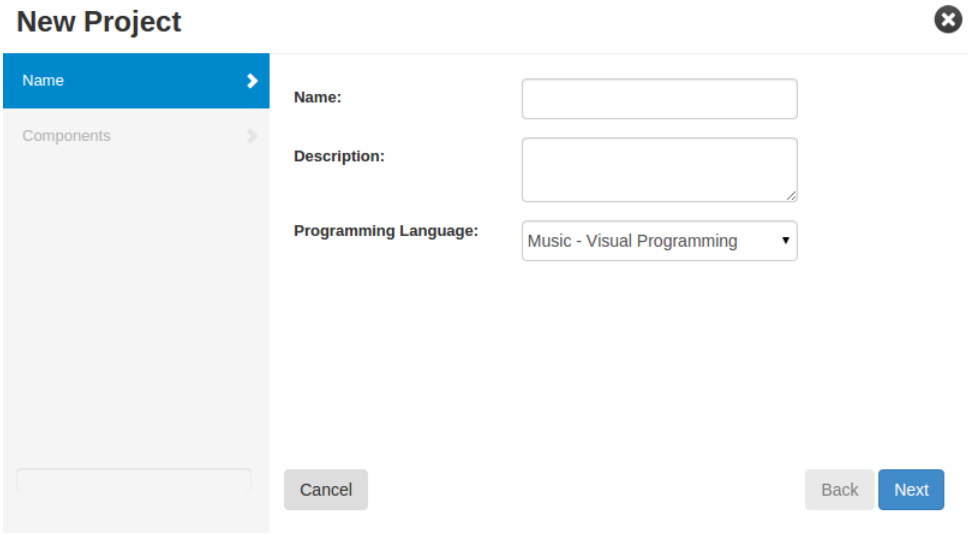


Figure 29: Create new project from example

Once created, you click on the new application's name to open it. A new window will open to display the example application (figure 30).

Figure 30: Radio station application

This application loads a radio stream from a certain address. The stream in stored as the *music* variable. The *Play audio stream* block starts to play the stream.

The application would start to play the stream and close so that you could not hear anything. In order to keep it running and be able to actually listen to radio, you used the *while* block.

The *Audio stream [] is playing* block returns true as long as the stream is still playing. For this case, where you have a radio stream, it should not stop playing. However, if you replace the current stream with one you generate from an audio file, the stream would stop playing when it reaches the end.

The *delay* block temporarily interrupts the execution of the program for the specified amount of time. Basically, you can state that the program checks every 1000 milliseconds that the stream is still playing. Once there is nothing more to play, the program finishes its execution.

Make your Raspberry Pi talk

Now that you saw how to use the Raspberry Pi as a radio, you will make it talk instead of just play music.

You need to disable all the block in the project. To do that you just right click on the block and select *Disable block*. Afterwards, you drag the *Say* block located in the *Multimedia* section. This allows us to make the Raspberry Pi talk. Finally, you write a text, select a language and run the project (figure 31).

Figure 31: Say block

Tips & Tricks

You can see at any time the Python or JavaScript code that gets generated as you drag and drop blocks. To do that, you just have to click *Show code*.

We recommend to do that as you work on your projects so that you get acquainted to other programming languages. After you get accustomed to Python or JavaScript, we recommend trying to create projects in that programming language.

You can get hints regarding the functions you should use while writing Python or JavaScript applications by dragging the block you would need and take a look at the generated code.

LED Blink

In this projects you are going to get accustomed with how the Raspberry Pi works by using the GPIO pins to make an LED blink.

What you need

- One Raspberry Pi connected to Wyliodrin ;
- One LED;
- One 200 Ohm resistor;
- Two male-female jumper wires.

The Setup

Connect the LED to the Raspberry Pi following the schematics in figure 32. You will see when you look at the LED that it has two legs. One is longer, that one is usually the anode. This one has to be connected to the GPIO pin of the Raspberry Pi. The shorter leg should to be connected to the resistor and then to the ground pin of the board.

Although the position of the resistor is not fixed, it can can either connect

the ground to the cathode or the anode to the GPIO pin, the cathode should be connected to the ground to obtain the usually desired behaviour. That means that we want the LED to ligh when the GPIO is set to HIGH and not to light when the GPIO is set to LOW. If you put the legs the other way around, the effect will be the opposite.

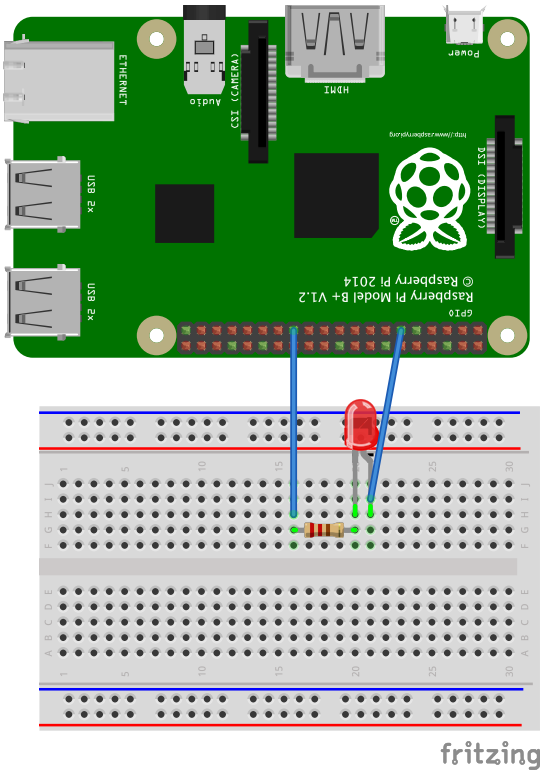


Figure 32: LED blink schematics

The Code

You go to the Wyliodrin Applications page and create a new application. you name it and select the for the programming language *Led Blink - Visual programming*. This will create an application that makes an LED connected to the pin 0 of the Raspberry Pi blink.

Once created, you click on the new application's name to open it. A new window will open to display the application (figure 33).

You can see that the *while* block is used together with *true* so that the application never stops running unless you stop it. The blocks placed inside the *while* block get executed forever.

You first turn on the LED that you connected on pin 0, you wait for 500 milliseconds then you turn off the LED, you wait again and so on. The result is that the LED keeps on lighting for half a second then being off for half a second. What you see, is an LED that blinks

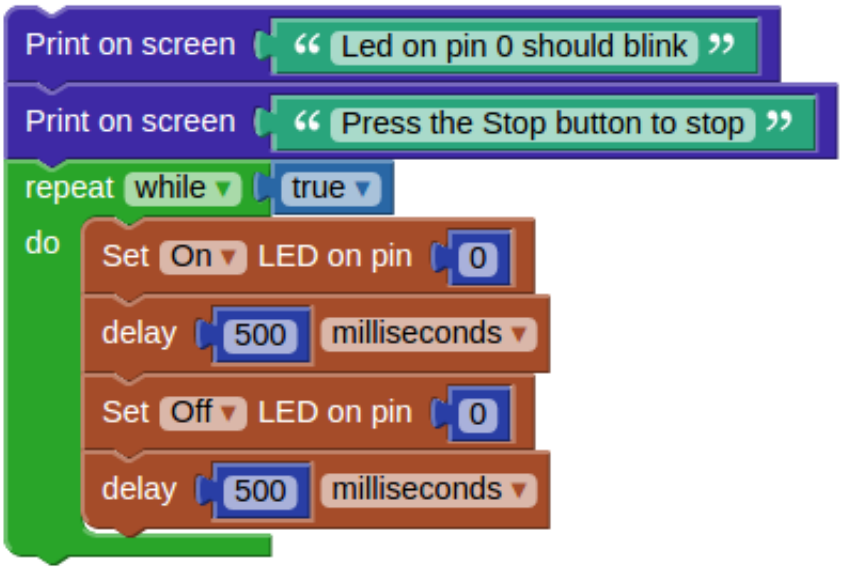


Figure 33: LED blink application

Run the application and the LED should start blinking.

You can also see the code generated from the blocks. To do that you just need to press the *Show code* button.

You can take a look at the Python code that makes the LEd blink.

Python

```
1  from wyliodrin import *
2  from time import *
3
4  pinMode (0, 1)
5
6  print('Led on pin 0 should blink')
7  print('Press the Stop button to stop')
8  while True:
9    digitalWrite (0, 1)
10   sleep ((500)/1000.0)
11   digitalWrite (0, 0)
12   sleep ((500)/1000.0)
```

First of all, you need to import the *wyliodrin* and *time* modules that will allows you to use some simple functions.

Firstly, you have to declare the pins you are goins to use and their purpose. In this case, pin 0 is used as *OUTPUT*. This is what *pinMode(0,1)* does (1=OUTPUT, 0=INPUT).

Next, there are the *print* functions that simply write the text in the shell.

Another useful function is *digitalWrite*. The function receives as parameter the pin and the value. As you are doing a digital write, the value can be either 0 or 1. If the value is 0, the pin will have no current. If the value is 1, there is current.

The *sleep* function makes the program wait for the desired number of seconds.

All this code is enclosed within the *while True* loop that makes it to be run forever, until the application is stopped.

Tips & Tricks

We said that usually the longer leg is the anode and the shorted one is the cathode. However, this is not standard, to make sure which is which, you should know that the cathode leg is always connected to the bigger part inside the LED.

SOS Morse Code Signaler

Let's learn the More Code!

This project's goal is to signal an S.O.S message using an LED. In order to make it, you will need to use what you have just learnt, which is the wiring of an LED.

Practically you will need the same circuit to connect the LED -see previous chapter-, but program it differently as to show the international Morse Code distress signal (Figure 34).

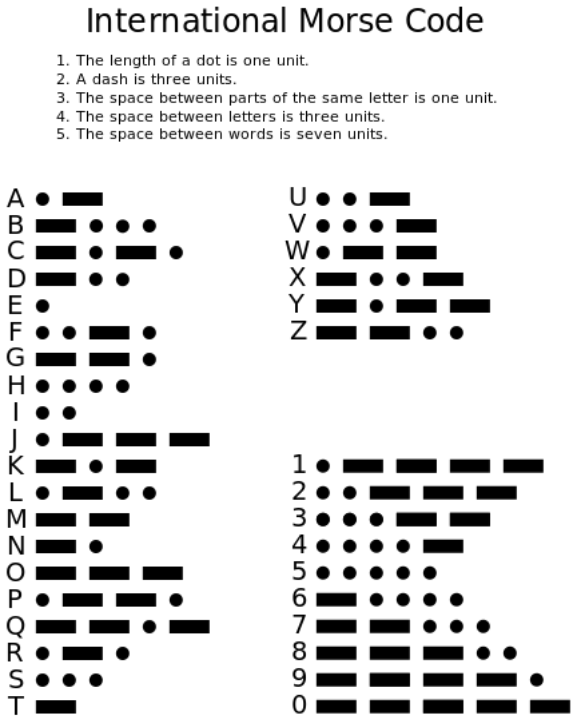


Figure 34: International Morse Code

What you need

You will use the same schematics as in the *LED Blink* chapter.

The Code

You go to the Wyliodrin Applications page and create a new application. You name it and select for the programming language New - Visual programming. Within the application put together the block as shown in Figure 35.

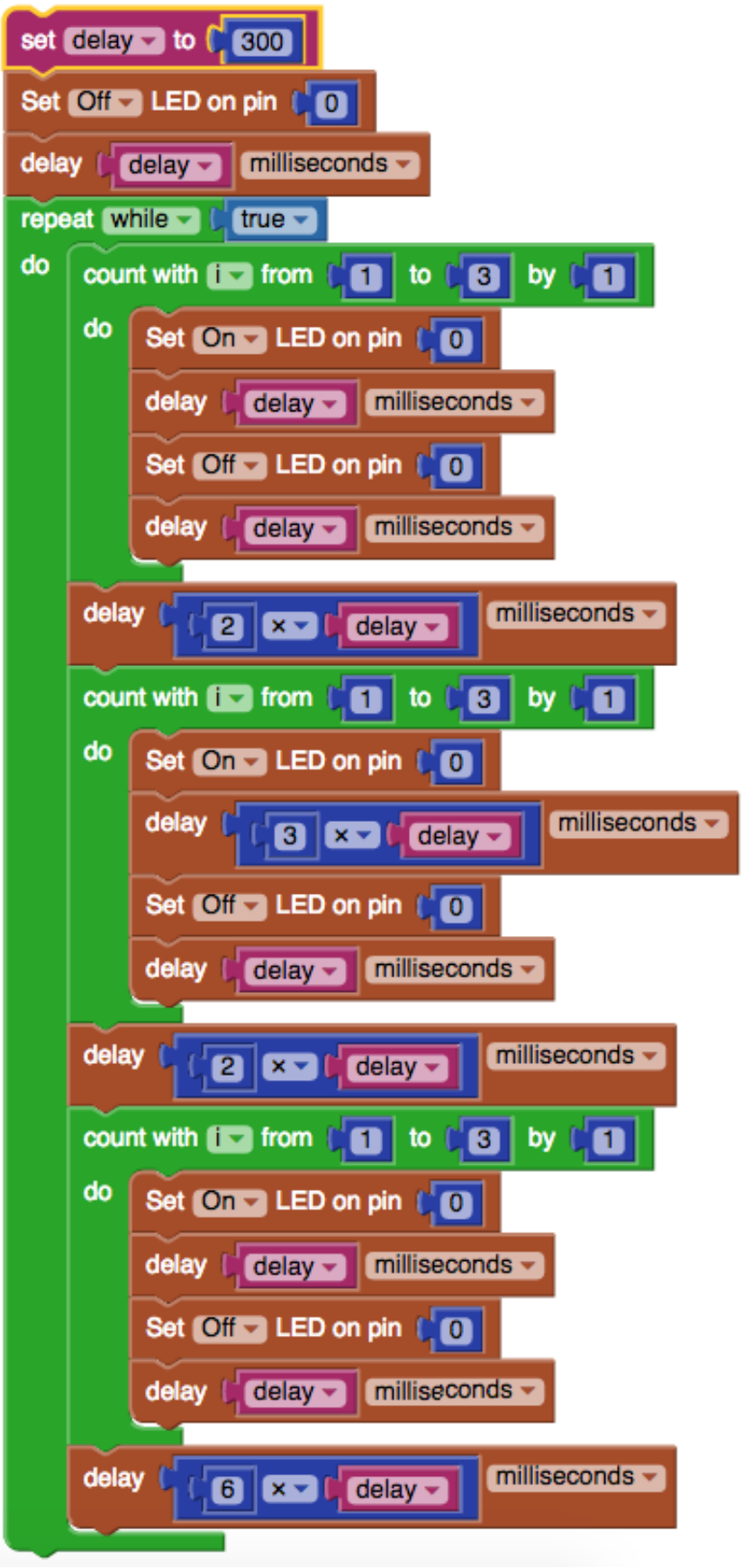


Figure 35: SOS Morse Code application

This may seem quite a hard task, thus to explain the layout of the blocks, let's split it into three parts.

First chunk's purpose is to set some prior conditions that will make the rest of the code easy to read, write, understand and function.

Here they are:

- "delay"- you need a pause of 0.3 seconds (300 milliseconds) between the blinks. The variable is set at the beginning and will be only called later;
- the LED should be off every time the sequence starts;
- you make sure that there is a pause before the signal starts, so there's no time for confusions.

The next two chunks are put inside a repeat-while-true loop which means the SOS is signaled as long as the project runs.

The second part consists of the letter S being lit up with the use of the LED. What you actually do is to repeat three times (the block count from 1 to 3) a short blink (LED on - LED off) with a pause between each blink of 0.3 seconds (delay).

Before you move on to the letter "O", you mark the end of the S signal by a delay twice longer.

The third piece of the block assembly is the "O" signal. This time, you have three longer blinks. You light up the LED, keep it that way for 0.9 seconds, turn it off, pause, and repeat twice.
Add again the letter S. It's important to use a block to include a delay that lasts more than any other, to emphasize that the "SOS" signal ended.

Python

```
1 from wyliodrin import *
2 from time import *
```

```
3
4  delay = None
5  i = None
6
7  pinMode (0, 1)
8  delay = 300
9  digitalWrite (0, 0)
10 sleep ((delay)/1000.0)
11 while True:
12   for i in range(1, 4):
13     digitalWrite (0, 1)
14     sleep ((delay)/1000.0)
15     digitalWrite (0, 0)
16     sleep ((delay)/1000.0)
17   sleep ((2 * delay)/1000.0)
18   for i in range(1, 4):
19     digitalWrite (0, 1)
20     sleep ((3 * delay)/1000.0)
21     digitalWrite (0, 0)
22     sleep ((delay)/1000.0)
23   sleep ((2 * delay)/1000.0)
24   for i in range(1, 4):
25     digitalWrite (0, 1)
26     sleep ((delay)/1000.0)
27     digitalWrite (0, 0)
28     sleep ((delay)/1000.0)
29   sleep ((6 * delay)/1000.0)
```

You may need the code in JavaScript or Python. To get it just click on the big, red *Show code* button.

Let's give the Python code some brief explanations.

It's vital to import the libraries that contain the functions which you are going to use: *wyliodrin* and *time*.

After that, you declare the variables "delay" and "i'.

The *pinMode* function that initiates the LED appears to tell the program that pin 0 is used as an output.

To set the LED off, you assign the value 0 to the pin where the LED is connected, in our case pin 0.

As explained in the application's description, the *while-true* loop will keep the led blinking for as long as the project is running. Within this loop we use three times the *for* function. It will repeat the sequence inside itself three times. The sequence is:

- write the value 1, which stands for on, on the pin 0;
- take a break of 300 (the delay's value) divided by 1000 - to have the result in milliseconds;
- write 0 (OFF) on the pin the LED is connected to.

You only need to use these simple steps to create the valid signal for SOS in Morse code.

Signal SOS with a Buzzer

What you need

- The previous schematics;
- One digital buzzer.

If you want to use your imagination and create a more complex project we make you a suggestion: use a buzzer.

The Setup

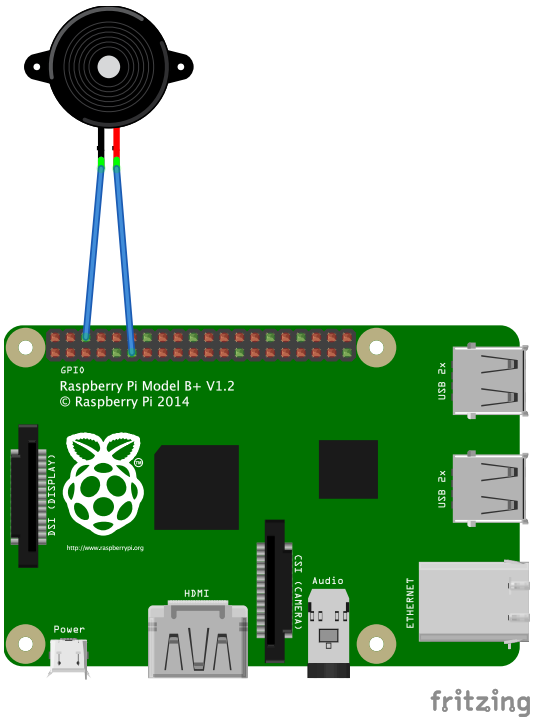


Figure 36: Buzzer connection schematics

To connect the buzzer, follow the schematics in Figure 36, which means you connect the ground pin of your buzzer to one of the GND pins of the board and the signal pin to one of the digital pins of the Raspberry pi.

Please take notice that there are only some buzzers that work on digital pins. Others need a PWM signal in order to function. In case you own such a buzzer, please read the following chapters in order to learn how to use the Arduino and the PWM pins and then connect your buzzer to such a

pin.

The Code

Concerning the way the blocks are assembled, as you can see in the Figure 37, all you need to add is the *on* and *off* on the digital pin to where the buzzer is connected (pin 1 in the example).

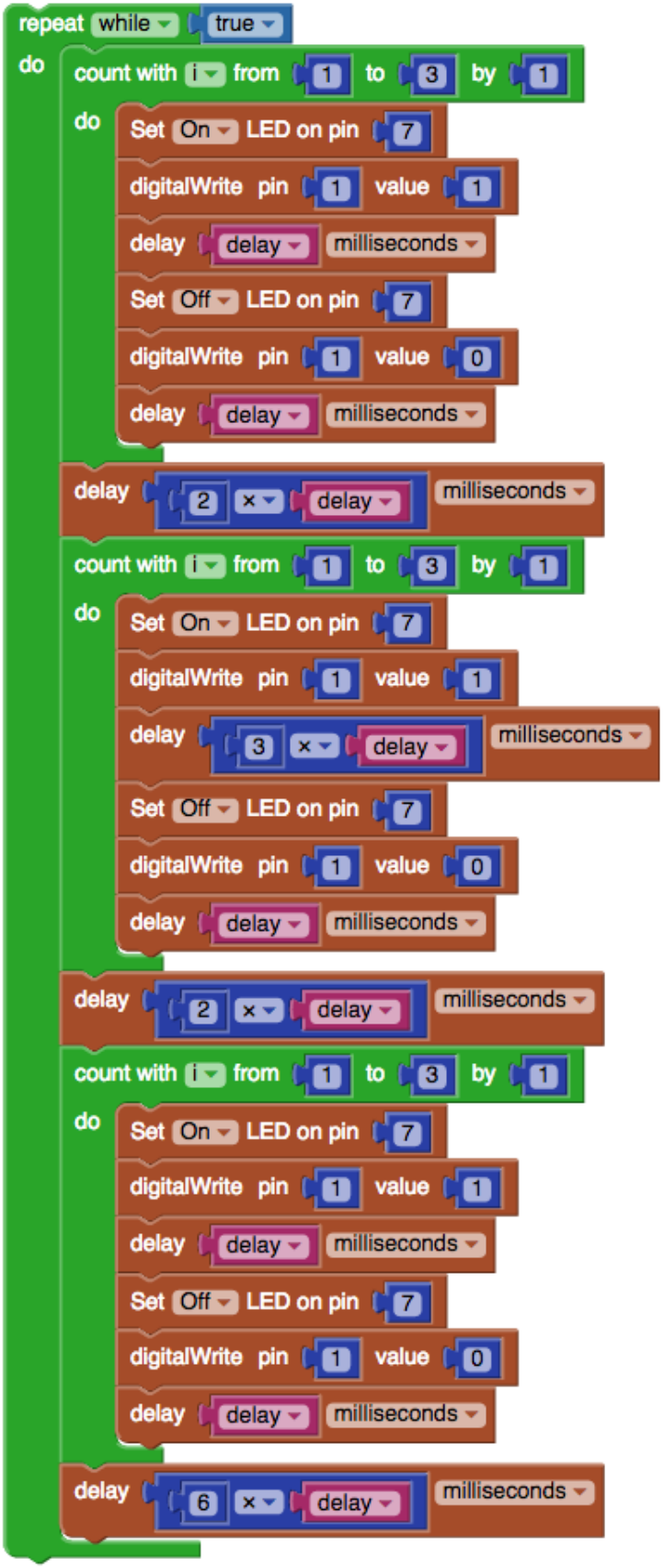


Figure 37: Buzzer SOS application

LED line

Ever dreamed of creating light games? This is what you will do next. And you can also make something useful out of it by signaling when your car is close to an obstacle in a really genuine way.

What this project is aiming is to easily create an LED chase effect and eventually control it with a potentiometer.

Simple LED Line

What you need

- One Raspberry Pi connected to Wyliodrin ;
- Five LEDs;
- Five 200 Ω Resistors;
- One potentiometer RM065;
- One breadboard;
- Jumper wires.

The Setup

To understand how the pieces should be put together, just follow the example in Figure 38

The setup is simple. Use the same way of connecting an LED as shown in chapter *LED Blink*.

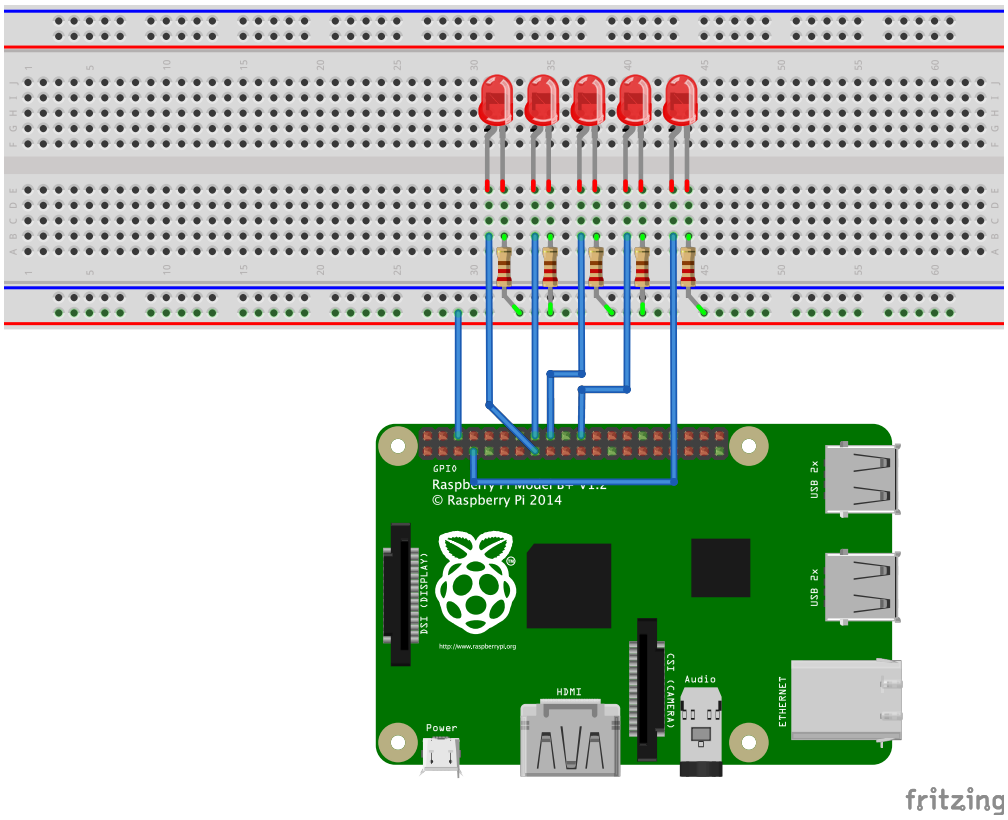


Figure 38: Connecting Leds

The best way to do it is to use only one ground pin, connect it to one of the lines of the breadboard and insert all the resistors on that line.

After the set up is done, you can go ahead and build the code.

The Code

The blocks used to build the Simple LED Line or the so called LED chase effect are ones that you may have become familiar with. Follow the assembling of blocks in Figure 39

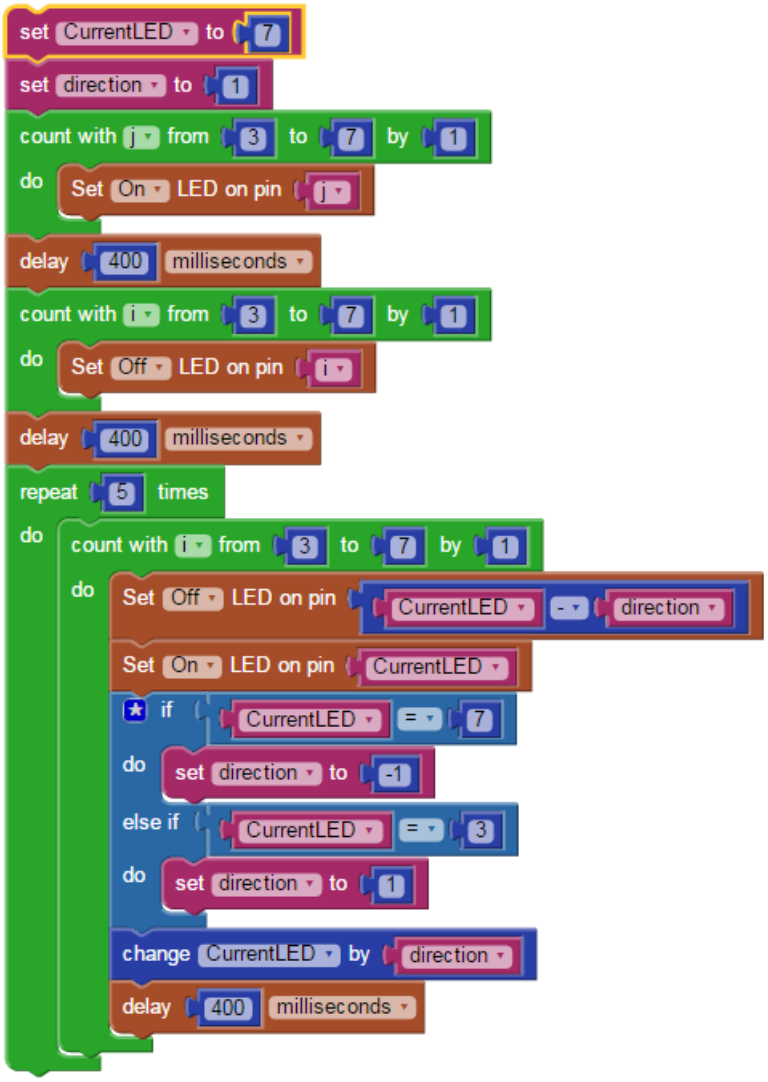


Figure 39: LED chase effect code

Here's what it represents: the first two blocks set the variables that you are going to use later.

In this set up you are going to work with pins 3 to 7. So you count the number of pins from 3 to 7 and turn on all the LEDs.

You continue by turning them all off. As you can see, it is advised to insert a pause between the two actions.

Next, a loop is inserted which goes on 5 times.

What you actually do is to turn off the previous LED and light up the one LED that your line reached, starting from the 7th pin.

Let's explain the flow, supposing the program has just started. You are at LED 7.

- Turn off LED 6.
- Turn on LED 7.
- Enter the *if* statement as your CurrentLED indeed equals 7. What it does is to make sure the direction will be in a descending order once it gets to the last LED.
- The *change* block modifies the value of your pin by -1 or 1. In this case, -1. Now the CurrentLED is 6.
- Stop for 0.4 seconds and start again
- This time it doesn't match any condition of *if*.

CurrentLED is not 3, nor 7. The loop will only turn off 7, turn on 6 and move on to 5.

You may need the code in JavaScript or Python. To get it just click on the big, red *Show code* button. You can observe that the Python code is also

included here.

Python

```
from wyliodrin import *
from time import *

CurrentLED = None
direction = None
j = None
i = None

CurrentLED = 7
direction = 1
for j in range(3, 8):
  pinMode (j, 1)
  digitalWrite (j, 1)
sleep ((400)/1000.0)
for i in range(3, 8):
  pinMode (i, 1)
  digitalWrite (i, 0)
sleep ((400)/1000.0)
for count in range(5):
  for i in range(3, 8):
    pinMode (CurrentLED - direction, 1)
    digitalWrite (CurrentLED - direction, 0)
    pinMode (CurrentLED, 1)
    digitalWrite (CurrentLED, 1)
    if CurrentLED == 7:
      direction = -1
    elif CurrentLED == 3:
      direction = 1
    CurrentLED = (CurrentLED if type(CurrentLED) in (int, float, long) \
    else 0) + direction
```

```
31      sleep ((400)/1000.0)
```

First and foremost the necessary libraries are imported. Afterwards the variables are declared. The ones that the have set values are defined.

Two *for* statements are used to turn on and off all the LEDs. Look up what *pinMode* and *digitalWrite* functions do in *Morse Code* chapter. Sleep is the function that initiates the delay in your program.

The loop you want is represented here by the *for* statement in which every line that is within will be executed 5 times (range(5)).

Now that you have the loop, the only thing you are in need of is to turn the LEDs on and off to create the pattern explained earlier. For this just implement the known functions with the variables already set and calculated as arguments.

When you change the value of *CurrentLED* what actually happens in Python is the following: you assign to your variable its current value to which you add the value of the *direction* variable. The latter is a number, so an initial verification of the former is required, as it has to be a number as well, or else the operation is impossible.

Add a potentiometer

What you need

- The same setup as before;
- One Arduino;
- One potentiometer;

- Male-male jumper wires.

The Setup

To make the project more entertaining, you can think of a way to control the speed of the succession of light. A suggestion would be to add a potentiometer to the set up (Figure 40).

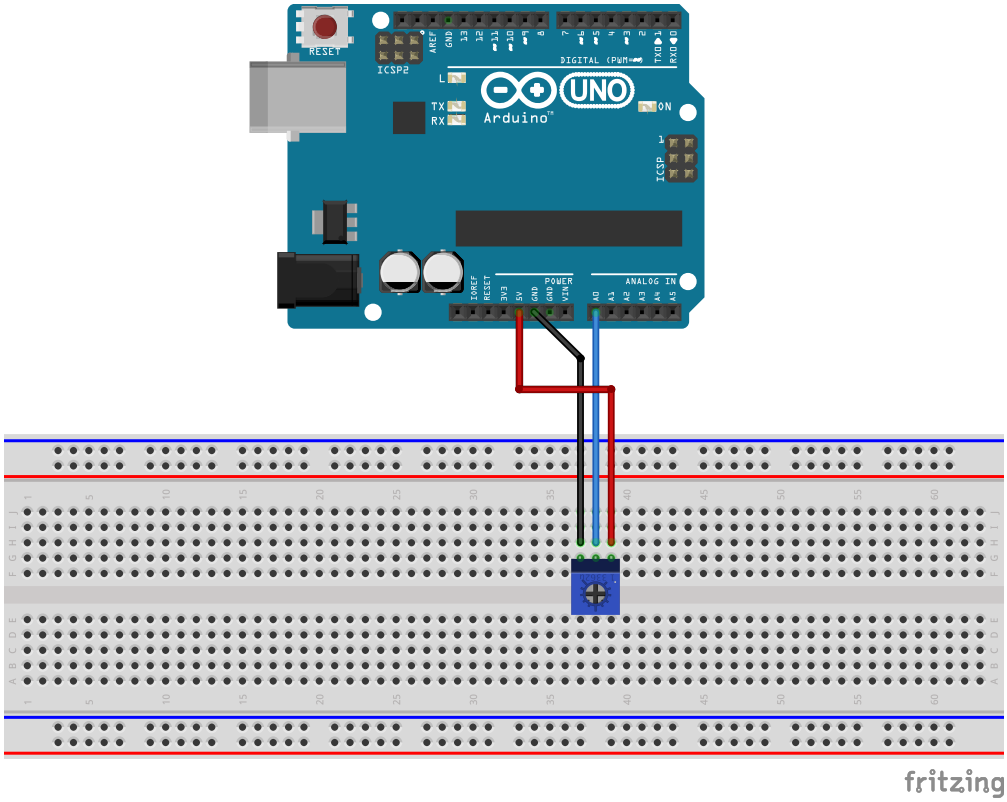


Figure 40: Connect a potentiometer

Keep in mind that after the initial part, consisting of the LED chase effect, you will add a potentiometer which uses an analog pin. Raspberry Pi boards don't have any, but they can be interconnected with an Arduino board to fulfill this function. Consequently, you will connect the Arduino board.

Arduino.ino file

Connect the boards with an USB cable, then create a new application. You will be asked to check the box that says "Connect an Arduino to the board". Now the application has, apart from the usual main, another file called Arduino.ino.

The *Arduino.ino* file uses the *Firmata* library. The library enables the communication between the Raspberry Pi and Arduino. Basically, the program running on the Arduino is the one described by *Arduino.ino*. That program waits for commands from the Raspberry Pi and executes them.

The protocol that makes the communication between the two boards possible is Firmata. This means that your Raspbery Pi will send commands to the Arduino. In return, the latter can deliver data to your main board, like the values read from the analog pins.

You can install it from your Arduino software or directly from your application. When you run the application, you will be asked to flash Arduino. Check the box and hit Save. Do this only the first time you use your Arduino.

This ".ino" file is the only one that runs on your Arduino board and as long as it is not replaced, it will remain the one running event if you power off the Arduino and power it up again. This is why you don't have to flash your board again even if you change the code. The code that gets changed is the one running on the Raspberry Pi (the one that gives commands and gets data from the Arduino), not the one on the Arduino.

Now you can do whatever project you want to with your Arduino and Raspberry.

Tips & Tricks

In case your projects resets when you use your Arduino alongside the Raspberry Pi, then the USB cable power supply might not be enough. The solution is to pug it in separately to a power source between 7V and 12V.

Pay close attention to the blocks you use when you involve your Arduino board in the circuit. There are different blocks, dedicated to Arduino.

The code

Look at the layout of the blocks, compared with the simple LED line (Figure 41).

The first difference is that the block which initiates the Arduino on the port. That usually is */dev/ttyACM0*. In case this does not work, you have to search for it in the */dev* directory on the Raspberry Pi. You can also type *dmesg* in the board's shell and you will see the port the Arduino connected to.

Another change is the block you use to read the value from the analog pin. You will find the right one in the block list, in the *Embedded* category, search for *Arduino* and select your *Analog read pin* block.

One major addition to this code is how the delay inside the loop, which is actually the time elapsed between two consecutive LED blinks, is given by the value registered by your potentiometer.

Thus, depending on how much you rotate the potentiometer, the LEDs' blinks will follow one another faster or slower.

If you want to see more clearly the variations introduced by the potentiometer, you can choose to add a chart from your dashboard. For details on how to do this, check the chapter *Introduction to Wyliodrin* . An example is given

in the *LED controlled from Dashboard* chapter.

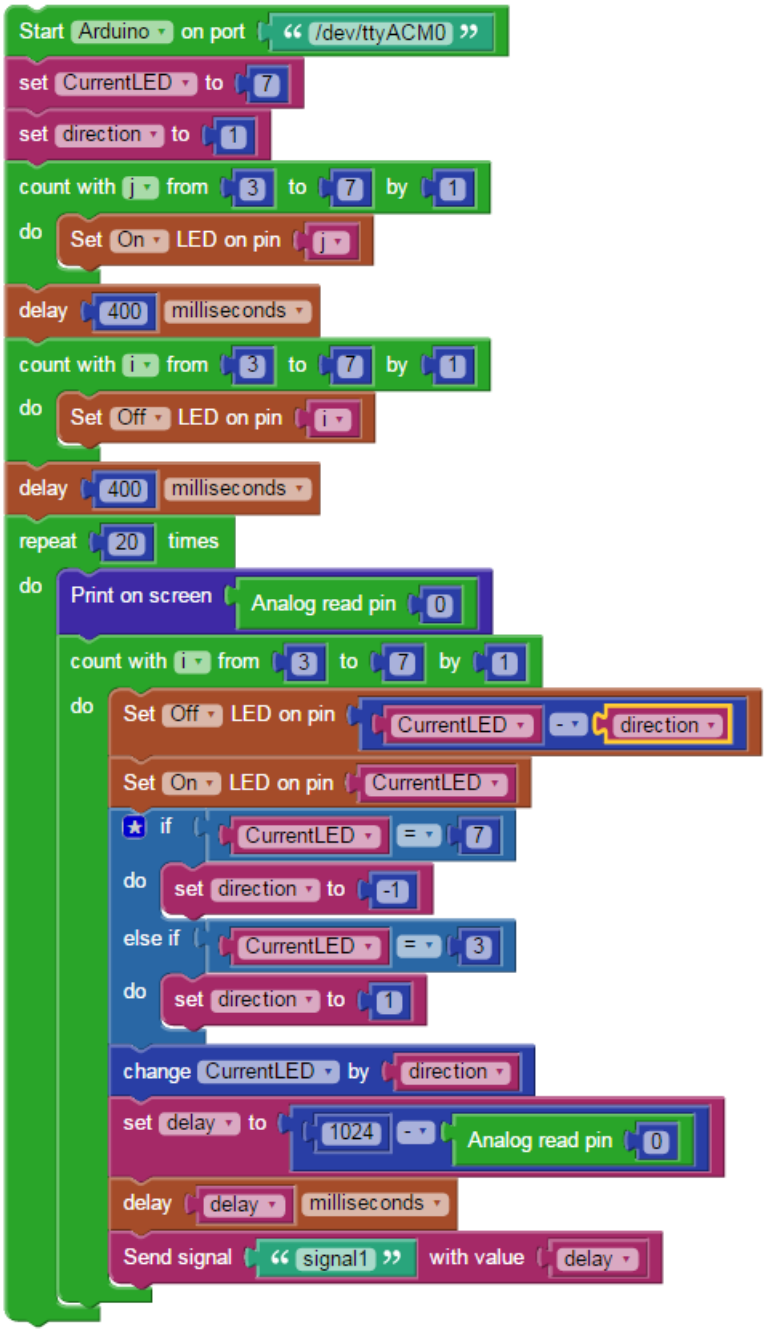


Figure 41: Code for the LED line with potentiometer

After you've chosen from the Dashboard menu the kind of chart you find most suitable, make sure that its name matches the *signal* field of the block that will send the values to the chart. Here, it is "signal1" but you can change it into any word you prefer.

You can see here what your chart should look like before running the program: Figure 42

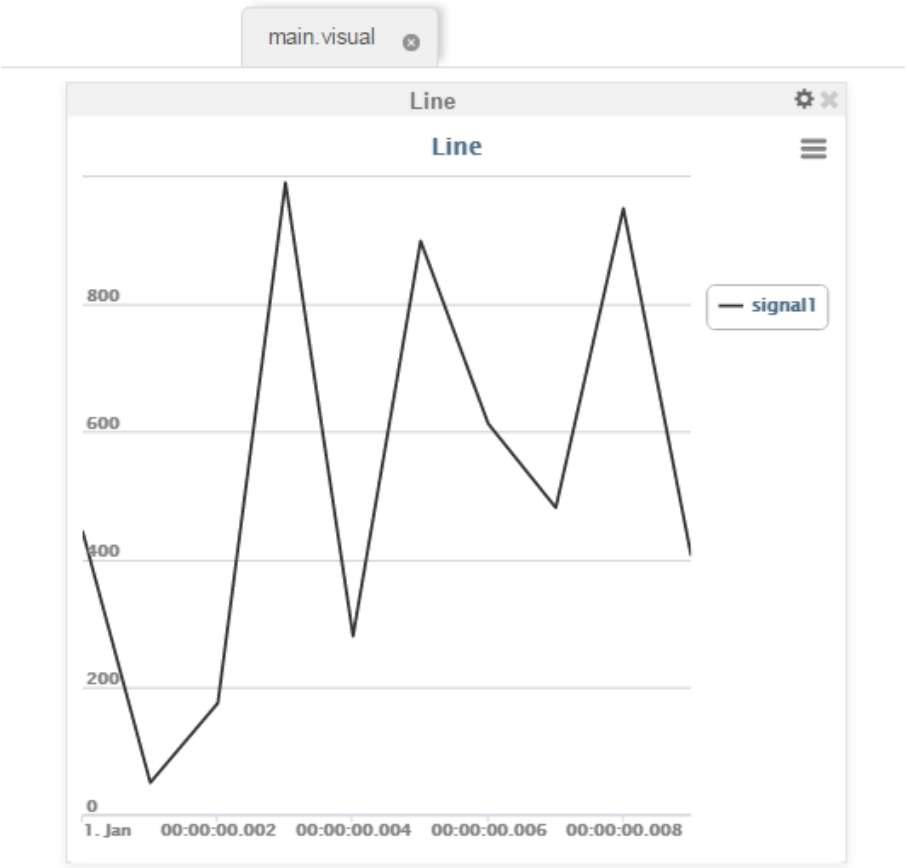


Figure 42: Chart with potentiometer's variations in time

The same changes can be noticed in the Python code as well, along with some others given by the Arduino supporting functions.

Python

```
from pyfirmata import Arduino
from pyfirmata import util
from wyliodrin import *
from time import *

CurrentLED = None
direction = None
j = None
i = None
delay = None

def setBoard(boardType, port):
  if boardType == 'arduino':
    board = Arduino(port)
  else:
    board = ArduinoMega(port)
  return board
board=setBoard('arduino', '/dev/ttyACM0')
reader = util.Iterator(board)
reader.start()

pin_var = board.get_pin("a:0:i")


CurrentLED = 7
direction = 1
for j in range(3, 8):
  pinMode (j, 1)
  digitalWrite (j, 1)
sleep ((400)/1000.0)
for i in range(3, 8):
  pinMode (i, 1)
  digitalWrite (i, 0)
```

```
34 sleep ((400)/1000.0)
35 for count in range(20):
36   print(round((pin_var.read() or 0) * 1023))
37   for i in range(3, 8):
38     pinMode (CurrentLED - direction, 1)
39     digitalWrite (CurrentLED - direction, 0)
40     pinMode (CurrentLED, 1)
41     digitalWrite (CurrentLED, 1)
42     if CurrentLED == 7:
43       direction = -1
44     elif CurrentLED == 3:
45       direction = 1
46     CurrentLED = (CurrentLED if type(CurrentLED) in (int, float, long) \
47     else 0) + direction
48     delay = 1024 - (round((pin_var.read() or 0) * 1023))
49     sleep ((delay)/1000.0)
50     sendSignal('signal1', delay)
```

Basically, the Arduino functions can't work without including the two libraries *Arduino* and *util*. More than this, you will have to initiate your board with the *setBoard* function to which you give as arguments the board type *arduino* and the port */dev/ttyACM0*.

You will next start an iterator thread which will deal with the pins. The thread aims at preventing the communication buffer to overflow. The communication buffer is used by the two boards to communicate, as stated earlier, the boards send messages one to another.

You know you are going to use the pin A0 and so it becomes much easier to access it through a variable *pin_var* via the *get_pin()* method. The method's argument, *"a:0:i"* is a string consisting of the type of pin (analog), its number (0) and the direction (input).

Next, there is the code that lights up the LEDs one by one. You should

already be familiar with the functions used. The only new code snippet is *pin_var.read()*. The read function is applied to the analog pin and it returns the value read from the sensor connected to it.

Finally, you have the *sendSignal* function that sends signals to the dashboard so that you can display data on the graphs.

Parking Sensor

Starting with what you have just created you can easily make a parking sensor

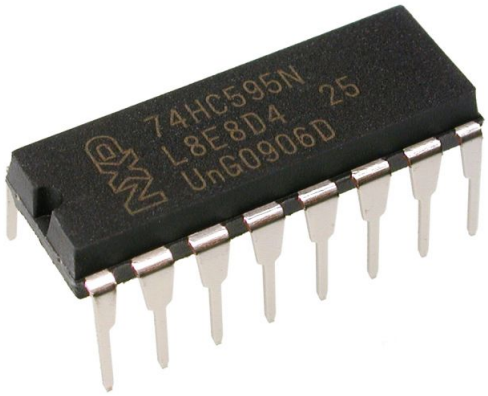
What you need

1. One Shift register - 74HC595;
2. One breadboard;
3. One Distance sensor Sharp GP2D120XJ00F;
4. Five 220 Ω Resistors;
5. Five LEDs;
6. Jumper wires.

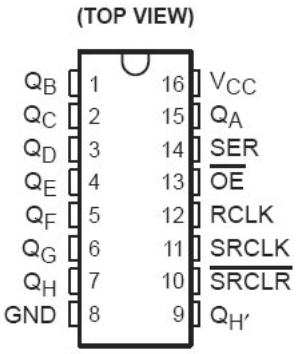
The Setup

When you want to use more LEDs it may occur that the pins you have are not enough. In this case, there is such a thing as a Shift register - 74HC595. This piece will only take out 3 pins of your board and give you access to 8

of its own instead. Otherwise said, you can use 8 LEDs connected to the shift register which will need 3 pins on the Raspberry Pi. You don't have to use all 8 pins of the 74HC595 and you can link two pieces to connect more LEDs.



(a) Shift register - 74HC595 [2]



(b) 74HC595 Pinning [3]

Figure 43

To get an idea on the layout of the pins, look at the Figure 43b.

What the shift register does is to shift out a byte of data one bit at a time. To see the set up of the circuit, you have the scheme in Figure 44

[3] https://content.solarbotics.com/products/photos/df309280702fe1b526588e932c7ccfa1/lrg/74hc595n-dscn3898.JPG

[3] http://www.octamex.de/shop/images/shop/product/b46584f19c5b36fbce5f2fb06041d594.jpg

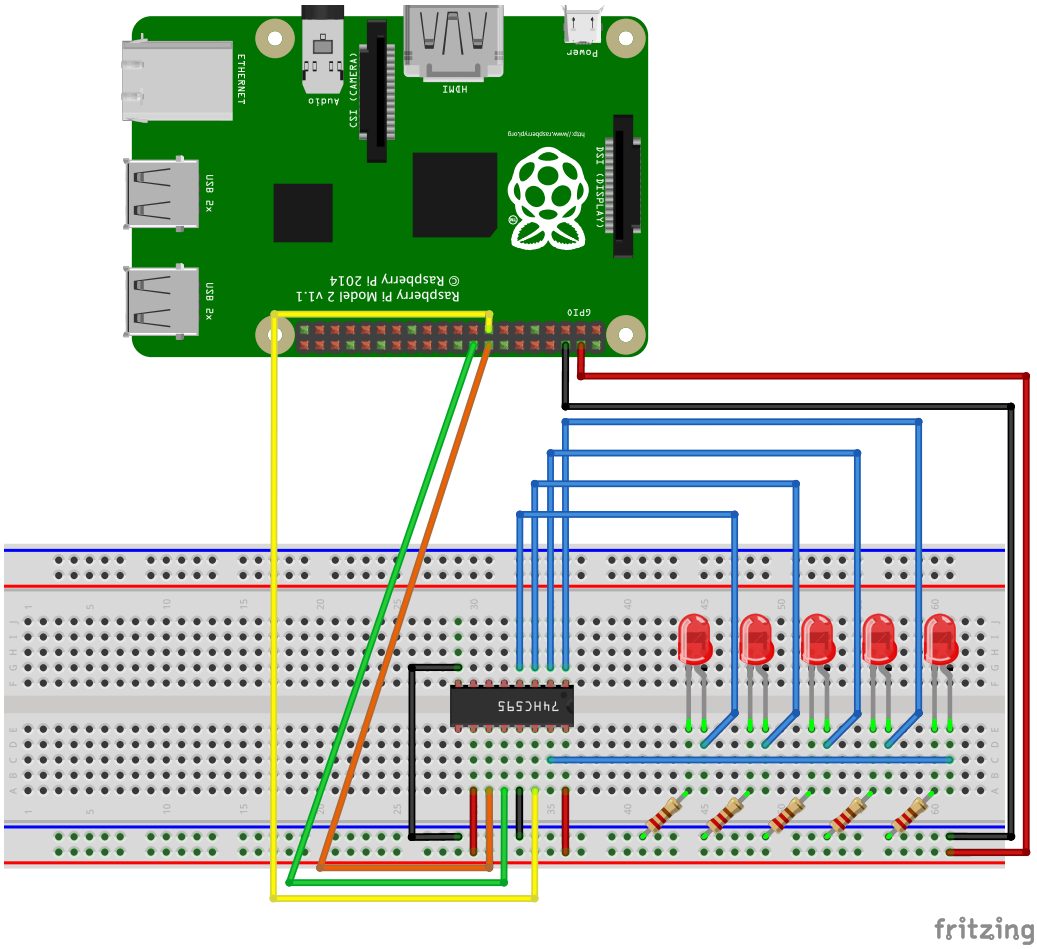


Figure 44: Set up for LEDs with 74HC595

The code

First and foremost define the method *get data*. This method will set the variable *data* with the value calculated as 2 at the power of its parameter minus one. Next, you will see why.

The function which controls your assembly of LEDs is the *Shift out* one. Here's what it does in a few simple steps.

- It sends to the shift register binary data in the form of a byte. So you can picture this: you have a byte that at first is 00000000. Each bit of this byte is assigned to one of the shift register's pins.

- Let's assume you want to light up the first LED, the one on pin *QA* in Figure 43b. The first pin is controlled by the last 0 in your byte, resulting in this: 00000001 which equals 2 at the power of 1 minus 1.

- To light up the first LED only, you give the value 1 to the variable LED which has the role of a parameter for *get data*. To light up the second one also you will need 00000011, so LED will have to be 2.

Back to your blocks in Figure 45. First you have all the LEDs off, so LED is assigned the value 0.

Then you have to start the clock of the shift register by setting the RCLK, on pin 5 to *LOW*.

The following block is the *Shift out* function with the initial value 0 as parameter. Nothing shall happen. Stop the clock by setting RCLK to *HIGH*. Nothing will happen until you rise the clock from low to high, that is from 0 to 1.

Move on to the next LED by increasing the variable's value by 1. This will have as consequence that at the second repetition out of the 6, the first LED will be on. At the third repetition you will have the last two LEDs alight and so on.

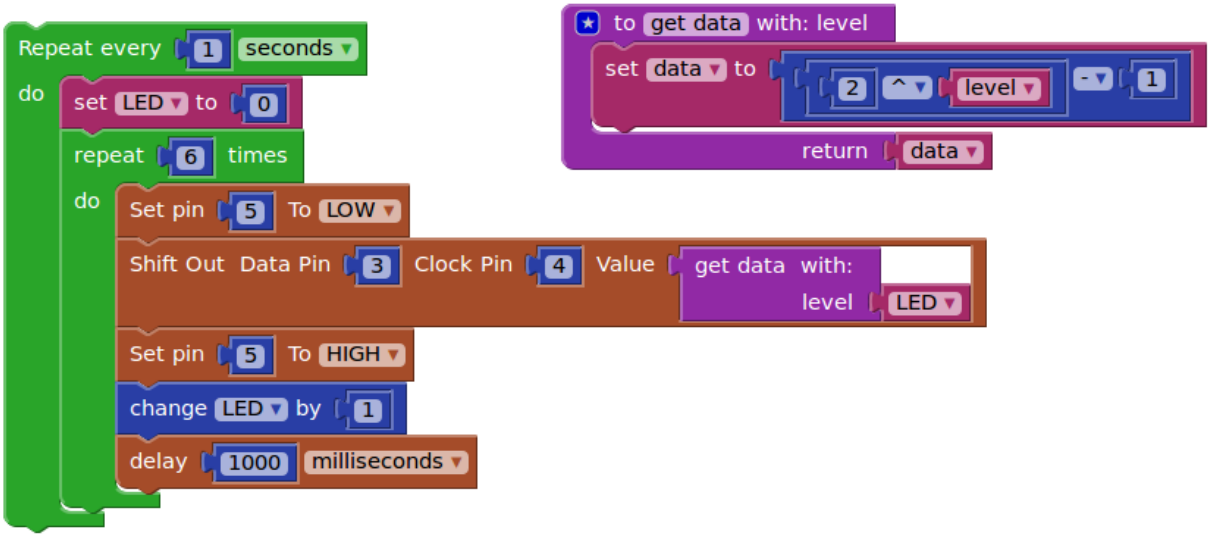


Figure 45: Shift out in Visual Programming

As far as the Python code is concerned, it follows the same train of thoughts. After you import all that is needed, set the pin as output and define the *get data* method *level* you proceed to your loops.

What's new in this code is this: *shift out* has as parameters the two pins for data and clock, the order in which it reads the binary number (MSBFIRST - most significant bit first / LSFIRST - last significant bit first) and the value it sends, in our case the return value of *get data* with LED as parameter.

All the other functions in this code are explained in detail throughout previous chapters.

When all the commands inside the big loop end, one second passes before it restarts.

Python

```
1 from wyliodrin import *
2 from time import *
3 from threading import Timer
```

```
4
5  level = None
6  LED = None
7  data = None
8
9  pinMode (5, 1)
10 pinMode (3, 1)
11 pinMode (4, 1)
12
13 def get_data(level):
14   global data
15   data = 2 ** level - 1
16   return data
17
18 def loopCode():
19   global LED
20   LED = 0
21   for count in range(6):
22     digitalWrite (5, 0)
23     shiftOut (3, 4, MSBFIRST, get_data(LED))
24     digitalWrite (5, 1)
25     LED = (LED if type(LED) in (int, float, long) else 0) + 1
26     sleep ((1000)/1000.0)
27   Timer(1, loopCode).start()
28 loopCode()
```

Now let's put this setup at use by replacing the rotary angle with a distance sensor. This way the frequency at which the LEDs light up signal if there is something near it or not. You can think of it as a parking sensor.

The Setup

Figure 46 shows how you should connect the sensor. As it is an analogical one, you can easily observe that you need to use the Arduino together with the Raspberry Pi. You should already be accustomed to how the Arduino interacts with the Raspberry Pi so we step directly to the code.

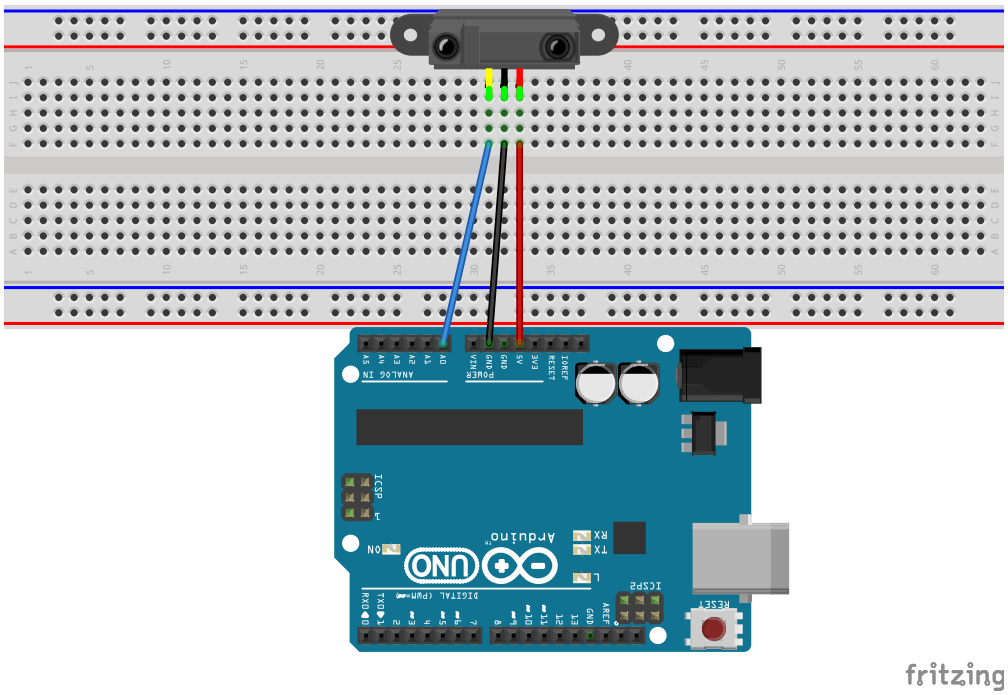


Figure 46: Distance sensor schematics

The Code

You can see the block you need in order to get the project running Figure 47.

First of all, you need to get the distance the obstacle is situated at. That translates in reading an analog value from the pin the sensor is connected

to.

The next step is to translate that value to the number of LEDs you need to light up. In order to achieve this, you have to define one extra function: *get level with* (Figure 48). The function receives the value obtained from the distance sensor and returns a number ranging from 0 to 4. The number tells you how many LEDs need to be turned on. Further on, the number is passed to the *get data* function. This is the same function as the one previously used. The function returns the value that has to be passed on to the shift register.

The rest of the code is the same. Also, the Python code should be of no novelty to you.

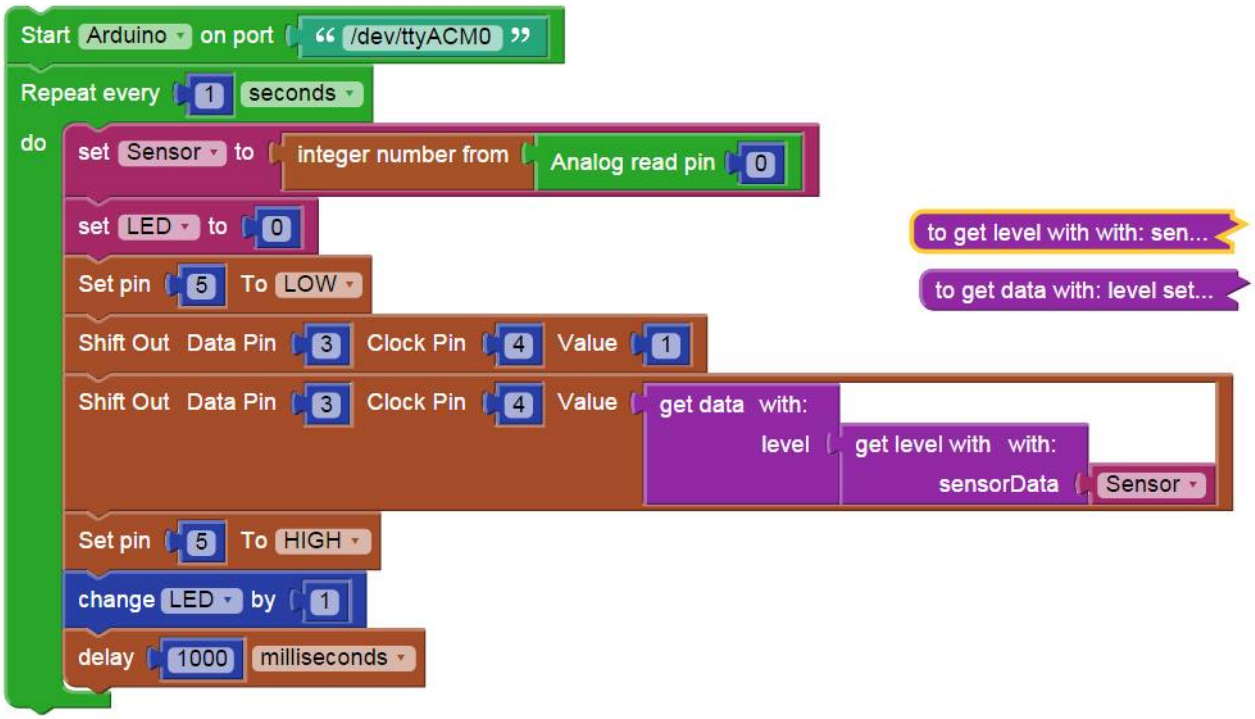


Figure 47: Distance sensor blocks code

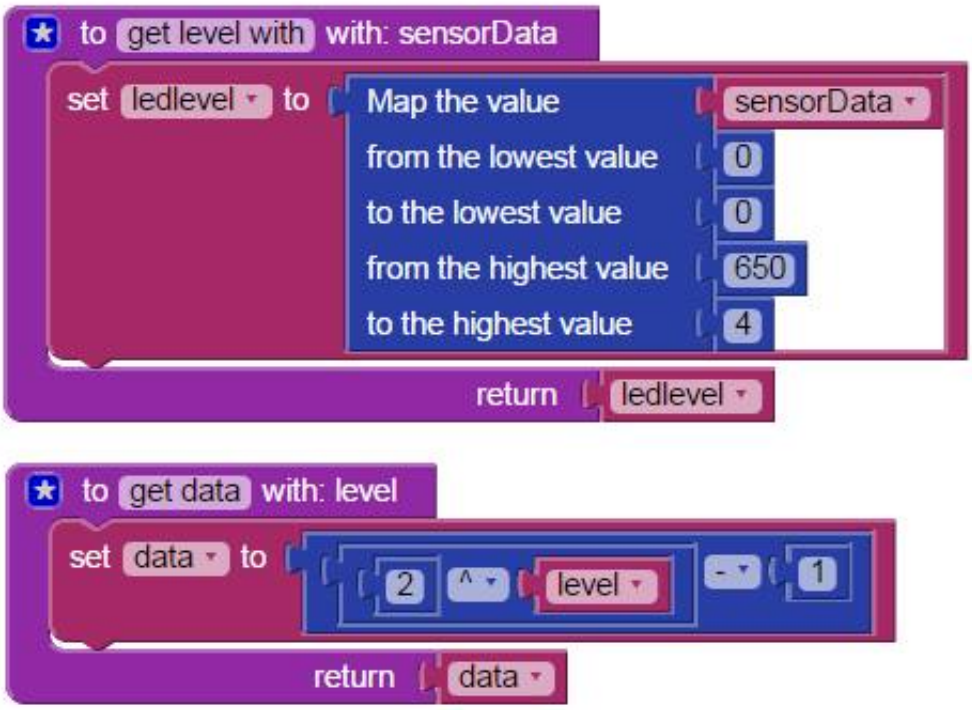


Figure 48: Distance scaling functions

Control LED from Dashboard

Here's a great feature that will allow you to make even simpler but a lot more interactive projects using your boards and the Wyliodrin platform.

What can you do? You can light up the lamp you have at home remotely, directly from the browser. By using this project you will not worry again that all the lights at home are off and that your home is in danger.

Let's try and take a different approach to the classical LED blink example. You will add a button to your dashboard through which you will control your LED. In this example you will remotely control the LED directly from the browser.

What you need

You will use the same set up as the one in the *LED Blink* chapter.

The Code

The first step in building this project, after you have created your new Visual Programming application, is to go to the Dashboard. There you have choose

the the element you wish to use, the switch in our case (figure 49)

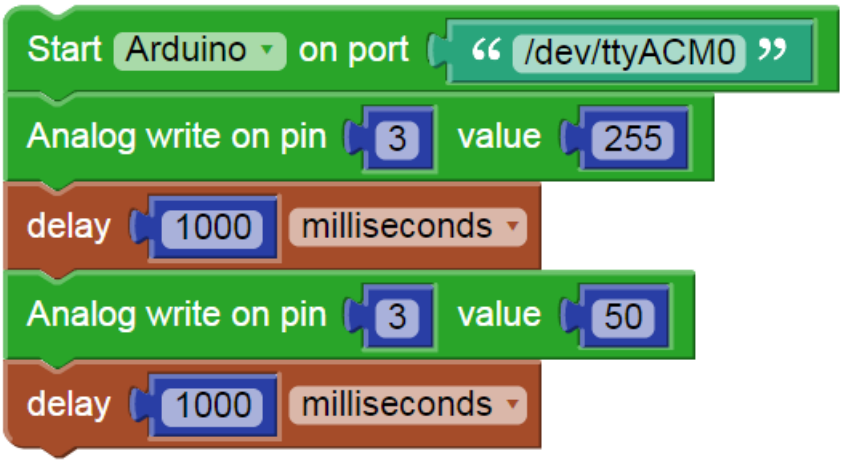


Figure 49: Dashboard view

You will see that the default name for your button is *signal1* . If you want a different name, just select the settings menu and rename it. What you insert there will be the name you will use in your code. For this example we have chosen *switch* for the signal name.

The dashboard button works in a similar way to the graphs used in the previous chapters. However, in this case, the signal is sent from the dashboard to the application. Everytime the button changes its state it sends a signal to the application.

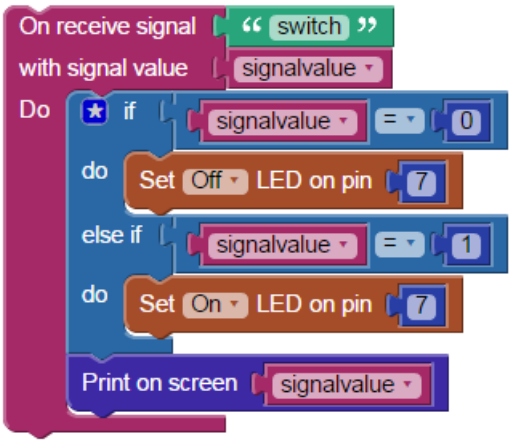


Figure 50: LED controlled by a switch

Figure 50 depicts the blocks you need to use in order to obtain an LED that lights up when the switch is on and lights off when the switch is off.

The most important thing to do is to choose from *Signals* the block that receives the value given by your switch. This block connects your application to the dashboard by making it subscribe to signals to be received. Basically, each time a signal is received the code inside this block gets executed. You can see that the *On receive signal* block also has two parameters: *switch* and *signalvalue*. Switch is of no difficulty, it represents the name of the signal the application subscribes to. *Signalvalue*, on the other hand, might represent a novelty. As you can imply from the block's looks, it is a variable. However, there is no place in this code where this variable gets a value. Why is that? Because the variable changes its value each time a new signal is received. The dashboard has to send the application a value and this is how they interact: through variable. You can think of this variable as of a container. The dashboard places the value there, then it notifies the application and the application gets the value from the container. As a result, you already have a value stored in *signalvalue* and you can simply start using it.

What you want to do is to verify if your switch is turned to *On*, in which case the value you receive through *signalvalue* is 1 and the LED should be

alight. The opposite action will happen if the switch is *off*.

You can mix dashboard elements to make it even more interesting. Let's add one that will associate a photo of a lit up LED to the actual LED turned on and the other way around.

If you scroll through the Dashboard menu you will find the *custom* element. Select it and then go on its settings menu. In the square text box type the URL link of an image, press Enter and on the next line put the second image's URL (figure 98).

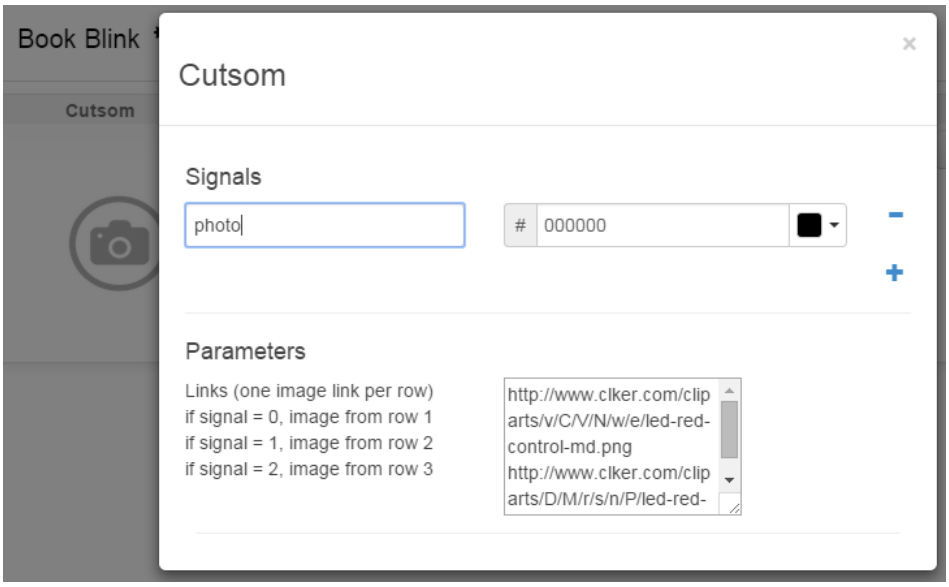


Figure 51: Custom settings menu

The logic behind this is that if the element receives a signal of value 0, it will use the first image that you inserted as a link, if it receives the value 1, it picks up the second image and so on.

As for the code, just add from the *Signals* category the block that sends the *photo* signal with the desired value. In this case the values will be 1 or 0, which stand for the LED on or off, as previously established.

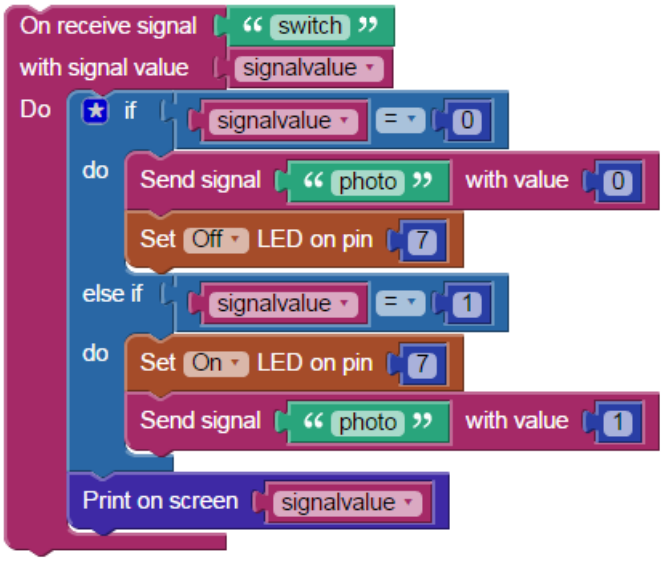


Figure 52: Switch and custom widgets

Python

```
from wyliodrin import *
import json

signalvalue = None
pinMode (7, 1)

def myFunction(__sender, __channel, __error, __message):
  global signalvalue
  signalvalue = int(json.loads(__message))
  if signalvalue == 0:
    sendSignal('photo', 0)
    digitalWrite (7, 0)
  elif signalvalue == 1:
    digitalWrite (7, 1)
    sendSignal('photo', 1)
  print(signalvalue)

```

```
18 openConnection("signal:" + 'switch', myFunction)
```

The same train of thoughts applies to the Python code as well.

The chosen variable is *signalvalue*, then you declare the pin 7 as an output with the use of function *pinMode*.

Next, you can see the declaration of the function called *myFunction* that receives four parameters. Afterwards, you set the value for the variable to the message received from your switch. What's left is to set the LED on and off according to the position of the switch and send the according value with the right signal name.

The four arguments of this function have each a separate role:

- *sender* is the ID of the board sending the message, this way you can ignore any unwanted messages;
- *channel* is actually the label marking the communication channel;
- *error* is a variable that tells you if the message was properly sent or not; in case this error is anything but false, the message becomes invalid;
- *message* is the one saved in the *signalValue* variable; it represents the value you receive through the signal, from the dashboard.

In line 18 there is a function *openConnection*. As explained in the blocks section above, the communication between the dashboard and the application is done by making the latter subscribe to signals to be received. This is exactly what this function does. The function receives two parameters, the name of the channel, which consists of the prefix *signal* followed by the label of the communication channel. The other parameter is a function that receives the parameters mentioned above. This function is called each time a new signal is received.

Make a pulsating LED

You may think that an LED's only feature is to either be alight or off. However, there is one more trick you can use it for. That would be making it light brighter or dimmer by using a special pin which can receive different values, thus giving you access to this behaviour.

What you need

- One Raspberry Pi connected to Wyliodrin ;
- One Arduino;
- One LED;
- One 220 Ohm resistor;
- Jumper wires.

The Setup

The exact same way of connecting the LED as before is implemented now, only this time you must use a PWM pin, which is marked by a tilde.

You might not know what a PWM (Pulse Width Modulation) does. It is basically a digital way of control, which means it oscillates between the values 1 and 0.

First of all, you should know that the Arduino has a clock that counts quanta of time. You can decide to split this quanta in two different parts: one during which you will send the value 0 and the other in which you send the value 1. Essentially, what the PWM brings in addition is a counter that memorizes the

percent of the quanta during which the output value was 1. Figure 53

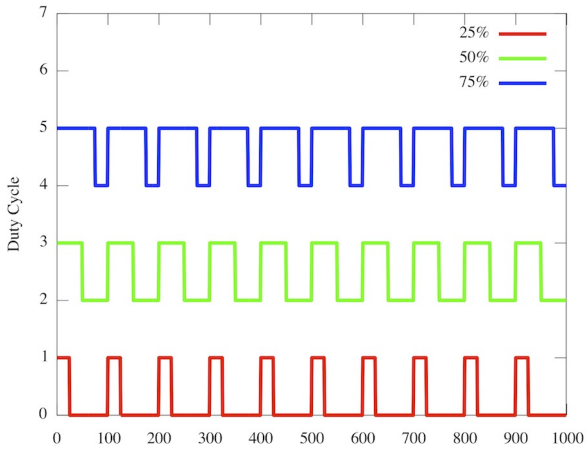


Figure 53: PWM graphics

The period while the signal has the value 1 will be the pulse width.

In a nut shell, you simulate an analog pin by changing the pulse width. The value of the PWM increases when you send more values that equal 1. However, this is not at all equivalent to having an output of 2.5 volts, for instance, instead of 5. This is why the PWM cannot be used with peripherals that require less than 5 V.

However, the PWM works perfectly for the LED. What's important to know is that you basically have an LED that changes its light intensity from bright to none so fast that the human eye can't perceive the change, so you and I see it as a continuous light but dimmer or brighter. This is due to the ability of human eye to integrate these values.

In order to control those pins you must use the *analogWrite* function. The values you write on the pin must range from 0 to 255, 0 being the equivalent of digital 0 and 255 the equivalent of digital 1. The percentage of the time it takes to skip from 0 to 1 is the value divided by 255 so we must create a mapping for those values.

The Code

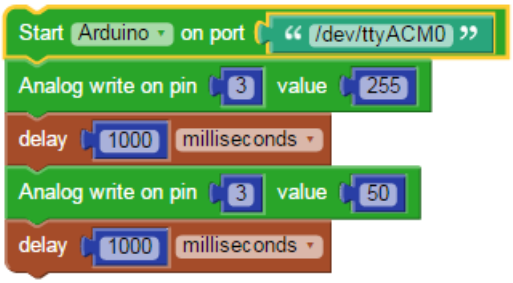


Figure 54: Different light intensities

You can see in Figure 54 that an Arduino is demanded to complete this task. The Rapberry Pi does have PWMs, but they are software PWMs. To make sure your project is a success, use Arduino PWMs. How and why you need to connect an Arduino is described in the previous chapters such as the *LED Line* chapter.

Initiate your board and use Arduino special blocks that write on analog pins. You have to write a certain value on the a PWM pin of your choice. Modify the value you write several times to see the difference.

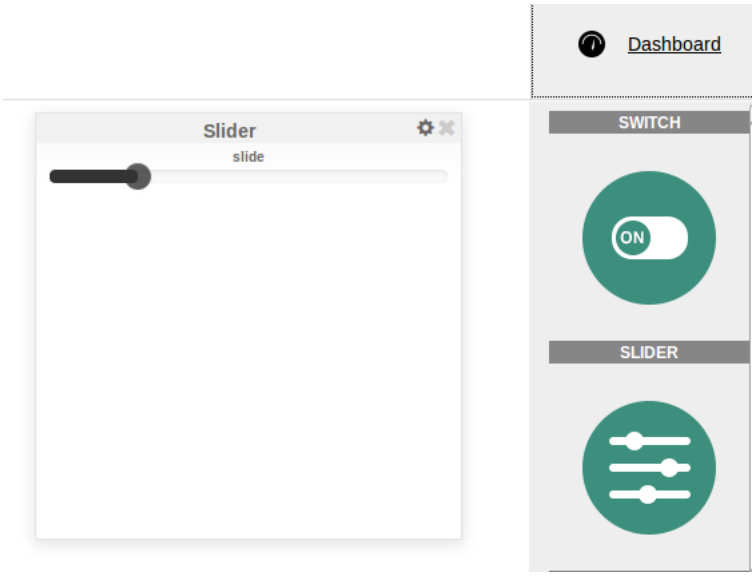


Figure 55: Different light intensities

Now you will want a simpler, quicker way of controlling it. For that go to dashboard and add a slider (Figure 55). Give it any name you want in the settings menu and also, choose the maximum value it can generate. The next step is to connect the slider to your code (Figure 56). Use the same chunk of blocks you used for the switch and complete it with the Arduino's Analog Write block.

You have a range of values you can receive from the slider, so you cannot just add one inside the block. You already know that the Arduino can generate a PWM signal that ranges from 0 to 255. As a result, these are the values you need to map the signal to.

In *Program/Numbers and Maths* section you will find the mapping block. First specify the value you want to map, which in our case is the *signalvalue*. The minimum value is 0 and the maximum one goes from the one that you had just set in the slider's settings to 255.

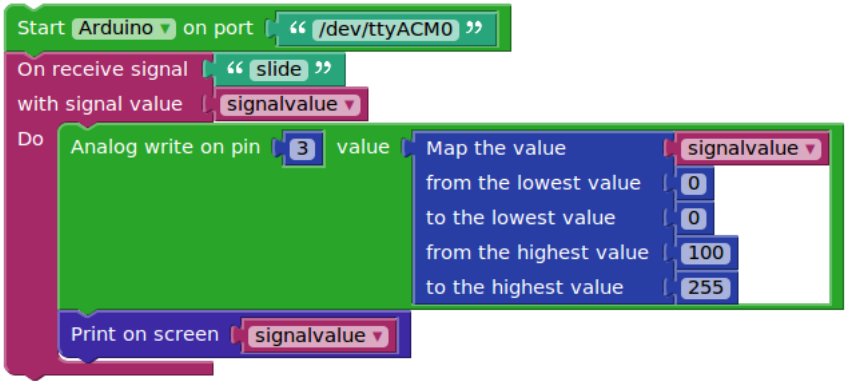


Figure 56: Coding the slider for the LED

In Python the according code is the following.

Python

```
1  from pyfirmata import Arduino
2  from pyfirmata import util
3  from wyliodrin import *
4  import json
5
6  signalvalue = None
7
8  def setBoard(boardType, port):
9    if boardType == 'arduino':
10     board = Arduino(port)
11   else:
12     board = ArduinoMega(port)
13   return board
14 board=setBoard('arduino', '/dev/ttyACM0')
15 reader = util.Iterator(board)
16 reader.start()
```

```
17
18 pin_var = board.get_pin("d:3:p")
19
20 def myFunction(__sender, __channel, __error, __message):
21   global signalvalue
22   signalvalue = int(json.loads(__message))
23   pin_var.write(map(signalvalue, 0, 100, 0, 255)/255.0)
24   print(signalvalue)
25
26 openConnection("signal:" + 'slide', myFunction)
```

All the functions used here were explained in this very chapter, towards the beginning and in the previous chapters.

Control LED from Twitter

You used the dashboard in order to turn on and off an LED. Now it's time to make something social out of this project. You will learn how to control an LED with the help of tweets.

At the end of the chapter you will have created a project that turns on an LED if you tweet *on* and turns it off if you tweet *off*.

What you need

You will use the same schematics as the one used in the *Blinking LED* or in the *Control LED from Dashboard* chapter.

The Code

For this project you will need to use some special twitter function, to be more exact, you will use a twitter API.

In order to have access to the twitter account, you need some permissions and you need to setup a "connection" between your account and the application you are going to create.

To achieve that, go to `https://apps.twitter.com`. Once you are logged

you can add a new application to your twitter account (Figure 57).

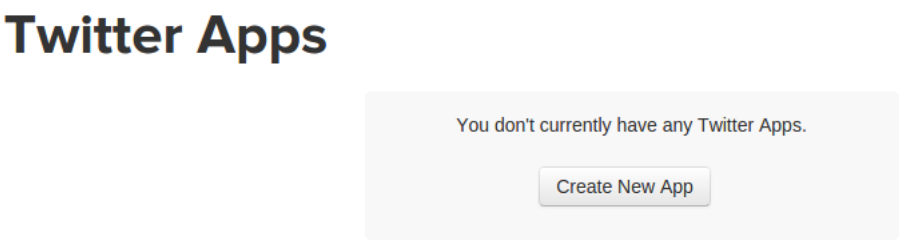


Figure 57: Add new application to twitter account

Further on you need to insert some details about the application, such as a title, a description and a link to the application. Once you achieved this, you will get to a page that displays links to all the tokens you need in order to get started (Figure 58).

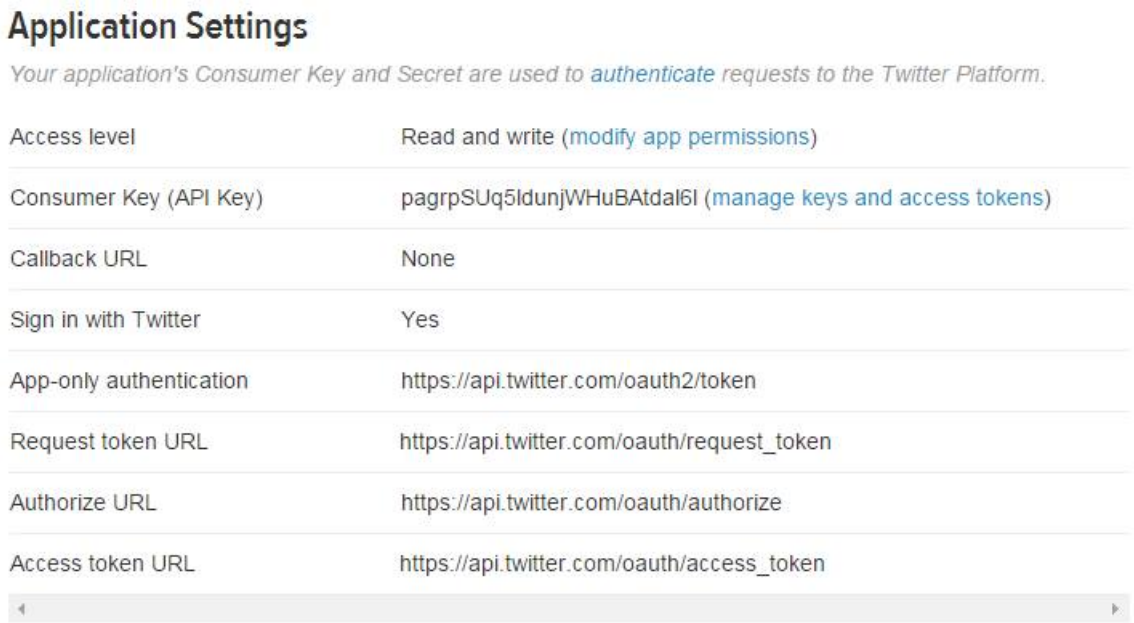


Figure 58: Application tokens

Now that you have everything you need, let's take a look at the code you need to build (Figure 59).

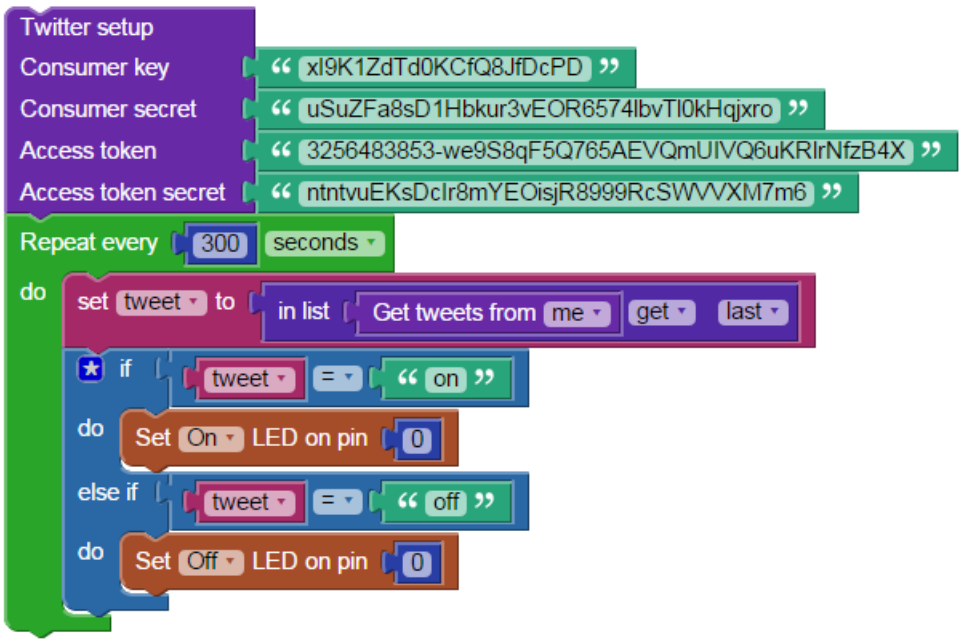


Figure 59: Twitter code blocks

As said before, first of all you need to connect your app to the twitter account. For this you have the *Twitter setup* block, which you can find under the *Social/Twitter* section. You already have the required keys and tokens, just replace them into this block.

The next step is to get your latest tweet from your timeline. You can do this by using the *Get tweets from* block. There you can use whether you want to get your tweets or the tweets of another user. If you prefer the second option, you will also need to insert the user's ID. This block returns a list with the tweets of the specified user, but you only need the latest of them. This is why you need to use the *in list [] get* block.

Now that you have only the latest tweet, you have to check what is says. In case is says *on*, you turn on the LED, in case it says *off*, you turn it off. All these steps are repeated every five minutes, this is the period the LED status will change. If you want to have a more interactive project, you can decrease

the repeat time.

Let's also take a look at the generated Python code.

Python

```
import sys
try:
  import tweepy
except:
  print("Please open the Shell and run 'social_install' script")
  sys.exit(1)

from wyliodrin import *

from threading import Timer

tweet = None

twitter_key = 'xI9K1ZdTd0KCfQ8JfDcPD'
twitter_secret = 'uSuZFa8sD1Hbkur3vEOR6574lbvTl0kHqjxro'
twitter_token = '3256483853-we9S8qF5Q765AEVQmUIVQ6uKRlrNfzB4X'
twitter_secretToken = 'ntntvuEKsDcIr8mYEOisjR8999RcSWVVXM7m6'

def twitterFunction(cKey, cSecret, aToken, aTSecret):
  auth = tweepy.OAuthHandler(cKey, cSecret)
  auth.set_access_token(aToken, aTSecret)
  twitter_tweet = tweepy.API(auth)
  return twitter_tweet

def twitterUserTimelineFunction(user):
  twitterFunction(twitter_key, twitter_secret, \
  twitter_token, twitter_secretToken)
  if user == 'myself':
    twitter_tweet =
```

```
30      public_tweets =twitter_tweet.user_timeline()
31    else:
32       public_tweet = twitter_tweet.user_timeline(user)
33    tweetArray = []
34    for tweetTimeline in public_tweets:
35      tweetArray.append(tweetTimeline.text)
36    return tweetArray
37
38  pinMode (0, 1)
39
40
41  def loopCode():
42    global tweet
43    tweet = (twitterUserTimelineFunction('myself'))[-1]
44    if tweet == 'on':
45      digitalWrite (0, 1)
46    elif tweet == 'off':
47      digitalWrite (0, 0)
48    Timer(300, loopCode).start()
49  loopCode()
```

First of all, the required modules are imported. One new module you are going to use is *tweepy*. This allows you to connect to twitter and get data from your or other user's account.

What you do next is to store the keys and tokens in four different variables. You will use them as parameters for the *twitterFunction*. This function connects to you twitter account and returns an object that contains all the required information.

Another important function is *twitterUserTimelineFunction*. This function has one argument, the user. If the user is *myself*, meaning that you want to get data from the same twitter account you connected to, after you authen-

ticate, you will have to call the *user_timeline* function. This returns your own tweets. If the user is not yourself, then you also have to pass on the user's ID.

You need only the text of the tweet, this is why you parse the list and create another one, *tweetArray* that conains only the text of each tweet.

In the *loopCode* you call the *twitterUserTimelineFunction* function and in order to retrieve its last element, you put the *[-1]* at the end.

You should be familiar with the rest of the code.

Show Facebook likes on a 7 Segment Display

In this chapter you are going social. We will show you how to to build a system that counts the Facebook likes of a page and displays that number on a 7 segment display. This way you can control in real time if your Facebook page has increased in likes without the need to go online every time.

What you need

- One seven segment display;
- Jumper wires;
- One 220 Ω resistor;
- One Raspberry Pi connected to Wyliodrin .

The Setup

First, let's start with a short description of the display. It consists of the 7 segments plus the dot, which are actually LEDs (Figure 60). Depending on how the 7 segment was designed, the eight LEDs either have a common

anode (connected to the VCC) or a common cathode (connected to the GND pin). Consequently, the display will be controlled by connecting each of the LEDs to the GPIO pins of the board. The whole idea is reduced to simply lighting on or off eight LEDs.

Any LED needs a resistor. As you know, the position of the resistor is not important so, in order to make it easier, just insert the resistor between the common pin, in your case the anode, and the VCC.

In the scheme in Figure 61 you can see how to connect the 7 segment display.

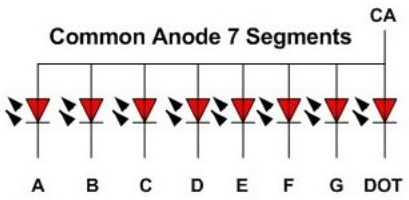


Figure 60: 7-segment display[4]

Each of the display's pins is connected to one of the segments. You will have to figure out which one each pin belongs to. For this you can search online the datasheet of your model.

[4] http://www.ermicro.com/blog/wp-content/uploads/2009/03/pictemp_04.jpg

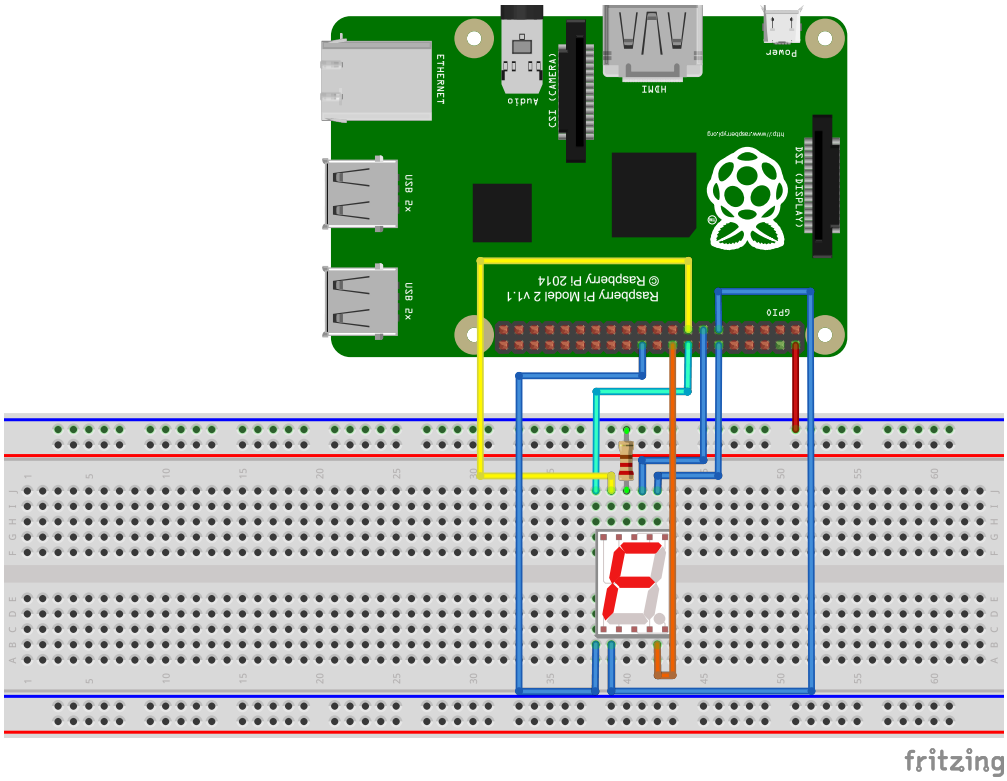


Figure 61: Connect the 7 segment display

The Code

The aim of the project is to display on the 7 segment the number of likes a Facebook page has.

In order to do that, you have to use the special blocks from the *Social* category. For this project, choose the one that gets the number of likes of a page you decide on. Visit the Facebook page and copy only the last part of the URL, which holds its actual name.

If the page has more than 9 likes it would not be possible to print the number on a 7-segment display, as it stands for one digit only. There's more than

one way to solve this problem: you either connect more displays in a row or you show one digit of the number you receive.

Look at the code in figure 62. You can easily obtain the last digit of the likes number by using the block that calculates the reminder of the division between your number and 10. Now you have the desired digit.

The next step is to initiate the 7 segment display. For this matter, you will have to put each segment on a certain pin. You have the figure drown on the block to provide you with the layout of segments. Use the datasheet to connect each of them.

Afterwards you must tell the display what to show, that would be the digit you've obtained.

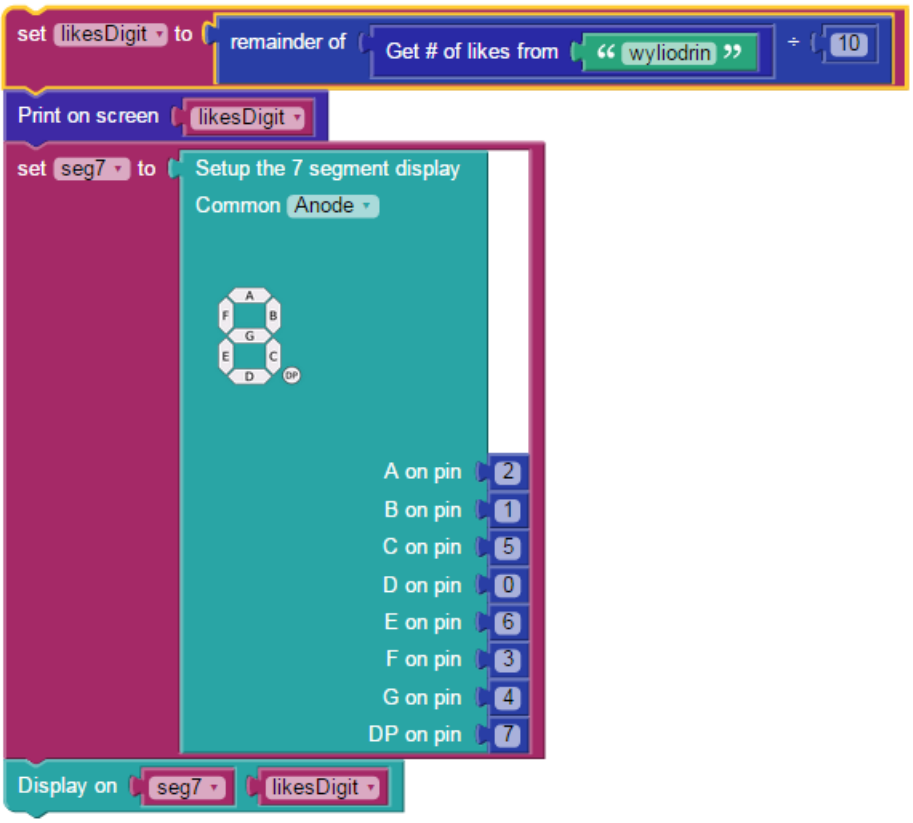


Figure 62: Facebook likes on 7 segment display in Visual Programming

Python

```python
import requests
from wyliodrin import *

likesDigit = None
seg7 = None

def getLikes(user):
  url = "https://graph.facebook.com/"+user
  response = requests.get(url)
  profile = response.json()
  likes = profile['likes']
  return likes

pinMode (2, 1)
pinMode (1, 1)
pinMode (5, 1)
pinMode (0, 1)
pinMode (6, 1)
pinMode (3, 1)
pinMode (4, 1)
pinMode (7, 1)

def displaySegment(sevenSegment, a, b, c, d, e, f, g, dp):
  digitalWrite(sevenSegment[1], abs(sevenSegment[0]-a))
  digitalWrite(sevenSegment[2], abs(sevenSegment[0]-b))
  digitalWrite(sevenSegment[3], abs(sevenSegment[0]-c))
  digitalWrite(sevenSegment[4], abs(sevenSegment[0]-d))
  digitalWrite(sevenSegment[5], abs(sevenSegment[0]-e))
  digitalWrite(sevenSegment[6], abs(sevenSegment[0]-f))
  digitalWrite(sevenSegment[7], abs(sevenSegment[0]-g))
  if len(sevenSegment)>=9: digitalWrite(sevenSegment[8], abs(sevenSegment[0]-dp))

```

```
33 def display(sevenSegment, value):
34   if value == "0":
35     displaySegment(sevenSegment, 1, 1, 1, 1, 1, 1, 0, 1)
36   if value == "1":
37     displaySegment(sevenSegment, 0, 1, 1, 0, 0, 0, 0, 1)
38   if value == "2":
39     displaySegment(sevenSegment, 1, 1, 0, 1, 1, 0, 1, 1)
40   if value == "3":
41     displaySegment(sevenSegment, 1, 1, 1, 1, 0, 0, 1, 1)
42   if value == "4":
43     displaySegment(sevenSegment, 0, 1, 1, 0, 0, 1, 1, 1)
44   if value == "5":
45     displaySegment(sevenSegment, 1, 0, 1, 1, 0, 1, 1, 1)
46   if value == "6":
47     displaySegment(sevenSegment, 1, 0, 1, 1, 1, 1, 1, 1)
48   if value == "7":
49     displaySegment(sevenSegment, 1, 1, 1, 0, 0, 0, 0, 1)
50   if value == "8":
51     displaySegment(sevenSegment, 1, 1, 1, 1, 1, 1, 1, 1)
52   if value == "9":
53     displaySegment(sevenSegment, 1, 1, 1, 1, 0, 1, 1, 1)
54   if value == 'A':
55     displaySegment(sevenSegment, 1, 1, 1, 0, 1, 1, 1, 1)
56   if value == "B":
57     displaySegment(sevenSegment, 1, 1, 1, 1, 1, 1, 1, 1)
58   if value == "C":
59     displaySegment(sevenSegment, 1, 0, 0, 1, 1, 1, 0, 1)
60   if value == "D":
61     displaySegment(sevenSegment, 1, 1, 1, 1, 1, 1, 0, 1)
62   if value == "E":
63     displaySegment(sevenSegment, 1, 0, 0, 1, 1, 1, 1, 1)
64   if value == "F":
65     displaySegment(sevenSegment, 1, 0, 0, 0, 1, 1, 1, 1)
```

```
66    if value == ".":
67      displaySegment(sevenSegment, 0, 0, 0, 0, 0, 0, 0, 1)
68
69
70  likesDigit = (getLikes('wyliodrin')) % 10
71  print(likesDigit)
72  seg7 = [1, 2, 1, 5, 0, 6, 3, 4, 7]
73  display(seg7, str(likesDigit))
```

For the python code, first you need to import the *requests* module. By doing this, you have access to functions that allow you to get data from the Internet. Further on, you will notice that there are some lines of code containing *requests.* (line 9 and 10). These lines use functions related to the *requests* module.

In Python there are multiple ways of importing modules. One would be the one on line 2, which you have seen in previous chapters. In this case, the module is imported together with all its functions and those functions can be used as any other function. Another way is the one on line 1. Here, the module is imported as a variable that has the name of the module. The variable is a structure that contains all the functions and attributes of this module. So if you want to call a function that is related to this module, you have to get it from the structure. This is why you first have to write the module followed by a dot, then the function. This way you will obtain the desired function.

Next, you define the method *getLikes* which has the argument *user*. The argument represents the page you wish to inquire about.

You will need the URL of the page, assigned to the variable *url*. You also need some information about this page. That you request and assign to *response* variable by using the *get* function of the *requests* module. The data you receive is however difficult to parse. Subsequently, you need to store it in a simpler format by using the *json* function. Now you have a JSON

structure that holds all the information. Then, you select the needed piece of information, which is here *likes*. Mission accomplished, now you have the number of likes.

The next step is to prepare the display. You set all the output pins, and define the method *displaySegment* which will make the connection of each position of segment (a,b,c etc.) with the corresponding pin. The *if* at the end is for the DP position assigned to the dot, because this one can be absent.

The first argument of this function is an array. You can think of it as a list of elements, of numbers is this case, in which each element can be accessed by its place in the list. Thus, *sevenSegment[1]* will be the second element in the list, as the first position is 0. An example is the variable *seg7* in line 72.

Here's what actually happens. If you look at the array *seg7* you will see that its first position is 1, and so it will be for any 7 segment with a common anode. In case the 7 segment has a common cathode, the first element of the array will be 0.

Remember, the displays that have common anode will turn on the segments only when the pins assigned to the latter are set to low.

This is why you store the first value in the array, because later you will use it to subtract that value from the value you want to write for each pin. You will consider that if you want an LED to be lit up, its corresponding value is 1, while for a turned off pin the value is 0. In case the 7 segment display has a common cathode, thus each segment lights up when it gets to HIGH, you will subtract 0, thus the value remains the same. In the other case, you will subtract 1, so the 1 becomes 0 and the 0 becomes 1, this is possible because you use the *abs* function. The function returns the absolute value.

All things taken into consideration, look at the *digitalWrite* function. Its first parameter would be the position of the pin you set for the *a segment*, correct? So, digitalwrite will turn that pin to high or low.

The next defined method builds each of the digits following this conclusion. In other words, it tells the display how to manage the pins so that it can show the digit you expect by giving them the relevant values. It applies to capital letters as well.

The remainder is calculated and the chosen Raspberry Pi pins are announced as an array in variable *seg 7* which is the parameter of the *display* method.

Connecting Sensors to Arduino

This chapter is the first step in order to turn your house into a smart one.

At the end of this project you will know how to get data from different sensors you can place around your home and display the gathered data on graphs.

As explained in the previous chapters, no analog pin dependent piece can be used with your Raspberry Pi only. You will need the help of an Arduino board.

Information on how you connect the two boards and details on why you need this measure taken can be found at the beginning of the *LED line* chapter.

Let's begin with the temperature sensor.

Build a virtual thermometer

What you need

- One temperature sensor;
- Three jumper wires;
- One Arduino board;
- One Raspberry Pi connected to Wyliodrin .

The Setup

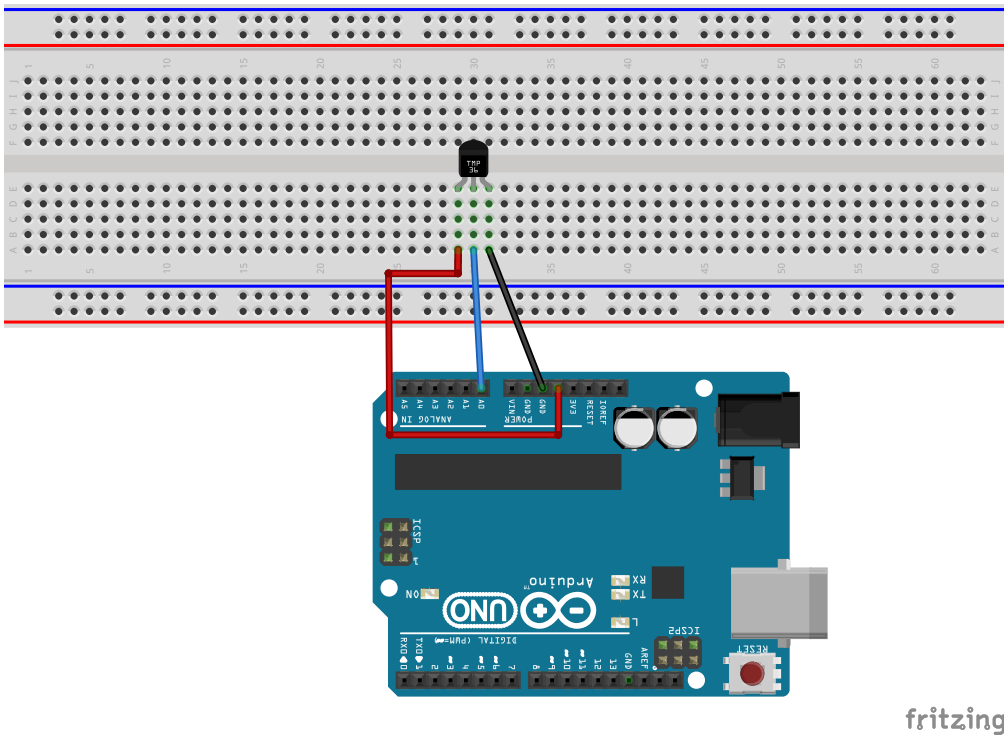


Figure 63: How to connect a temperature sensor

In figure 63 you can see a 3-leg temperature sensor. There is an alternative to this one, the thermistor. The latter has two legs and it should be connected as a voltage divider, which is explained in the electronics chapter.

In this case, the sensor needs to be connected to the Gnd pin, the VCC and an analog pin. This sensor consists of a thermistor and several other electronic pieces. The leg situated in the middle outputs a different voltage depending on the temperature.

The Code

In order to get the values from the the temperature sensor you will just have to use the Arduino blocks and read the analog pin. The values that you obtain are in a range of 0 to 1023. However, this doesn't look at all like the values you would expect from a thermometer.

To transform these values into Celsius degrees, you are going to use this formula.

$$temperature = (value * \frac{5}{1024} - 0.500) * 100 \tag{5}$$

It is very important that you convert the value received from the analog pin into a *float* number, otherwise you will not obtain a proper value. Now, it's advised that you assign this value to a variable as shown in figure 64.

To get from this value to Celsius degrees you have to follow the formula above. The resulting value is stored in the *temperature* variable. As promised, you will now display the temperature on a thermometer using the *send signal* block.

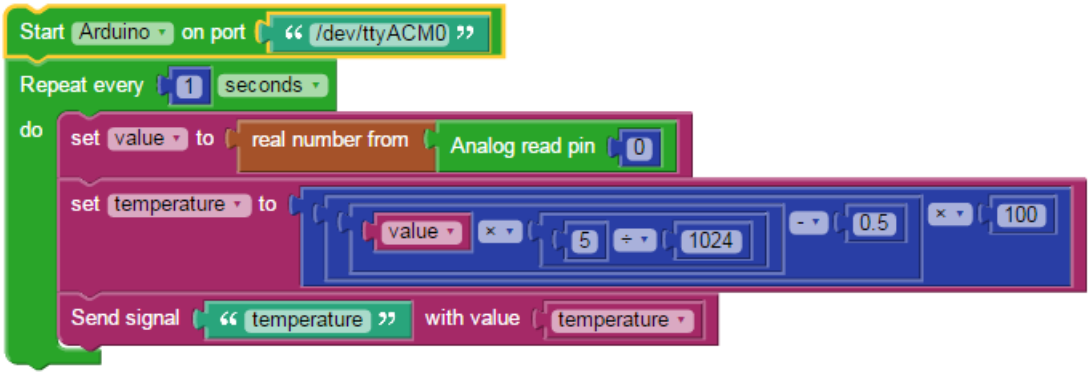


Figure 64: Thermometer in visual programming

In dashboard choose the thermometer and assign it a signal with the same name as used in the blocks (Figure 65).

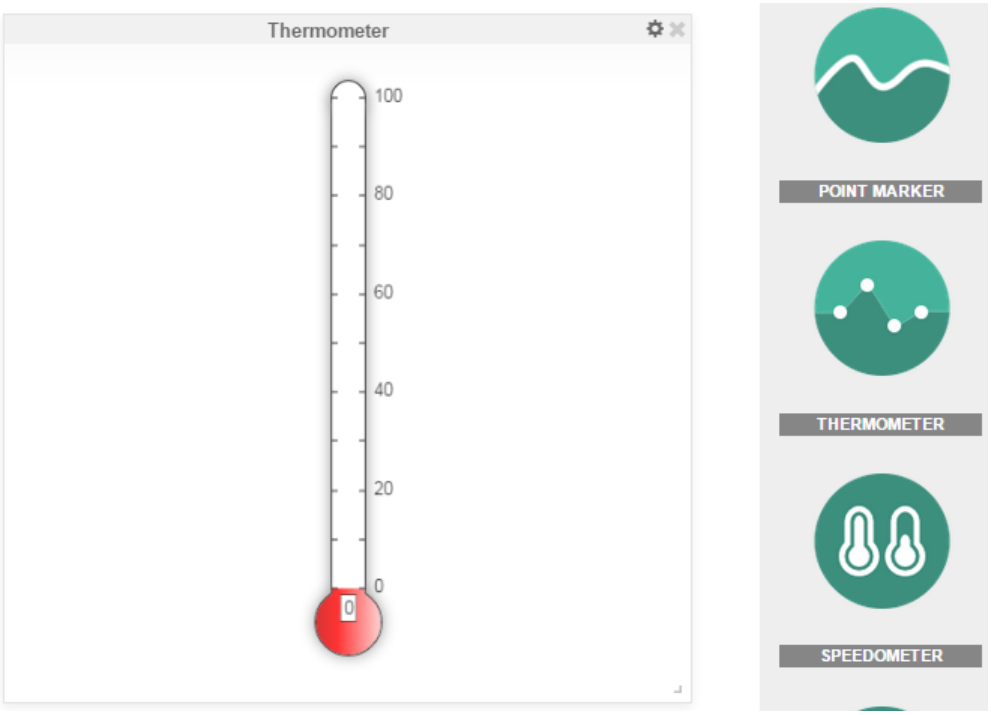


Figure 65: Thermometer in dashboard

Python Code

In Python you will notice, apart from the imports and the Arduino functions which are explained in the *LED Line* chapter, the conversion of the read value to Celsius degrees.

Python

```python
from pyfirmata import Arduino
from pyfirmata import util
from wyliodrin import *
from threading import Timer

value = None
temperature = None

def setBoard(boardType, port):
  if boardType == 'arduino':
    board = Arduino(port)
  else:
    board = ArduinoMega(port)
  return board
board=setBoard('arduino', '/dev/ttyACM0')
reader = util.Iterator(board)
reader.start()

pin_var = board.get_pin("a:0:i")

def loopCode():
  global value, temperature
  value = float (round((pin_var.read() or 0) * 1023))
  temperature = (value * 5 / 1024 - 0.5) * 100
  sendSignal('temperature', temperature)
  Timer(1, loopCode).start()
```

```
27 loopCode()
```

Use Twilio

You can make this project interactive by making a call when the temperature surpasses a given limit. In order to do this, you can use the Twilio service.

Twilio is a cloud communications service that allows software developers to use programming to make and receive phone calls or send and receive text messages.

Attention! You can call or send messages to your own number for free, but you must upgrade to a paid account for using other numbers.

For more information please go to `https://www.twilio.com/pricing`.

The code is simple: Figure 66.

First of all you have to set up your account. For this you need two pieces of information: account and token. Both can be found on the Twilio site.

The token can be found underneath your name, in the *Account* section. You can reveal it by clicking on the lock. Copy and paste it inside your block. The Account also requires the AccountSID which is located in the same section, above the token (Figure 67).

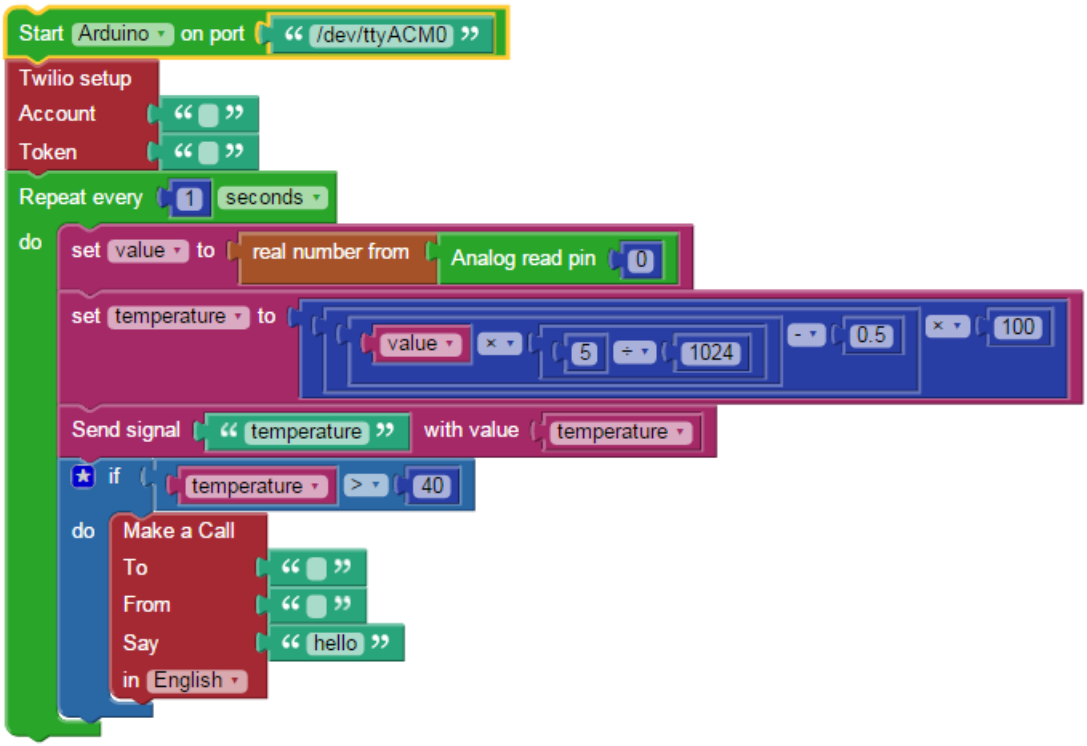


Figure 66: Send a message using Twilio

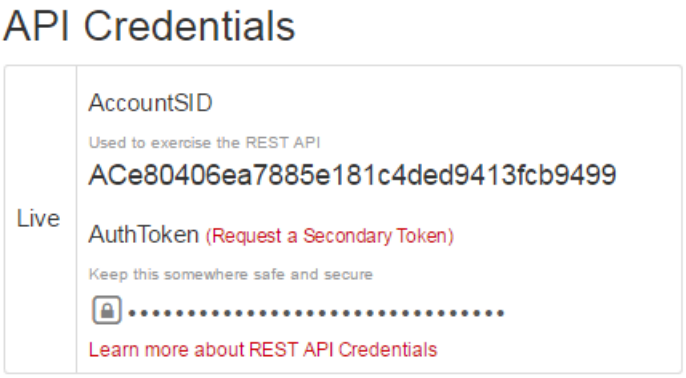


Figure 67: Twilio AccountSID and token

Now that the setup is complete, you can proceed to making the call. Choose the block *Make a Call* under Twilio, and *to* will be your own number with

the country code. In the *from* section, go on Twilio's site, find the *Dashboard* page and, in the *Developer Tools*, click on *Explore* for calls. There's where you can find your Twilio number (Figure 68).

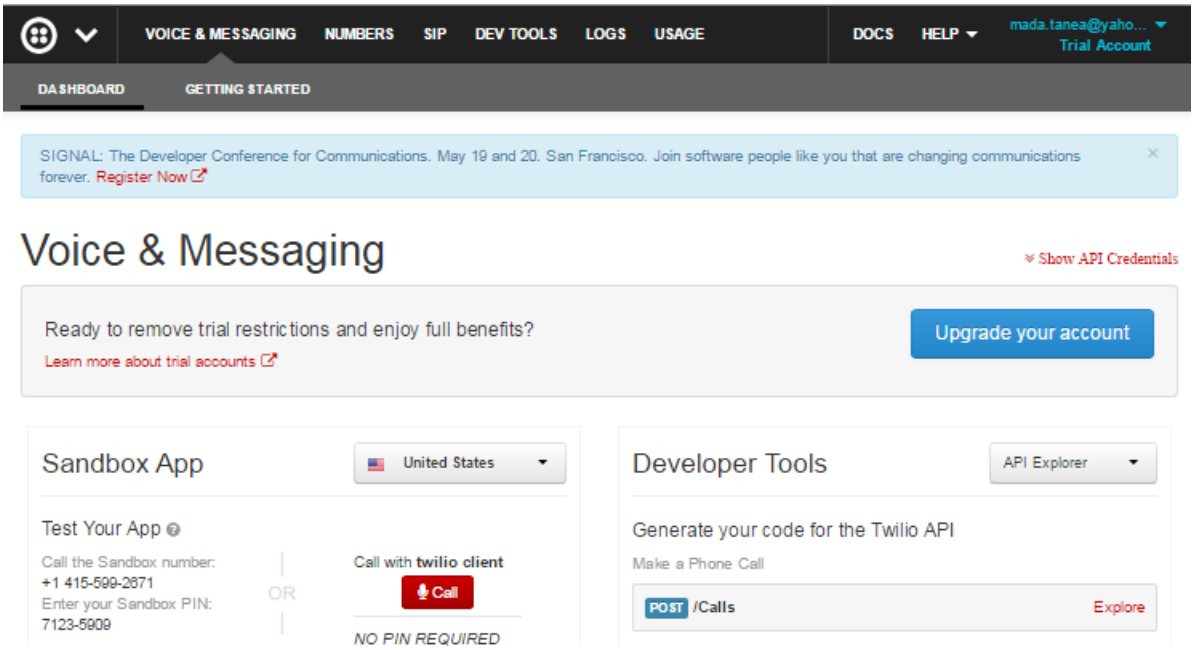


Figure 68: Your Twilio phone number

The Python code:

Python

```
1  from pyfirmata import Arduino
2  from pyfirmata import util
3  import sys
4
5  try:
6    from twilio.rest import TwilioRestClient
7  except:
8    print("Please open the Shell and run 'social_install' script")
9    sys.exit(1)
10
11 from wyliodrin import *
```

```
12 from urllib import *
13 from threading import Timer
14
15 value = None
16 temperature = None
17
18 def setBoard(boardType, port):
19   if boardType == 'arduino':
20     board = Arduino(port)
21   else:
22     board = ArduinoMega(port)
23   return board
24 board=setBoard('arduino', '/dev/ttyACM0')
25 reader = util.Iterator(board)
26 reader.start()
27
28 twilio_account = ''
29 twilio_token = ''
30
31 pin_var = board.get_pin("a:0:i")
32
33
34 twilio_client = TwilioRestClient(twilio_account, twilio_token)
35 def loopCode():
36   global value, temperature
37   value = float (round((pin_var.read() or 0) * 1023))
38   temperature = (value * 5 / 1024 - 0.5) * 100
39   sendSignal('temperature', temperature)
40   if temperature > 40:
41     call =twilio_client.calls.create(to='', from_='', \
42     url='http://twimlets.com/echo?'+ \
43     urlencode({'Twiml':'<Response><Say voice="alice" \
44     language="en-EN">'+'hello'+'</Say></Response>'}))
```

```
45   Timer(1, loopCode).start()
46 loopCode()
```

What is new, compared to the first Python code, is the exception in the imports, which basically makes sure there is such a script on your board's configuration that knows the Twilio functions. If it is already installed, it will import *TwilioRestClient*, otherwise it will show the corresponding message.

You define the two variables *twilio_account* and *twilio_token* and you identify the *twilio_client* with them, as you can see in line 34. Then, you use the Twilio functions to make the call with the message you wrote in the message box. as follows. First, you create a call with the information given before, the identification ones for the caller's account and the number which will be called. In line 42 you choose from *twimlets*, an internet virtual voice provider, the voice that will say the word in your message box, depending on the language.

Use a thermistor

As mentioned at the beginning, the same project can be easily built with a thermistor. Note that there is a difference between the two sensors, consisting of the fact that the temperature sensor previously used has a linear resistance-temperature characteristic, while the other one usually has the output voltage as a logarithmic or exponential function. The wiring is quite simple, just follow the schematics of a voltage divider.

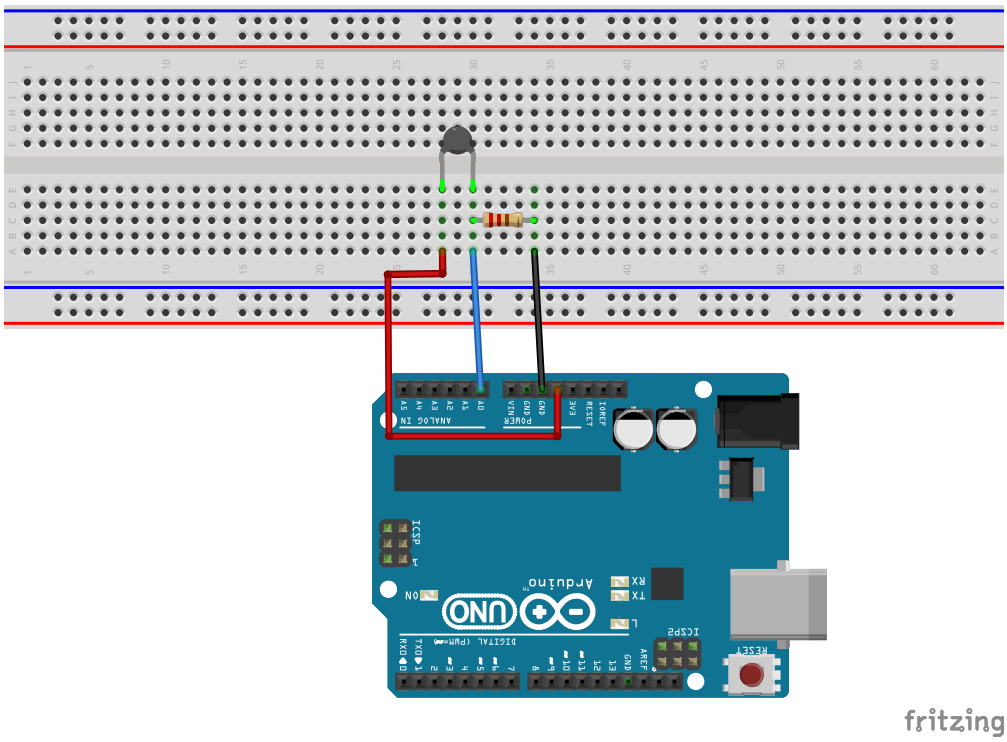


Figure 69: Thermistor schematics

For the code you will need to know that this time the formulas you use are different to convert the values into Centigrade.

$$V_m = \frac{analogRead(pin) * 5}{1023} \tag{6}$$

$$R_t = (\frac{5}{V_m} * 10000) \tag{7}$$

$$Ratio = \frac{1}{B} * ln\frac{R_t}{10000} \tag{8}$$

$$temperature = \frac{1}{\frac{1}{298} + Ratio} \tag{9}$$

B = 4050, is one of the constants that comes with the sensor's specifications.

Vm is the voltage that you get in your analog input while Rt is the resistance of the thermistor. By having this value, you just have to apply the following formulas and you will obtain the temperature in Celsius degrees. In Visual Programming it will look the same as in Figure 70

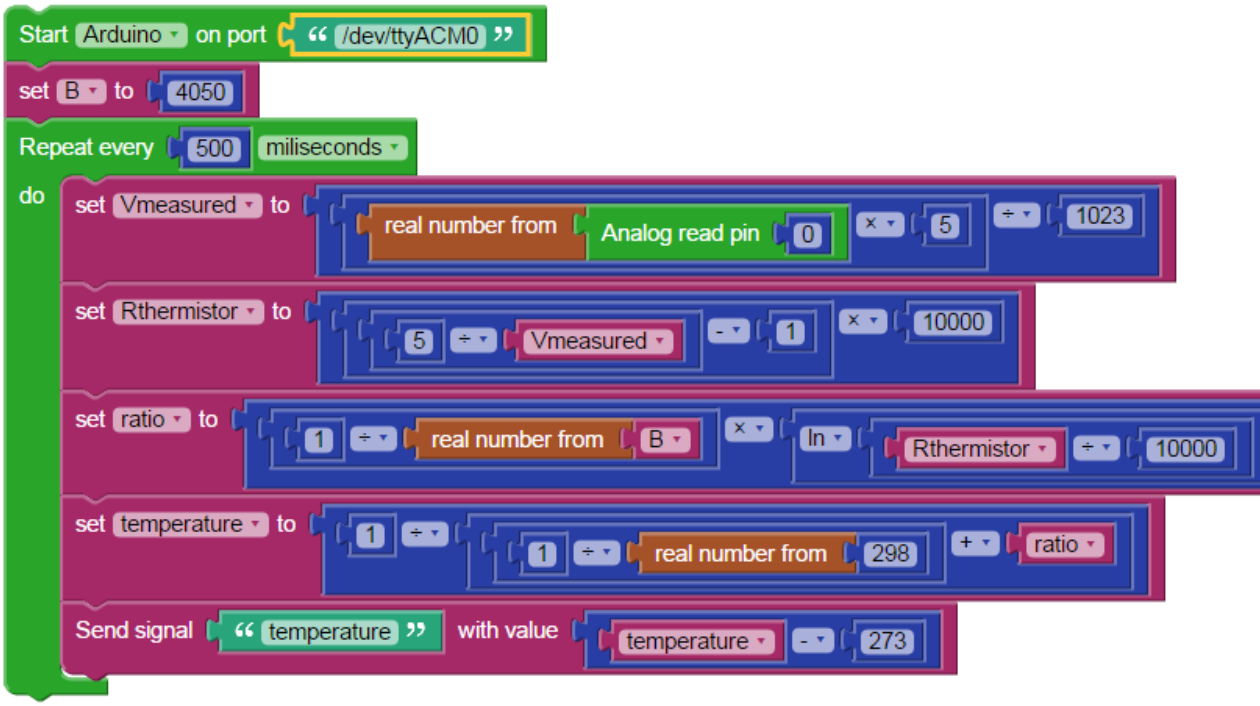


Figure 70: Read values from a thermistor in Visual Programming

You only have to calculate what the formulas above will give and put it in a variable. Send the value of this variable to the thermometer in the Dashboard.

The Python code will follow the usual Firmata functions along with the formulas above.

Python

```
from pyfirmata import Arduino
from pyfirmata import util

import math
from wyliodrin import *

from threading import Timer

B = None
Vmeasured = None
Rthermistor = None
ratio = None
temperature = None

def setBoard(boardType, port):
  if boardType == 'arduino':
    board = Arduino(port)
  else:
    board = ArduinoMega(port)
  return board
board=setBoard('arduino', '/dev/ttyACM0')
reader = util.Iterator(board)
reader.start()

pin_var = board.get_pin("a:0:i")


B = 4050

```

```
30 def loopCode():
31   global Vmeasured, Rthermistor, ratio, B, temperature
32   Vmeasured = (float (round((pin_var.read() or 0) * 1023))) * 5 / 1023
33   Rthermistor = (5 / Vmeasured - 1) * 10000
34   ratio = 1 / (float (B)) * math.log(Rthermistor / 10000)
35   temperature = 1 / (1 / (float (298)) + ratio)
36   sendSignal('temperature', temperature - 273)
37   Timer(0.5, loopCode).start()
38 loopCode()
```

See how the light changes in a room

You that now have a thermometer can you find something extra to make it more interesting? Try a light sensor.

What you need

- One photocell;
- One 220 Ω resistor;
- Three jumper wires.

The Setup

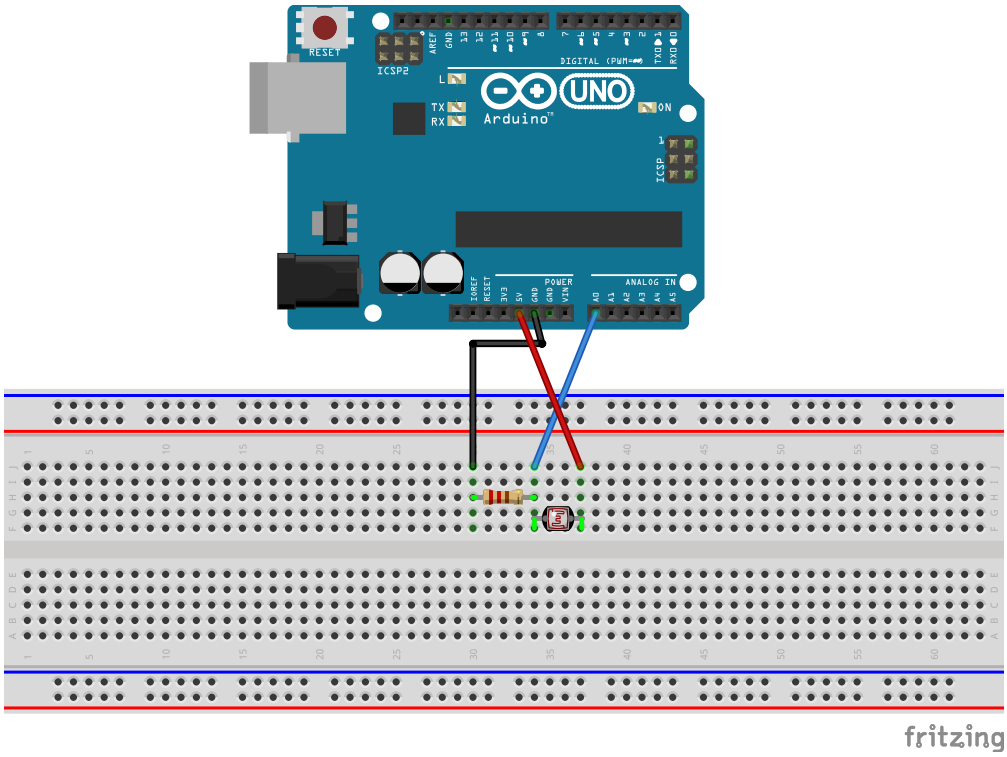


Figure 71: Connect the photocell

The photocell works just like a voltage divider, as well (check the *Introduction to Electronics* chapter for more details). You need to connect it to the Gnd and Vcc and from between the photocell and the resistor to the analog pin of the Arduino. This light sensor has a resistance that varies with the light, the more light there is, the lower the resistance of the photocell. Depending on how you place the sensor you can obtain a pull up or pull down voltage divider.

The Code

The new code brings one more block, the one that will send the value you read from your sensor to the chart. Name the signal and go to dashboard to choose the chart and confirm the name in its settings.

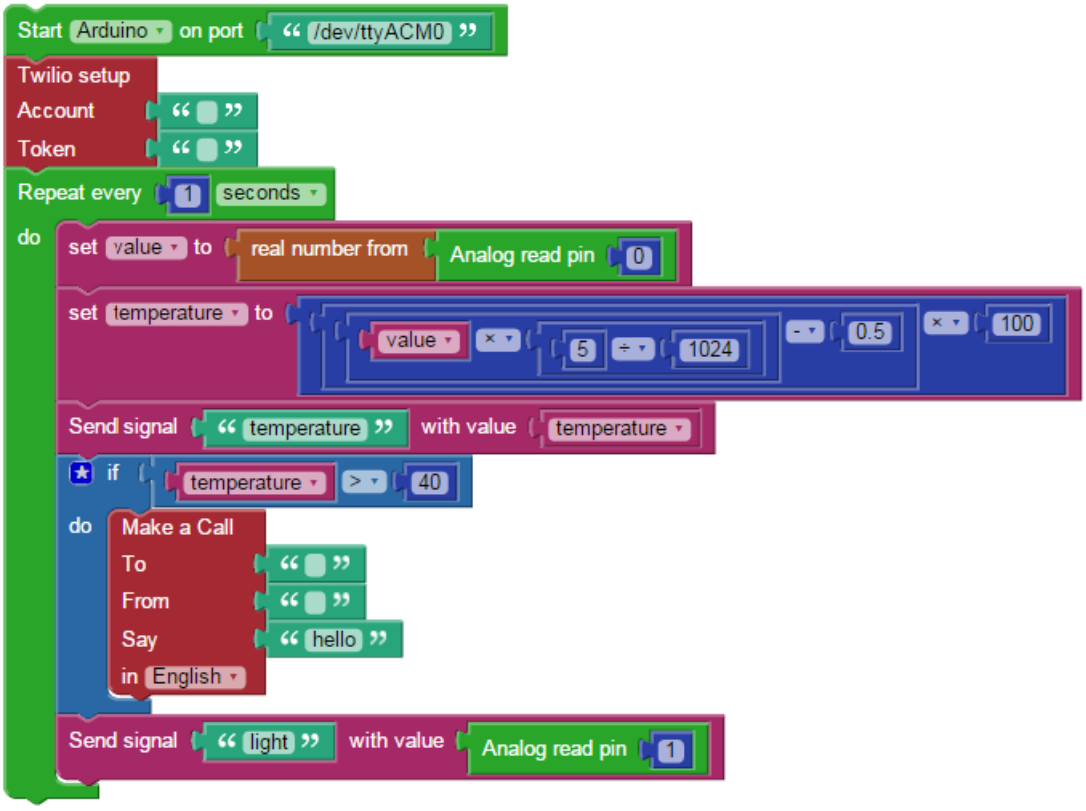


Figure 72: Add a light sensor in Visual Programming

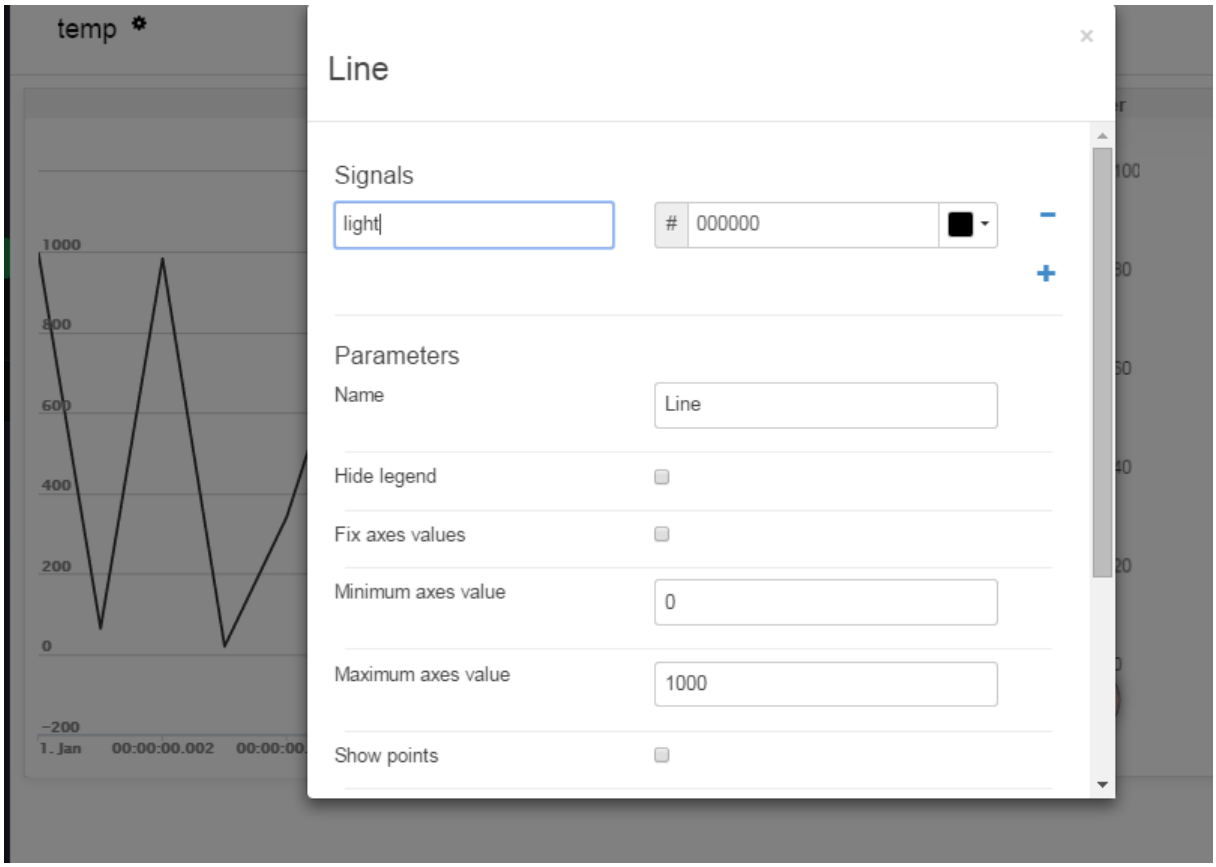


Figure 73: Light sensor graph's settings

Write "Hello World" on the LCD

In this project you are going to connect an LCD to the Raspberry Pi and write a short message on it.

What you need

- One Raspberry Pi connected to Wyliodrin ;
- One 16x2 LCD;
- One breadboard;
- Eight male-male jumper wires;
- Eight male-female jumper wires;
- One potentiometer.

The Setup

The LCD has a parallel interface, meaning that the board has to manage several interface pins at once to control the display.

The first pin is the *RS*(Register Select) that controls where exactly in the LCD's memory you are writing data to.

There is an *Enable* pin that enables writing to the registers.

The third one is the *R/W*(Read/Write) pin that selects from reading or writing mode.

The LCD also has *8 data* pins (D0 -D7). The states of these pins (high or low) are the bits that you're reading to a register when you read, or the values you are writing when you write.

There are also power supply pins (5V and GND) for powering the LCD, LED Backlight (Bklt- and Bklt+) pins that you can use to turn off and on the LED backlight and a display contrast pin (Vo) to control the display contrast.

What does 16×2 LCD mean? It means that the LCD has 16 columns and 2 rows. You place the LCD on the breadboard and connect it to the Raspberry Pi as shown in figure 74.

The first two pins on the right are the used to power up the LCD. The next four pins, the ones with green cables, are the data pins. They allow the LCD to communicate with the board. The next two pins connected by yellow cables are the control pins. There are also three pins used for contrast. Two of them are used to power on the backlight and there is one more pin directly connected to the potentiometer. This way you can control the contrast just by rotating it.

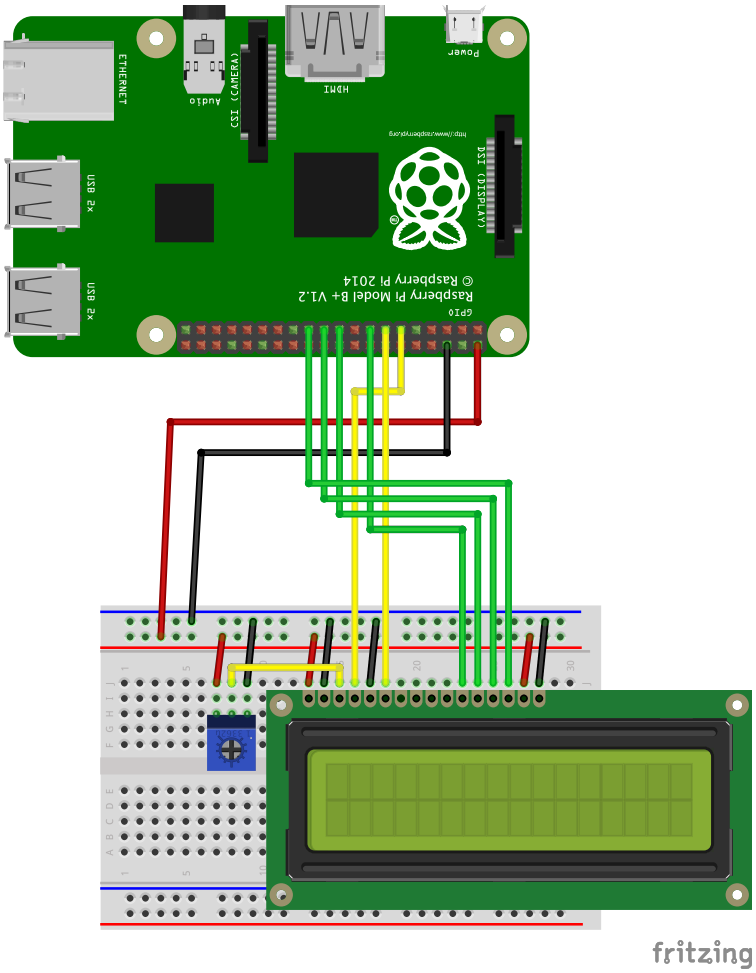


Figure 74: LCD schematics

The Code

You go to the Wyliodrin Applications page and create a new application. You name it and select for the programming language *New - Visual programming*. Once created, you click on the new application's name to open it.

First of all, you need to initialise the LCD. For this you select the *Peripherals/LCD/Init LCD* block. you need to complete the numbers of the pins for

the LCD. They are the following:

- RS pin - 0;
- Enable pin 2;
- Data 1 pin - 3;
- Data 2 pin - 12;
- Data 3 pin - 13;
- Data 4 pin - 14.

To write something, you will use the *Perpherals/LCD/Print on LCD []* block. you place it below the *Init* block and write *Hello World* in it (Figure 75).

Once the project runs, you should see "Hello World" on the LCD, however, the contrast might be to high and the text can be difficult to read. In order to modify the contrast, you just roll the rotary angle.

You run the project.

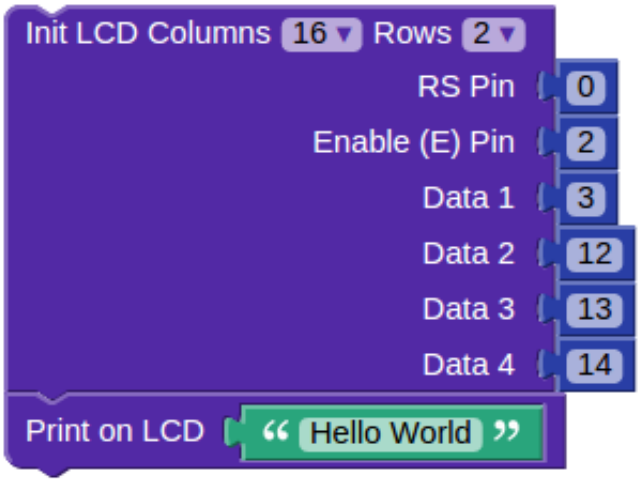


Figure 75: LCD application

Use a button

Remember the LED lamp you built earlier? Now you will be able to turn it on an off with a button.

This project will show us how to connect a button to the Raspberry Pi and use it to light up an LED.

What you need

- One Raspberry Pi connected to Wyliodrin ;
- Two 200 Ohm resistor;
- Four male-female jumper wires;
- Two male-male jumper wires;
- One button;
- One LED.

The Setup

First of all, you will connect the button. The button should be connected similar to the schematics in figure 76.

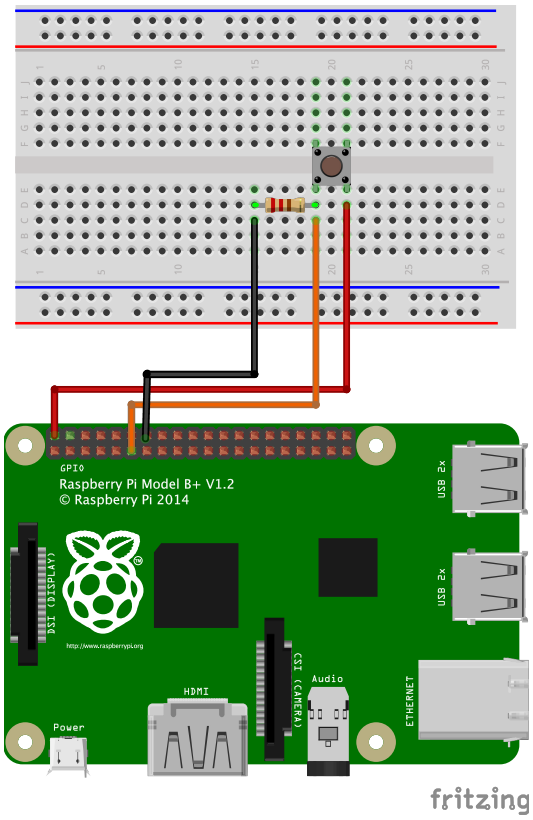


Figure 76: Button Schematics

You can notice that the button was connected according to the schematics presented in the electronics chapter. Basically, you have the VCC on one side and the ground together with the resistor and the GPIO connection on the other side. Once you press the button, the two sides get connected.

The Code

You will use a graph that will show when the button is pressed and when not.

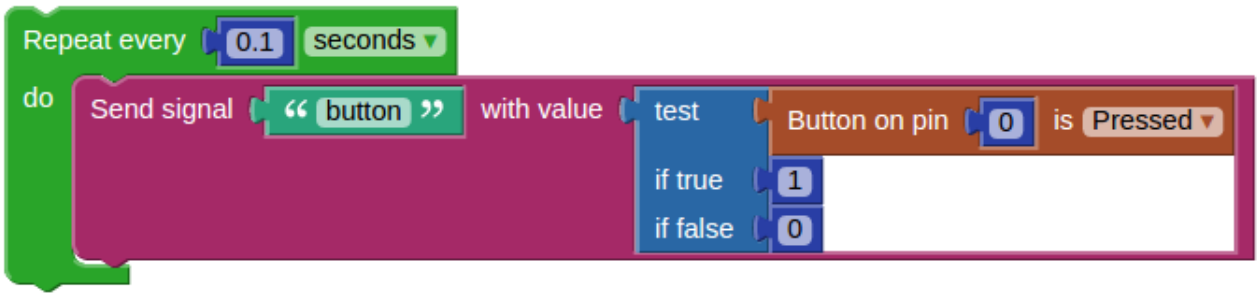


Figure 77: Button application

What the application will do is to check whether the button is pressed every 100 milliseconds and send a specific signal to a graph. This is why you used the *repeat every [] seconds* block.

Inside the *repeat* you used the *Signals/Send signal [] with value []* block. The block send a signal with the name "button" and with a certain value. The value is either 0 or 1. It depends on the state of the button. The *Sensors/Buttons/Button on pin [] is pressed* block returns true is the button is pressed, or false if the button is not pressed.

The *Program/Logic/test* block tests the value of the condition and it returns a certain value if it is true or if it is false. In your case, the condition is to have the button pressed. If it is true you return 1, otherwise you return 0.

The Python code is rather simple. Basically, every 0.1 seconds there is an *analogRead* on pin 0. The returned value is evaluated by the *if* statement and it is reduces to 1 if the button is pressed and 0 otherwise.

The value of 0 or 1 obtained after the evaluation is sent as a signal with the

name *button*.

You can plot the signal on a spline line graph, as you have done in the previous chapters.

Python

```
from wyliodrin import *
from threading import Timer

pinMode (0, 0)

def loopCode():
  sendSignal('button', 1 if (digitalRead (0) == 1) else 0)
  Timer(0.1, loopCode).start()
loopCode()
```

Light up LED at button pressed

What you need

- The previous configuration;
- One LED;
- One resistor;
- Female-male jumper wires;
- Male-male jumper wires.

To give the previously connected button a purpose, we suggest you light up an LED when the button is begin pressed.

The Setup

You need to use the schematics in figure 78.

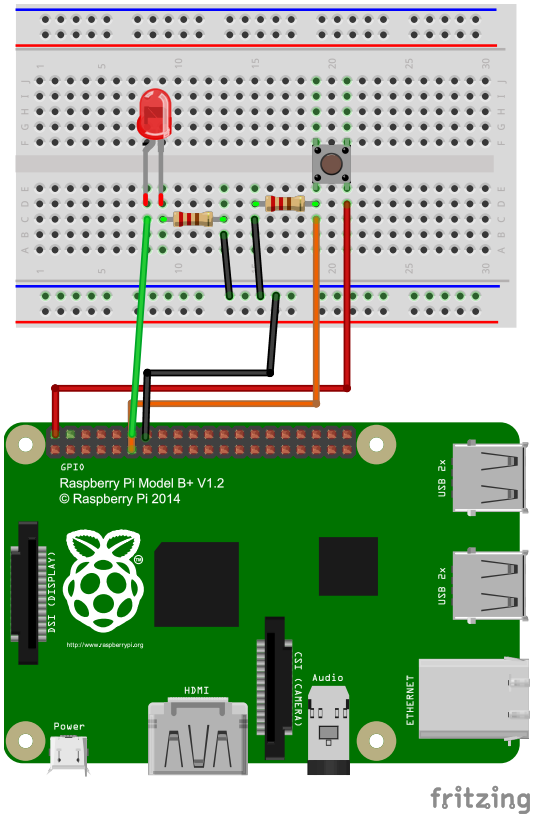


Figure 78: Button and LED Schematics

You can see there that apart from the button, there is a simple LED connect to pin 1 of the Raspberry Pi, just like in the previous chapters.

The Code

Now let's write the code that allows you to light up the LED only when the button is pressed.

The blocks look like the ones in figure 79.

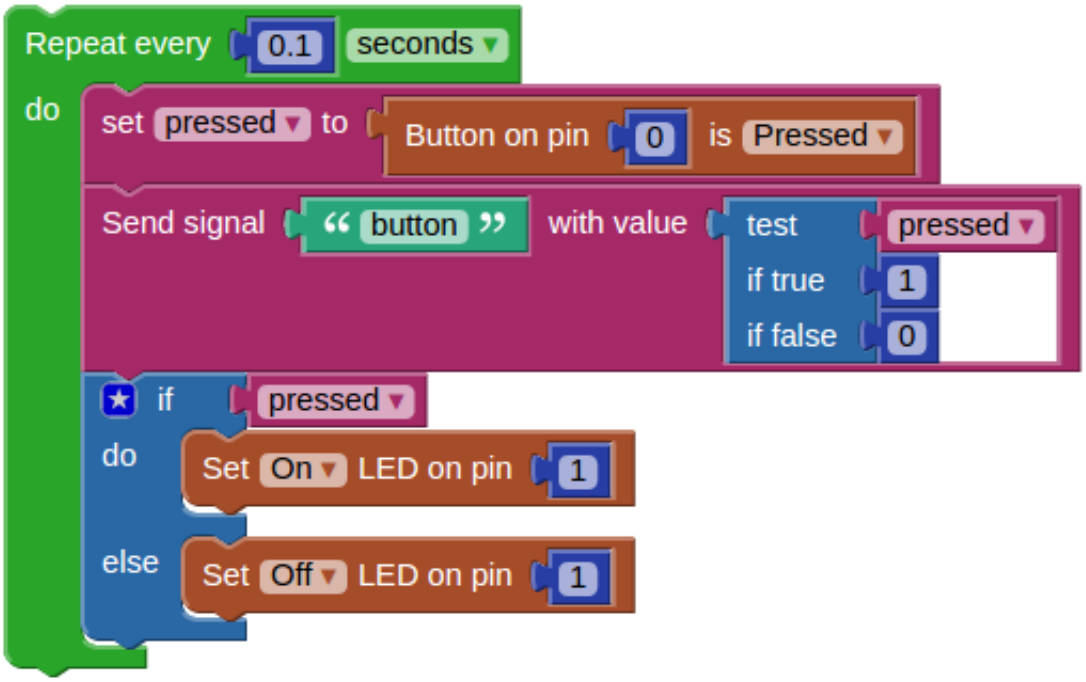


Figure 79: LED light up code

You can notice that we introduced a variable in this program. The *pressed* variable takes the value read from the pin 0. Why is a variable needed? Because otherwise you would have to do the reading twice. This is more efficient as the read value is stored and further on the *pressed* variable is used.

In order to light up the LED that is connect on pin 1 on the Raspberry Pi, you have to check if the button is pressed. If it is pressed, you turn the LED to On, otherwise, you turn it to Off.

You can also find the generated Python code below.

Python

```
1 from wyliodrin import *
2 from threading import Timer
```

```
3
4  pressed = None
5
6  pinMode (0, 0)
7  pinMode (1, 1)
8
9  def loopCode():
10   global pressed
11   pressed = digitalRead (0) == 1
12   sendSignal('button', 1 if pressed else 0)
13   if pressed:
14     digitalWrite (1, 1)
15   else:
16     digitalWrite (1, 0)
17   Timer(0.1, loopCode).start()
18 loopCode()
```

You can notice that after the modules are imported, the *pressed* variables is declared and its value is set to None.

Now that the LED is introduced, pin 1 is set to output. Then follows the *loopCode* inside which the variable is mentioned again. The *global* which precedes it states that the *pressed* variable inside the loop is the same with the one declared above.

You should already be familiar with the rest of the code. Basically every 0.1 seconds the value from the button is read and it is stored into the variable. Then the signal is sent and also, if the case, the LED is lit up.

Light up a multicolor LED

Let's replace the simple blinkling lamp with a multicolor one.

The purpose of this project is to light up an RGB LED in different colors. In order to achieve this, you need three PWM pins to control the LED. As the Raspberry Pi has only one PWM pin, you are going to connect an Arduino to it and connect the LED to its pins.

What you need

- One Raspberry Pi connected to Wyliodrin ;
- One Arduino;
- One RGB LED;
- Three 200 Ohm resistors;
- Male-female jumper wires.

The Setup

Connect the Arduino to the Raspberry Pi through the USB cable.

Connect the RGB LED to the Arduino following the schematics in figure 80.

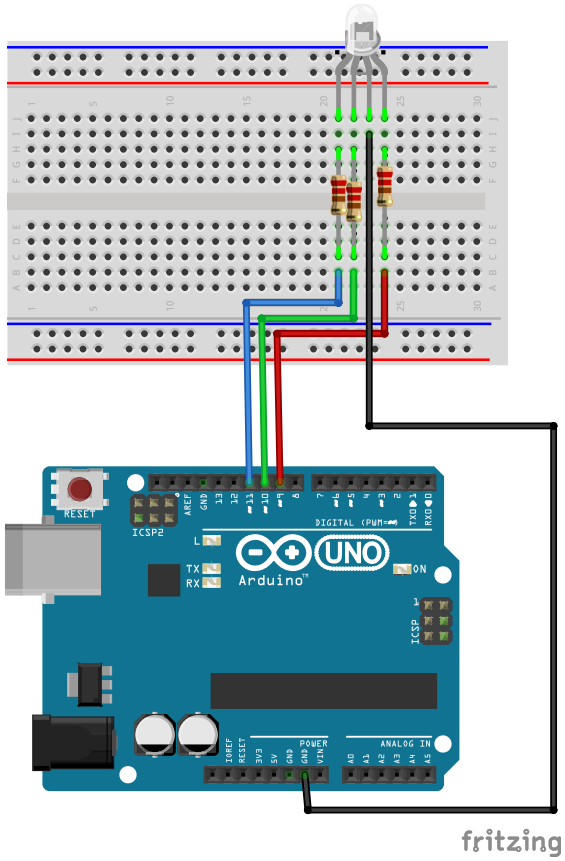


Figure 80: RGB LED schematics

The RGB LED is simply an LED consisting of three simple LEDs: a red one, a green one and a blue one. Each of these LEDs is controlled by a PWM pin, so that it can light brighter or dimmer. As a result, the color of the RGB LED will be the result of these three colors.

The Code

You go to the Wyliodrin Applications page and create a new application. you name it and select for the programming language *New - Visual programming*. On the second page, select Connect an Arduino to the board (Raspberry Pi only). This will add an Arduino component to your application.Once created, you click on the new application's name to open it. you can notice that this project has two generated files: *main.visual* and *Arduino.ino*.

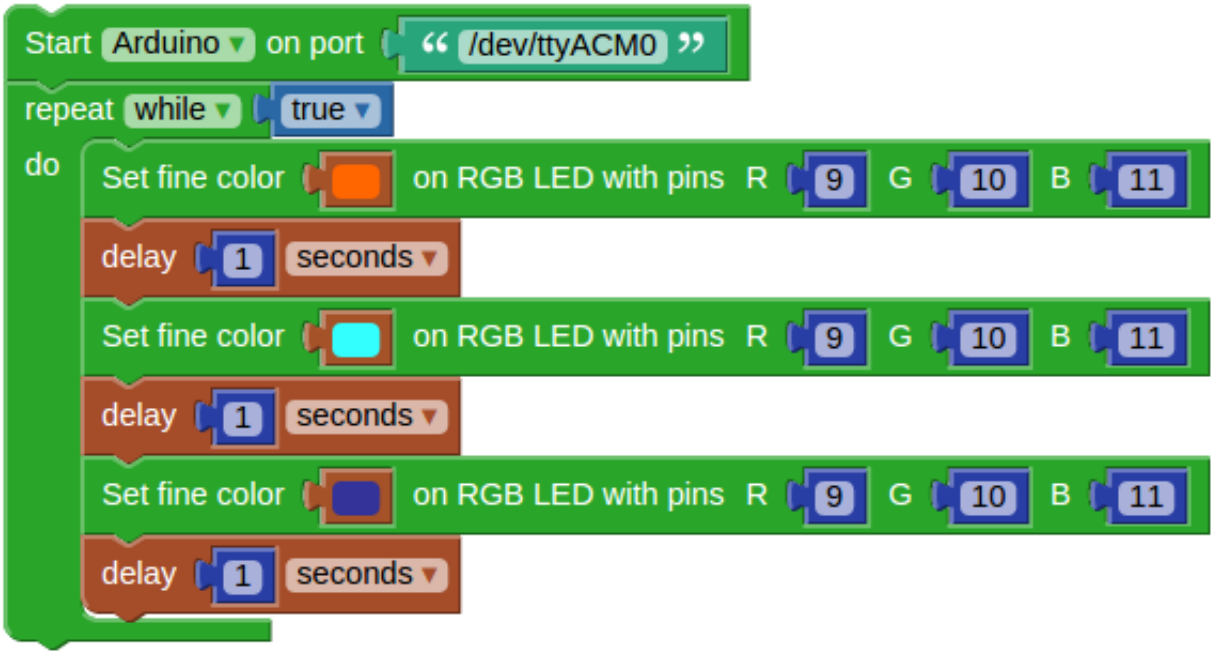


Figure 81: RGB LED application

Figure 81 shows the code that allows us to light the RGB LED in three different colors.

The first block you used is *Embedded/Arduino/Start Arduino on port []*. It imports the Firmata library and it sets up the communication between the Raspberry Pi and the Arduino. you have to specify the port the Arduino is connected on. Usually, the port is */dev/ttyACM0*. If case this does not work,

you have to search for it in the /dev directory on the Raspberry Pi.

Afterwards, you used the *while true* block so that the LED keeps on changing colors.

In order to set the color for the LED you used the *Embedded/Arduino/Set fine color [] on RGB LED with pins R[] G[] B[]* block. To use the block you have to specify the color you want to use and the pins to which the LED is connected. The block sends a message to the Arduino through the Firmata protocol and it commands the Arduino to set certain values on the specified pins.

Once you click the *Run the application* button, the project will get deployed on the board. However, this will take longer than usual because the *Arduino.ino* file has to be compiled and deployed on the Arduino. Once this is done, the LED will start changing colors every one second.

Tips & Tricks

If you want to change the code, you can skip the stage where the *Arduino.ino* file gets deployed. This is possible because once the Arduino runs the Firmata protocol, you only have to change the command passed to the Arduino. So you can just change the code without the need to use the *Embedded/Arduino/Start Arduino on port []* block again and wait for the deployment.

Social Lamp

In Wyliodrin boards can communicate, so you can control a project on a board, by means of another remote one. So to say, you can have access to a friend's board if he or she agrees and opens their boards to signals of yours. You can, for example, send a text message and print it on an LCD of their own and even control their board's pins.

This is how you will be able to create a social lamp by the end of this chapter.

Here is what you will have built at the end of the project. You will be able to light a different color on the RGB connected to one board and by pressing a certain button connected to another Raspberry. This way you can communicate with your friend in a secret language, a color language, for instance without the need for words.

This is going to be a more interesting and complex project. It is not a complicated one, but it will introduce many new notions. Let's go through it one step at a time.

What you need

- Two Raspberry Pi boards;
- Three buttons;

- One Arduino board;
- One RGB LED;
- Three 220 Ω resistors;
- Male-female jumper wires;
- Male-male jumper wires.

The Setup

To build the circuits you will only need to connect three buttons to a board (find the schematics in chapter *Use a button*) and an RGB LED to a separate board (instructions in *Light up a multicolor LED* chapter).

The Code

You will have two different applications, one for the buttons boards, that will be called the sender and one for the RGB LED board, which is going to be the receiver or the recipient.

For each category you will need specific blocks.

The sender

As difficult as it may seem, the code is very coherent and intuitive.

Let's begin with the communication blocks: Figure 82. As you ca see, you will have to fill in three spaces.

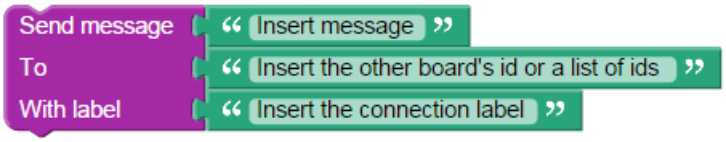


Figure 82: Blocks for the sender board

- The message is what you want the other board to receive.
- *To* will be the address of your recipient. You find that on Wyliodrin 's home page. Go to your board and, if you click on the settings, you will find Board ID. Copy and paste the receiver board's ID (Figure 83).
- The label will be your communication channel's identifier. You need to open this channel to let your board send signals to those who use the same identifier in the receiver block. In the label space write a word of your choice.

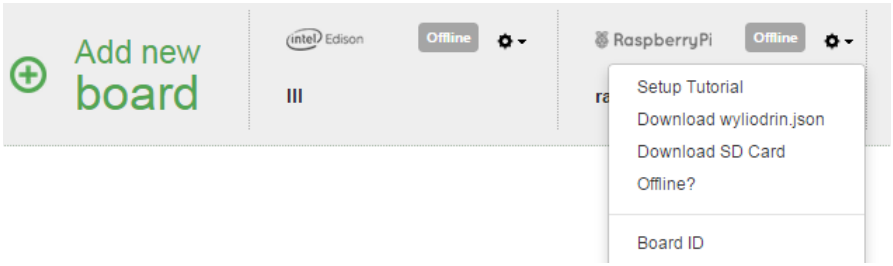


Figure 83: Board ID

The second step will be to define three variables where to store the intensity of the light of each color, from 0 to 255. They will be equal to 0 in the beginning.

To make the circuit responsive you will add a loop that will execute something each 100 milliseconds. Inside it, you'll have the following condition: if you press a certain button (write inside the block the pin where you've wired the first button), you increase the value in a corresponding color variable. It's tiring to increase it one by one so you can add a variable called step that

will add 10 each time you press the button. Another import aspect is not to overfill the channel of communication by sending too many signals. Thus, you will set one more Boolean variable that will register the changes, *updated*. Initially false, the variable will change when the button is touched.

The same course of action will be followed for every color.

Now, it's time to send the message, but only when you have a new update (when update is true).

The message itself will be a color given by the value you have for each of the primary colors you've set. In *Program* , under *Colors* you will find the block you need.

Don't forget to notify the application that the change has been made by turning *update* to false.

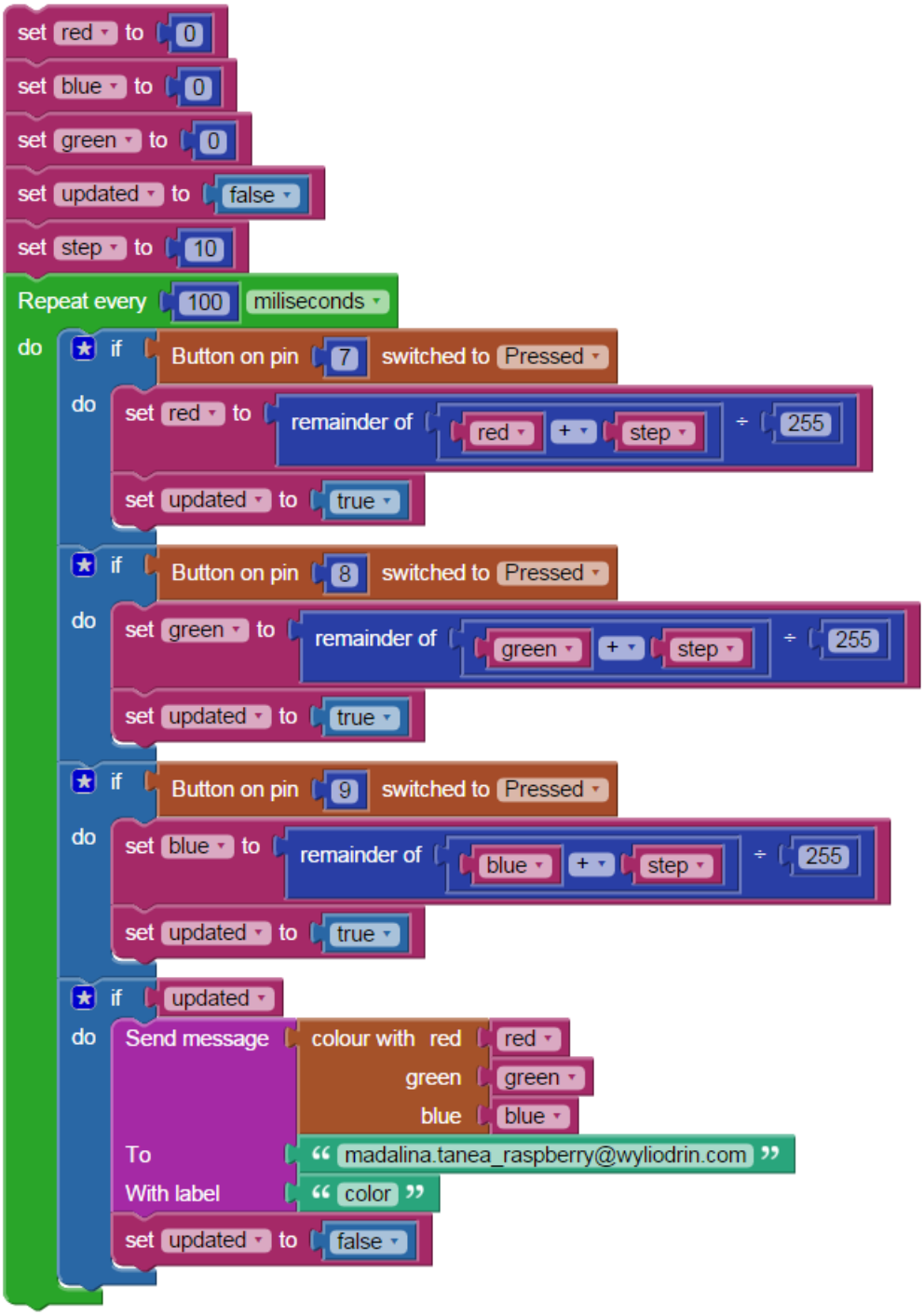


Figure 84: Social Lamp - sender with blocks

Concerning the Python code, there are a few new functions to explain.

Python

```
from wyliodrin import *
import json
from threading import Timer

red = None
blue = None
green = None
updated = None
step = None

pinMode (7, 0)
buttons = {}

def buttonSwitched(button, expectedValue):
  value = digitalRead (button)
  stable = True
  for i in range (100):
    valuenext = digitalRead (button)
    if value != valuenext:
      stable = False
  if stable:
    if button in buttons and value != buttons[button]:
      buttons[button] = value
      return value == expectedValue
    elif not button in buttons:
      buttons[button] = value
      return False
    else:
      return False
  return False
```

```
31
32 buttons[7] = digitalRead (7)
33 pinMode (8, 0)
34 buttons[8] = digitalRead (8)
35 pinMode (9, 0)
36 buttons[9] = digitalRead (9)
37 initCommunication()
38
39 def colour_rgb(r, g, b):
40   r = round(min(100, max(0, r)) * 2.55)
41   g = round(min(100, max(0, g)) * 2.55)
42   b = round(min(100, max(0, b)) * 2.55)
43   return '#%02x%02x%02x' % (r, g, b)
44
45 red = 0
46 blue = 0
47 green = 0
48 updated = False
49 step = 10
50 def loopCode():
51   global red, step, updated, green, blue
52   if buttonSwitched (7, 1):
53     red = (red + step) % 255
54     updated = True
55   if buttonSwitched (8, 1):
56     green = (green + step) % 255
57     updated = True
58   if buttonSwitched (9, 1):
59     blue = (blue + step) % 255
60     updated = True
61   if updated:
62     sendMessage('madalina.tanea_raspberry@wyliodrin.com', 'color',\
63     json.dumps(colour_rgb(red, green, blue)))
```

```
64      updated = False
65    Timer(0.1, loopCode).start()
66  loopCode()
```

After importing all the libraries, modules, initiating the variables and setting the buttons pins as outputs, you will create an array called *buttons*.There you will store the value read from each digital pin to which you've connected the buttons.

The next step in this code is to make sure you have a steady value for your button. To do this, you verify 100 times that the value you read initially from your button stays the same. In order to avoid a possible error, you check if the resulting button and its value is the same one you assigned to the array.

Before going forward with the code, you will have to open the communication channel to make the connection between the receiver and the sender possible.

The method defined as *colour_rgb* is the one which will mix the percentage in every primary color, as you give it to make a new color. Its arguments are the values you set for each of the colors red, green, blue. Therefore, if the button on pin 7, which is pressed, fulfills the conditions set by the method *buttonSwitched*, then you increase the value of *red*, but make sure it doesn't surpass 255.

If a change is made, you send a message to the receiver board's ID, with the label color and the color you composed as the message.

The Receiver

On the receiver's end you will need the respective block. It consists of two variables which have set values: message and sender. They will correspond

to the first application you've just built.

However, make sure as label you have the exact same word you wrote in the sender application, otherwise the connection won't be possible.

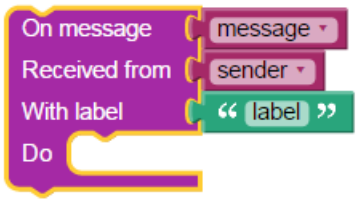


Figure 85: Blocks for the receiver board

You are going to use the PWM pins of the Arduino, so the first thing to do is to initiate the board and its port.

Obviously, you now have to add the action inside your receiver block.

You start by verifying if the sender is the expected one, by taking the other board's ID and, in the favorable case, choose from the Arduino category, the block that will set as fine color the one you composed in the other application. That will be your message.

Add the pins for each color from the RGB LED.

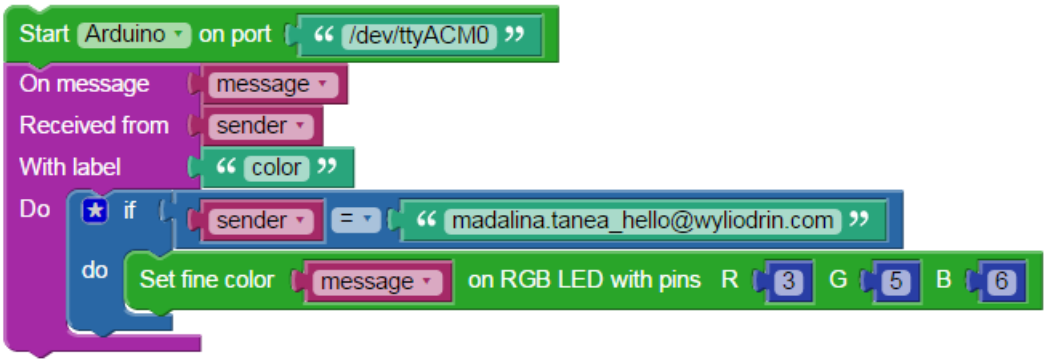


Figure 86: Social Lamp - receiver with blocks

As for the Python code, you have the explained Firmata function in the *LED Line* chapter.

The method *colorToRGB* will interpret the color it gets as a parameter, from a hexadecimal number.

The method *myFunction* is described in chapter *Control LED from Dashboard*.

Python

```
from pyfirmata import Arduino
from pyfirmata import util
from wyliodrin import *
import json
import struct

message = None
sender = None

def setBoard(boardType, port):
  if boardType == 'arduino':
    board = Arduino(port)
  else:
    board = ArduinoMega(port)
  return board
board=setBoard('arduino', '/dev/ttyACM0')
reader = util.Iterator(board)
reader.start()

def colorToRGB (color):
  return struct.unpack ('BBB', color[1:].decode('hex'))

def basic_color (color):
  value = 0
```

```
25    if color>=128:
26      value = 1
27    else:
28      value = 0
29    return value
30
31  pin_var = board.get_pin("d:3:p")
32  pin_var2 = board.get_pin("d:5:p")
33  pin_var3 = board.get_pin("d:6:p")
34
35  def myFunction(__sender, __channel, __error, __message):
36    global message, sender
37    sender = __sender
38    message = json.loads(__message)
39    if sender == 'madalina.tanea_hello@wyliodrin.com':
40      color = colorToRGB (message)
41      pin_var.write(color[0]/255.0)
42      pin_var2.write(color[1]/255.0)
43      pin_var3.write(color[2]/255.0)
44
45  openConnection('color', myFunction)
```

Smart Parking

This project will be a very interesting and practical one. Imagine you can build a parking yourself. How would you do it? Here's a suggestion. Starting from this example you can make a variation of similar projects.

At the end of this chapter you will have a smart parking. What does that mean? You will have a sensor activated barrier entrance, two sensitive parking lots connected to an LCD that will help monitor the number of available spots inside, an emergency fan that will start if there are gas leaks and a presence sensor which will turn on an LED to notify when there's someone in the car park.

You will need:

- Three grove light sensors
- One touch button
- One grove presence sensor
- One grove LED
- One grove servo motor
- One relay
- One Arduino board

- One grove shield for raspberry
- One Arduino grove shield
- One grove LCD

How to connect them?

First of all you need to place the two shields, one on the Raspberry Pi and the other one on the Arduino. You should know that thanks to the Raspberry Pi shield, now you can connect sensors directly to the Raspberry Pi and get their values. This is possible because the shield has analog to digital converters (ADCs).

- Connect one of the light sensors to the Arduino analog pin 0 (A0);
- Connect the two others to the pins A1 and A2 on the Raspberry shield;
- Also connect the gas sensor to A3 of the Raspberry;
- The touch button and gas will be connected to the digital pin D4 and D5 of the Raspberry;
- Connect the servo motor to the analog pin 5 of the Arduino;
- The LCD goes to to I2C pin of the Raspberry shield;
- The fan will be wired to the relay on pin D2 of the Raspberry .

How does it work?

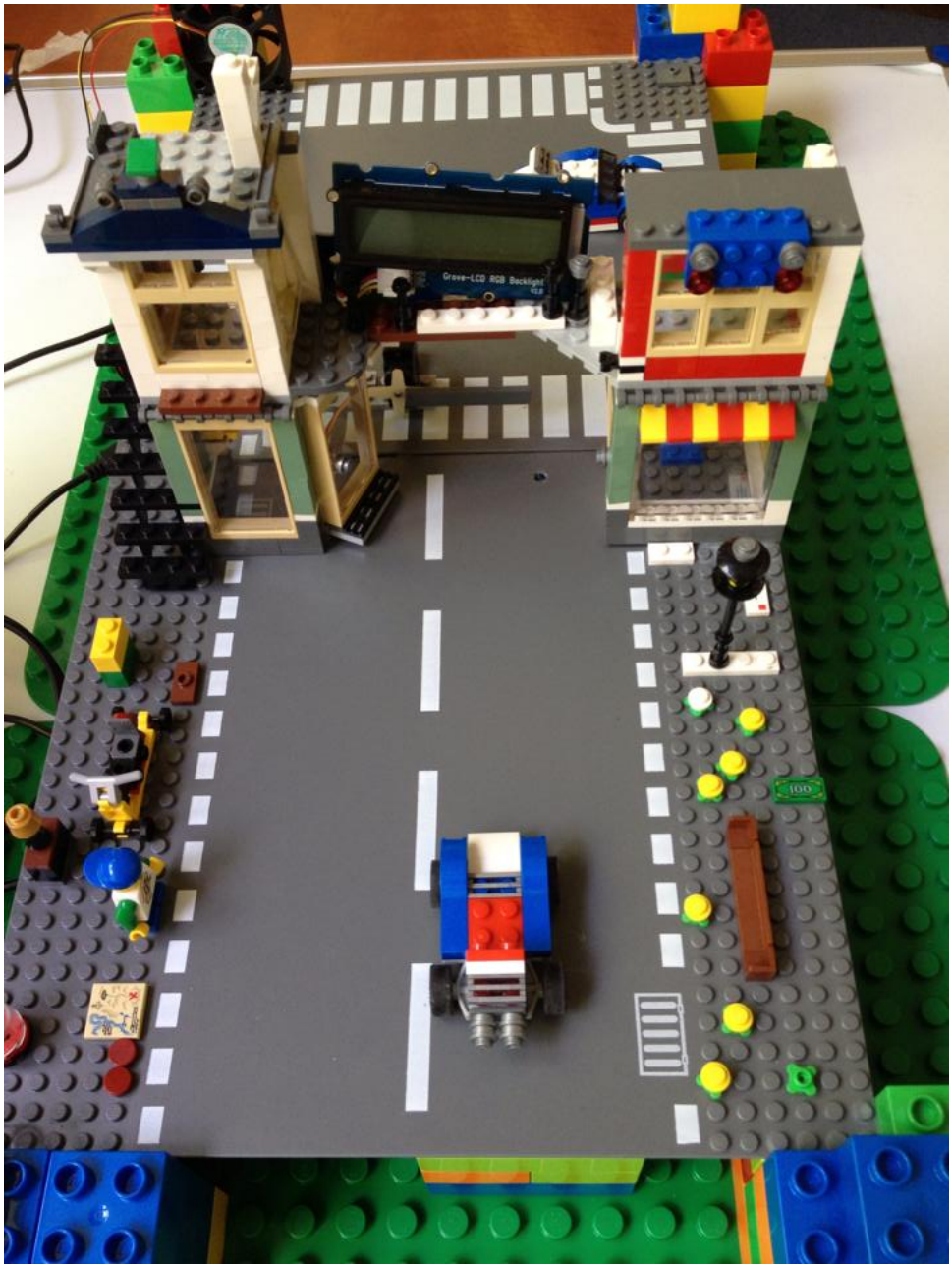


Figure 87: Smart Parking

You have two light sensors that will notify you when a parking spot is taken, as you can see in Figure 88

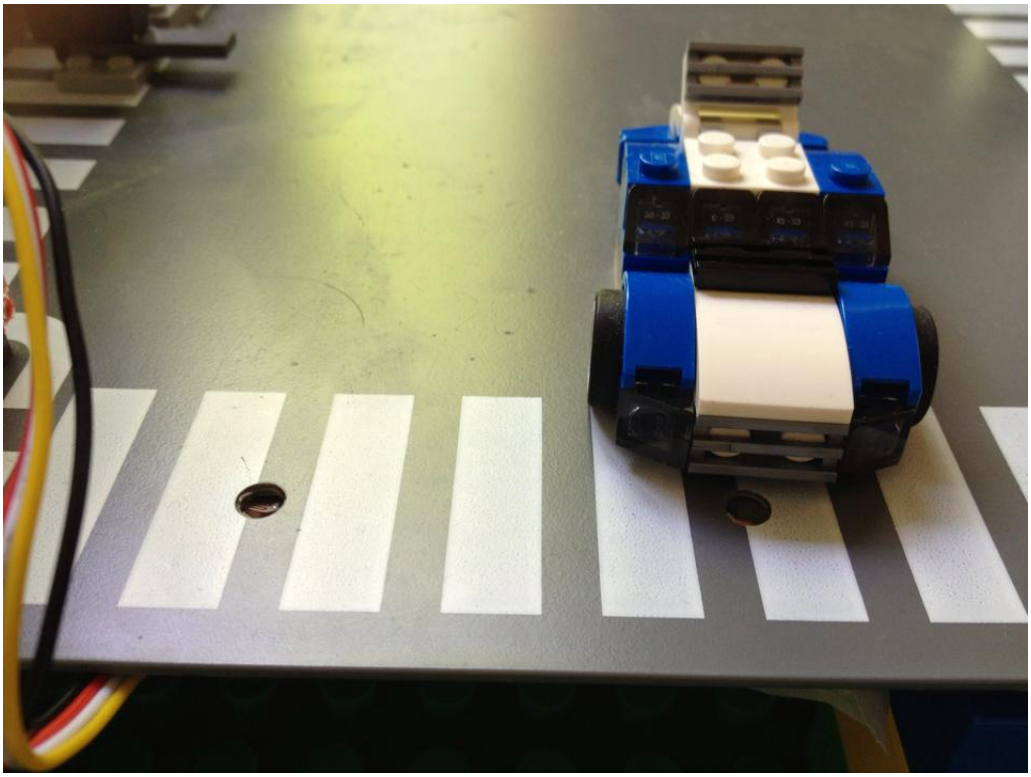


Figure 88: Two light sensors that mark the parking spots

The barrier will be accessed from the outside via the light sensor with the same function as the previous ones. From the inside, there is a touch button that will make the barrier go up. Figure 89

On the LCD you will have a message on the state of the parking lot

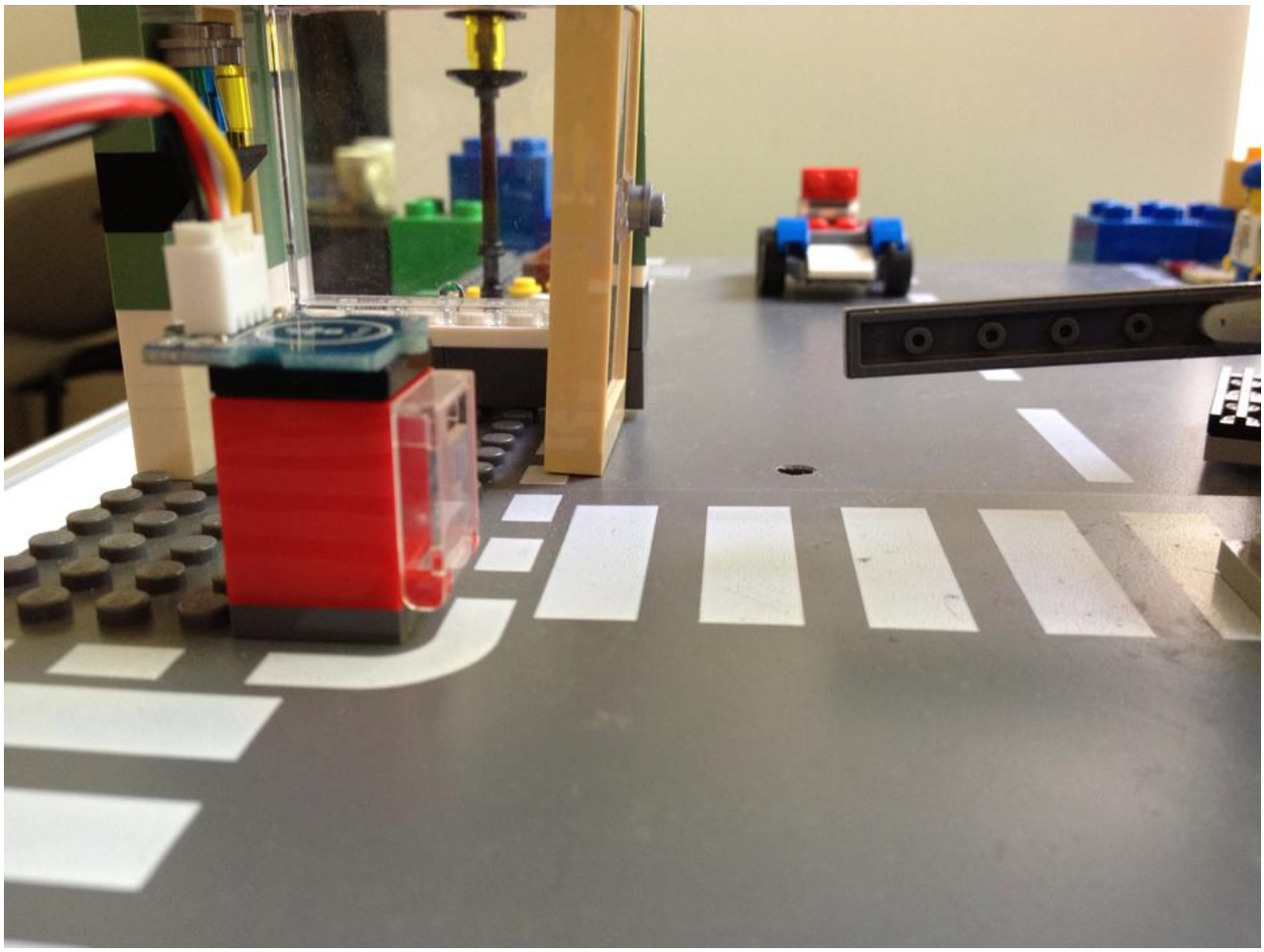


Figure 89: Touch button to raise the barrier

The gas sensor will turn on the fan in case the cars give off gases, the presence sensor will automatically turn on the LED (Figure 90)

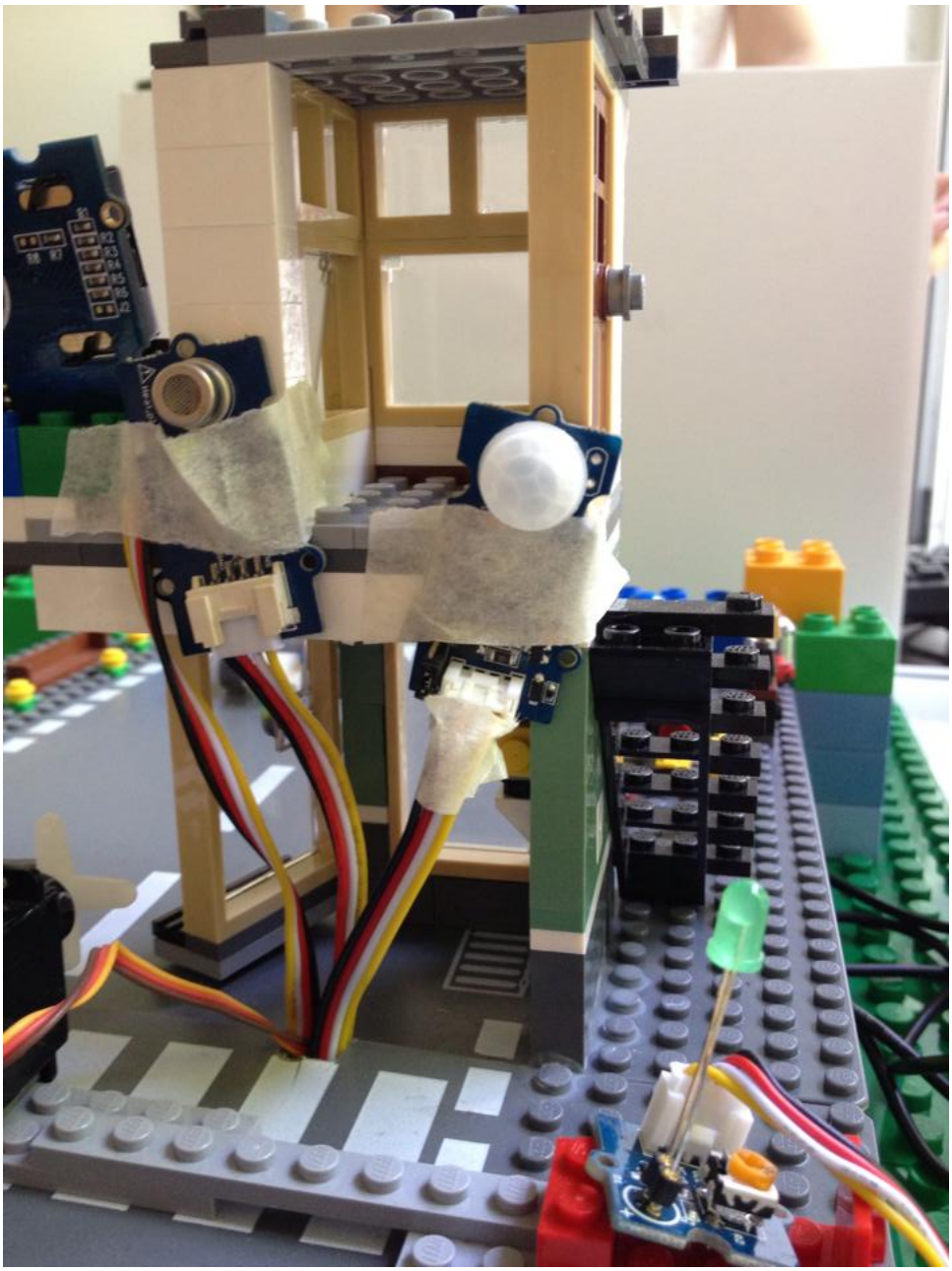


Figure 90: Gas sensor, Presence sensor plus LED

The code

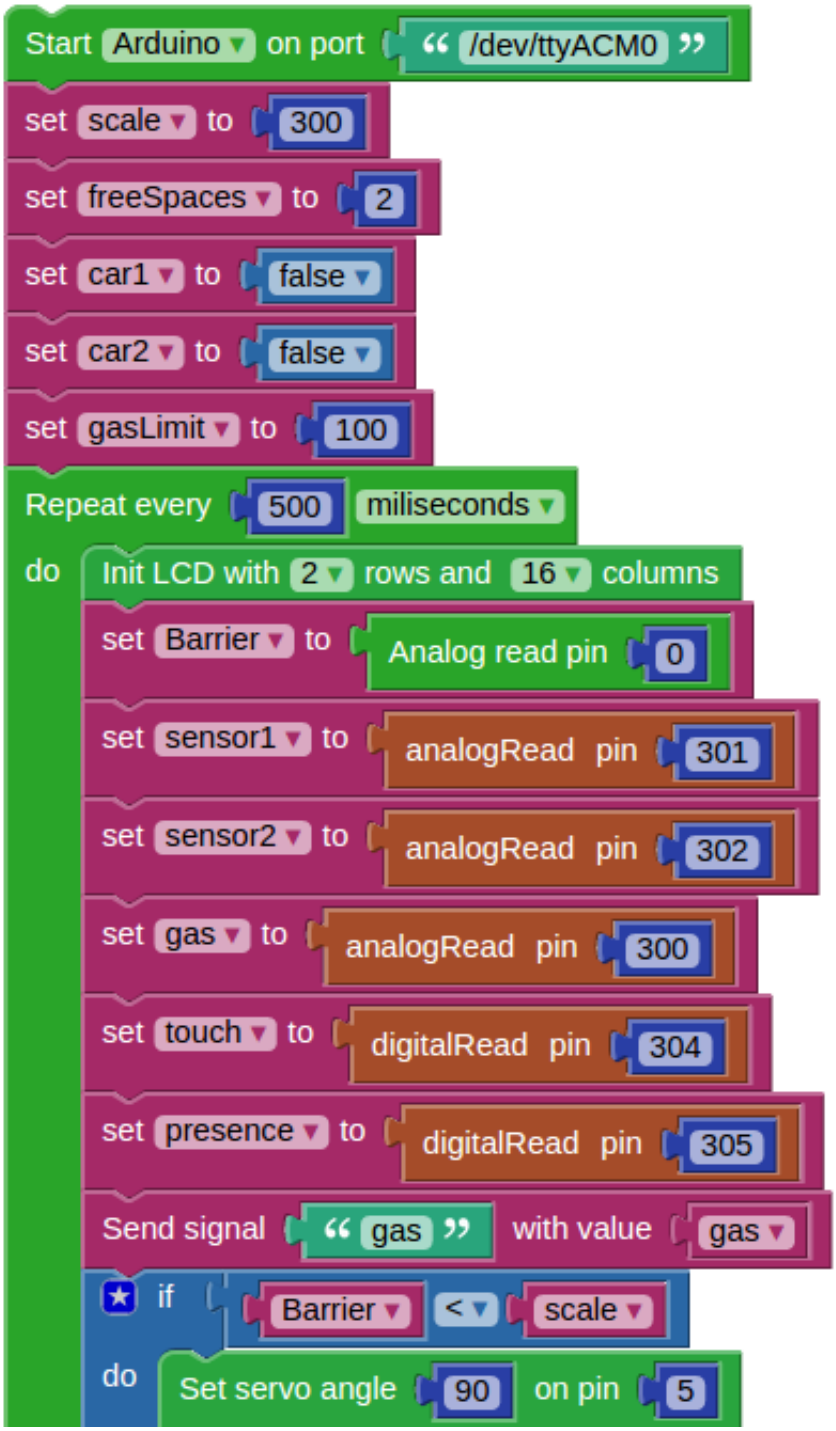


Figure 91: Initialisation of variables

The application is quite a long one, this is why we will take it step by step.

First you will have to set the variables which will store the values sent from the various sensors.

Please note that the servo which represents the barrier is connected to the Arduino, so you will need to use the appropriate blocks.

Because of the grove sensors, you will write for example *301* when you refer to pin 1, 302 for pin 2 and so on (Figure 91).

Now, you will need to adjust the limit at which the light is low enough to consider there is a car which occupies the parking space. This limit will be set from the dashboard using a slider (Figure 92). Thus, if the light is too strong or low, you will be able to adjust your sensors. The signal sent from the slider is called *light*.

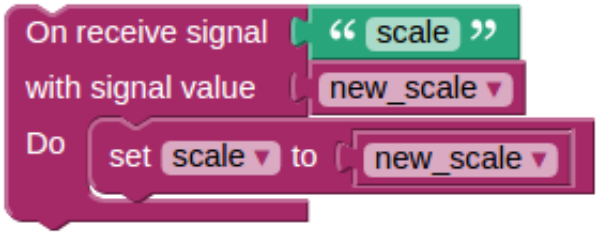


Figure 92: Light limit adjustment

For the barrier, if the light is lower than the set value, that means there is a car waiting to pass. The barrier will go up and after a while long enough not the fall over the vehicle, it will go down. Remember the barrier is controlled by a servo connected to the Arduino, so the block will be from the corresponding category. Once the barrier is up you will see the bar chart going up as it receives the *barrierup* signal. In addition, there is a custom graph that receives the same signal and it displays a picture of a barrier which is up.

You also need to control the barrier when you want to get out of the parking lot. To do that you have a touch sensor that lifts it up when touched (Figures 93 and 94).

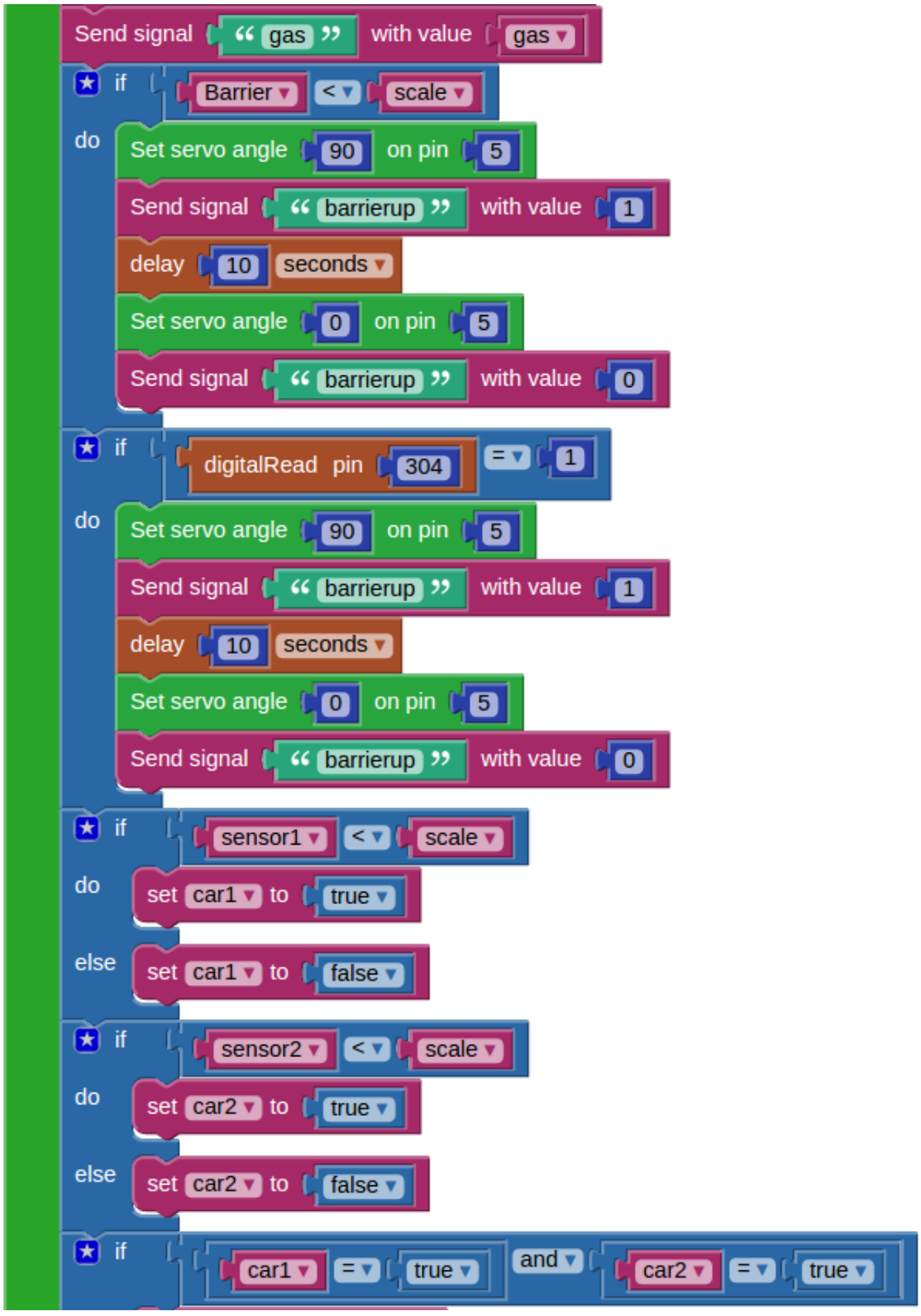


Figure 93: Count free parking sports

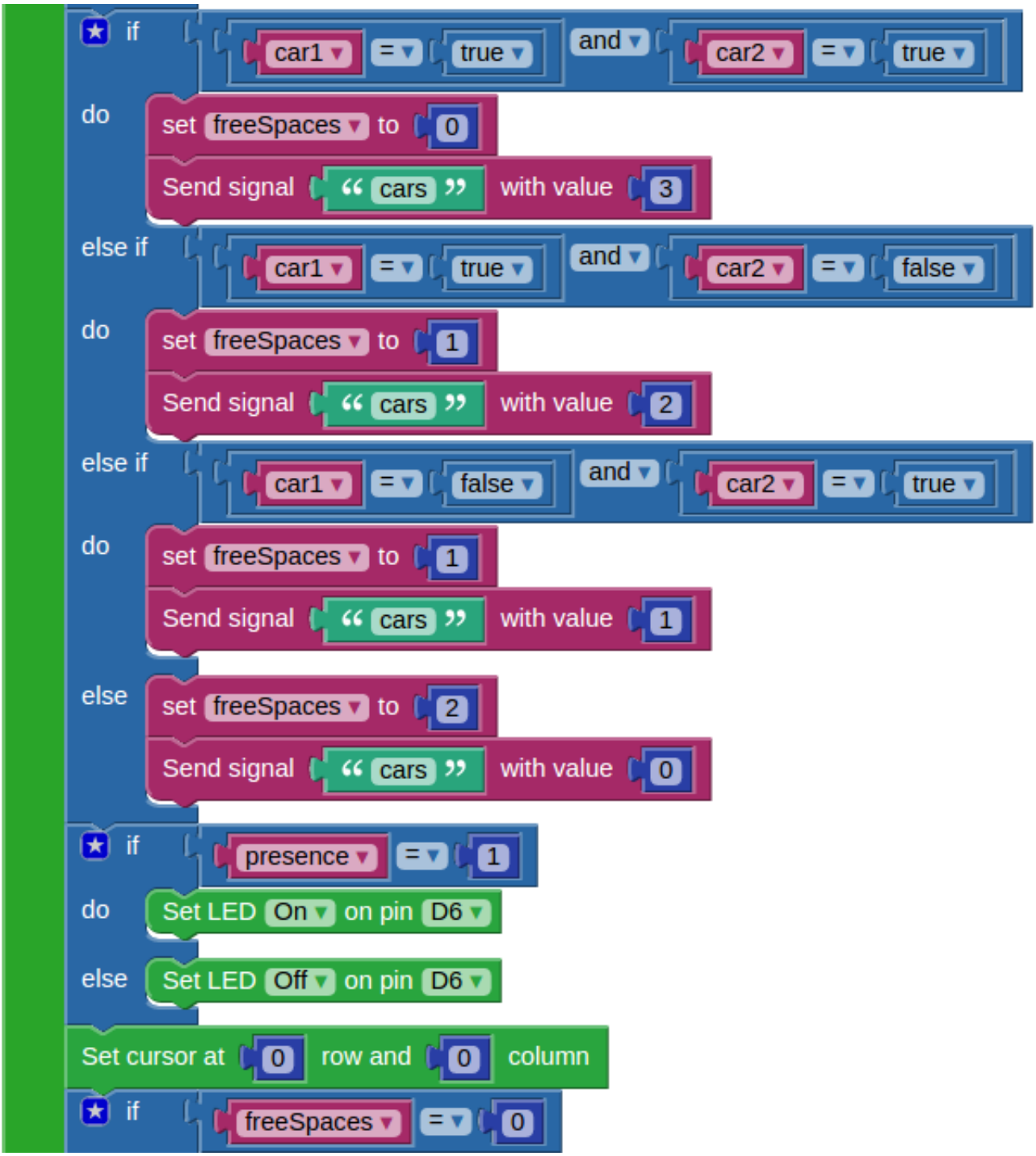


Figure 94: Get touch sensor data

Furthermore, the presence sensor will send a signal to a graph, in the dashboard and set on or off the LED. This can also be seen in Figure 94.

Initially, the LCD will show *Available: 2*. When one of the sensors notifies there is one busy space, the variable storing whether the parking spot is busy or not will change its value. Next you have to verify if which of the two spots are free and give the *freeSpaces* variable a value according to the present situation. This is also the moment when you will send the signal to the custom graph in order to display which parking place is free and which is not. In case the parking has all spaces full, the color of the LED will turn from green to red and the message displayed is going to be *Parking is full*. You will also see the car in the dashboard on the custom graph (Figure 95).

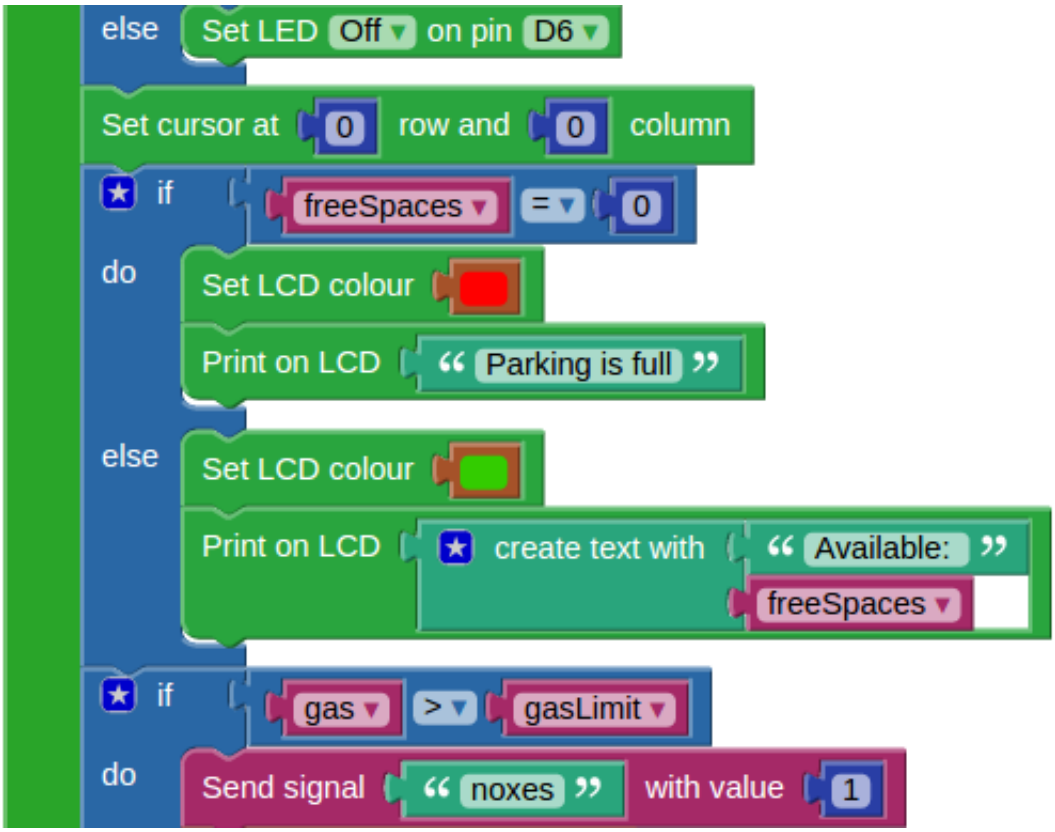


Figure 95: Display available spaces on the LCD

Next, you have a gas limit to compare to the value sent by the sensor. When it is surpassed, it will start the fan. You can also see the fan moving in the dashboard. You need to compare the value detected by the gas sensor to your limit and turn on the relay, which will start the fan. The value the gas sensor receives is displayed on a speedometer. What is more, you can add a custom graph which receives the *noxes* signal and displays a rotating fan when the case (Figure 96).

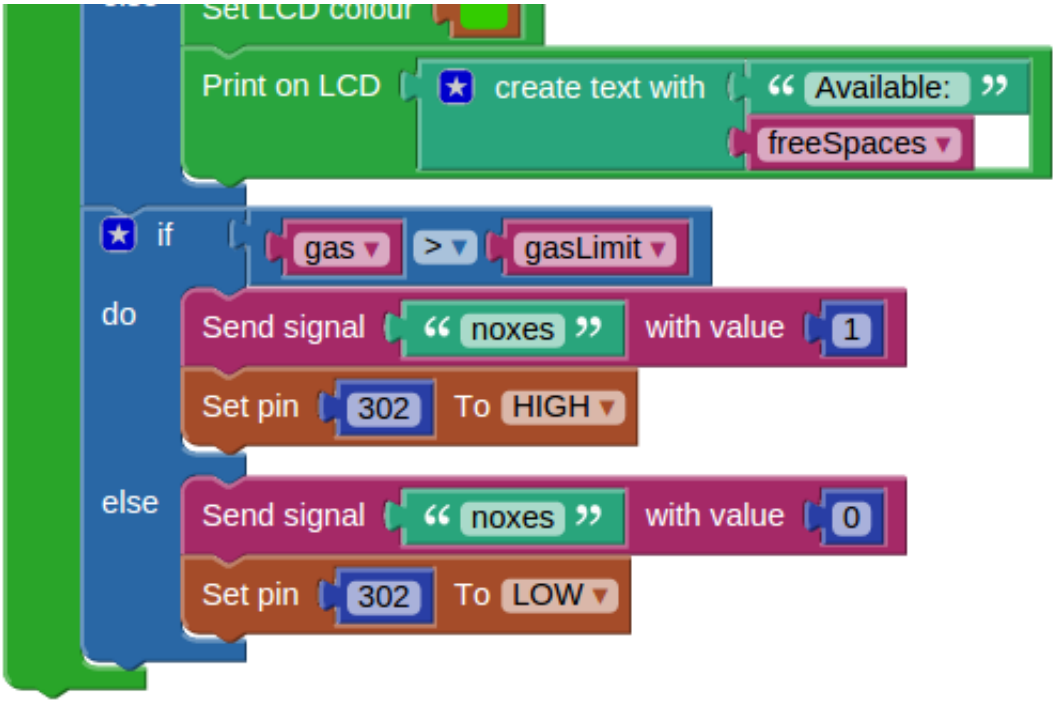


Figure 96: Monitor gases

You can also control the barrier from the dashboard with the help of a push button that sends the *barrier* signal to the application. In Figure 97 you can see the signal you are waiting for, the signal from the push button which lifts the barrier.

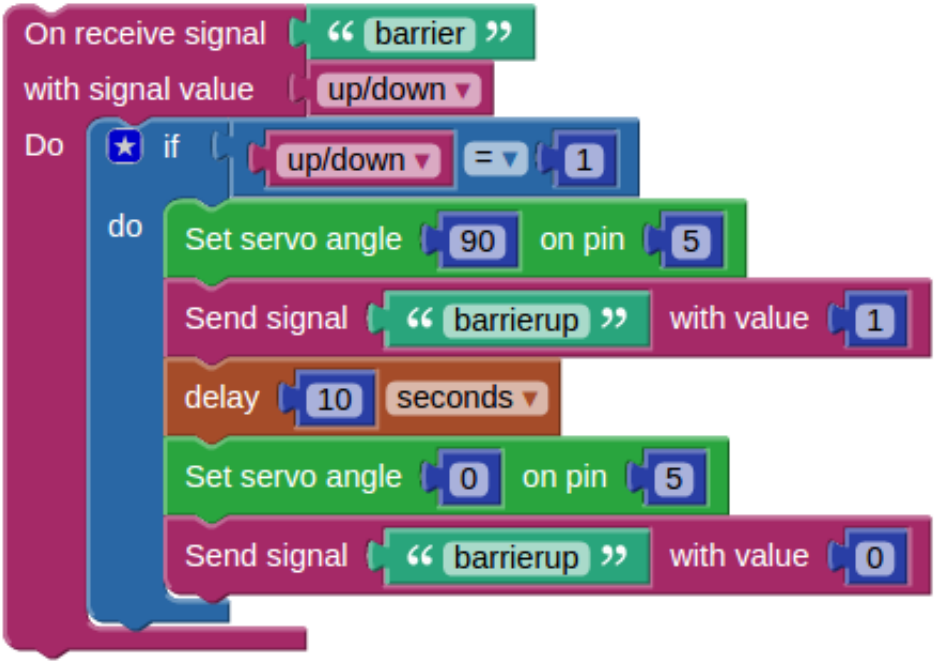


Figure 97: Barrier button

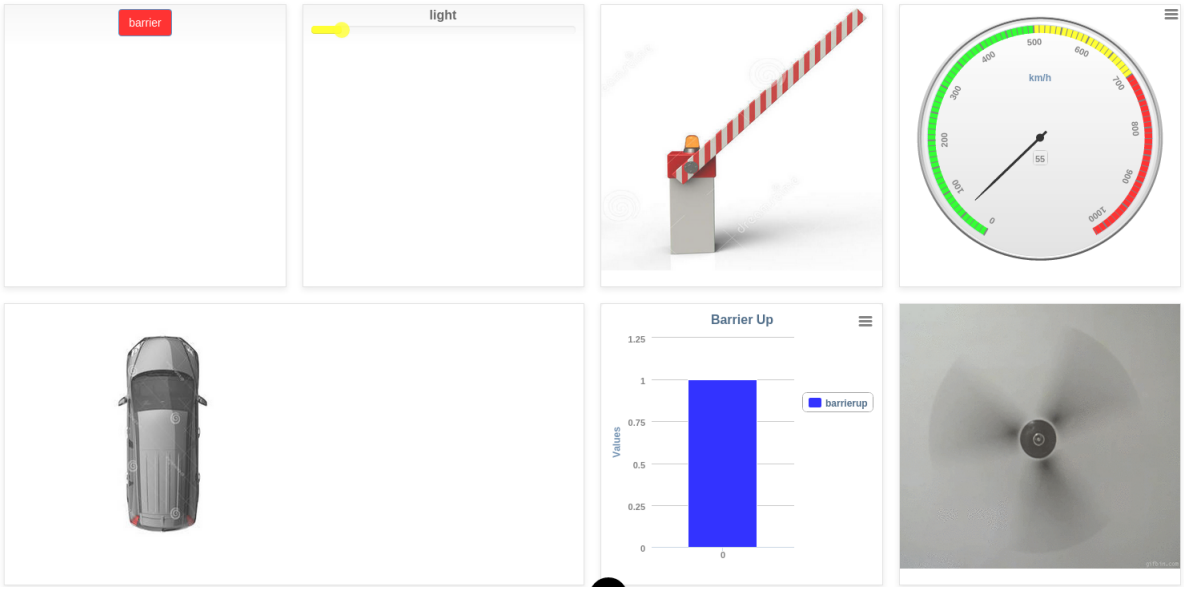


Figure 98: Smart parking dashboard

Picture 98 shows the dashboard at a certain moment. With all these graphs you can know at all time the status of your parking lot and you can also remotely control it.

Useful links

For the Custom graphs you will need some nice pictures representing the cars, the barrier and the fan. You can use these links:

- For the barrier graph:
  - `https://dl.dropbox.com/s/2ld85wzona372pf/barrier0.jpg?dl=0`
  - `https://dl.dropbox.com/s/l56lx86j6v4bm33/barrier1.jpg?dl=0`
- For the cars graph:
  - `https://dl.dropbox.com/s/9pbpd2elff2ajpg/cars0.jpg?dl=0`
  - `https://dl.dropbox.com/s/clh2o08lee50yem/cars1.jpg?dl=0`
  - `https://dl.dropbox.com/s/tm4lyc8n747dp3k/cars2.jpg?dl=0`
  - `https://dl.dropbox.com/s/ltbetinr59j685t/cars3.jpg?dl=0`
- For the noxes graph:
  - `https://dl.dropbox.com/s/49dba9lfnxv5sub/spinner0.png?dl=0`
  - `https://dl.dropbox.com/s/5kp8givgeywmjqr/spinner1.gif?dl=0`

Part III

Reference

Visual Programming

A visual program language is a programming language that lets users create programs by using graphic elements.

This chapter presents the specific blocks that were used in this book.

Program

Functions

Functions

Creates a function.

You can have functions that return something (Figure 99) of functions that do not return anything (Figure 100).

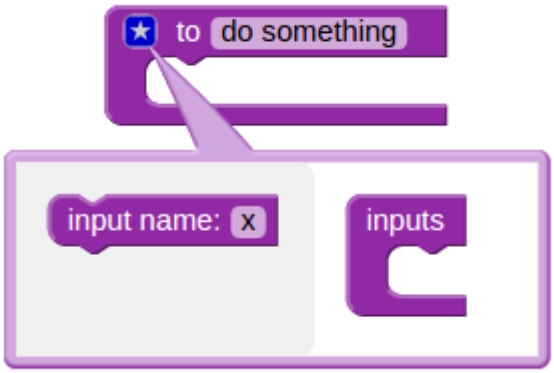


Figure 99: Functions that do not return Block

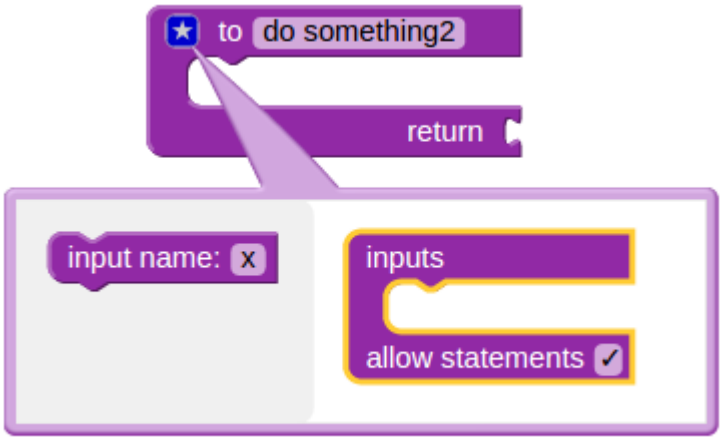


Figure 100: Functions that return Block

Loops

Repeat while [](Figure 101):

Figure 101: Repeat While Block

This block will repeat your block as long as your condition is true.

Count with (Figure 102):

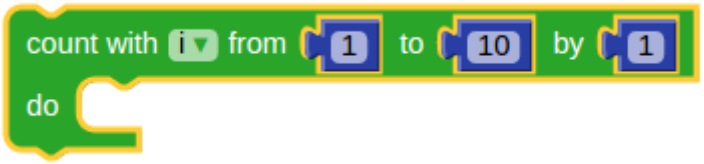


Figure 102: Count Block

Advances a variable from the first value to the second value by the third value.

Repeat every [] (Figure 103):

Figure 103: Repeat every [] Block

It will repeat the statements every set time interval.

Repeat [] times (Figure 104):

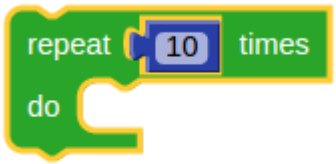


Figure 104: Repeat [] times Block

It runs the code in its body the specified number of times.

Logic

True (Figure 105):

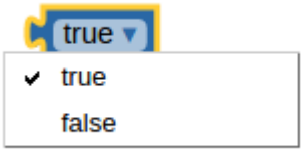


Figure 105: Boolean Block

Can be used as a boolean value: true or false.

Comparisons (Figure 106):

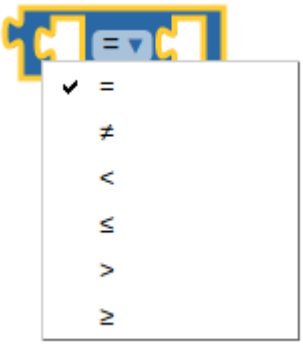


Figure 106: Comparison Block

Takes two inputs and returns true or false depending on how the inputs compare with each other.

Logical Operators (Figure 107):

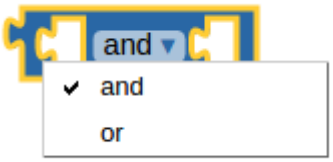


Figure 107: Logical Operators Block

The *and* block will return true only if both of its two inputs are also true.

The *or* block will return true if either of its two inputs are true.

If/else (Figure 108):

Figure 108: If/Else Block

If the expression or variable being evaluated is true, then do the first block of statements. If it's false do the second block of statements.

Variables

Set item to [] (Figure 109):

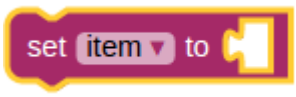


Figure 109: Set Block

Assigns a value to a variable, creating the variable if it doesn't exist.

Item (Figure 110):

Figure 110: Item Block

This block provides the value stored in a variable.

Lists

Create list (Figure 111):

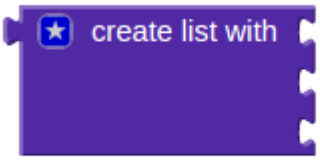


Figure 111: Create list Block

Creates a list of items.

Numbers And Maths

Truncate (Figure 112):

Figure 112: Truncate Block

Round a number to the nearest integer.

Map (Figure 113):

Figure 113: Map Block

It will map the value.

Number (Figure 114):

Figure 114: Number Block

Reminder of (Figure 115):

Figure 115: Reminder Block

The Remainder or Modulus Operator returns the remainder of a division between two numbers.

Change (Figure 116):

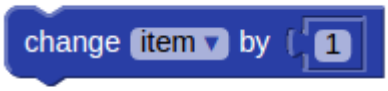


Figure 116: Change Block

Increments a particular variable by a set number.

Text

Text (Figure 117):

Figure 117: Text Block

Create text (Figure 118):

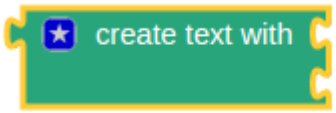


Figure 118: Create text Block

It creates a piece of text by joining together any number of items.

Screen And Keyboard

Print on screen (Figure 119):

Figure 119: Print on Screen Block

Print a text on the screen and put the cursor on a new line.

Timing

Delay [] (Figure 120) :

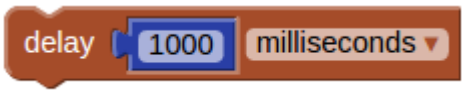


Figure 120: Delay Block

This will delay the program execution for a period of seconds, milliseconds or microseconds.

Date And Time

Get hour (Figure 121):

Figure 121: Get hour Block

Gives either the current hour/minute/second.

Social

Twilio

Twilio Setup (Figure 122):

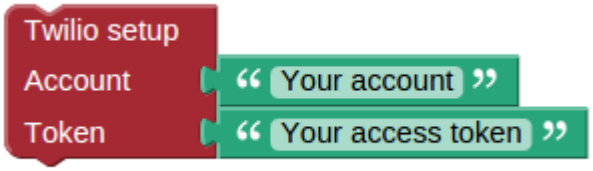


Figure 122: Twilio Setup Block

Make a call (Figure 123):

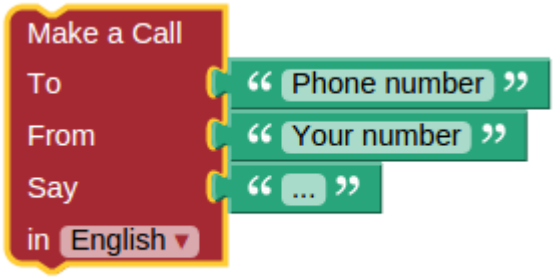


Figure 123: Make a Call Block

Make a call with music (Figure 124):

Figure 124: Make a Call with music Block

Board Communication

Send message (Figure 125):

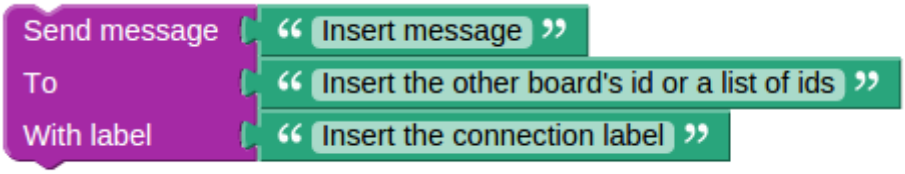


Figure 125: Send message Block

Sends a message to another board using a particular label.

On message (Figure 126):

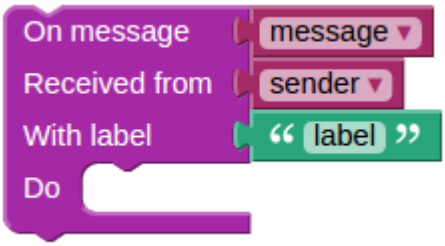


Figure 126: On message Block

Tests that a message variable is received from a particular other board using the correct label and enables further actions to be carried out.

Pin Access

analogWrite (Figure 127):

Figure 127: analogWrite Block

Sets a value between 0 and 255 on the pin.

analogRead (Figure 128):

Figure 128: analogRead Block

Reads a value between 0 and 1023 from the pin.

digitalRead (Figure 129):

Figure 129: digitalRead Block

Reads a value from one of the digital pins and returns 0 or 1.

digitalWrite (Figure 130):

Figure 130: digitalWrite Block

Sets value 0 or 1 on the pin.

Peripherals

Pins

Set pin [] to LOW/HIGH (Figure 131):

Figure 131: Set pin Block

Set a digital value on a pin. You have to add the pin number and the value you want: *HIGH* means 1 and *LOW* means 0.

Shift out (Figure 132):

Figure 132: Shift Out Block

Writes a value serially on a pin. This will send out a value bit by bit to the data pin and generate a clock signal on the clock pin. It will start with the LSB (Least Significant Bit). Usually this function is used with shift registers.

LED

Set On/Off LED on pin [] (Figure 133):

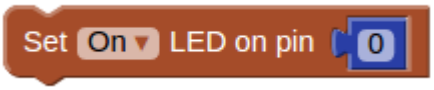


Figure 133: Set On/Off LED Block

You have to add the pin number and the value you want.

This block turns the LED connected to the pin number you added on or off .

Set basic color on RGB LCD (Figure 134):

Figure 134: Set Fine Color Block

Sets the LED with the specified pins for RGB at the selected color. You can select only base color.

Sets the LED with the specified pins for RGB at the selected color. You can select any color. This block can be used only if the pins used are PWMs.

Arduino

Arduino (Figure 135):

Figure 135: Start an Arduino

Starts an Arduino using the specified port. This needs to be placed before any other Arduino statement.

analogWrite (Figure 136):

Figure 136: analogWrite Block

Sets a value between 0 and 255 on the Arduino pin.

analogRead (Figure 137):

Figure 137: analogRead Block

Reads a value between 0 and 1023 from the Arduino pin.

digitalRead (Figure 138):

Figure 138: digitalRead Block

Reads a value from one of the Arduino digital pins and returns 0 or 1.

digitalWrite (Figure 139):

Figure 139: digitalWrite Block

Sets value 0 or 1 on the Arduino pin.

Set basic color on RGB LCD (Figure 140):

Figure 140: Set Fine Color Block

Sets the LED with the specified Arduino pins for RGB at the selected color. You can select only base color.

Set fine color on RGB LCD (Figure 141):

Figure 141: Set Fine Color Block

Sets the LED with the specified Arduino pins for RGB at the selected color. You can select any color. This block can be used only if the pins used are PWMs.

7 Segment Display

Set 7-seg (Figure 142):

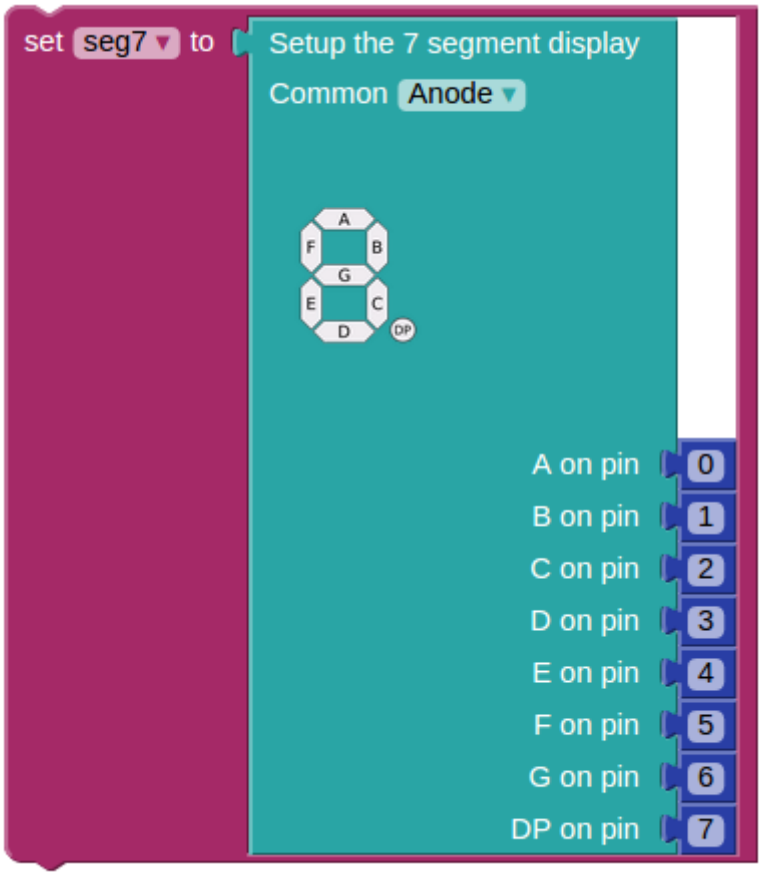


Figure 142: Set 7-segment Block

Initialize the 7-segment display.

Display on 7-seg (Figure 143):

Figure 143: Display on 7-seg Block

Displays a number on the 7-segment display.

LCD

Init LCD (Figure 144):

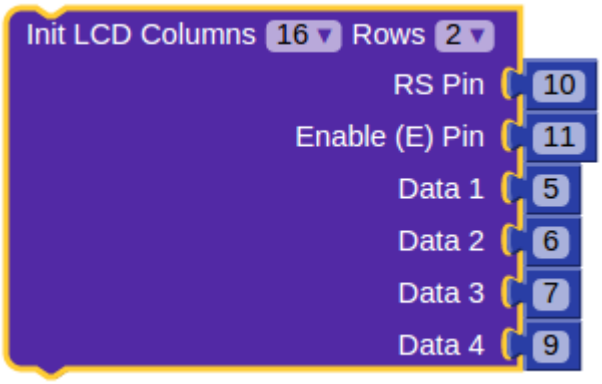


Figure 144: Init LCD Block

Initializes the LCD with 1, 2 or 4 rows and 16 or 20 columns.

Print on LCD (Figure 145):

Figure 145: Print on LCD Block

Prints the text you typed on the LCD.

Sensors

Buttons

Button on pin [] switched to (Figure 146):

Figure 146: Button Block

Shows the change of state of the respective button to Pressed/Released.

Internet

Sevices

Weather

Get temperature from [] (Figure 147):

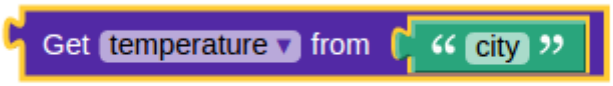


Figure 147: Get Temperature Block

It returns the temperature from a city.

Signals

Send signal (Figure 148):

Figure 148: Send Signal Block

It will send a signal with a value to a graph on the dashboard.

Receive signal (Figure 149):

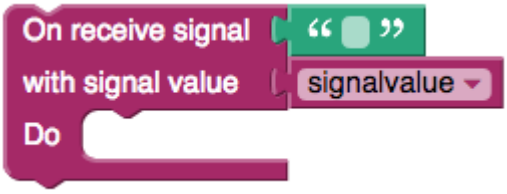


Figure 149: Receive Signal Block

When a signal with is received, the statements inside the block are executed. The signalvalue variable is set to the received signal value. This is used for switch and slider.

Multimedia

Load audio stream (Figure 150):

Figure 150: Load audio Block

Returns an Audio Stream variable. Most of the other functions request this

type of variable as a parameter. The Audio Stream can be created from a file, in this case the second parameter will be the file's name, or from a URL and the second parameter will be the address.

Play audio stream (Figure 151):

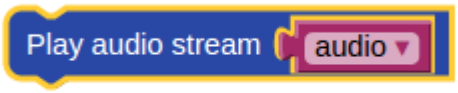


Figure 151: Play audio Block

Plays an Audio Stream.

Audio stream is playing (Figure 152):

Figure 152: Audio stream is playing Block

Verifies whether the player is playing.

Part IV

Appendix

Resistor Color Code

Every resistor has colored bands. Those bands and colors are not random at all, they help to identify the specifications of the resistor.

Here is how you can see the value of a resistor depending on its colored bands.

Every color represents a different number.

Here is the table of all the colors and numbers:

Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Grey	8
White	9

Counting from right to left, the second color is the multiplier. Digits of the first colors must be multiplied with the number of this color.

Black	1
Brown	10
Red	100
Orange	1000
Yellow	10000
Green	100000
Blue	1000000
Gold	0.1
Silver	0.01

And the last color is the tolerance. Tolerance is the precision of the resistor and it is a percentage.

Brown	1
Red	2
Gold	5
Silver	10
Nothing	20

There are a lot of programs that can calculate the value of a resistor but if you do not have access to the Internet and you need to know a certain value you can use the table from Figure 153.

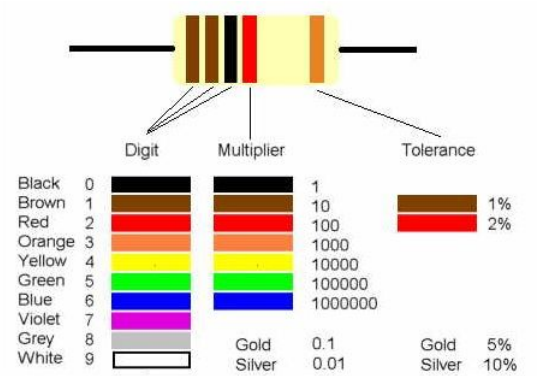


Figure 153: Resistor Color Code

Examples:

1.Yellow and green represent 4 and respectively 5. They represent the first and second digits, so you will have 45.

The third band is orange. As a multiplier, it is x 1000, so you will calculate 45 x 1000 thus, the resistance is 45,000 Ω.

The forth band, silver, represents the tolerance, so the final expression of the resistance is 45,000 +/- 10Ω (Figure 154)

Figure 154: Resistor Color Code

2. A resistor colored Orange-Orange-Black-Brown-Violet would be 3.3 kΩ with a tolerance of +/- 0.1(Figure 155)

Figure 155: Example 2

Manually Deploy a Project On the Raspberry Pi

There might be the case that you do not want to connect your board to Wyliodrin . In this case you cannot automatically deploy the project by using the platform. You have the possibility to manually run your projects on the board while using Wyliodrin only to write and store the applications.

You simply need to follow these steps in order to get the application from the cloud to your board and get it started:

1. Create the application using the Wyliodrin IDE.

2. Download the application on your computer. To do this, select the *Properties* button of the desires app and hit *Download application*.

3. Copy the application on the Raspberry Pi. You can use the SD Card in order to achieve this, but there are also other methods. You can use whichever you prefer.

4. Connect to the board via a terminal. You can use SSH, for example.

5. Install libwyliodrin on the Raspberry Pi. You can download the library from github (`https://github.com/Wyliodrin/libwyliodrin/tree/master`) and follow the instructions to compile it.

6. Go to the project's folder.

7. Run the *make run* command.

Raspberry Pi Does Not Appear Online

In case your Raspberry Pi does not appear online on the Wyliodrin platform, please try the following steps. After each step, check if the board becomes online.

1. Check that all the cables (power, Ethernet) are properly connected to the board. If you have a WiFi adapter, make sure is is compatible with your board.

2. Check if your power adapter is string enough and has the right voltage, the Raspberry Pi requires at least 1A.

3. Reset the board (plug out the power cable from the board and plug it in again).

4. Logout and login back.

5. Make sure you have the latest Wyliodrin SD Card Image for your board. You can download it by clicking the options button right next to the board's name and selecting *Download SD Card*.

6. Download *wyliodrin.json* once again and write it to the SD Card. Make sure the name is exactly *wyliodrin.json*. You can find the file by clicking the options button next to the board's name and selecting *Download*

wyliodrin.json .

7. Remove the board and add it again. Please make sure to download *wyliodrin.json* again and write it to the SD Card.