



MASTER PYTHON

On Raspberry Pi

LEARN THE ESSENTIALS TO DO ANY PROJECT,
STEP-BY-STEP, WITHOUT LOSING TIME UNDERSTANDING
COMPLEX CONCEPTS YOU'LL NEVER USE



PATRICK FROMAGET

TABLE OF CONTENTS

FOREWORD	3
SETTING UP YOUR ENVIRONMENT	6
ABOUT THE PYTHON LANGUAGE	14
THE BASICS OF PYTHON	18
PYTHON ADVANCED	31
FIRST PROJECT: CONTROL YOUR CAMERA	50
LET'S PLAY:INTERACT WITH MINECRAFT	58
CONTROL YOUR LIGHTS AT HOME	70
COMMUNICATE WITH THE WORLD	85
A FINAL PROJECT: THE SENSE HAT	95
PYTHON LIBRARIES	107
CONCLUSION	115
ACTION STEPS SOLUTIONS	116
BONUS – THE DISCORD BOT	125

FOREWORD

CONGRATULATIONS, YOU MADE THE RIGHT CHOICE

Programming in general is a great skill to have. In a company, those who can code are known to be logical and creative people, good problem-solvers and project planners. Even if you don't work in IT, it might help you with your job and life in general.

Python is perfect to get started in programming, the language is not very complicated compared to others out there. There are a few syntax rules, but it's easy to read.

Python is also an important piece of the Raspberry Pi universe, and I'm glad you are choosing to invest in yourself by learning more about this programming language.

WHAT WILL YOU GET FROM THIS BOOK?

In this book, I'll guide you through all of the steps from being a newbie to knowing enough to experiment with any project you want.

If you already have a bit of experience with programming, you can probably skip a few pages, but we'll quickly get to the Python language and to projects we can do on Raspberry Pi with basic knowledge.

I want to ensure that you understand the goal of this book. You won't become the best Python programmer in the world overnight, master the most complex concepts of programming, or even follow the common best practices.

This book is the first step and will give you an idea of the basics to help you overcome your fear of programming, and have fun with any project on Raspberry Pi that requires a bit of Python code.

THE USUAL TRAP

I hope you have a specific goal in mind to spend a bit of time reading this book, doing the exercises and experimenting on your Raspberry Pi.

But I must warn you, please don't skip any steps and make sure you understand the theory before jumping to the practice.

I know from experience that the syntax and the language as a whole is not the most complicated part in learning how to code. If it's your first language, please don't skip anything.

I worked as a web developer, and from time to time, I had to train employees to read and debug PHP code. They were often from various departments (web design, marketing, IT support, etc.), so most were absolute beginners, it was their first language and first lesson.

Each time they had the same problems, and it wasn't what I expected while preparing the training.

Getting the syntax right requires a bit of practice to allow you to see the patterns and identify errors. In general, after a few hours of practice, it gets better and better. But understanding the base logic of the code is something entirely different. It took them many hours to grab the concept of conditions, arrays and loops for example. These are the basis of any script, and until you understand this, it'll be very hard to do anything.

That's why I will take the time to explain these concepts clearly, by using examples from real life that anybody can understand. But please, don't skim the first chapters too fast, as it's an essential part of this book. I tried to only include important concepts that you'll really use. In the second part, I'll show you interesting projects, but you'll be lost if you skip the first part.

HAVE FUN

You now know that programming is important, whatever your career or goal is. I also told you that the first part might be difficult, but no worries I'll try to keep it interesting, and we'll quickly move to the practice.

Despite all of this, please have fun while reading this book. If you get bored or if you don't understand something, take a break, and come back to it the next day.

Also, try to go a little further in each exercise. The magic in programming happens when you achieve something you wanted, without copy/pasting the solution. If I ask you to light a bulb for example, do it and then try to make it blink or something else. You'll learn much faster by coding your own ideas.

I hope that you are motivated and ready to move on. I'm certain this book will help you, so read it carefully and enjoy the process!

SETTING UP YOUR ENVIRONMENT

INTRODUCTION

In this first chapter, I'll make sure you have everything you need to get started. That might be a breeze for some of you, so I allow you to skim fast if you are comfortable with this.

I'll be using Raspberry Pi OS on my end to explain everything, so I recommend that you use it as well. It's a stable distribution with all the prerequisites needed to code in Python, so we don't need to look for any complications here.

Next, I'll introduce some text editors that you can use to program in Python. You can switch at any time while reading this book and experimenting with your projects, but you need at least one properly configured text editor to get started.

RASPBERRY PI OS DESKTOP INSTALLATION

I assume most of you are experienced users, so I'll give the short version here, and you can always check the full tutorial on the website if needed.

Recommendations

Raspberry Pi OS is the official operating system for the Raspberry Pi (hence the name).

There are 3 versions available: Lite, Desktop and Desktop with recommended software.

You can use any of these versions to code in Python, but I recommend starting with the Desktop version if you have a decent Raspberry Pi model (Pi 3 or more).

Raspberry Pi OS Desktop with recommended software will include everything you need to code, including some text editors. Also, as I will explain for this specific version, it's better if you have the same.

I recommend using a dedicated SD card for this purpose. This way, you won't need to reinstall anything later, and will keep your old scripts safe for later use. A 16 GB Micro SD-Card should be enough, you can find my current recommendations in my resources pages: <https://raspberrytips.com/resources/>

System installation

Here are the steps to follow to install Raspberry Pi OS on your SD card:

- Go to the official website and download Raspberry Pi Imager if you don't have it yet. The URL is <https://www.raspberrypi.org/software/>.
- Install it on your computer like any other software, and start it.
- Once done, click on the "Choose OS" button on the left. In the menu, go to "Raspberry Pi OS (Other)" and click on "Raspberry Pi OS Full".
- Insert your SD card into your computer. If you don't have an SD card reader, you can use a USB adapter (link in my resources pages too).
- Click on "Choose Storage" and pick your SD card in the list.
- Then click on "Write" to start the file's copy.

It will erase your SD card, download the system image and copy the files on the SD card.

It might take a while depending on your hardware and Internet connection.

Configuration

Once done, insert the SD card into your Raspberry Pi and start it.

A mouse, keyboard and screen is highly recommended. You need to be comfortable to code in Python.

At the end of the first start, follow the welcome wizard to configure your Raspberry Pi.

The goal is to:

- Set the correct locale and keyboard layout. I recommend keeping the system in English, so that you can easily follow my instructions.
- Change the default password.
- Adjust your display settings if needed.
- Connect to your network if you use Wi-Fi.
- Update your system.

If you need a step-by-step guide for this, you can check this article:

<https://raspberrytips.com/install-raspbian-raspberry-pi/>.

It'll explain everything, including the additional configuration you can do after that (if you need to change the hostname or set a static IP for example).

Remote access

This is not mandatory, but for some of you it might be a good idea. If you have a comfortable computer with the latest screen, mouse and keyboard, you might prefer using it rather than working directly on the Raspberry Pi with old accessories stacked in a corner of your desk.

If you are in this case, I recommend enabling SSH and maybe graphical access with a tool like VNC or XRDP. You can find all the information here:

<https://raspberrytips.com/remote-desktop-raspberry-pi/>.

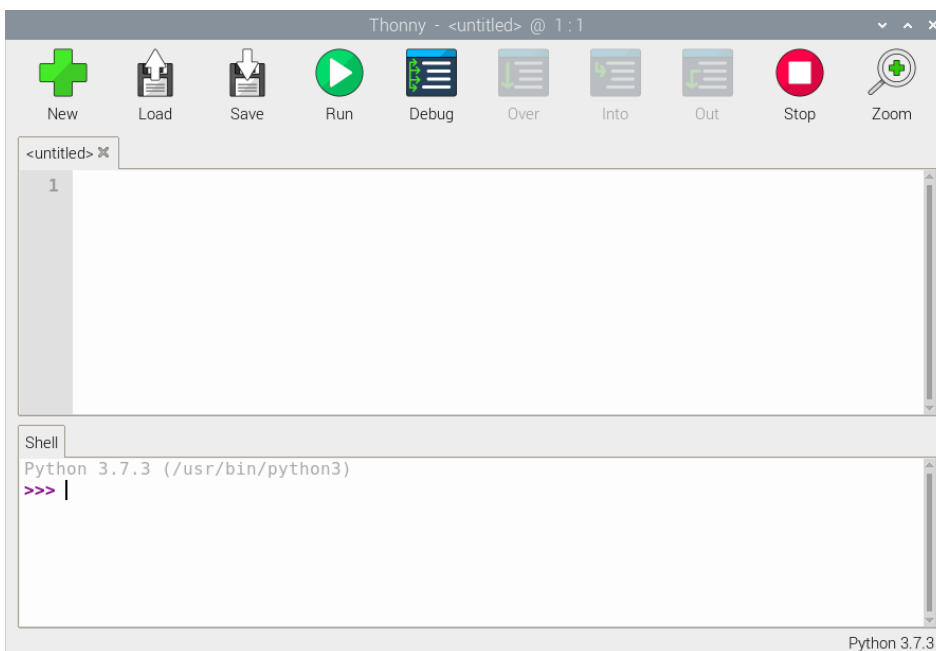
Obviously, if you only have a Raspberry Pi, or are better equipped than your computer, you can skip this step :).

CODE EDITOR INSTALLATION

Once the system part is done, the only thing you need for the moment is to select the text editor you'll use for your Python scripts. If you choose the Full version of Raspberry Pi OS, some text editors are already installed.

I'll introduce the three I recommend for beginners, and link to an additional resource for those who want more advanced tools.

Thonny Python IDE



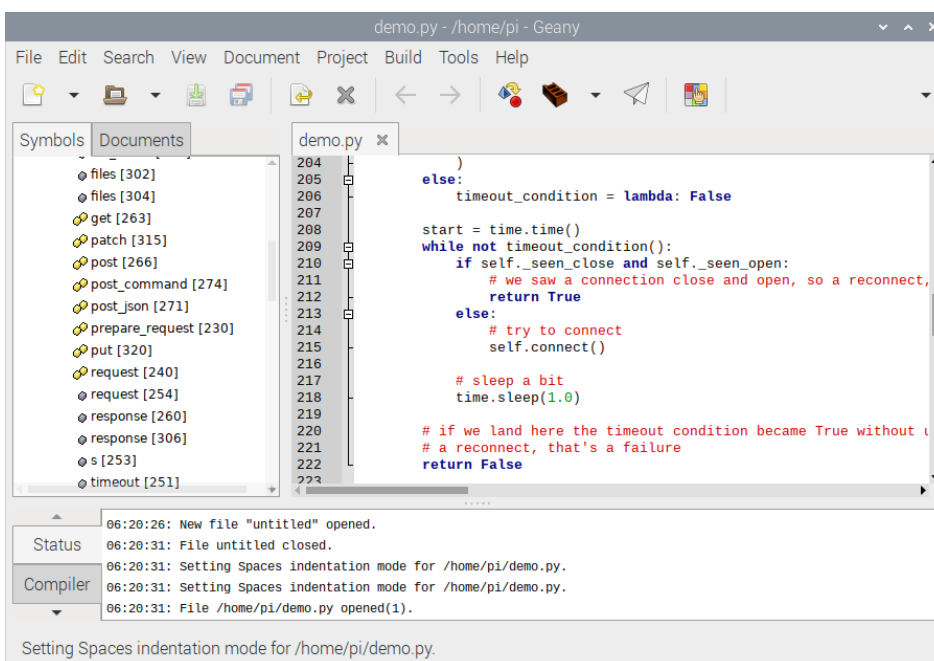
Thonny is perfect for beginners, if you have a few lines to write and don't want to be lost in too many menus and submenus, this is the perfect tool for you.

As you can see on the picture, there is a shortcut panel with big buttons to do the main actions. Then the top of the screen is used to type your code, and you will be able to see the result in the shell tab below.

If you are just starting with Python or programming, I recommend working with this tool.

This tool will allow you to save time. If you need something else later, it isn't a big deal to switch to another editor.

Geany



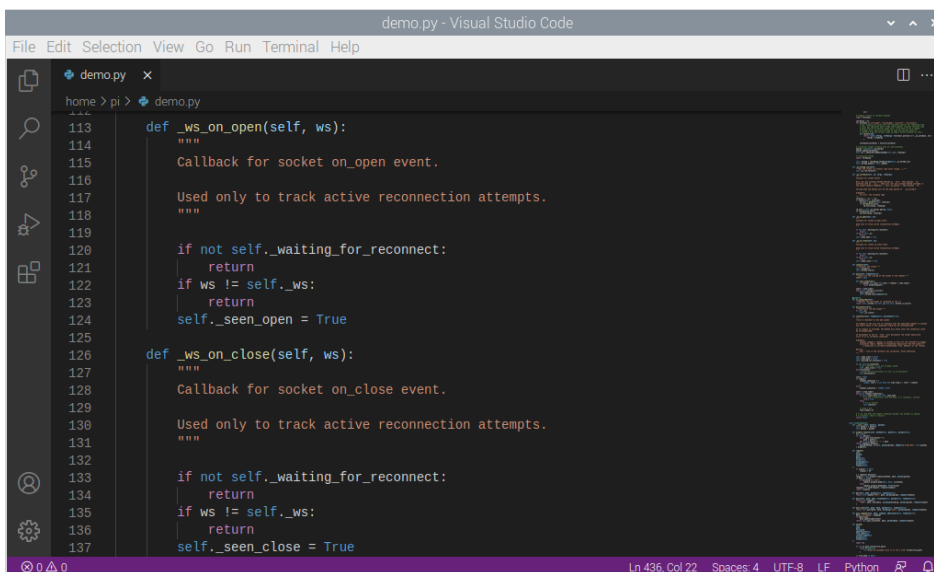
Geany is a programming editor that I have used a lot in the past, when I was on Linux for work. I had a big web project with thousands of files, and it worked very well on Geany, even better than many paid solutions.

So Geany is a great lightweight editor, which is perfect for Raspberry Pi.

If Thonny is too basic for you, Geany might be a good alternative. It's pretty easy to navigate as the main features are quickly available in the top bar, but if needed it can do a lot more than Thonny:

- Appearance customization
- Syntax highlighting
- Autocompletion
- Project management
- Smart navigation
- Etc.

Visual Studio Code



The third option is Visual Studio Code. Yes, it's a tool from Microsoft, but it's powerful, and it's free. You can now install it easily on Raspberry Pi OS, as it's available in the recommended software list. So, why not?

It's a bit more advanced than the other tools. It's Microsoft, so it's not the most intuitive, but it offers cool features. For example, you have access to a giant list of plugins you can install in a few clicks to add even more features (Git, SVN, themes, languages autocompletion, etc.).

Another interesting feature for those who want to work remotely, away from their computer, is that there is a plugin for Python remote access. You can create your code on your computer, and run it on the Raspberry Pi directly (no need to copy/paste it or transfer the files with each modification).

To install it, go to the main menu > Preferences > Recommended software.

In the "Programming" category, find "Visual Studio Code" (last one) and check the box.

Click the "Apply" button to install it.

Other solutions

Basically, any text editor can work to code in Python, so there are many other solutions available.

If you want nothing but text you can try with Nano or Vim, and if you prefer a high-end IDE you can find some options too.

I think the three I previously introduced should be enough for most readers, but if you want to discover a few other options, you can check out this article:

<https://raspberrytips.com/top-text-editors-on-raspberry-pi/>.

ACTION STEPS

At the end of each chapter, I will summarize the steps you have to do on your Raspberry Pi. It will offer a quick recap of the chapter and some exercises to allow you to put the new concepts into practice.

To consider this chapter completed, you need to:

- Download and install Raspberry Pi OS on a new SD card.
- Configure the basic settings for a comfortable usage after that (network, display, keyboard, etc.).
- Do the system updates,
- Choose an editor, install it if needed and start it.

If everything is working as expected, you can jump to the next chapter where we'll start discussing the Python language.

ABOUT THE PYTHON LANGUAGE

INTRODUCTION

Before jumping directly to the code, it's a good idea to try and understand a bit more about the programming language. In this chapter, I'll explain what Python really is, why it has been created and when you should use it.

I know it's probably not what you were looking for, but it's an important step in discovering this language. Python is pushed on Raspberry Pi, but that isn't a good reason to use it all the time. Sometimes, Python isn't always the smarter solution. Don't worry, I'll quickly show you why.

Python is a high-level language to program all kinds of software (like C, C++ or other languages). There is no compilation step, and like a bash script, it can be run directly.

The filename extension for Python scripts is ".py", for example: myscript.py.

ORIGINS

Python is not a new language, it was created in the late 80s by a Dutch programmer, Guido van Rossum (not Mr. Python). He led the project alone until 2018, bringing many new features and two new main versions (Python 2 and 3). The project is now led by a Steering Council of five members that supervise the language development.

Without going into too many details, Python supports many "styles" of programming, including the object-oriented programming method that scares many people.

I chose to skip this, and only focus on the easiest concepts. We'll see some basic structures and create some functions, but we don't need to use the most advanced methods on Raspberry Pi.

If you are interested in a programming career, you'll need to spend a bit of time learning these concepts. However, when you're only a developer, doing this as a hobby, you don't need it.

The main philosophy of Python is to keep everything simple and easy to read. Tim Peters, one of the major contributors has even described this philosophy in a few aphorisms, known as "The Zen of Python":

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

*Although never is often better than *right* now.*

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

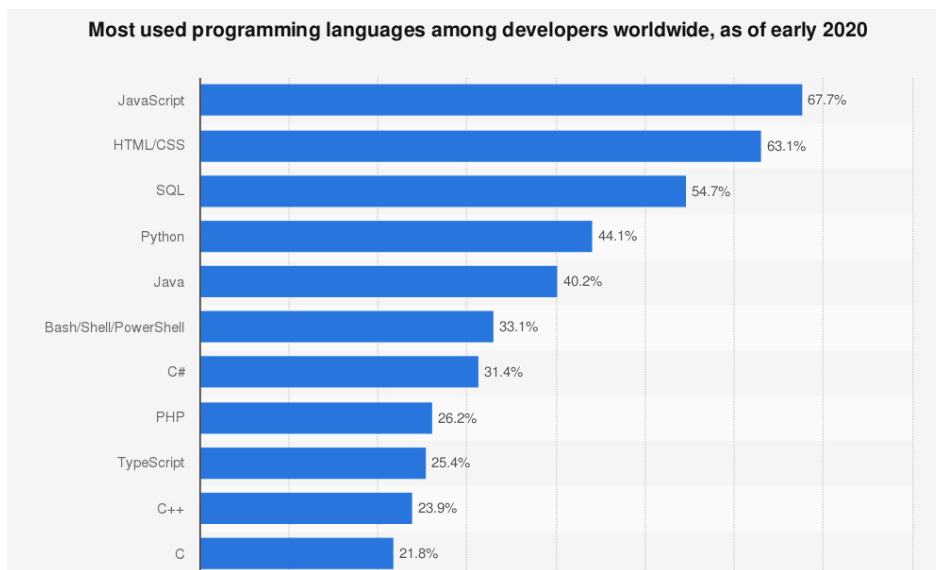
Namespaces are one honking great idea -- let's do more of those!

So, one of the main goals is to keep your code simple, beautiful and readable. Yes, if you are the only one to read it, nobody really cares, but it's always better to have a bit of context when starting a new programming language. Java and C++ don't have the same philosophy at all :).

USAGE

Even if Python is an old language, it's still widely used by programmers all over the world. There is a giant community and most major companies use Python in their products. Videos on YouTube and Netflix are played with Python, Facebook and Spotify also use it, so Python won't disappear tomorrow.

In fact, it's one of the most used languages in the world, according [to a study by Statista in 2020](#):



PYTHON AND RASPBERRY PI

The goal of the Raspberry Pi Foundation is to help young students learn how to code.

The Raspberry Pi foundation sends a lot of Raspberry Pi devices to UK schools, and creates clubs to teach children how to code.

As we discussed in the previous chapter, Raspberry Pi OS comes with Python by default and with a complete IDE already installed (in the Desktop version). So having a Raspberry Pi and using it to learn to program with Python is a great idea - it's made for this.

By the way, it's not mandatory to have a Raspberry Pi device to learn Python. It wouldn't be one of the most popular languages if it was limited to the Raspberry Pi.

You can use it on any operating system, as Python is a cross-platform language.

You only need to install it and find an editor to fit your needs on your current operating system.

CONCLUSION

I told you it will be short, but I think it was important to provide a bit of context on where this language comes from, what the main goals are and why it's installed on Raspberry Pi.

If you know the answer to these questions, you can move to the next chapter, where we'll focus on the code logic and the basic Python syntax.

THE BASICS OF PYTHON

INTRODUCTION

This is the chapter you've been waiting for! The one where you start writing weird sentences that your computer understands (or not). But no worries, we'll slow down a bit here and tackle only the essentials in this chapter, before moving to more complex notions in the next one.

Python uses basic English for the main keywords. There are a few syntax rules to remember, but you'll soon see that it's not very complicated.

HELLO WORLD

Whatever language you learn, it always starts with the famous "Hello World!". The idea is to learn how to display something on the screen first, so let's do this like the tradition wants.

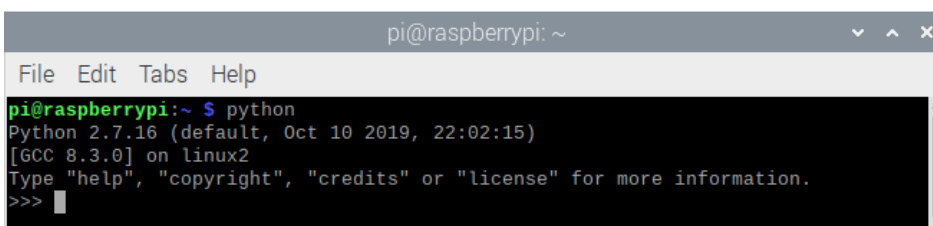
Python console

We won't use it a lot in this book (probably never again), and I don't think you'll use it in your projects either, but I need to introduce the Python console to you. It's a way to run Python code line-by-line and see the result directly.

To open it, it's straightforward. Open a terminal and type:

python

You'll get something like this:



```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ python
Python 2.7.16 (default, Oct 10 2019, 22:02:15)
[GCC 8.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

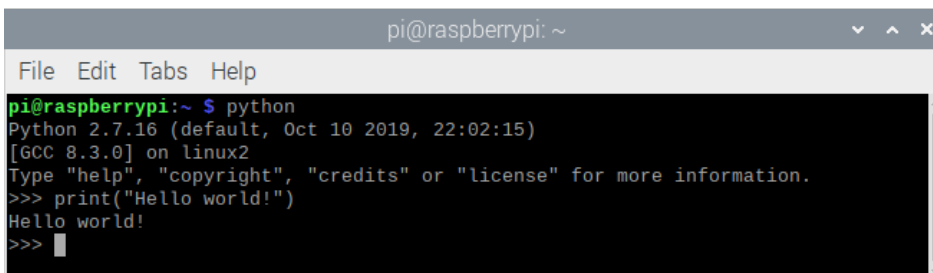
Then you can type any Python code and get the result instantly. But first I have to tell you what to write there :).

For each action you want your code to do, there will be a specific keyword, possibly with a syntax to remember.

So, our first goal is to display a message on the screen, the keyword will be "print", and the syntax is to put the text we want to display in parentheses. The text also has to be delimited by double quotes.

Example: *`print("Hello world!")`*

Type this in the Python console to see the result:

A screenshot of a terminal window titled "pi@raspberrypi: ~". The window has a menu bar with "File", "Edit", "Tabs", and "Help". The terminal shows the command "python" being executed, which starts the Python 2.7.16 interpreter. The interpreter displays version information and a prompt ">>>". The user enters "print('Hello world!')", and the interpreter outputs "Hello world!". The prompt ">>> " is visible again at the bottom.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ python  
Python 2.7.16 (default, Oct 10 2019, 22:02:15)  
[GCC 8.3.0] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print("Hello world!")  
Hello world!  
>>> 
```

If you experience an error at this step, make sure that you typed exactly what is written above. Like most programming languages, Python is pretty strict with the syntax you use. There is no room for creativity in programming syntax.

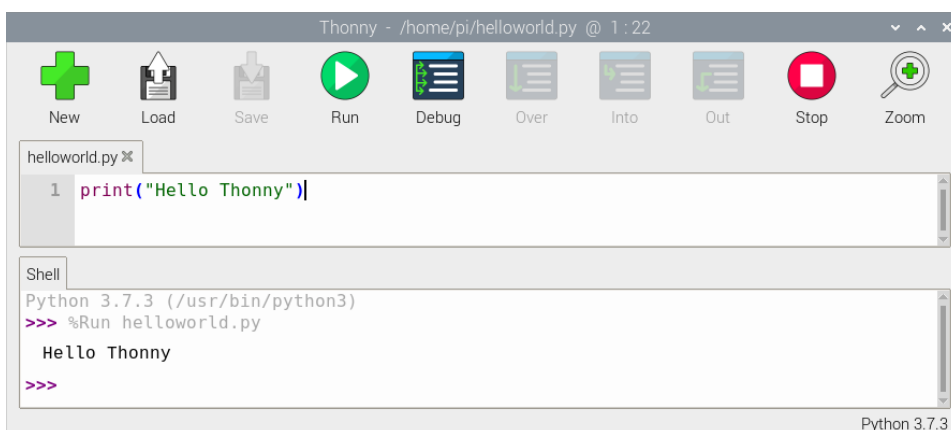
Use CTRL+D to exit the console, as we don't need it anymore.

Create a script

Most of the time, you won't use the Python console. Instead, you will put all of your lines in a file, and run them all in one command.

For now, we'll do this with Thonny, one of the default Python editors on Raspberry Pi OS:

- Open the main menu.
- Click on Programming > Thonny Python IDE.
- Put the same code line in the source code part of the editor:
`print("Hello world!")`
- Click on "Save" in the top bar.
- Give your file a name.
Remember, Python scripts have a .py file extension, so for example:
`helloworld.py`
- Then click on the "Run" button.
- The result should appear in the Shell part of the editor:



helloworld.py

In this case, we only have one line in our script, so there is no real advantage to creating a file instead of typing the command in the console. However, we will add new lines in the following sections, so creating a file for each goal is good practice.

Note: *Just in case, you may have noticed that the Python version we use in both cases isn't the same. It doesn't matter for now, but there are two Python versions installed on Raspberry Pi (Python 2 and 3). When you use the "python" command, it's version 2. You need to use "python3" to start a console with Python 3. Thonny runs Python version 3 by default. We'll get back to this later.*

VARIABLES

We'll now add a new concept: variables. A variable is used to store values or data of any kind in your script. So, if you need to keep something in memory along your script, creating variables will be really useful.

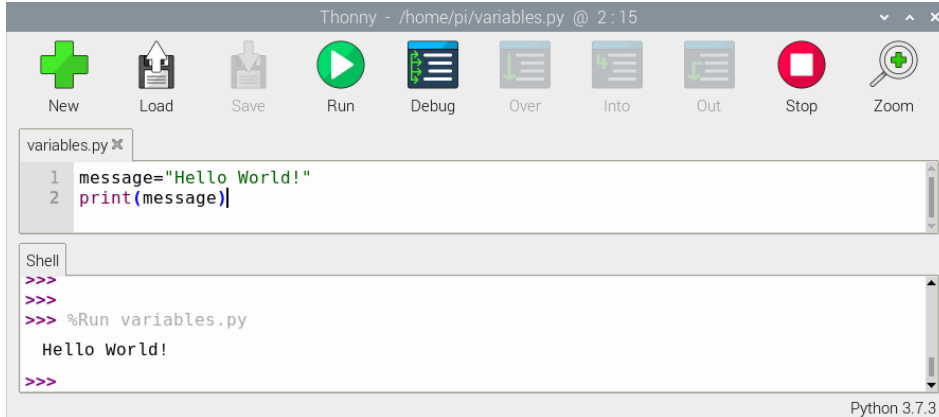
Create a variable

We'll start with a dumb example, so that you can understand what I'm talking about:

- Create a new file in Thonny.
- Copy and paste these lines:

```
message="Hello World!"  
print(message)
```

- Save the file and run it.
- It should display the same thing as in the last part.



```
Thonny - /home/pi/variables.py @ 2:15
```

New Load Save Run Debug Over Into Out Stop Zoom

variables.py

```
1 message="Hello World!"
2 print(message)
```

Shell

```
>>>
>>>
>>> %Run variables.py
Hello World!
>>>
```

Python 3.7.3

variables.py

As you can see, we create a new element named "message" in the script, and we set its value to "Hello World!". Then we can use it in the second line. Python will replace the "message" word with its value.

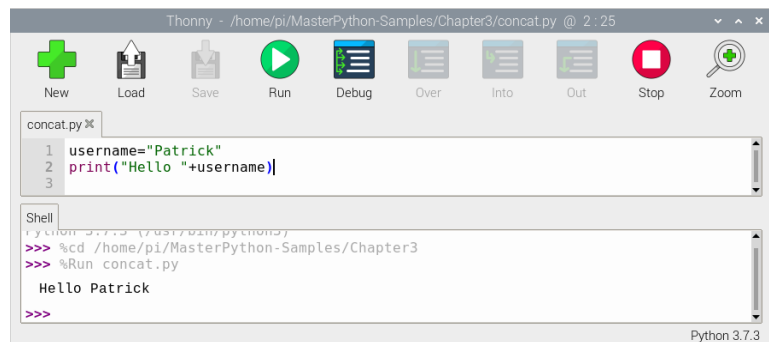
Note: Double quotes are only used to delimit text values. We don't put them in the print function this time, or it will display "message" in text instead of the variable value.

Concatenation

It's the previous example, there is no point in using a variable. It's just a way to use two lines while we have been doing the same thing with one line previously. Yes, you are right!

Here is an example of how you can use variables more wisely:

```
username="Patrick"
print("Hello "+username)
```



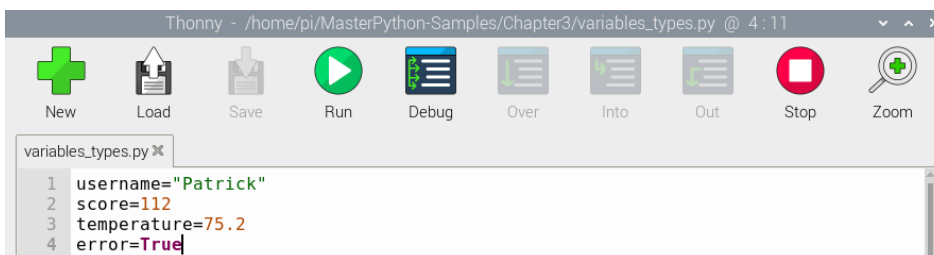
concat.py

The + symbol is a way to stick two strings together. It will be pretty handy in the future.

Obviously, the username has to come from somewhere else, or it's, once again, a more complicated way to do the same thing as in the first script. But you get the idea, I just want to go step-by-step. In the next chapter, we'll learn how to do this interactively.

Variable types

We have only seen text variables in the previous examples, but there are many other types we can use, such as numbers (integer or float) and boolean (True or False). We'll also discover more complex types of variables in the next chapters, but for now, just remember these.



```
Thonny - /home/pi/MasterPython-Samples/Chapter3/variables_types.py @ 4 : 11
New Load Save Run Debug Over Into Out Stop Zoom
variables_types.py
1 username="Patrick"
2 score=112
3 temperature=75.2
4 error=True
```

variables_types.py

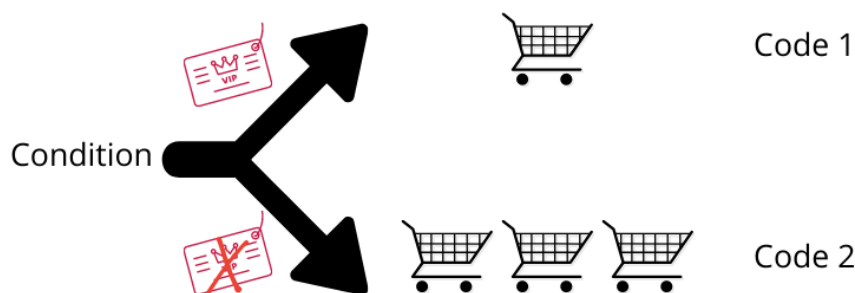
CONDITIONS

Conditions are something you'll use in most scripts. It's a tool you can use to execute a different piece of code, depending on the condition.

Theory

To illustrate this, let's say you are at the supermarket. If you have a loyalty card, you can go to the fast checkout. Otherwise, you have to queue and wait a bit at the next checkout.

In this case, the condition is "do you have a loyalty card?". The answer is "yes" or "no", and depending on this condition, your path will be different.



In programming in general, we often have ways to do the same thing. You can do a different action depending on the result of an expression. It will be useful in many cases, for example:

- If the input string is empty, display an error.
- If the button is pressed, turn on the LED.
- If the password is correct: do the action, else: ask again.

In practice

In Python, you have three keywords to remember:

- **If:** the first condition is true.
- **Elif:** the first condition is false, we test another scenario.
In my supermarket example, it could be something like a checkout for customers with less than 10 items.
- **Else:** default case if none of the other conditions are true.

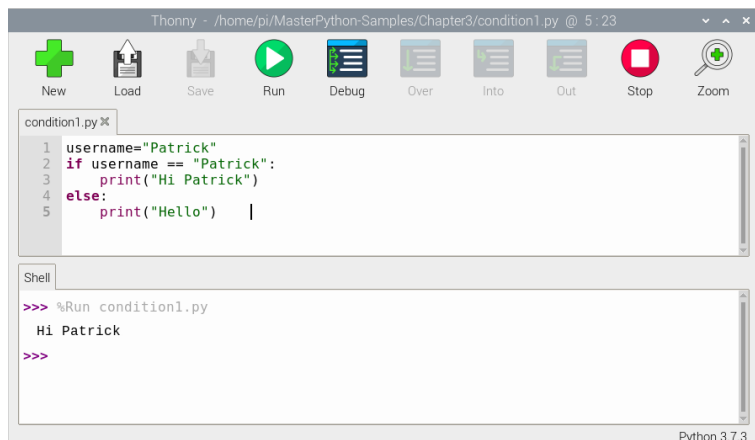
If and *elif* statement will be used with a condition and a piece of code to execute if the condition is true. *Else* doesn't need a condition, it's the default case if all the others are false.

Note: A condition which can be only true or false is named a boolean statement.

Example 1

A first basic example can look like this:

```
username="Patrick"
if username == "Patrick":
    print("Hi Patrick")
else:
    print("Hello")
```



```
Thonny - /home/pi/MasterPython-Samples/Chapter3/condition1.py @ 5:23
New Load Save Run Debug Over Into Out Stop Zoom
condition1.py %
1 username="Patrick"
2 if username == "Patrick":
3     print("Hi Patrick")
4 else:
5     print("Hello") |
Shell
>>> %Run condition1.py
Hi Patrick
>>>
Python 3.7.3
```

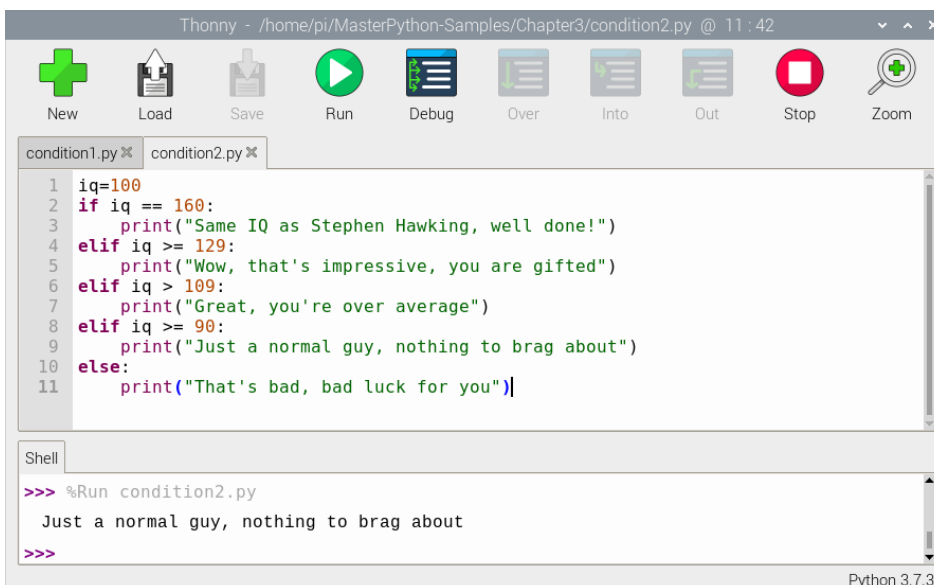
condition1.py

Ok, I have a few things to explain here:

- For each condition, the syntax is to use the main word (if, elif or else).
- For if and elif you'll find the boolean statement just after (value == "Patrick"). It's true if the username contains "Patrick", false otherwise.
- The else doesn't have any conditions, it will execute the code below if the username is not "Patrick".
- The ":" at the end of the line is here to indicate the end of the condition and the start of the related code.
- The spaces at the beginning of the line are essential. After the condition statement, each line starting with the spaces is linked to the condition above. If you don't put the spaces in, Python will consider that it's the following of the script, that should be done in any case, whether the condition is true or not.

Example 2

Let's take another example with more cases:



```
Thonny - /home/pi/MasterPython-Samples/Chapter3/condition2.py @ 11:42
New Load Save Run Debug Over Into Out Stop Zoom
condition1.py condition2.py
1 iq=100
2 if iq == 160:
3     print("Same IQ as Stephen Hawking, well done!")
4 elif iq >= 129:
5     print("Wow, that's impressive, you are gifted")
6 elif iq > 109:
7     print("Great, you're over average")
8 elif iq >= 90:
9     print("Just a normal guy, nothing to brag about")
10 else:
11     print("That's bad, bad luck for you")
Shell
>>> %Run condition2.py
Just a normal guy, nothing to brag about
>>>
Python 3.7.3
```

condition2.py

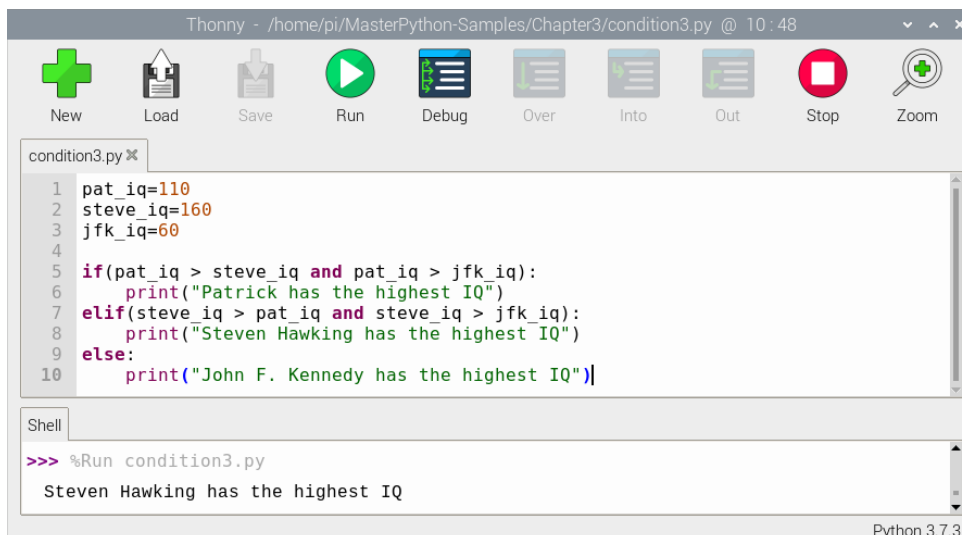
So, with this random example, there are a few takeaways for you:

- You can have one or more elif statements
- Conditions can work with all variable types. In the first example we used a string, here it's a number, but everything can work.
- There are several conditional tests you can use:

Equal	==	Not equal	!=
Greater than	>	Greater or equal to	>=
Less than	<	Less or equal to	<=

Example 3

The last example I want to show you happens when you have more complex tests to do.



```

Thonny - /home/pi/MasterPython-Samples/Chapter3/condition3.py @ 10:48
New Load Save Run Debug Over Into Out Stop Zoom
condition3.py
1 pat_iq=110
2 steve_iq=160
3 jfk_iq=60
4
5 if(pat_iq > steve_iq and pat_iq > jfk_iq):
6     print("Patrick has the highest IQ")
7 elif(steve_iq > pat_iq and steve_iq > jfk_iq):
8     print("Steven Hawking has the highest IQ")
9 else:
10    print("John F. Kennedy has the highest IQ")
Shell
>>> %Run condition3.py
Steven Hawking has the highest IQ
Python 3.7.3

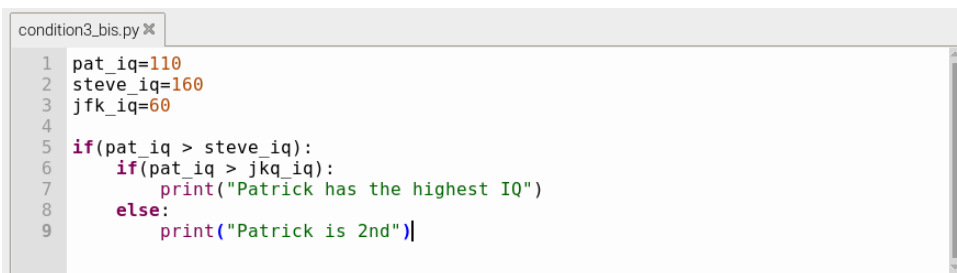
```

condition3.py

When you have more complex conditions and more variables, you can use the "and" and "or" keywords to check them in one line.

Note: *The smartest of you will have noticed that the script "conditon3.py" doesn't work with any values. Don't worry, we'll fix this in the action steps.*

You can also use nested conditions, and add another condition under a first one, for example:



```
condition3_bis.py x
1 pat_iq=110
2 steve_iq=160
3 jfk_iq=60
4
5 if(pat_iq > steve_iq):
6     if(pat_iq > jfk_iq):
7         print("Patrick has the highest IQ")
8     else:
9         print("Patrick is 2nd")|
```

condition3_bis.py

You might need to do this if you want the three in the correct order. But if you only want the highest IQ, the previous example is better.

ACTION STEPS

We have seen many things in this chapter, and if you are new to all of this, you might not be sure if you got it all right. That's exactly why we have the action steps at the end of each chapter. It's time to try all of this on your own, and see what you remember.

In this chapter, we learned how to create our first script ("Hello world"), how to use variables and concatenate them to another string. And just after that, how to use conditions to execute a different piece of code depending on some variable values.

Here is an exercise to try all of this:

- Write a simple code that displays "Good morning Patrick" on your screen.
- Create a variable with the first name, and set it to use your name instead of mine.
- Create a second variable with the time of the day (hour only), and change the message depending on the time of the day:
 - Good morning before 12pm.
 - Good afternoon between 12pm and 6pm.
 - Good evening after 6pm.

Tip: I recommend using a 24H time format to make this easier.

Try to do this alone if possible, read the chapter again if needed. But in any case, you can find the solution at the end of the book and in the code sample files.

For the most advanced, try to fix the issue we can have with the condition3.py script. Indeed, if two IQ values are the same, the script doesn't work. If you set JFK IQ to 160, it will still tell that Steve has the highest IQ. How can you fix this?

I won't give the solution for this one, you can come back to this later if you want.

Tip: What we have seen in condition3_bis.py might be useful for this.

PYTHON ADVANCED

INTRODUCTION

In the previous chapter, we have discussed the basics of Python and most programming languages: variables, conditions and text display. In a way, this chapter will be similar, as we still have many bricks to discover before starting our first project.

These two chapters probably aren't the most fun to read, but they provide the foundational knowledge you need to go further and to enjoy your projects in the future. I purposely removed concepts you won't use, so you can move quicker to the project chapters. However, there is still a lot to learn if you are a beginner, so hang in there as you'll need these concepts with any language in the future.

LISTS

Theory

I almost skipped this notion because I'm not sure if you'll use it very much. But it's something you might see in some tutorials, and using lists will make it easier to explain the next notion.

Lists are a specific type of variable. In the last chapter, we discussed numbers, strings and booleans. Lists allow us to store several values in the same variable.

The easiest way to illustrate this is to think of a shopping list. You can create a boolean variable for each item (`need_bread = true`, `need_milk = false`, etc.), but a better way is to have the items you need on one list.

Another way to imagine a list is to think about a column in a spreadsheet. Each line has a different value and the column is our list variable.

In practice

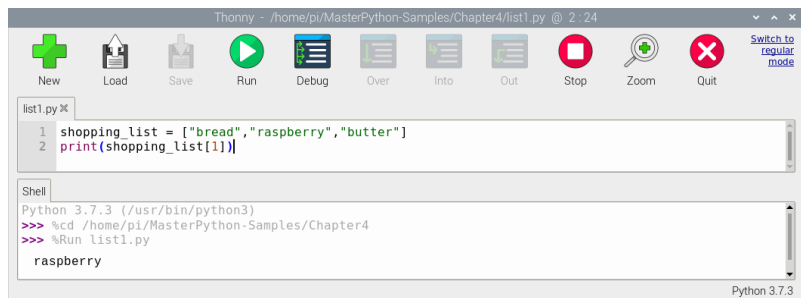
In practice, the syntax starts like any other variable, and we have a specific format to set the different values.

Here is an example:

```
shopping_list = ["bread", "raspberry", "butter"]
```

Once set, we can access each item in the list like this:

```
print(shopping_list[1])
```



The screenshot shows the Thonny IDE interface. The top toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. The main editor window displays a file named 'list1.py' with the following code:

```
1 shopping_list = ["bread", "raspberry", "butter"]
2 print(shopping_list[1])
```

Below the editor is a Shell window showing the execution of the script:

```
Python 3.7.3 (/usr/bin/python3)
>>> %cd /home/pi/MasterPython-Samples/Chapter4
>>> %Run list1.py
raspberry
```

The status bar at the bottom right indicates 'Python 3.7.3'.

list1.py

Note: The first item in a list has the index 0, that's why this example displays "raspberry" and not "bread".

There are other types of lists in Python, and several methods you can use to interact with them. We'll discuss some of them later in the book, but for now, the most important thing to understand is that you can create this kind of variable and access each element individually by knowing its position in the list.

LOOPS

Theory

Loops are not very complicated, but most people have a hard time when first introduced to the concept. Loops are essential in any programming language, so we'll take the time to properly introduce them.

Have you ever seen the factory scene in the movie Modern Times (Charlie Chaplin)?

Loops are a bit like this: do the same thing until I say something else.

In the movie, Charlie is a factory worker, who has to tighten bolts on each item on the moving belt.

In your code, you can have many cases where a loop is needed, for example:

- As long as the password entered is incorrect, ask for it again and again.
- For each file in a folder, do something.
- Infinite loop: Do the same thing continuously, or every X minutes.

This concept allows us to have a dynamic code that is dependent on something else.

It also avoids having several identical pieces of code, or having to execute the script too often. Let's see a few examples in practice, I hope it will make this even clearer.

In practice

In Python there are two types of loops you can use:

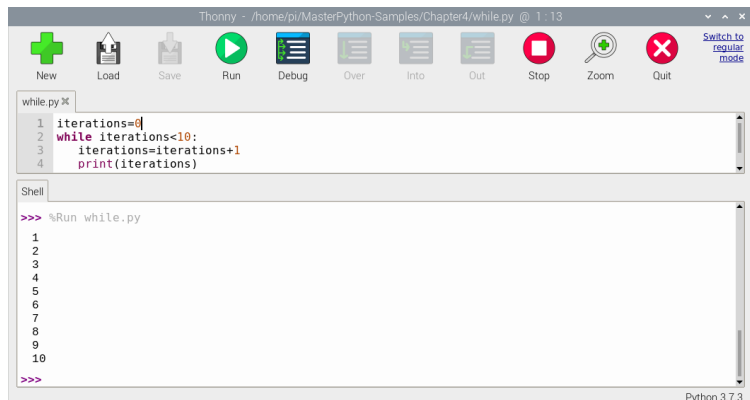
- **While**: it's a bit like a conditional statement (If), but the code inside is repeated while the statement stays true.
Ex: while there are items on the moving belt, tighten the bolts.
- **For**: in this case, it's for a predefined set of iterations, like the second examples below (do something for each file in a folder).
Ex: for each item on the shopping list, find them at the grocery store.

While

Let's start with the "while" loop, as it's similar to the conditions we discussed in the previous chapter.

Here is an example:

```
iterations=0
while iterations<10:
    iterations=iterations+1
    print(iterations)
```



The screenshot shows the Thonny IDE interface. The top toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. The main editor window displays a Python script named 'while.py' with the following code:

```
1 iterations=0
2 while iterations<10:
3     iterations=iterations+1
4     print(iterations)
```

Below the editor is a Shell window showing the output of the script:

```
>>> %Run while.py
1
2
3
4
5
6
7
8
9
10
>>>
```

The status bar at the bottom right indicates 'Python 3.7.3'.

while.py

As you can see, the code part under the "while" statement has been repeated 10 times.

At each new iteration, the condition is evaluated. If it's still true, the code below is executed.

Each time we increment the "iterations" value. When the "iterations" variable is 10, the condition becomes false and the loop is no longer executed.

We jump to line 5 (after the loop) and continue the script execution.

In this case, that's the end of the script, as you can have other lines after that.

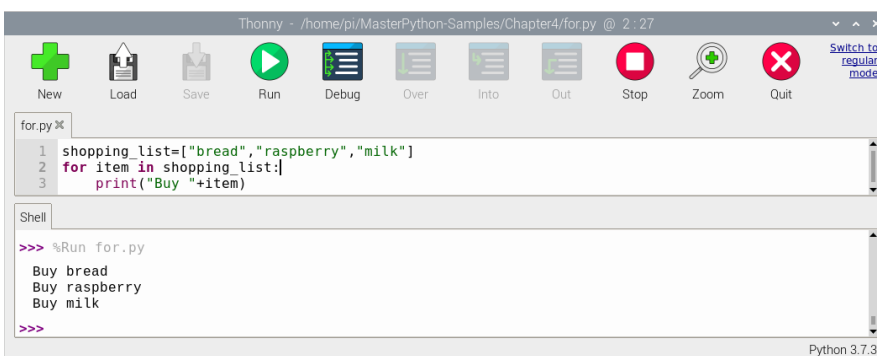
Note: Instead of `iterations=iterations+1`, you will also see `iterations+=1` in some scripts. It's the same thing.

For

"For" is the second type of loop you can use in your Python scripts. It's different from a "while" loop because you have a predefined set of executions you want to do. It can be dynamic (depending on the number of files in a folder), but it generally doesn't depend on the loop content, and there is no condition tested at each iteration.

Let's use the shopping list example to illustrate this:

```
shopping_list=["bread","raspberry","milk"]
for item in shopping_list:
    print("Buy "+item)
```



The screenshot shows the Thonny Python IDE interface. The top toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. The main editor window displays a Python script named `for.py` with the following code:

```
1 shopping_list=["bread","raspberry","milk"]
2 for item in shopping_list:
3     print("Buy "+item)
```

Below the editor is a Shell window showing the output of the script after running:

```
>>> %Run for.py
Buy bread
Buy raspberry
Buy milk
>>>
```

The bottom right corner of the IDE window indicates the Python version is 3.7.3.

The script displays each item individually and exits at the end of the list.

The "for" syntax uses two variables: for *<value>* in *<list>*.

At each iteration, the "item" variable is overwritten with the next value in your shopping list.

It allows us to do a specific action for each item (in this case, just display it).

Note: Loops can also be used with other variables in entry, like strings or range. The concept is similar, you don't need them for now.

FUNCTIONS

Theory

With functions, we'll push the idea of code repetition a bit further.

A function is a way to define a few lines of code, in a way that we can execute it by using its name anywhere in the script. A function can have parameters (as variable values set when you call it) and can also return something as a result.

So far, we have only seen concepts you absolutely need to know. But, the three remaining notions are more advanced. Read them, but you are allowed to put a little less effort into them, and come back later, once the previous concepts are fully understood and practiced.

Back to our function theory. In fact, you already used one function, maybe without knowing it was one: the `print()` function.

In almost each example, we have used it to display something on the screen.

"Print" is a function that takes a value in a parameter and displays it in the output log.

We have no idea what it does exactly, and it's the main benefit of a function. If you know which parameters they need, and what they will do with it, you don't need to know or understand the exact code being used.

To illustrate this, let's take another example: in your car you have accelerator and brake pedals. You can see them as functions. You know what it does to your car if you press them, but you don't have to know exactly how it works.

The "accelerator" function takes the pressure on the pedal as a parameter, and uses it to control the speed of your car. Engineers have worked once on the amount of gas being fed into the engine depending on this pressure, we trust them and just use the function. We don't redefine it every time we get into our car.

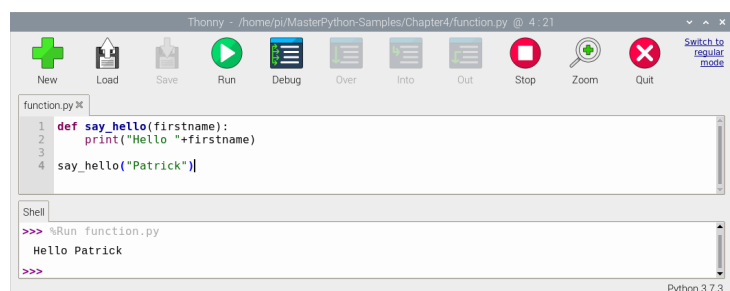
In practice

In Python, you can define a function once in your script, and call it as often as you want after that. I told you that this concept isn't mandatory, as it's especially useful in bigger projects, not really in the small scripts you will create most of the time on Raspberry Pi.

But in some cases you'll find them in samples you find online or in documentation, so it's better to have an understanding of them. For example, it's often used in robotics and GPIO interactions.

But let's start with a basic example as usual:

```
def say_hello(firstname):  
    print("Hello "+firstname)  
  
say_hello("Patrick")
```



function.py

I have a few things to explain here:

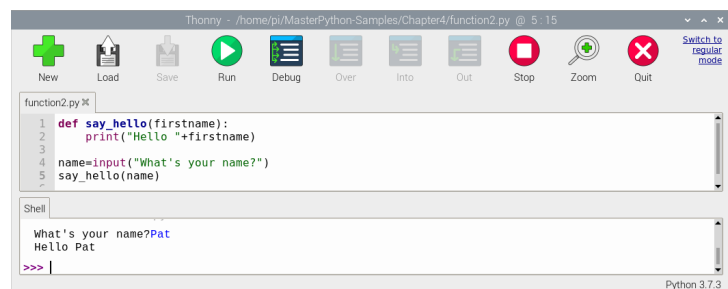
- `def`: It's the keyword we use to tell Python we are defining a function.
- `say_hello`: it's the name of the function, that we will use to call it later.
- `firstname`: it's a parameter we will send to the function, to display the correct name. Once set on the `def` line, we can use it inside the function as any standard variable.
- And on the last line we call this function, with a specific value to display "Hello Patrick".

So basically, this is a much more complicated way to code the "Hello World" example we learned in the previous chapter, but you get the idea. We'll see better examples in the next chapters.

It's still useful if you use it many times in your script, and want to change the greetings one day with "Hi <firstname>" for example. You will need to change it only once instead of everywhere.

I told you that "`print()`" is a function you can use in any script to display something. Python also provides other useful functions. For example, the "`input()`" function can be useful to have a dynamic way to say hello:

```
def say_hello(firstname):  
    print("Hello "+firstname)  
  
say_hello("Patrick")
```



function2.py

We no longer use a static name in this example, we ask the user directly with the "input()" function and display the greetings with the "say_hello()" function, using the answer as a parameter.

Note: Note that I'm not using two variables in this example: "name" and "firstname", just to show you that the parameter doesn't need to be the same in the definition and when you call the function.

If you are interested in the built-in functions in Python, you can find the complete list in the official documentation: <https://docs.python.org/3/library/functions.html>.

MODULES

Theory

Modules takes the whole idea of reusing code in your scripts a step further:

- **With loops**, we can repeat the same code several times.
- **With functions**, we can reuse the same code in different places in our script.
- **With modules**, we can reuse code from other projects (generally made by someone else).

Let's say you want to bake a raspberry pie, you can either try different ingredients, quantities and cooking times at random until you find the perfect recipe. Or you can just go online and take the one with thousands of positive reviews.

Using modules in Python corresponds to the second solution. If someone spent hours to create a module that does everything perfectly and shared it, we'll simply reuse it in your script instead of starting from scratch.

In practice

Usage

Python includes some built-in modules that you can use directly on Raspberry Pi, for example:

- **Time:** Provides many functions to manage date and time.
<https://docs.python.org/3/library/time.html>
- **Math:** Includes most of the mathematical functions you might need (rounding, exponential, square root, etc.).
<https://docs.python.org/3/library/math.html>
- **Tkinter:** Most functions in this module can be used to create a basic GUI in your program (message boxes, buttons, etc.).
<https://docs.python.org/3/library/tkinter.html>

Note: to find the entire list of modules available on your system you can start the Python console and type: `help('modules')`.

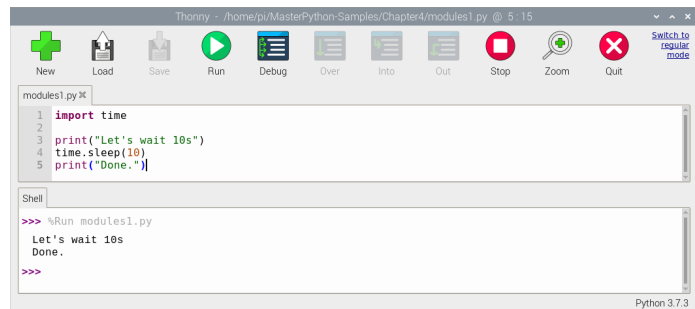
To use one module in a Python script, we need to import it, generally at the beginning of our program. Then we can use the functions listed on the help page.

Here are two examples that do the same thing:

Example 1

```
import time

print("Let's wait 10s")
time.sleep(10)
print("Done.")
```



modules1.py

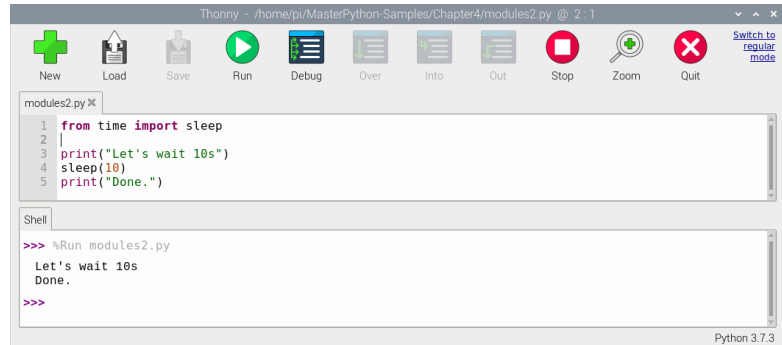
Two new things in this script:

- The import line, at the beginning, tells Python to use the "time" module. When doing this, all the functions included in this module are now available and can be used in this script.
- On line 4, we use "time.sleep(10)". We call the function sleep() from the "time" module.

Example 2:

```
from time import sleep

print("Let's wait 10s")
sleep(10)
print("Done.")
```



modules2.py

This second example does the same thing, but in this case, we only import the sleep function. The "time" module includes a lot of functions, and we only need one.

As we specify that we import the sleep function from the time module on line 1, we don't have to use "time." on line 4.

Installing new modules

It's possible to do many things with the included modules on Raspberry Pi, but you can also install new ones. We'll do this for almost all of our projects in the following chapters, that's why it's important to understand this process.

There are two ways to install new modules for Python: apt and pip.

APT:

You probably already know about the "apt" command, it's the one you can use to install new applications on Raspberry Pi OS (and any Debian-based distribution). Some Python packages can be installed with apt to add new modules available for your Python scripts.

For example:

- python3-gpiozero is a simple API to interact with GPIO pins.
- python3-picamera provides an interface to control the Raspberry Pi camera module.
- python3-sense-hat to import all functions related to the Sense HAT.
- Etc.

You can easily install any of these modules with apt, like this:

sudo apt update

sudo apt install python3-sense-hat

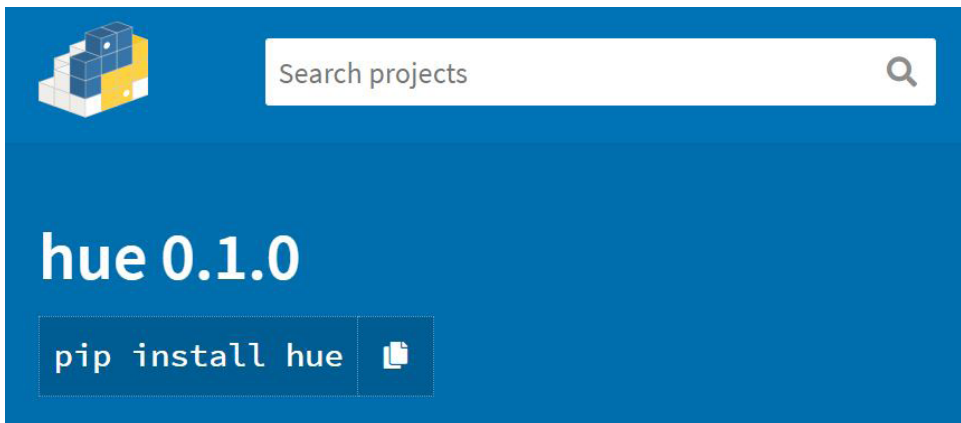
Note: Make sure to install the module for the version you are currently using. If you are programming with Thonny, it's Python version 3, so you need python3-*<module>*, not python-*<module>*.

PIP:

APT will work perfectly for modules related to Raspberry Pi and other very popular libraries. But if you need other modules (or more recent versions), you can also use PIP to install some of them.

PIP is the Package Installer for Python. You can visit this website to find any module you are looking for: <https://pypi.org/>.

For example, we'll use the "hue" module in the next chapter. Once on the module page, you'll see the command line you can use to install it:



Don't forget to use sudo to get administrator privileges, so:

sudo pip install hue

You can then use this module directly in your next script. You'll generally find documentation or at least a code example on the same page.

We'll install and use new modules in the next chapters, so don't worry too much about this for now, I just wanted to introduce them to you.

Note: same thing as with APT, use "pip3" instead of "pip" if you are programming with Python 3.

EXCEPTIONS

Theory

Yes, it's the last theoretical concept before our first project!

It's not the most important thing. In fact, I hesitated to include it in this book.

I don't think you'll use it in your scripts, but you will probably find it used in projects you copy/paste online, so we can take a few minutes to introduce it.

Exceptions are a best practice to handle errors in your programs.

At toll, if someone can't pay, the lane will be closed. This avoids additional problems with the lane (like car accidents or whatever).

Exceptions are similar, but they are automated. If something unexpected happens, your script can include a way to handle the issue automatically.

Here are a few examples:

- You ask for a pin code and the user answers with a password. It may crash your program because it's not the correct value type. You can create an exception to display an error explaining why it crashed (or ask again).
- You try to read or write a file, but the file doesn't exist. An exception can display a human-readable error instead of the default error code.
- You control lights with your script, but you stop the script on the Raspberry Pi with CTRL+C. The script can be programmed to turn off the light on exit.

Exceptions are really useful for larger scale projects, as they provide a better experience for the end user. But in most cases, you'll be the only user on Raspberry Pi, and if you understand the error code you don't really need them.

So, I'll just introduce them so that you can understand the code if they are used in a project you get somewhere else, but you'll probably create them yourself for now.

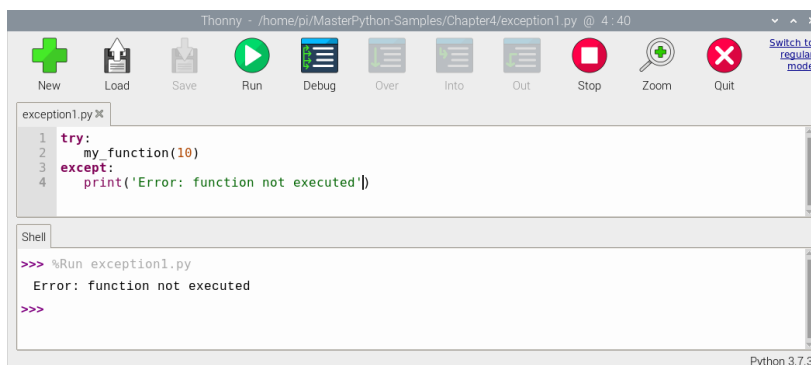
In practice

An exception is defined in two parts:

- The code to execute, written under the "try" statement.
- The code to execute if there is an error, written under the "except" statement.

Here is a basic example:

```
try:
    my_function(10)
except:
    print('Error: function not executed')
```



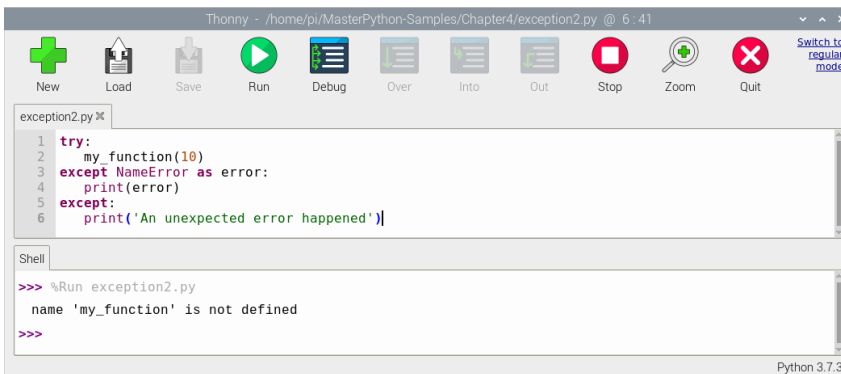
exception1.py

As the function "my_function" doesn't exist, the script won't work.

But with the try/except code, I can display an error message. On a bigger script, it might be useful to find where the problem is, but it doesn't help a lot.

In some cases, a better way would be to have several "except" statement, to handle each error scenario:

```
try:
    my_function(10)
except NameError as error:
    print(error)
except:
    print('An unexpected error happened')
```

The screenshot shows the Thonny IDE interface. The top toolbar includes icons for New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, and Quit. The main editor window displays the code from 'exception2.py' with line numbers 1 through 6. Below the editor is a Shell window showing the command '%Run exception2.py' and the resulting error message: 'name 'my_function' is not defined'. The status bar at the bottom right indicates 'Python 3.7.3'.

exception2.py

In this case, I have a first except block that is executed only if the "NameError" exception is raised. I can grab the error message with the keyword "error" and either display it on the screen or add it in a log file.

I also added an except block at the end for any other error that might happen in my script.

It's good to know that this syntax exists in Python, it's a good practice to use them for public programs, but I'm not sure if you'll use them for your projects on Raspberry Pi. Don't worry too much about this for now, it's time to practice :).

ACTION STEPS

It's time to put all we have learned in this chapter into practice. As a reminder, we have seen several advanced concepts:

- Lists
- Loops
- Functions
- Modules
- Exceptions

I hope you remember what each of these are and have a general idea on how to use them. If not, it's probably better to start by reading this chapter again and taking notes to refer to them later.

Here is the base recipe of a raspberry pie:

- 4 cups raspberries
- 1 cup sugar
- 3/4 cup flour
- 1 egg

I won't ask you to bake it, but to create a Python script to display the recipe:

1. Create one variable to store all the ingredients.
2. Display all of them, one by one, each one on a new line.
Wait one second after each ingredient before displaying the next one.
3. Create a function to display each ingredient individually, starting the line with a "- ".
The goal is to have a similar display as in the list above.

You can find the solution at the end of the book and in the code samples files, but try to do it yourself first. If you can, try to go a step further, maybe you can install and use other modules or add exception management in your script.

FIRST PROJECT: CONTROL YOUR CAMERA

INTRODUCTION

We have spent enough time on theory for now, so let's start putting what we've learned into practice. I always learn much faster by practicing than by reading theory tutorials. So let's start with something not too complicated to warm up and use the first concepts we discussed.

In this first project, we'll use Python to control a camera. We'll focus on the functions that allow us to take pictures and videos, but you can also include this project in a bigger one, where the camera is only one action in a whole program.

PREREQUISITES

As soon as we start working on projects, you'll need a few components to replicate the projects on your side. Don't worry, these are basic projects, so we don't need a lot of stuff.

For this first project, make sure you have:

- **A Raspberry Pi:** any model will be fine.
- **An SD card with Raspberry Pi OS installed.** A desktop interface is ideal to check the pictures directly, but you can continue with a Lite version if you prefer.
- **A camera module:** I'll use the official one, but it doesn't matter.

That's it.

You are welcome to check my resources page if you need any advice before buying your hardware: <https://raspberrytips.com/resources>.

You can also check my tutorial on how to install and configure Raspberry Pi OS if needed: <https://raspberrytips.com/install-raspbian-raspberry-pi/>.

Once done, install your camera on the Raspberry Pi, and enable it in raspi-config.

You can find all the information you need in this article: <https://raspberrytips.com/install-camera-raspberry-pi/>.

Before doing anything in Python, start by testing the camera with this command:

```
raspistill -o img.jpg
```

It should create an image file in your current directory.

If it doesn't work, it won't work with a Python script.

Once the command is working without error and the image file is viewable, you can move onto the next part.

TAKE A PICTURE

Basics

For this first exercise, we'll use a module named `picamera` to control your camera module in Python. It should already be installed on Raspberry Pi OS, but if needed you can always install it manually with:

```
sudo apt install python3-picamera
```

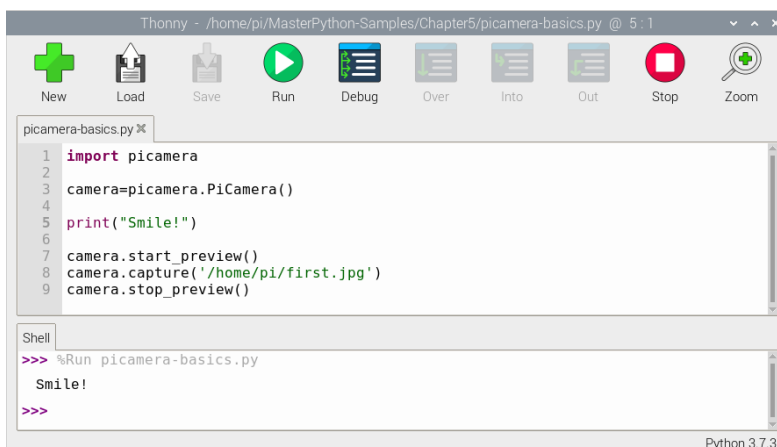
Create a new script in Thonny or your favorite editor, and paste the following lines:

```
import picamera

camera=picamera.PiCamera()

print("Smile!")

camera.start_preview()
camera.capture('/home/pi/first.jpg')
camera.stop_preview()
```



picamera-basics.py

This script is a digest of many things we've learned in the previous chapters:

- We import the module name "picamera" to use it in our code without redoing everything to handle the camera
- We create a new variable "camera", which is the result of the PiCamera() function. Don't be disturbed by the "picamera." prefix, it just tells us to use the function from this module (in case of conflicts with other modules).
- Then on line 7 to 9, we use three functions that are needed to take a picture

Note: you can find all the documentation related to the PiCamera library on this website: <https://picamera.readthedocs.io/en/release-1.13/>.

I know this script is really basic: we don't handle exceptions, we don't give the user time to smile, etc. But, I promised to only give you the important information, and I don't want to overwhelm you with the first real script :).

Timelapse

Ok, so basically our first script is the same as the "raspistill" command, it's not smart for us to create a Python script with 10 lines to do the same thing. Let's try to improve it a bit, and create a timelapse.

A timelapse is a video created from images taken at regular intervals that gives the impression to see what happens in an accelerated way.

So, we need:

- To add a loop in our script in order to create several pictures instead of one.
- To use a variable in the file name, as we want to avoid overwriting it at each shot.
- And to have a wait time between each photo.

Here is how you can do this with a few more lines:

```
from time import sleep
import picamera

nb_images = 10
waiting_time = 10
camera=picamera.PiCamera()

for image in range(nb_images):
    camera.start_preview()
    camera.capture('/home/pi/timelapse'+str(image)+'.jpg')
    camera.stop_preview()
    sleep(waiting_time)
```

picamera-timelapse.py

- I'm adding variables at the beginning, so we can easily change the values later. You can absolutely do it without them, as we only use them once in the script, there is no real need for a variable.
- Then we can use the second loop type from the previous chapter: "for". I voluntarily use a range in this loop to show you how it works. "range(10)" will create a sequence of number from 0 to 9, so 10 iterations.
- In the loop we can use the "image" variable created in the "for" statement to save the picture to a different file each time. We have seen how to concatenate strings in the third chapter of this book, and we need to use the "str()" function to convert the number into a string. We can't concatenate strings and numbers directly.
- After that, we just need to add a line with the "sleep()" function to wait for a few seconds before the next picture (next loop iteration).

Try to do it on your Raspberry Pi in front of a moving scene, it should be fun. You are welcome to play with the variables value to get a better result.

Note: Once you have all the pictures in the same folder, you can use a tool like *ffmpeg* to create a video with them. It should be something like:

sudo apt install ffmpeg

ffmpeg -r 24 -pattern_type glob -i '.jpg' -s hd1080 -vcodec libx264 -crf 18 -preset slow timelapse.mp4*

OPTIONS

There are many options you can add to improve the results you get in Python with your camera.

I already gave you the documentation website, where you can find all the details, but here are a few examples that might be interesting to play with:

- **Resolution:**
By default, the resolution is 1280x720 (with my camera at least). You can change it with:
camera.resolution = (1920, 1080)
- **Rotation**
I don't know about you, but I find my camera is always upside-down. If you also experience this issue, you can change the rotation parameter to fix this:
camera.rotation = 180
By adding this line, the image will be the right way.
- **Frame rate, iso and shutter speed:**
This is probably for more advanced photographers, but you can also change other options like the frame rate and shutter speed. It might yield interesting results.

Just set the variable value with:

```
camera.framerate = 0.5  
camera.shutter_speed = 300  
camera.iso = 800
```

If you are looking for something else, please check the documentation at <https://picamera.readthedocs.io/en/release-1.13/>.

RECORD A VIDEO

This example will be short as it's very similar to taking a picture, but just so that you know, you can also record a video with the picamera module.

Here is an example:

```
import picamera  
camera = picamera.PiCamera()  
camera.resolution = (1920, 1080)  
camera.start_recording('/home/pi/video.h264')  
camera.wait_recording(10)  
camera.stop_recording()
```

picamera-video.py

Two quick things to explain in this script:

- The `start_recording()` and `stop_recording()` function replace the `start_preview()` and `stop_preview()` we use for the image capture.
- The `wait_recording()` function is an improvement of the `sleep()` function. It also checks for errors while recording (for example if you no longer have enough disk space).

You can read the video.h264 file with VLC (preinstalled on Raspberry Pi OS).

ACTION STEPS

This chapter was an introduction to what you can do in Python, it should have been a smooth way to practice the theory we discussed in the first chapters. I hope it went well for you, as it wasn't supposed to be too complicated.

We already practice a lot in this chapter, which is the goal of the second part of this book. I hope you understand that you'll only learn by doing it yourself on your own Raspberry Pi. You'll make mistakes, have errors, etc. That's a good thing!

You won't learn if you just copy/paste some random code that works every single time.

Anyway, let's try a more difficult exercise:

- Create a Python script to create a timelapse: one shot every two seconds for 60 seconds.
- Create a function to take the picture. You'll call this function each time you take a shot. The function has at least the image prefix as a parameter.
- Knowing that the "time" module has a function named `time()` that returns the timestamp (= number of seconds elapsed since 1970). Example:
1622525260.3697095.
And that Python has a built-in function named "`round()`" that you can use to remove the decimals value of the timestamp. Example: **1622525260**. Change the function to use the integer part of the timestamp after the prefix in the image file name, example: **timelapse-1622525260.jpg**

Each bullet point is one step, try to do them in order. It's not a big deal if the last step is too hard for you, but try to at least do the first two.

After that, you can consider that you know how to interact with a camera in Python, and how to use functions, modules and loops effectively.

As always, you'll find the solutions at the end of this book if you are stuck with one question.

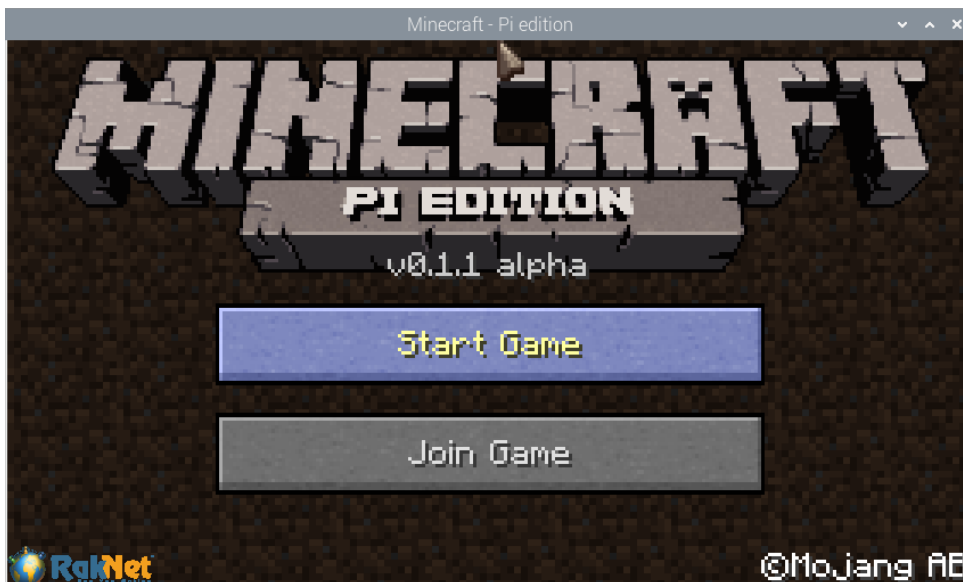
LET'S PLAY: INTERACT WITH MINECRAFT

INTRODUCTION

In this chapter, we'll try to use your new Python skills to play a game a bit differently than you might be used to.

On Raspberry Pi OS with Desktop, there is a special edition of Minecraft installed. It's possible to play with it, even if the features are limited, but that's not our goal today.

This Minecraft version is installed on Raspberry Pi because it's possible to interact with the game from a Python script.



As you probably remember, the main goal of the Raspberry Pi creation was to promote code in schools. What better way to do this than using one of the favorite games of these kids?

In this chapter, you'll try it for yourself, it should be fun. But first, I need to explain some details about Minecraft for those who haven't played this game for years (like me...).

MINECRAFT INTRODUCTION

Presentation

Just in case you lived in a cave the last ten years, here's a short introduction to Minecraft.

Minecraft is a sandbox game where the entire world is generated with blocks of the same size. The player spawns in this environment and tries to survive while doing anything they want. The game includes a mix of exploring, building, crafting, and combat. You can also fight with passive or hostile mobs (like zombies or cows).

Here is what it looks like when you start the game on a Raspberry Pi:



About the game

Minecraft exists in several game modes:

- **Survival:** you need to gather blocks and resources, craft things, and survive during the night.
- **Creative:** you get all the blocks you want for free and can't die.
- **Adventure:** for map creators, you can't break blocks, but you can use levers and buttons.
- **Spectator:** no interaction, you are always flying and can go through blocks.

On Minecraft Pi, as it's mostly an educational game, you are in the creative mode. You get a sword and some blocks in the quick bar at the bottom of the screen. It's possible to get more blocks (see the next paragraph), it's always sunny, and you can't die.

In creative mode, it's possible to break blocks in one shot. In survival mode, it depends on the tools you use (wood tools are slower than diamond tools for example).

Start a game

The first thing to do is to start the game and create a new world:

- Start the game (Main menu > Games > Minecraft Pi).
- Try to put the game on a side of your screen, you'll need space for the Python editor later.
- To move the window, click on the console blue bar with the small cursor (yes you have two ...).
- Then click on "Start Game".
- You're now in the "Select world" menu. Click on "Create New" to create your world.
- Wait a few seconds for the world to generate.

You can now control your player using the mouse to see the world around you. I'll give you all the control keys.

Controls

- **Camera:** Move the mouse.
- **Break block:** Left-mouse.
- **Place block:** Right-mouse.
- **Moving:** W,S,A,D (Z,S,Q,D on an AZERTY keyboard).
- **Jumping:** Space.
- Auto-jump is enabled when moving.
- Double-space enables the fly mode and then uses flying to gain altitude. Use Shift to decrease altitude.
- **Access inventory:** E.
- **Pause/Quit:** ESC.

Try to move around a little in your world to get used to the movements.

Coordinates

Minecraft uses coordinates to know each player's position, and each block has a different position.

The player's position is visible in the top left of the Minecraft window.

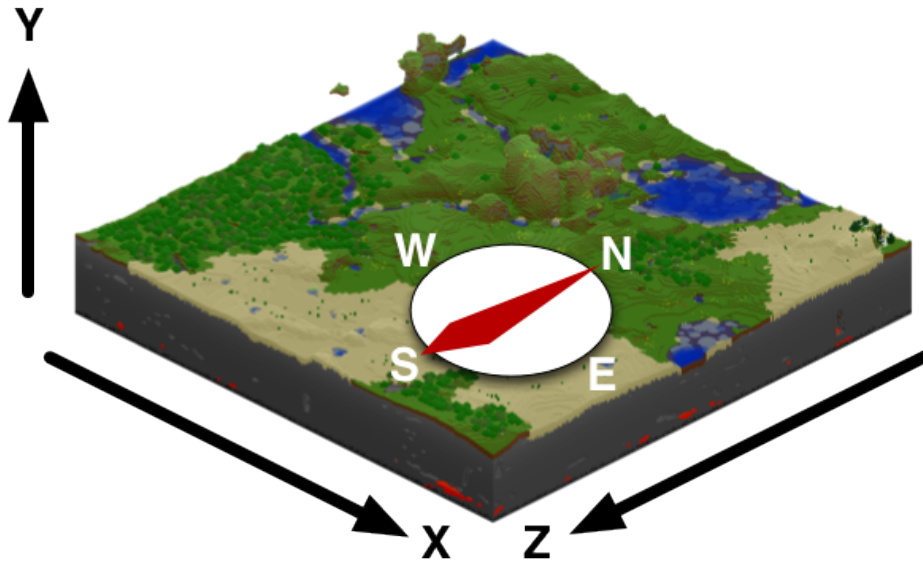
Try to move and see how it changes.

Each time you move one block away, one of these values changes by 1.

For example, you can have something like this:

- X: 4.6 – The east/west position.
- Y: 0.0 – The altitude.
- Z: -0.7 – The north/south position.

This picture should help you understand:



Try to move your player again in the game and see how the position changes in the indicator.

At any time, we know the player's position, and we'll use it later.

Blocks IDs

When you open the inventory (E) you can see that Minecraft has many blocks available:



Basically, each block in the game has an ID.
Stone is 1, Grass is 2, Dirt is 3, etc.

So if we want to set a specific block near the player, we only need to know its ID.
There is a website you can use to get the list of IDs:
<https://minecraft-ids.grahamedgecombe.com/>.

LINK MINECRAFT AND PYTHON

Now that you are up-to-date about the Minecraft game, we can try to interact with it in Python.

Chat

Similar to the first script we made in Python in chapter 3, we'll also try a "Hello world" with Minecraft first. This is tradition!

Keep Minecraft open on one side of your script, with a game started.

On the other side, create a new script with Thonny.

Then copy and paste these 3 lines in Thonny:

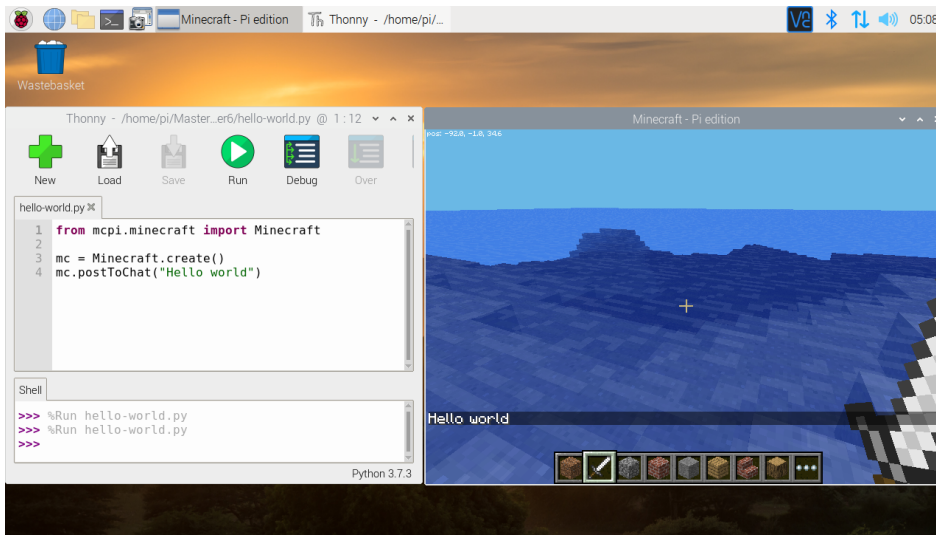
```
from mcpi.minecraft import Minecraft
mc = Minecraft.create()
mc.postToChat("Hello world")
```

hello-world.py

Almost nothing new here in the code:

- On the first line, we import the Minecraft module.
- On the second line, we define the variable "mc" that we'll use as a prefix with any Minecraft function.
It's really similar to what we have seen in the last chapter with the camera.
- After that, we can use the functions provided by the Minecraft module.
"postToChat" is one of them, as the name says, it's to write something in the chat ingame.

If you run this script, you should receive the "Hello world" message in your game:



Block

We can now try something funnier. It's possible to switch blocks in your close environment in the game.

Change a block at a specific position

The first thing you can do is to use the function "setBlock" to create a block at a specific position. This function has four parameters: the three coordinates and the block id:

```
from mcpi.minecraft import Minecraft
mc = Minecraft.create()
mc.setBlock(0, 0, 0, 56)
```

setblock.py

Try this in your game, change the coordinates if you are far from the 0/0 point. The order is x, y, z.

56 is the ID of the diamond ore, but you can use whatever you want.

When you run this script, you should see a diamond block appear at this position.

Get the player position

Instead of changing the coordinates in `setBlock` to fit your player position, we can use another function to get the player position: *"getPos()"*.

Here is the corresponding script:

```
from mcpi.minecraft import Minecraft
mc = Minecraft.create()
x, y, z = mc.player.getPos()
mc.setBlock(x, y-1, z, 56)
```

setblock-position.py

I use "y-1" to change the block under the player's feet.

We need to talk

There are two things I should explain at this stage, both on line 4 of this short script.

The first (simple) thing is that the `getPos()` function returns three values (for the coordinates) and that we can get them all assigned to specific variables like in this script.

Instead of writing:

`x = 2`

`y = 7`

You can do:

`x, y = 2, 7`

I'm not certain if it's clearer to read the second option, so I don't think it matches with the Python philosophy, but it's possible. And with a function returning several values it's really convenient.

The second (less simple) thing is that I lied a bit in the previous chapters about the functions we use from external modules (it was for your good!). These functions are special, as they come from a class, something used in object-oriented programming.

Until now, I only explained procedural oriented programming, which is the natural way of doing code for small projects.

I chose to skip this part in the theory chapter because it's often overwhelming for new students. It's typically used for large projects and not for creating simple scripts.

The idea is to add an abstraction level to your program, and group related functions in classes and objects.

Let's say you have a big program that handles everything in a bank.

You can do a giant script with hundreds of functions to manage employees, customers, bank accounts, payments, etc. Or you can use object-oriented programming and break it down in small parts:

- The customer class will include variables (name, address, etc.) and functions (new(), leave(), update(), etc.).
- The account class can include variables (type, amount, date_open, etc.) and functions (deposit() and withdraw(), etc.).
- Etc.

Anyway, a class is like a module. It includes variables and functions that you can use by creating an object using this class. An object is like a super variable, created from a class, that we can use to call functions from the class, like the functions we use in this chapter.

So as you read the scripts you have tested, just know that Minecraft is the main class and player is another class. That's why the syntax is a bit strange when we call the `getPos()` function.

`mc.player.getPos()`: "mc" is the object we create from the Minecraft class on line 3, "player" is an object automatically defined in the Minecraft class that has the `getPos()` function.

The point is used to call a variable or function from a specific class.

When we use `Minecraft.create()`, we call the `create()` function from the `Minecraft` class.

And by the way, a function defined in a class is named a method.

Don't worry if all of this seems really complicated, it isn't. It's just a different way to organize what we've learned up until now. And if you completely skip the object/class info and just remember the variable/function we discussed in the first chapters, you'll do OK.

You know that the `getPos()` function returns the information you need, and that there is a strange syntax to use to call this function, it's enough. Just copy/paste it.

I don't think you'll create classes and objects yourself if you create a program from scratch. That's why I choose to skip these notions. But when you use external modules, they are often made using these same concepts. Ironically, they used them to make your life easier :).

Teleportation

Anyway, let's get back to practice.

Another fun thing you can do is to teleport the player to another position.

To do this, you can use the "`setPos()`" function that works almost the same way as "`setBlock()`".

It has three parameters: `x`, `y`, `z`.

Here is a basic script on how to use it:

```
from mcpi.minecraft import Minecraft
mc = Minecraft.create()
mc.player.setPos(100,50,100)
```

teleport.py

Your player might move to a bad location, inside blocks or whatever. Try other values if it's not good the first time.

Note: *mc* is the object created from the *Minecraft* class. *Player* is an object created from the *player* class defined in the *Minecraft* class. We call the *setPos* method from the *player* object.

Other functions

There are many other functions available, so feel free to play a bit with them and try other things.

I didn't find an official API reference. I don't know if Mojang has this hidden somewhere, but I didn't find it.

The only page I found is the website linked to the "Adventures in Minecraft" book.

This website gives you all the functions, parameters, and explains what each does:
<https://www.stuffaboutcode.com/p/minecraft-api-reference.html>.

With the list of functions and examples, you should be able to use any of them now. And in fact, we'll see that in the action steps :).

ACTION STEPS

Your goal is to master Python, or at least to be at ease when you need to use external libraries. The Minecraft API is a great example. Even if you understand everything we discussed regarding Python, the syntax, the concepts, etc., you'll be quickly limited if you don't know how to use external modules and apply what it's in the documentation.

So for these action steps, I'll let you try exactly this.

You have the documentation website and we have seen how to use a few functions. Now, I'll let you work with them and others.

- Create a script that saves your player's position.
- Build a basic house around your player's position (less than 10 blocks away).
- If you're comfortable, try to use different block IDs to build the house, create a door, windows and a roof.

If it seems too hard for you, a 4x4x4 cube with only one block type will already be a great success.

As always, you can find the solution at the end of this book. But try to do it yourself first!

CONTROL YOUR LIGHTS AT HOME

INTRODUCTION

In this chapter, you'll experiment with the same skills we've discussed in a different environment. If you understood previously, this should be easy. If not, it's a great way to try again with another application.

More and more homes are equipped with smart lights. Depending on the brand and models, you can at least switch them on and off from your smartphone, and generally change the light intensity or even the color.

Most of these light bulbs have an API to control them in Python, which is what we'll learn in this chapter. I have the Philips Hue lights so I'll show you how with them, but I know other brands also have a similar API (Xiaomi and Govee for example).

PHILIPS HUE INTRODUCTION

The Philips Hue collection includes all of the smart lights from Philips. It's a concept including bulbs, a bridge, and an app to manage the lights.

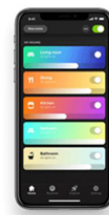
- **The bridge is the router between lights and your smartphone app.**
It needs access to your network (RJ45), and if possible to an Internet connection. The configuration is easy if you use a DHCP server on your network.
- **Lights connect to the bridge automatically through wireless.**
As soon as you switch the lights on, they are available on the app. No configuration needed.
- **And the Hue app is available on your smartphone to manage all of this.**
You can switch lights on and off, change colors, and create routines.



Hue lights



Hue Bridge



Hue app

If you don't have a Philips Hue package or colored lights at home yet, you must know that there are several models and packs available in the market.

There are the basic on/off white lights, bulbs where you can change the color and an LED strip that's perfect for Christmas or for background ambiance.

To experiment with the code that I give you in this chapter, it would be best to get at least a starter kit. You can find my current recommendation in my resources page: <https://raspberrytips.com/resources/expert/>.

If you have lights from another brand, the API will be different, but you should be able to adapt the code to get the same results. Most of these APIs have similar functions available.

If you can't afford smart lights for now, this chapter will be less fun, but you can probably follow what I explain and apply it to something else. Or maybe use the code and try to display something on your screen instead of changing the light status or color.

PREREQUISITES

Before anything else, we need a few things:

- If you bought a new kit for this chapter, follow the instructions to install it. Configure all of the lights with the app on your phone before you do anything else.

- Find the Hue bridge IP address. You can get it by using this command on your Raspberry Pi:

```
sudo apt install nmap
```

```
nmap -sP 192.168.1.0/24
```

Replace the network with your own if you use something different.

- Install the Python library that we'll use to control the lights:

```
sudo pip3 install phue
```

The package "python3-pip" should be installed first if you have any errors with this.

Note: Similar to the way I previously told you to install packages named `python3-<library>`, you will need to install libraries with `pip3` instead of `pip` when you use Python 3 in your scripts.

DISCOVER THE API

On your Hue bridge, there is an API available.

The base URL is `http://<IP>/api/v1`.

I'll show you basic calls you can make, how it works, and after that, we'll create the first scripts.

In this project, the Raspberry Pi will talk to the Hue bridge, with Python making HTTP requests to get or change the Hue system configuration. So, the Hue API gives us HTTP functions to change things for the lights (like switch a light off for example).

Unlink the one we used for Minecraft in the last chapter, this API is protected with a username we need to create before making any calls (You may have seen the "unauthorized user" error if you tried the API base URL).

API Tester

To test the API without coding, Philips gives us a tool to create requests in a form. This form is available at `http://<IP>/debug/clip.html`.

In this form you'll find three parts:

- **URL:** The URL you want to call.
You'll see later that each function has a different URL.
- **Method:** GET, PUT, POST or DELETE.
These are different actions you can make with your request.
For example, GET is to see the configuration while PUT is to change something.
- **Message Body:** For specific functions, you must add more details to explain what you want to change in the Hue configuration.
- **Command Response:** When you send requests, you'll see the bridge's answer in this field.
This will help you debug your requests.

Now that you understand what it is, let's start by creating your username.

Create the user

In most requests, we need authentication to allow access to the Hue bridge configuration.

To create your authentication token, fill the form like this:

- URL: /api

Message Body:

```
{"devicetype": "TestApp#RaspberryPi"}
```

Put what you want for the device type (app name or device as you want).

- Then press the bridge button.
- In the following 30s, hit the POST button to run the query.

This should look like this in the form:



The screenshot shows the CLIP API Debugger interface. It has three main sections: URL, Message Body, and Command Response. The URL field contains '/api'. Below the URL field are four buttons: GET, PUT, POST, and DELETE. The Message Body field contains the JSON string `{"devicetype": "TestApp#RaspberryPi"}`. The Command Response field shows the JSON response from the Hue bridge:

```
[ { "success": { "username": "2AmTZoTjz0nymGCEa-8e-14wRs13KGmIsCSqeHgM" } } ]
```

In the "Command response" field, you can see the username generated by the Hue bridge.

Note it, we'll use it in all the next calls.

Get a list of the lights

Before changing the configuration in a specific room, we need to get the list of all lights installed. Each light bulb has an ID that we'll need later in our scripts.

To get a list with all lights available, fill the form with this:

- URL: `/api/<username>/lights`.
Replace `<username>` with the one you got in the previous step (long random string).
- Then press "GET" to get the answer.

The answer is in JSON format, which isn't easy for a human to read. You should see one paragraph by light.

Each paragraph starts with a number (1, 2, 3, etc.) and will contain information about the corresponding light.

Try to find which number corresponds to which light.

For example: 1=>Bedroom, 2=> LED strip, etc.
Take notes of these IDs, as we'll need them now.

First try in Python: Switch the light on and off

```
from phue import Bridge
b = Bridge('192.168.222.2')
#Uncomment this line to register the app (see below)
b.connect()
#Change the light state
b.set_light(1, 'on', False)
```

switchoff.py

Try this code in your favorite editor, but don't forget to change the bridge IP address first.

You may need to change the light ID on the last line, if "1" is already off or if you can't see it from your computer.

As you have done when using the API debugger, you also need to create a username to use the API on your Raspberry Pi. To do this, you have to use the function "connect()" once.

So:

- Press the button on the bridge.
- Run this script.
- If it works, comment the line "b.connect()".

You'll get an error if your Pi is not authorized or if you didn't press the button:

```
'The link button has not been pressed in the last 30 seconds.')
```

```
phue.PhueRegistrationException: (101, 'The link button has not been pressed in the last 30 seconds.')
```

Once everything is set up correctly, this first script should be easy to understand for you:

- On the first line we import the Bridge class from the phue library.
- Then we create an object, "b" that can call all of the methods from the Bridge class.
- We use the "connect()" method the first time, but we don't need it after that. Once your Pi is authorized, the bridge will allow immediate access to the API.
- Finally, we use the "set_light()" method to change properties on a light bulb. In this case, we turn it off. Yes, "on=False" is basically "off=True".

Change "False" to "True" on the last line to set your light back on.

CREATE A CHRISTMAS TREE

Now that we have set everything up, and have learned the basics on how to interact with the lights, we can fast-forward to something fun. If you have the LED Strip from Philips Hue, you can copy and paste my code as it is.

If you don't have the LED Strip from Philips Hue or use another brand, you'll need to adapt the code, but that's great too. You now have enough knowledge to start experimenting on your own.

Don't worry, I'll explain everything so that it'll be easy to do it on your side.

The goal here is to use several concepts we have previously discussed in order to code a Christmas tree.

Change the light color

With lights from Philips, the color value is defined with two numbers: x and y.

X and y are numbers between 0 and 1, and changing their values will change the light color.

There is a complicated formula to convert RGB colors to XY colors, but we don't need it in this project. We'll set random colors.

In fact, the easiest way I have found to get a specific XY color, is to set it with the app on your smartphone, and then use the API Debugger tool to get the exact value.

Another way is to use the "rgbxy" library, that can convert RGB and HTML colors to XY equivalents directly in Python (more details here: <https://libraries.io/pypi/rgbxy>).

Anyway, to change the color, we can use the same method as before:

```
b.set_light(1, 'xy', [0.2,0.7])
```

You can try to change the previous script and try different values (between 0 and 1) to see how it works.

Note: when we got all the lights in the API debugger, you may have seen other variables like brightness and saturation. You can change their values the same way.

Merry Christmas

We'll now put this in practice and gradually change the color of your LED strip to act like a Christmas tree garland.

- To keep things simple, we won't change the value of "y", it will always be 0.
- We'll increase the value of "x" in increments of 0.03.
Ex: 0, 0.03, 0.06, etc.
- When x reaches 1, its maximum value, we'll flip the direct and decrease the value of "x". Then we'll repeat when it's back to 0.

I would love it if you can try to do it on your own before reading my script. I think it's a great exercise to practice and experiment with the theory we discussed in the first chapter.

Hint: you'll need variables, conditions, an infinite loop and obviously the phue library.

Anyway, here is my script:

```
from phue import Bridge
from time import sleep

b = Bridge('192.168.222.2')
led_strip = 3

b.set_light(led_strip, 'on', True)
x=0
y=0
direction=1

while True:
    if x>1 or x<0:
```

```
        direction=1-direction
    if direction:
        x=x+0.03
    else:
        x=x-0.03
    print(str(x)+"/"+str(y))
    b.set_light(led_strip, 'xy', [x,y])
    sleep(0.5)
```

switchoff.py

- We start by defining a bunch of variables. It's always better to work with variables, especially with explicit names.
- Then we create an infinite loop "while True:" because we want the light to change color all the time while the script is running.
- If the x value has reached one of its limits (0 or 1) we need to revert the incremental steps direction.
- If the direction is positive, we add 0.03 to the last value. If it's negative, we remove 0.03 to the last value.
- We use the method to change the light color and wait a bit before changing the color again. You can adjust the sleep time to make it change faster or slower, but try to keep a minimum value here.
- I added one line to display the value for those who don't have smart lights, but also to see if the x value changes as expected.

And that's it; a fun project with less than 20 lines of code. I think you should understand all of the code now, and start to see the power of Python in a concrete project.

I think it's a good idea to take time to experiment from there.

If you don't have the Philips lights, you may have already worked enough. However, if you just copied and pasted my script, try to do something different. Maybe you can try to change the y value for example.

ACTION STEPS

This action step is a big one, but it's good for you to think more on your own now that the code logic and the Python syntax is understood. I think half the work when programming is to find a way to write the logic in a simple script, whatever the language you use.

Goal

Anyway, the goal in this exercise is to automate the lights at home. We'll define a routine to light them at a specific time and shut them down at another time.

If you don't have smart lights, please try it as well. Just replace the `set_lights()` function with a `print()` function.

Here is what I have done at home:

- My lights come on at 4:30 a.m. (yes, I get up early).
- They turn off automatically at 8 p.m.
- Do they stay on all day? Nope, I have a smart configuration that I want you to try here.

I connect to an API to get the sunrise and sunset times of the current day at my location. In the morning, the lights turn off 2 hours after the sunrise. Then I put them back on 2 hours before sunset.

At the time of writing (in the summer), the sunrise is at 3:50 in the morning, so my lights are on from 4:30 to 5:50. Sunset is 10 p.m., so the lights don't come back on in the evening, as I don't need them before 8 p.m. I have found that a 2-hour offset works well to make sure I get enough lights when there is bad weather.

You'll need to adjust this depending on your lifestyle and sun exposition, but basically that's the idea in this exercise.

Automate the static hours

The first part shouldn't be very complicated.

Create two scripts: `lights_on.py` and `lights_off.py`.

These scripts correspond to the first two bullet points in the previous goal's details.

For example, you'll schedule `lights_on.py` to run at the time you wake up and use `lights_off.py` at the time you want to turn the light off.

We have already discussed this in this chapter, but you might need some help to schedule them.

Task scheduling on Linux is controlled by a service named "cron".

You'll find everything you need in this article:

<https://raspberrytips.com/schedule-task-raspberry-pi/>.

Read this article and program your scripts.

Your smart lights will now turn on and off at the specified times, but they will stay on all day. We need to work on the last point to turn them off and depending on the sunrise and sunset times.

Connect to the API

When you have complex projects, it's always a good idea to break them down into smaller parts. The first easy step was to schedule the first two scripts at specific times.

Now, we'll find and connect to an API to get the sunrise and sunset times. But we won't use them directly, this step is just to get them in a new script.

To do this, I'm using the API provided for free by this website:

<https://sunrise-sunset.org/api>.

It's simple to use, as all the parameters go directly in the URL, for example:

<https://api.sunrise-sunset.org/json?lat=36.7201600&lng=-4.4203400>.

The result is in JSON format and looks like this:

```
{
  "results":
  {
    "sunrise": "7:27:02 AM",
    "sunset": "5:05:55 PM",
    "solar_noon": "12:16:28 PM",
    "day_length": "9:38:53",
    "civil_twilight_begin": "6:58:14 AM",
    "civil_twilight_end": "5:34:43 PM",
    "nautical_twilight_begin": "6:25:47 AM",
    "nautical_twilight_end": "6:07:10 PM",
    "astronomical_twilight_begin": "5:54:14 AM",
    "astronomical_twilight_end": "6:38:43 PM"
  },
  "status": "OK"
}
```

Here are the steps you have to follow to use this API:

- Create a new script, something like lights_auto.py.
- Find a Python library that you can use to reach an URL and get its content.
- As the response will be in JSON, you also need a library to read JSON format.
- Finally, we'll compare the sunrise and sunset times to the current times. You can use the datetime library we have seen in chapter 5 to do this.

In most projects, Google will be your best friend to find the libraries you need and how to use them. If it's your first project and have no idea where to look, I will help you a bit.

Here are the three libraries we'll use in this exercise:

- URLLib: <https://docs.python.org/3/library/urllib.request.html>
- JSON: <https://docs.python.org/3/library/json.html>
- Datetime: <https://docs.python.org/3/library/datetime.html>

Start creating your script. Don't forget to do it step by step:

- First open the URL and make sure you can get the result.
- Then use the JSON library to convert it to simple variables.
To access a specific value in a JSON formatted response, you need to use the library to convert it to a Python list.
Then you'll use the corresponding entry in square brackets, like the lists we have seen in chapter 4, but with the index name instead of a number.
For example: *results['sunset']*
- Once you have the times in two variables you can use the datetime library to find the difference between them and the current time.
- The goal here is just to have two variables: `diff_sunset` and `diff_sunrise`.
For example, `diff_sunrise` can be 2000 and `diff_sunset` 55000.
- We don't need to do anything with the lights yet.

Don't be overwhelmed by the number of steps you see, just work on each one. Make sure it works as expected and then move to the next one once you are ready.

You can find my version of this script at the end of the book, but I'm sure there are several ways to do it, so if your script works it's probably fine.

Adjust the lights schedule with the API results

The hardest part is done, so now we just have to include the `phue` library and switch the light on and off depending on the time of the day. We'll use the same functions we have previously discussed, and just add a few conditions to make this work as expected.

As a reminder, the goal is to turn on the lights 2 hours before sunset and turn them off 2 hours after sunrise.

Here are a few hints to guide you in this final step:

- If the sunset is in less than 2 hours, we'll turn on the lights.
- If the sunrise is in less than 22 hours, we'll turn the lights off.
Yes, that's a shortcut, you can do this differently if you find a better way.
- There are 3600 seconds in one hour.
Don't forget that `diff_sunrise` and `diff_sunset` are in seconds.
- There is nothing to do between sunset and sunrise.
We have the scheduled tasks for this.

If it works (alone or thanks to the solution), feel free to adapt it then to your schedule. Note that there is also a way to gradually decrease the light intensity. That could be a great add-on before turning the lights off at night.

I hope you didn't have too much difficulty with this exercise. It was complicated on purpose, but when broken down into small steps it should be doable. If you think you are a little behind, don't worry. Just take some time to check the solution again, maybe quickly review chapters 3 and 4 to see what you have missed.

But most of the time, the difficult part is the logic, not the syntax. You'll understand the solution, but not how to get there. Don't worry, it will come automatically through more and more projects.

COMMUNICATE WITH THE WORLD

INTRODUCTION

This chapter is probably easier than the previous, but it's also important.

Sometimes you create Python scripts that run on schedule (like the lights on/off in chapter 7), and you don't look at your script after that. If your lights don't work anymore, you'll surely notice, but for less visible scripts this might not be the case.

The goal of this chapter is to use Python scripts to send messages. I suppose it will mostly be notifications to yourself, either by email or push notifications on your smartphone, but it can also be useful if you have other people to contact with.

For example, I have a script that connects to my NAS, where I store my backups, that emails me if there are no recent files. This could be really useful for some Python projects, which is why I included this in this book. And, with your new skills, it shouldn't be very complicated.

EMAILS

When you configure a mail client on your computer, it will ask you for an SMTP server to send your messages to. Python works the same way. We'll open a connection to your mail server and send the message from there.

I'll show you how to do this with Gmail, as it's the most popular mail provider, but it should work with any mail server that supports SSL.

Before anything else, you'll need:

- The email server address.
Ex: smtp.gmail.com
- The SMTP port.
Ex: 465
- Your username and password.

Sending an email with Python

As usual, there are a few libraries that we'll use to do this:

- smtplib: <https://docs.python.org/3/library/smtplib.html>
- ssl: <https://docs.python.org/3/library/ssl.html>

As we'll send the email to the SMTP server with SSL, we need both of them to do this. Let's not waste time here, that's the script you can use to send an email in Python:

```
import smtplib, ssl

#Define your server parameters
server = "smtp.gmail.com"
port = 465
user = "you@gmail.com"
password = "yourpassword"

context = ssl.create_default_context()

#Create your email
dest = "contact@raspberrytips.com"
subject = "Your Python script works"
body = "And your book is great by the way :)"

#Assemble the message
message = f"""Subject: {subject}
To: {dest}
From: {user}
{body}"""

with smtplib.SMTP_SSL(server, port, context=context) as server:
    server.login(user, password)
    server.sendmail(user, dest, message)
```

basic-email.py

A few things to explain here:

- The SSL context includes all the default settings to establish the SSL connection later, we don't need to get into more details here, it's just required to make this work.
- The message part of this project is new. When you have multiline variables to concatenate you can use the triple quotes to set the value. By using the "f" before the triple quotes, and then putting variables in braces, they will be replaced by their value. It's a strange syntax, but very convenient in this case. Don't hesitate to print the message value to see how it works.
- The "with" statement then is a different way to define a variable. Basically, it's the same thing as this:
`server = smtplib.SMTP_SSL(server, port, context=context)`
But it will close the connection once the paragraph below is executed. It's often a good practice to use this when working with unmanaged resources (like files and connections). The file or connection will be closed even if there is an exception, for example if the connection doesn't work or your password is rejected.

Note: Your usual password might not work in this script. For example, with Gmail you need to generate an app password first, that will work without the 2-steps authentication process. You can do this here: <https://myaccount.google.com/apppasswords>.

Attachments

Sending attachments can also be something interesting in your scripts. For example, if you control a security camera with Python, you can send the picture in an attachment when a movement is detected.

It's the same base, but there is a new object to introduce: the MIME part.

MIME stands for "Multipurpose Internet Mail Extensions" and as the name says, it's used to extend an email message and add an attachment.

To define a MIME attachment, we'll need a few methods, but all of them come from the email library: <https://docs.python.org/3/library/email.html>.

As always, I'll give you the code, and explain after - just in case there are any questions left:

```
import smtplib, ssl

from email import encoders
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

#Define your server parameters
server = "smtp.gmail.com"
port = 465
user = "you@gmail.com"
password = "yourpassword"

context = ssl.create_default_context()
```

```
#Create your email
dest = "contact@raspberrytips.com"
subject = "Your Python script works"

#Assemble the text message
message = MIMEMultipart()
message["From"] = user
message["To"] = dest
message["Subject"] = subject

#Add the attachment
filename="attachment.txt"
with open(filename, "r") as attachment:
    part=MIMEBase("application", "text")
    part.set_payload(attachment.read())
    encoders.encode_base64(part)
    part.add_header("Content-Disposition", f"attachment; filename= {filename}")
    message.attach(part)

text = message.as_string()
with smtplib.SMTP_SSL(server, port, context=context) as server:
    server.login(user, password)
    server.sendmail(user, dest, text)
```

attachment.py

- We still need `smtplib` and `ssl`, but we also import the functions we need from the email library. As it's a big one, we only include the modules we'll use.
- The message definition is different, but maybe easier to read than the syntax we used in the previous script.
- Then we need a file that we'll add into the attachment. In this case, it's a text file, creatively named "attachment.txt".
As you can see, we use the "with" statement to open a file, like for the SMTP connection.
- The open function takes two parameters, the file name and the open mode.

r	Read	x	Create
a	Append	t	Text (default)
w	Write	b	Binary

- Then we use the MIME library to attach this file to the message.
- Once done, we can use the `smtplib` functions to send the email, like in the first example.

Note: if you want to attach a picture instead of a text file, you need to make a few changes in the paragraph where we open and crease the MIME payload. Use "rb" and "image" instead of "r" and "text" for example. Check the documentation for more details.

You can use these two scripts as a base in your projects. Once you understand the logic, it shouldn't be very complicated to adapt them to your needs.

NOTIFICATIONS

Emails are great, but for the most critical information I tend to use push notifications. This way, there is no spam, no filter and no need to wait until I check my emails. It's guaranteed that I'll get the information quickly.

Sending push notifications to Android and iOS phones is not an easy task, you'll need to create an app first. And, that's not the goal of this book. We want to execute something simple.

The easiest way I have found to do this is to use an app named "Pushbullet". It has an API and a Python library to help us with this task.

Install Pushbullet

Pushbullet is a free service. They have a Pro plan, but I don't think you'll need it anytime soon. I've been using their free service for years and it's perfect.

Start by creating your account on <https://www.pushbullet.com/>.

You can easily do this by using your Google or Facebook account.

Once done, install the Pushbullet apps:

- On Android, you can find it in the Play Store.
- There is an extension for Chrome and Firefox.
- And it's also available as an app on Windows.

You'll find everything there: <https://www.pushbullet.com/apps>.

Unfortunately, if you are an iPhone/Mac user, you'll have to use the browser extension, as there doesn't seem to be anything else available for you.

Once your account is created and the app installed on at least one platform, we can start to use it with Python.

Use the Pushbullet API

Back on your Raspberry Pi, the first thing is to install the libraries with PIP:

```
sudo pip3 install pushbullet.py
```

You'll need your access token from Pushbullet for this first test. You can find it in your account settings: Settings > Access Tokens > Create Token.

Keep it safe, you'll need this each time you want to send notification in your Python scripts.

Let's create a first script to try this:

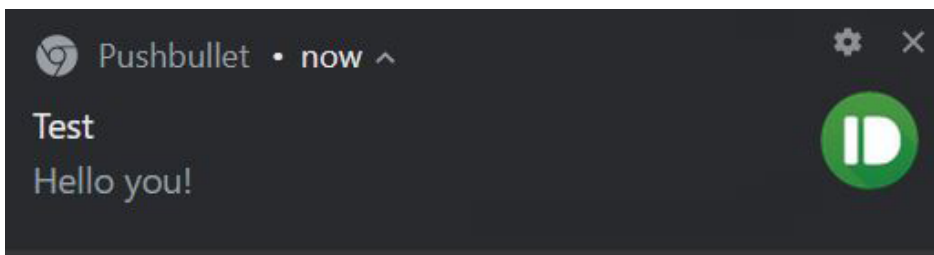
```
from pushbullet import PushBullet

token = "YOUR_TOKEN"

pb = PushBullet(token)
pb.push_note("Test", "Hello you!")
```

attachment.py

That's it, you should receive a push notification:



This is the minimum code portion you'll need in a script to send a notification to your phone or computer.

Obviously, you can do a bit more than that, such as include links or pictures in your notification, delete sent pushes or send a push to only one specific device.

You can find more details about this in the official documentation and library GitHub:

- Official documentation: <https://docs.pushbullet.com/>
- Library documentation: <https://github.com/rbrcsk/pushbullet.py>

Note: *Never name your script the same as the library name, "pushbullet.py" won't work. You should always use a different name to avoid conflicts.*

ACTION STEPS

In this chapter, we have learned that even if scripts are running alone on a Raspberry Pi, we often need notification when something goes wrong (or a user action is required). You now know how to send email and push notifications in these situations.

Overall this chapter wasn't as complicated as the last one. The syntax to send an email might be a bit strange, but nothing is impossible for you now! That's why in these action steps, we'll again use the API from the last chapter for a quick exercise.

The goal is to send a push notification each day at midnight with the sunrise and sunset times for the next day:

- Create a new script with the required libraries.
- Define the location coordinates and the API token for Pushbullet.
- Send a well formatted push with the sunrise and sunset times.

Once done, you can consider that you should know how to work with APIs and JSON formats, which is essential in many scripts you'll write in the future.

A FINAL PROJECT: THE SENSE HAT

INTRODUCTION

I think I have already taught you everything you need to achieve any project where Python scripts are required. This chapter is just the ultimate project to put everything into practice together: variables, functions, loops and even some hardware.

The Sense Hat is the most popular HAT on Raspberry Pi. It's an expansion card, created by the Raspberry Pi Foundation, that provides many sensors, a LED matrix and a joystick. Python code can be used to control all of this.

In this chapter, we'll use it to do something fun, but it shouldn't require new skills in Python. You should already know everything, this is just a giant practice chapter.

ABOUT THE SENSE HAT

Presentation

Originally, the Sense HAT was created by the Raspberry Pi Foundation as "Astro Pi" (you may have heard this name). The goal was to send a few Raspberry Pi with many sensors onboard the International Space Station (ISS). After this successful flight, the Sense HAT was created as a commercial product that was available for anyone on Earth :).

The sense HAT provides many new sensors to the Raspberry Pi:

- Accelerometer (get the movement speed of the PI)
- Gyroscope (capture the rotation movement of the Raspberry Pi)
- Magnetometer (magnetic field measurement)
- Air pressure sensor
- Temperature and humidity sensors

And there is also an LED display matrix and a joystick on the top of it. Everything is controllable in Python scripts, which we'll discuss in the next part of this tutorial.

Do I Need One?

As you are interested in Raspberry Pi and Python, I highly recommend getting a Raspberry Pi before reading this chapter. It has so many sensors and features that you can use in different projects, it's not expensive and is one of the most popular extensions (so it's easy to find help about it). Also, it's made by the Raspberry Pi Foundation, so it works on any Raspberry Pi model.

For more details and to check the price, visit my resources page here:

<https://raspberrytips.com/resources/intermediate/>

Whatever you choose, know that it's possible to follow this chapter without it because there is an emulator available on Raspberry Pi OS. You can start the application and interact with it instead of the physical HAT.

If for any reason you can't get the Sense HAT, just replace this line in all of the scripts I give in this chapter:

```
from sense_hat import SenseHat
```

with:

```
from sense_emu import SenseHat
```

By doing this (and running the emulator), you should be able to experience the same results.

THE SENSE HAT API

The Sense HAT has a Python API available with all the functions needed to interact with the sensors and the LED matrix. I will explain the main functions you can use now, but if in the future you need additional information, you can review the documentation here: <https://pythonhosted.org/sense-hat/api/>.

As this is a product from the Raspberry Pi Foundation, which is intended to learn how to code, there is nothing too complicated in it. With your current skills in Python, it should be pretty straightforward.

Note: The *sense-hat* package is required. It should be installed by default on your Raspberry Pi OS, but it's available in apt and Add/Remove software tools if needed.

Display Text

Let's start with something interesting, it's possible to use the LED matrix to display text.

Hello world

Create a first script to test this, and past the following code:

```
from sense_hat import SenseHat  
  
sense = SenseHat()  
sense.show_message("Hello world")
```

helloworld.py

As with have seen in the previous chapter:

- We need to import the SenseHat class from the sense_hat library.
- Then we create an object named "sense" that will be used to call any method from this class.
- And the show_message function will display the text on the LED display.

Advanced features

The `show_message` function has other optional parameters, the complete syntax is:
`show_message(<text>, <speed>, <color>, <background>)`

- Speed: default is 0.1, increase or decrease this value to change the scrolling speed.
- Color: represented as a list containing the RGB color of your text. Ex: (255,0,0) is red.
- Background: same format for the background color.

Example:

```
sense.show_message("Hello world", 0.2, (0,0,255), (255,255,255))
```

This will display "Hello world" in blue over a white background, two times slower than the first example.

Note: *the background stay on at the end, you can use this function to reset the LED display:*
`sense.clear()`.

Set Pixels

It's also possible to light up some specific pixels to the desired color.

Change one pixel

The first function you can use is:

```
set_pixel(x, y, color)
set_pixel(x, y, r, g, b)
```

Yes, there are two ways to use it. You can either pass the color in one parameter, as a list, or each RGB value separately.

Here is a complete example:

```
from sense_hat import SenseHat

sense = SenseHat()

red = (255,0,0)
sense.set_pixel(2,6,red)
```

setpixel.py

Change all pixels at once

Another method is to set the value of each pixel in one giant list of values. It might seem more complicated, but it can be pretty useful in some cases.

The function is `set_pixels`, with an "s" and takes only one parameter: a list containing 64 lists of RGB values.

Here is an example to make this more clear:

```
from sense_hat import SenseHat

sense = SenseHat()

X = [255, 0, 0] # Red
O = [255, 255, 255] # White

question_mark = [
    O, O, O, X, X, O, O, O,
    O, O, X, O, O, X, O, O,
    O, O, O, O, O, X, O, O,
    O, O, O, O, X, O, O, O,
    O, O, O, X, O, O, O, O,
    O, O, O, X, O, O, O, O,
    O, O, O, O, O, O, O, O,
    O, O, O, X, O, O, O, O
]

sense.set_pixels(question_mark)
```

setpixels.py

By the way, this is an example from the official documentation.

So, we define your two lists for the main colors (red and white in this case).

And then a list with 64 elements (8 lines of 8 pixels), defining which color to use for each pixel.

You can use this to test your pixel art skills, but also to interact with the user. In some cases, it will be way faster to do this instead of many `set_pixel()` lines.

Get Data from the Sensors

Collecting data from the Sense Hat sensors is straightforward, but I have to give you the functions you can use to do this:

```
from sense_hat import SenseHat

sense = SenseHat()

#temperature
temp = sense.get_temperature()
print("Temperature: " + str(temp) + " °C")

#humidity
humidity = sense.get_humidity()
print("Humidity: " + str(humidity) + " %")

#pressure
pressure = sense.get_pressure()
print("Pressure: " + str(pressure) + " Millibars")
```

collectdata.py

Names are self-explanatory and there are no parameters, so it's straightforward.

They return a float value (with too many decimals places!), so you need to use the str function to concatenate them in a print function. You can also use the round() function to limit the decimal length.

Other functions are available, you can check the documentation to find them. Also, the magnetometer needs calibration before being used, you can find some guidance on the official website: <https://www.raspberrypi.org/documentation/hardware/sense-hat/>.

Joystick Events

The last thing I want to show you before the exercise is the joystick. We can use it to detect movement and for example do something depending on the user interaction (and that will be the goal in the action steps!).

The joystick will throw an event when the user interacts with it. An event has several parameters: a timestamp, action (pressed, released, held) and direction (up, down, left, right, middle).

There are two functions you can use to get this event in Python:

- `wait_for_event()`
As the name suggests, the script is paused until something happens. The function returned three values: timestamp, action and direction.
- `get_events()`
Rather than waiting for new events, you can check if something has recently happened. This function gives you the history of all events that have happened since the previous call.

Here is a basic script to show you how it works:

```
from sense_hat import SenseHat
from time import sleep

sense = SenseHat()

event = sense.stick.wait_for_event()
print("The joystick was {} {}".format(event.action, event.direction))

sleep(0.1)
event = sense.stick.wait_for_event()
print("The joystick was {} {}".format(event.action, event.direction))
```

joystick.py

When you run the script, it won't do anything until you use the joystick.

The first message will show a "pressed" action with the direction, and the second will correspond to the "released" event.

Obviously, the idea here is to do something depending on these events, that is what we'll try to do in the exercise below.

ACTION STEPS

You now have a better idea about the Sense HAT features, and how you can use them altogether for interesting projects. You can find many examples online (you can even create games with it!), but we'll do an exercise here to put this in practice.

The goal of these action steps is to interact with a LED pixel by using the joystick. Clear the LED matrix and set one pixel in the middle of it. Now, we want to move the pixel depending on the joystick events. If you press the joystick to the left, the pixel should move to the left too.

If you have an idea on how to do this, let's go! It's always better if you manage to break down any project into smaller parts yourself. There are many ways to do the same thing, and I prefer you to do it as you want.

Precision: when you define variables outside a function and want to update their values in a function, you need to use the "global" keyword to tell Python you want this.

Example:

```
def change_name():  
    global firstname  
    firstname = "Robert"  
  
    firstname = "Patrick"  
    print(firstname)  
    change_name()  
    print(firstname)
```

The first print will show "Patrick" and the second "Robert". If you remove the line starting with "global", both print functions will display "Patrick".

A bit lost on how to complete this project without help? Don't worry, here are my suggestions to complete this exercise step by step:

- Import the libraries and create the sense object as in the previous examples.
- Define two variables representing the position of your pixel (x and y for example).
- Create an infinite loop that waits for new joystick events, and display the event parameters (for debug purposes).
- Now create a function that you'll call instead of the debug display. The goal of this function will be to move the pixel depending on the action and direction.
- In this function, create conditions to change the x and y values depending on the action and direction.
Ex: if the joystick is pressed to the left, then decrease the x value.
- Once x and y are updated correctly, you can set the pixel on the LED matrix with the new values. Don't forget to turn off the previous pixel before turning on the new one, we don't want to create a snake :).

And as always, you can find my solution in the samples folder and at the end of this book. But if your method works it's great, mine is not necessarily better, just take a few minutes to compare them. If yours didn't work, try to understand why by reading my solution.

If it was easy for you, you can go further and create games with it. Remember pacman? Why not set random pixels in another color that add points to the player when it collects them? Or bad pixels that end the script if they are touched?

You can also find other fun examples on Trinket.io: <https://trinket.io/sense-hat>.

PYTHON LIBRARIES

INTRODUCTION

After reading this book, you should have a good understanding of the principles behind any Python project. When you find an idea for a project, you know to import the required libraries and use your skills to achieve your goals. It's probably less complicated than you initially thought.

Now that you've learned the fundamentals of Python and experimented with several projects by following my advice in this book, it's time for the next step: building on your own code.

- Step 1: Understand the logic behind any programming language.
- Step 2: Learn how to code this logic in a specific language (ex: Python).
- Step 3: Ability to read and understand almost any source code written by someone else.
- Step 4: Create your code to achieve a defined goal.

This annex will help you with step 4. Coding your scripts doesn't mean to not look for examples at all, but it's your choice and your style that will be implemented. I will give you 10 interesting libraries you can use in various projects. They work well on Raspberry Pi and are easy to use because they are properly documented.

1 - GPIO

Name	GPIO
Goal	Interact with the GPIO pins on Raspberry Pi
Package	sudo apt install rpi.gpio
Documentation	https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/
Basic example	<pre>import RPi.GPIO as GPIO GPIO.setmode(GPIO.BCM) GPIO.setwarnings(False) led = 4 #Turn on the LED print "LED on" GPIO.output(led,1) #Wait 5s time.sleep(5) #Turn off the LED print "LED off" GPIO.output(led,0)</pre>
Links	https://raspberrytips.com/raspberry-pi-gpio-pins/
Alternative	gpiozero https://projects.raspberrypi.org/en/projects/physical-computing

2 - GUIZERO

Name	Guizero
Goal	Create simple GUI (User interfaces) with Python
Package	sudo apt install python3-guizero
Documentation	https://lawsie.github.io/guizero/
Basic example	<pre>from guizero import App app = App(title="Hello world") app.display()</pre>
Links	https://projects.raspberrypi.org/en/projects/getting-started-with-guis

3 - TWITTER

Name	Twython
Goal	Use the Twitter API in Python
Package	sudo pip3 install twython
Documentation	https://twython.readthedocs.io/en/latest/
Basic example	<pre> from twython import Twython from auth import (consumer_key, consumer_secret, access_token, access_token_secret) twitter = Twython(consumer_key, consumer_secret, access_token, access_token_secret) message = "Hello World!" twitter.update_status(status=message) print("Tweeted: " + message) </pre>
Links	https://projects.raspberrypi.org/en/projects/getting-started-with-the-twitter-api
Alternative	python-twitter, tweepy

4 - PYGAME

Name	Pygame
Goal	Games development
Package	<code>sudo apt install python3-pygame</code>
Documentation	https://www.pygame.org/docs/

5 - FLASK

Name	Flash
Goal	Create a website with Python
Package	<code>sudo pip3 install flask</code>
Documentation	https://flask.palletsprojects.com/en/2.0.x/
Basic example	<pre>from flask import Flask app = Flask(__name__) @app.route('/') def index(): return 'Hello world'</pre>
Links	https://projects.raspberrypi.org/en/projects/python-web-server-with-flask

6 - MySQL

Name	MySQL Connector
Goal	Store information in a database
Package	<code>sudo pip3 install mysql-connector-python</code>
Documentation	https://dev.mysql.com/doc/connector-python/en/
Basic example	<pre>import mysql.connector cnx = mysql.connector.connect(user='scott', password='password', host='127.0.0.1', database='employees') cnx.close()</pre>
Alternatives	MiniDB, SQLite3, etc.

7 - OPENCV

Name	OpenCV
Goal	Computer vision library, image processing and machine learning
Package	<code>sudo apt install python3-opencv</code>
Documentation	https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html

8 - REQUESTS

Name	Requests
Goal	Make HTTP requests in Python
Package	sudo apt install python3-requests
Documentation	https://dev.mysql.com/doc/connector-python/en/
Basic example	<pre>import requests response = requests.get('https://raspberrytips.com')</pre>

9 - MATPLOTLIB

Name	Matplotlib
Goal	Create data visualizations
Package	sudo apt install python3-matplotlib
Documentation	https://matplotlib.org/
Basic example	<pre>import matplotlib.pyplot as plt plt.plot([57,42,24,6]) plt.ylabel('random numbers') plt.show()</pre>

10 - PILLOW

Name	Pillow
Goal	Image manipulation
Package	sudo pip3 install pillow
Documentation	https://pillow.readthedocs.io/en/stable/
Basic example	<pre>from PIL import Image img = Image.new('RGB', (100, 50), color = 'red') img.save('red_square.png')</pre>

CONCLUSION

These libraries are a small example of what you can do with Python, just to give you a few ideas. But there are libraries for everything, so feel free to use your friend Google to look for alternatives or specific modules for your projects.

Once you have found a library and the documentation page, you should be able to use any of them with your new Python skills!

CONCLUSION

So, how do you feel? Do you think you have improved your Python knowledge? I promised to teach you only the essentials, in a step-by-step process to create, understand and improve any Python script on Raspberry Pi.

You might still need a bit of practice to feel more comfortable with your new programming skills, but I'm sure you have learned a lot by reading and working through this book.

Also, I would love to have your feedback about this book. I like to know what you expected at the beginning, how the learning curve was, and how you feel after completing everything. If you have a few seconds for me, you can follow this link to share your review:

<https://raspberrytips.com/master-python-review>.

I have many tutorials about Python on RaspberryTips.com that you can check for details or ideas for projects. The advantage of the site is that it's possible to exchange thoughts in the comment section, if you have corrections to add or questions to ask.

You are welcome to contact me for any suggestions you have or any mistakes you have found in this book. My role is to help you by giving you the answer if I can, or by pointing you to people who can help you better than I.

See you soon on RaspberryTips,

Patrick

ACTION STEPS SOLUTIONS

Note: All the corresponding source codes are available in the "solutions" folder.

CHAPTER 3

Question 1

```
goodmorning1.py ✕  
1 print("Good morning Patrick")
```

Question 2

```
goodmorning2.py ✕  
1 name="Bob"  
2 print("Good morning "+name)
```

Question 2

```
goodmorning3.py ✕  
1 name="Bob"  
2 time=14  
3 if time<12:  
4     print("Good morning "+name)  
5 elif time<18:  
6     print("Good afternoon "+name)  
7 else:  
8     print("Good evening "+name)  
9  
Shell  
python goodmorning3.py  
Good afternoon Bob  
>>>
```

CHAPTER 4

```
from time import sleep

def display_ingredient(ingredient):
    print("- "+ingredient)

recipe=['4 cups raspberries','1 cup sugar','1/3 cup flour','1 egg']
for item in recipe:
    display_ingredient(item)
    sleep(1)
```

CHAPTER 5

```
import time
import picamera

def take_picture(camera, prefix):
    ts=str(round(time.time())) #you can call several functions on one line
    camera.start_preview()
    camera.capture('/home/pi/'+prefix+'-'+ts+'.jpg')
    camera.stop_preview()

camera = picamera.PiCamera()
for image in range(30):
    take_picture(camera, "timestamp")
    time.sleep(2)
```

CHAPTER 6



```
from mcpi.minecraft import Minecraft

mc = Minecraft.create()
x,y,z = mc.player.getPos()

mc.setBlocks(x-2,y-1,z-2,x+2,y-1,z+2,5) #floor
mc.setBlocks(x-2,y,z-2,x-2,y+2,z+2,5) #wall1
mc.setBlocks(x-2,y,z-2,x+2,y+2,z-2,5) #wall2
mc.setBlocks(x+2,y,z-2,x+2,y+2,z+2,5) #wall3
mc.setBlocks(x-2,y,z+2,x+2,y+2,z+2,5) #wall4

mc.setBlocks(x-2,y+3,z-2,x+2,y+3,z+2,17) #roof

mc.setBlock(x-2,y+1,z,20) #window 1
mc.setBlock(x,y+1,z-2,20) #window 2
mc.setBlock(x+2,y+1,z,20) #window 3

mc.setBlock(x,y+1,z+2,0) #door 1
mc.setBlock(x,y,z+2,0) #door 2
```

CHAPTER 7

Step 1 - API, sunrise and sunset

```
import urllib.request, json, time
from datetime import datetime

#Location
lat=36.7201600
lon=-4.4203400

#API URL
url="https://api.sunrise-sunset.org/json?lat="+str(lat)+"&lng="+str(lon)
response = urllib.request.urlopen(url)
data = json.loads(response.read())
sunset=data['results']['sunset']
sunrise=data['results']['sunrise']

#Debug
print("Sunrise: "+sunrise)
print("Sunset: "+sunset)

#Current time in the same format
FMT = '%I:%M:%S %p'
current = time.strftime(FMT)
print("Current time: "+current)

#Differences with sunset and sunrise
diff_sunset = datetime.strptime(sunset, FMT)-datetime.strptime(current, FMT)
diff_sunset = diff_sunset.seconds
```

```
diff_sunrise = datetime.strptime(sunrise, FMT)-datetime.strptime(current, FMT)
diff_sunrise = diff_sunrise.seconds
```

```
#Debug
print(diff_sunset)
print(diff_sunrise)
```

Step 2 - Adding the lights management

```
import urllib.request, json, time
from datetime import datetime
from phue import Bridge

#Connect to bridge
b = Bridge('192.168.222.2')

#Define the lights to manage
living_room = 1

#Location
lat=36.7201600
lon=-4.4203400

#URL API
url="https://api.sunrise-sunset.org/json?lat="+str(lat)+"&lng="+str(lon)
response = urllib.request.urlopen(url)
data = json.loads(response.read())
sunset=data['results']['sunset']
sunrise=data['results']['sunrise']
```

```
#Current time in the same format
FMT = '%I:%M:%S %p'
current = time.strftime(FMT)

#Differences with the sunset and sunrise
diff_sunset = datetime.strptime(sunset, FMT)-datetime.strptime(current, FMT)
diff_sunset = diff_sunset.seconds

diff_sunrise = datetime.strptime(sunrise, FMT)-datetime.strptime(current, FMT)
diff_sunrise = diff_sunrise.seconds

#The light turn on following the scheduled cron in the morning (4:30 a.m.)
#Nothing to do when sunrise is closer than sunset
if diff_sunset < diff_sunrise:
    #Turn on 2 hours before sunset
    if diff_sunset < 3600*2:
        print("Sunset is coming, lights on")
        b.set_light(living_room, 'on', True)
    #Turn off about 2 hours after sunrise
    elif diff_sunrise < 3600*22:
        print("Sun is here, no need for lights")
        b.set_light(living_room, 'on', False)
else:
    print("Nothing to do")
```

CHAPTER 8

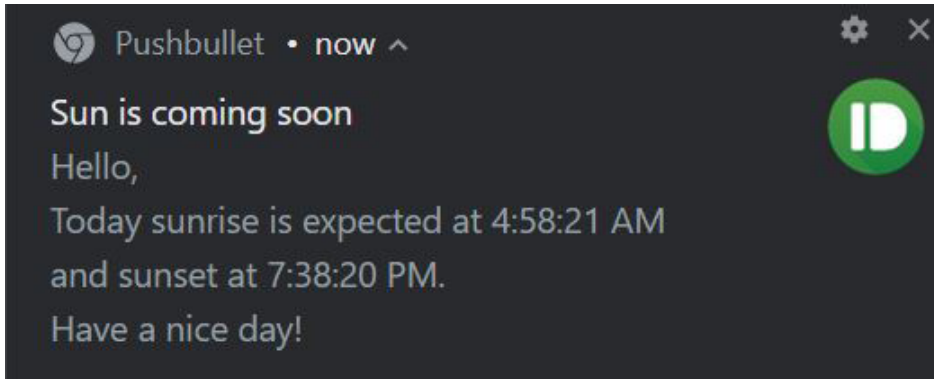
```
from pushbullet import PushBullet
import urllib.request, json, time
from datetime import datetime

lat=36.7201600
lon=-4.4203400
token="YOUR_TOKEN"

#GET SUNSET AND SUNRISE
url="https://api.sunrise-sunset.org/json?lat="+str(lat)+"&lng="+str(lon)
response = urllib.request.urlopen(url)
data = json.loads(response.read())
sunset=data['results']['sunset']
sunrise=data['results']['sunrise']

message=f"""Hello,
Today sunrise is expected at {sunrise}
and sunset at {sunset}.
Have a nice day!"""

#SEND IT IN A PUSH NOTIFICATION
pb = PushBullet(token)
pb.push_note("Sun is coming soon",message)
```



To send the message each day, add a new line in your crontab. For example:

```
0 0 * * * /usr/bin/python3 /home/pi/sunpush.py
```

CHAPTER 9

```
from sense_hat import SenseHat
from time import sleep

def move_pixel(action, direction):
    global x, y
    if action=="pressed":
        print("Direction: "+direction)
        print("X: "+str(x))
        print("Y: "+str(y))

        if direction=="up" and y>0:
            y=y-1
        elif direction=="down" and y<8:
```



```
        y=y+1
    elif direction=="left" and x>0:
        x=x-1
    elif direction=="right" and x<8:
        x=x+1
    else:
        print("Error: Can't go in this direction")

    sense.clear()
    sense.set_pixel(x, y, 255, 0, 0)

sense = SenseHat()
x = 4
y = 4

sense.clear()
sense.set_pixel(x, y, 255, 0, 0)

while True:
    event = sense.stick.wait_for_event()
    move_pixel(event.action, event.direction)
    sleep(0.5)
```

BONUS – THE DISCORD BOT

INTRODUCTION

Discord is a free text, voice and video messaging app. It's similar to Slack if you want, but is mainly oriented for gamers. As we can find gamers at school or in business, there are now more and more companies and students that move to discord to use it as a Slack alternative, as almost all features are available for free.

Anyway, the goal here is not to sell you discord, but to show you interesting features.

It's possible to create bot users on Discord, in order to add more features in your discussion channels. For example, bot users can be used for management commands, music playing and connecting to external devices (like a Raspberry Pi).

You can check out this website to see a few examples of Discord bots created by the community: <https://top.gg/>.

In this chapter, we'll learn how to create a bot on Discord, and how to program it with a Python script running on your Raspberry Pi.

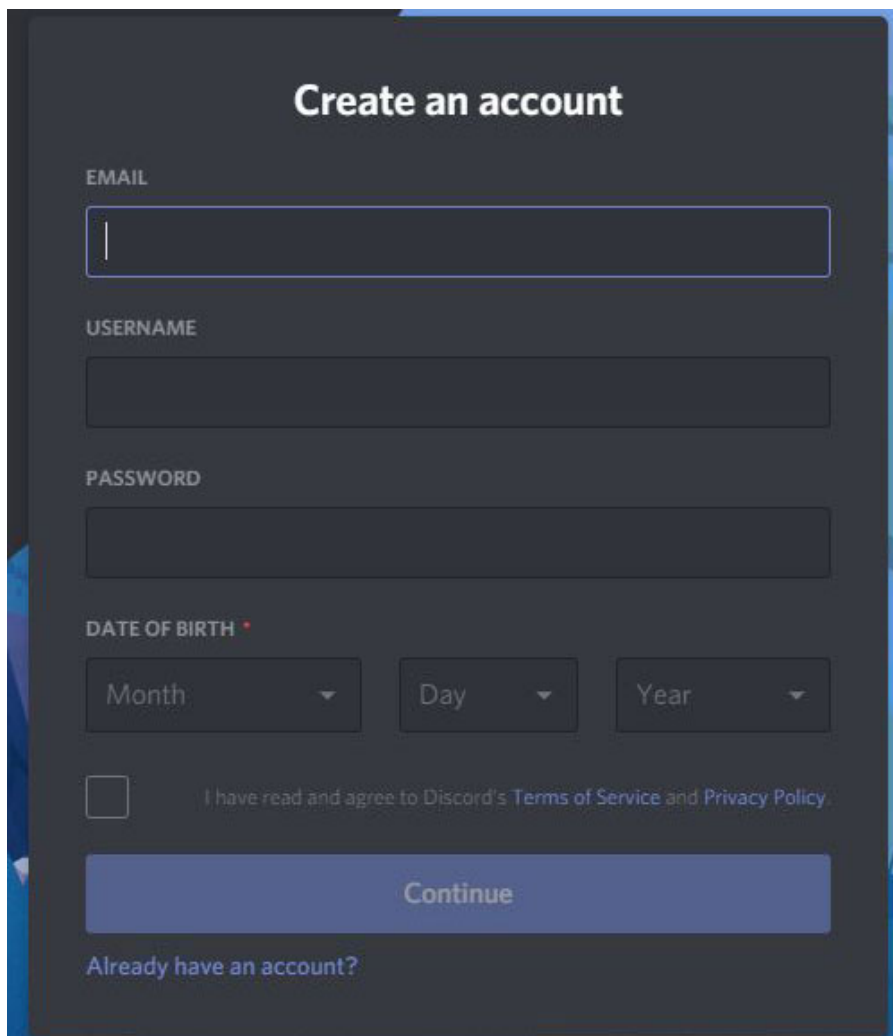
CREATE YOUR BOT ON DISCORD

Ok, let's jump on Discord now for all the prerequisites required to create a bot. If you already know Discord or even have your server, you can probably skip a few sections here.

Register an account

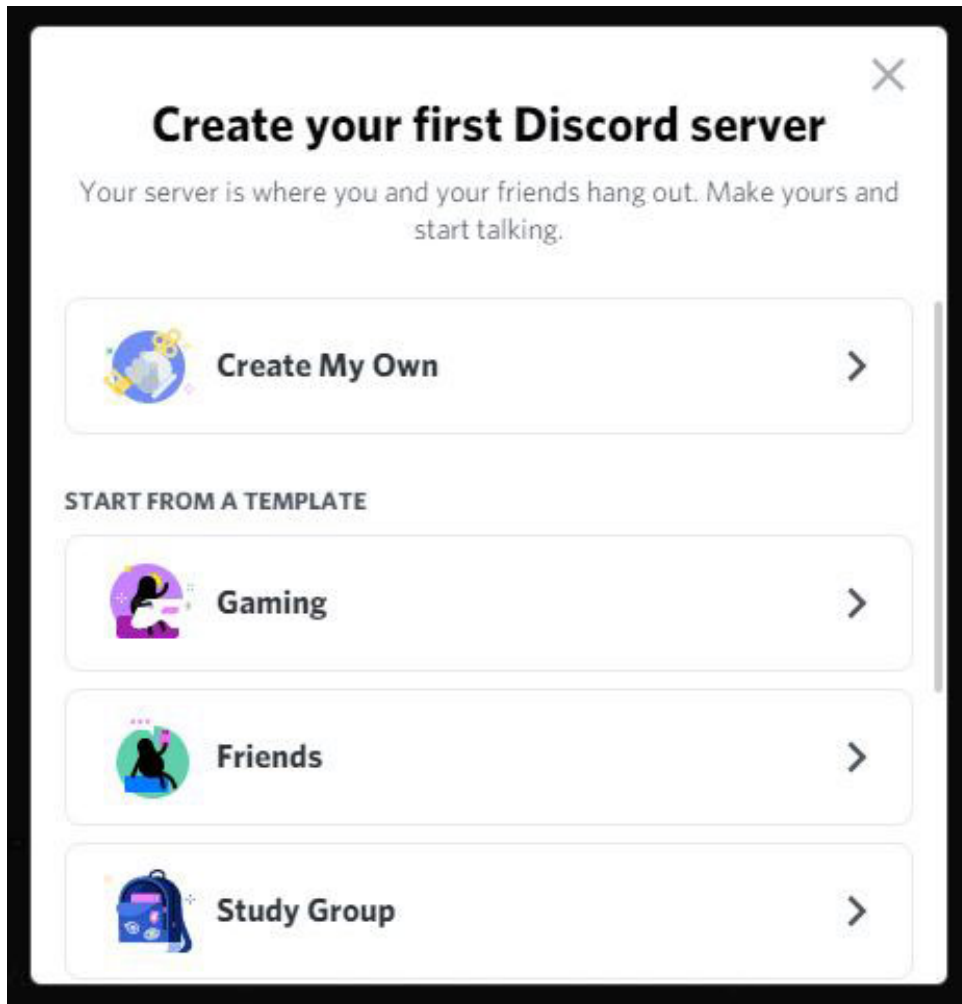
If you don't have a Discord account yet, start by doing this:

- Go to [Discord.com](https://discord.com) and click on Login, then Register.
- Fill the following form on the website



The image shows a screenshot of the Discord website's registration page. The title 'Create an account' is centered at the top. Below it are several input fields: 'EMAIL', 'USERNAME', and 'PASSWORD'. The 'DATE OF BIRTH' field is split into three dropdown menus for 'Month', 'Day', and 'Year'. At the bottom of the form is a checkbox for 'I have read and agree to Discord's Terms of Service and Privacy Policy.' and a large blue 'Continue' button. Below the button is a link that says 'Already have an account?'.

- Once logged, directly create a new server:



The first one is perfect for this test.

- Then, choose a name and an icon:

Community Guidelines.' At the bottom left is a 'Back' button, and at the bottom right is a green 'Create' button." data-bbox="104 216 712 635"/>

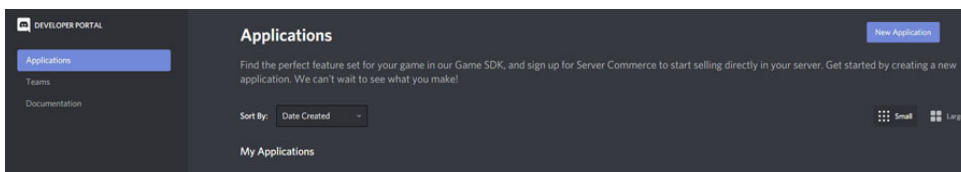
That's it, you have your account and server ready and you can move to the next step.

Don't forget to confirm your email address, as it's mandatory to create a bot.

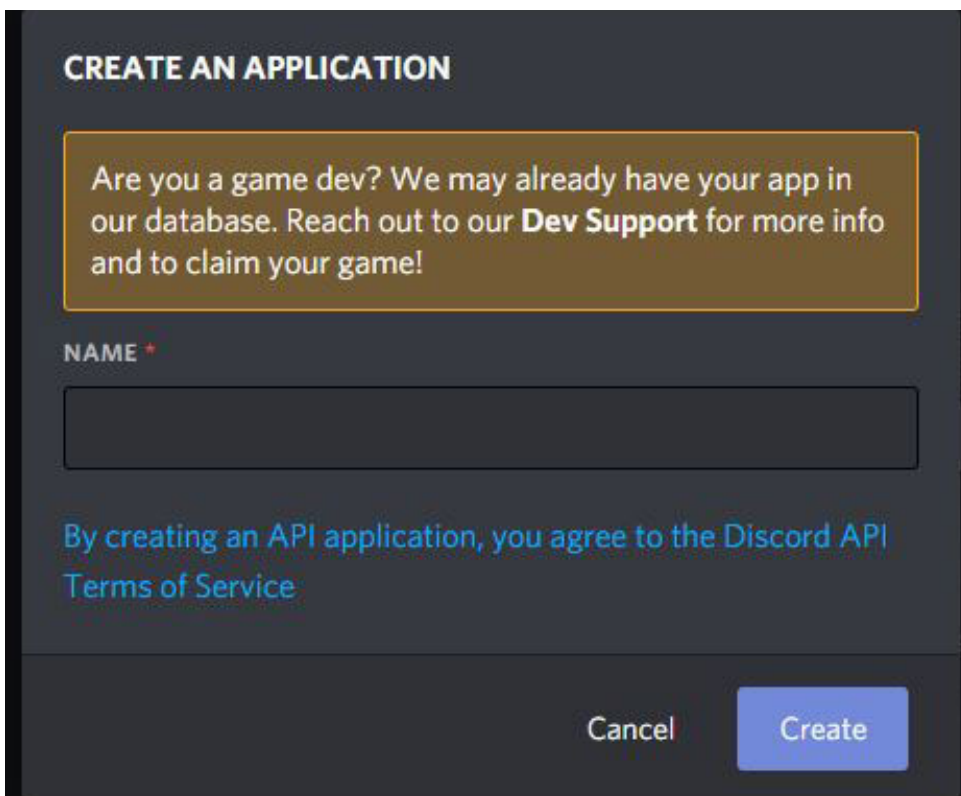
Create an application

We need to declare a new application before creating a bot:

- Start by opening the Discord developer console by clicking [here](#). It looks like this:



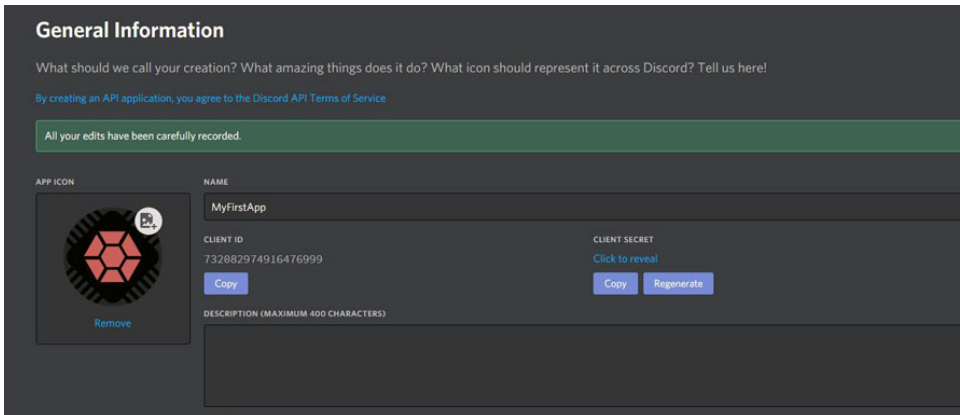
- Create a new application by clicking on the top right button.
- Choose



As I noticed later, your bot will receive the application name. So, "test" or "myfirstapp" is probably not a good idea.

- To create the app, you absolutely need to have your email confirmed

That's it, the application is created, we can now create a new bot:



The screenshot shows the 'General Information' tab of a Discord application. At the top, it asks for a name and icon, with a note that creating an API application agrees to the Discord API Terms of Service. A green status bar indicates 'All your edits have been carefully recorded.' Below this, the 'APP ICON' section shows a Raspberry Pi icon with a 'Remove' button. The 'NAME' section shows 'MyFirstApp'. The 'CLIENT ID' is '732882974916476999' with a 'Copy' button. The 'CLIENT SECRET' is partially visible as 'Click to reveal' with 'Copy' and 'Regenerate' buttons. A 'DESCRIPTION (MAXIMUM 400 CHARACTERS)' field is at the bottom.

Create a bot

- Click on "Bot" in the left menu.
- Then click on "Add bot" and accept the confirmation message.
- A bot that is created is automatically linked with your application.

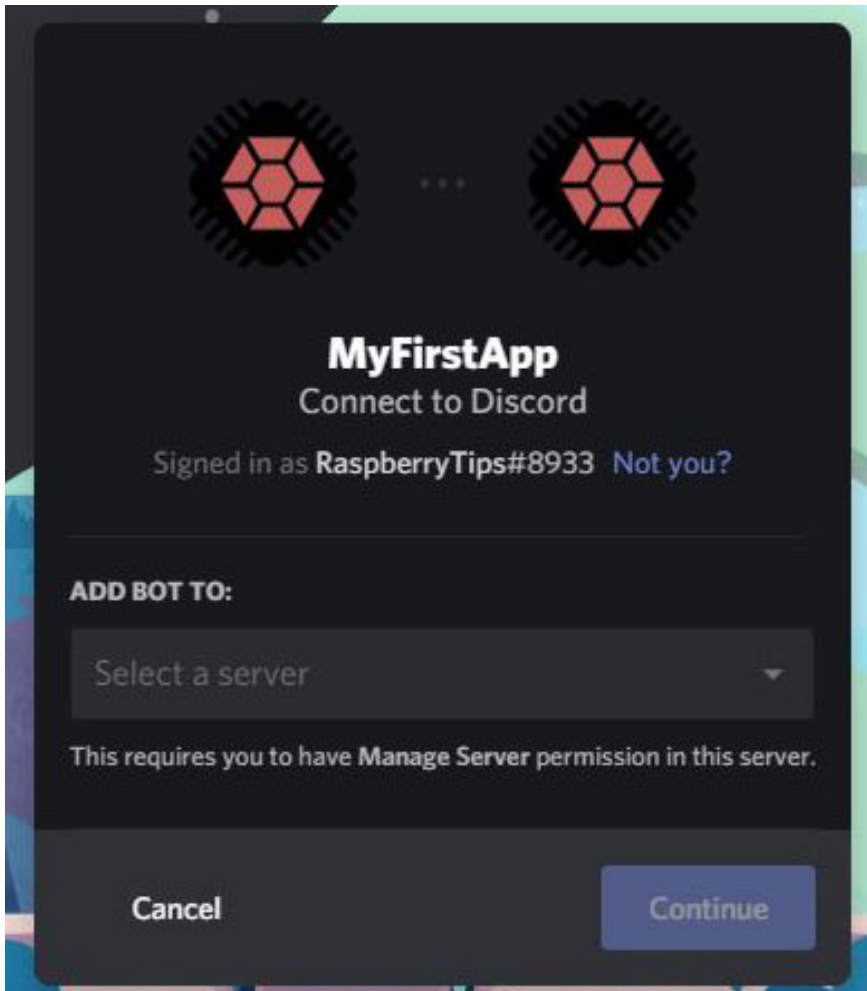
Invite the bot on your server

The last step is to invite the bot to join your server:

- The first thing is to go to this URL:
https://discord.com/api/oauth2/authorize?client_id=<CLIENTID>&scope=bot&permissions=1

Replace <CLIENTID> with your client ID, you can find it on the application page (check my previous image for an example).

- On the page that opens, choose the server to join:



- Click on Continue.
- Then click on Authorize on the next screen.
- The bot is now connected on the server. You can check on your server to be sure, you should have something like:



The part on Discord is now completed, we have everything we need to start to work on the Raspberry Pi.

CODE YOUR BOT WITH PYTHON

Prerequisites

With new projects, we usually need to install a library to interact with Discord in your Python scripts.

Use the following command to install it on your Raspberry Pi:

```
sudo pip3 install discord.py
```

It will probably install a bunch of other libraries that are required for it to work correctly. Once done, you are ready to create your first script.

You can find the library page here: <https://pypi.org/project/discord.py/>.

And they have a pretty good documentation available here too:
<https://discordpy.readthedocs.io/>.

First script

Goal

If you have never used a Bot before, I need to explain a few things first.

In general, bots don't react to any message typed on Discord. There will be a discussion among users, and they will be triggered only for specific messages. Often these messages start with a special character like an exclamation point for example.

The idea, in this first example, is to answer "pong" when someone sends the command "ping".

We'll use the ">" as a prefix for our bot commands.

Script code

Here is the code to do this, and I'll explain after you test it:

```
import discord
from discord.ext import commands

bot = commands.Bot(command_prefix=">")

@bot.command()
async def ping(ctx):
    await ctx.send("pong")

bot.run("YOUR_TOKEN")
```

discord-bot.py

So, a few explanations here:

- As usual, we import the libraries before using them. It's composed of several modules, so we need to import discord but also the commands class.
- Then we create an object named "bot" created from the bot class. The only parameter we set is the command prefix we'll use on discord (">" for this example).
- I'll quickly pass over the next line (starting with an at sign), it's what we call a decorator in Python. Just remember to put all your bot commands under this line. If you want to understand this concept, you can read this page in the Python wiki: <https://wiki.python.org/moin/PythonDecorators>.
- Then we define the function that will answer the bot command. It has to be named the same as the command you will use (so ping in this case, as we'll use ">ping" on Discord).

- This is an asynchronous function, meaning that the bot won't wait for the answer. In this case, it's not essential. But if your function takes time, the bot needs to handle the other commands first. It will get the answer once available, it doesn't matter when.
- The "send" command is used to send the message to discord.

Going further

Try it and see how it works, try to play with it a bit and add your own ideas here.

I invite you to read the examples from the GitHub repository to get a better idea on what is built-in in the library. Commands and basic answers are not the only thing you can do with it.

Here is the link with the examples:

<https://github.com/Rapptz/discord.py/tree/master/examples>.