

STEMKIT

4SCHOOLS

STEMKIT CURRICULUM DESIGN & DEVELOPMENT

Output Identification: O2A1



Co-funded by the
Erasmus+ Programme
of the European Union

This project has been funded with support from the European Commission.
This communication reflects the views only of the author, and the Commission cannot be held responsible for
any use which may be made of the information contained therein.

Table of Contents

1	General introduction about the project.....	3
2	Goals of this curriculum.....	3
3	Introduction to Scratch 2.0 [P6-ARC].....	4
3.1	Glossary of terms.....	4
3.2	Main content	4
3.2.1	Introduction to Scratch 2.0 basic functions	5
3.2.2	Practical Example of Scratch projects	9
3.2.3	Conclusions	11
3.3	Assessment test.....	11
3.4	References.....	12
4	Scratch GPIO (Control GPIO pins/receive inputs) [P3-DANMAR]	13
4.1	Glossary of terms	13
4.2	Main content	13
4.2.1	Basic information about Scratch GPIO	13
4.2.2	GPIO naming convention	15
4.2.3	GPIO interaction	15
4.3	Practical examples.....	16
4.3.1	Blinking LED – example 1	16
4.3.2	Controlling LED with a tactile switch – example 2	18
4.3.3	Expanding the use of GPIO – example 3	20
4.4	Assessment test.....	22
4.5	References.....	23
4.6	Additional resources.....	23
5	Introduction to Raspberry Pi Edition of Minecraft [P4-HESO / P5-SCHOLE]	24
5.1	Glossary of terms	24
5.2	Main content	24
5.2.1	Introduction to Minecraft Pi basic functions	24
5.2.2	Minecraft Pi elements and gameplay	26
5.2.3	Controlling Minecraft Pi with Python.....	27
5.2.4	Interaction with the physical world through the GPIO	33
5.2.5	Importing new maps and resource packages	38
5.3	Practical examples.....	39
5.3.1	Example 1: Trap player between blocks!.....	39
5.3.2	Example 2: Build a house!.....	40
5.3.3	Example 3: Build a house with a twist!	41
5.3.4	Example 4: Big Block of blocks	42
5.3.5	Example 5: Create a Volcano! (advanced)	42
5.4	Assessment test.....	43
5.5	References.....	44
5.6	Additional resources.....	44
5.7	Conclusion	46
6	Raspberry Pi GPIO programming using Python [P2-AKNOW].....	47



6.1	Glossary of terms	47
6.2	Main content	47
6.2.1	Introduction to Raspberry Pi GPIO pins.....	48
6.2.2	Introduction to Python programming language	51
6.2.3	Introduction to electronic circuits	52
6.3	Practical examples	66
6.3.1	Example 1: Use a Buzzer and Multiple LEDs with Raspberry Pi GPIO Pins 67	
6.3.2	Example 2: Measure the Speed of Sound	67
6.3.3	Example 3: Make a Proximity Alarm with Light and Sound.....	67
6.4	Assessment test.....	69
6.5	References.....	70
6.6	Additional resources.....	70
6.7	Conclusion	71
7	Physical Computing [P1-ECAM].....	72
7.1	Glossary of terms	72
7.2	Main content	72
7.2.1	Introduction to Physical Computing	72
7.2.2	Key Competences in Physical Computing	74
7.3	Practical examples	75
7.3.1	Example 1	75
7.3.2	Example 2.....	79
7.4	Assessment test.....	82
7.5	References.....	83
7.6	Additional resources.....	84
7.7	Conclusion	84
8	General conclusion	85

1 General introduction about the project

Children today are born into technology and using it comes natural to them. However, there is a need for them to acquire technological skills, such as programming. STEM skilled labour force is in high demand in Europe and demand will continue to increase due to the advent of Industry 4.0 and Advanced Manufacturing Technologies. New ways of engaging children into programming and STEM are needed but more screen time is not the best approach. Hands-on play is more fun and many times more educational.

The bridging of the online and the offline worlds may offer a more engaging and healthier environment for children to learn how to program and develop STEM skills.

The STEMKIT4Schools Erasmus+ project has then as primary objective to produce approaches and tools to help those working with children reach out to them with a view to help them engage with programming and develop STEM related skills. It aims to achieve this not by increasing screen time but by encouraging hands on play through the creation of games playable on a retro design DIY wooden computer in combination with electronic gadgets related to STEM subjects.

2 Goals of this curriculum

A complete curriculum has been designed by the partners of the project to answer the goals of STEMKIT4Schools, including lessons plans for using Minecraft Pi/Scratch/Python/Kits with the STEMKIT computer in the classroom. The lesson plans are part of the Educator's guide for teachers. The electronics kits are designed and built to complement the teaching of programming, physical computing and STEM subjects.

The final STEMKIT course teaches basic elements of games with Minecraft Pi, Python, Scratch, as well as physical computing (using basic digital, analogue, and electromechanical components) and collaboration (engage and share with others).

The STEMKIT DIY computer is based on Raspberry Pi, meaning it can harness the power and practically unlimited resources for Raspberry while the features of the Raspbian operating system provide additional capabilities.

The re-purposed STEMKIT curriculum using instructional design principles can be found inside the Learning Portal in the form of interactive Learning Objects and will be delivered to the learners who will be able to follow the interactive course at their own time and pace while they can utilize the social learning tools offered to engage with their peers (engage with comments, forums, chat). Additional collaboration features such as notifications, group feeds, blogs, file sharing, questions/answers, voting will be also provided.

In the next sections, the following 5 items are presented: Introduction to Scratch 2.0 [P6-ARC], Scratch GPIO (Control GPIO pins/receive inputs) [P3-DANMAR], Introduction to Raspberry Pi Edition of Minecraft [P4-HESO / P5-SCHOLE], Raspberry Pi GPIO programming using Python [P2-AKNOW] and Physical Computing [P1-ECAM].



3 Introduction to Scratch 2.0 [P6-ARC]

3.1 Glossary of terms

Term / Concept	Definition / Explanation
Scratch 2.0	<p>Scratch is a programming language, created by MIT Media Lab, an open-source development environment that makes it easy to create interactive art, stories, simulations, and games. It is aimed at educating people with little or no programming experience, primarily children between the ages of 8 and 16.</p> <p>Scratch 2.0, also known as Scratch 2,[1] was the second major version of Scratch, following Scratch 1.4. It featured a redesigned editor and website, and it was the first version that included the online editor as well as an offline one.</p> <p>Scratch was completely rewritten in Adobe Flash for version 2.0 but still ran projects from older versions of Scratch. It was still completely free and without ads.</p>
Blocks (2.0)	<p>Blocks were puzzle-piece shapes that were used to create code in Scratch. The blocks connected to each other vertically similar to a jigsaw puzzle, where each data type (hat, stack, reporter, boolean, or cap) had its own shape and a specially shaped slot for it to be inserted into, which prevented syntax errors. Series of connected blocks were called scripts.</p>
Editing the Block Colors	<p>In the online Scratch 2.0 editor, by shift-clicking the Edit menu an option called "Edit block colors" appeared. By selecting this, a menu would appear with 3 HSL sliders and tools for modifying the block colors of a specific block category.</p>

3.2 Main content

Scratch is a block-based **visual programming language** and website targeted primarily at children, that allows users to learn computer programming while working on personally meaningful projects such as animated stories and games. (Maloney, 2010)

Scratch is used as by schools in multiple disciplines (math, computer science, language arts, social studies).

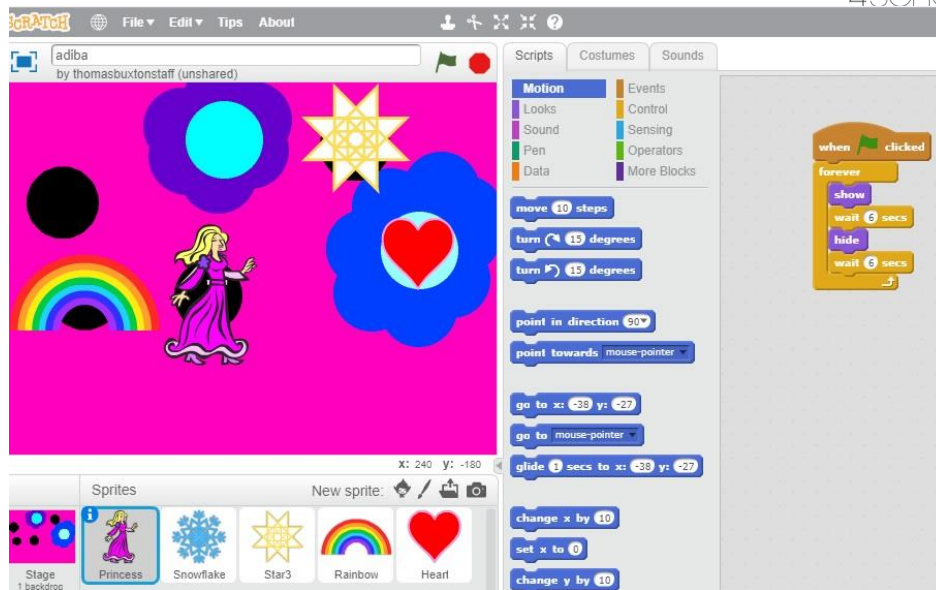


Fig.1 Screenshot from Scratch project (Source <https://www.thomasbuxton.towerhamlets.sch.uk/blogs/year3/2017/11/17/year-3-computing-scratch-projects/>)

The original design of Scratch was motivated by the needs and interests of young people (ages 8 to 16) at after-school computer centers such as the Intel Computer Clubhouses (Resnick, 2003). Initially, Scratch was used primarily in informal learning settings such as community centers, after-school clubs, libraries, and homes; it is used in schools as well. (<http://web.media.mit.edu/~jmaloney/papers/ScratchLangAndEnvironment.pdf>)

3.2.1 Introduction to Scratch 2.0 basic functions

The Scratch project began in 2003, and the Scratch software were publicly launched in 2007. Scratch is free, available in nearly 50 languages, the software is often redistributed by school systems and educational organizations.

- ✓ An important goal of Scratch is to introduce programming to those with no previous programming experience.
- ✓ Scratch targets younger users than the other two systems, focuses on self-directed learning, it includes tools to draw images and record sounds.
- ✓ Scratch builds on the constructionist ideas of Logo (Kay 2010; Steinmetz 2002), to help users make their projects personally engaging, motivating, and meaningful.
- ✓ Scratch makes it easy to import or create many kinds of media (images, sounds, music); it was designed to invite scripting, provide immediate feedback for script execution, and make execution and data visible.
- ✓ The Scratch user interface strives to make navigation easy.
- ✓ Scratch is tinkerable because it lets users experiment with commands and code snippets the way one might tinker with mechanical or electronic components.
- ✓ Scratch does not require that the user create complete scripts before running the projects. Program fragments can be left in the scripting pane and are saved with the project. (Maloney, 2010)



Fig. 2. The Scratch user interface, (Maloney, 2010)

- ✓ Scratch provides visual feedback to show script execution.
- ✓ Scratch can also show command sequencing and flow of control. Enabling single-stepping (selected from a menu) causes blocks to flash as they run.
- ✓ It uses a single window, multi-page design to ensure that key components are always visible.
- ✓ Scratch avoids floating palettes; the left pane is the command palette with buttons to select categories, the middle pane shows the scripts for the currently selected sprite, with folder tabs to view and edit the costumes (images) and sounds owned by that sprite. The large pane on the upper right is the stage, where the action happens. The bottom-right pane shows thumbnails of all sprites in the project, with the currently selected sprite highlighted.
- ✓ To invite scripting, the command palette is always visible. The commands are divided into eight categories such as Motion, Looks, Sound, and Control. This avoids long, potentially overwhelming, lists of commands: in most palettes, all the commands can be viewed without scrolling. In each category, the most self-explanatory and useful commands appear near the top of the command palette. Command blocks are color-coded by category, helping users find related blocks.

(Source:

<http://web.media.mit.edu/~imaloney/papers/ScratchLangAndEnvironment.pdf>)

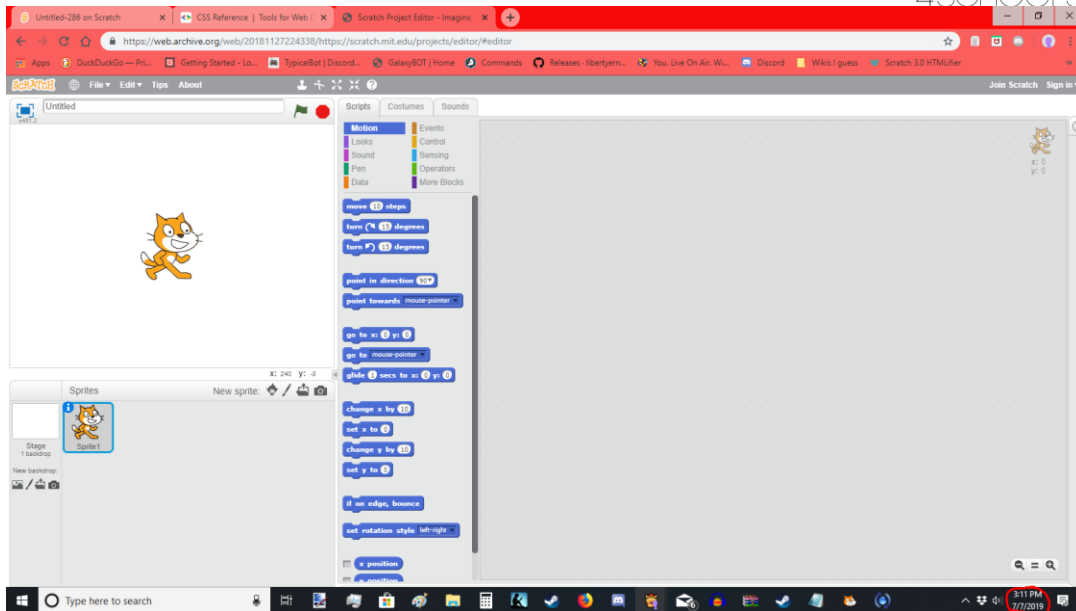


Fig. 3. The Scratch 2.0 editor working online (<https://scratch.mit.edu/>)

Scratch updates the display after every command. Seeing the effect of every command, even if only as a brief flash on the screen, provides important visual clues when troubleshooting.

Scratch 1.0 had 92 command blocks. As Scratch has evolved, it has been a constant struggle to keep down the number of commands.

More commands have been added than removed. Scratch 1.4 has 125 command blocks, although some of them do not appear until needed.

If you have an older computer, or cannot install the Scratch 2.0 offline editor, you can try installing Scratch 1.4.

If you are a network administrator: a Scratch 2.0 MSI has been created and maintained by a member of the community and hosted for public download at <http://llk.github.io/scratch-msi/>.

One of the bugs in Scratch 2.0 was the ability for Scratchers to follow users who they could not normally follow, such as themselves or deleted users.

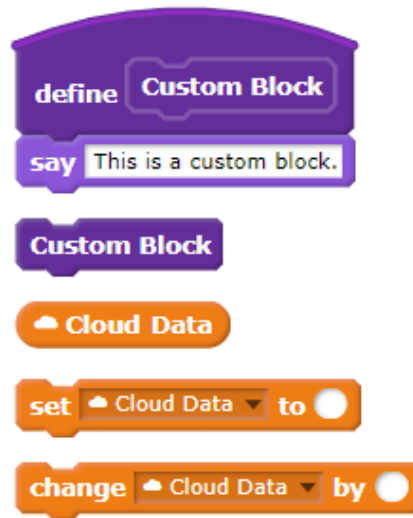






Fig. 4 Examples of New Features and Blocks

Scratch only allows blocks to be connected in meaningful ways. A command block connects when dropped into command sequence, but a function block will not connect if dropped in the same place. As the user drags a block, Scratch gives visual feedback showing possible sequence insertion points (command blocks) or parameter slot targets (function blocks). (Maloney, 2010)

Table 1. Scratch Block Types (Maloney, 2010)

	<p>A <i>command</i> block has a notch on the top and a matching bump on the bottom. Command blocks can be joined to create a sequence of commands called a <i>stack</i>.</p>
	<p>A <i>function</i> block returns a value. Function blocks do not have notches.</p>
	<p>A <i>trigger block</i> has a rounded top. It runs the tack below it when the triggering event occurs.</p>
	<p><i>Control structure</i> command blocks have openings to hold nested command sequences.</p>

In Scratch, a control structure block is an indivisible unit. The closing arm of a loop or conditional block is part of the block itself—it cannot be misplaced—and the nesting of the enclosed command sequence is manifest.



Fig. 5. Shapes indicate type. On the left, command blocks with parameter slots for boolean, number, and string parameters. On the right, boolean and number function blocks. (Maloney, 2010)

3.2.2 Practical Example of Scratch projects

The Scratch blocks language eliminates syntax errors, allowing users to focus on interesting problems right away. Runtime error messages are avoided through failsoft commands, while a carefully designed concurrency model avoids race conditions. For each educational activity, you can try the Tutorial, download a set of Coding Cards, or view the Educator Guides.

The Scratch programming language emphasizes simplicity. The type system and object model were designed to work smoothly without prior explanation, yet to make perfect sense on closer examination.

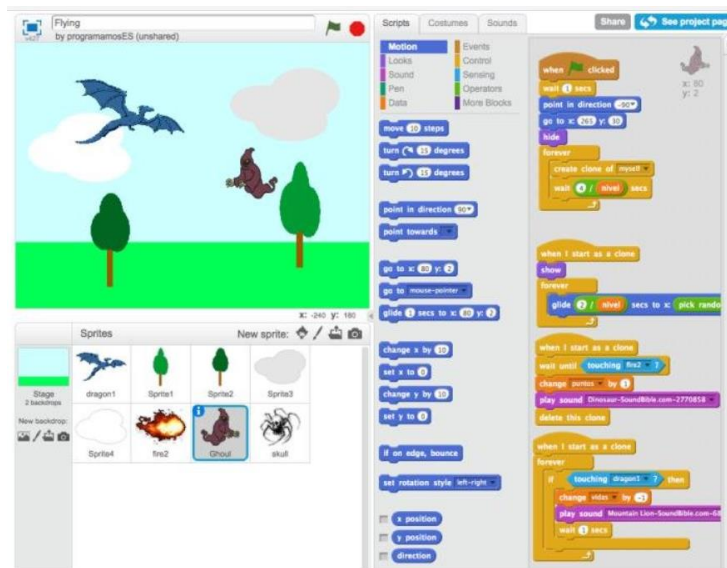


Fig. 6. Example screenshot of a Scratch project. On the right, the visual elements used to program in the Scratch environment. (ComputerProgrammingInTheEnglishClassroom.pdf)

The structure of the Scratch interface makes it easy for us to tinker and explore ideas. To create a game, story, or animation in Scratch, we stack blocks together to form a script that gives instructions to the project sprites.

As we create projects, we evaluate our work and determine whether the results meet our expectations. It's very easy because everything happens in one interface.

Activity - Greeting Card (<https://scratch.mit.edu/projects/11739928>)

Educators can remix Ideas:

- Edit the card for a different occasion
- Change the pictures to match a theme
- Play an animation inside the card

Students can incorporate their voices and images into a project, creating something that helps others learn about themselves and the people, issues, and things they care about. With the Scratch Coding Cards, students can learn to create interactive games, stories, music, animations, and more!

The player should have a script that controls its movement using the mouse pointer. As well, the target should be programmed to show in random spots. If your game design is different, you may need to adjust the scoring system to suit your Scratch project.

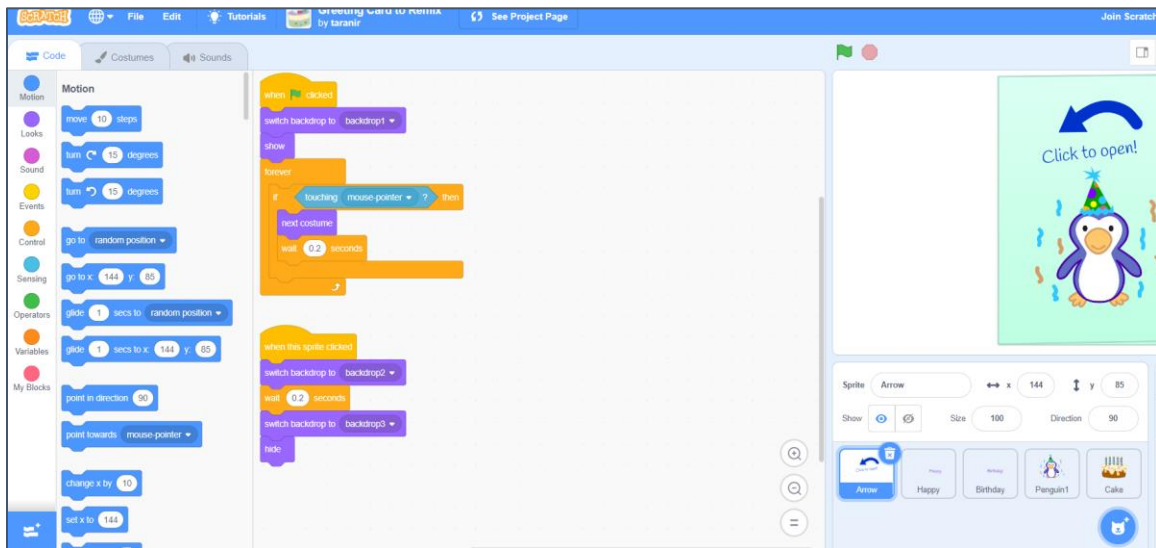
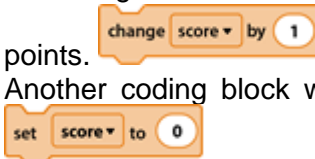


Fig. 7. Example screenshot of a Scratch project, (<https://scratch.mit.edu/projects/11739928/editor/>)

The script you will build requires the use of a *variable*. A variable is a factor that can change. You will create a score variable. It will be used to hold the number of points.

A coding block will be used to increase the *SCORE* by a specified number of points.

Another coding block will reset the *score* variable to zero when a new game begins.



The Scratch programming environment and language work together to create a system that is exceptionally quick to learn, keeping users engaged for years; users can be programming within fifteen minutes.

3.2.3 Conclusions

Scratch has a small number of commands, allows sprites to be exchanged without breaking dependencies, fostering collaboration and code sharing. The system is always live, with no run/edit switch, so commands or code snippets can be run with a click, and graphical feedback shows execution. Variables and lists have concrete visualizations, so the effect of data operations can be seen immediately. (Maloney, 2010)

The ability to code computer programs is an important part of literacy in today's society. When people learn to code in Scratch, they learn important strategies for solving problems, designing projects, and communicating ideas.

Scratch Projects - based on learning units will include different disciplines and this new perspective will allow students to apply what they learn to new situations, leading to deeper learning; students will be engaged in design activities, pursuing personal interests, interacting through creative collaborations and reflecting on useful experiences.

3.3 Assessment test

1. Scratch isn't just for computer science classes. Scratch can be incorporated into any content area in any classroom.
 1. **Yes**
 2. No
2. Scratch is available both online and as a downloadable file.
 1. **Yes**
 2. No
3. By getting pupils to create a math's game with their scratch program they can then apply what they have learnt in their lessons in a practical activity.
 1. **Yes**
 2. No
4. Students can incorporate their voices and images into a Scratch project.
 1. **Yes**
 2. No
5. You can keep score in Scratch by creating a ...
 1. Sprite
 2. Loop
 3. **Variable**
 4. Function
6. Several statements joined together is a ...
 1. Run

2. **Sequence**
 3. Loop
 4. Variable
7. How do you create a loop in scratch?
1. **Use a repeat block**
 2. Snap a block
 3. Use a condition block
 4. Use a variable
8. True or False: you can add your own images into your Scratch projects
1. **True**
 2. False
9. Performing an action depending on IF a criterion is met is called a ..
1. sequence
 2. statement
 3. loop
 4. **condition**
10. Repeating a statement more than once is called a ..
1. **Loop**
 2. Repetition
 3. Event
 4. Sequence

3.4 References

- KAY, A. 2010. Squeak etoys, children, and learning. <http://www.squeakland.org/resources/articles>
- Resnick, M., Maloney, j., Monroy-Hernandez, 2009. Scratch: Programming for all. Comm. ACM 52, 11, 60–67.
- ComputerProgrammingInTheEnglishClassroom.pdf
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. 2010. *The scratch programming language and environment*. ACM Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages. DOI = 10.1145/1868358.1868363. <http://doi.acm.org/10.1145/1868358.1868363>
- <https://education.abc.net.au/home#!/media/1214681/intro-to-scratch-20>
- <https://scratch.mit.edu>
- <http://web.media.mit.edu/~jmaloney/papers/ScratchLangAndEnvironment.pdf>
- <https://www.thomasbuxton.towerhamlets.sch.uk/blogs/year3/2017/11/17/year-3-computing-scratch-projects/>

4 Scratch GPIO (Control GPIO pins/receive inputs) [P3-DANMAR]

4.1 Glossary of terms

Term / Concept	Definition / Explanation
Pi GPIO extension	Pi GPIO extension is an add-on component to Scratch that allows to interact with Raspberry's GPIO pins. This extension provides two additional blocks to read and set the state of all 28 GPIO pins provided by Raspberry.
Raspbian	Raspbian is an operating system based on Debian that is optimised to be used on Raspberry devices. It provides a simple GUI interface and is bundled with Scratch installation and other programming tools that enable the users to interact with the Raspberry hardware.

4.2 Main content

Scratch is often considered as an entrance to the world of programming. Simple algorithms can be developed using this software and the presentation layer composed of many visuals is appealing. However, Scratch for more advanced and interesting applications can seem a bit limited. Thankfully, the GPIO support that can be enabled allows to use Raspberry's pins to control many different physical circuits, devices and appliances.

4.2.1 Basic information about Scratch GPIO

The Scratch environment is great for those starting with microcontroller programming. With the use of Scratch, it is possible not only to design specific algorithms and run them, but also to control Raspberry's GPIO ports to enhance the functionality of the entire system. Scratch supports GPIO out of the box, and this can be done by adding a special Pi GPIO extension.

Once the GPIO extension is added to Scratch, two new blocks are available for further use. These are: stack block and Boolean block. The stack block is a simple instruction that can be used to set any GPIO port to either high or low state. The Boolean block, on the other hand, can be used to check whether any given GPIO pin is in high or low state. Doing so enables Scratch to control all the available GPIO pins of the Raspberry board.

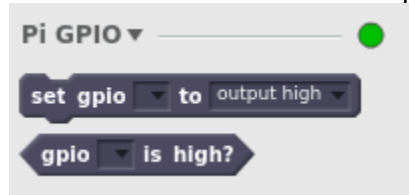


FIGURE 1 ADDITIONAL BLOCKS PROVIDED BY GPIO EXTENSION

Generally speaking, GPIO extension for Scratch opens a wide range of new possibilities, as the user is no longer restricted in designing algorithms inside Scratch software, but

instead is able to interact with physical devices such as diodes, buzzers and so on. The blocks provided by the extension are fully compatible with standard (built-in) Scratch blocks, thus expanding the wide area of options even further.

In order to add the GPIO extension to Scratch, it is required to open the application and from Scripts tab select the last option More Blocks. Once the More Blocks button is clicked, it is required to click on Add an Extension.

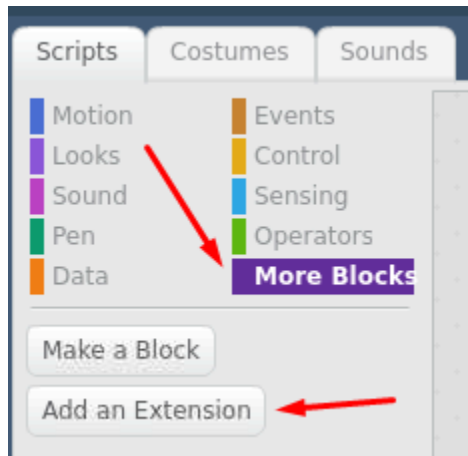


FIGURE 2 ENABLING GPIO EXTENSION

When the screen with Extension Library is shown, the selection of Pi GPIO extension should be made and the choice needs to be confirmed.

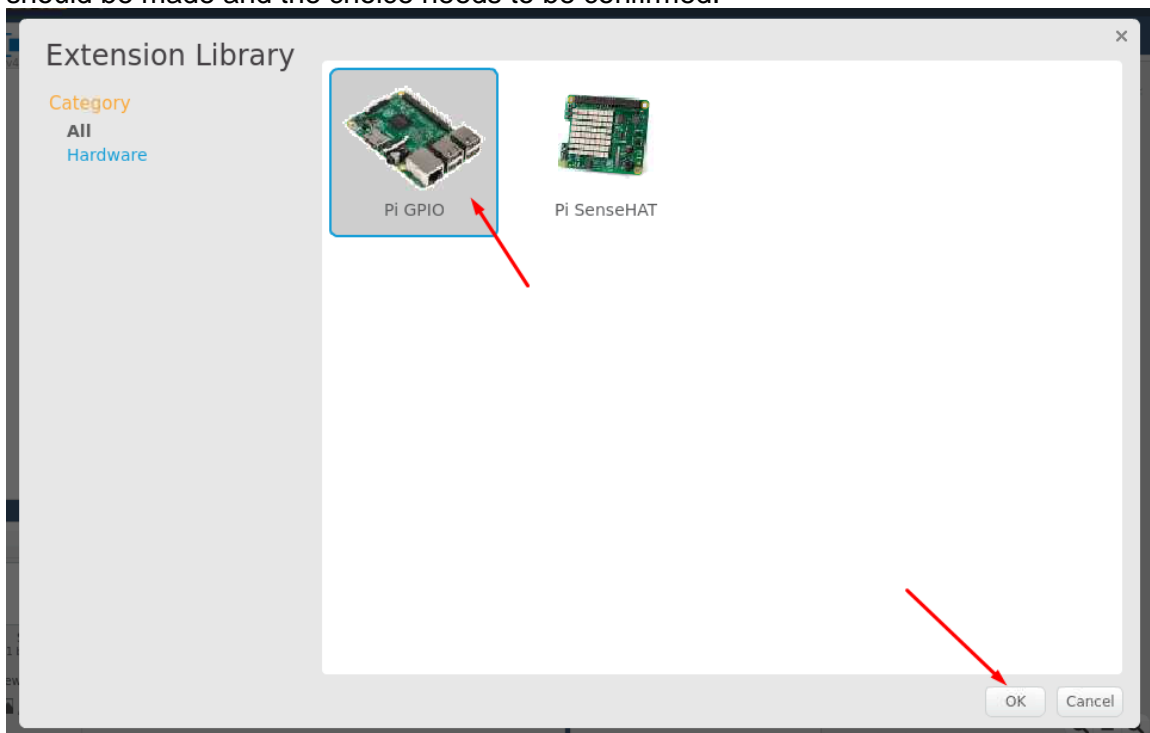


FIGURE 3 ENABLING GPIO EXTENSION

After completing the above steps, Scratch is ready to control the Raspberry's GPIO.

In order to ensure a seamless experience, the best way to get the environment up and running, it is generally advisable to install a Raspbian operating system on Raspberry, as this distribution has everything already set up. Different Scratch versions are available, so the users can select the one they are most familiar with.

4.2.2 GPIO naming convention

It is important to mention that the numbers of GPIO pins used in Scratch follow the naming convention used by Raspberry itself. This can be confusing especially for the beginners, as Raspberry in most models uses a 40-pin connector and the pins are numbered from 1 to 40. In order to avoid confusion, it is best to have a so-called pinout chart for Raspberry. An example available on the official Raspberry's documentation is presented below.

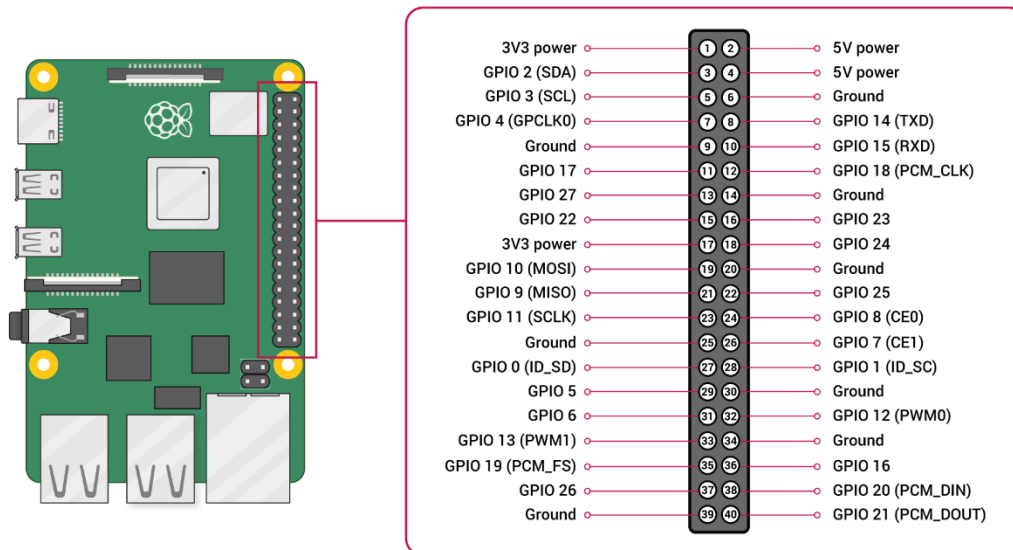


FIGURE 4 RASPBERRY PINOUT CHART

Therefore, if the code in Scratch uses number 26, this denotes the physical pin 37 (left column, second from the bottom). It is important to get acquainted with the location of specific GPIO pins to ensure that there is no confusion between numbers denoting physical pins and those denoting GPIO pins.

4.2.3 GPIO interaction

As it was already mentioned, GPIO blocks are fully compatible with blocks provided by Scratch. Therefore, the simplest example to demonstrate how specific pins can be controlled from Scratch, a simulation of lighting an LED can be made in a very basic way. The following assumes that the LED is connected on GPIO pin 11.



FIGURE 5 USING AN OUTPUT GPIO PIN IN SCRATCH

The above example uses the stack block provided by the GPIO extension. The Boolean block can also be used, for example to act upon detecting a high state on any given GPIO pin.



FIGURE 6 USING AN INPUT GPIO PIN IN SCRATCH

It might seem that having only two additional blocks is somewhat limiting. In reality, though, microcontroller programming is all about two states: high and low. What is more, all modern digital devices operate this way. The high and low states can be translated into logical values of 0 and 1, and in reality, this means no voltage (0V) or voltage is present (3.3V), respectively. For Raspberry, the GPIO operate at 3.3V and it is good to keep that in mind for connecting various devices, sensors and equipment to these pins.

The GPIO extension can also be an interesting drop-in replacement for existing Scratch projects. Instead of interacting with the user solely with the use of Scratch screen, some operations can be transferred to physical GPIO pins. As an example, instead of showing a message or playing a sound, GPIO extension can be used to emit the sound, turn on LED and so on. The possibilities of this setup are limited only by the imagination and creativity of a person working on a specific project using Scratch with Raspberry's GPIO support.

4.3 Practical examples

On the next few pages a few practical examples are provided. These examples are fairly easy to be created and their main purpose is to demonstrate how Scratch can be used to control GPIO pins on Raspberry.

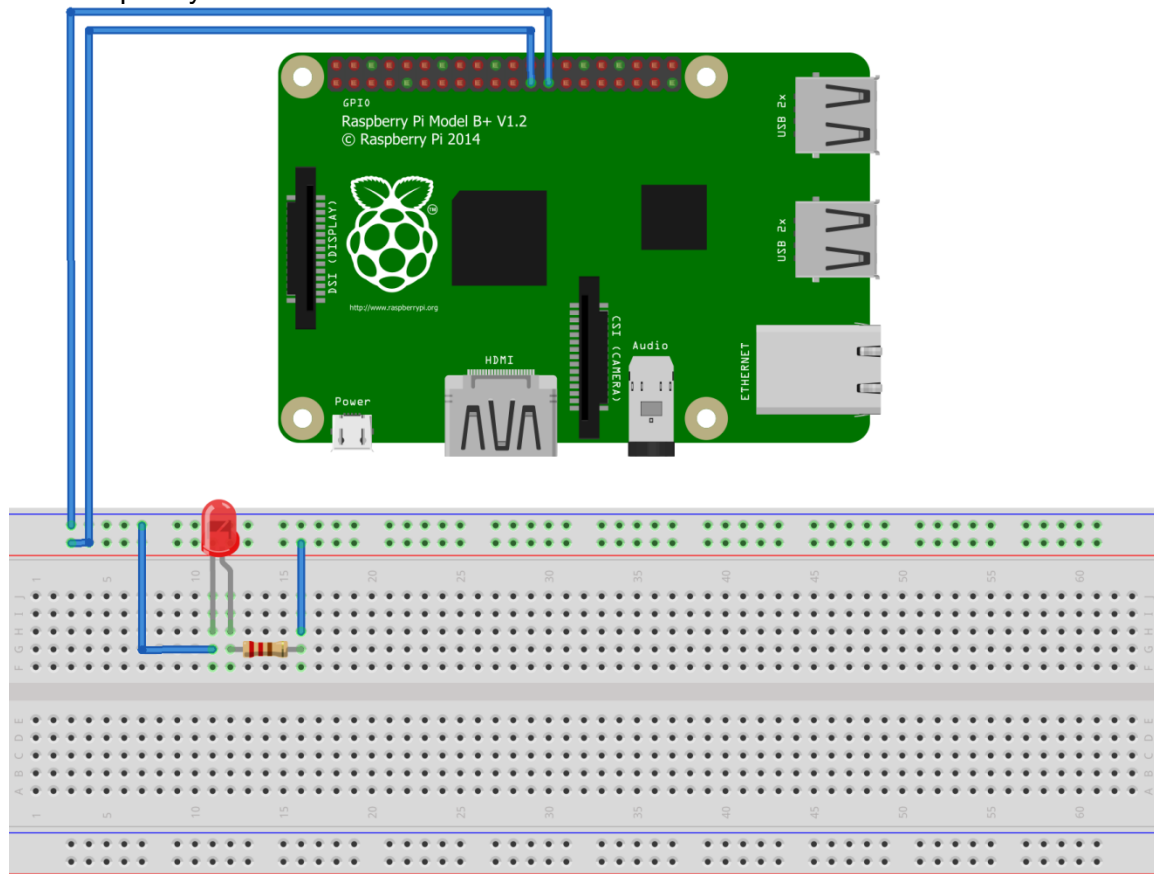
4.3.1 Blinking LED – example 1

The first practical example is going to introduce the basics of working with GPIO controlled by Scratch. The idea is that a very simple circuit will be used that will blink the LED. The LED is going to be controlled by GPIO pin number 11 (physical pin 23). While any other

GPIO can be used, the GPIO pin 11 is very convenient to use, as it has ground (GND) next to it (physical pin 25).

In order to set up this circuit, an LED, a resistor suitable for LED, a breadboard and two jumper wires are needed.

The Raspberry should be connected with the breadboard as indicated below.



fritzing

FIGURE 7 CONNECTING LED TO RASPBERRY

In order to make this very simple circuit work, the selected GPIO pin 11 should be controlled by Scratch that will set its state to high, pause for one second, set the state to low and again pause for one second. These four instructions can be placed inside a forever loop, but also other types of loops can be used, such as repeat. The sample set of blocks is demonstrated below for reference.



FIGURE 8 TOGGING THE GPIO PIN STATE IN SCRATCH

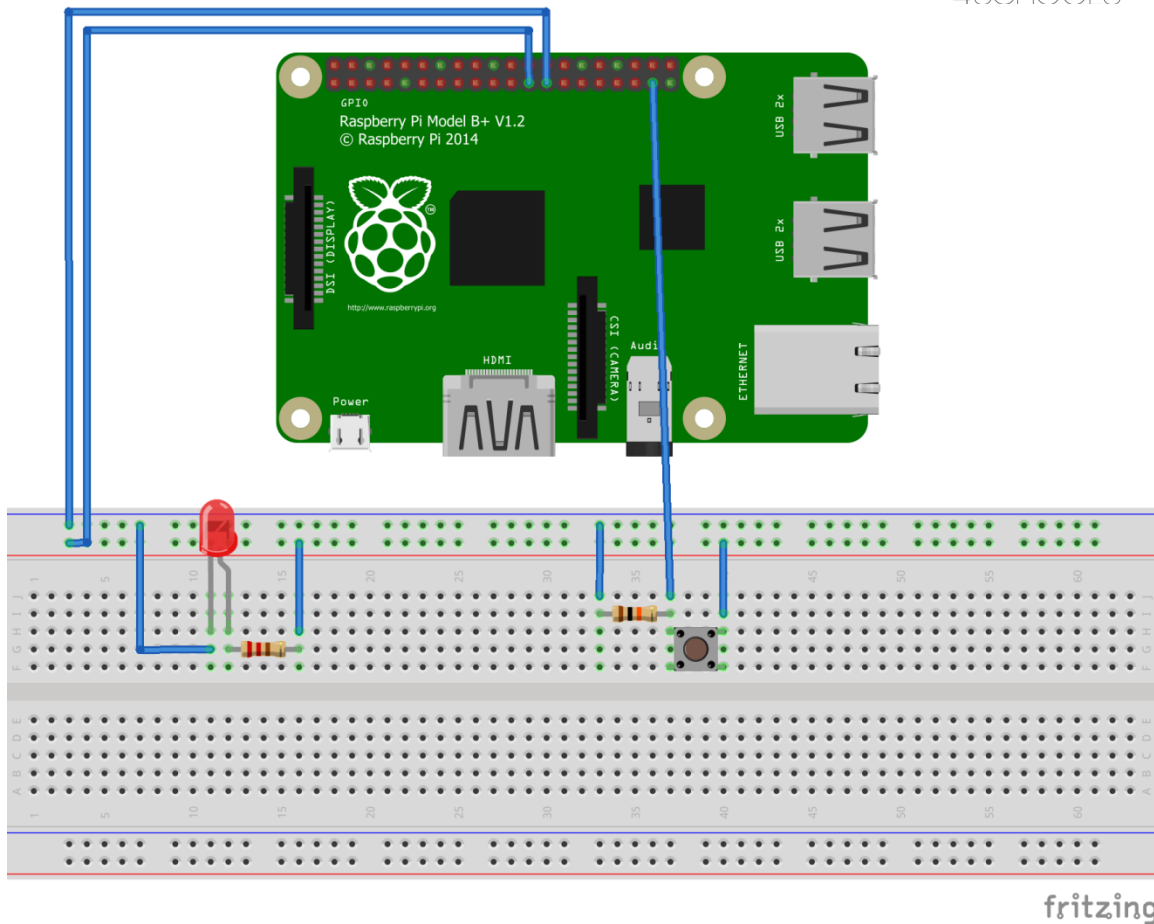
What happens here is that Scratch is communicating with Raspberry using the GPIO extension and issues instructions that allow to change the state of the output pin (11) from high to low. This translates into voltage being present on GPIO pin in specific time intervals.

While this example is very simple, it provides a starting point on controlling Raspberry's GPIO with the use of Scratch. In this simple circuit, the GPIO pin 11 is acting as an output pin, which means that it only receives instructions to change its state accordingly. With GPIO extension it is also possible to read the state of the GPIO pin that will then act as an input pin. This approach is presented in the next example.

4.3.2 Controlling LED with a tactile switch – example 2

In order to present how Scratch can read Raspberry's GPIO pin as an input, a simple circuit can be constructed. In addition to the example presented above, an additional tactile switch is required along with a 10k pull-down resistor and additional jumper wires. The circuit that was used in a previous example needs to be expanded by a tactile switch that will set the GPIO pin to a high state once pressed. To do so, GPIO pin number 26 can be used (physical pin 37).

Because the expected behaviour is that the GPIO pin 26 reads low state when the tactile switch is not pressed and high otherwise, there is a need for a pull-down resistor to be placed between the GND and the GPIO pin 26. Thanks to this, whenever the request to read the state of that pin is issued, it is going to read low (0V) when the tactile switch is not pressed. On the other hand, when the tactile switch is pressed, the reading should be high. This can be achieved by connecting the tactile switch between the GPIO pin 26 and 3.3V line provided by Raspberry on physical pins 1 and 17 that are already provided with jumper wires to a breadboard. Pressing the tactile switch will make the current to flow from a 3.3V line, thus setting the GPIO pin 26 in a high state. An example circuit is provided below for reference.

**FIGURE 9 TACTILE SWITCH CONTROLLING LED CIRCUIT**

Finally, the Scratch code can be provided to make the whole circuit work as planned. To do so, a forever loop can be used. Within this loop the GPIO pin 26 needs to be read. In Scratch terms, a stack block can be used as a parameter to the if-statement. The remaining instructions follow the already outlined logic of the circuit.

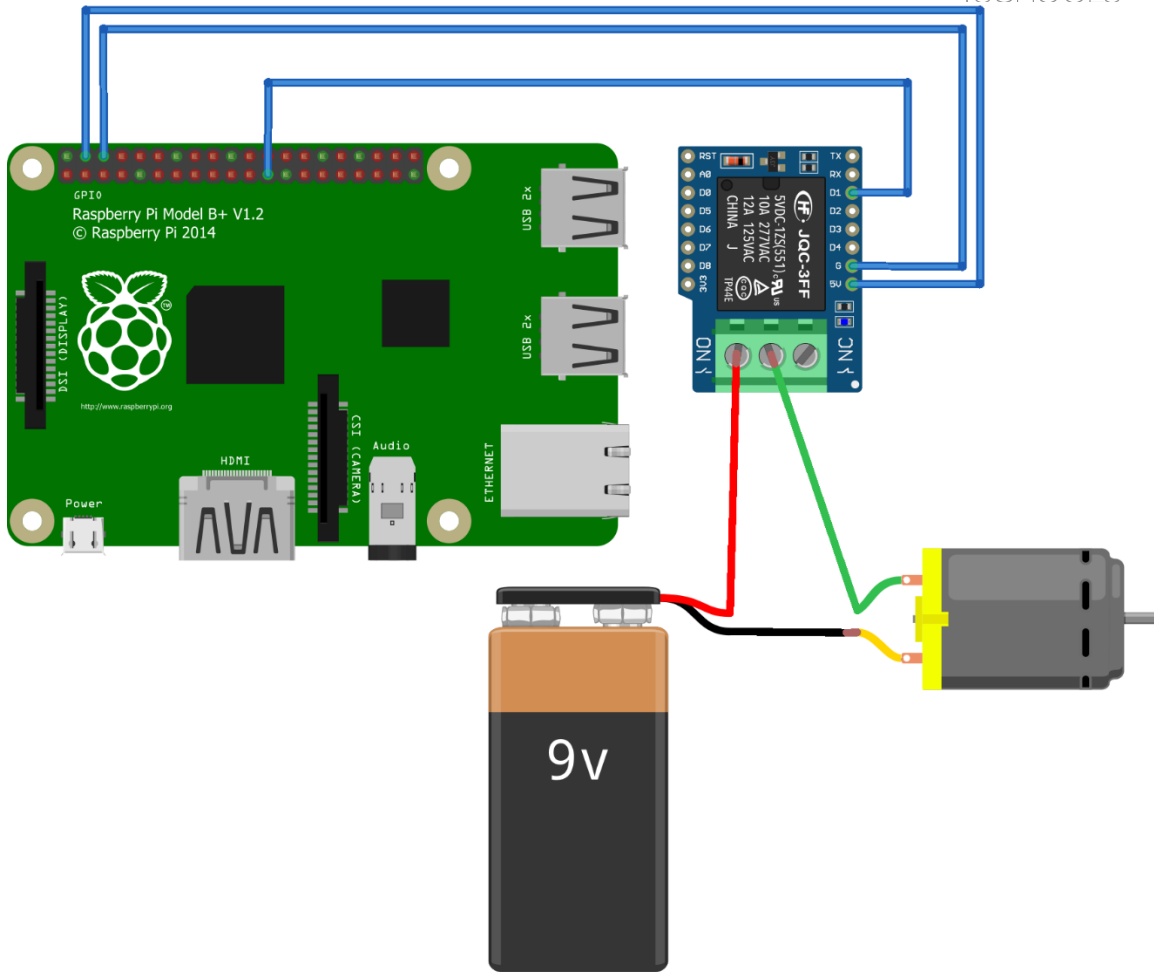
**FIGURE 10 READING GPIO STATE IN SCRATCH**

As soon as the tactile switch is pressed, the GPIO pin 26 is going to be put in a high state. If such state is detected, the previously used GPIO pin 11 should be set to high in order to turn the LED on. The difference between the two GPIO pins is that pin 26 is considered as an input pin (and Scratch checks its state continuously), whereas pin 11 is an output pin that controls the LED by turning it on (providing voltage in high state) and off (no voltage in low state).

4.3.3 Expanding the use of GPIO – example 3

Using Scratch to control Raspberry's GPIO can lead to a false illusion that in a worst-case scenario the code is not going to work. In reality, however, interacting with Raspberry hardware requires an extra attention. One of the examples is that the GPIO pins are not allowed to draw more than 50mA of current. While this is fine for an LED or a buzzer, any more current-hungry resources are not going to work and may result in a permanent damage to the Raspberry board. Such cases, however, can be solved with the use of a relay and an external power source.

A relay switch typically operates at 5V and this voltage is provided by Raspberry on pins 2 and 4. At the same time, Raspberry's GPIO output pin set to a high state is enough for triggering the relay and switching it to operating mode. The external circuit connected to the relay will be closed, and the current will start to flow, powering any device. This can be pictured by setting up a circuit in which Scratch controls the relay with the use of a GPIO output pin and the relay then powers up a motor that requires (as an example) 9V. The relay switch can be connected to pins 4 (5V output) and 6 (GND). The relay's control/steering can be connected to GPIO pin 11 that will be used as an output pin. Then, the external circuit (a 9V motor with own power source) needs to be wired with the relay switch. An example setup is provided below for reference.



fritzing

FIGURE 11 USING A RELAY SWITCH TO TURN ON MOTOR

Inside Scratch, the instruction that will turn the relay switch on is a stack block that is going to set GPIO pin 11 to high state. This can be triggered using a sprite representing the cat. Another improvement over previous examples is that there will be one input provided from Scratch that will act differently based on the current state (reading) of GPIO pin 11. For this purpose, there can be an action that will be executed whenever the cat's sprite is clicked. If the current reading of the GPIO output pin is high, it will be set to low. If it is low, it will be set to high. GPIO pin will remain in the set state, therefore acting as a simple two-state switch. Additionally, the cat is going to say "Hello!" to have an additional visual feedback that the block of code is indeed executed. An example Scratch code is provided below for reference.



FIGURE 11 SPRITE ACTING AS A TWO-STATE SWITCH READING ITS OWN STATE

In this example it was demonstrated that Raspberry's GPIO ports can be used in a quite flexible way. Even though the GPIO pin 11 is mostly used as an output pin driving (turning the relay switch on and off), at the same time it is possible to build the logics of the circuit around it by checking its current state.

4.4 Assessment test

1. Scratch requires an additional extension to work with Raspberry's GPIO
 - a. **True**
 - b. False
2. The GPIO extension adds three additional blocks to Scratch
 - a. True
 - b. **False**
3. If the GPIO pin is used as an output, Scratch is not able to read its state
 - a. True
 - b. **False**
4. The maximum current that any connected circuit draws from GPIO port is:
 - a. 25mA
 - b. 35mA
 - c. **50mA**
5. Checking the state of a GPIO pin can be done through:
 - a. a stack block
 - b. **a Boolean block**
 - c. a control block
6. The GPIO extension uses numbers that denote:
 - a. Raspberry's numbered pins
 - b. **Raspberry's GPIO-assigned numbers**
 - c. Scratch naming convention
7. Setting a GPIO pin to high state in Scratch will result in pin's voltage to be:
 - a. 0V
 - b. 1V
 - c. **3.3V**



8. Scratch is able to control only selected GPIO pins
 - a. True
 - b. False**
9. The suggested operating system to work with Scratch and GPIO is:
 - a. Windows
 - b. Raspbian**
 - c. macOS
10. The stack block provided by GPIO extension is used to:
 - a. change the state of GPIO pin**
 - b. read the state of GPIO pin
 - c. all of the above

4.5 References

- <https://www.raspberrypi.org/documentation/usage/gpio/>
- <https://www.raspberrypi.org/documentation/usage/gpio/scratch2/README.md>
- <https://www.circuits.dk/everything-about-raspberry-gpio/>
- <https://thepihut.com/blogs/raspberry-pi-tutorials/tutorial-tactile-switch>
- <https://raspberrypihq.com/use-a-push-button-with-raspberry-pi-gpio/>

4.6 Additional resources

1. Raspberry Pi GPIO: <https://www.raspberrypi.org/documentation/usage/gpio/>
2. Scratch Wiki: <https://scratch-wiki.info/>
3. An alternative GPIO extension for Scratch:
<https://raspberry-valley.azurewebsites.net/GPIO-with-Scratch/>
4. Sparkfun education on Scratch and GPIO:
<https://sparkfuneducation.com/how-to/scratch-gpio-control-guide.html>
5. Tutorial for beginners on Scratch and GPIO:
<https://www.magicbytes.com/coming-soon/gaming-computer-lab/tutorials-s/learn-to-code/scratch-gpio-beginner>

5 Introduction to Raspberry Pi Edition of Minecraft [P4-HESO / P5-SCHOLE]

5.1 Glossary of terms

Term / Concept	Definition / Explanation
Raspberry Pi	Raspberry Pi is a credit card sized, fully functional computer which operates on Raspberry Pi OS.
Minecraft	Minecraft is an open-world educational game where players can build their own virtual worlds with blocks that represents different material.
Raspberry Pi OS	The operating system for Raspberry Pi.
Python	Object-oriented programming language that will be used to build things automatically in Minecraft.

5.2 Main content

Minecraft Pi is a version of Minecraft, with minimal features, developed for Raspberry Pi. Pi edition is intended as an educational tool for novice programmers, allowing users to enjoy the game and learn programming at the same time. This document presents the most important and practical guidelines for Minecraft Pi, such as how to control the player, manually build with blocks and use the Python interface to manipulate the world around you. It is meant for educational purposes and is considered a quick but all-inclusive manual for introducing a new player to Minecraft Pi.

Module 3 consists of the following elements:

- Introduction to Minecraft Pi basic functions
- Minecraft Pi elements and gameplay
- Controlling Minecraft Pi with Python
- Interaction of Minecraft Pi with the physical world through the Raspberry Pi's GPIO:
 - Connecting LEDs, buttons and switches
 - Create electronic kits to interact with Minecraft Pi
- Importing new Minecraft Pi maps and resource packages
- Practical examples of Minecraft Pi and Python
- Assessment quiz to test acquired knowledge
- Extra resources to advance your knowledge even further

5.2.1 Introduction to Minecraft Pi basic functions

Minecraft Pi is an open-world game where players use blocks that represent different materials to build virtual worlds. You can create anything from a single house to a huge castle and from a small crops field to a big city.

Minecraft Pi can be manipulated using Python scripts which interact with various game functions. The Raspberry Pi edition of Minecraft comes with an API (Application Programming Interface) which allows you to control the game using Python programming.

Python will be used to manipulate blocks and structures, send in-game messages, automate building processes and create mini games.

Minecraft Pi also supports multiplayer, which means that more than one player can play and interact with each other in the same map. When several STEMKIT computers are connected via the same Wi-Fi or Ethernet network, the multiplayer mode is enabled, and several users can connect to the same Minecraft world.

To run Minecraft Pi on your STEMKIT computer you should double-click the desktop icon of Minecraft Pi, or go to **Main Menu** (Raspberry Pi logo on the up-left corner) → **Games** → **Minecraft Pi** and click on it (Figure 1). When the game loads, click **Start Game** → **Create New**.

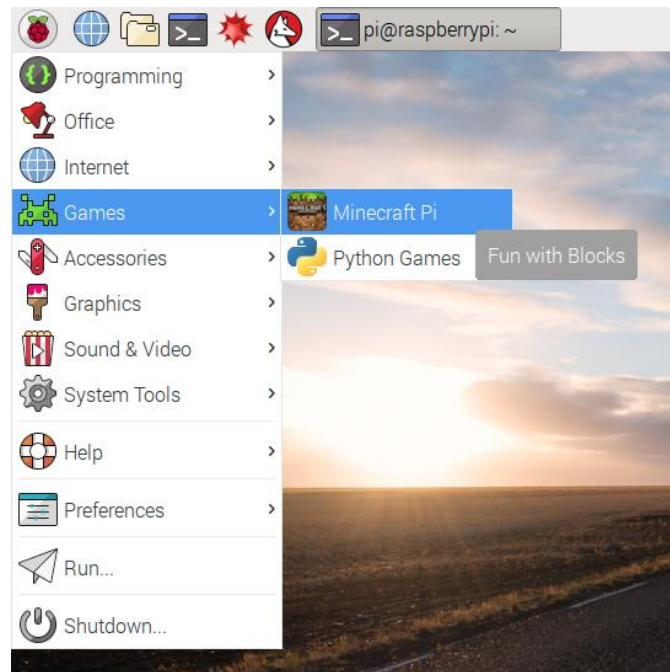


FIGURE 1 RUNNING MINECRAFT PI.

Basic functions include navigation and control when entering a Minecraft world. Use the mouse to look around, left click to smash blocks and right click to build blocks. Most of the control functions comes from the keyboard, as shown in Figure 2.

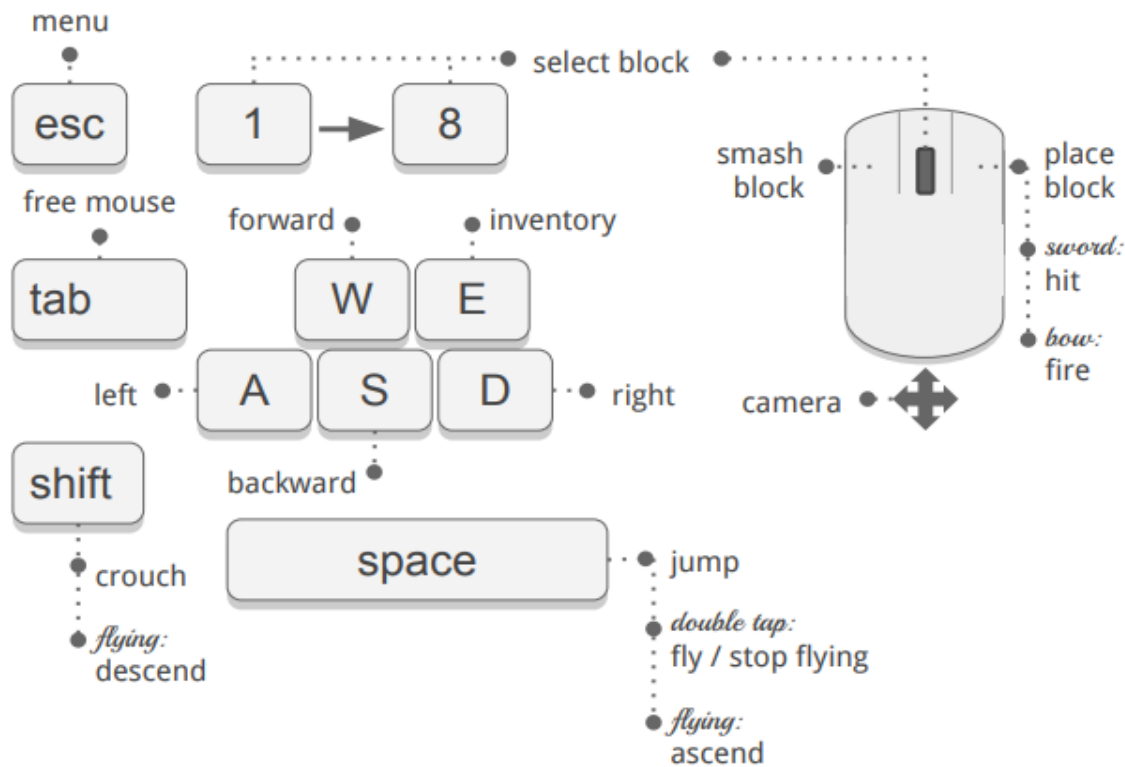


FIGURE 2 MINECRAFT CONTROLS OVERVIEW. SOURCE: [HTTPS://ARGHBOX.WORDPRESS.COM/2013/07/28/MINECRAFT-PI-CONTROLS/](https://arghbox.wordpress.com/2013/07/28/minecraft-pi-controls/)

5.2.2 Minecraft Pi elements and gameplay

When first entering the game, the user starts with a sword in hand, which can be used to remove blocks. On the bottom of the screen there is a partial view of the inventory with different block types and tools which can be accessed quickly by rolling up or down the mouse wheel. These items are usually the most common used in game but can be changed depending on what the user wants to build and what material wants to have quick access to.

Tapping the Space bar, the character jumps. Double tap it to start ascending (flying around). When Space bar is released, ascending stops. In order to fall to the ground, the player needs to double tap Space bar again.

Pressing "E", will open the inventory which contains all the different block types offered in Minecraft Pi. Browse over and left click to the block you want to use.

Pressing TAB will release the mouse from the game, giving the opportunity to the user to use other Raspberry Pi programs. When writing scripts, the TAB key will enable the user to navigate through different Python files, the Python compiler, or the Command line, depending on how the user wants to program.

Pressing the SHIFT key while in game will make the character crouch. During flight the SHIFT key will make your character descend.

Finally, keys 1 to 8 on the keyboard give the user quick access to materials and tools already in the quick access inventory.

Blocks are the basic units of structure in Minecraft. Each block is 1m^3 . They are arranged in a grid and are snapped to it, which means a block cannot be in more than one cell. There are some exceptions, such as beds which consist of two blocks, each making up one half of it.

Blocks build up the in-game environment and can be harvested and utilized in various fashions. Some blocks, such as dirt and sandstone, are opaque, while other blocks, such as glass and ice, are transparent. Some blocks, such as torches, will emit light. Almost all blocks ignore gravity, with the exception of sand and snow.

There are various types of blocks in Minecraft Pi. Natural blocks refer to blocks generated as part of the terrain. Plants, animals and fungi are Minecraft's representation of various life forms.

Each type of block has its unique block ID associated with it. Block IDs are needed to manipulate different types of blocks when writing code in Python. It is helpful to have a list of these ID numbers to use them in your code.

For example, block ID for "Stone" is 1, for "Grass" is 20 and so on. The complete list of Block ID numbers can be found in the following link: <https://www.raspberrypi-spy.co.uk/2014/09/raspberry-pi-minecraft-block-id-number-reference/>.

5.2.3 Controlling Minecraft Pi with Python

Minecraft Pi has an API that allow the user to communicate with and control the game using Python scripts. The API allows you to write programs which control, alter and interact with the Minecraft world. The user can create massive structures at the click of the button or a bridge which automatically appears allowing the user to walk across massive chasms or a game of minesweeper, a huge real time clock, a programmable directional cannon, or turn blocks into bombs (O'Hanlon, 2013) (<https://www.stuffaboutcode.com/2013/04/minecraft-pi-edition-api-tutorial.html>).

Since Minecraft is a world of cubes or blocks, with a relative size of $1\text{m} \times 1\text{m} \times 1\text{m}$, every block has its own unique position in the world in x, y, z coordinates (**x** is forward/back, **z** is left/right, and **y** is up/down).

Using Python scripts, you can do, among others, the following:

- Get the player's position.
- Change or set the player's position.
- Get the type of block the user stands on or is near him
- Change a block type with another block type.
- Change camera angles.
- Post messages to the player.

To open Python interface go to Main Menu → Programming, and click on Thonny Python, as shown in Figure 3.



FIGURE 3 LOCATING THONNY PYTHON

In our first lines of code we will learn how to connect Minecraft Pi to our script. First of all, save your file using the *first.py* name. Remember to always save your files and use the *.py* file extension so your program can be interpreted a Python file.

Figure 4 shows the lines of code that are required in order to connect Minecraft Pi with Python.

```

first.py x
1  from mcpi.minecraft import minecraft
2
3  mc = Minecraft.create()
  
```

FIGURE 4 CONNECT PYTHON TO MINECRAFT PI.

Firstly, we import Minecraft. Then, we save the Minecraft game object to a variable called “mc”, which will be used later to call commands in your program.

Our first complete program will communicate a message to our character in Minecraft. Create a new file and save it under the name *hello.py*.

Figure 5 shows the code you need to write:



```
hello.py ✕  
1 from mcpi.minecraft import minecraft  
2  
3 mc = Minecraft.create()  
4  
5 mc.postToChat("Hello World!")
```

FIGURE 5 SENDING A MESSAGE TO THE PLAYER.

When you write the code, click Save and tap F5 on your keyboard to run your script. A message appears in Minecraft.

Finding player's location:

Finding the player's position is necessary before building any structures. To do that we use the command `mc.player.getPos()`. There are two ways of finding player's position:

1. We use a variable named `pos` and we save player's coordinates, as shown in Figure 6.

```
pos.py ✕  
1 from mcpi.minecraft import minecraft  
2  
3 mc = Minecraft.create()  
4  
5 pos = mc.player.getPos()
```

FIGURE 6 SAVING PLAYER'S POSITION ON VARIABLE POS.

2. We save player's position is `xyz` coordinates, as shown in Figure 7 (`x` is forward/back, `z` is left/right, and `y` is up/down)

```
pos.py ✕  
1 from mcpi.minecraft import minecraft  
2  
3 mc = Minecraft.create()  
4  
5 x, y, z = mc.player.getPos()
```

FIGURE 7 SAVING PLAYER'S POSITION ON XYZ COORDINATES.

Teleporting our player:

Having our player's coordinates also means that these coordinates we can manipulate them, meaning that we can teleport our player in various places in a Minecraft world.

To do that we use the command `mc.player.setPos()`. Figure 8 shows how to use this command to teleport our player 200 spaces up in the air and then watch it falling to its original position on the ground.

```
tel.py x
1 from mcpi.minecraft import minecraft
2
3 mc = Minecraft.create()
4
5 x, y, z = mc.player.getPos()
6
7 mc.player.setPos(x, y+200, z)
```

FIGURE 8 TELEPORTING OUR PLAYER 200 SPACES UP IN THE AIR.

Managing blocks:

Apart from finding our player's position and manipulate it, we can use player's position to interact with blocks. One of the things we can do is to know the exact block our player is standing on. To do that, firstly we find the tile position by using the `mc.getTilePos()` command and then we use the `mc.getBlock(x, y, z)` command, as shown in Figure 9.

```
tile.py x
1 from mcpi.minecraft import minecraft
2 |
3 mc = Minecraft.create()
4
5 x, y, z = mc.getTilePos()
6
7 blockBelowPlayerType = mc.getBlock(x, y, z)
```

FIGURE 9 FINDING TILE POSITION AND SAVING IT INTO COORDINATES.

Then we can generate blocks around our player by using the `mc.setBlock(x, y, z, blockType, blockData)` command. Coordinates `xyz` refer to the location in which we generate a block, `blockType` refers to the difference block type IDs, and `blockData` refers to extra properties that some blocks have (e.g., different colors). Figure 10 shows how to generate a block of stone (`blockType` is 1) next to our player.

```

block.py x
1  from mcpi.minecraft import minecraft
2
3  mc = Minecraft.create()
4
5  x, y, z = mc.player.getPos()
6
7  mc.setBlock(x+1, y, z, 1)
  
```

FIGURE 10 GENERATING A BLOCK OF STONE NEXT TO THE PLAYER.

As mentioned, some blocks have extra properties. For example:

- Wool (ID-35) □ 0: white, 1: orange, 2: magenta, 3: light blue, 4: yellow, etc.
- Wood (ID-17) □ 0: oak, 1: spruce, 2: birch, etc.
- Tall grass (ID-31) □ 0: shrub, 1: grass, 2: fern
- Torch (ID-50) □ 0: pointing east, 1: west, 2: north, 3: south

The whole list of block types and block properties can be found in the following link: <https://minecraft.gamepedia.com/Block>.

Finally, the API allows to generate multiple blocks together so we can create various structures at the click of a button. To do that we use the `setBlocks(x1, y1, z1, x2, y2, z2, blockType, blockData)` command. It works by specifying two sets of coordinates which the API uses to fill the gap between them with a certain block type. Figure 11 shows how to build a 10x10x10 block made of gold.

```

goldblock.py x
1  from mcpi.minecraft import minecraft
2
3  mc = Minecraft.create()
4
5  gold = 41
6
7  x, y, z = mc.player.getPos()
8
9  mc.setBlocks(x+1, y, z+1, x+11, y+11, z+11, gold)
  
```

FIGURE 11 GENERATING A 10X10X10 BLOCK OF GOLD.

Special blocks:

Special blocks within Minecraft Pi refer to blocks that can interact with the surroundings of a Minecraft world. These blocks range from simple mechanisms to flowing lava. The whole list of special blocks can be found in the following link: <https://minecraft.gamepedia.com/Block>. There are two special blocks that are most commonly used in Minecraft: TNT and Lava blocks.

TNT blocks can be used by the player to generate controlled explosions which destroy structures, mountains and fields. The player can place a TNT block like any other block,

but when right clicking on it a couple times that TNT block is detonated, giving a 4-second window to the player to move away before exploding. Its Block ID is 46. Figure 12 shows how to generate TNT blocks using Python code (note that the last digit should always be “1” so the TNT block explodes).

```
tnt.py ×  
1 from mcpi.minecraft import minecraft  
2  
3 mc = Minecraft.create()  
4  
5 tnt = 46  
6  
7 mc.setBlock(x, y, z, tnt, 1)
```

FIGURE 12 GENERATING TNT BLOCKS USING PYTHON PROGRAMMING.

Lava is a light-emitting fluid block that causes fire damage and it spreads in a 3x3 area above it and a 5x5 area below it. Its Block ID is 10. Lava can burn flammable structures such as grass and wood, but also the player so we have to be careful not to step on it. When Lava cools down, it becomes rock. Figure 13 shows how to generate Lava using Python code.

```
lava.py ×  
1 from mcpi.minecraft import minecraft  
2  
3 mc = Minecraft.create()  
4  
5 lava = 10  
6  
7 mc.setBlock(x+3, y, z+3, lava)
```

FIGURE 13 GENERATING LAVA BLOCKS USING PYTHON PROGRAMMING.

Importing Minecraft Pi modules:

To ensure our programs run smoothly without crashing the game, we need to import some Minecraft modules at the beginning of our Python script. These modules also let us access properties and parameters that are needed in order to manipulate a Minecraft world. Figure 14 shows where and how these modules should be imported.

```
modules.py x
1  from mcpi.minecraft import minecraft
2  from mcpi.block import block
3  from time import sleep
4
5  mc = Minecraft.create()
6
7  # rest of program
8  # ...
9  # ...
10 # ...
```

FIGURE 14 NECESSARY MODULES IMPORTED IN A PYTHON SCRIPT.

Recap of Python commands:

- `postToChat("our message")` – communicate with the player in the game;
- `player.getPos()` – get the precise position of a player;
- `player.setPos(x, y, z)` – set (change) the player's position;
- `player.getTilePos()` – get the position of the block where the player currently is;
- `getBlock(x, y, z, blockType, blockData)` – get a block type for a specific position;
- `setBlock(x, y, z, blockType, blockData)` – set (change) a block to a specific block type;
- `setBlocks(x1, y1, z1, x2, y2, z2, blockType, blockData)` set lots of blocks all at the same time by providing 2 sets of x, y, z coordinates.

5.2.4 Interaction with the physical world through the GPIO

This section describes some tutorial so you can make Minecraft to interact with the physical world through the GPIO. You will learn to program buttons to interact with your Minecraft Pi game, automate processes, manipulate blocks and create mini games, and finally to program LEDs to simulate events that happen in Minecraft.

Tutorial 1: Connecting a push button

In this tutorial you will learn how to connect a push button through the GPIO of Raspberry Pi. You should familiarize yourself with the GPIO pins, which pins to use and their numbering. Figure 15 shows the standard Broadcom (BCM) pins names. Numbering is not in numerical order so be aware.

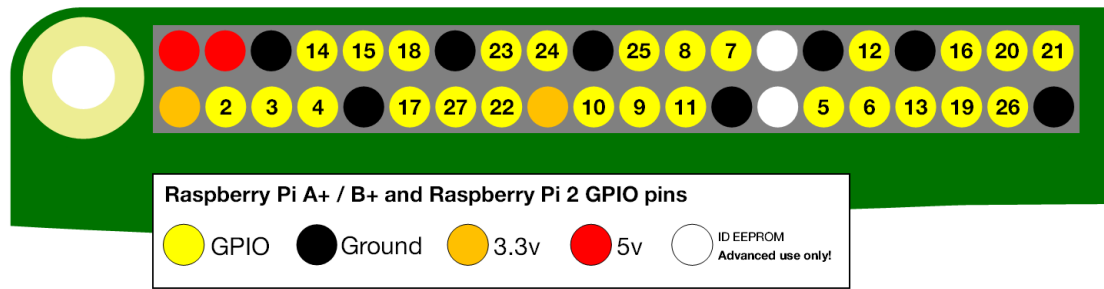


FIGURE 15 GPIO PINS NUMBERING

The following materials are required for this tutorial:

- 1 x breadboard
- 1 x push button
- 1 x 220 Ohm resistor
- 2 x female-to-male jumper cables

Connectivity is very simple and is shown in Figure 16 Note that 1 jumper cable is connected to a 3.3V pin (red) and the other one is connected to a GPIO input/output pin (pin number 16).

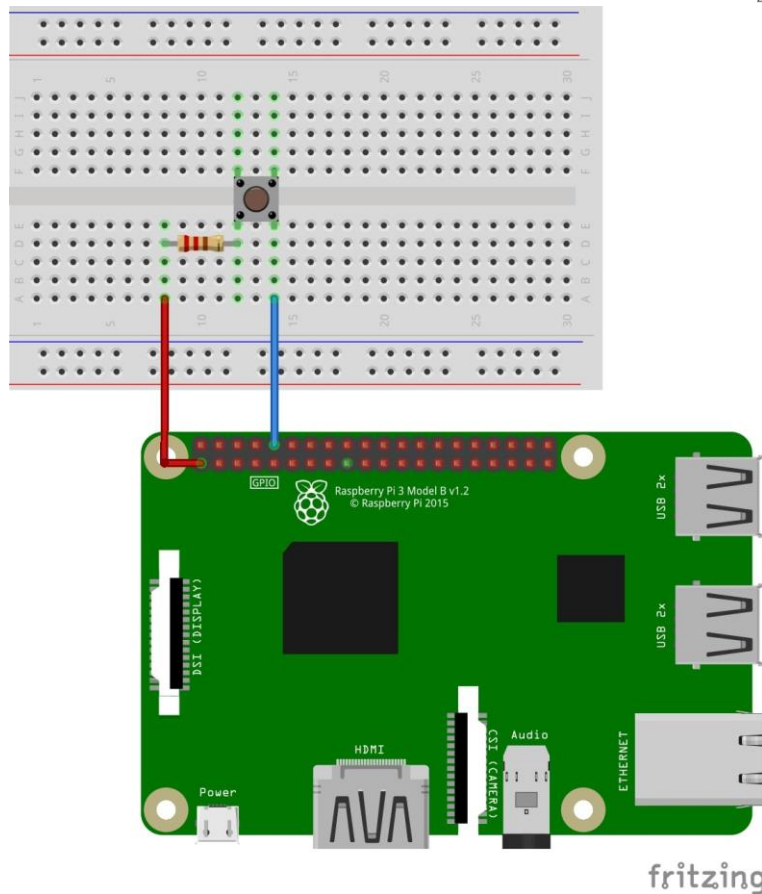


FIGURE 16 CONNECTING A PUSH BUTTON TO RASPBERRY GPIO.
SOURCE: RASPBERRYPIHQ.COM

Now that your circuit is ready, you can jump into a new Python file to test the button's functionality. Figure 17 shows the necessary Python code so you can test your button.

```
button_test.py * x
1 import RPi.GPIO as GPIO
2 import time
3
4 GPIO.setmode(GPIO.BCM) #tell the Pi what headers to use
5 GPIO.setup(15, GPIO.IN) #tell the Pi pin 15 is an input
6
7 while True:
8     if GPIO.input(15) == True: #look for button press
9         print "Button works!" #log result
10        time.sleep(0.5) #wait 0.5 seconds
```

FIGURE 17 PYTHON CODE FOR TESTING BUTTON FUNCTIONALITY

Save and press F5 to run your script. Now every time you push the button the message "Button works!" should appear in Python's terminal window. To stop the script, press *Ctrl+C*.

Tutorial 2: Super mining in Minecraft

In this second tutorial you will use the previous circuit to interact with your Minecraft Pi game. You will create a program that destroys a block of blocks in a Minecraft world every time you push the button. Create a New File and Save it as *mining.py*.

Follow the code in Figure 18 and see what happens in the Minecraft world when you run your script (F5) and push the button. Note that you can manipulate the block types and coordinates as you wish, but do not go too crazy because the Raspberry Pi might struggle.

```
mining.py x
1  import RPi.GPIO as GPIO
2  import sleep
3  from mcpi.minecraft import Minecraft
4
5  mc = Minecraft.create() #connect Minecraft Pi with Python
6
7  GPIO.setmode(GPIO.BCM) #tell the Pi what headers to use
8  GPIO.setup(15, GPIO.IN) #tell the Pi pin 15 is an input
9
10 while True:
11     if GPIO.input(15) == True #look for button press
12         x, y, z = mc.player.getPos() #read the player's position
13
14         mc.setBlocks(x, y, z, x+10, y+10, z+10, 0) #mine 10 blocks
15         mc.setBlocks(x, y, z, x-10, y-10, z-10, 0) #mine 10 blocks
16
17     time.sleep(0.5) #wait 0.5 seconds
```

FIGURE 18 PYTHON CODE FOR TESTING BUTTON FUNCTIONALITY

Tutorial 3: Create a diamond detector

In the third tutorial you will create a program that detects diamond blocks in Minecraft. First you need to create a new circuit using the following material:

- 1 x breadboard
- 1 x LED (any color)
- 1 x 220 Ohm resistor
- 2 x female-to-male jumper cables

Connectivity is very simple and is shown in Figure 19. Note that 1 jumper cable is connected to a ground pin (black) and the other one is connected to a GPIO input/output pin (pin number 23).

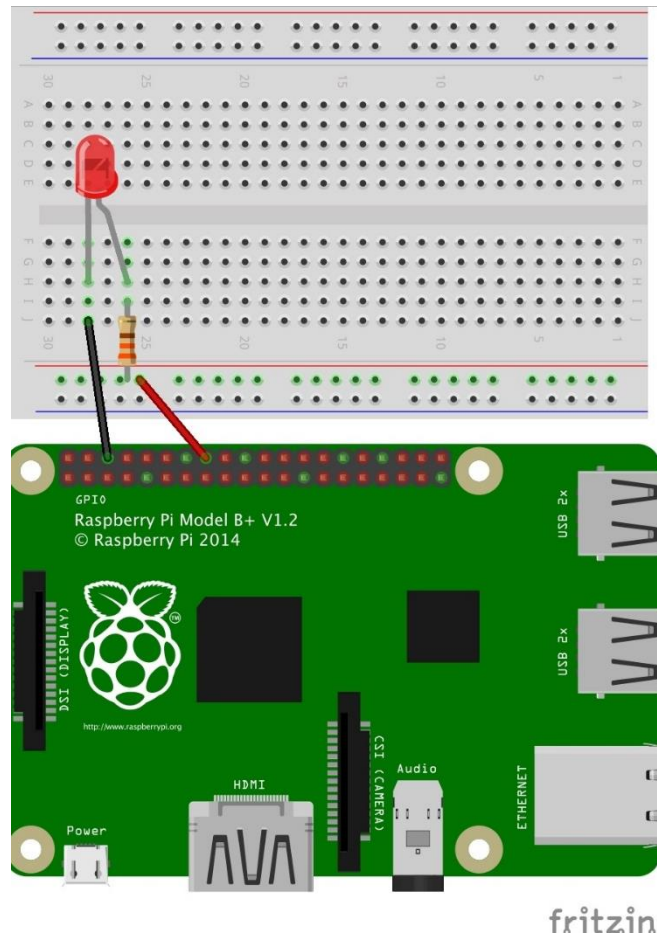


FIGURE 19 CONNECTING AN LED TO GPIO.
SOURCE: RASPBERRYPIHQ.COM

When you have created the circuit, you can open Python, create a New File and Save it as *detector.py*. Follow the code in Figure 20, run your script (F5) and move around in the Minecraft world.



detector.py ✕

```
1 import RPi.GPIO as GPIO
2 import time
3 from mcpi.minecraft import Minecraft
4 from mcpi.blocks import Blocks
5
6 mc = Minecraft.create() #connect Minecraft with Python
7
8 led_pin = 23 #store the GPIO pin number in variable
9
10 GPIO.setmode(GPIO.BCM) #tell the Pi what headers to use
11 GPIO.setup(23, GPIO.OUT) #tell the Pi pin 23 is an output
12
13 while True: #repeat indefinitely
14     x, y, z = mc.playerPos()
15     for i in range(10): #check every block until block 10
16         if mc.getBlock(x, y-i, z) == 56:
17             GPIO.output(led_pin, True) #turn LED on
18             time.sleep(0.5) #wait
19             GPIO.output(led_pin, False) #turn LED off
20             time.sleep(0.5) #wait
```

FIGURE 20 PYTHON CODE FOR CREATING A DIAMOND DETECTOR

5.2.5 Importing new maps and resource packages

Minecraft Pi offers the possibility to import new maps and resource packages to explore new worlds and enhance your gaming experience. Adding new maps is an easy procedure:

1. First you need to find a world that you like and download it to your Raspberry.
2. When you find one you like, you need to download it as a .zip file. To do that, simply click download and follow the steps.
3. When the file is downloaded, you need to extract the .zip file. Right click and then click extract.
4. In Raspberry Pi OS, open up the **filesystem**. Select **view** and then **show hidden files**.
5. Find the Minecraft **folder**, double click it, then double click again the folder named **“games”** and then double click again the folder named **“com.mojang”**. There you will find a folder named **“minecraftWorlds”**. Double click and you are ready to add new worlds. Simply drag and drop your unzipped worlds into the folder.



6. **Bonus tip:** while you are in the Minecraft worlds directory, you can rename the worlds you have already created in the game by simply **right click** on the them and then **rename**. This is the only way to rename your worlds.

You will find a huge variety of maps in the following links:

- <https://thebraithwaites.co.uk/minecraft-pi-edition-maps-texture-packs-survival-and-more/>
- <https://www.minecraftforum.net/forums/minecraft-pocket-edition/mcpe-maps?page=2>

When it comes to resource packages the procedure is similar as above, but a different directory is needed.

For example, you can download a texture pack from [here](#). Follow the instructions on the link. You need to find the folder “**Assets**” in the Minecraft directory and place the unzipped files there. Load the game again and the texture pack should be ready to use.

More resource packages along with maps can be found in the following links:

- <https://www.planetminecraft.com/collection/10099/resource-packs/>
- https://www.planetminecraft.com/texture_pack/the-pi-pack---132---037/
- <https://thebraithwaites.co.uk/minecraft-pi-edition-maps-texture-packs-survival-and-more/>
- <https://www.minecraftforum.net/forums/minecraft-pocket-edition/mcpe-maps?page=2>

5.3 Practical examples

For testing your knowledge and practicing further we propose some practical examples. These are recommended to be done after each tutorial and include the following:

- Example 1: Trap player between blocks
- Example 2: Build a house
- Example 3: Build a house with a twist
- Example 4: Big block of blocks
- Example 5: Create a volcano (advanced)

5.3.1 Example 1: Trap player between blocks!

The code below gets the players tile position, it then calls the Minecraft API's `getBlock()` function to find out the type of block the player is standing on (by subtracting 1 from the y co-ordinate) before using `setBlock()` to create blocks of the same type the player is standing on around him (<https://www.stuffaboutcode.com/2013/04/minecraft-pi-edition-api-tutorial.html>).

For example, if your player is standing on STONE, then STONE blocks will appear around him.

trap.py *
✖

```
1 from mcpi.minecraft import minecraft
2 from mcpi.block import block
3 from time import sleep
4
5 mc = Minecraft.create()
6
7 playerTilePos = mc.player.getTilePos()
8 blockBelowPlayerType = mc.getBlock(playerTilePos.x, playerTilePos.y - 1, playerTilePos.z)
9 mc.setBlock(playerTilePos.x + 1, playerTilePos.y + 1, playerTilePos.z, blockBelowPlayerType)
10 mc.setBlock(playerTilePos.x, playerTilePos.y + 1, playerTilePos.z + 1, blockBelowPlayerType)
11 mc.setBlock(playerTilePos.x - 1, playerTilePos.y + 1, playerTilePos.z, blockBelowPlayerType)
12 mc.setBlock(playerTilePos.x, playerTilePos.y + 1, playerTilePos.z - 1, blockBelowPlayerType)
13 mc.postToChat("Trapped you")
14 time.sleep(5)
```

Use the code above, along with what you learnt so far and write your script. Then run the script in Python programming interface while running Minecraft and see what happens. Is your player trapped between block of the same type as the one standing on? If not, what is wrong? Can you find the mistake? If everything is working according to the description, then the next thing to do is to free your player. Try to do this by yourself!

**Hint: removing blocks is done using `setBlock()`, but rather than making the block solid like WOOD or STONE we set it to AIR.*

5.3.2 Example 2: Build a house!

You want to build a house, but do not want to spend hours building blocks one by one. The following code will create a simple house-looking building from scratch. The code is pretty basic and explained below. Various comment lines are inserted for convenience.



house.py *

```
1 from mcpi.minecraft import minecraft
2 from mcpi.block import block
3 from time import sleep
4
5 mc = Minecraft.create()
6
7 x, y, z = mc.player.getPos()
8
9 # build walls
10 mc.setBlocks(x+2, y-1, z+2, x+7, y+3, z+8, 5)
11 # remove interior
12 mc.setBlocks(x+3, y, z+3, x+6, y+2, z+7, 0)
13 # make doorway
14 mc.setBlocks(x+2, y, z+5, x+2, y+1, z+5, 0)
15 # make window 1
16 mc.setBlocks(x+4, y+1, z+8, x+5, y+1, z+8, 102)
17 # make window 2
18 mc.setBlocks(x+4, y+1, z+2, x+5, y+1, z+2, 102)
```

Pretty simple, right? Use the code above, along with what you learnt so far and write your script. Then run the script in Python programming interface while running Minecraft and see what happens. What about adding a second floor and a roof? Use your imagination and programming skills along with the `mc.setBlocks()` command to alter the code and see what happens!

5.3.3 Example 3: Build a house with a twist!

You learnt to build a house by running your script, but what about building as many houses as you want and wherever you want at the click of the button?

You can assign an arcade-gaming button to build houses every time you hit it. How?

Firstly, you have to connect a button to the GPIO board as you learnt (*make sure you connect to the right pins on the GPIO board. This example uses GPIO24 as an input pin, but you can use whichever best suits your needs and change the code accordingly*).

Then, follow the script below (Various comment lines are inserted for convenience.):



```
house_twist.py *  
1 #connect Python with Raspberry Pi GPIO board  
2 import RPi.GPIO as GPIO  
3  
4 from mcpi.minecraft import minecraft  
5 from mcpi.block import block  
6 from time import sleep  
7  
8 mc = Minecraft.create()  
9  
10 GPIO.setmode(GPIO.BCM) #set mode for GPIO  
11 GPIO.setup(24, GPIO.IN) #set input pin to GPIO24  
12  
13 while True:  
14     if GPIO.input(24) == True:  
15         x,y,z = mc.player.getPos() #get players position  
16         mc.setBlocks(x+2,y-1,z+2,x+7,y+3,z+8, 5) #make shell  
17         mc.setBlocks(x+3,y,z+3,x+6,y+2,z+7, 0) #remove inside  
18         mc.setBlocks(x+2,y,z+5,x+2,y+1,z+5, 0) #make doorway  
19         mc.setBlocks(x+4,y+1,z+8,x+5,y+1,z+8, 102) #make window 1  
20         mc.setBlocks(x+4,y+1,z+2,x+5,y+1,z+2, 102) #make window 2  
21         mc.setBlocks(x+7,y+1,z+4,x+7,y+1,z+6, 102) #make window 3  
22         print ("House is built")  
23         time.sleep(0.1) #wait 0.1 sec
```

Run your script and then hit the assigned button. A house should appear next to you. Want to build more? Hit the button multiple times as you move in the Minecraft world. Use your imagination and programming skills and edit the `mc.setBlocks()` coordinates to add more floors, extend the area of the structure, create more doors and windows, etc. Can you program a button to build a block of buildings?

5.3.4 Example 4: Big Block of blocks

Another exercise using the `mc.setBlocks()` command. You can create massive blocks of a certain block type for further use. Try using the following code in your script and see what happens when you run your program.

```
tnt = 46  
mc.setBlocks(x+1, y+1, z+1, x+11, y+11, z+11, 46, 1)
```

Use the code above, along with what you learnt so far and write your script. Then run the script in Python programming interface while running Minecraft and see what happens. Does it work? If not, what is wrong? If everything is working according to the description, then the next thing to do is change the blocks types and shapes and build your unique structures. Try to do this by yourself!

5.3.5 Example 5: Create a Volcano! (advanced)

The following code will show you how to create an active volcano in the Minecraft world from scratch. Because this exercise is advanced compared to the previous, the whole code is given below (<https://learnlearn.uk/raspberrypi/2018/02/10/minecraft-python-volcano-tutorial/>). Various comment lines are inserted for convenience.

```
volcano.py*
1 # import Minecraft
2 import mcpi.minecraft as Minecraft
3 # import block library
4 import mcpi.block as block
5 # import random and time modules
6 import random, time
7
8 mc = Minecraft.create()
9 mc.postToChat("Minecraft Volcano!")
10
11 mc.setBlocks(-100, 0, -100, 100, 50, 100, block.AIR)
12
13 height = 20
14 center = 20,0,0 #x,y,z
15
16 for i in range(height): # Create the base!
17     size = height - i
18     mc.setBlocks(center[0] - size, center[1] + i, center[2] - size, center[0] + size, center[1] + i, center[2] + size, block.STONE)
19
20 for i in range(height * height):
21     randomx = random.randint(center[0] - height, center[0] + height)
22     randomz = random.randint(center[2] - height, center[2] + height)
23     mc.setBlock(randomx, center[1] + height + 10, randomz, block.GRAVEL)
24
25 while True:
26     mc.setBlock(center[0], center[1] + height, center[2], block.LAVA_FLOWING)
27     time.sleep(1)
```

Use the code above, along with what you learnt so far and write your script. Then run the script in Python programming interface while running Minecraft and see what happens. Does it work? If not, what is wrong? If everything is working according to the description, then the next thing to do is to make your volcano erupt water instead of lava. If that was too easy, try creating a circular volcano instead of a square one.

5.4 Assessment test

1. Minecraft Pi support full features of the Minecraft game.
 1. Yes
 2. **No**
2. I can run Minecraft by:
 1. Double clicking the desktop icon
 2. Navigating to the main menu
 3. Using the command line to run the program
 4. **All of the above**
3. Minecraft Pi has an API to control the game using Python programming interface and a number of scripts.
 1. **Yes**
 2. No
4. A block in Minecraft is 1m³.
 1. Yes
 2. No
 3. **Not always**
5. What does the following line of code do: *from mcpi.minecraft import Minecraft*:
 1. Code is wrong
 2. **Connects Python programming interface to Minecraft**



3. Changes camera angle in game
4. None of the above
6. We can teleport our player by using the `setPos()` command.
 1. **Yes**
 2. No
7. What does the following line of code mean, `gold = 41`:
 1. We set the price of an item at 41 units of gold
 2. We set 41 blocks of gold to be used in game
 3. **41 represents the block ID of gold saved in a variable named gold**
 4. None of the above
8. What does the last digit mean in the following line of code, `mc.setBlock(x, y, z, wool, 2)`:
 1. **Extra property of block type wool**
 2. We request 2 blocks of wool
 3. Represents some kind of coordinates
 4. None of the above
9. The line `GPIO.setup(23, GPIO.IN)` tells the Pi that this pin 23 is used as an input.
 1. **Yes**
 2. No
10. To create a circuit for using a button with my Raspberry Pi, I need a breadboard, two jumper cables and the button.
 1. Yes
 2. **No**

5.5 References

- Richardson, C., (2013), Minecraft Pi book, retrieved from <https://arghbox.files.wordpress.com/2013/06/minecraftbook.pdf>
- Minecraft controls, retrieved from <https://arghbox.wordpress.com/2013/07/28/minecraft-pi-controls/>
- Block ID numbers, retrieved from <https://www.raspberrypi-spy.co.uk/2014/09/raspberry-pi-minecraft-block-id-number-reference/>
- O'Hanlon, (2013), Minecraft: Pi Edition - API Tutorial, retrieved from <https://www.stuffaboutcode.com/2013/04/minecraft-pi-edition-api-tutorial.html>
- Special blocks in Minecraft, retrieved from <https://minecraft.gamepedia.com/Block>
- Minecraft Wiki, retrieved from https://minecraft.gamepedia.com/Pi_Edition

5.6 Additional resources

If you want to go into more details about Minecraft Pi, please consider visiting the following resources:

- Minecraft API: <https://www.stuffaboutcode.com/p/minecraft-api-reference.html>
- Raspberry Pi: <https://www.stuffaboutcode.com/p/raspberry-pi.html>
- Minecraft Wiki: https://minecraft.gamepedia.com/Pi_Edition



- Manhattan project in Minecraft:
<https://www.stuffaboutcode.com/2013/04/minecraft-pi-edition-manhattan-stroll.html>
- Massive Analogue Clock: <https://www.stuffaboutcode.com/2013/02/raspberry-pi-minecraft-analogue-clock.html>
- Planetary gravity simulation: <https://www.stuffaboutcode.com/2013/03/raspberry-pi-minecraft-planetary.html>
- Coding Tips: <http://www.laschina.org/wp-content/uploads/2017/09/Minecraft-Coding-Tips.pdf>
- Raspberry Pi projects:
<https://projects.raspberrypi.org/en/projects?software%5B%5D=python&hardware%5B%5D=raspberry-pi>
- Minecraft Pi Handbook:
<http://repository.erasmusplus.website/RETROSTEM/Material/Minecraft%20Pi%20-%20Handbook.pdf>

5.7 Conclusion

Key conclusion points for **Module 3 – Introduction to Minecraft Pi**:

If you followed this resource with your Raspberry Pi, you are expected to:

- Access Minecraft Pi and create a new world.
- Navigate around Minecraft Pi using the movement controls on your keyboard.
- Know how to place and destroy a block, and navigate through different types of blocks in the in-game inventory.
- Connect Python to Minecraft Pi.
- Use Python programming interface.
- Manipulate blocks using Python code and scripts.
- Have a good understanding of the rest of Minecraft Pi functions.
- Make Minecraft interact with the outside world with the use of buttons and LEDs.
- Import new worlds and discover resource packages compatible with Minecraft Pi.

6 Raspberry Pi GPIO programming using Python [P2-AKNOW]

6.1 Glossary of terms

Term / Concept	Definition / Explanation
Raspberry Pi GPIO	Raspberry Pi GPIO is the row of pins along the top edge of the board. A 40-pin header is found on all current Raspberry Pi boards. Most of the functionality of the Raspberry Pi comes from these pins which can be configured and controlled using a programming language. Any of the GPIO pins can be designated in software as an input or output pin and used for a wide range of purposes such as to control LEDs, buzzers, motors, servos, to interact with sensors, to communicate with other devices, etc.
Python	Python is an interpreted, object-oriented, high-level programming language. Python has simple, easy to learn syntax that emphasizes readability and therefore reduces the overall time needed to learn it and to develop and maintain a program. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

6.2 Main content

Module 4 – Raspberry Pi GPIO programming using Python offers an introduction of the use of the General-Purpose Input Output (GPIO) pins on your Raspberry Pi computer. The GPIO is a powerful feature of the Raspberry Pi and can be found on the top edge of the board. A 40-pin GPIO header comes with all current Raspberry Pi computers and most of its functionality comes from these pins. After completing module 4, the student is expected to know how to control and interact with the Raspberry Pi GPIO using the Python programming language.

Module 4 consists of the following elements:

- Introduction to Raspberry Pi GPIO
- Introduction to Python programming language
- Introduction to electronic circuits:
 - Blink an LED using the Raspberry Pi GPIO pins and Python
 - Activate a buzzer with a push button
 - Connect and use a sensor via the GPIO
 - Create a proximity alarm or stop-light.
- Practical examples of using the GPIO with Python programming
- Assessment quiz to test acquired knowledge
- Extra resources to advance your knowledge even further

6.2.1 Introduction to Raspberry Pi GPIO pins

The Raspberry Pi GPIO is the row of pins along the top edge of the board. A 40-pin header is found on all current Raspberry Pi models. Most of Raspberry Pi's functionality comes from the GPIO which can be configured and controlled using a programming language. Figure 1 shows where the GPIO pins are located on a Raspberry Pi.

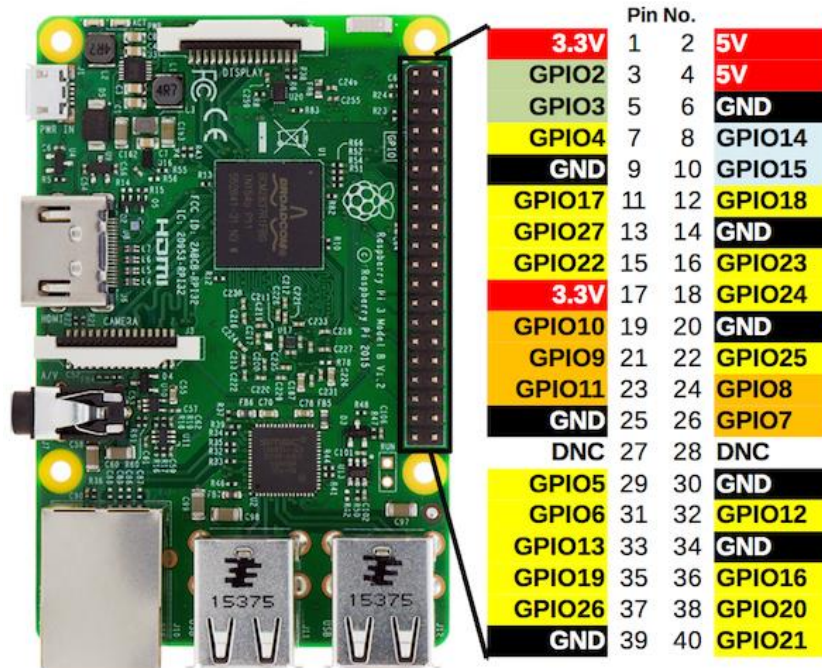


FIGURE 21 RASPBERRY PI GPIO PINS

Any of the GPIO pins can be designated in software as an input or output pin and used for a wide range of purposes such as to control LEDs, buzzers, motors, servos, to interact with sensors, to communicate with other devices, etc.

The user needs to familiarize with the GPIO pins, which of the pins can be used, their naming and numbering and what properties they have. Figure 2 shows the standard Broadcom (BCM) pin names. Note that the numbering of the GPIO pins is not in numerical order.

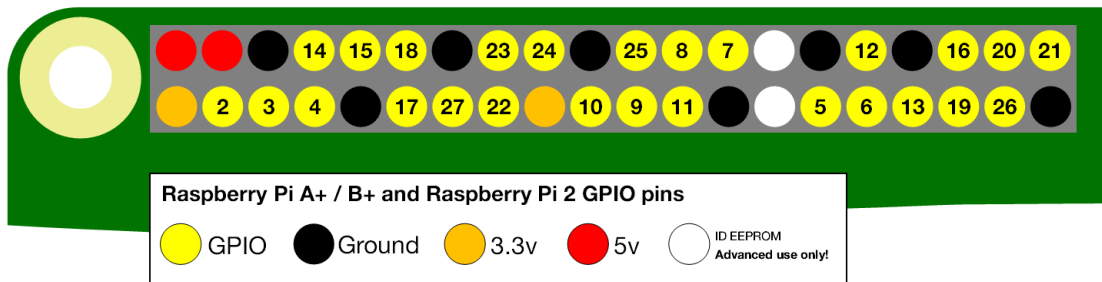


FIGURE 22 STANDARD BROADCOM (BCM) PIN NAMES

To compare these to your Raspberry Pi, orient the pins so that they are at the top left side of the board, as shown in Figure 3.

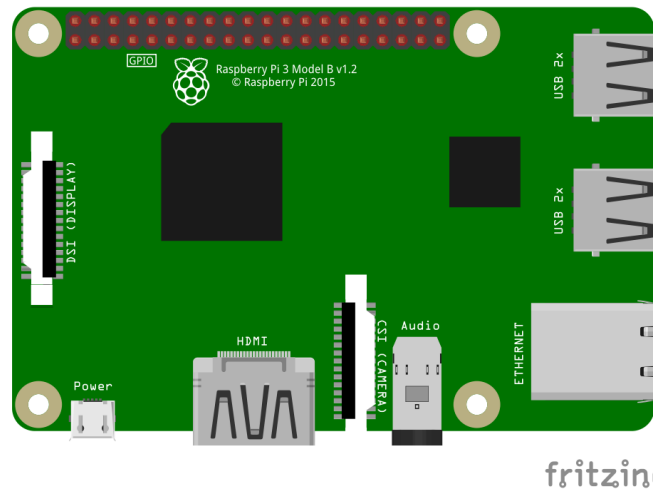


FIGURE 23 SCHEMATIC TOP VIEW OF THE RASPBERRY PI COMPUTER

All those pins coloured in yellow are the typical GPIO pins. The black ones are the ground pins and the rest are the voltage pins at 5V or 3.3V. The order of pins never changes, no matter which version of the Raspberry Pi model B you have, starting from the top down.

Voltage and Ground pins: Two 5V pins and two 3.3V pins are present on the board, as well as a number of ground pins (0V), which are unconfigurable. The Ground pins should be considered as one of the most important pins because any circuit created should be connected to one of them, otherwise it may damage the board or have unexpected behaviour. The remaining pins are all general purpose 3.3V pins, meaning outputs are set to 3.3V and inputs are expected to be at 3.3V.

Outputs: A GPIO pin designated as an output pin can be set to high (3.3V) or low (0V).

Inputs: A GPIO pin designated as an input pin can be read as high (3.3V) or low (0V).

The GPIO pins have basically two states: HIGH or LOW, and by combining these binary choices, many more outcomes can be created with circuits to interact with the physical world with programs. In a following chapter high and low signals will be used to make the Raspberry Pi interact with electronic components and sensors.

It is important to distinguish the GPIO pins. Some people use pin labels, for example the printable [Raspberry Leaf](#) (Figure 4).

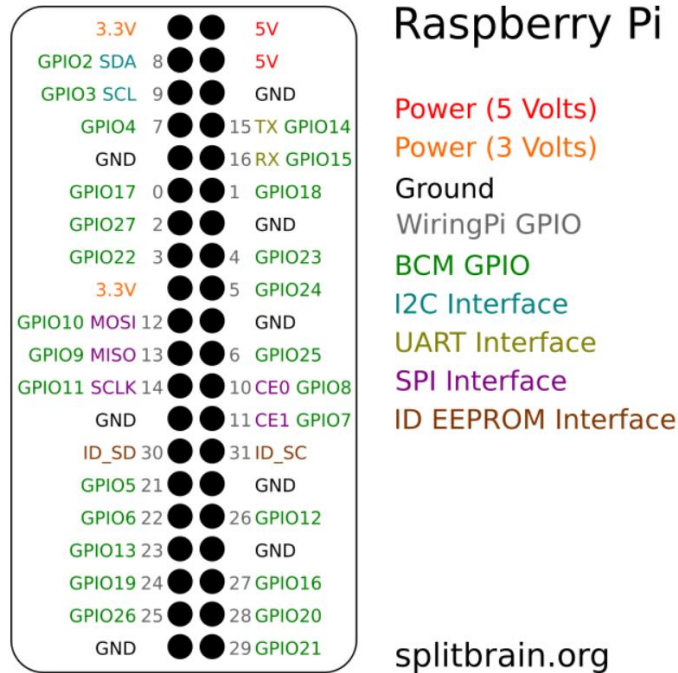


FIGURE 24 RASPBERRY PI B+ LEAF

Moreover, a handy reference can be accessed on the Raspberry Pi by opening a terminal window and running the `pinout` command (Figure 5). This tool is pre-installed on the Raspberry Pi OS desktop image and it is provided by the GPIO Zero Python library.



FIGURE 25 RASPBERRY PI PINOUT TOOL

It is possible to control the GPIO pins using a number of programming languages and tools. In Module 4, you will learn to use the Python programming language to interact with your Raspberry Pi's GPIO.

6.2.2 Introduction to Python programming language

Python is a general-purpose object-oriented programming language that is easy to learn, read and write, and with Raspberry Pi, it lets you connect your program with the physical world. Python has a very clean syntax that is easy to read and uses standard English keywords.



FIGURE 26 PYTHON LOGO

Raspberry Pi has a pre-installed Python 3 development environment that is called Thonny. You can open Thonny from the application menu of your Raspberry Pi desktop. Thonny is equipped with a REPL (Read-Evaluate-Print-Loop) which is a prompt you can enter Python command into. Thonny has a syntax highlighting built-in and some autocompletion support which makes your life easier when typing code.

Python supports the following:

- Print
- Indentation
- Variables
- Comments
- Lists
- Iterations
- Range
- Length
- If statements

```
print("Hello world")
```

FIGURE 27 SIMPLE EXAMPLE OF PYTHON SYNTAX

For detailed information about how each of the above is used can be found by following these links:

1. <https://www.raspberrypi.org/documentation/usage/python/README.md>
2. <https://www.raspberrypi.org/documentation/usage/gpio/python/README.md>

6.2.3 Introduction to electronic circuits

A series of tutorials are presented in this section. Besides the Raspberry Pi, the following components are required:

- 10 x Male-to-Female jumper wires
- 1 x Breadboard
- 3 x LED bulbs

- 6 x Resistors (between 300 and 1K Ohm). We will need 3x 1K Ohms for the distance sensor, and one 300 to 1K Ohm resistance per LED bulb
- 1 x HC-SR04 Ultrasonic Distance Sensor
- 1 x buzzer

While connecting simple components to the GPIO pins is safe, it is important to be careful how you wire things up. LEDs should have resistors to limit the current passing through them. Do not use 5V components for 3V3 components. Do not connect motors directly to the GPIO pins.

LED lights draw different amounts of power depending on the light, which may damage the Raspberry Pi if it is connected without a resistor to decrease the current. You may start with a resistor of large resistance and gradually decrease it until you are satisfied with the LED light's intensity. Most LEDs work just fine with 300-1K Ohm resistance. The more resistance you use, the dimmer the LED will be. If you connect an LED with a 300 Ohm resistance and you do not see any light, then either the bulb is bad, or it is not connected properly. Try another LED and check that it is correctly connected, i.e. the LED's longer lead is the positive end, and it should be connected to a GPIO pin, and the shorter lead is the negative end and should go to the ground of the circuit.

First, if you are not familiar with a breadboard, the idea is to help you to quickly test a circuit, do prototyping, as well as to help you learn about working with circuits without doing any soldering.

A typical breadboard looks something like:

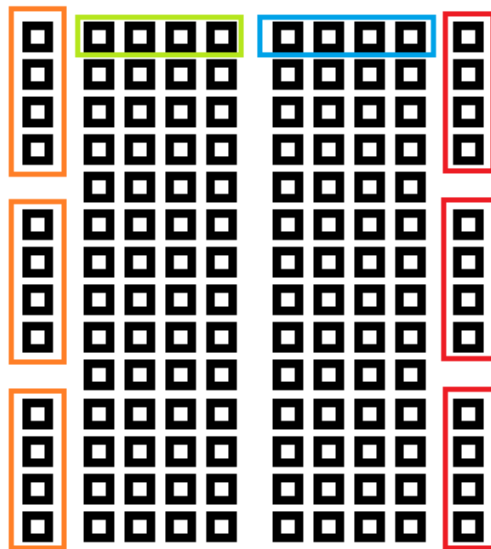
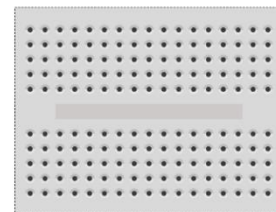
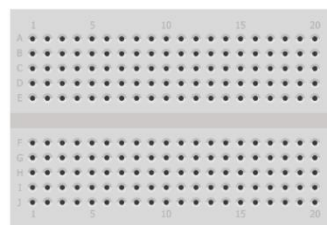
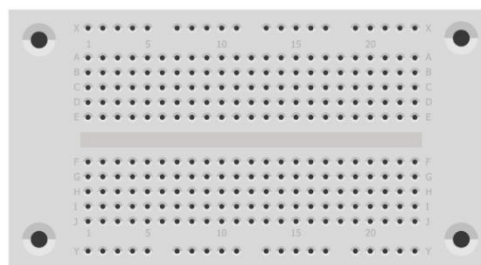
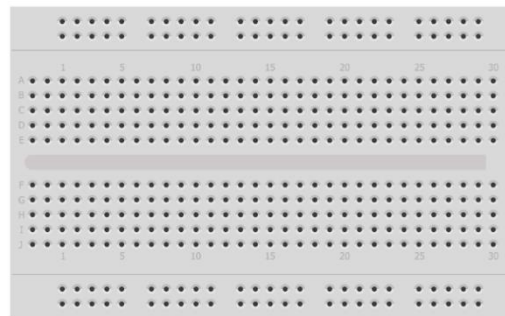
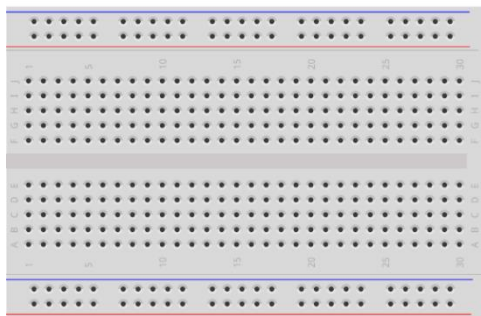
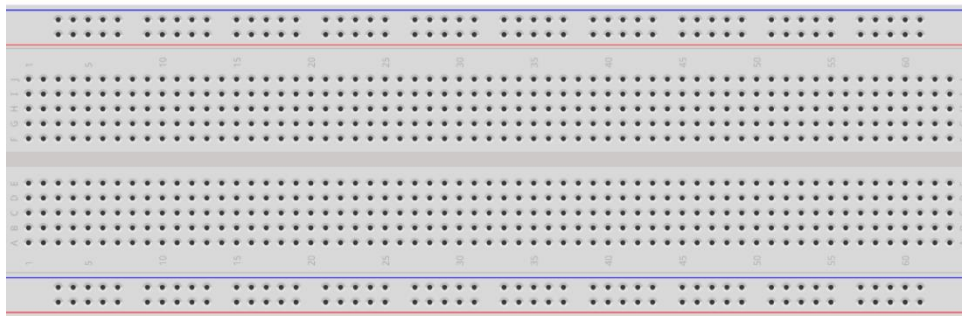
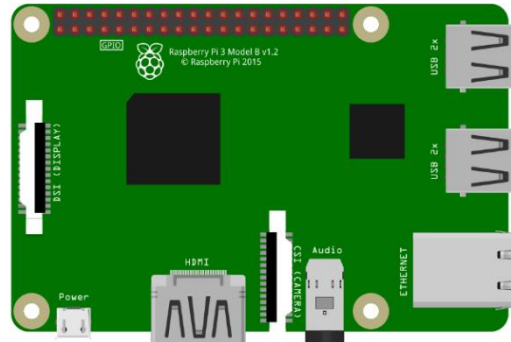


FIGURE 28 SCHEMATIC TOP VIEW OF A TYPICAL BREADBOARD

As shown in Figure 8, the rows along the edges are connected vertically, and the centre portions are connected horizontally. If there is a larger gap, like in the middle of this board, that's not connected. If you want these portions to be connected, you must connect them with a jumper wire.

There are breadboards of various sized that can be utilized depending on the size of the circuit. The most common are presented in Figure 9 along with a Raspberry Pi for size references.



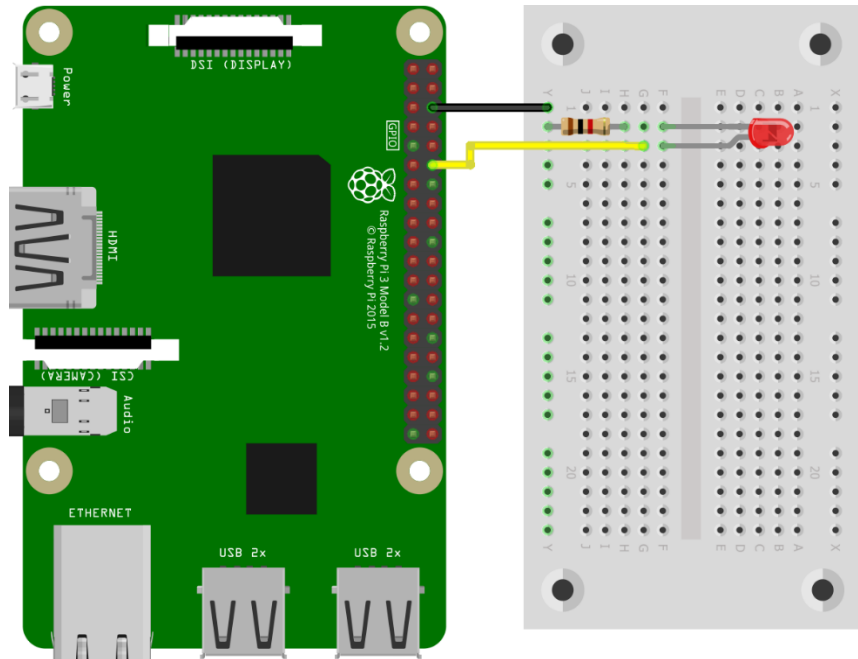
fritzing

FIGURE 29 SCHEMATIC VIEW OF VARIOUS BREADBOARDS

In addition to the components that were explained in the beginning of this section, you will also need a keyboard which you can connect to the USB ports of your Raspberry Pi.

6.2.3.1 Blink an LED

Before you proceed, turn off your Raspberry Pi and unplug it. For this circuit, you will need a breadboard, an LED, a resistor of 220 Ohm and two female-to-male jumper wires. The complete circuit is in the schematic diagram, depicted in Figure 10.



fritzing

FIGURE 30 SCHEMATIC OF LED CIRCUIT

Now you can turn your Raspberry Pi on, and you will begin to develop some simple code in Python. In most cases the GPIO Python library is already installed in the operating system of the Raspberry Pi. If not, then you need to install it. To do that, open a terminal and type the command line: `$ sudo apt-get install python-rpi.gpio`
 From the top-left desktop menu select Programming and then Thonny Python (see screenshots that follow).

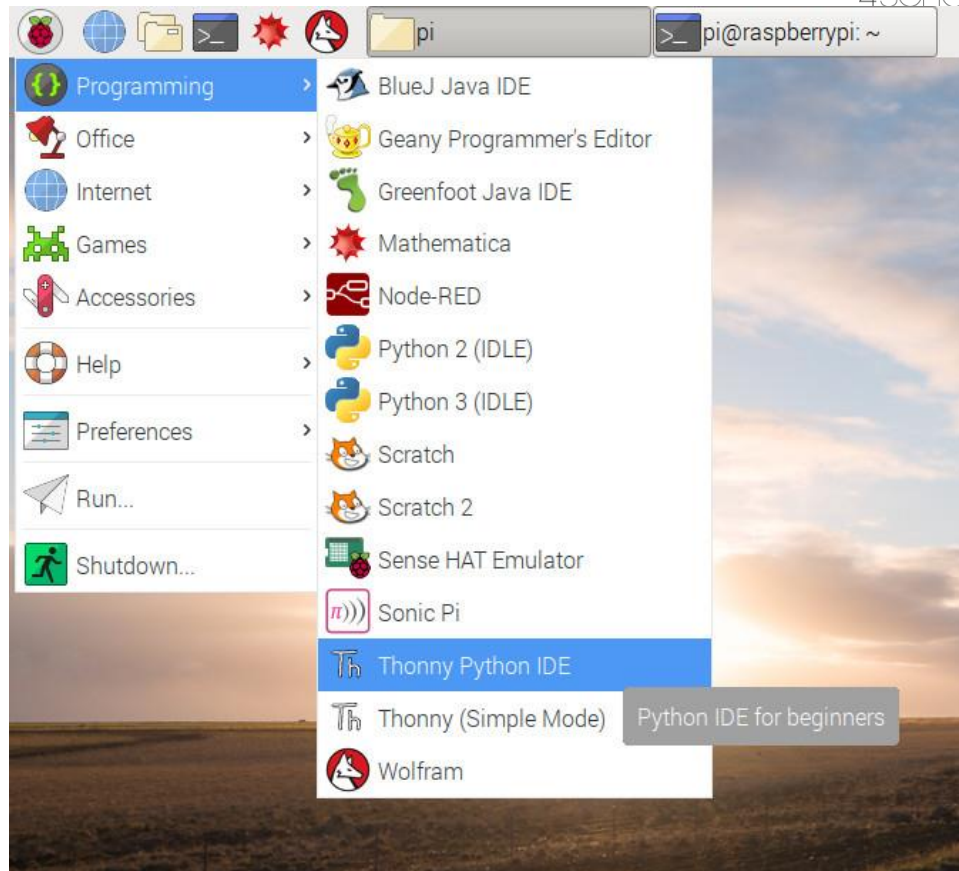


FIGURE 31 LOCATING THONNY PYTHON

You first need to create a new Python file by clicking **File → New File** and save it under the name `led.py`.

Then, you are ready to start typing the first lines of your program:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
```

The BCM numbering is the pin number that the Broadcom chip considers it to be. You will use the BCM numbering which is also shown in previous figures. It is useful to print this and to always have it somewhere handy, as you will need to check that your circuit connections are correct and that they correspond to the code you program.

Now, continuing your code script you set:

```
GPIO.setup(18, GPIO.OUT)
```

Here, you are setting up the pin to be a pin that outputs information. You can also set the pin to take in information instead.

```
GPIO.output(18, GPIO.HIGH)
time.sleep(3)
```

Here, you tell pin 18 to output a high signal, and then pause for 3 seconds.

```
GPIO.output(18, GPIO.LOW)
GPIO.cleanup()
```

Finally, you change the pin's output to a LOW signal, and then use `GPIO.cleanup()` to reset the pin's statuses to their default states before finishing with the program. This is done so the next program that runs comes into a state that it expects. It is quite easy to

forget and leave various pins on setup differently and only find out later that they act strangely.

The complete script is as follows. Various comment lines are inserted for convenience.

```
#####  
  
# first import libraries and set GPIO numbering mode  
import RPi.GPIO as GPIO  
import time  
GPIO.setmode(GPIO.BCM)  
  
# setup pin of LED as output and turn it on  
GPIO.setup(18, GPIO.OUT)  
GPIO.output(18, GPIO.HIGH)  
# sleep for 3 seconds and then turn off LED  
time.sleep(3)  
GPIO.output(18, GPIO.LOW)  
  
# cleanup gpio before exiting  
GPIO.cleanup()  
#####
```

You are now ready to run your program. First save your program and then click the F5 button to execute your script.

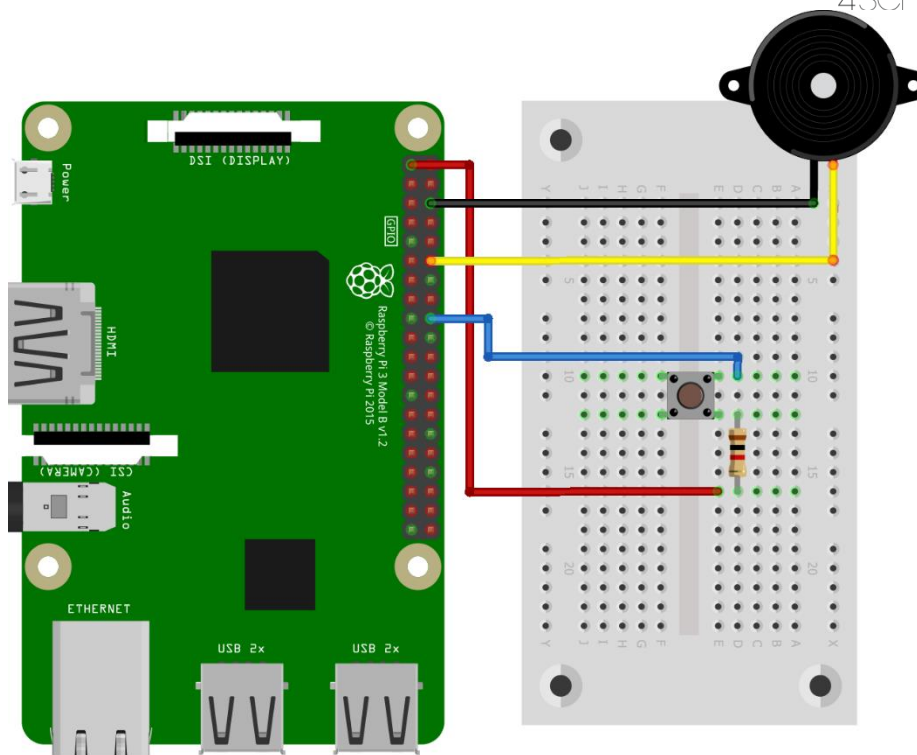
When the program runs, the LED should turn on for three seconds and then should turn off. You can try adding more LEDs in your circuit and code or add a buzzer to make a sound the LEDs are turned on. You may also try to practice simple Python constructs such as loops to make the LEDs blink several times.

6.2.3.2 Activate a buzzer

Before proceeding with this tutorial, you will first try to use a GPIO pin as input. You will need a push button and a circuit as shown in the schematic diagram below (Figure 11). In this case we will use a buzzer instead of an LED, which is connected to GPIO pin 18. In your program you will set this pin as output. You will also connect a push button to GPIO pin 24 which you will set up as input. The other end of the button should be connected to the 3.3V voltage pin of Raspberry Pi via a resistor. Thus, when you push the button then GPIO pin 24 will be in HIGH state, i.e. in 3.3V.

When you are done with your circuit, you can turn on the Raspberry Pi and start typing your program in a new Python file (`button.py`). The complete script is presented below along with several comment lines for explanations.

In your program you first import the Python modules that you need, then you set up the GPIO pins for buzzer and button as output and input, respectively. Then, just to check, you turn on the buzzer for 3 seconds. At the end, you start a loop that runs forever. In the loop you check if the input pin is in HIGH state. If yes, then this means that the button is pushed and so you want to set the output pin to be HIGH to turn on the buzzer. Otherwise the output pin is set to LOW, so the buzzer is turned off.



fritzing

FIGURE 32 SCHEMATIC DIAGRAM OF CIRCUIT WITH PUSH BUTTON AND BUZZER CONNECTED TO GPIO PINS

```
#####
# first import modules and GPIO numbering mode
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()
# pin for buzzer (or led)
pin_out = 18
GPIO.setup(pin_out, GPIO.OUT)
# pin for push button
pin_button = 24
GPIO.setup(pin_button, GPIO.IN)
# turn on buzzer for 3 sec
GPIO.output(pin_out, GPIO.HIGH)
time.sleep(3)
GPIO.output(pin_out, GPIO.LOW)
# do this loop forever! Press Ctrl-C to stop it
# When button is pressed then buzzer is on
while True:
    if GPIO.input(pin_button) == GPIO.HIGH:
        print("button pushed")
        GPIO.output(pin_out, GPIO.HIGH)
```

```

else :
    GPIO.output(pin_out, GPIO.LOW)
#####

```

6.2.3.3 Connect a sensor

In this tutorial, you are going to use a sensor, the HC-SR04 ultrasonic distance sensor, along with the GPIO input.

The HC-SR04 distance sensor measures distance based on emitting a sound burst, and timing how long it takes to receive the echo back. Using the known constant that is the speed of sound, we can mathematically determine the distance of any object in front of this sensor by simply measuring how much time passed as the sound waves were emitted, hit the object in front of the sensor, bounced back, and came back to the sensor.

Before you proceed, turn off your Raspberry Pi and unplug it. For the circuit of this tutorial you will need:

- Breadboard
- 4x Male-to-Female jumper wire cables
- 1x 1K Ohm and 1x 2K Ohm resistor, or 3x 1K Ohm resistors
- 1x HC-SR04 ultrasonic distance sensor

The distance sensor comes with 4 pins: power (VCC), trigger (TRIG), echo (ECHO), and ground (GND). The power pin will be connected to the Raspberry Pi's 5V pin, trigger will be assigned to a GPIO pin as output, echo will be assigned to a GPIO pin as input, and ground will be connected to a ground pin on the Raspberry Pi.

Figure 12 shows the complete setup.

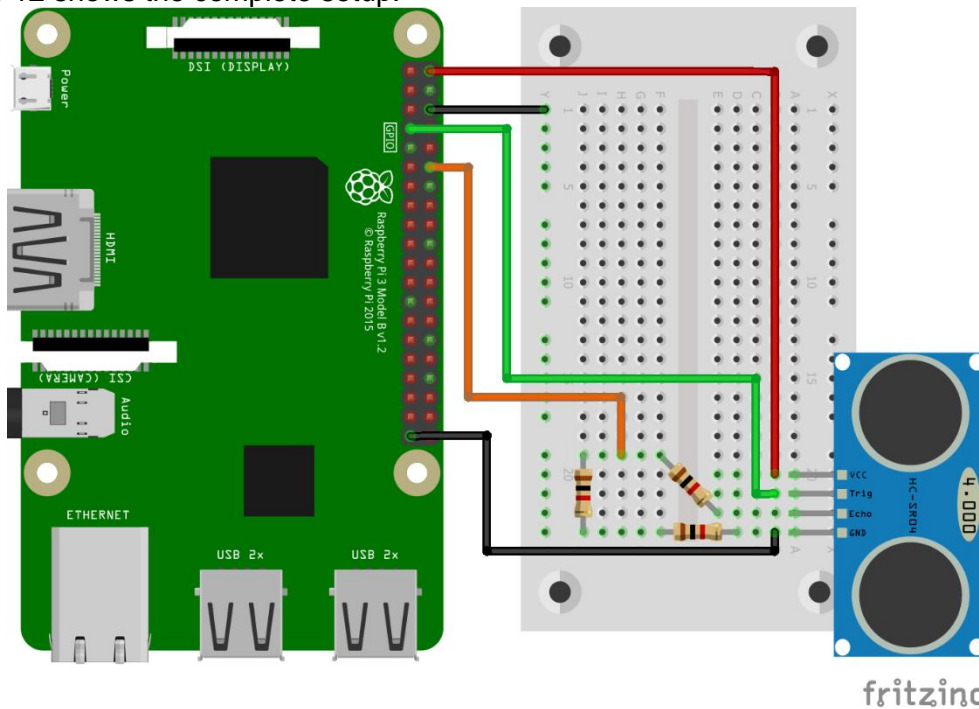


FIGURE 33 SCHEMATIC DIAGRAM OF CIRCUIT WITH DISTANCE SENSOR CONNECTED TO GPIO PINS

Now you can turn on the Raspberry Pi, create a new Python file and start the new program (`sensor.py`).

As in the previous tutorial you first import the same libraries and initial mode.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
```

Then you define TRIG and ECHO pins and set them up as output and input respectively:

```
TRIG = 4
ECHO = 18
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
```

Here, you are defining TRIG and ECHO as the Broadcom pin numbers that you intend to use for that part of the sensor. This is done because you need to reference both pins multiple times.

Then with the following, you emit a burst of ultrasound from the sensor:

```
GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False)
```

You need to measure the time it takes for the burst's echo to be detected back by the sensor:

```
while GPIO.input(ECHO) == False:
    tstart = time.time()
```

After that you issue a signal out and then measure the time interval:

```
while GPIO.input(ECHO) == True:
    tend = time.time()
sig_time = tend-tstart
```

When you finally do get an input time, you can subtract the end time from the start time and calculate distance by multiplying by the speed of sound. The speed of sound in dry air at 20 °C is 343.26 m/sec.

Note also that you need to multiply by $\frac{1}{2}$ because the sound burst actually travels the distance to object or obstacle twice (i.e. it is emitted by the sensor, travels until is bounced on an object and travels back to sensor to be detected):

```
distance = sig_time * 34326.* 0.5
print("signal(sec), distance(cm):", sig_time, distance)
GPIO.cleanup()
```

You can print the distance, and then clean up the pins.

Your output should be something like: `signal(sec), distance(cm): 0.001 17.163`

Do note that, after about 20 degrees of angle, your distance might be very skewed since the initial bounce might miss and it might take the burst several bounces off things to come back to the sensor.

The complete code script for this tutorial is as follows. Various comment lines are inserted for convenience and clarity.

```
#####
# first import libraries and set GPIO numbering mode
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
```



```
# define TRIG and ECHO pins and set them up as output and input
TRIG = 4
ECHO = 18
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)
# emit a burst of ultrasound
GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False)

# measure time interval
while GPIO.input(ECHO) == False:
    tstart = time.time()
while GPIO.input(ECHO) == True:
    tend = time.time()
sig_time = tend-tstart

# calculate distance and print value
distance = sig_time * 34326.* 0.5
print("signal(sec), distance(cm):", sig_time, distance)

# cleanup gpio before exiting
GPIO.cleanup()
#####
```

You may continue by combining circuits and codes from this tutorial and the previous one in order to add one or more LEDs (e.g. see Figure 13 below).

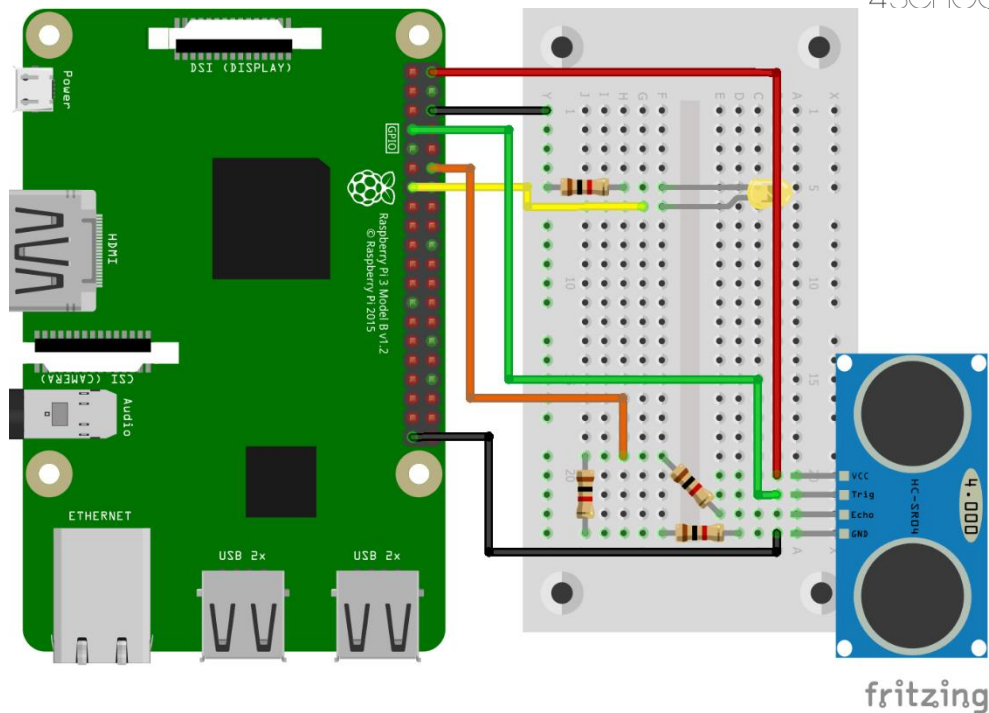


FIGURE 34 SCHEMATIC DIAGRAM OF CIRCUIT WITH LED AND DISTANCE SENSOR CONNECTED TO GPIO PINS

6.2.3.4 Create a proximity alarm

In this tutorial, we will put together what we have learned so far to create a proximity alarm or stop-light, like the ones that cars have in order to assist drivers when they are parking their cars.

The main idea of a proximity alarm or stop-light is to show green when there is plenty of room, turn yellow as distance is increased, and then red when a minimum distance is reached, i.e. the vehicle should stop. We're going to build this system by combining the circuits and the code we developed in our previous tutorials.

First, we just leave the distance sensor circuit as is and add the circuits for the three LED lights. Figure 14 shows the complete setup.

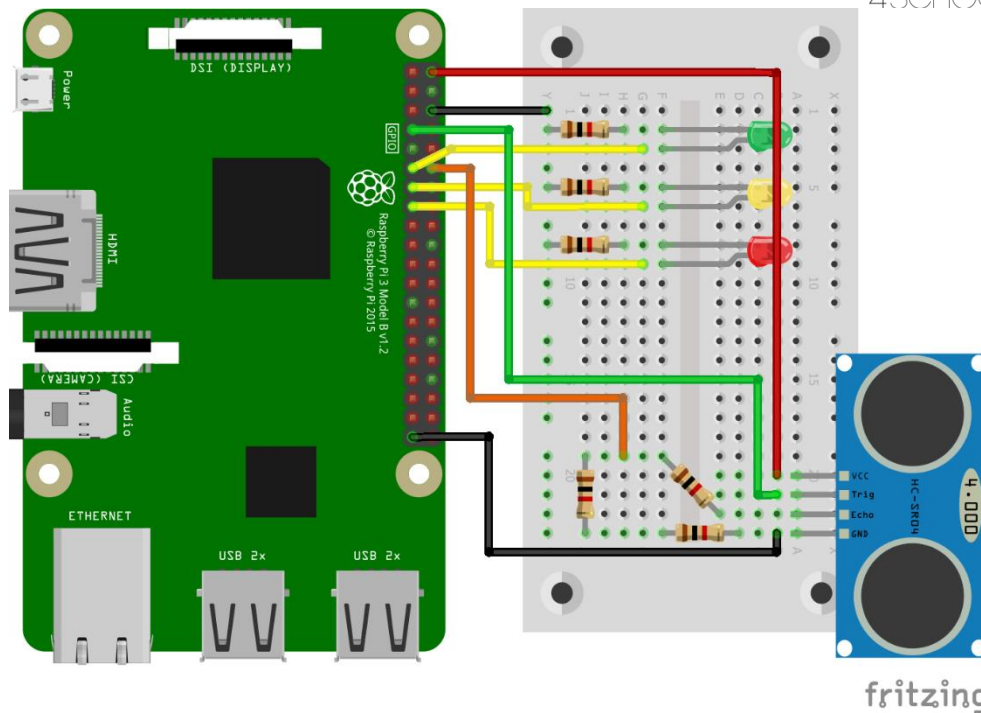


FIGURE 35 SCHEMATIC DIAGRAM OF CIRCUIT WITH DISTANCE SENSOR AND MULTIPLE LEDs CONNECTED TO GPIO PINS

Once everything is connected, it is recommended to double-check it before you power on your Raspberry Pi. Next, you need to modify your previous distance sensor script and introduce a continuous loop and turn certain parts of the code into functions. You can start by copy-pasting the existing code from the previous tutorial in a new Python file (proximity_alarm.py). The new code will look like the following:

```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
# doing this first, since using a while True.
GPIO.cleanup()
GPIO.setmode(GPIO.BCM)
TRIG = 4
ECHO = 18
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)

def get_distance():
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)
    while GPIO.input(ECHO) == False:
        tstart = time.time()
    while GPIO.input(ECHO) == True:
        tend = time.time()
    sig_time = tend-tstart
```




```
# Distance in cm
distance = sig_time * 34326.* 0.5
#print("signal(sec), distance(cm):", sig_time, distance)
return distance

while True:
    distance = get_distance()
    time.sleep(0.05)
```

Notice that you have put most of the code into function `get_distance()` and then call it in a while loop that runs forever. Also note that you need to put some sleep time between each running of the function, otherwise the sensor may not behave normally. If you do not let it sleep for a moment, it will just get overwhelmed and shutdown. Now, you just need some simple code that turns on the LED lights. For example, a green light, assuming you plugged the power for the green light into BCM pin 17:

```
GREEN = 17
GPIO.setup(GREEN,GPIO.OUT)
def green_light():
    GPIO.output(GREEN, GPIO.HIGH)
```

You could just copy and paste this for the other LEDs, but notice that nothing is turning them off if we change lights, so in `green_light()` you need something like:

```
def green_light():
    GPIO.output(GREEN, GPIO.HIGH)
    GPIO.output(YELLOW, GPIO.LOW)
    GPIO.output(RED, GPIO.LOW)
```

Now you can just do some copying and pasting and do this for all lights:

```
GREEN = 17
YELLOW = 27
RED = 22
GPIO.setup(GREEN,GPIO.OUT)
GPIO.setup(YELLOW,GPIO.OUT)
GPIO.setup(RED,GPIO.OUT)
def green_light():
    GPIO.output(GREEN, GPIO.HIGH)
    GPIO.output(YELLOW, GPIO.LOW)
    GPIO.output(RED, GPIO.LOW)
def yellow_light():
    GPIO.output(GREEN, GPIO.LOW)
    GPIO.output(YELLOW, GPIO.HIGH)
    GPIO.output(RED, GPIO.LOW)
def red_light():
    GPIO.output(GREEN, GPIO.LOW)
    GPIO.output(YELLOW, GPIO.LOW)
    GPIO.output(RED, GPIO.HIGH)
```

Then, your while loop could look something like:

```
while True:
    distance = get_distance()
```

```
time.sleep(0.05)
print(distance)
if distance >= 30:
    green_light()
elif 30 > distance > 10:
    yellow_light()
elif distance <= 10:
    red_light()
```

So, if the distance is greater than or equal to 30 cm, a green light is showed. If it is between 10 and 30 cm, yellow light is shown, and then a red light is shown for less than or equal to 10 cm.

The complete code script is as follows. Various comment lines are inserted for convenience and clarity.

```
#####
# first import libraries and set GPIO numbering mode
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
# doing this first, since we're using a while True.
GPIO.cleanup()
GPIO.setmode(GPIO.BCM)

# define TRIG and ECHO pins and set them up as output and input
TRIG = 4
ECHO = 18
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)
# function to get distance value
def get_distance():
    # emit a burst of ultrasound
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)
    # measure time interval
    while GPIO.input(ECHO) == False:
        tstart = time.time()
    while GPIO.input(ECHO) == True:
        tend = time.time()
    sig_time = tend-tstart
    # Distance in cm
    distance = sig_time * 34326.* 0.5
    #print("signal(sec), distance(cm):", sig_time, distance)

    return distance

# define pins for LEDs and setup them as outputs
GREEN = 17
```

```
YELLOW = 27
RED = 22
GPIO.setup(GREEN,GPIO.OUT)
GPIO.setup(YELLOW,GPIO.OUT)
GPIO.setup(RED,GPIO.OUT)
# function to turn on green, turn off yellow and red
def green_light():
    GPIO.output(GREEN, GPIO.HIGH)
    GPIO.output(YELLOW, GPIO.LOW)
    GPIO.output(RED, GPIO.LOW)
# function to turn on yellow, turn off green and red def
yellow_light():
    GPIO.output(GREEN, GPIO.LOW)
    GPIO.output(YELLOW, GPIO.HIGH)
    GPIO.output(RED, GPIO.LOW)
# function to turn on red, turn off yellow and green
def red_light():
    GPIO.output(GREEN, GPIO.LOW)
    GPIO.output(YELLOW, GPIO.LOW)
    GPIO.output(RED, GPIO.HIGH)

# loop that runs forever
while True:
    distance = get_distance()
    time.sleep(0.05)
    print(distance)
    # check distance and turn on light accordingly
    if distance >= 30:
        green_light()
    elif 30 > distance > 10:
        yellow_light()
    elif distance <= 10:
        red_light()
#####
```

When you run this program, your device in operation will indicate lots of space (with green light), getting closer (yellow light), stop (red light).

6.3 Practical examples

For testing your knowledge and practicing further we propose some practical examples. These are recommended to be done after each tutorial and include the following:

- Example 1: Use a Buzzer and Multiple LEDs with Raspberry Pi GPIO Pins
- Example 2: Measure the Speed of Sound
- Example 3: Make a Proximity Alarm with Light and Sound

6.3.1 Example 1: Use a Buzzer and Multiple LEDs with Raspberry Pi GPIO Pins

In this exercise, try to modify the circuit and code you developed in the first tutorial in order to connect and use multiple LEDs, and to program them to blink with some pattern (e.g. when one is on the other is off, or blinking in different time intervals). For this exercise, make a circuit as shown in Figure 15. You can also connect a buzzer to make some sound.

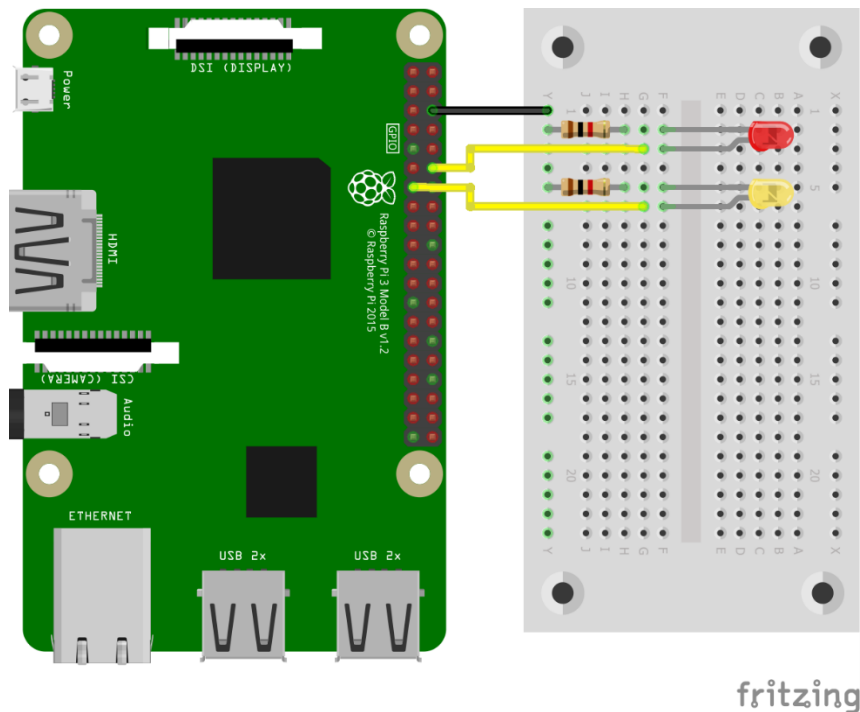


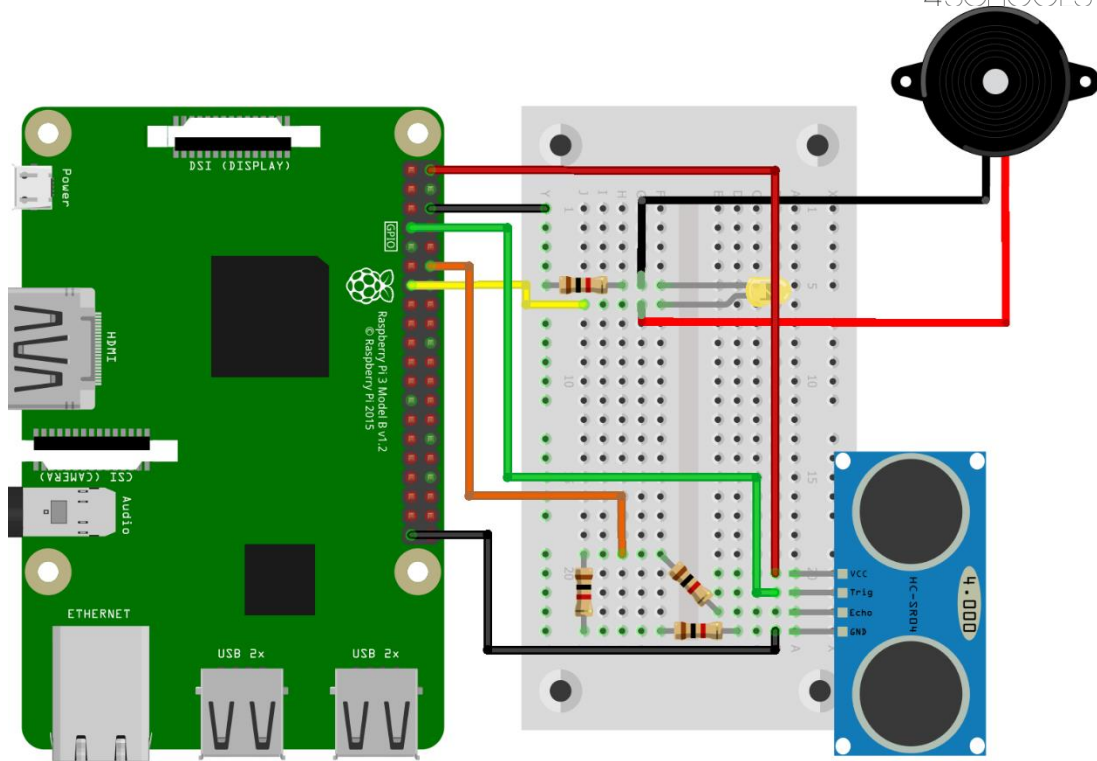
FIGURE 36 SCHEMATIC DIAGRAM OF CIRCUIT WITH MULTIPLE LEDs CONNECTED TO GPIO PINS

6.3.2 Example 2: Measure the Speed of Sound

In this exercise try to measure the speed of sound using the sensor and circuit from the second tutorial. For this you need to modify the code so that it prints out the signal time instead of the calculated distance. Take several measurements for various known distances between your hand, obstacle or object and the sensor. Write down in a table pairs of known distances (e.g. that you measured with a ruler) and your time measurements and then estimate the speed of sound yourself.

6.3.3 Example 3: Make a Proximity Alarm with Light and Sound

In this exercise try to add a buzzer to your alarm from Tutorial 3 to make sound according to distance. For this exercise you may make a circuit as shown Figure 16.



fritzing

FIGURE 37 SCHEMATIC DIAGRAM OF CIRCUIT WITH DISTANCE SENSOR, LED AND BUZZER CONNECTED TO GPIO PINS



6.4 Assessment test

1. We can setup a GPIO pin to be:
 - a. Input
 - b. Output
 - c. Input or Output**
2. You should turn off your Raspberry Pi before using its GPIO pins.
 - a. True**
 - b. False
3. In which state a GPIO pin can be?
 - a. High
 - b. High or Low**
 - c. Low
4. Raspberry Pi's GPIO has 3.3V and 10V voltage pins.
 - a. True
 - b. False**
5. One of the most important pins of the 40-pin header are:
 - a. The ground pins**
 - b. The voltage pins
 - c. The GPIO pins
6. Using Python programming and the GPIO pins you can interact with the physical world.
 - a. True**
 - b. False
7. Using Python in Raspberry Pi we can program its:
 - a. ground pins
 - b. voltage pins
 - c. GPIO pins**
8. Connecting the GPIO with a breadboard we need male-to-male jumper wires.
 - a. True
 - b. False**
9. Python is:
 - a. A general-purpose programming language**
 - b. A programming language only for Raspberry Pi
 - c. A snake with legs
10. Raspberry Pi is a fully functional, credit-size computer.
 - a. True**
 - b. False

6.5 References

- <https://www.raspberrypi.org/documentation/usage/gpio/>
- <https://github.com/splitbrain/rpiplusleaf>
- <https://www.raspberrypi.org/documentation/usage/gpio/python/README.md>
- <https://www.raspberrypi.org/documentation/usage/python/README.md>
- <https://pythonprogramming.net/gpio-raspberry-pi-tutorials/>

6.6 Additional resources

- Raspberry Pi GPIO: <https://www.raspberrypi.org/documentation/usage/gpio/>
- Physical Computing with Python:
<https://projects.raspberrypi.org/en/projects/physical-computing>
- Getting started with Python programming and the Raspberry Pi:
<https://raspberrypi.org/getting-started-with-python-programming-and-the-raspberry-pi/>
- The Python Tutorial: <https://docs.python.org/3/tutorial/>
- The MagPi magazine: <https://www.raspberrypi.org/magpi/>

6.7 Conclusion

Key conclusion points for **Module 4 – Raspberry Pi programming using Python:**

Once you have read this chapter, you are expected to have learned, understood and practiced the following learning elements:

- What a Raspberry Pi GPIO pin is and its basic functionality.
- How to make basic circuits with LEDs, buzzers, sensors and connect them to Raspberry Pi GPIO pins.
- How to setup, control and program GPIO pins using Python language.
- Familiarization with, and utilization of, the main constructs and syntax of the Python language including functions, variables, for loops, while loops, if statements.

7 Physical Computing [P1-ECAM]

7.1 Glossary of terms

Term / Concept	Definition / Explanation
Physical computing	Physical computing means interacting with real-world objects by programming them from a computer. Examples include programming an LED to flash, reading environmental data from a sensor, or even controlling robotic objects. Applications like these are all around us in everyday life, from traffic signals and ticket barriers to driverless cars and assembly lines. Behind each of these applications are algorithms and programs that govern their behaviour. Physical computing is combining hardware and software in order to create something useful or productive, or simply just for fun.
Application	An application is designed to carry out a specific task other than one relating to the operation of the computer itself. It is designed to help people perform an activity.
API	API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.
Peripheral	Any device attached to a computer but not part of it.
Programmer (hardware)	A programmer, device programmer, chip programmer, device burner, or PROM writer is a piece of electronic equipment that arranges written software to configure.

7.2 Main content

In this chapter the defining characteristics of physical computing are described. Key competences to be gained with physical computing will be identified.

7.2.1 Introduction to Physical Computing

“Physical Computing: Sensing and controlling the Physical world with computers” (T.Igoe, Tisch NYU).

This is a recent movement requiring skills on Electronics, Computing, Data Processing, Physics and even Design concept. The goal of Physical Computing is to create new devices in interaction with the real world.

In this chapter, this concept of Physical Computing will be introduced and participants will be taught the theoretical & practical aspects of it.

This chapter is built around examples as the development of an autonomous mobile robot, able to move thanks to different sensors installed on it and regarding several rules.

Definition from Wikipedia the free encyclopaedia:

“Physical computing involves interactive systems that can sense and respond to the world around them. While this definition is broad enough to encompass systems such as smart

automotive traffic control systems or factory automation processes, it is not commonly used to describe them. In a broader sense, physical computing is a creative framework for understanding human beings' relationship to the digital world. In practical use, the term most often describes handmade art, design or DIY hobby projects that use sensors and microcontrollers to translate analog input to a software system, and/or control electro-mechanical devices such as motors, servos, lighting or other hardware.”

What Is Physical Computing?

Physical Computing is an approach to learning how humans communicate through computers that starts by considering how humans express themselves physically. A lot of beginning computer interface design instruction takes the computer hardware for given — namely, that there is a keyboard, a screen, perhaps speakers, and a mouse — and concentrates on teaching the software necessary to design within those boundaries. In physical computing, we take the human body as a given, and attempt to design within the limits of its expression.

This means that we have to learn how a computer converts the changes in energy given off by our bodies, in the form of heat, light, sound, and so forth, into changing electronic signals that it can read interpret. We learn about the sensors that do this, and about very simple computers, called microcontrollers, that read sensors and convert their output into data. Finally, we learn how microcontrollers communicate with other computers.

Physical computing means creating or using devices that interact with the world around them. A physical computer senses its environment, processes that information, and then performs some action. This “sense – think – act” cycle can also be used to define a robot. Creating a robot involves engineering or manufacturing. Then you also need to write a computer program to make the robot do something interesting! Robots don't always look like the ones shown in movies. Some are humanoids or robotic cars, but you can also make robotic pets or gardens. The sky's the limit! Robots provide students with a way to see their code working in the world (Article from: <https://marketbrief.edweek.org/the-startup-blog/physical-computing-a-primer/>).

Physical computing covers the design and realization of interactive objects and installations and allows students to develop concrete, tangible products of the real world that arise from the learners' imagination. This way, constructionist learning is raised to a level that enables students to gain haptic experience and thereby concretizes the virtual.

Physical computing takes a hands-on approach, which means that you spend a lot of time building circuits, soldering, writing programs, building structures to hold sensors and controls, and figuring out how best to make all of these things relate to a person's expression.

What is needed to know before taking a physical computing class?

First, most of the real work happens outside of class, both in the shop building and programming, and in the everyday world watching people and figuring out what they do that you're interested in sensing and interpreting.

Second, if you've never programmed a computer before, you'll need to know a little bit about that. In particular, it helps if you've done some graphic interface programming, because later in the course, we learn how to control screen graphics and sound with the

output from sensors. At the very least, you should know and be comfortable with the following:

- What an if statement is, and how to use one.
- What variables are in a computer program, and how they're used
- What a repeat loop is, and how to use them. If you're comfortable with statements like for-next, while-wend, or repeat while... -end repeat, then you're in good shape

Three Pillars of Physical Computing

Physical computing is quite a new and unfamiliar concept among informatics teachers and informatics didactic researchers. Literature review has shown that different authors put different emphases on the following three pillars of physical computing: typical products, tools and processes of physical computing need to be investigated in order to clarify the meaning. It will not be possible to clearly draw lines between the three fields. Processes involve tools and aim at particular products, so different perspectives will therefore certainly put a stronger focus on any of the three variables, but not neglect the other two. As a starting point it is assumed that physical computing means to creatively design tangible interactive objects or systems using programmable hardware.

7.2.2 Key Competences in Physical Computing

Understanding Computing Systems

Interactive objects or installations made with physical computing are entire computing systems containing hard- and software components that students can assemble themselves and investigate further. Depending on the level of complexity they undergo in the particular setting, they can come all the way from an intuitive understanding (e.g. when controlling a programmable toy) to a deep understanding of interactive computing systems (e.g. when constructing an intelligent letterbox). Particularly aspects of hardware design help them to develop the abilities to identify and understand interactive systems in their every-day environments.

Formulating Problems

With physical computing, the basic ability to precisely formulate problems is formed and practiced as a first step in the process of designing and creating interactive objects. Students are required to unambiguously describe what is supposed to happen from an outside perspective, thus they focus on the problem formulation separately from thinking about possible ways of problem solving.

Organising and Analysing Data

In physical computing projects data can be collected automatically with self-made weather stations, a voting system to elect the next class representative or an automatic traffic recorder to count the number of cars passing outside school. This way, students learn with real-world data collected in their own environment by measurement objects they have designed and built themselves. They will find out about the coding and decoding of data and information while working with sensors that deliver data, which need to be interpreted and actuators that receive data, which has to be generated from information.

Algorithmic Thinking

Algorithmic thinking is also a crucial element of the physical computing process. Students at any level are required to precisely describe series of events, both serial and parallel.

Physical computing in particular demands students to develop algorithms that allow their objects to run continuously and interact steadily with the environment.

Effectiveness and Efficiency

Key aspects of computational thinking include identifying, analysing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources. In physical computing projects, ineffective or inefficient solutions are particularly evident. Interactive objects or installations should give immediate feedback, may include concurrent processes and should include intuitive interfaces. If they fail to meet the expectations, e.g., due to the choice of inappropriate sensors, excessive pauses or delayed responsiveness, this is immediately noticeable.

7.3 Practical examples

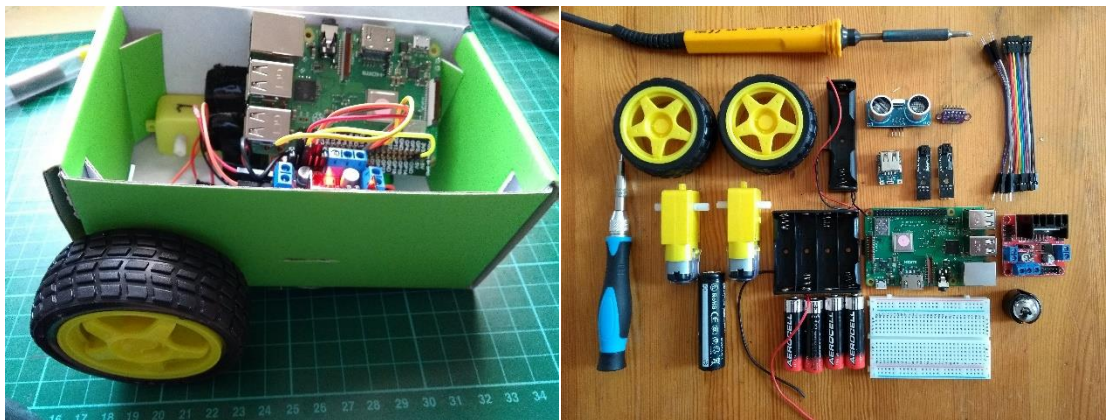
For testing your knowledge and practicing further we propose some practical examples.

- **Example 1: Robot Buggy (A Raspberry Pi on wheels?)**
- **Example 2: YouTube Boombox (Raspberry Pi-powered lo-fi video player for YouTube)**

7.3.1 Example 1

This beginner's project teaches you how to set up a motor controller board, build a chassis, and use Python to control the motors. Don't be afraid, it's really easy to set up. In this project you will build a robot buggy that you can program to move around using simple Python commands.

- How to set up a motor controller board with two motors
- How to control motors using Python
- How to build a robot chassis



What you will need:

- Raspberry Pi 3
- Motor controller board
- 2 × 3V - 6V DC motors
- 2 × wheels
- 1 × AA battery holder (for 4 AA batteries)
- 4 × AA batteries
- Ball caster
- Wire or jumper leads
- A USB Battery pack
- Screw driver
- Soldering iron and solder
- Wire strippers
- Small cardboard or plastic box and glue/tape

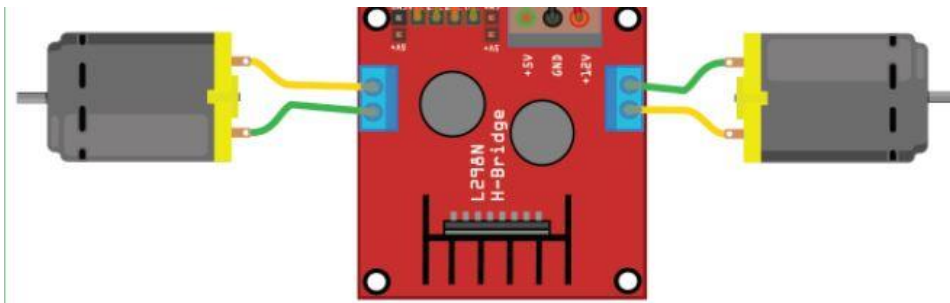
Assembling the motors and board

The first thing you will want to do is to connect your motor controller board to your Raspberry Pi, the battery pack, and your two motors, to test that they are all working.

The instructions here are for L298N Dual H Bridge DC Stepper Motor Driver Controller Board, and they will be pretty similar for most motor controller boards.

Connect the motors to the board

You will need to connect the motors to the board. For this you will require a small screwdriver.



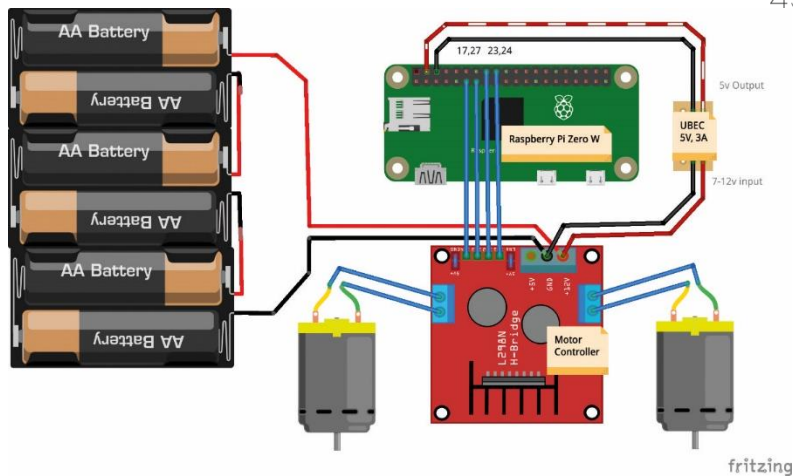
Powering the motors

The motors require more power than the Raspberry Pi can provide. Therefore, you will use four AA batteries to power them.

Connecting the board to your Raspberry Pi

The board used in this project needs to be wired to the Raspberry Pi. Other boards may connect differently, and some boards can simply be placed onto the Raspberry Pi GPIO pins as a HAT.

On the board used here there are pins labeled In1, In2, In3, and In4, as well as two GND pins. Which GPIO pins on your Pi that you use is up to you; in this project, GPIO 17, 27, 23, and 24 have been used.



Left, right, forward, backward

You need to figure out which is your left motor and which is your right motor. You also need to know which way they are driving to go forward, and which way they are driving to go backwards.

Assemble your robot

There is no “right” way to build your prototype robot chassis, but there are a few things to bear in mind:

The chassis needs to house the Raspberry Pi, motor controller, and batteries.

The chassis needs to allow the mounting of a pair of wheels.

You may want to later add a couple of line sensors, and an ultrasonic distance sensor or a lidar sensor to the chassis.

It’s always a good idea to build a prototype chassis first. Eventually, you can learn how to laser-cut or 3D print a chassis, but in this project, a cardboard box is used as a temporary solution.



To use your Raspberry Pi without connecting a mouse, monitor, or keyboard, you can remotely access it via SSH or VNC.

You can now write a program to control your robot and make it do any number of things. Here's a simple script to make it go in a square shape (you may need to change the sleep functions slightly):

```
from gpiozero import Robot
from time import sleep
robot = Robot(left = (17, 27), right = (23, 24))
while True:
    robot.forward()
    sleep(3)
    robot.stop()
    robot.right()
    sleep(1)
    robot.stop()
```

Now is your opportunity to program your robot!

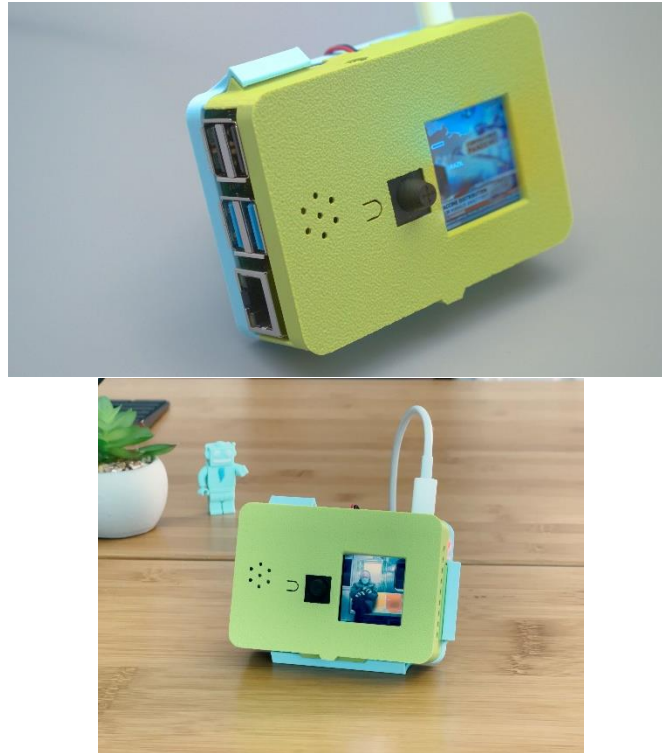
Try and complete one of the following challenges:

1. Make your robot drive in a perfect circle
2. Make your robot drive in a zigzag pattern
3. Make a small maze from household objects and program your robot to navigate it

Don't forget, there are only five basic commands to move your robot:

```
robot.forward()
robot.backward()
robot.right()
robot.left()
robot.stop()
```

7.3.2 Example 2



This little device may look like the back of a compact camera, but it's actually a Raspberry Pi-powered lo-fi video player for YouTube. The 3D printed case also features a built-in speaker and a small display. The Pi is set to kiosk mode and automatically plays any YouTube music stream. A built-in joystick can be used to change the stations and adjust the volume.

- The BrainCraft HAT has everything you need to make an all-in-one YouTube player.
- The built in joystick can be used to change the stations and adjust the volume.
- This lets you quickly switch between different YouTube channels without having to use a keyboard or mouse.
- You can also pause and play the video by pressing the button next to the joystick.

What you will need:

Electronic Parts:

- Raspberry Pi 4 Model B - 4 GB RAM
- 16GB Card with NOOBS 3.1 for Raspberry Pi Computers including 4
- Speaker - 40mm Diameter - 4 Ohm 3 Watt
- JST PH 2-Pin Cable - Female Connector 100mm
- Full Size Wireless Keyboard with Trackpad

3D Printing:

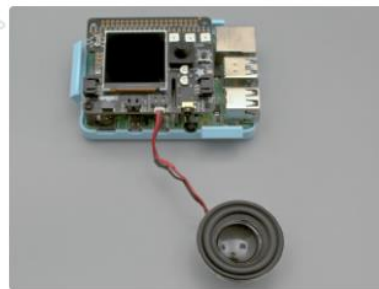
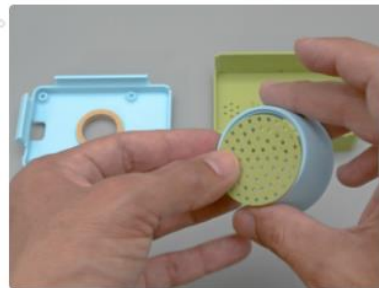
STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below.

- [BrainTube-screen.stl](#)



- BrainTube-case.stl
- BrainTube-tripod.stl
- BrainTube-speaker-grill.stl
- BrainTube-speaker-cone.stl
- BrainTube-speaker-ring.stl

<https://www.thingiverse.com/thing:4739213>



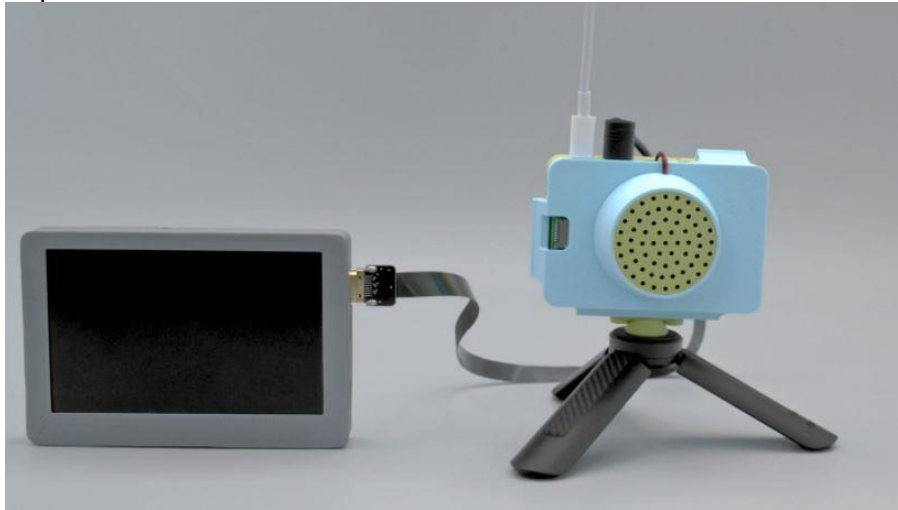
Assemble

- Press Fit Speaker Holder
The speaker ring part fits into the cutout on the BrainTube-Case part.
- Speaker Grill
Fit the Speaker-Grill part into the Speaker-Cone part.
- Add Raspberry Pi to Case



Place the Pi over the standoffs and align the board with the port openings on the case.

- Speaker
 - Solder the speaker wires to the JST female connector. Use heat shrink to insulate the connections
- Screen Case
 - The front cover is fitted over the HAT with the speaker cable fitted through the slit on the HDMI port opening.
 - Add gaffers tape or kapton tape to insulate the speaker magnet. The Speaker press fits into the ring.
 - Align the speaker cone over the speaker wires and then press fit over the speaker.



Set up Raspberry Pi

- Set up SD card
 - Attach an HDMI monitor, to set up the Pi OS. Use the official Raspberry Pi Imager to burn the latest OS on a SD card.
- Enable SSH
 - After all updates are complete, enable SSH
- Install Blinka Libraries
 - It allows many of the libraries that were written for CircuitPython to run on CPython for Linux. Follow the step below to set up the Blinka libraries:
Blinka Setup
- Configure Audio, Fan and Display



- Install Kiosk-mode Script
Use the kiosk-mode script to enable fullscreen video playback
- Edit Playlist
Add your own streams:

7.4 Assessment test

1. What is an input or output device?
 - a computer like an iPad, smartphone, laptop or PC
 - **a piece of hardware that is used to exchange information with a computer.**
 - a type of vehicle like a car or truck
 - something you would download and install on your iPhone
2. What is data?
 - the place where information is saved in a computer
 - a small part of a computer
 - a type of robot
 - **pieces of information used by machines and/or people**
3. The keyboard is one of the most prominent _____ devices of a computer.
 - **input**
 - output
 - input and output
4. When a printer prints something, it _____.
 - inputs
 - **outputs**
 - inputs and outputs
5. A VR headset is an example of _____. for a computer.
 - input
 - output
 - **input and output**



6. In Python, what can you NOT do in the Shell?
- **save your code and run it again**
 - run your code
 - output the answers to maths problems
7. Which of these is an advantage of using variables? (Choose all correct answers).
- **can be used to store input**
 - **don't need to type information in again and again**
 - **makes our code shorter and simpler to read**
 - **easier to use information in the code**
8. Outputting text in Python programming language. Which is correct?
- print("Happy New Year!")
 - output("Happy New Year!")
 - Print("Happy New Year!")
 - **print("Happy New Year!")**
9. Which is the correct way to write "divide a by 2 and multiply by b" in Python?
- a / 2 x b
 - a \ 2 x b
 - **a / 2 * b**
 - a \ 2 * b
10. How do we make a variable called d containing the text Good morning in Python?
- d(Good morning)
 - d = (Good morning)
 - d = Good morning
 - **d = "Good morning"**

7.5 References

- https://en.wikipedia.org/wiki/Physical_computing
- <https://www.tigoe.com/blog/what-is-physical-computing/>
- <https://www.qaeducation.co.uk/news/physical-computing#:~:text=Physical%20computing%20means%20interacting%20with,or%20even%20controlling%20robotic%20objects.>
- <https://guides.temple.edu/c.php?g=419841&p=2863656>
- Maren Przybylla, Ralf Romeike, Key Competences with Physical Computing https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/8290/file/cid07_S351-361.pdf
- <https://sunnie-sva-physicalcomputing.tumblr.com/post/66904922347/physical-computing-mid-term-project-music>
- <https://quizizz.com/admin/quiz/5e153511ae952c001b88401a/physical-computing-review-quiz-1-d>

7.6 Additional resources

- Filiz Kalelioglu, Sue Sentence, (2020) Teaching with physical computing in school: the case of the micro:bit, *Education and Information Technologies* 25, 2577–2603.
- Tom Igoe, Making Things Talk: Practical Methods for Connecting Physical Objects, Make; 1 edition (September 28, 2007), ISBN-10: 0596510519 (source of examples). Second Edition, Released: August 2011 (est.) ISBN-10: 1449392431, ISBN-13: 978-1449392437, <http://oreilly.com/catalog/0636920010920>
- Joshua Noble, Programming Interactivity: A Designer's Guide to Processing, Arduino, and OpenFrameworks, O'Reilly Media, July 2009
- Alvaro Cassinelli, Daniel Saakes, Data Flow, Spatial Physical Computing, TEI 2017, March 20–23, 2017, Yokohama, Japan

7.7 Conclusion

Physical Computing is a complex activity that requires learners to be aware of hard- and software issues at the same time. On the introductory level (primary school), students may work with programmable toys or programmable bricks and drag & drop programming environments to learn the fundamentals of algorithmic thinking. Construction kits that have pre-assembled sensors and actuators to be plugged into a microcontroller either directly or with a shield allows older children to come to visible and tangible achievements very quickly. With advancing in physical computing both, on the hard- and the software side, students undergo learning processes that strengthen computational thinking and key competences that are necessary for all aspects of life. Physical computing can enrich future informatics classrooms with valuable competences that are focused in computer science education better than in many other subjects, as they are innate in the subject and not to be imposed artificially. Future research on this topic will therefore investigate how students of different age groups experience physical computing activities and what learning processes they undergo.

8 General conclusion

Even if children are born into technology, they seem to need to acquire technological skills, such as programming. New ways of engaging children into programming and STEM are needed. We choose Hands-on play through the creation of games playable on a retro design DIY wooden computer in combination with electronic gadgets related to STEM subjects. This approach is fun and more educational instead of more screen time. The bridging of the online and the offline worlds may offer a more engaging and healthier environment for children to learn how to program and develop STEM skills.

The primary objective of the STEMKIT4Schools Erasmus+ project is to produce approaches and tools to help those working with children by developing STEM related skills. It aims to achieve this not by increasing screen time but by encouraging hands on play.

A complete curriculum has been designed by the partners of the project to answer the goals of STEMKIT4Schools, including lessons plans for using Minecraft Pi/Scratch/Python/Kits with the STEMKIT computer in the classroom. The lesson plans are part of the Educator's guide for teachers. The electronics kits are designed and built to complement the teaching of programming, physical computing and STEM subjects. The final STEMKIT course teaches basic elements of games with Minecraft Pi, Python, Scratch, as well as physical computing (using basic digital, analogue, and electromechanical components) and collaboration (engage and share with others). The STEMKIT DIY computer is based on Raspberry Pi, meaning it can harness the power and practically unlimited resources for Raspberry while the features of the Raspbian operating system provide additional capabilities. The re-purposed STEMKIT curriculum using instructional design principles can be found inside the Learning Portal in the form of interactive Learning Objects and will be delivered to the learners who will be able to follow the interactive course at their own time and pace while they can utilize the social learning tools offered to engage with their peers (engage with comments, forums, chat). Additional collaboration features such as notifications, group feeds, blogs, file sharing, questions/answers, voting will be also provided.

The output O2A1 "The STEMKIT CURRICULUM DESIGN & DEVELOPMENT" contains the following 5 items namely: Introduction to Scratch 2.0 [P6-ARC], Scratch GPIO (Control GPIO pins/receive inputs) [P3-DANMAR], Introduction to Raspberry Pi Edition of Minecraft [P4-HESO / P5-SCHOLE], Raspberry Pi GPIO programming using Python [P2-AKNOW] and Physical Computing [P1-ECAM]

Scratch has a small number of commands, allows sprites to be exchanged without breaking dependencies, fostering collaboration and code sharing. The system is always live, with no run/edit switch, so commands or code snippets can be run with a click, and graphical feedback shows execution. Variables and lists have concrete visualizations, so the effect of data operations can be seen immediately. (Maloney, 2010)

The ability to code computer programs is an important part of literacy in today's society. When people learn to code in Scratch, they learn important strategies for solving problems, designing projects, and communicating ideas.

Scratch Projects - based on learning units will include different disciplines and this new perspective will allow students to apply what they learn to new situations, leading to

deeper learning; students will be engaged in design activities, pursuing personal interests, interacting through creative collaborations and reflecting on useful experiences.

Introduction to Minecraft Pi has allowed the following key conclusion points.

If you followed this resource with your Raspberry Pi, you are expected to:

- Access Minecraft Pi and create a new world.
- Navigate around Minecraft Pi using the movement controls on your keyboard.
- Know how to place and destroy a block, and navigate through different types of blocks in the in-game inventory.
- Connect Python to Minecraft Pi.
- Use Python programming interface.
- Manipulate blocks using Python code and scripts.
- Have a good understanding of the rest of Minecraft Pi functions.
- Make Minecraft interact with the outside world with the use of buttons and LEDs.
- Import new worlds and discover resource packages compatible with Minecraft Pi.
- Raspberry Pi programming using Python has allowed the following key conclusion points.
- Once you have read this chapter, you are expected to have learned, understood and practiced the following learning elements:
 - What a Raspberry Pi GPIO pin is and its basic functionality.
 - How to make basic circuits with LEDs, buzzers, sensors and connect them to Raspberry Pi GPIO pins.
 - How to setup, control and program GPIO pins using Python language.
 - Familiarization with, and utilization of, the main constructs and syntax of the Python language including functions, variables, for loops, while loops, if statements.

Physical Computing is a complex activity that requires learners to be aware of hard- and software issues at the same time. On the introductory level (primary school), students may work with programmable toys or programmable bricks and drag & drop programming environments to learn the fundamentals of algorithmic thinking. Construction kits that have pre-assembled sensors and actuators to be plugged into a microcontroller either directly or with a shield allows older children to come to visible and tangible achievements very quickly. With advancing in physical computing both, on the hard- and the software side, students undergo learning processes that strengthen computational thinking and key competences that are necessary for all aspects of life. Physical computing can enrich future informatics classrooms with valuable competences that are focused in computer science education better than in many other subjects, as they are innate in the subject and not to be imposed artificially. Future research on this topic will therefore investigate how students of different age groups experience physical computing activities and what learning processes they undergo.