Ghent University

Cloud Applications and Mobile (E736010)

Lab 1: Sensors and Actuators

Professor:
Prof. dr. ir. Sofie Van Hoecke

Assistants:
Ing. Thibault Blyau
Dr. Jennifer B. Sartor

2024 - 2025

Table of contents

1 Introduction

We will build an IoT-stack (not to be confused with the IoT-stack developed at IDLab), shown in Figure 1 and Figure 2, in the following lab sessions using a bottom-up approach. The sensors and actuators connected to a Raspberry Pi will collect data, which will be fed to microservices in the cloud, which will then be read by an app on a smartphone. In this lab we will focus on the bottom-most layer, or those boxes in the green "Sensors" box in Figure 2.

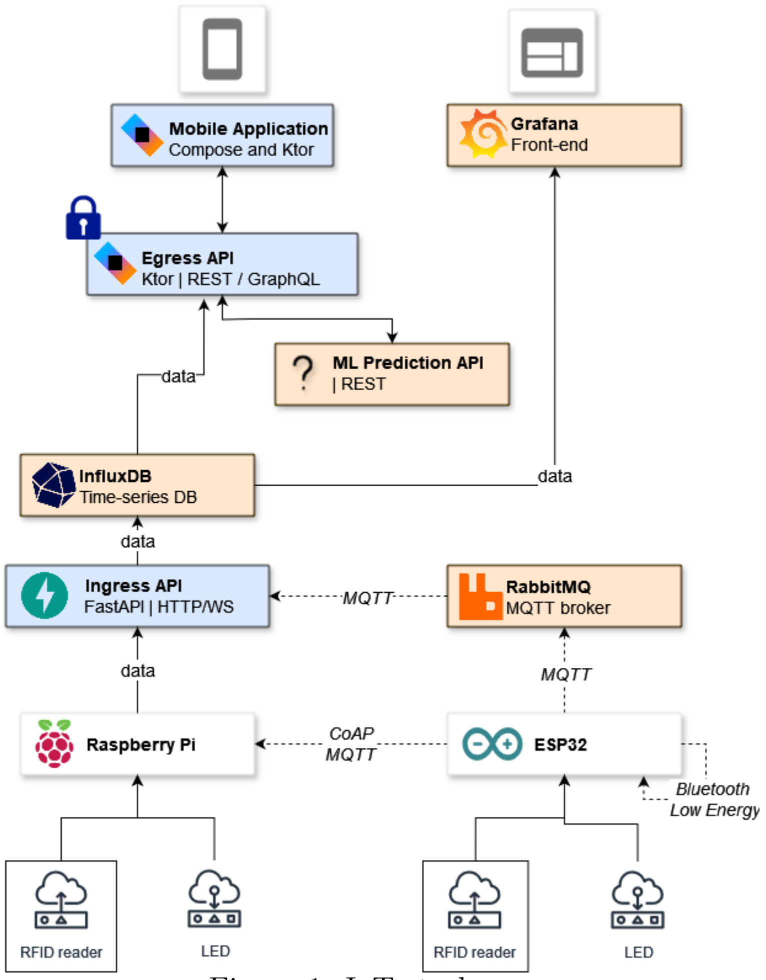


Figure 1: IoT-stack

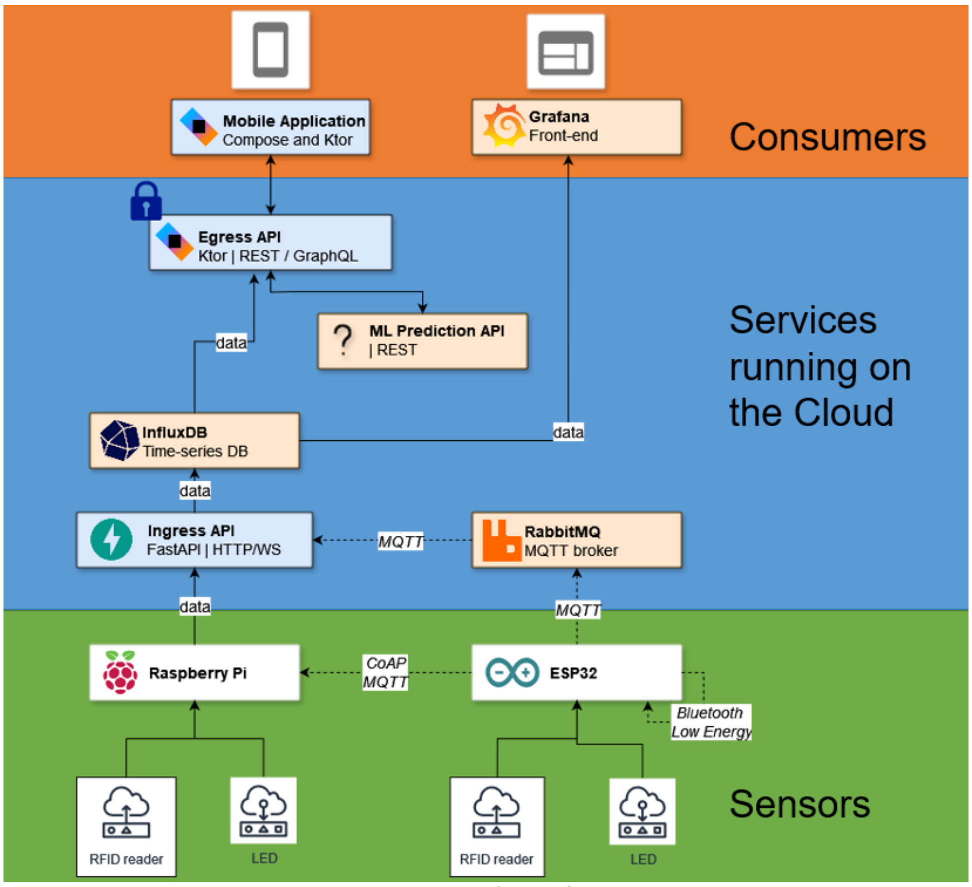


Figure 2: IoT-stack with categories

In this session we will first set up our RPi and make sure we can connect to it. We will then learn how to control an LED and get readings from a sensor (RFID tags read by an RFID reader RC522) with a Raspberry Pi (RPi) and an ESP32 microcontroller. The LED is controlled by running a program on either the RPi or the ESP32. The RFID tag will be read and then output to either the console (RPi) or the serial interface (ESP32).

2 Goals

1. Make an SSH connection with a Raspberry Pi
2. Make a circuit for a sensor and an actuator
3. Control a sensor and an actuator with an RPi
4. Control a sensor and an actuator with an ESP32

3 Material

3.1 Own material

- Laptop & charger
- Smartphone & charger (optional until Lab4)
- Ethernet cable (optional)
- **Micro-USB cable for the ESP32**

3.2 Borrowed material

3.2.1 Material available in classroom

- Screen with HDMI
- HDMI cable
- Keyboard
- Mouse

3.2.2 Borrowed material handed out

- Raspberry Pi 3 Model B(+)
- EU Power Supply (2.5A 5.1V)
- SD-card (16 GB)
- Resistors (3x 1KΩ or 3x 1.2KΩ)
- ESP32 microcontroller (WROOM JOY-iT or DOIT)
- RFID reader RC522
- RFID tags
- Web of Things sensorkit
  - Breadboard
  - LEDs
  - 10K Ohm Resistors
  - 330 Ohm Resistors
  - M/M Jumper Wires
  - M/F Jumper Wires
  - (Humidity and Temperature Sensor (DHT22))
  - (PIR Motion Sensor)

4 Software to Install

- Via the Jetbrains Toolbox App: PyCharm (**Professional Edition**) (Free with Ghent University e-mail when you register here.)
- Arduino IDE

5 Configuration

5.1 Starting with the RPi

In this section, you will learn how to set up and configure your very own IoT device. By the end of this section, you'll have all the tools at hand to be able to program it and implement all the concepts presented in the next lab sessions.

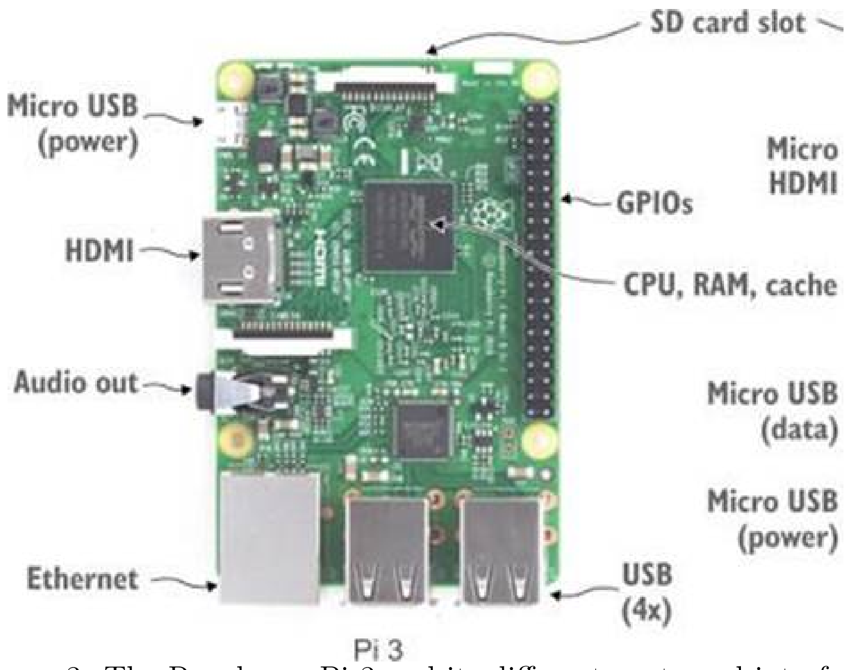


Figure 3: The Raspberry Pi 3 and its different ports and interfaces

We will use the **Pi 3** as it offers all the required connectivity out of the box and doesn't require any soldering to connect sensors and actuators to the general-purpose input/output (GPIO) pins. The Pi 3 boasts a quad-core 1.2 GHz CPU, 1 GB RAM, a micro-SD slot, and a Broadcom VideoCore IV graphic unit. In terms of connectivity, the Pi 3 has four USB ports (and a Micro USB, which is used to power it), an HDMI port, a 3.5mm jack, an Ethernet connector, and 40 GPIO pins. Finally, unlike its predecessors, the Pi 3 also offers out-of-the-box Wi-Fi and Bluetooth connectivity, making it a fully WoT-ready device.

In this lab session we will focus on setting up the Pi to make it fit for the WoT.

The operating system has already been installed for you. It's the Raspberry Pi OS (previously called Raspbian) and is essentially a port of the Debian Linux system tailored to fit the needs of the Pi and its users. The advantage of this OS is that it has been widely used and tested on the Pi and is easy to install and customize; therefore, it provides a stable and popular operating system to build on.

5.2 Setting up the operating system

Plug the **HDMI cable** into a **screen**, a USB **mouse** and **keyboard** into the USB slots, and finally the **micro USB** cable to **power** (see Figure 3 to find the right ports).

The OS should now boot. When you start it for the first time, the Pi boots up with the X Window graphical environment.

The wizard asks you to:

- **Set your Country**: Country, Language and Timezone.
- **Set username**: `pi`
- **Set password**: `IOTXX` (replace XX with the number shown on you Raspberry Pi box)
- Select **Wi-Fi-network**: Choose for **ssid: UGent_IoT_Lab**

- Enter **Wi-Fi-password**: **ugent_iot_lab**
- Check for **updates** and install them with the commands: `sudo apt update` and `sudo apt upgrade`. This might take awhile to run. You can proceed to Task3 (Section 6.4) if you want to make more efficient use of your time. If this doesn't work because of SSL errors, check Appendix A and the network connectivity of your RPi (e.g. `ping google.com`).
- **Reboot** with `sudo reboot`.

The OS should now be ready.

Raspberry Pi OS has the SSH server disabled by default. It can be enabled manually from the desktop or via the command line:

Launch Raspberry Pi Configuration from the Preferences menu in the **GUI** Preferences > Raspberry Pi Configuration > tab Interface > Select Enabled next to SSH, Serial Port, Serial Console and SPI > OK.

Alternatively, `raspi-config` can be used in the **terminal**. Open a terminal, enter `sudo raspi-config` in the terminal. First select `3 Interface Options`, then navigate to `P2 SSH`, press Enter and select `<Yes>` to the question `Would you like the SSH server to be enabled?`. The message `The SSH server is enabled` appears, click `Ok`. Choose `Finish`. Enable the Serial Port, Serial Console and SPI following the same steps.

Should any of these steps turn sour, please go to the Raspberry Pi website for help.

5.3 Connecting the Pi to a network

Next, you need to connect the Pi to a network (if you didn't set up the Wi-Fi connection during the OS setup). Plug an Ethernet cable from your router into the Pi (see Figure 3) and/or add Wi-Fi connectivity to your Pi.

Use these Wi-Fi credentials when physically in the lab:

- **ssid="UGent_IoT_Lab"**
- **psk="ugent_iot_lab"**.

We will now configure the Pi to always use a specific IP address by configuring a **static IP address** when using the Wi-Fi interface. Use the network manager using `nmtui` in the terminal. Go to 'Edit a connection' and select the `UGent_IoT_Lab` network. Change 'IPV4 Configuration' to `Manual` and press 'Show' next to it. Specify following information in the opened section:

- Addresses: 192.168.0.x (see the extra file on Ufora for the IP address assigned to you)
- Gateway: 192.168.0.1
- DNS servers: 8.8.8.8

Set IPV6 configuration to `Ignore` and select `OK` and back out this interface until you see the terminal again.

Reboot the wireless adapter by executing this command:

```
sudo ip link set wlan0 down && sudo ip link set wlan0 up
```

You can also set up a Wi-Fi connection through the CLI instead of the GUI by adding the following lines to `/etc/wpa_supplicant/wpa_supplicant.conf`:

```
network={
ssid="<your Wi-Fi name>"
psk="<your Wi-Fi-password>"
}
```

Note that you can also connect your Pi to **Eduroam** if you add following lines to `/etc/wpa_supplicant/wpa_supplicant.conf`:

```
network={
ssid="eduroam"
scan_ssid=1
key_mgmt=WPA-EAP
eap=PEAP
identity="username@UGent.be"
password="yourpassword"
phase1="peaplabel=0"
phase2="auth=MSCHAPV2"
}
```

FYI, ping doesn't work on the Eduroam network.

5.4 Changing the hostname

Your Pi should now be up, running, and connected. Although you can write and run all exercises directly on the Pi using a keyboard, mouse, and screen, a more practical option is to run it "headless" (that is, without a display/keyboard attached to it) and remotely connect to it via **SSH**. The only problem in this mode is **finding your Pi** in the first place. This is actually a well-known problem in the Internet of Things known as the bootstrap or discovery problem: given a device connecting the very first time to a network, how do you find its address?

To resolve this problem, use the **Avahi mDNS server** of your Pi. mDNS is a discovery protocol that gives your Pi an address that nearby computers can use to find it. Avahi is installed by default on the latest versions of the Raspberry Pi OS, so you can go ahead and use it. By default, Avahi will set up the Pi to respond to the `raspberrypi.local` domain. However, we won't have you ping `raspberrypi.local` in the lab, because every student's Pi will have the same name. The **default hostname** is `raspberrypi` for all of them, **causing conflicts** on the network and causing another Pi to fail to resolve their hostname and thus making them unreachable via hostname-based protocols like Samba file sharing.

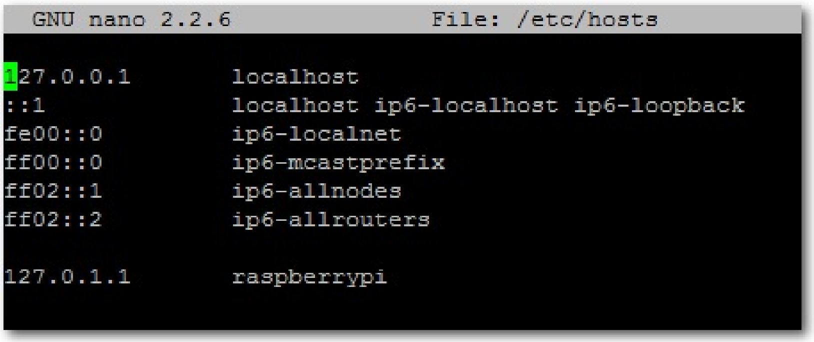
Because we have more than one Pi in the lab room, we better change your hostname.

However, if you are at home instead of in the lab, you can try to locate, or "ping" your RPi as `raspberrypi.local` before you change the hostname, by following the instructions in section 5.5.

To change the Pi's hostname, open up the terminal on the Pi and type the following command to open the hosts file:

```
sudo nano /etc/hosts
```

Your hosts file will look like this (Figure 4):



```
  GNU nano 2.2.6                  File: /etc/hosts

127.0.0.1       localhost
::1             localhost ip6-localhost ip6-loopback
fe00::0         ip6-localnet
ff00::0         ip6-mcastprefix
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters

127.0.1.1       raspberrypi
```

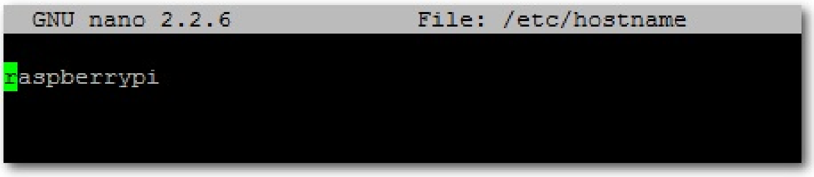
Figure 4: Pi hosts

Leave all the entries alone except for the very last entry labeled `127.0.1.1` with the hostname `raspberrypi`. This is the only line we need to edit. Replace `raspberrypi` with another hostname. Use the name on the yellow or white label on the box of the RPi as the new hostname. **All hostnames start with IOT**. Press `CTRL+X` to close the editor; agree to overwrite the existing file (= Y) and save it (= Enter).

Back in the terminal, type the following command to open the hostname file:

```
sudo nano /etc/hostname
```

This file only contains your current hostname:



```
  GNU nano 2.2.6                  File: /etc/hostname

raspberrypi
```

Figure 5: Pi hostname

Replace the default `raspberrypi` with the same hostname from the previous step. Again, press `CTRL+X` to close the editor, agree to overwrite the existing file (= Y) and save it (= Enter).

Finally, we need to **reboot** the system for the changes to take effect. In the terminal, enter the following command:

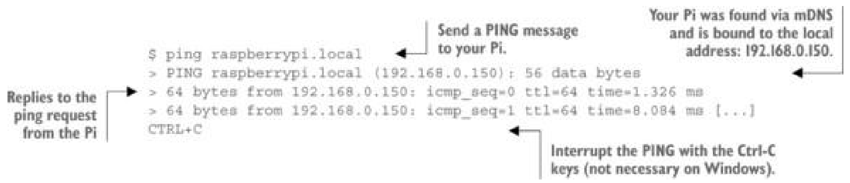
```
sudo reboot
```

In the next step, we'll verify that this all worked by remotely accessing the Pi.

5.5 Remotely accessing your Pi

Only perform the ping operation **in the lab** after you update the Raspberry Pi's hostname.

To check that you can remotely access your Pi, run the command shown in Figure 6 from a terminal, while replacing raspberrypi.local with <IOTXX-hostname>.local or the IP address. Unfortunately, you might not find your Pi, because mDNS isn't supported out of the box on Windows machines. However, an update for Windows 10 added support for mDNS. It will also work if you installed an application bundling an mDNS service such as iTunes. But if you didn't, you'll need to install an mDNS service such as Bonjour Print Services for Windows. You can also use the IP address of your RPi instead of its hostname. Keep in mind that this IP address likely will change when you switch networks (see section 5.3). A warning: if you're connected to the Eduroam network, ping won't work.



```
$ ping raspberrypi.local
> PING raspberrypi.local (192.168.0.150): 56 data bytes
> 64 bytes from 192.168.0.150: icmp_seq=0 ttl=64 time=1.326 ms
> 64 bytes from 192.168.0.150: icmp_seq=1 ttl=64 time=8.084 ms [...]
CTRL+C
```

Send a PING message to your Pi.
Your Pi was found via mDNS and is bound to the local address: 192.168.0.150.
Replies to the ping request from the Pi
Interrupt the PING with the Ctrl-C keys (not necessary on Windows).

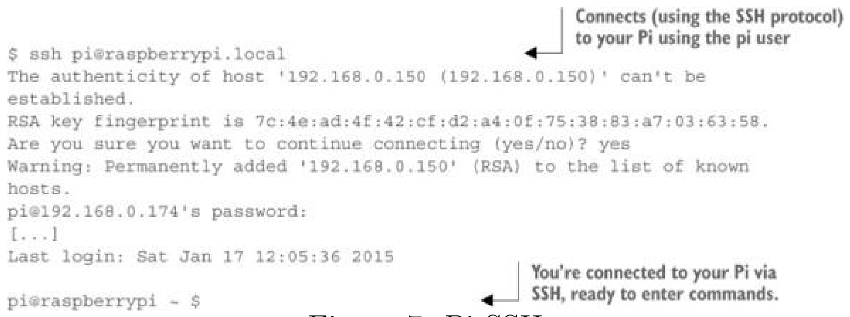
Figure 6: Ping to Pi

If everything worked well, you should now be able to access your Pi via its local DNS address: **raspberrypi.local** or **IOTXX.local**. Try this in the lab ONLY after changing the hostname.

An alternative way to achieve the same thing is to use the static IP you configured earlier: `ping 192.168.0.x` (Where `x` is the number you were assigned by the list in the lab).

5.6 Connecting to your device

If you were able to ping the Pi successfully, you should be able to connect to it using SSH (Secure Shell). An SSH client should already be installed on your Linux, Mac OS, Windows 10 or Windows 11 machine, so all you need to do is open your terminal and enter the command `ssh pi@<IOTXX>.local` (also shown in Figure 7). You should replace `IOTXX` with your IOT hostname from above.



```
$ ssh pi@raspberrypi.local
The authenticity of host '192.168.0.150 (192.168.0.150)' can't be
established.
RSA key fingerprint is 7c:4e:ad:4f:42:cf:d2:a4:0f:75:38:83:a7:03:63:58.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.150' (RSA) to the list of known
hosts.
pi@192.168.0.174's password:
[...]
Last login: Sat Jan 17 12:05:36 2015

pi@raspberrypi - $
```

Connects (using the SSH protocol) to your Pi using the pi user
You're connected to your Pi via SSH, ready to enter commands.

Figure 7: Pi SSH

5.7 PyCharm

The only IDE that will be supported by the assistants in these lab sessions is **PyCharm Professional Edition**, which is currently on version number 2024.3. You can get this otherwise expensive software for free by registering at https://www.jetbrains.com/shop/eform/students using your Ghent University email address. Later you will receive a confirmation email.

Download the JetBrains Toolbox for easy access to all your JetBrains coding tools. **Download and install the latest PyCharm Professional Edition**.

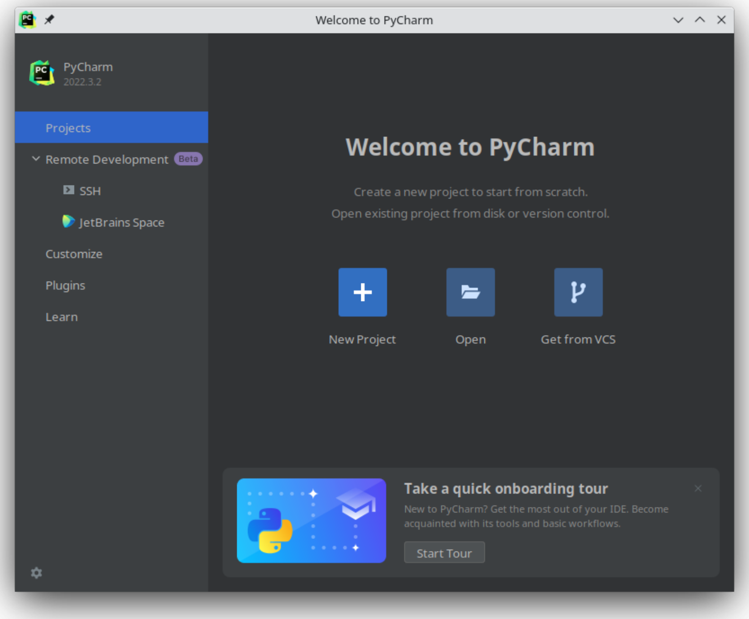
1. Your RPi should be up and running by now and have a hostname that starts with IOT. Your RPi should be visible in the same network as your laptop.
2. When you have selected your background color and get to the start window of PyCharm, click **Create New Project**. See Figure 8.
3. Name your project "RPiTest", select **Custom environment** as the interpreter type and select **Select existing** for the environment. The type should be set to **Python** and the Python path to **any existing Python version** on your computer. This interpreter setting is only used to create the project. We'll change the interpreter to Python installed on the RPi in the one of the next steps. See Figure 9.
4. After PyCharm is open, make sure your **Status Bar** is visible in PyCharm. (Hamburger menu > View > Appearance > Status Bar).
5. Currently the project is set up to fully run on your computer. We'll switch to a remote python interpreter on your RPi. This makes the code run on the RPi, while letting you edit the code on you computer. You'll also be able to see the code outputs in your PyCharm terminal. Let's create a new virtual environment (if you don't know what virtual environments are, read the introduction on this page to get an idea why they're useful) on the RPi and change the project settings to use this newly created environment. Starting in the top navigation bar (you can permanently enable it via Hamburger menu > View > Appearance > Show Main Menu in Separate Toolbar), click **File > Settings > Project: Project name > Python Interpreter > Add Interpreter > On SSH...**. A window pops up to configure your SSH interpreter. See Figure 10. Select **New SSH connection**
6. Specify the correct information for your RPi (host, port, and username). Host should be your RPi IP address or hostname (IOTXX) of your RPi. The standard username is 'pi'. Click "Next". Next, enter your RPi password, which should be 'IOTXX' (with XX being the number on your RPi box).
7. Now that we set up the connection, a python environment on the RPi needs to be specified. Make a new virtual environment by selecting **New** environment. Set the **location** for the new virtual environment to `/home/pi/.virtualenvs/<Project Name>` and the **base interpreter** to `/usr/bin/python3`. These options define the location of your new virtual environment and what Python version to base itself off. The next option defines where the project files of your project will be uploaded to so they can be executed on the RPi. Open the **Sync folders** option by clicking on the folder icon next to the text. Change the **remote path** by clicking the folder icon and selecting a folder on the RPi. We suggest creating and selecting the folder `/home/pi/Documents/<Project Name>` for this project. All this specifies that we want to make a new Python virtual environment based on the version located at `/usr/bin/python3` and we want to store it at `/home/pi/.virtualenvs/<Project Name>`. We want to copy the local project files to the folder `/home/pi/Documents/<Project Name>` when executing them remotely. See Figure 11. Click "**Create**" to use these new settings.
8. A new project has been made, the interpreter is set to a virtual environment on the RPi and the deployment path for this project has been specified. PyCharm should be able to index all the files and packages of the project and its interpreter. Now, **create a test script** by just printing 'Hello World!' to the console and see if it correctly executes.

If it doesn't run and you get a "File not found" error, go to the deployment menu in **Tools > Deployment** and make sure that **Automatic Upload (Always)** is selected. You can also force the upload by selecting the upload (**Upload to pi@<hostname>**) or sync (**Sync with Deployment to pi@<hostname>**) option from the same menu. See Figure 12.

9. The example script should now be executable without errors. Try changing the print statement and see if the automatic upload works if the **run** button is pressed again.

10. One last thing we also need to know is how to install new packages on the RPi and use them in PyCharm. Go to **File > Settings > Project: Project name > Python Interpreter** and click on the **plus icon** to add a new package. In the list of packages, search for **numpy** and click install. When numpy is installed, add some numpy code to your test script (e.g. `print(numpy.random.randint(low=0, high=20, size=10))`). Intellisense should be working after indexing has finished and it should be able to execute the numpy function on the RPi.

11. An alternative way of installing the packages is using the CLI. On the RPi, **activate the virtual environment** using `source /home/pi/.virtualenvs/<Project name>/bin/activate`. When activated correctly, the environment name will be shown in front of the user- and hostname in the terminal. Now you can install packages with `pip install <package name>`. When installing Numpy throws an error related to (Open)BLAS: You'll need to manually install `libopenblas` for Numpy and many other packages to work (as they're dependant on numpy). You can do this using `sudo apt install libopenblas-dev`.



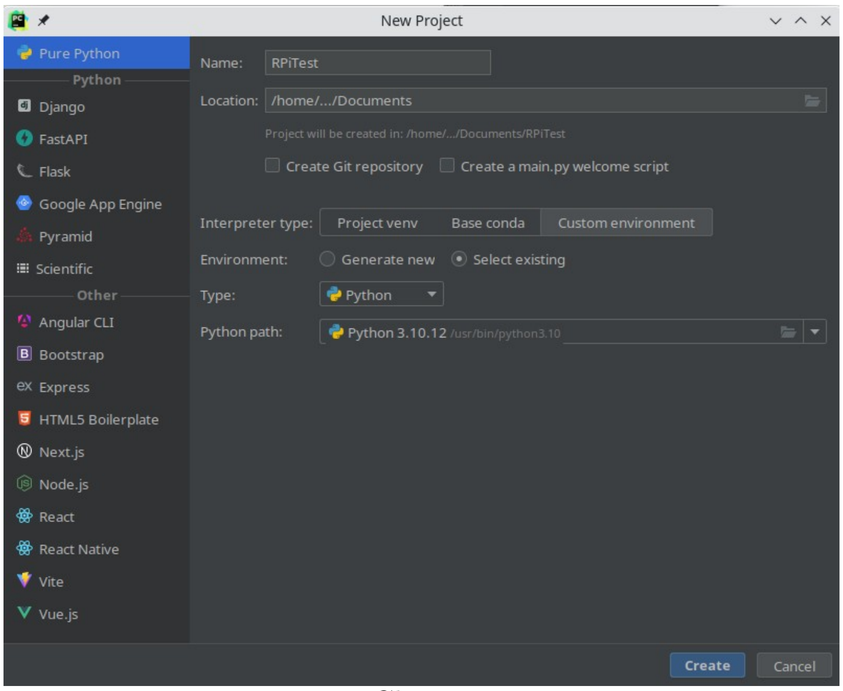
Welcome to PyCharm

Create a new project to start from scratch.
Open existing project from disk or version control.

Take a quick onboarding tour

New to PyCharm? Get the most out of your IDE. Become acquainted with its tools and basic workflows.

Figure 8: PyCharm Start Window


New Project

Name: RPiTest

Location: /home/.../Documents

Project will be created in: /home/.../Documents/RPiTest

Create Git repository   Create a main.py welcome script

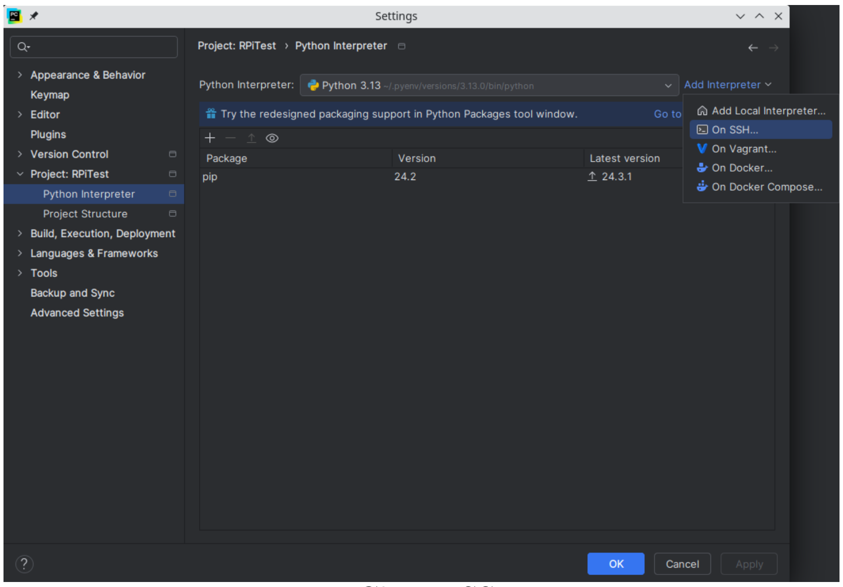
Interpreter type: Project venv   Base conda   [✓ Custom environment]

Environment: Generate new   [✓ Select existing]

Type: Python

Python path: Python 3.10.12 /usr/bin/python3.10

Figure 9: PyCharm New Project


Settings

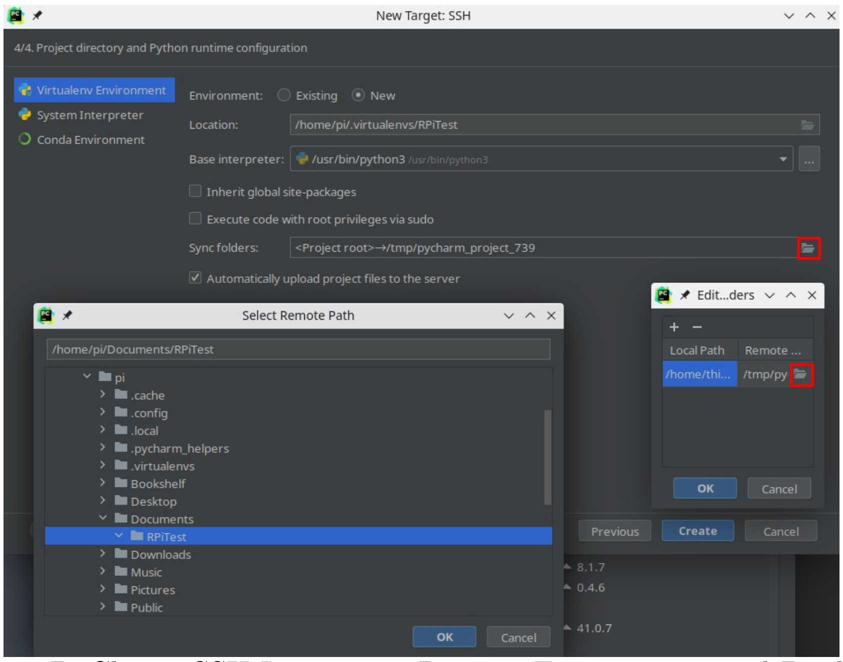
Project: RPiTest › Python Interpreter

Python Interpreter: Python 3.13 ~/.pyenv/versions/3.13.0/bin/python

Try the redesigned packaging support in Python Packages tool window.

Package	Version	Latest version
pip	24.2	24.3.1

Figure 10: PyCharm SSH Interpreter



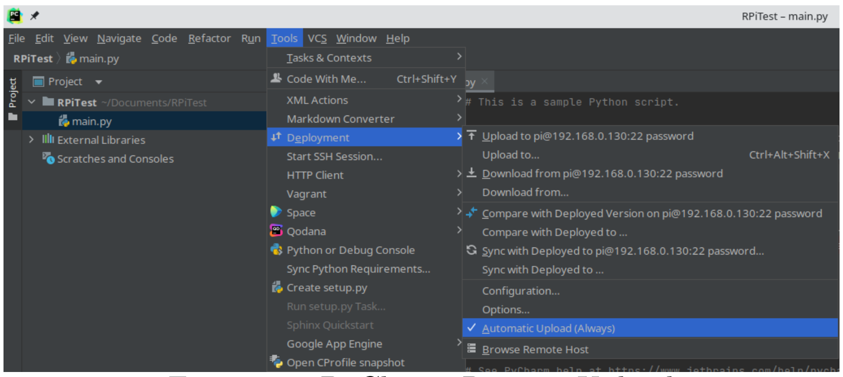
Environment: Existing [✓ New]
Location: /home/pi/.virtualenvs/RPiTest
Base interpreter: /usr/bin/python3 /usr/bin/python3
Inherit global site-packages
Execute code with root privileges via sudo
Sync folders: <Project root>→/tmp/pycharm_project_739
[✓] Automatically upload project files to the server

/home/pi/Documents/RPiTest

```
pi
  .cache
  .config
  .local
  .pycharm_helpers
  .virtualenvs
  Bookshelf
  Desktop
  Documents
    RPiTest
  Downloads
  Music
  Pictures
  Public
```

/home/thi... /tmp/py

Figure 11: PyCharm SSH Interpreter Remote Environment and Deployment


RPiTest – main.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
RPiTest main.py
Project
RPiTest ~/Documents/RPiTest
main.py
External Libraries
Scratches and Consoles
Tasks & Contexts
Code With Me... Ctrl+Shift+Y
XML Actions
Markdown Converter
Deployment
Start SSH Session...
HTTP Client
Vagrant
Space
Qodana
Python or Debug Console
Sync Python Requirements...
Create setup.py
Run setup.py Task...
Sphinx Quickstart
Google App Engine
Open CProfile snapshot
Upload to pi@192.168.0.130:22 password
Upload to... Ctrl+Alt+Shift+X
Download from pi@192.168.0.130:22 password
Download from...
Compare with Deployed Version on pi@192.168.0.130:22 password
Compare with Deployed to ...
Sync with Deployed to pi@192.168.0.130:22 password...
Sync with Deployed to ...
Configuration...
Options...
✓ Automatic Upload (Always)
Browse Remote Host

```
# This is a sample Python script.
```

Figure 12: PyCharm Remote Upload

6 Tasks

The way to connect some sensors and actuators on most platforms, RPi and ESP32 included, is to connect the sensors and actuators to the general-purpose input/output (**GPIO**) ports. Essentially, a GPIO is a pin on which current can be read or output. GPIOs have two modes: an input mode and an output mode. When the output mode is selected, the pin can be set to HIGH, which means it outputs 3.3 volts; when the pin is set to LOW, it is off and doesn't output any voltage. With the input mode, you essentially can read a value on the pin. Unlike other embedded platforms (such as Arduino), the Pi supports only digital input. This means that you can work only with components that supply series of 0s (LOW, 0 volts) or 1s (HIGH, 3.3 volts) to the input pins—that is, with digital components. As an example, an LED is a digital actuator and a button is a digital sensor. The Arduino, on the other hand, has a built-in analog-to-digital converter (ADC).

Analog components, on the other hand, are those that do not provide or consume only LOWs and HIGHs but also supply or consume variable voltages on the pins. As an example, a cheap, light-dependent resistor is an analog light sensor and a potentiometer is an analog actuator. The RPi cannot output a true analog signal, but is equipped with a PWM (pulse-width modulation) feature. The pulse-modulated signal is an approximation of an analog signal in a defined range of voltage. The RPi can similarly use special components or libraries to read signals (input) via PWM, if they are not high frequency signals.

In the following section, you'll connect an LED (actuator) and an RFID reader (sensor) to the Pi through the GPIOs, and connect the RFID reader to the ESP32 through its GPIOs. Let's begin with the hardware part. For this you'll need a **breadboard** (see Figure 13) to prevent you from having to solder components when creating a prototype. The exterior blue row is the one that gets connected to the ground (GND, -). All holes in this row are connected through a metal plate. The exterior red row is the one that will receive the power (VCC, +). All rows are connected. The inner columns are meant to hold components like LEDs and sensors or resistors.

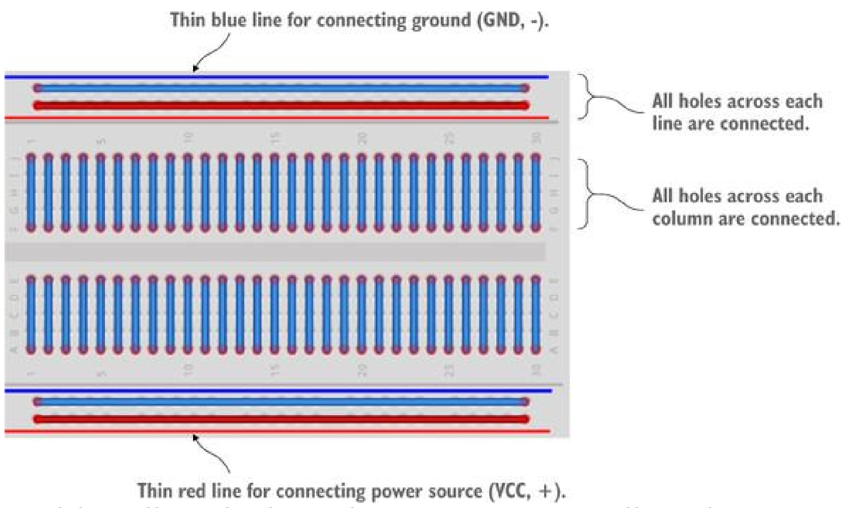
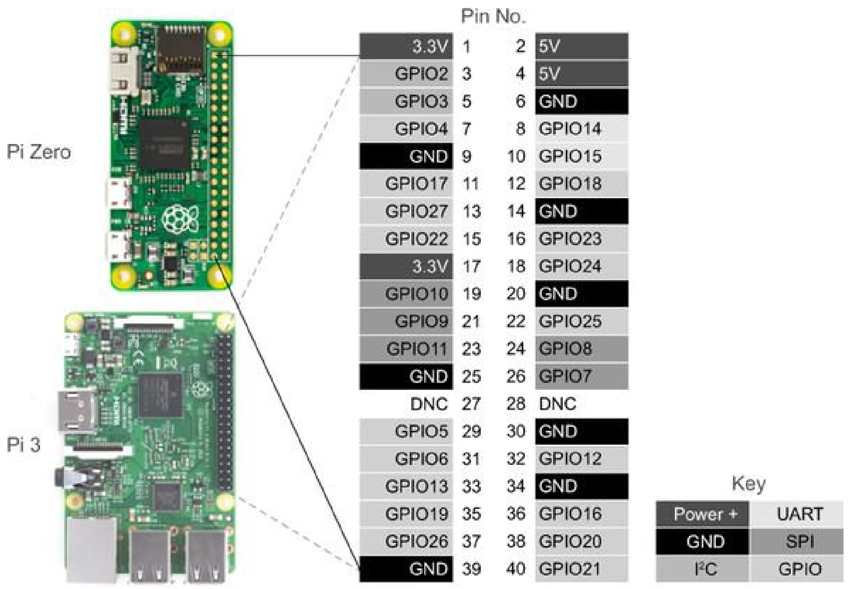

Thin blue line for connecting ground (GND, -).
All holes across each line are connected.
All holes across each column are connected.
Thin red line for connecting power source (VCC, +).

Figure 13: A typical breadboard where the outer rows as well as the inner columns are connected. The exterior line marked with a thin blue line is usually used for connecting to the ground, and the line marked with a thin red line is for connecting to the power source.

Don't forget to take screenshots or short movies of the results of each task to show that you got the task working. You'll need to turn these screenshots or videos in with your lab report.

6.1 RPi GPIO

The GPIO numbering differs depending on the model of the Pi. Unfortunately, the numbering is anything but intuitive. Figure 14 helps you to understand what each GPIO pin corresponds to for the Raspberry Pi 3 and Zero. The GPIO pins have had the same layout since the Pi A+.



	Pin No.	Pin No.	
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

Key	
Power +	UART
GND	SPI
I²C	GPIO

Figure 14: Layout of the GPIO, power, and ground pins on the Raspberry Pi Zero and Pi 3

On Linux, GPIOs aren't that mysterious. The values that are being read or written to the pins are available through **files**, so you could theoretically read these virtual files directly from your Python code. We will use a **library** to do so: RPi.GPIO. You will have to install this library in your virtual environment on your RPi.

6.2 #task1 blink.py with RPi– The Hello World for the IoT

The simplest piece of code one can write is the famous Hello World program that displays "`Hello World`" on the console. The Hello World equivalent in the IoT world is to make a real LED blink, so let's build exactly that, as shown in Listing 1 below.

Start by placing the elements on the breadboard, as shown in Figure 15. Place the **LED** and the **330 Ohm resistor** (color code: orange, orange, brown, gold/silver) on the breadboard according to the schematics shown in Figure 15. (Note that resistors don't have a direction; they must simply be plugged in, so you can connect them in any manner.) The resistor prevents the LED from melting by limiting the current going through it. It also makes sure that the LED doesn't blow if you invert the connection of the VCC and GND pins. Note that you can also use resistors with a greater resistance; for example, **1K Ohm** (brown, black, red, gold/silver). This will reduce the brightness of the LED. Connect the column with the **short leg** of the LED to the GND (-, blue) line using the resistor and the one with the **long leg** to the VCC (+, red) line using a **cable**. Finally, connect a cable (ideally a **black** one to signify ground) to pin 6 (GND) and one (ideally a **red** one to signify power) to pin 7 (GPIO4).

The hardware is now ready. To interface with the GPIO pins using Python, we'll need to install the `RPi.GPIO` library in our RPi Python environment.

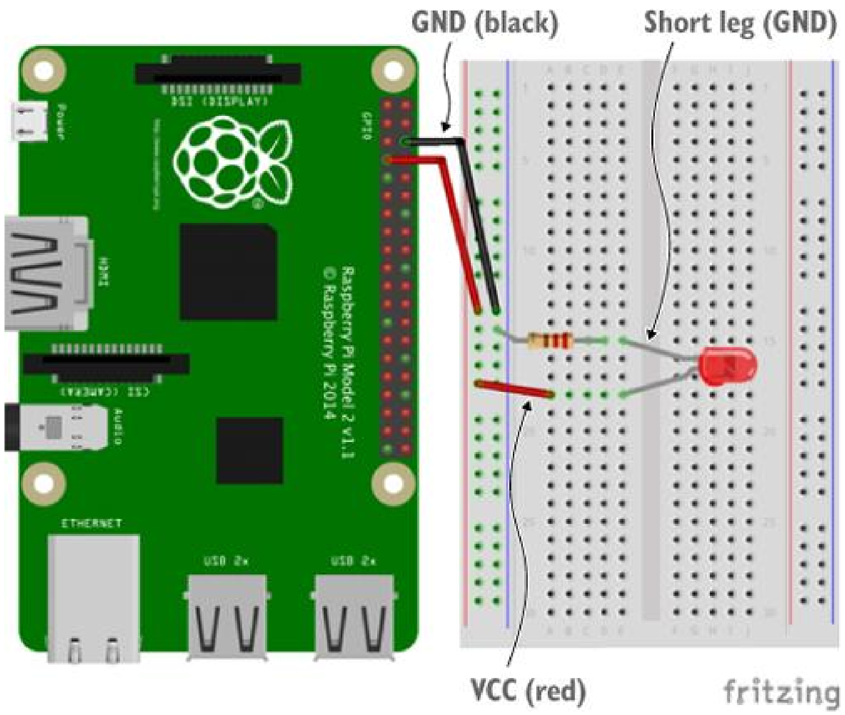


Figure 15: Wiring an LED to the GPIO ports of the Pi through a resistor. The resistor and the LED are plugged into the holes of the breadboard.

As mentioned before, you'll use pin 7, corresponding to GPIO4 (see Figure 14), to turn the power to the LED on and off repeatedly. You'll have run some code that opens the pin in output mode, which means you'll "push" current on it. It then loops, activates and deactivates the pin in turn (turning the LED on and off), sleeping in between. We want the program to shutdown cleanly when throwing SIGINT, which corresponds to pressing `Ctrl+C`. In Python this is a keyboard interrupt exception and can be caught in a the 'except' part of a Try-Except block. When the interrupt is thrown, the program releases the pin and will turn the LED off before exiting.

```python
import RPi.GPIO as GPIO  # Import the library
from time import sleep

LED_PIN = 4

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)   # Use BCM pin numbering (GPIOx)
# Initiate pin as output with initial value of LOW (= off)
GPIO.setup(LED_PIN, GPIO.OUT, initial=GPIO.LOW)

try:
    while True:
        GPIO.output(LED_PIN, GPIO.HIGH)  # Turn LED on HIGH
        print('Changed LED state to ON')
        sleep(1)  # Sleep for 1 second
        GPIO.output(LED_PIN, GPIO.LOW)
        print('Changed LED state to OFF')
        sleep(1)
except KeyboardInterrupt:  # Listen to the event triggered by Ctrl-c
    GPIO.cleanup()  # Cleanly close the GPIO pins before exiting
    print('Bye bye!')
```

Listing 1: blink.py

Save this file as `blink.py` inside your PyCharm project created above. Then go to **Run** and run blink (or click on the green arrow that points right), which will start the program on the RPi for you.

If everything works as expected, you should now see your LED blinking. You can stop the program through PyCharm by clicking on the red square. Don't forget to take a video of this task successfully working so you can turn it in with your report.

Note: If you want to run this directly on the Pi without going through PyCharm, execute these command in the RPi's command line:

- First, activate the virtual environment using the command:
  `source /home/pi/.virtualenvs/<Project name>/bin/activate`
- Second, run the script using the command: `python <filename>`
- To stop the program, hit `Ctrl+C`.

6.3 #task2 RFID Reader with RPi

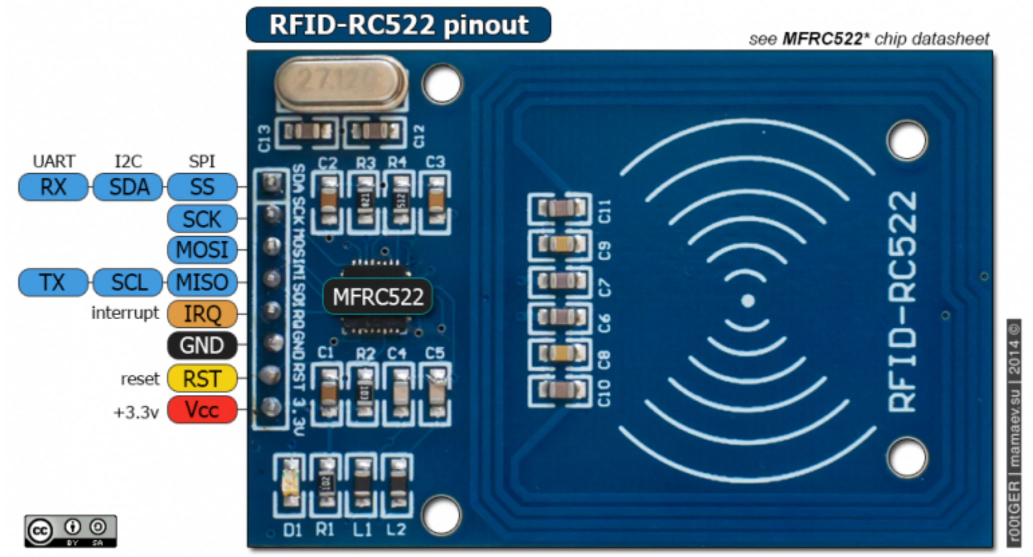


Figure 16: Pinout of the RFID reader RC522.

You'll make your Pi sense the environment by connecting it to an RFID reader. The sensor that we will use is a RFID RC522 reader. Information about its interfacing pins can be seen in Figure 16. Some pins are shared among 3 communication interfaces: SPI, I2C and UART. Only one communication mode can be used at a time. The pins are the SS/SDA (Serial Data Signal)/RX pin; the SCK (Serial Clock) pin which works as SCK if the SPI interface is enabled; the MOSI (Master Out Slave In) pin which works as MOSI if the SPI interface is enabled; the MISO (Master In Slave Out)/SCL/TX pin; the IRQ (Interrupt Request) pin which is an interrupt pin that can alert the ESP32 when an RFID tag comes into its detection range; the GND (Ground) pin; the RST (Reset-Circuit) pin which is a pin for reset and power-down; and finally, the VCC pin which is 3.3v (3.3v Power In). We'll use SPI to communicate with the RFID reader. However, we will not use the IRQ pin's functionality, and so it does not need to be connected in this lab. This means that all but one of the pins need to be connected to our Raspberry Pi's GPIO pins.

You should've already enabled SPI on the RPi when setting up the OS. If it is enabled the command `lsmod | grep spi` should contain `spi_bcm2835` in its output.

Next we need to install a few libraries on the RPi so we can work with the RFID reader easily.

First we need to update the RPi to make sure it's running the latest version of all the software. Run the following commands:

```
sudo apt update
sudo apt upgrade
```

These might take awhile to run if this is the first time you do this. You can proceed to Task3 (Section 6.4) if you do not want to wait.

Then we need to install the `spidev` library. The `spidev` library helps handle interactions with the SPI and is key for the Raspberry Pi to interact with the RFID RC522.

Finally, we need to install the `mfrc522` library that makes it easier to talk with the RFID RC522.

Now we are ready to work with the RFID reader, connect it and program it. Begin by connecting it to your Pi, as shown in Figures 17 and 18, and as detailed in the steps below.

- The SS pin connects to Pin 24 (GPIO8).
- The SCK pin connects to Pin 23 (GPIO11).
- The MOSI pin connects to Pin 19 (GPIO10).
- The MISO pin connects to Pin 21 (GPIO9).
- The IRQ pin is unused and can be left open.
- The GND pin on the RFID reader connects to Pin 6 (GND).
- The RST pin connects to Pin 22 (GPIO25).
- The VCC pin on the RC522 connects to Pin 1 (3.3V).

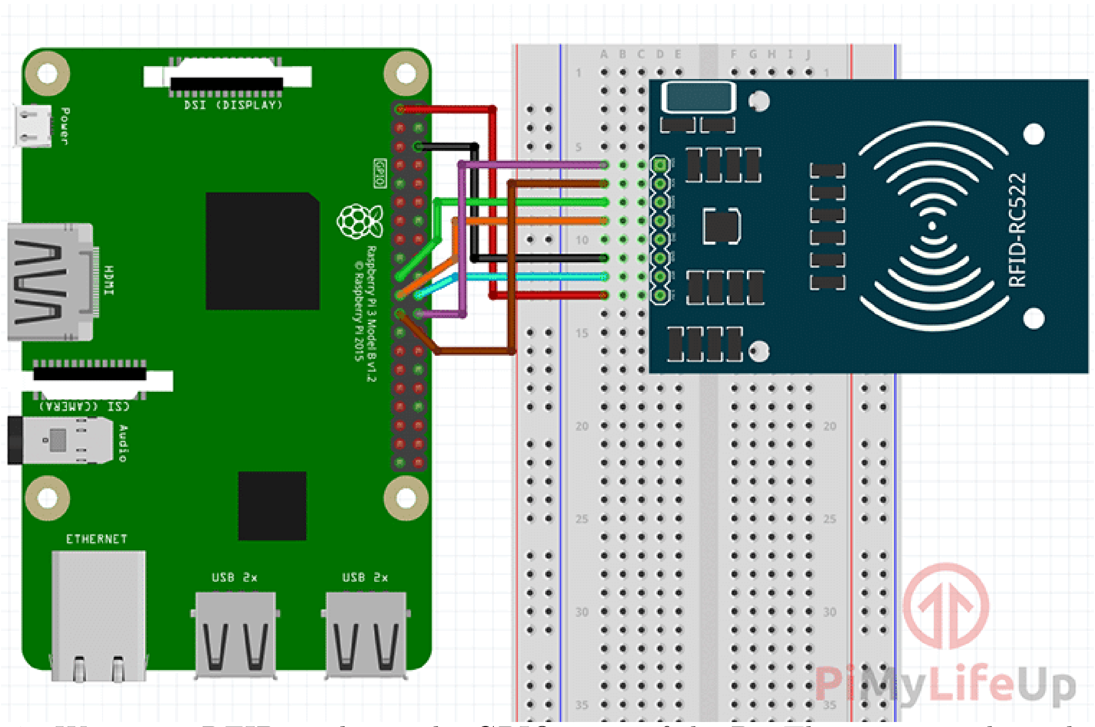


Figure 17: Wiring an RFID reader to the GPIO ports of the Pi. The sensor is plugged into the holes of the breadboard.

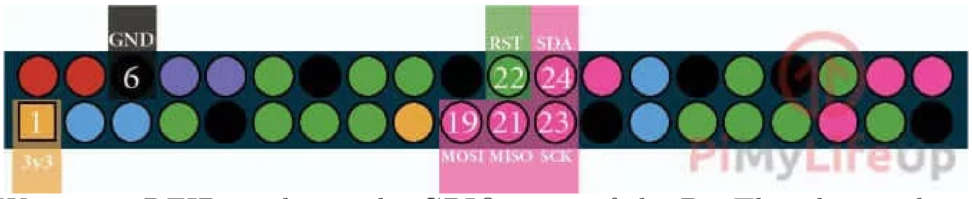


Figure 18: Wiring an RFID reader to the GPIO ports of the Pi. This shows where the pins of the RFID reader connect to in regards to the GPIO ports of the RPi.

You're now ready to interact with the sensor using Python code. The code that will run on the RPi to read an RFID tag from the RFID reader is shown in Listing 2.

This code is straight-forward. First, the necessary libraries are imported. Then, we create a reader instance of the class `SimpleMFRC522`. In a try block, we try to read what is on an RFID tag that comes close to the reader. And finally, we clean up the GPIO pins.

```python
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()

try:
        tag_id, tag_text = reader.read()
        print(tag_id)
        print(tag_text)
finally:
        GPIO.cleanup()
```

Listing 2: readRFID.py

In PyCharm, you can add `readRFID.py` to your project and run it. If everything works fine, you should see the tag ID and text of a tag that is scanned by the RFID reader printed in the PyCharm console. Don't forget to take a video of this task successfully working so you can turn it in with your report.

6.4 #task3 Blink LED with ESP32

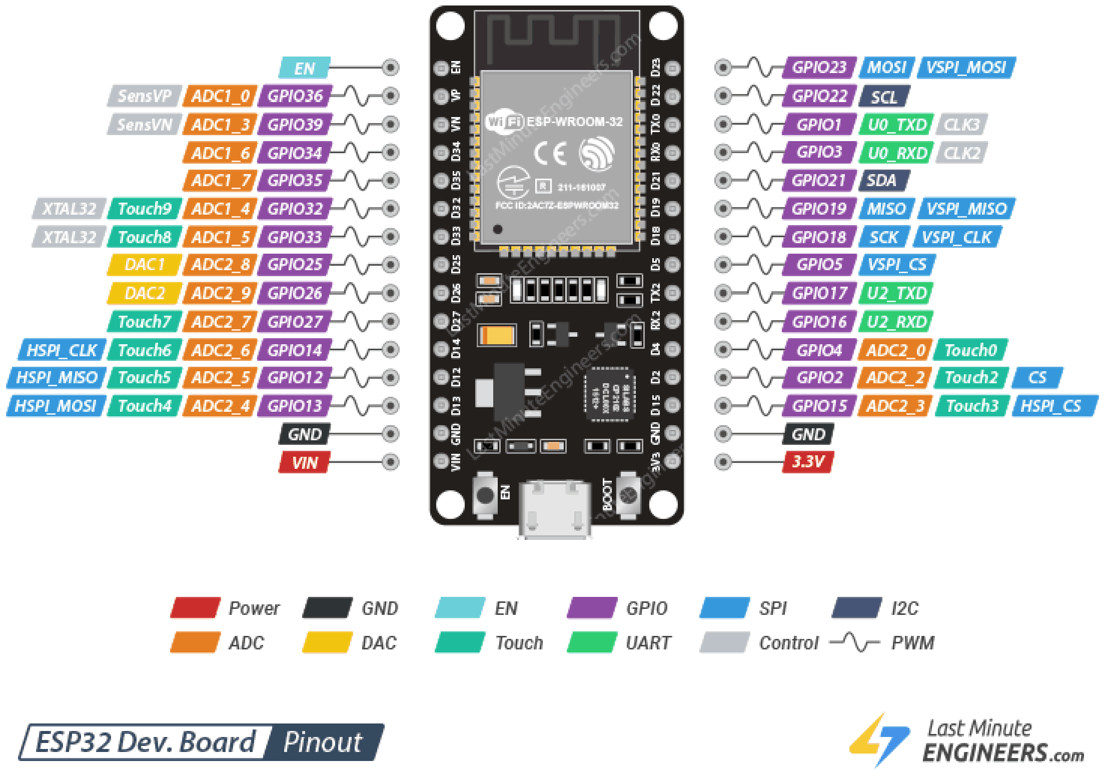


Figure 19: The pinout of our ESP32.

The ESP32 is the successor to the ESP8266. It has everything an IoT-programmer could desire: fast performance (dual core 32-bit processor), fast Wi-Fi, Bluetooth 4.2 and Bluetooth Low Energy, plenty of GPIO with support for many protocols: SPI, I2C, UART, ... This board can be programmed in the Arduino IDE and even in MicroPython! See Figure 19 for an overview of the GPIO pins of our ESP32.

There's an add-on for the Arduino IDE that allows you to program the ESP32 using the Arduino IDE. Follow these steps for Windows users or these steps for Linux and Mac users to prepare your Arduino IDE to work with the ESP32. Please check the board name to select in your IDE (explained below) when you get to the "Test Installation" section of the tutorial.

Plug your micro-USB cable into your ESP32, and the USB side into your laptop to turn on the ESP32. There should be a built-in LED on the ESP32 that shines red. You might have to download drivers to your laptop so you can communicate with your ESP32.

Start the Arduino IDE that you downloaded and installed. Within the Arduino IDE, you have to select the right board and port to be able to send code to your ESP32 so it can run. Go to Tools > Board menu. Choose "NodeMCU-32S". Choose the correct port as well under Tools > Port, which might be called something like "USBtoUART". If you don't know the port to choose, go to your laptop's Device Manager and look under the "Ports (COM & LPT)" tab to see which COM number to select. You can see if an additional port is listed when your ESP32 is plugged into your laptop.

Open a new file in the Arduino IDE and paste the code from Listing 3 to blink the built-in LED. There is some initialization code for the LED in the setup function, and then the code in the loop function just turns the built-in LED on and off continuously.

```
void setup() {
  pinMode (LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

Listing 3: esp32_blink.ino

Then in the top left corner of your Arduino IDE, there is a check-mark. If you click on the check-mark, the code will be compiled and verified. If there are errors, they will appear at the bottom of the window in red. The right arrow next to the check-mark compiles and verifies the code, then uploads it to the ESP32, where it will directly execute. Click on the right arrow.

Note: For some modules, you might have to hold the button labeled `Boot` on the ESP32 during uploading to avoid getting an error.

If all goes well, the LED built into the ESP32 will start blinking. Note that this code is stored in the ESP32's memory, so it will start executing again after the microcontroller is unplugged and then plugged in again. Don't forget to take a video of this task working successfully to turn in with your report.

6.5 #task4 RFID reader with ESP32

Now it's time to connect the RFID reader to the ESP32 microcontroller. First, unplug the ESP32 from your laptop to set up the connections with the RFID reader. Then look at Figures 16 and 19 to help you connect your ESP32 to your RC522, following the steps below.

1. Connect the SS pin to GPIO5.
2. Connect the SCK pin to GPIO18.
3. Connect the MOSI pin to GPIO23.
4. Connect the MISO pin to GPIO19.
5. The IRQ pin is unused and can be left open.
6. Connect the GND pin on the RFID reader to a GND pin on the ESP32
7. Connect the RST pin to GPIO27
8. Finally, connect the VCC pin on the RFID reader to the 3.3V pin on the ESP32

You can plug your ESP32 back into your laptop. Then use the given code (Listing 4) to start a new project in the Arduino IDE. The code includes the library that was written to interact with the RFID reader sensor (MFRC522.h). It then sets up a few of the pins that we connected the RFID readers'pins to (make sure these are the correct pin numbers). We then initialize the RFID sensor. In the setup method, the code sets up the correct baud rate for the data to be output on the serial monitor in your Arduino IDE (Remember "baud rate" is just the number of symbols per second, or the rate at which the signal changes when sent over a channel). The setup method also initializes SPI and the RFID library. Finally in the loop function, the code checks if a new RFID tag is present and reads its type and UID, printing them to the serial monitor. Finally, it performs some clean-up of the RFID reading.

In order for this code to work, we need to import the MFRC522 library into our IDE. Go to Tools > Manage libraries. Search for the MFRC522 library and select the library by Github-Community. Click "Install" and install all dependencies as well.

Now click on the right arrow of your file to compile, verify, and upload the code to your ESP32. Then open the serial monitor by going to Tools > Serial Monitor. Set the baud rate to match the rate in your code, and you should see readings from the RFID tags that are placed on the RFID reader. Try scanning different tags, and make sure you take a short video of the result to add to your report.

```
/*
 * This ESP32 code is created by esp32io.com
 *
 * This ESP32 code is released in the public domain
 *
 * For more detail (instruction and wiring diagram),
 * visit https://esp32io.com/tutorials/esp32-rfid-nfc
 */

#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN  5  // ESP32 pin GPIO5
#define RST_PIN 27 // ESP32 pin GPIO27

MFRC522 rfid(SS_PIN, RST_PIN);

void setup() {
  Serial.begin(9600);
  SPI.begin(); // init SPI bus
  rfid.PCD_Init(); // init MFRC522

  Serial.println("Tap an RFID/NFC tag on the RFID-RC522 reader");
}

void loop() {
  if (rfid.PICC_IsNewCardPresent()) { // new tag is available
    if (rfid.PICC_ReadCardSerial()) { // NUID has been readed
      MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
      Serial.print("RFID/NFC Tag Type: ");
      Serial.println(rfid.PICC_GetTypeName(piccType));

      // print UID in Serial Monitor in the hex format
      Serial.print("UID:");
      for (int i = 0; i < rfid.uid.size; i++) {
        Serial.print(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
        Serial.print(rfid.uid.uidByte[i], HEX);
      }
      Serial.println();

      rfid.PICC_HaltA(); // halt PICC
      rfid.PCD_StopCrypto1(); // stop encryption on PCD
    }
  }
}
```

Listing 4: esp32_rfid.ino

7 Questions

1. What are the main differences between the Raspberry Pi and the ESP32 that we've used in this lab? Hint: Notice the difference between what you had to do to get going with each piece of hardware. List some positives and negatives of each.

8 Summary

Many types of embedded platforms are commonly used in the Internet of Things. We connected a Raspberry Pi to a network, set it up so that we could easily access it remotely over SSH and configured it to be ready for the Web of Things. Embedded devices allow you to use a breadboard, wires, and resistors to wire various sensors and actuators to the GPIO pins. Using Python on your Pi makes it easy to write simple applications that read data from your sensors and control the LED using the `RPi.GPIO` library asynchronously over the GPIO ports. Similarly, we programmed the ESP32 to control an LED and read data from sensors.

9 Material you need to submit

You will need to submit your screenshots or videos of each of the tasks to show that you got each task working. We will not require you to turn in a written report going over each of the tasks of this lab, given that we gave you all of the code for these tasks. However, you will have to answer the question above. You should write up your answer in a document and turn in this document in the .pdf format. If you had to do anything noteworthy in the configuration section (or had issues), you can also include this in your text. Archive everything together, and name this archive **Lab1_FamilyName_FirstName.zip**. Hand everything in using Ufora before the deadline.

Appendices

A Set date and time on RPI after setup (needed for SSL certifications while downloading packages)

If the installation of packages gives certificate problems, you should check if the date is configured correctly. Since the Pi has no RTC (Real Time Clock), setting the date and time is a per-boot activity, not a config activity. You can change the date manually using: `sudo date -s "Mon Oct 8 21:05:00 UTC 2018"` (Note: adjust to current date and time)
You should only have to do this once. If the set time is close the real time, it should update automatically using the NTP (Network Time Protocol) if the port 123 isn't blocked on your router.

You can check if NTP is active using `timedatectl`, which should list the set local and UTC time as well as the set time zone and if the NTP service is active. If it doesn't say 'NTP service: active', enable it using `sudo timedatectl set-ntp true` and reboot the RPi (`sudo reboot now`) for it to take effect. Check if the service is now active and the time is still correct.

B Working with electronics

To avoid the malfunctioning of or damage to electronics please observe the following [1]:

- Avoid handling the printed circuit board or touching the components while it is powered and only handle by the edges to minimise the risk of **electrostatic discharge (ESD)** damage.
- The product should be placed on a stable, flat, **non-conductive surface** in use and should not be contacted by conductive items.
- Use **ESD bags** to prevent damage.

[1] http://www.farnell.com/datasheets/2819352.pdf