



# STEAM activities for



and...



and...





**Qualified Generations with STEAM Education**

**2022-1-SK01-KA220-SCH-000087555**

under addendum to contract 2



**Co-funded by  
the European Union**

Title

**Microbit**

Subjects

**Science** (Physics with maths, Biology, Geology,...)

**Technology** (BBC micro:bit , Python programming, TI Hub, Probes, ...)

**Engineering** (smart irrigation system, ...)

**Arts** (under tech with maths - music, painting,...),

**Mathematics** (Computational thinking, Statistics, Probabilities, ...)

Author

**Bruno Reimão**

**Joaquim Sousa**

**Rui Rua**

**Raul Gonçalves** (and team coordinator)

School/ Country

**AE Ermesinde/ Portugal**



**2024**

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

# Lets programming?

## Activities



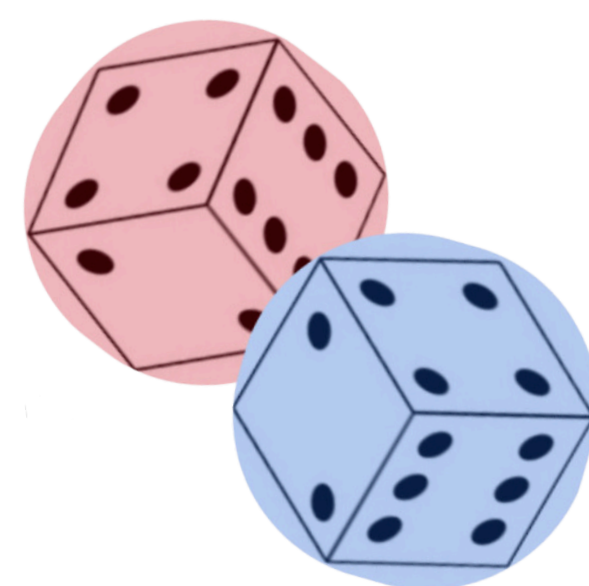
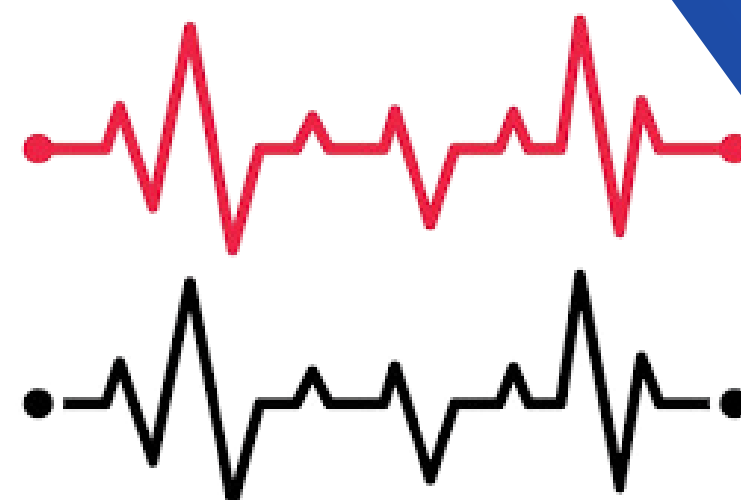
### ... with BBC micro:bit

1 What is your name?

2 The heartbeat.

3 Which side will come out?

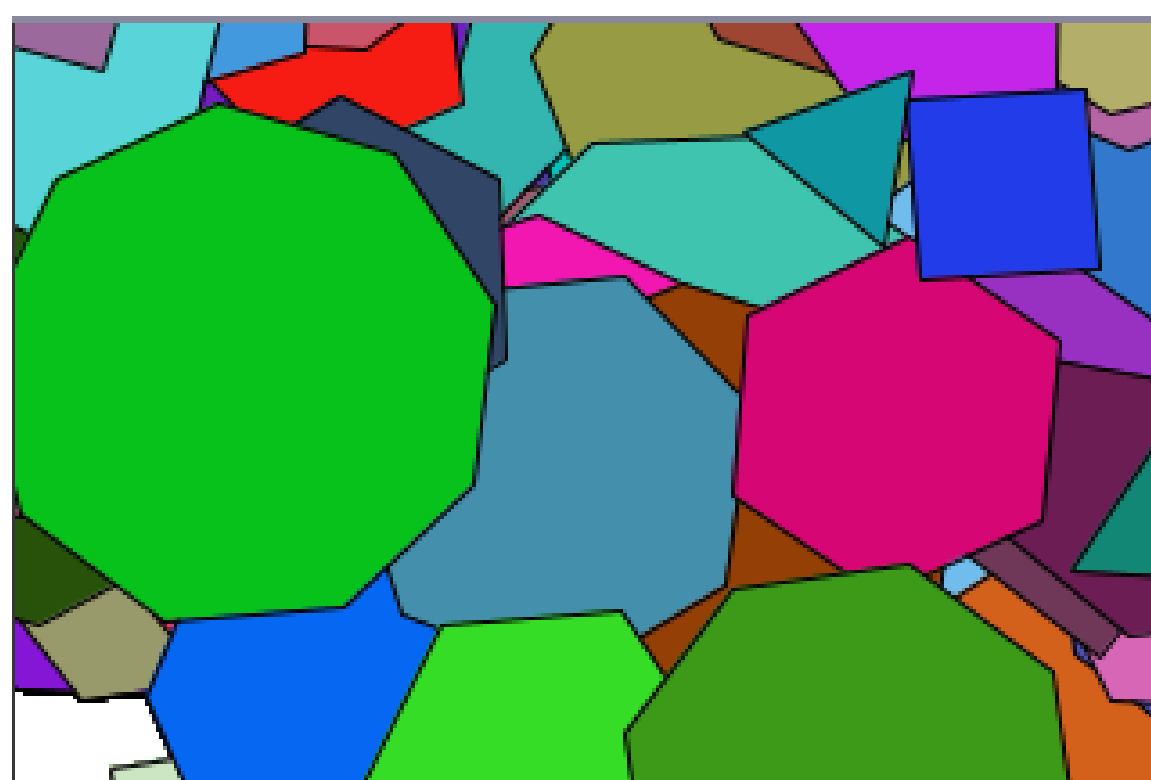
4 m&m - micro:bit and music.



### ... without BBC micro:bit

5 What a beautiful work of art!

6 "The" smart irrigation system!

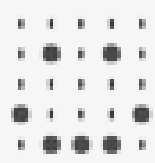
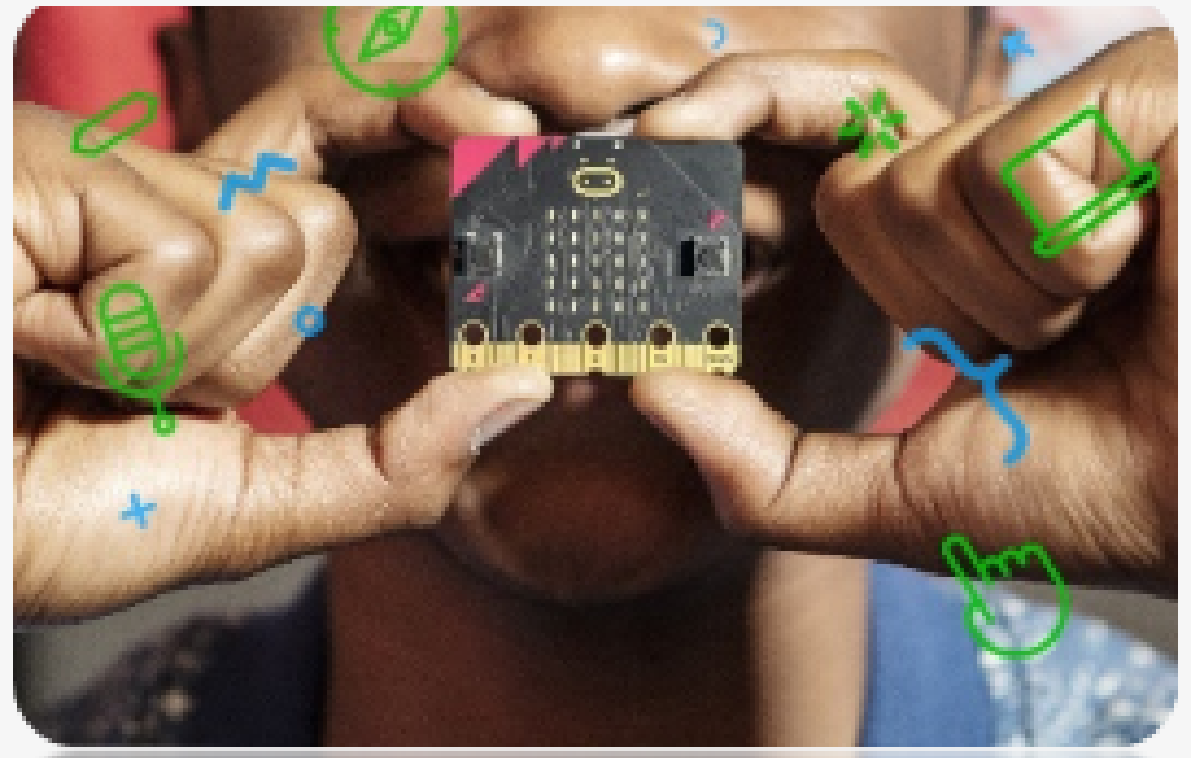




# What is the BBC micro:bit?

The micro:bit is a **small computer**, a multifunctional microcontroller board.

This microcontroller features an LED light display, sensors and many I/O features that encourage interaction with the user and the world around them.



Display of image patterns and animations on the 5x5 LED matrix.



Play music with a speaker.



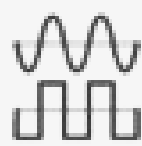
Establish communication between two calculators via micro:bit radio functions.



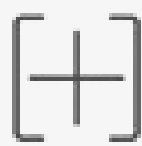
Interpretation of values from the integrated compass, accelerometer, temperature sensor and light level sensor.



Interaction with buttons and capacitive touch sensor.



Communication with analog and digital input/output.



Interaction with external sensors (with optional expansion board).



Connection to NeoPixel LED strips.

## The BBC micro:bit on the TI calculator

The micro:bit can be connected to the TI-Nspire™ CX II-T graphing calculator, allowing real-time interface with the calculator's keyboard and screen in connexion with the micro:bit.

You can therefore program the micro:bit directly from calculators or in the TI-Nspire technology computer software.

Furthermore, in Python “copy-and-paste” is supported with the standard micro:bit syntax. On the TI-Nspire™ CX II-T graphing calculator there is the microbit.tns module that was created to communicate with the latest version: micro:bit V2.

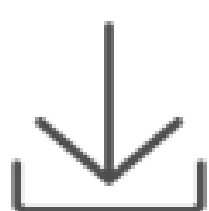
This version of the micro:bit includes a built-in speaker, microphone and a capacitive touch sensor.



# How to install the micro:bit module on the calculator?



By clicking on the link <https://education.ti.com/en/product-resources/microbit>, or reading the QR Code on the side, you can access the module installation in three steps:



## 1. Download the files.

These files include the “runtime” hexadecimal file and a user module .tns file. Two files are included in a .zip file (in English) for an easier download.

## 2. Install the user module on the calculator.

This will enable the Python programming functionality required to communicate with the micro:bit. Once installed, the micro:bit module will appear below the “More modules” menu.



You can find step-by-step instructions on how to prepare your micro:bit and TI-Nspire CX II-T graphing calculator included in the .zip file available for download.

## 3. Install the “runtime” hexadecimal file for the micro:bit.

To do this, drag and drop the hexadecimal file into the micro:bit file directory when connected to a computer. If the process is successful, the TI logo will appear on the LED screen.



# How to connect the micro:bit to the calculator?

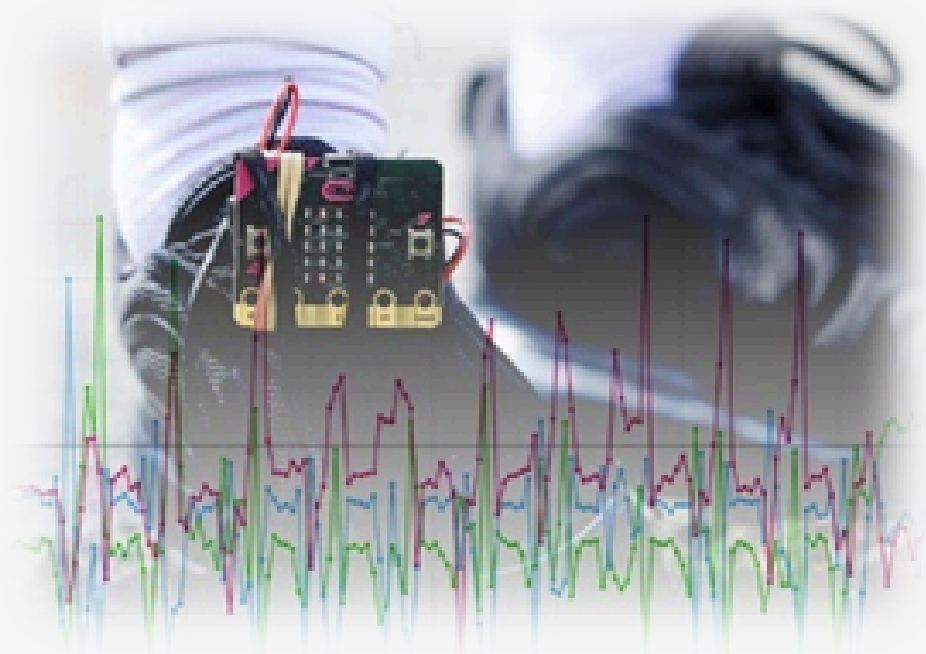
Using a Mini-A – Micro-B USB Cable (sold separately).

Additionally, connections can be made to the TI-Nspire™ CX software using the USB Type A – Micro-B cable.

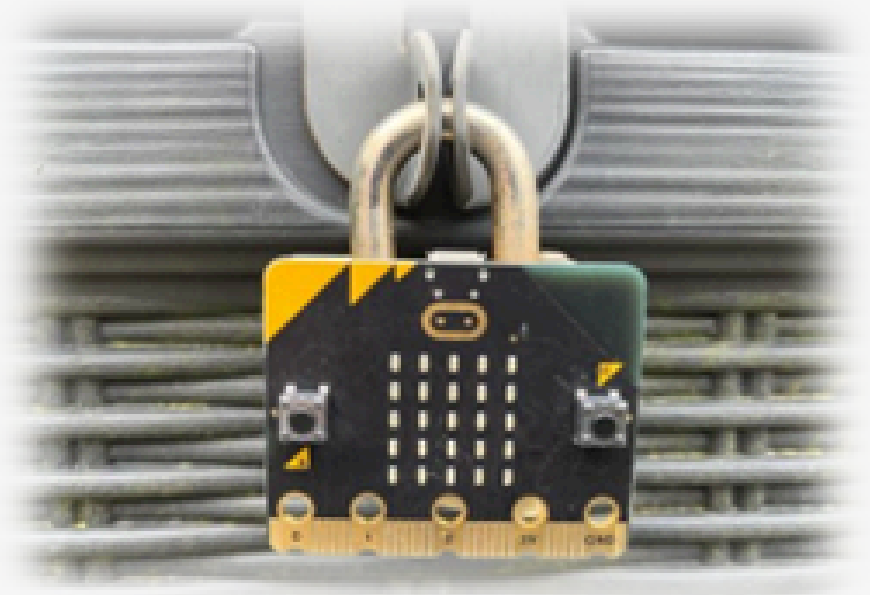


# The introduction to reality in a technological world...

... in data collection, starting the approach to Data Science.



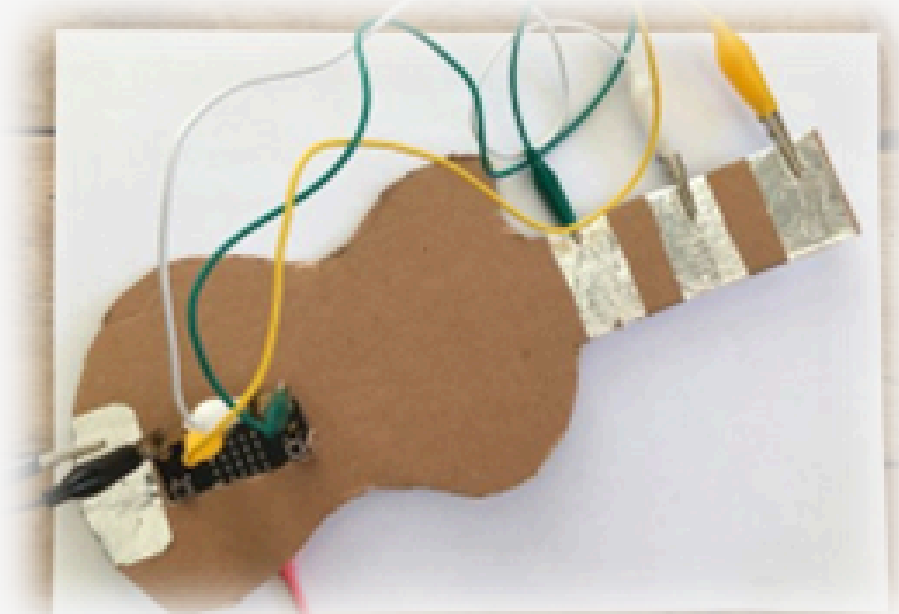
...bringing the notion of digital security.



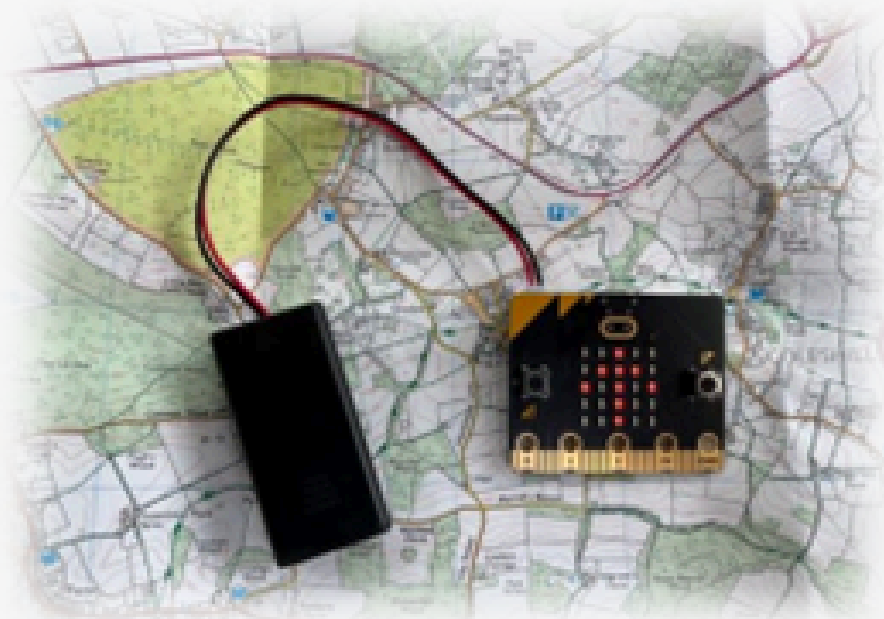
...making learning fun!



...being creative with lights, sounds and movement.



...getting outdoors with technology.

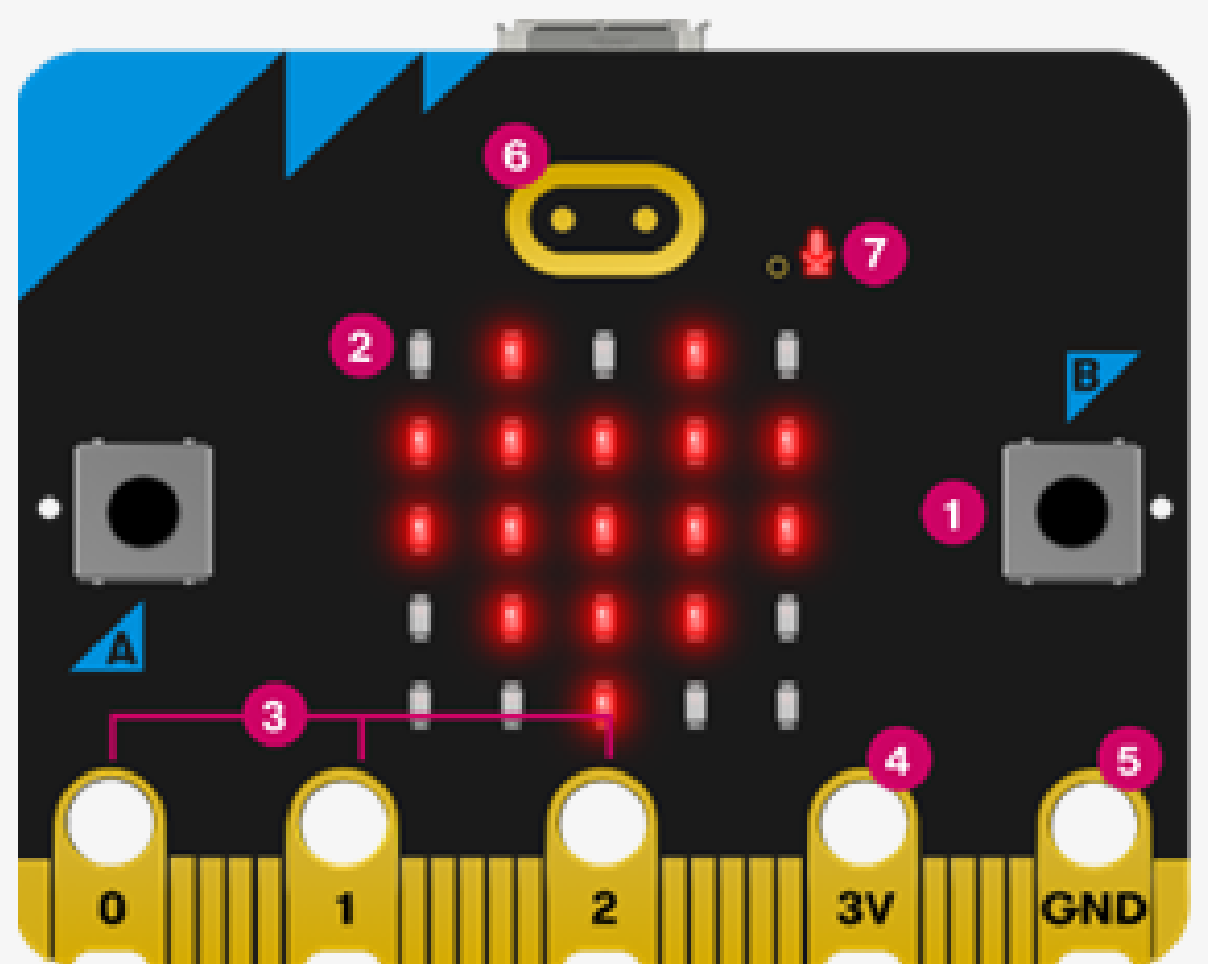


...exploring environmental issues.



# The constitution of the micro:bit V2 ...

... head on.



- 1 Buttons A and B**

The micro:bit has two buttons on the front that can be used separately or together to make things happen.
- 2 LED display and light sensor**

The 25 LED lights arranged in a 5x5 grid make up the display that shows images, words and numbers. In addition, LEDs can also act as sensors, measuring the amount of light that falls on your micro:bit.
- 3 Pins 0, 1 and 2**

These pins allow you to connect headphones, feel touch and add electronic components to expand the micro:bit's uses.
- 4 Pin 3V – 3 volt power**

You can power external LED lights and other electronic components using this pin.
- 5 Pin GND**

It is used to complete electrical circuits when connecting headphones, LED lights or external switches to the micro:bit.
- 6 Touch logo**

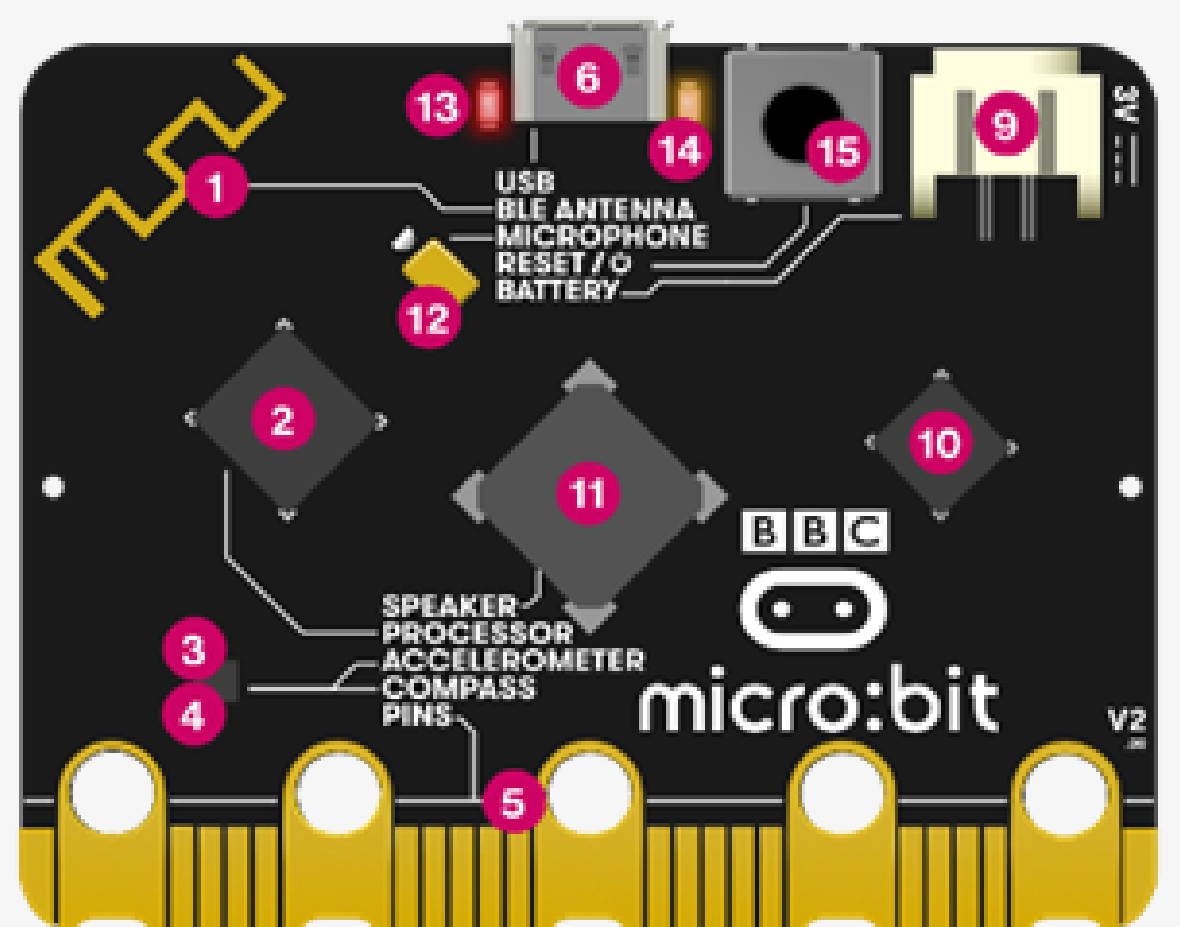
The golden logo also doubles as a touch sensor. It can be used as an extra button in programs, in addition to the A and B buttons.
- 7 Microphone LED**

On the micro:bit, programs can be created that react to high and low sounds and measure noise levels with the integrated microphone. The microphone LED shows when the microphone is actively measuring sound levels. To the left of the LED, there is a small hole where the sound enters.



# The constitution of the micro:bit V2 ...

... from behind.



8

## Battery input

Instead of powering your micro:bit from the USB socket, you can disconnect it from the computer and use a battery. This can be useful if you want to take the micro:bit outside, use it or play with it. It can work for a long time using just two AAA batteries.

9

## Chip de Interface USB

The interface chip is used to send new code to the micro:bit, sending and receiving a set of data to and from the calculator or computer via USB.

10

## Speaker

You can easily add music and new sounds to projects.

11

## Microphone

The microphone and LED indicator are attached to the back of the board. The LED lights up when monitoring sound levels and is visible as a microphone icon on the front of the card. The front also has a small hole to allow sound to enter the microphone.

12

## Power red LED light

The red LED on the back of the micro:bit shows when it is powered, whether via batteries or a USB cable.

13

## USB Yellow LED

The yellow LED light blinks when the calculator or computer is communicating with the micro:bit via USB, for example, when you update a program file.

14

## Power and Reset Button

Pressing this button will restart the micro:bit and run the program again from the beginning.

To learn more about each component that makes up the micro:bit V2, simply access

<https://microbit.org/get-started/features/overview/>

or by reading the QR Code on the side.



# Activity 1

## What is your name?

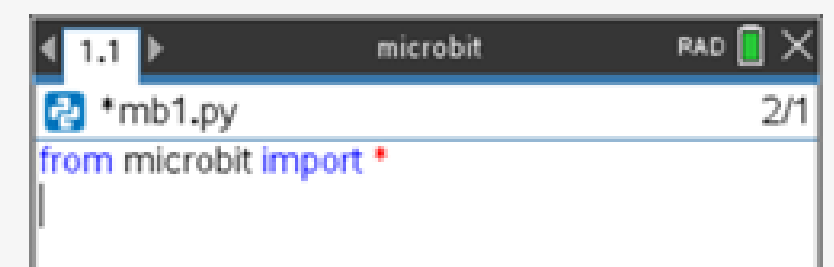
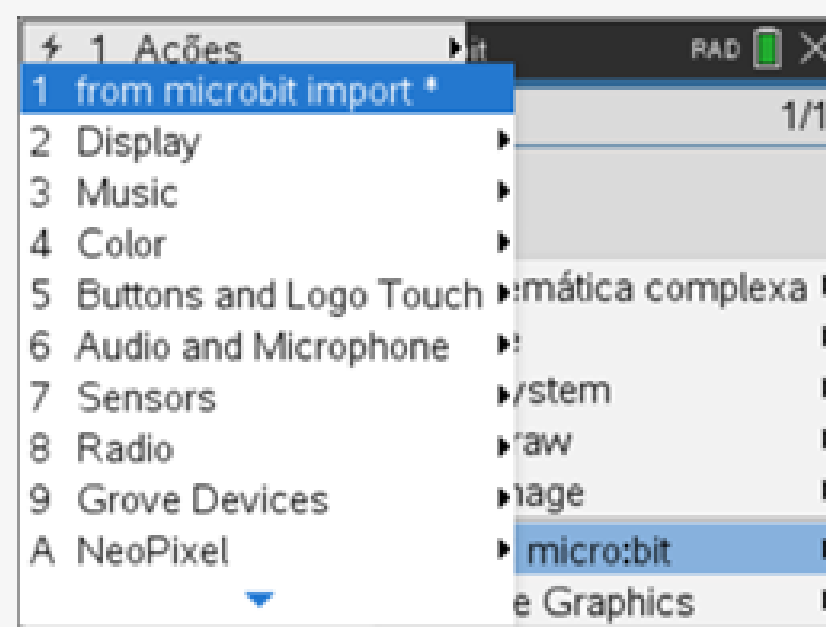
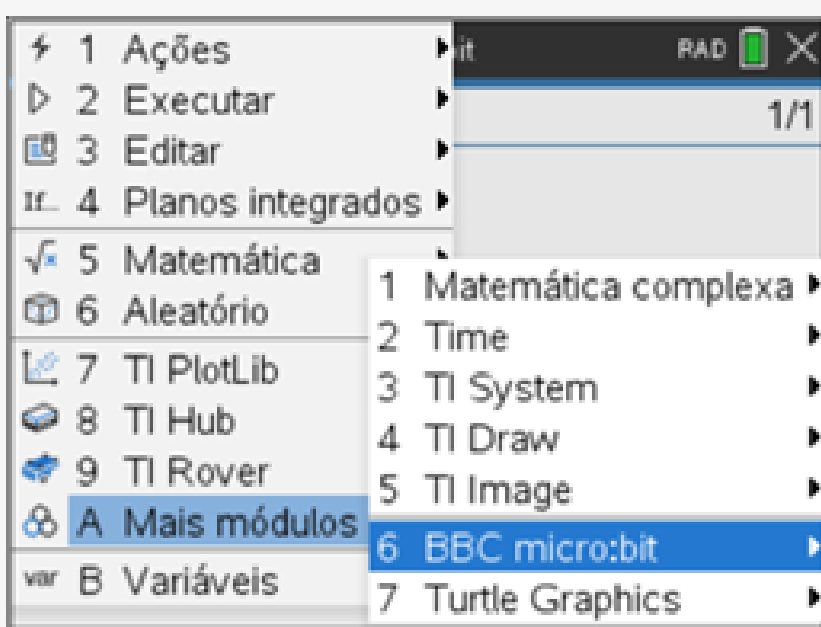


As you would expect, just like any other first programming experience, you will start by displaying an expression on the micro:bit display!

To insert the new program, you start by creating a document where you can write the lines of code! To do this, select **A: Add Python** and then **I: New**.

At the beginning, you will need to give the program a name. For example, 'mb1'.

Then, to be able to work with the micro:bit module, you need to import it. To do this import, you can either write **from microbit import \*** using the calculator keyboard or access **menu**:

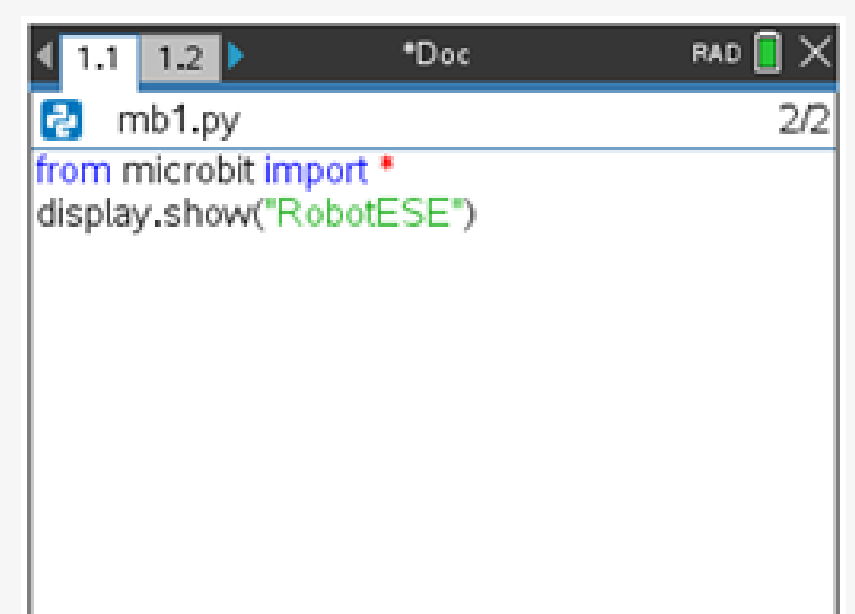
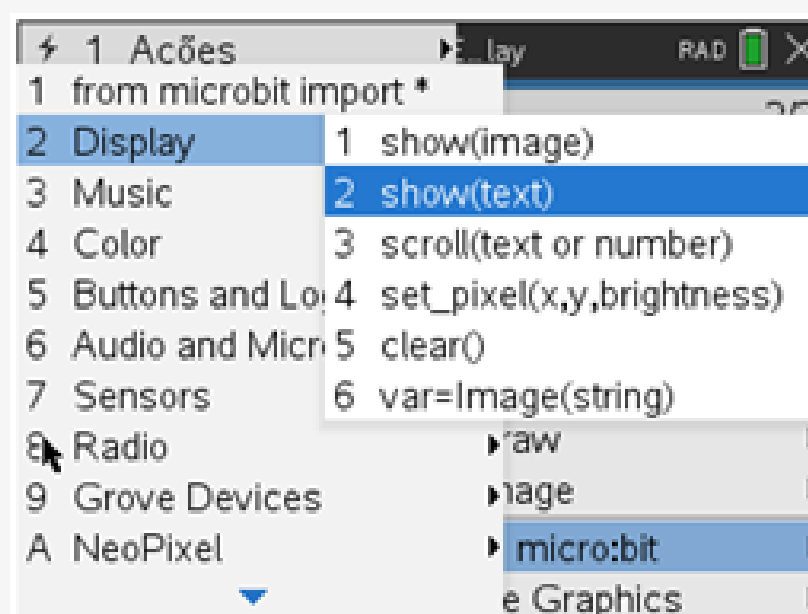
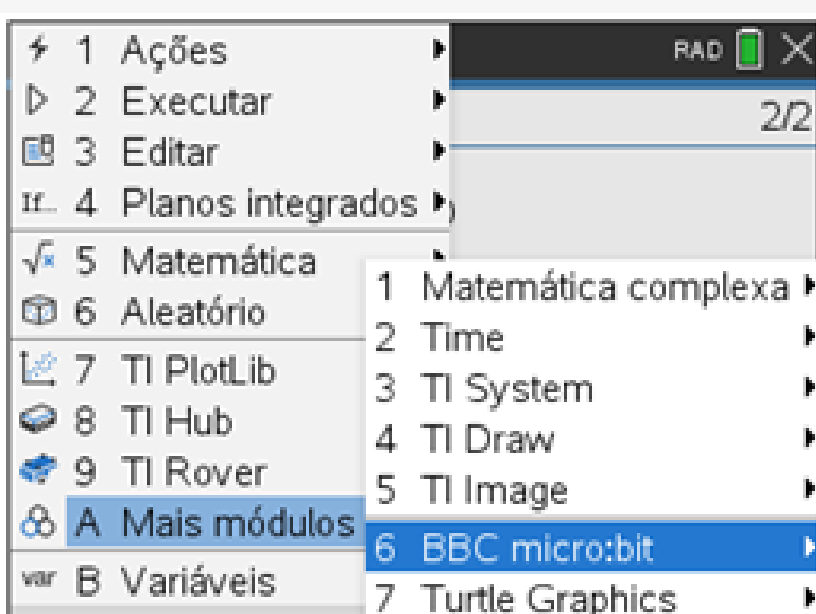


**Objective: to make the name 'RobotESE' appear on the micro:bit display.**

To display an expression on the micro:bit display, use the **display.show(text)** command.

The command is entered as **display.show(value)**. The **(value)** is just a placeholder that will have to be replaced by something: either by a set of characters (called a string) or by the name of a variable (as we will see later).

Inside the parentheses, replace the value with the word or expression in quotation marks: "RobotESE"

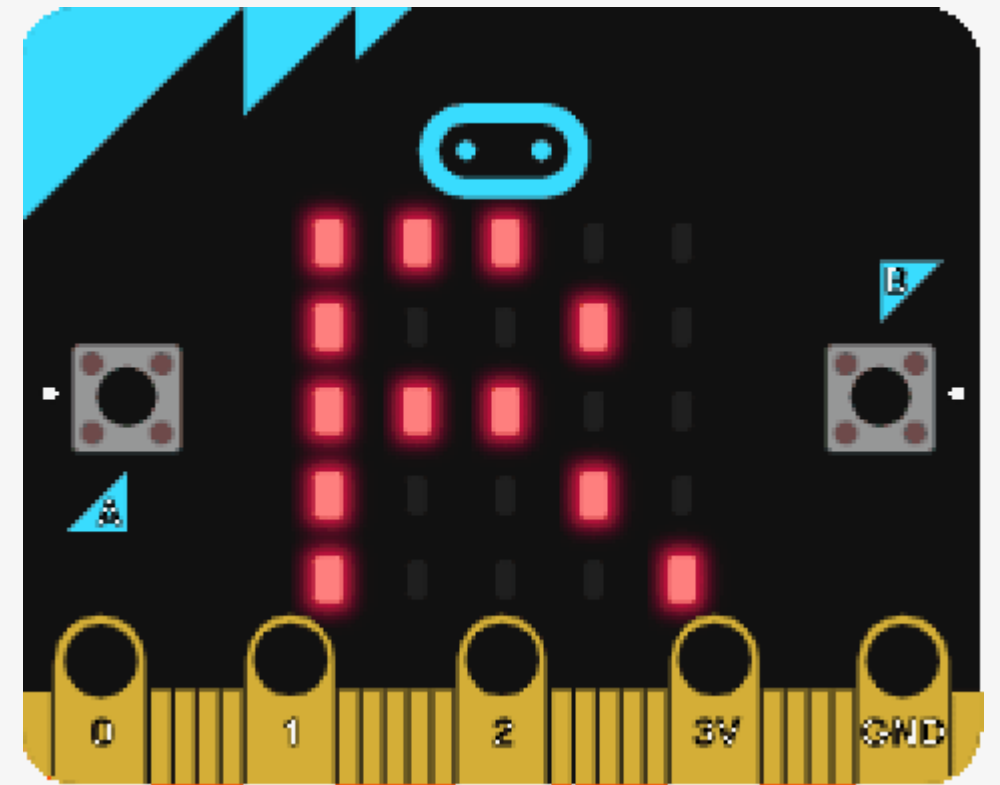


# Activity 1

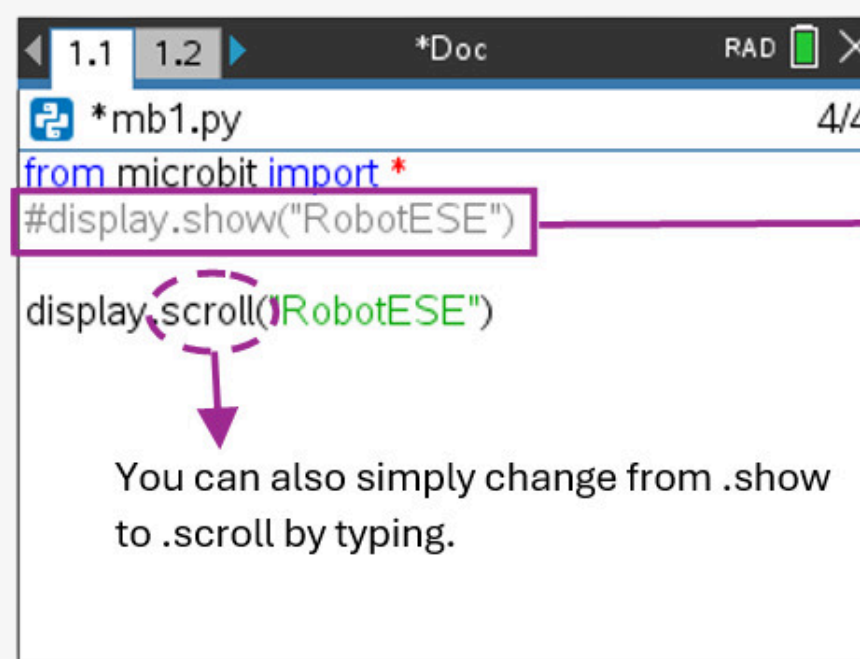
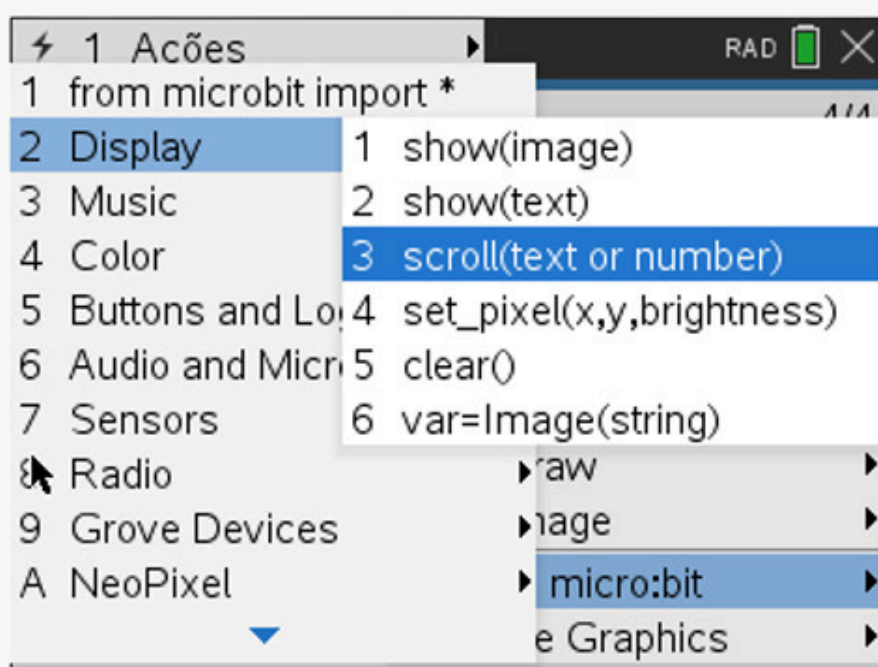
## What is your name?



When you start this program by pressing `ctrl + R`, you will see the letters of the expression entered appear one at a time on the display.



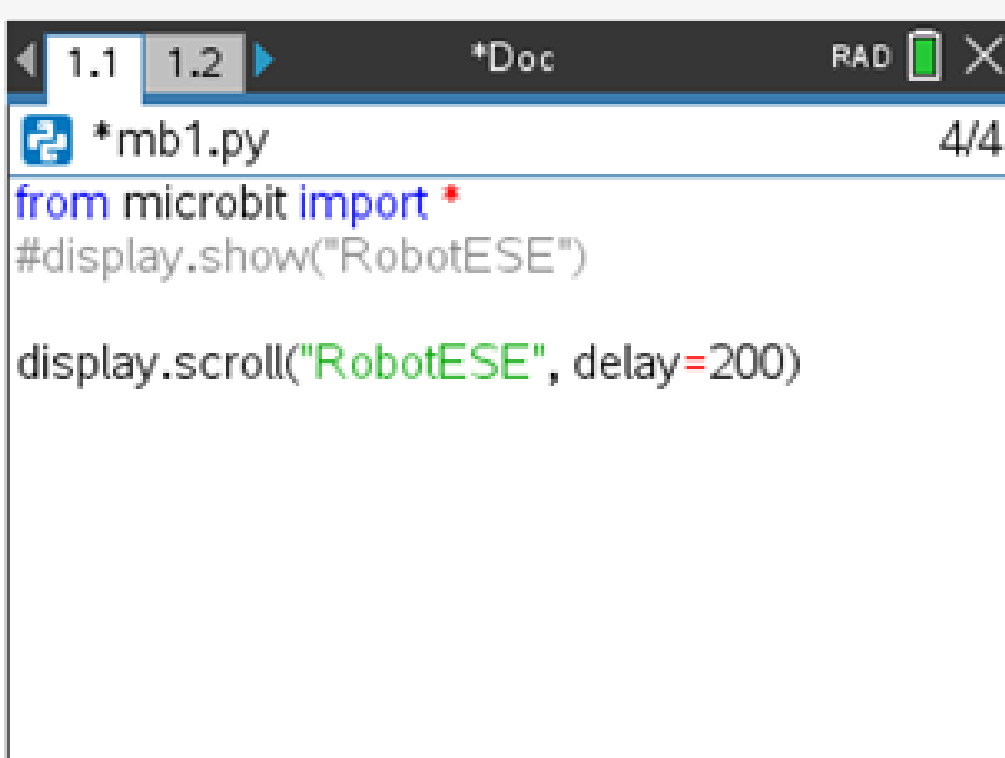
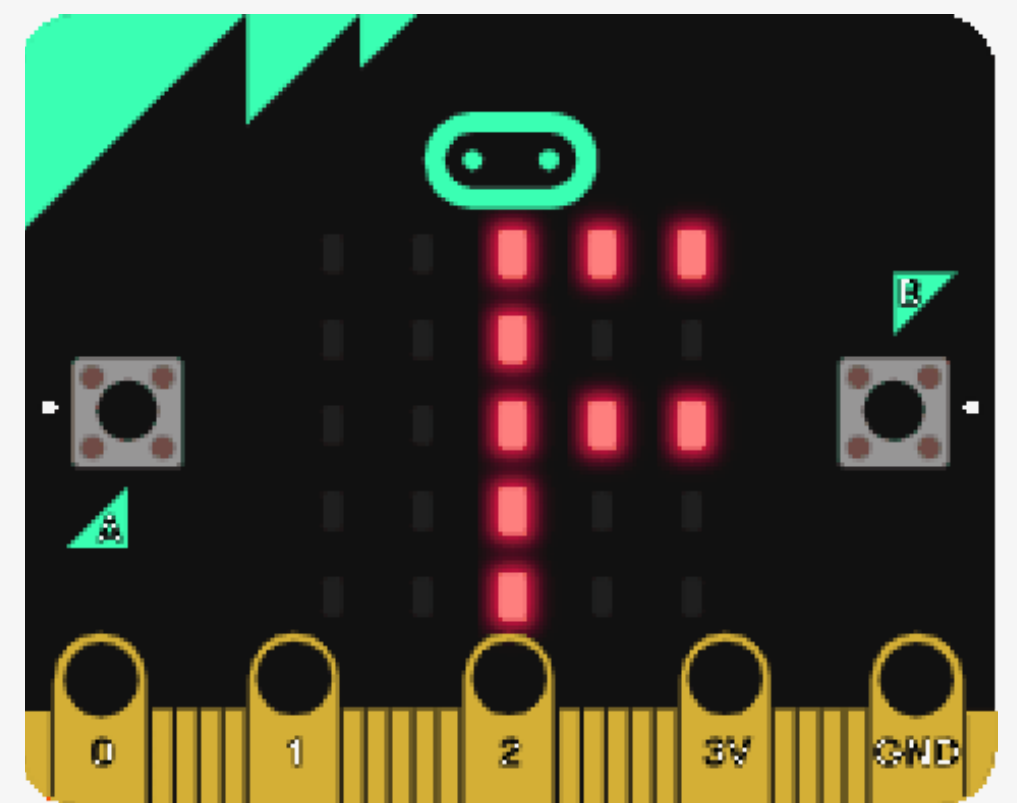
Another way to display the name or expression is using the command: **display.scroll("RobotESE")**.



This line is in #comment mode (to do this, place the cursor on this line and press `ctrl + T`) to deactivate it and then run the program again.

You can also simply change from .show to .scroll by typing.

This method causes the expression to move from right to left like a stripe across the display making it easier to read.



The speed at which the expression is moved can be controlled by adding the "**delay=**" parameter

This command causes a delay of, for example, 200 milliseconds (0.2 seconds) in the movement.

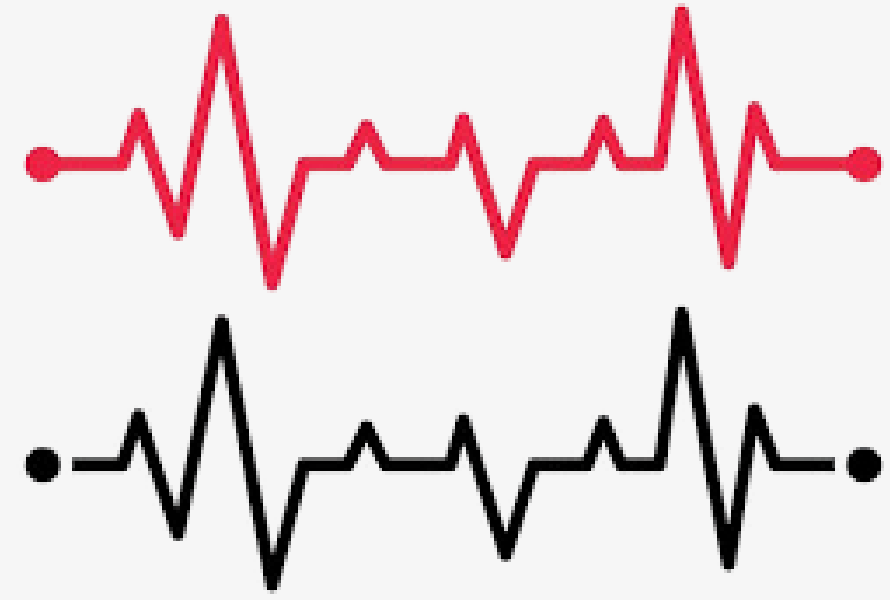
By accessing the [AI.tns](#) file, you can see the program analyzed here, either by reading the QR Code on the side or by clicking on it:





# Activity 2

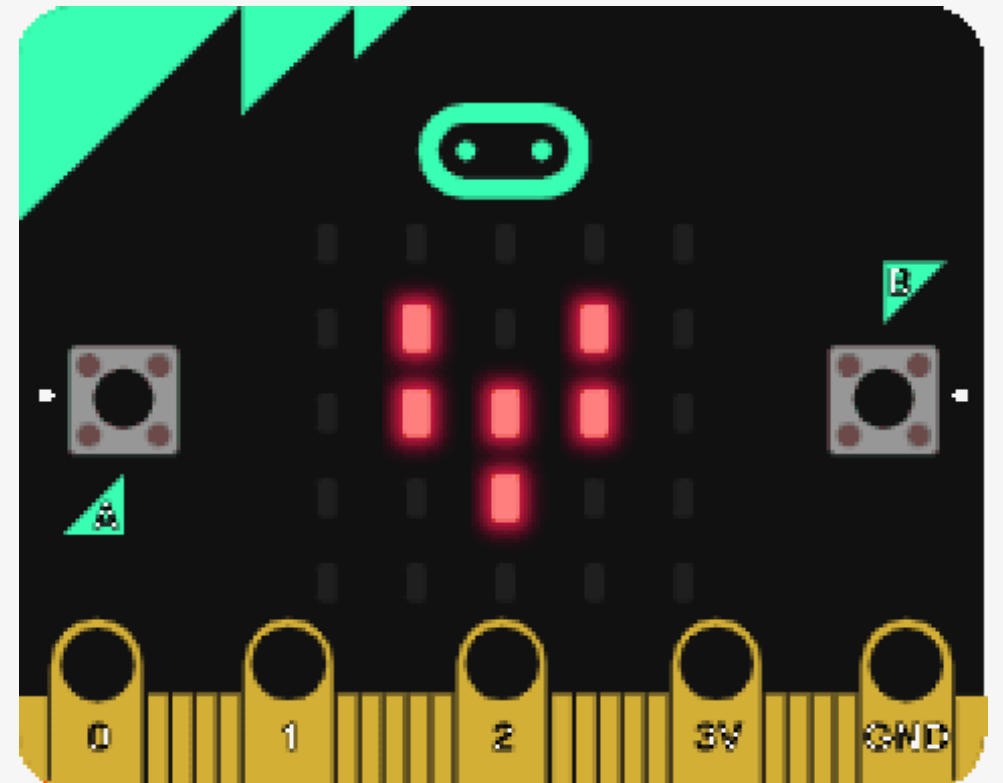
## The heartbeat



On a new page, we want to add a program that allows you to see a heart beating on the display. This effect corresponds to making two hearts (one smaller than the other) flash repeatedly, as shown in the following animation.

To recreate the effect, we need to build two hearts, one smaller than the other, to create the desired effect.

But how is it done? **It's easy!**



It is necessary to create two hearts, one larger than the other to then give the beating effect, to do this:

```

1.1 1.2 1.3 *Doc RAD
*mb3.py 3/3
from microbit import *

```

```

1 Acões bit RAD
1 from microbit import *
2 Display 1 show(image)
3 Music 2 show(text)
4 Color 3 scroll(text or number)
5 Buttons and Lo 4 set_pixel(x,y,brightness)
6 Audio and Micr 5 clear()
7 Sensors 6 var=Image(string)
8 Radio raw
9 Grove Devices iage
A NeoPixel micro:bit
e Graphics

```

```

1.1 1.2 1.3 *Doc RAD
*mb3.py 2/3
from microbit import *
var=Image("valor:""valor:""valor:""valor:""valor")

```

Each new image that we want to create must be placed in a variable so that we can later call it and make it appear on the micro:bit display.

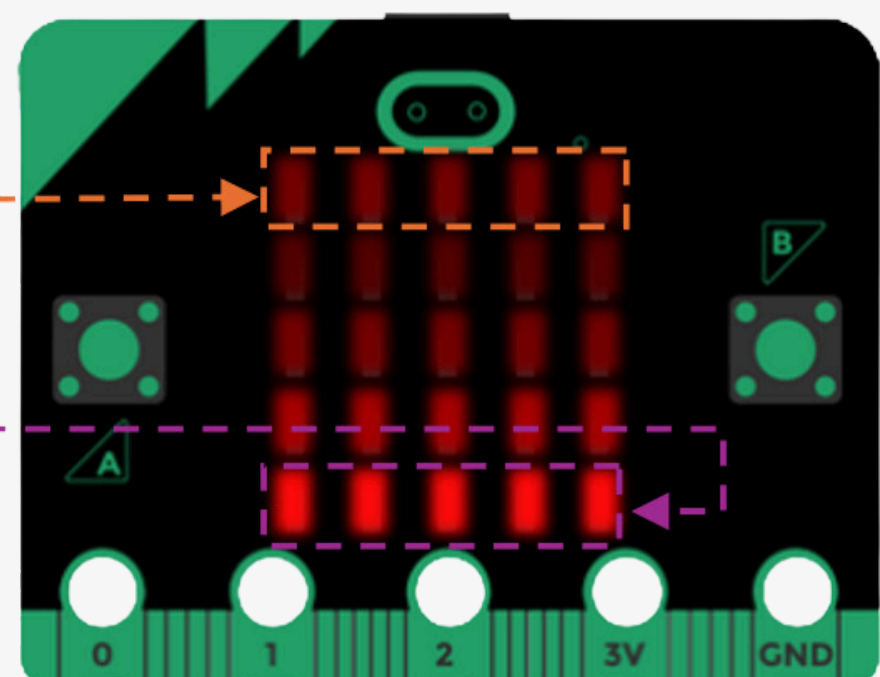
Each line of the display is represented by a line of five numbers. Each number specifies the brightness intensity for each of the pixels, which increases from 0 to 9. Each row of pixels is defined by 5 digits, each from 0 (off) to 9 (maximum intensity).

It is through this attribution of brightness to each pixel that a new image is created. In the figure you can see the last line of the display with a brighter color than the rest:

```

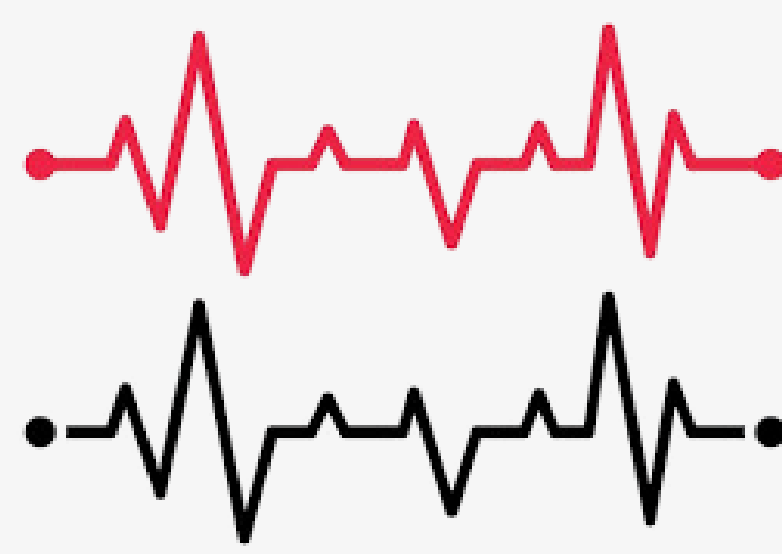
1.1 1.2 1.3 *Doc RAD
*mb3.py 6/7
from microbit import *
var=Image("33333:")
"22222:"
"33333:"
"55555:"
"99999:"

```

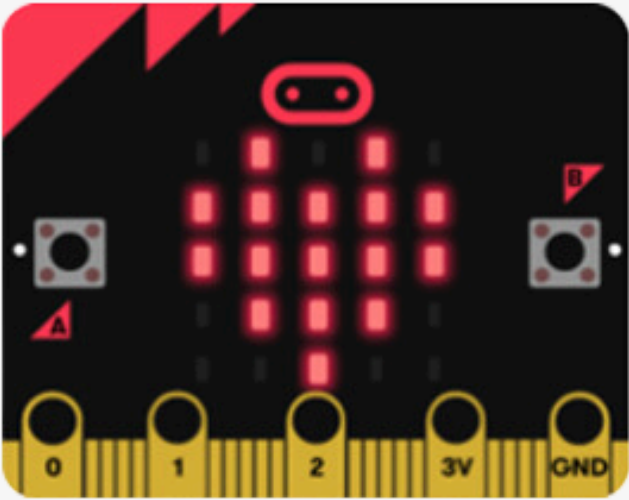


# Activity 2

## The heartbeat

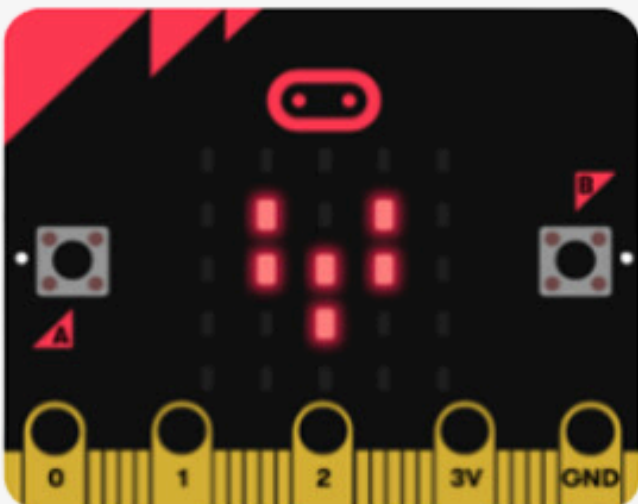


### Big heart



|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 9 | 0 | 0 | 0 |
| 9 | 9 | 9 | 9 | 9 |
| 9 | 9 | 9 | 9 | 9 |
| 0 | 9 | 9 | 9 | 0 |
| 0 | 0 | 9 | 0 | 0 |

### Small heart



|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 9 | 0 | 9 | 0 |
| 0 | 9 | 9 | 9 | 0 |
| 0 | 0 | 9 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

```

1.1 1.2 1.3 *Doc RAD 11/11
*mb3.py
from microbit import *
corG=Image("09090:"
"99999:"
"99999:"
"09990:"
"00900")
corP=Image("00000:"
"09090:"
"09990:"
"00900:"
"00000")
    
```

It is possible to write the previous program in different ways (below, 3 ways):

```

1.1 1.2 1.3 *Doc RAD 11/12
*mb3.py
from microbit import *
corG=Image("09090:"
"99999:"
"99999:"
"09990:"
"00900")
corP=Image("00000:"
"09090:"
"09990:"
"00900:"
"00000")
        
```

```

1.1 1.2 1.3 *Doc RAD 3/3
*mb3.py
from microbit import *
corG=Image("09090:""99999:""99999:""09990:""00900:")
corP=Image("00000:""09090:""09990:""00900:""00000:")
        
```

```

1.1 1.2 1.3 *Doc RAD 3/3
*mb3.py
from microbit import *
corG=Image("09090:99999:99999:09990:00900:")
corP=Image("00000:09090:09990:00900:00000:")
        
```

To display the symbols on the micro:bit display, use the command: **display.show(image)**

This command can be found at:

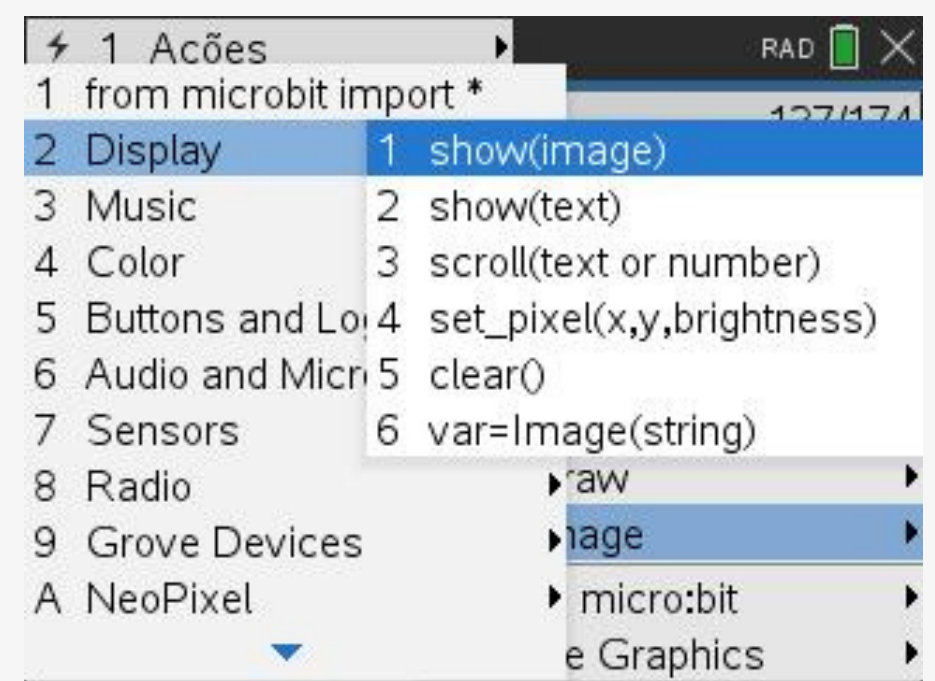
**menu** > A: More modules > 6: BBC micro:bit > 2: Display > 1: show (image)

The following program excerpt is obtained:

```

1.2 1.3 1.4 *Doc RAD 14/15
mb3.py
"99999:"
"09990:"
"00900")
corP=Image("00000:"
"09090:"
"09990:"
"00900:"
"00000")

display.show(corG)
display.show(corP)
    
```



### What will happen when the program starts running?

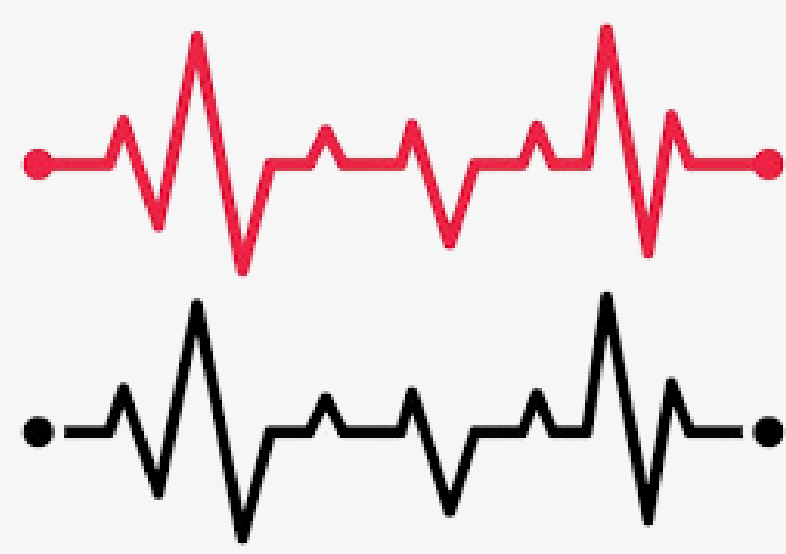
The largest heart will quickly be displayed followed by the smallest.

To make the two hearts flash repeatedly ('beating heart'), the two display commands must be included in a repeating structure.



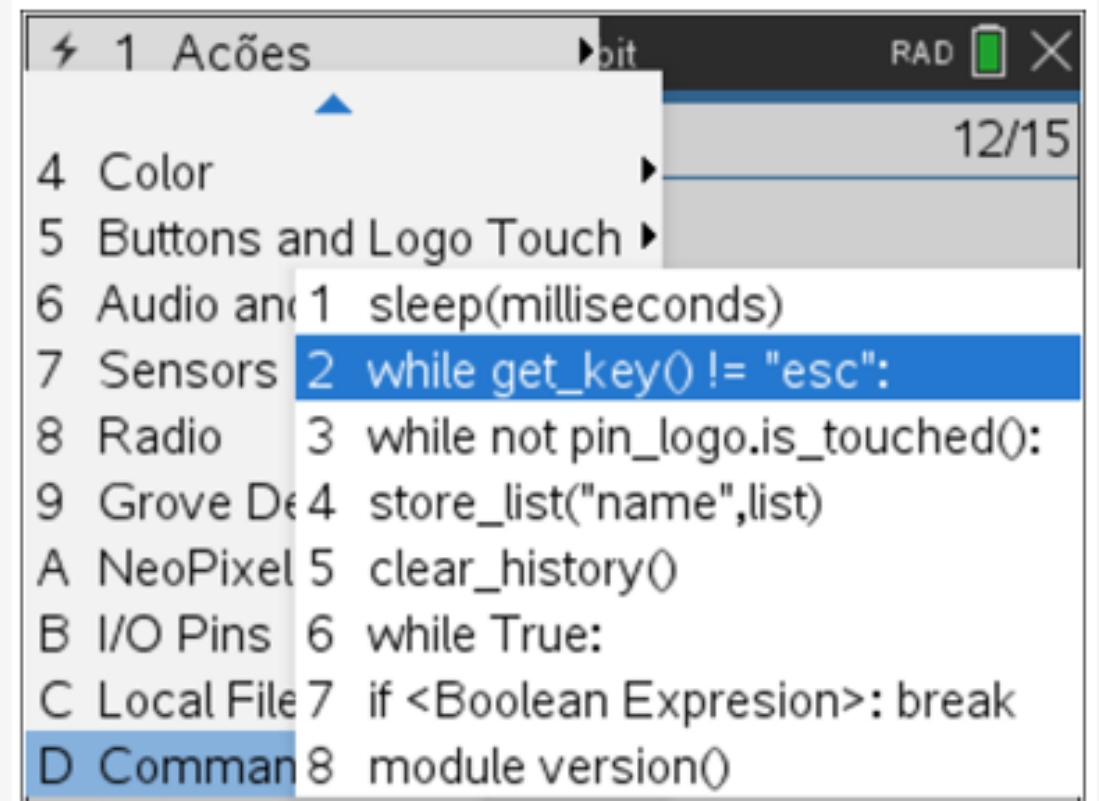
# Activity 2

## The heartbeat



The main idea is that the two images follow one after the other in the opposite order, that is, until a key is clicked, the program works. This instruction comes with the **while** loop, in which before the two display commands you must insert **while get\_key() != 'esc':** found in

`menu` > A: Mais módulos > 6: BBC micro:bit  
> D: Commands > 2: while get\_key() != 'esc'.



```
1.2 1.3 1.4 *Doc RAD 11/16
*mb3.py
....."00900")
corP=Image("00000:"
....."09090:"
....."09990:"
....."00900:"
....."00000")

while get_key() != "esc":
    display.show(corG)
    display.show(corP)
```

### ATTENTION!

Indentation is crucial in Python programming. This is how Python interprets blocks in repetition or conditional structures. If the two display commands do not have an indentation with the same number of spaces, a syntax error will occur.

Use the `[ ]` key or the `[tab]` key to indent each line.

**Now, you have to run the program. Watch your heart beat!**  
**To end the program, simply press the `[esc]` key.**

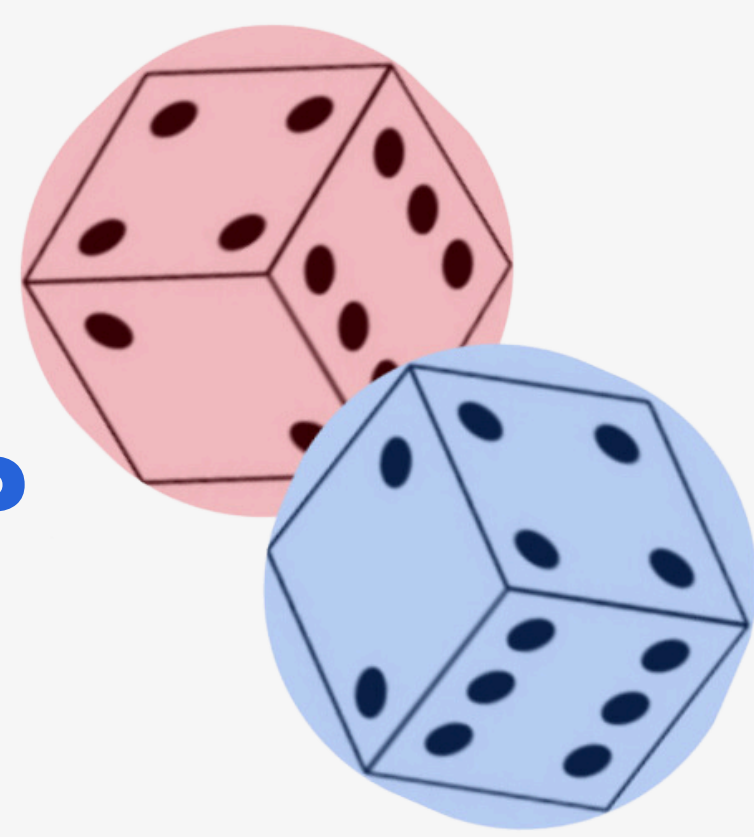
By accessing the [A2.tns](#) file, you can see the program analyzed here, either by reading the QR Code on the side or by clicking on it:





# Activity 3

## Which side will come out?



The aim is to simulate a dice roll, which requires:

- Randomly generate an integer from 1 to 6.
- Show on the display the image relating to the face that was exited.

### How to randomly generate integers from 1 to 6?

To randomly generate an integer from 1 to 6, you need to import, in addition to the usual modules, the random module.

This module can be found at:

`menu` > 6: Random > 1: from random import\*



```

1.1 1.2 1.3 *Doc RAD 1/4
*m1b.py 4/4
from microbit import *
from random import *

face=randint(1,6)
    
```

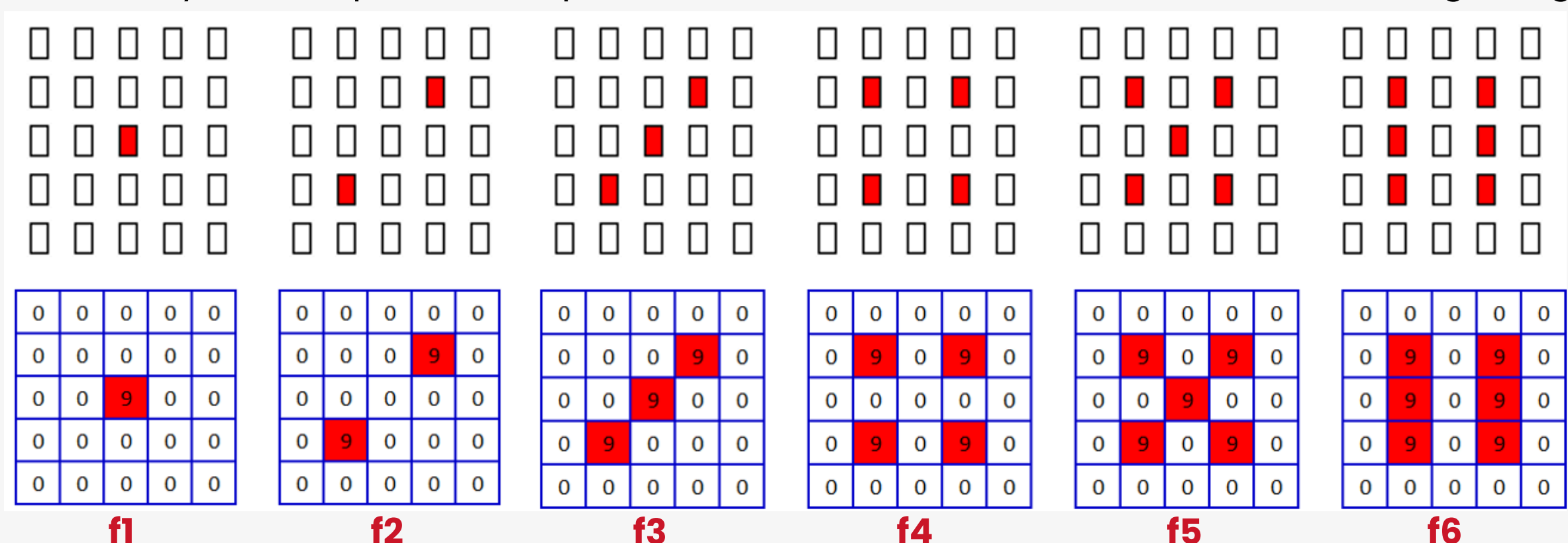
A random integer between 1 and 6 is assigned to a variable, here called a **'face'**.

The respective line of code to be inserted can be written using the calculator keyboard, but the **randint()** command can be found in

`menu` > 6: Random > 4: randint(min, max).

### How to show the face of the dice roll that came out on the display?

After generating the random number, the aim is to observe an image on the micro:bit display that simulates the face of the dice roll corresponding to the generated number. As already seen in previous experience, it can be considered as in the following image:



f1

f2

f3

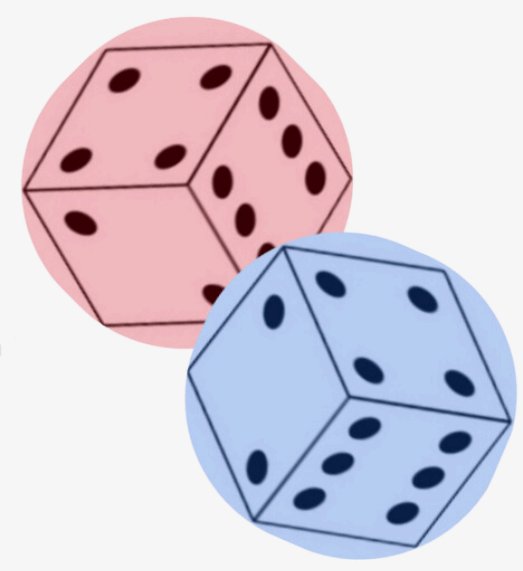
f4

f5

f6

# Activity 3

## What number will come out?



We thus have the following lines of code for each side of the die:

```
f1=Image("00000:00000:00900:00000:00000")
f2=Image("00000:00090:00000:09000:00000")
f3=Image("00000:00090:00900:09000:00000")
f4=Image("00000:09090:00000:09090:00000")
f5=Image("00000:09090:00900:09090:00000")
f6=Image("00000:09090:09090:09090:00000")
```

For this purpose, below is the algorithm in natural language and in Python language:

**If** the generated random number is 1:

Then: is shown on the display, f1;

**But if** the generated random number is 2:

Then: is shown on the display, f2;

**But if** the generated random number is 3:

Then: is shown on the display, f3;

**But if** the generated random number is 4:

Then: is shown on the display, f4;

**But if** the generated random number is 5:

Then: is shown on the display, f5;

**Otherwise:**

It appears on the display, f6.

```
if face == 1:
    display.show(f1)
elif face == 2:
    display.show(f2)
elif face == 3:
    display.show(f3)
elif face == 4:
    display.show(f4)
elif face == 5:
    display.show(f5)
else:
    display.show(f6)
```

The conditional structure 'if...elif...else...' is Python code for "If... but if... then...".

This structure can be found in [menu](#) > 4: Integrated Plans > 2: Control > 3: If... elif... else...

Therefore, here is a possible program for what you want:

```
from microbit import *
from random import *

face=randint(1,6)

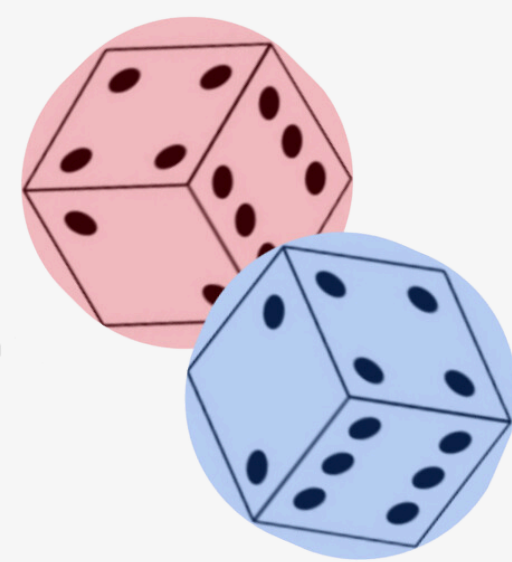
f1=Image("00000:00000:00900:00000:00000")
f2=Image("00000:00090:00000:09000:00000")
f3=Image("00000:00090:00900:09000:00000")
f4=Image("00000:09090:00000:09090:00000")
f5=Image("00000:09090:00900:09090:00000")
f6=Image("00000:09090:09090:09090:00000")

if face==1:
    display.show(f1)
elif face==2:
    display.show(f2)
elif face==3:
    display.show(f3)
elif face==4:
    display.show(f4)
elif face==5:
    display.show(f5)
else:
    display.show(f6)
```



# Activity 3

## What number will come out?



### Is it possible to simulate rolling the dice without using the

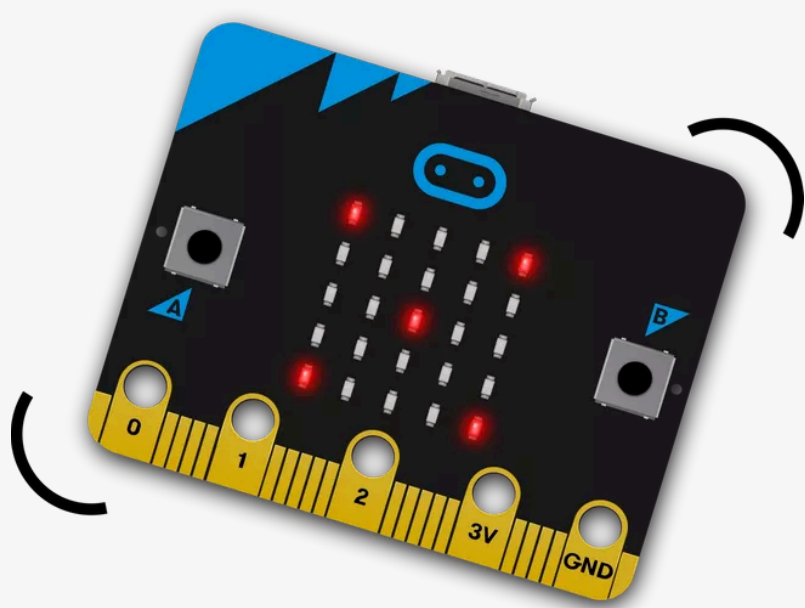
`ctrl` + `R` ?

Two alternative ways to simulate rolling a die without running the program with `ctrl` + `R`, can be:

1. Using the micro:bit motion sensor to simulate a gesture.
2. Using a micro:bit button.

Let's see below how to execute these alternatives:

#### 1. Using the micro:bit motion sensor to simulate a gesture.



The micro:bit has a motion sensor that allows it to be recognized and used, for example, when it is being shaken or turned upside down or down.

To start, given that the launch takes place whenever the user wants, you have to enter the cycle **while `get_key() != 'esc'`** (already mentioned how this command can be found).

As the process depends on whether or not the micro:bit is being shaken, you must enter the line with the following code:

```
if accelerometer.was_gesture('shake')
```

The command

```
accelerometer.was_gesture('shake')
```

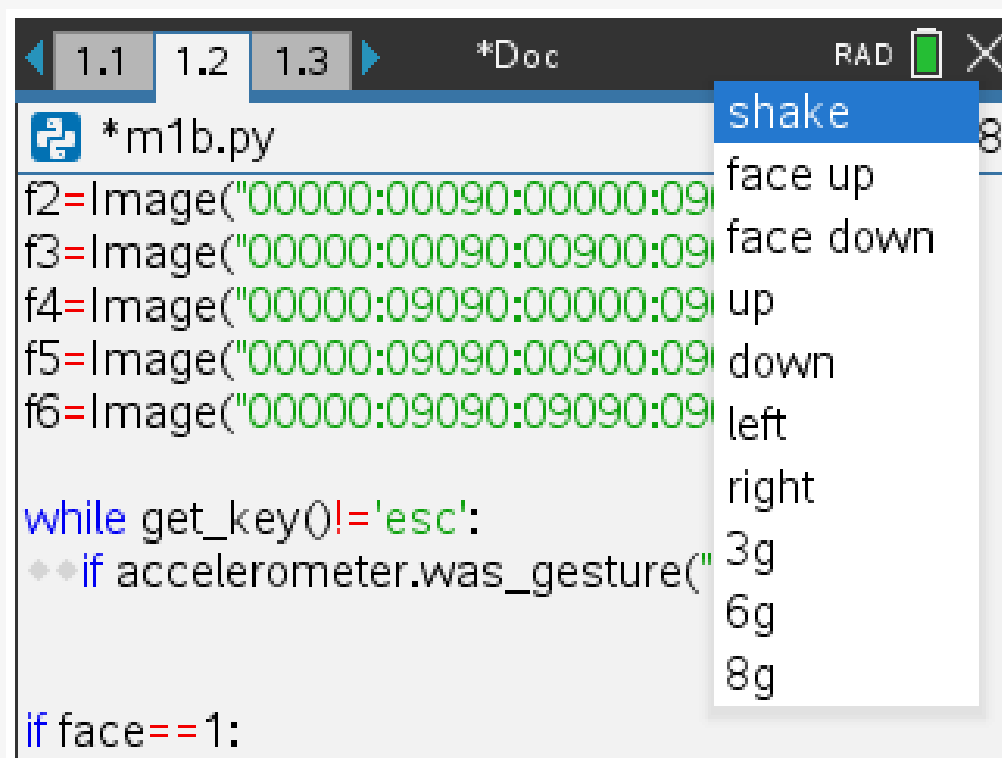
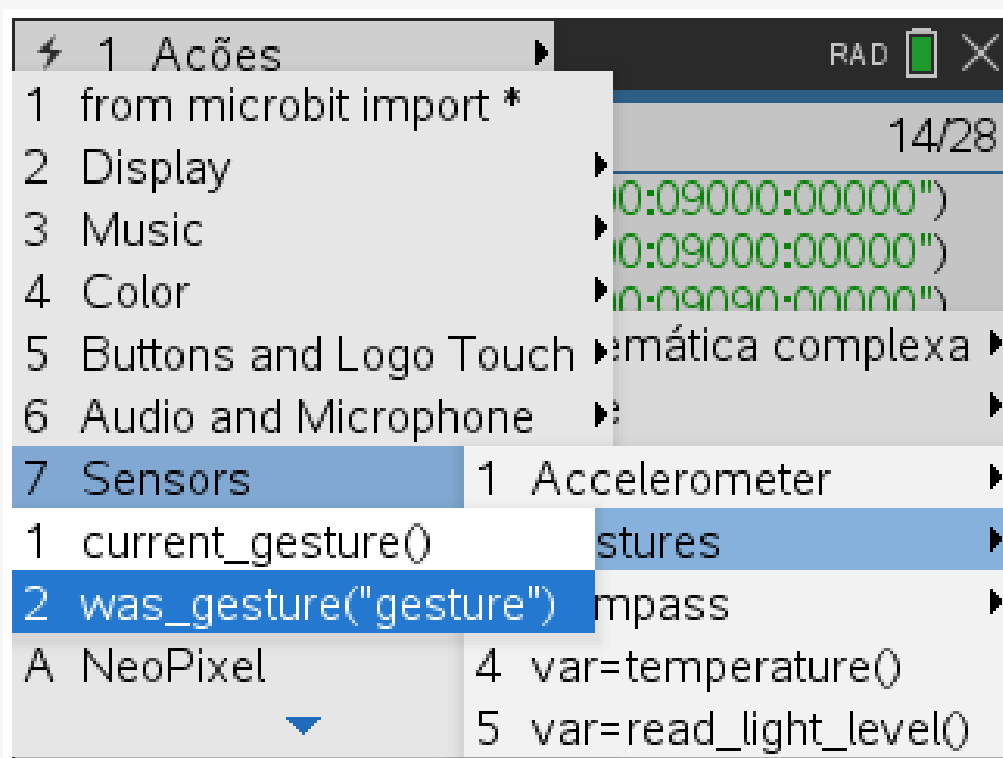
can be found at:

```
menu > A: More modules > 6: BBC micro:bit > 7: Sensors  
> 2: Gestures > 2: was_gesture("gesture")
```

```
1.1 1.2 1.3 *Doc RAD 14/28
*m1b.py
face=randint(1,6)

f1=Image("00000:00000:00900:00000:00000")
f2=Image("00000:00090:00000:09000:00000")
f3=Image("00000:00090:00900:09000:00000")
f4=Image("00000:09090:00000:09090:00000")
f5=Image("00000:09090:00900:09090:00000")
f6=Image("00000:09090:09090:09090:00000")

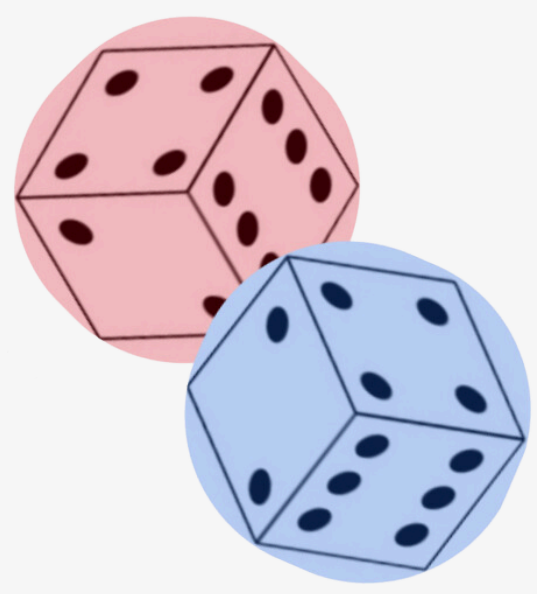
while get_key()!='esc':
```





# Activity 3

## What number will come out?



### 1. Using the micro:bit motion sensor to simulate a gesture.

If the micro:bit “feels” that it is being shaken, it will show on the display a number of dots corresponding to the face of the die.

To clear the display, use the **display.clear()** command.

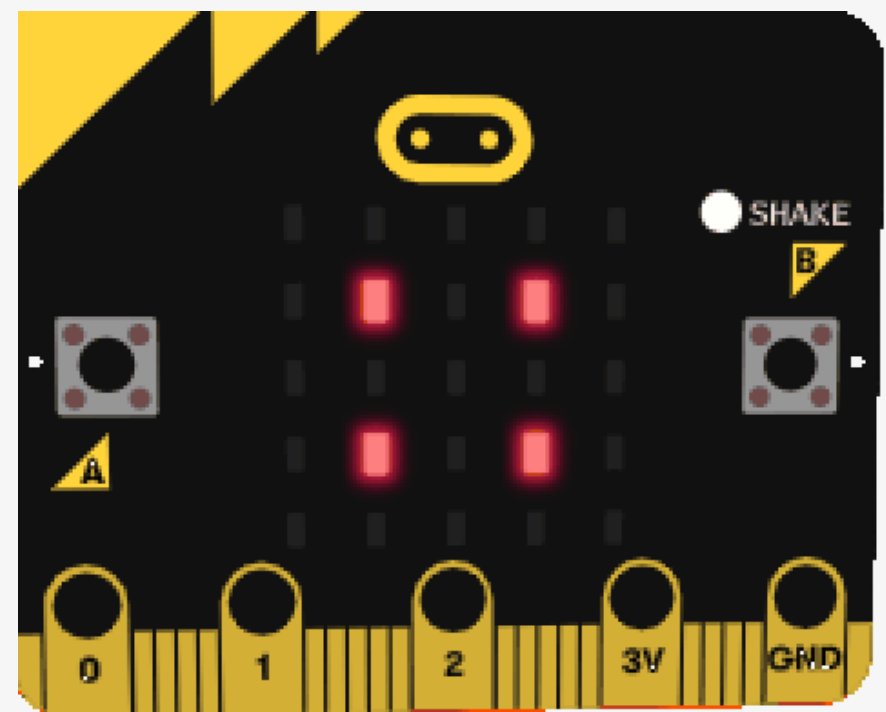
Thus, we have the program with the intended effect:

```
from microbit import *
from random import *

f1=Image("00000:00000:00900:00000:00000")
f2=Image("00000:00090:00000:09000:00000")
f3=Image("00000:00090:00900:09000:00000")
f4=Image("00000:09090:00000:09090:00000")
f5=Image("00000:09090:00900:09090:00000")
f6=Image("00000:09090:09090:09090:00000")

while get_key()!='esc':
    if accelerometer.was_gesture("shake"):
        display.clear()
        face=randint(1,6)
        if face==1:
            display.show(f1)
        elif face==2:
            display.show(f2)
        elif face==3:
            display.show(f3)
        elif face==4:
            display.show(f4)
        elif face==5:
            display.show(f5)
        else:
            display.show(f6)
```

```
1.1 1.2 1.3 *Doc RAD 16/26
*m1b.py
while get_key()!='esc':
    if accelerometer.was_gesture("shake")
    display.clear()
    face=randint(1,6)
    if face==1:
        display.show(f1)
    elif face==2:
        display.show(f2)
    elif face==3:
        display.show(f3)
```



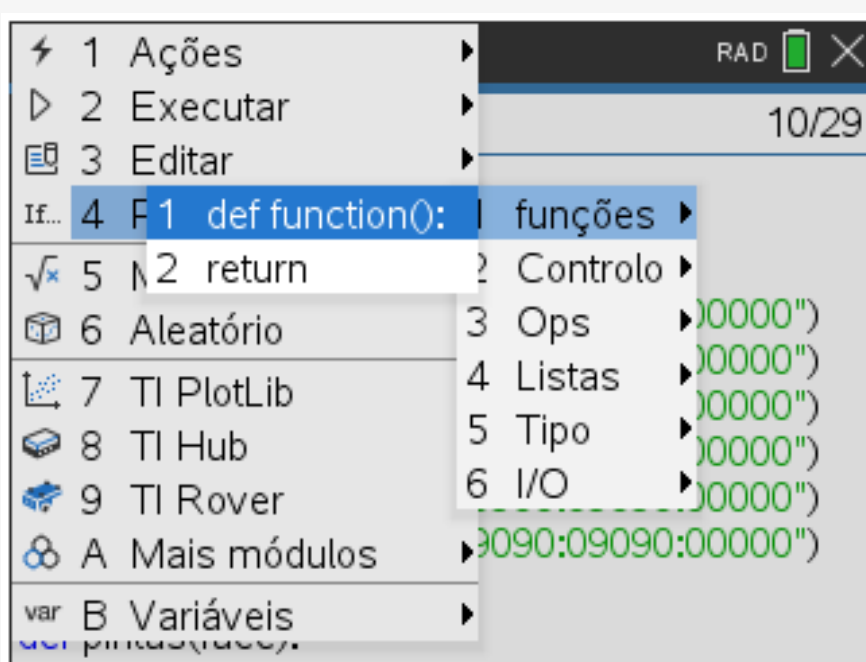
We can reorganize the code so as not to have so many lines in the **while** loop, by creating a function (subroutine) that will be called “pintas”:

```
from microbit import *
from random import *

f1=Image("00000:00000:00900:00000:00000")
f2=Image("00000:00090:00000:09000:00000")
f3=Image("00000:00090:00900:09000:00000")
f4=Image("00000:09090:00000:09090:00000")
f5=Image("00000:09090:00900:09090:00000")
f6=Image("00000:09090:09090:09090:00000")

def pintas(face):
    if face==1:
        display.show(f1)
    elif face==2:
        display.show(f2)
    elif face==3:
        display.show(f3)
    elif face==4:
        display.show(f4)
    elif face==5:
        display.show(f5)
    else:
        display.show(f6)

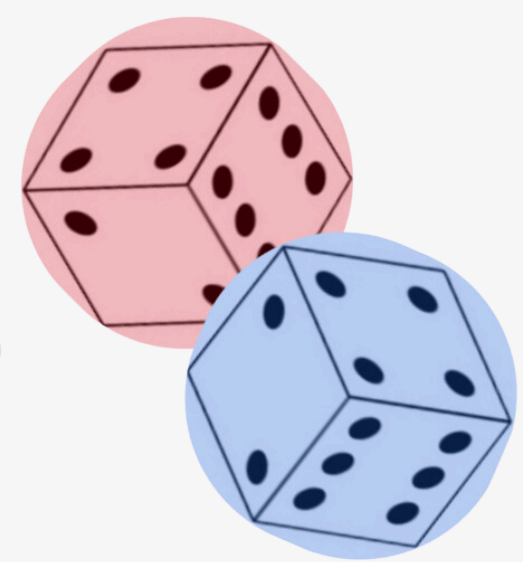
while get_key()!='esc':
    if accelerometer.was_gesture("shake"):
        display.clear()
        face=randint(1,6)
        pintas(face)
```



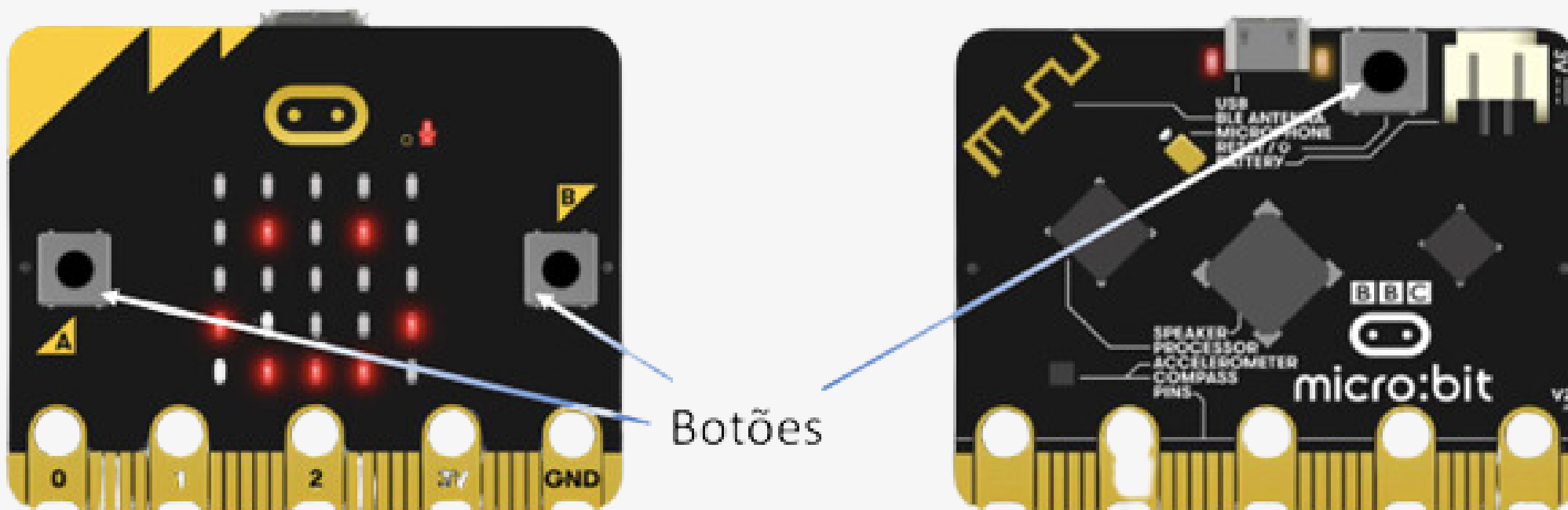
**For example:**  
if face=1 then pintas(face)=pintas(1)  
and face f1 will appear on the display  
(only with one dot)

# Activity 3

## What number will come out?



### 2. Use of a micro:bit button.



The idea is similar to the previous case, but instead of the micro:bit “sensing” that it is being shaken, it reacts to pressing button A or button B.

Thus, we have to:

```
from microbit import *
from random import *

f1=Image("00000:00000:00900:00000:00000")
f2=Image("00000:00090:00000:09000:00000")
f3=Image("00000:00090:00900:09000:00000")
f4=Image("00000:09090:00000:09090:00000")
f5=Image("00000:09090:00900:09090:00000")
f6=Image("00000:09090:09090:09090:00000")

def pintas(face):
    if face == 1:
        display.show(f1)
    elif face == 2:
        display.show(f2)
    elif face == 3:
        display.show(f3)
    elif face == 4:
        display.show(f4)
    elif face == 5:
        display.show(f5)
    else:
        display.show(f6)

while get_key() != 'esc':
    if button_a.is_pressed():
        display.clear()
        face=randint(1,6)
        pintas(face)
    if button_b.is_pressed():
        display.clear()
        face=randint(1,6)
        pintas(face)
```

By accessing the **A3.tns** file you can see the programs analyzed here and a possible response to the proposed challenge, or by reading the QR Code below or clicking on it:



### Challenge:

Two dice are rolled and the numbers of the faces that come up are added. The challenge consists of adapting the previous program, in which Button B is used to display, on the display, the sum of the spots on the two dice.

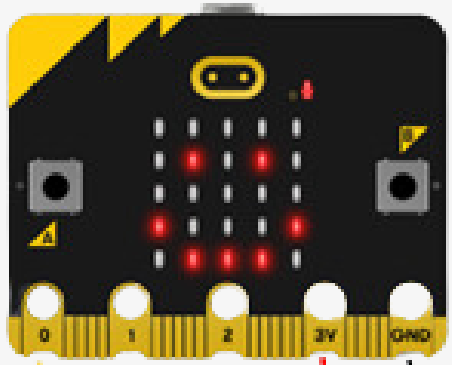
To answer to the challenge, we suggest that you consider the following questions:

1. How many sums are there?
2. Since the sum number is to be displayed on the display, the faces to be displayed must be pre-defined. How to make?
3. If to simulate a launch it was necessary to generate a random number. What about rolling two dice?



# Activity 4

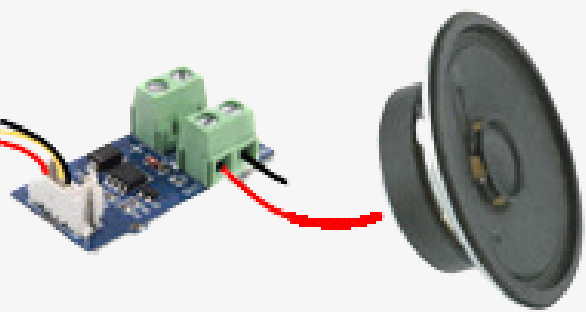
## m&m - micro:bit and music



The micro:bit allows you to emit sounds, play music and has an integrated speaker.

To hear the sounds, connect the micro:bit to headphones or an amplified speaker to pins **0** and **GND** of the micro:bit, at least.

Pin 0 works as an output when music plays on the micro:bit.



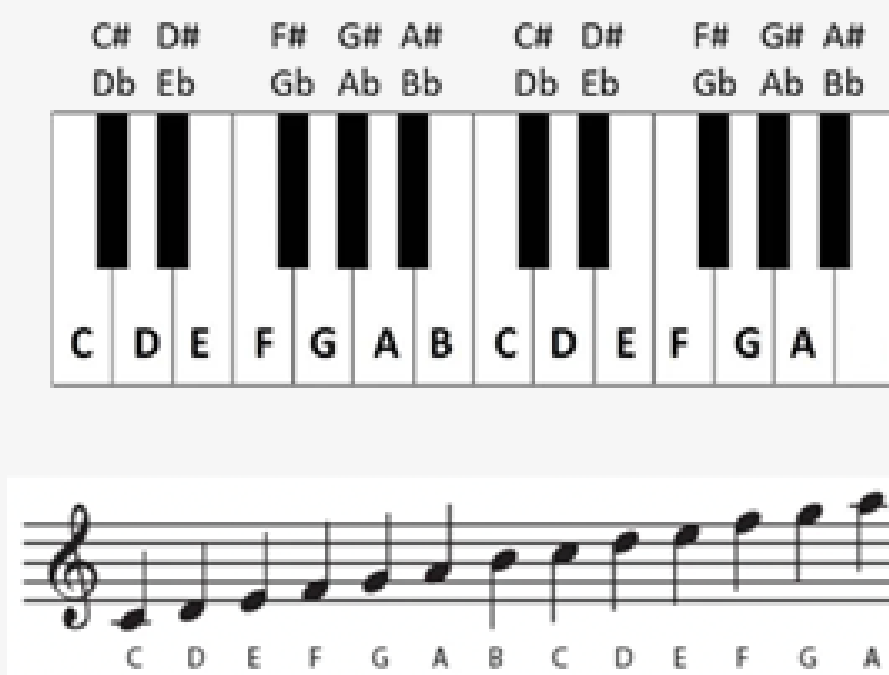
### 1 Play Dó-Ré-Mi.

By matching musical notes with lyrics, any musical excerpt that you want to play on the micro:bit must be placed in a list.

This list must be assigned to a variable, so that, when it is 'called', the micro:bit knows what to play.

In this list, the letters corresponding to the musical notes are placed, in order. The command that allows you to play is **music.play(music)** which can be found at:

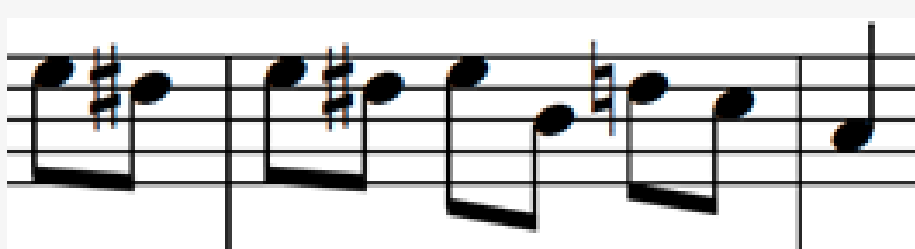
[menu](#) > A: More modules > 6: BBC micro:bit > 3: Music > 3: play(song)



```

1.2 1.3 1.4 *Doc RAD
*musica.py 4/5
from microbit import *
doremi=['C','D','E','F','G','A','B','C']
music.play(doremi)
    
```

### 2 Play an excerpt of a song.



This command controls the speed of the song (the first parameter) and the number of beats per minute (the second parameter). It can be found at:

```

1.2 1.3 1.4 *Doc RAD
*musica.py 6/6
from microbit import *
música=['E5','D#','E','D#','E','B4','D5','C','A4:8']
music.set_tempo(4,250)
music.play(música)
    
```

[menu](#) > A: More modules > 6: BBC micro:bit > 3: Music > 5: set\_tempo(ticks, BPM)



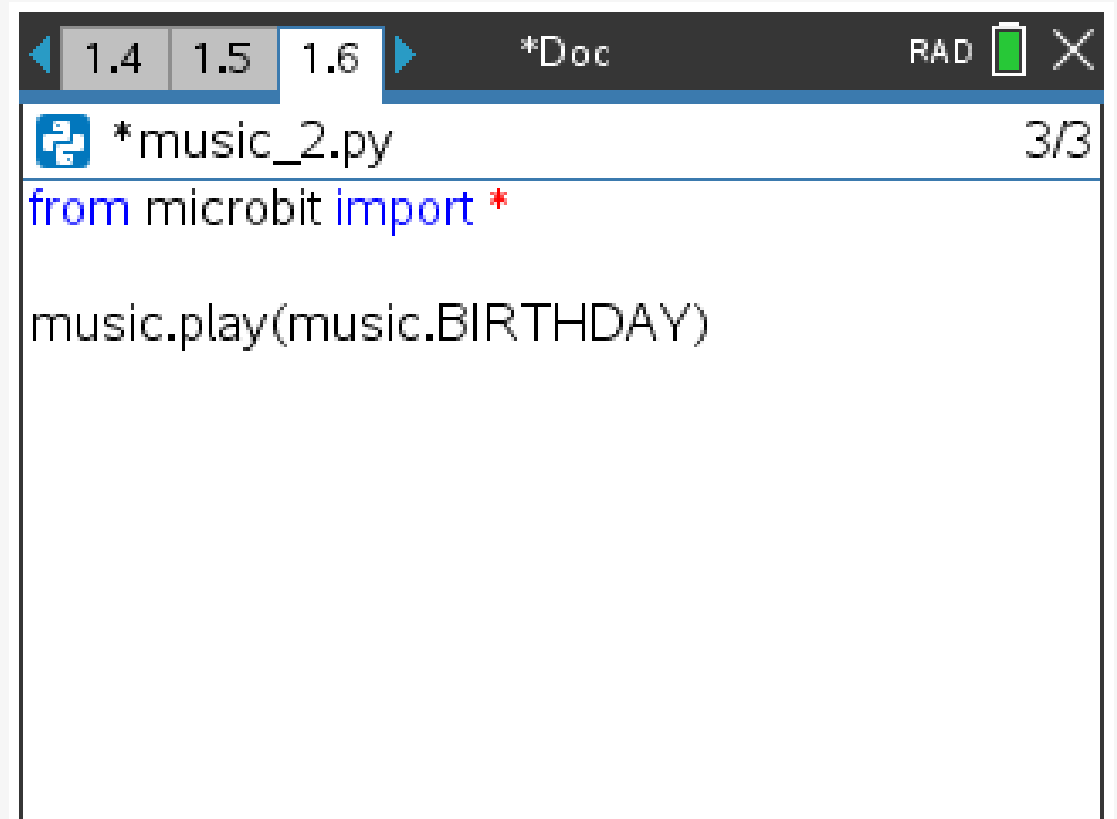
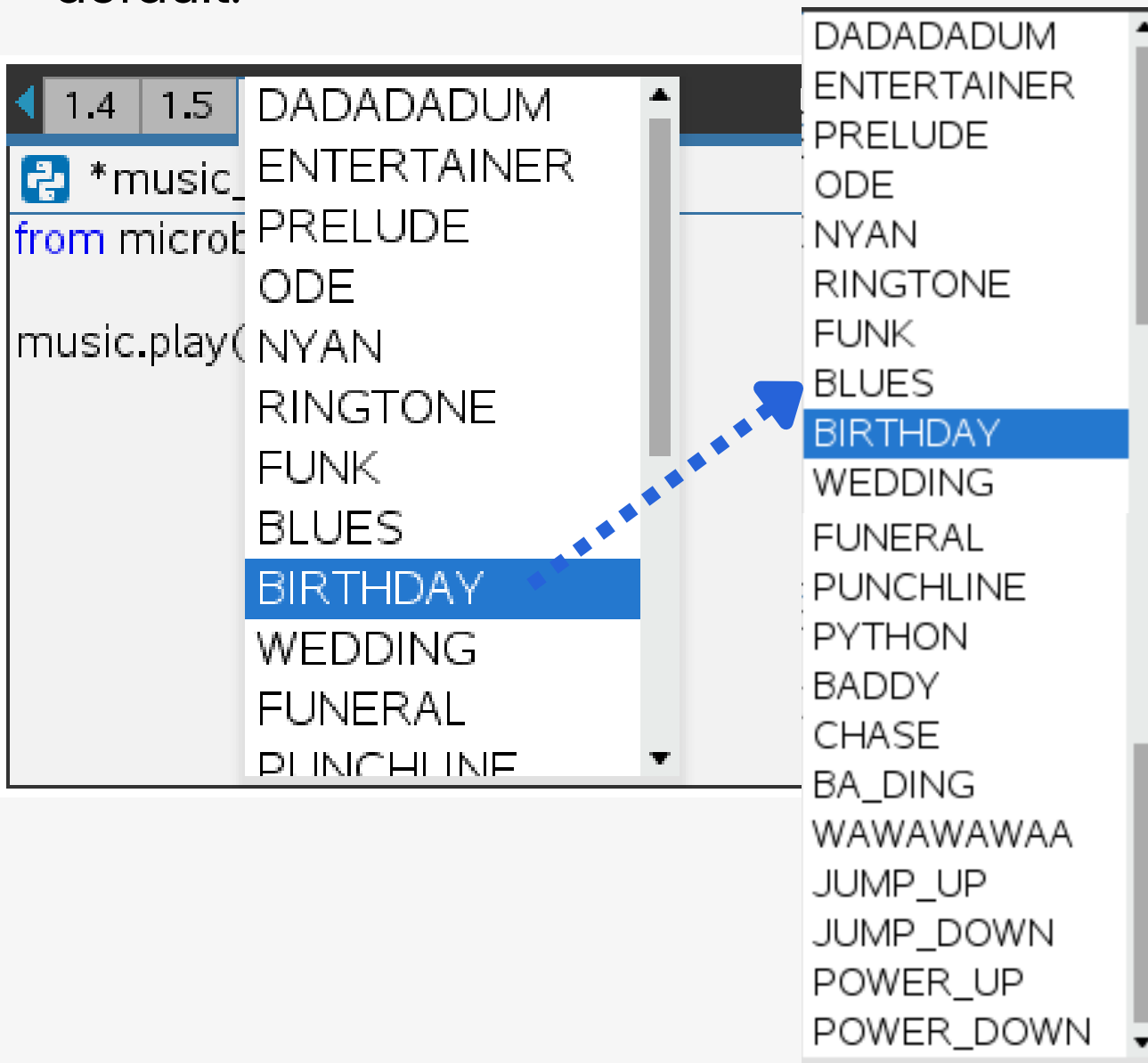


# Activity 4

## m&m - micro:bit and music

### 3 "Happy Birthday to You..."

The micro:bit already has songs created by default:



### 4 What national anthem to play?

In sporting competitions, there is a tradition of hearing the national anthem of the gold medal-winning country at each medal ceremony.

Let's imagine that we are in a sports championship that features three countries: Türkiye, Finland and Portugal.

The intention is to play the anthem and raise the flag of the country that won the gold medal.

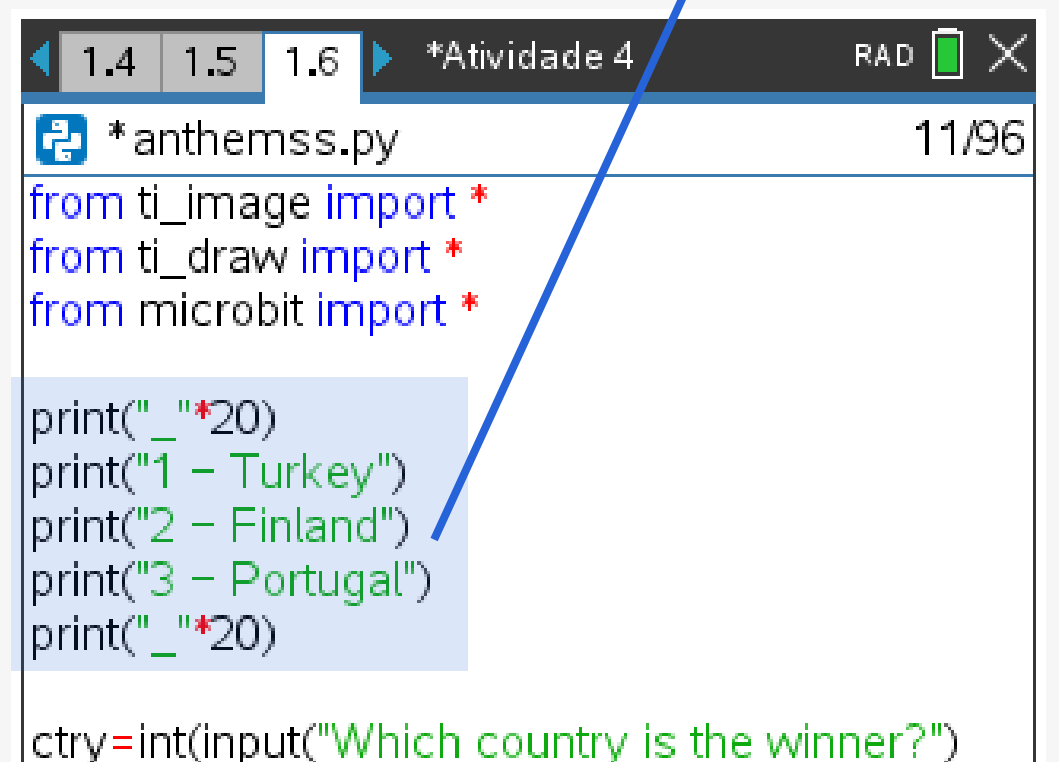
To do this, in addition to the microbit module, you need the **ti\_image** and **ti\_draw** modules, which allow, when requested, to show the country's flag.

These commands can be found at:

**menu** > A: Mais módulos > 4: TI Draw **or** 3: TI Image  
> 1: from ti\_draw import \* **or** 1: from ti\_image import\*



These first lines allow you to identify the possible countries that are in the championship and have a chance of reaching the podium.



# Activity 4

## m&m - micro:bit and music



### 4 What national anthem to play?

The organizers of the sporting event, to make the procedure of showing the flag and playing the anthem more effective, thought of associating each country with a number, so that they could now answer the question “**Which country is the winner?**”.

To do this, place the line of code

```
ctry=int(input("Which country is the winner?"))
```

For this input to be assigned to a variable, which must be given a name, it must be identified that it is an integer numeric value, with the int function.

This function allows you to receive data from the user.

```
*Atividade 4
RAD
*anthemss.py 11/96
from ti_image import *
from ti_draw import *
from microbit import *

print("_"*20)
print("1 - Turkey")
print("2 - Finland")
print("3 - Portugal")
print("_"*20)

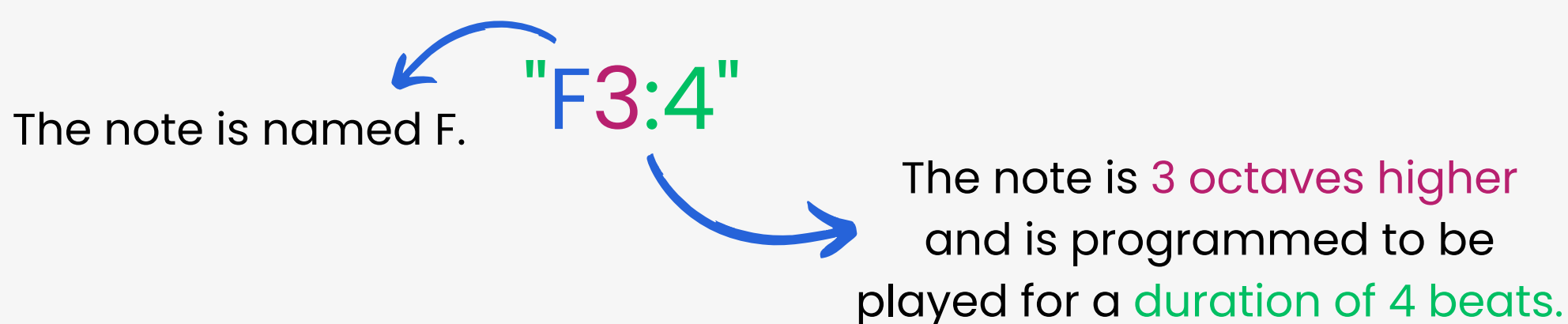
ctry=int(input("Which country is the winner?"))
```

To obtain the input command, you can write using the keyboard or, from the menu:

**menu** > A: More modules > 2: Integrated plans > 6: I/O > 2: input()

(To write words, letters or text, as in this case, that are not variables, you must put quotation marks ".")

As you can see, each note has a **name** (like C# or F), an **octave** (how high or low the note should be played) and a **duration** (how long it lasts over time). For example:



**Octaves** are indicated by a number: 0 is the lowest octave, 4 contains middle C, and 8 is the highest octave.

**Durations** are also expressed as numbers. The higher the duration value, the longer it will last.

If the note name **R** is used, the micro:bit will “play” a rest (i.e., silence) for the specified duration.

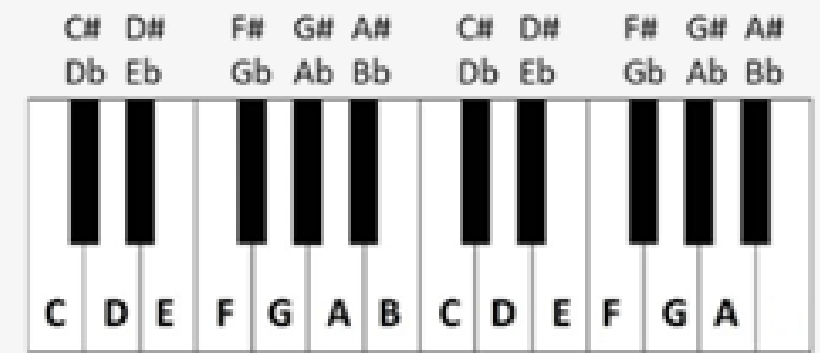
# Activity 4

## m&m - micro:bit and music



### 4 What national anthem to play?

The correspondence of the musical notes with the lyrics for each national anthem is as follows:



#### National Anthem of Portugal – “A Portuguesa”



```
anthem_PT=["Bb4:1","G:3","Eb:1","C5:8", "Bb4:7","C5:1","Bb4:4","Ab4:4","G4:3","Eb4:1","D4:3" "Eb4:1",  
"G4:8","A4:3","Bb4:1","C5:3","D5:1","Bb4:10", "R:2","Bb4:3","D5:1","F5:8","D5:4","Bb4:3","Ab4:1","Ab4:8",  
"G4:3","Bb4:3","G4:1","Eb4:8","Bb3:1","Eb4:3","F4:1","G4:3","Bb4:1","F4:12","R:4","F4:7","G4:1","Eb4:4","Eb4:4",  
"Ab4:7","Bb4:1","G4:3","G4:3","Bb4:1","Eb5:8","Bb4:4","Db5:3","C5:1","Bb4:4","F4:9","R:2","F4:1","Gb4:4","Bb4:4",  
"Bb4:4","Ab4:3","Gb4:1","F4:9","R:2","F4:1","Gb4:4","Bb4:4","Bb4:4","Ab4:3","Gb4:1","Bb4:9","R:4","Bb4:1",  
"Eb5:8","D5:7","Bb4:1","C5:8","Bb4:3","R:1","Eb4:3","F4:1","G4:4","G4:4","Bb4:", "G4:4","F4:12","R:3","Bb4:1",  
"Eb5:8","D5:7","Bb4:1","C5:8","Bb4:3","R:1","Bb4:3","Eb5:1","G5:8","F5:8","Eb5:9","G4:2","Ab4:2","Bb4:2",  
"C5:4","C5:4","Bb4:5","R:1","Bb4:1","Eb5:9"]
```

#### National Anthem of Türkiye – “İstiklâl Marşı”



```
anthem_TR=["B3:5","B3:1","B3","B3:5","B3:1","B3","B3:2","B","B","B","B","R:2","B2:4","E3","F#3","G","D#:3","F#:1",  
"E:11","R:1","E3:4","A","B","C4:3","B3:1","G#:3","B:1","A:12","B:2","A#:1","B","F#:4","F#:7","A:1","G:3","D#:1","E:3","F#:1",  
"G:3","A:1","B:3","C4:1","D:3","E:1", "D:4","D3:2","C#:1","D","B:4","A","G:12","B2:2","A#:1","B","F#3:4","B2","B3:3","A:1", "G:2",  
"F#:1","G:1","E:4","E4:7","D:1","C:3","B3:1","A:3","G:1","F#3:3","E:1","B:4","B2","E3:12","B2:4","E3","F#3","G","D#:3",  
"F#3:1","E:12","E:4","A","B","C4:3","B3:1","G#:3","B:1","A:12","B:2","A#:1","B","F#3:4","F#:8","A:1","G:3","D#:1","E:3",  
"F#3:1","G:3","A:1","B:3","C4:1","D:3","E:1","D:4","D3:2","C#:1","D","B:4","A","G:12","B2:2","A#:1","B","F#3:4","B2","B3:3",  
"A:1","G:2","F#3:1","G:1","E:4","E4:7","D:1","C:3","B3:1","A:3","G:1","F#3:3","E:1","B:4","B2","E3:12", "B2:4"]
```

#### National Anthem of Finland – “Vårt land”



```
anthem_FIN=["F3:2","Bb:3","F:1","D:2","F:2","Bb:2","C4:2","D:8","Bb3:4","G:3","C4:1","Bb3:4","A:4","Bb:4","R:4",  
"F3:2","D:2","Eb:2","F:6","Bb:2","C4:3","F3:1","D4:8","Bb3:4","G:3","C4:1","Bb3:4","A:4","Bb:5","R:3","F3:4","C4:3",  
"Bb3:1","A:2","G:2","F:2","Eb:2","D:2","G:2","F:4","R:2","F3:4","C4:3","Bb3:1","A:2","G:2","F:2","Eb:2","D:2","G:2","F:4",  
"R:2","F3:2","Bb:3","F:1","D:2","F:2","Bb:2","C4:2","D:8","Bb3:4","G:3","C4:1","Bb3:4","A:4","Bb:5","R:3","F3:4","C4:3",  
"Bb3:1","A:2","G:2","F:2","Eb:2","D:2","G:2","F:4","R:2","F3:4","C4:3","Bb3:1","A:2","G:2","F:2","Eb:2","D:2","G:2","F:4",  
"R:2","F3:2","Bb:3","F:1","D:2","F:2","Bb:2","C4:2","D:8","Bb3:4","G:3","C4:1","Bb3:4","A:4","Bb:4","R:4","F3:2","D:2",  
"Eb:2","F:6","Bb:2","C4:3","F3:1","D4:8","Bb3:4","G:3","C4:1","Bb3:4","A:4","Bb:5","R:3","F3:4","C4:3","Bb3:1","A:2",  
"G:2","F3:2","Eb3:2","D:2","G:2","F:4","R:2","F3:4","C4:3","Bb3:1","A:2","G:2","F:2","Eb3:2","D:2","G:2","F:4","R:2",  
"F3:2","Bb:3","F:1","D:2","F:2","Bb:2","C4:2","D:8","Bb3:5","G:5","C4:2","Bb3:5","A:5","Bb:8"]
```



# Activity 4

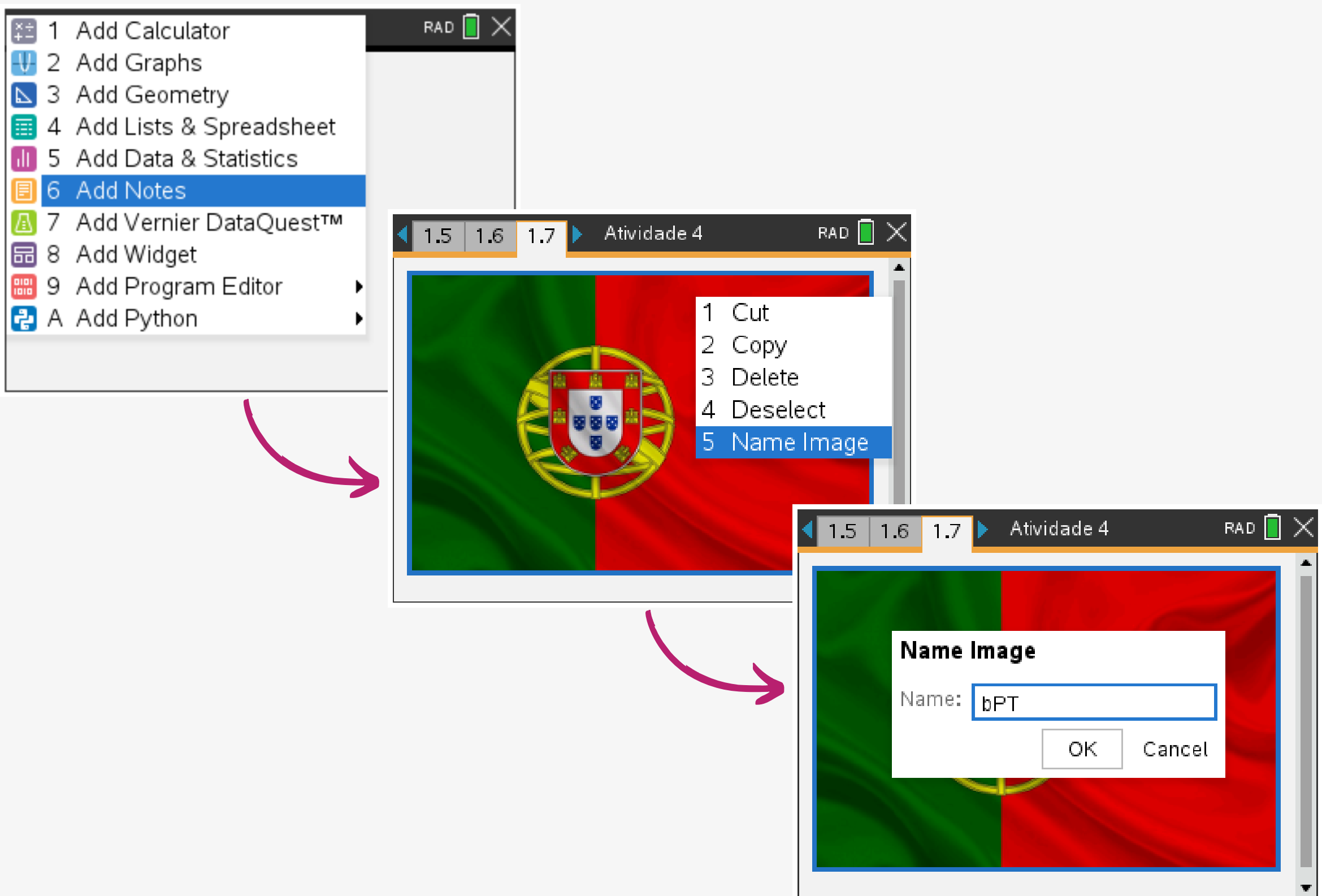
## m&m - micro:bit and music



### 4 What national anthem to play?

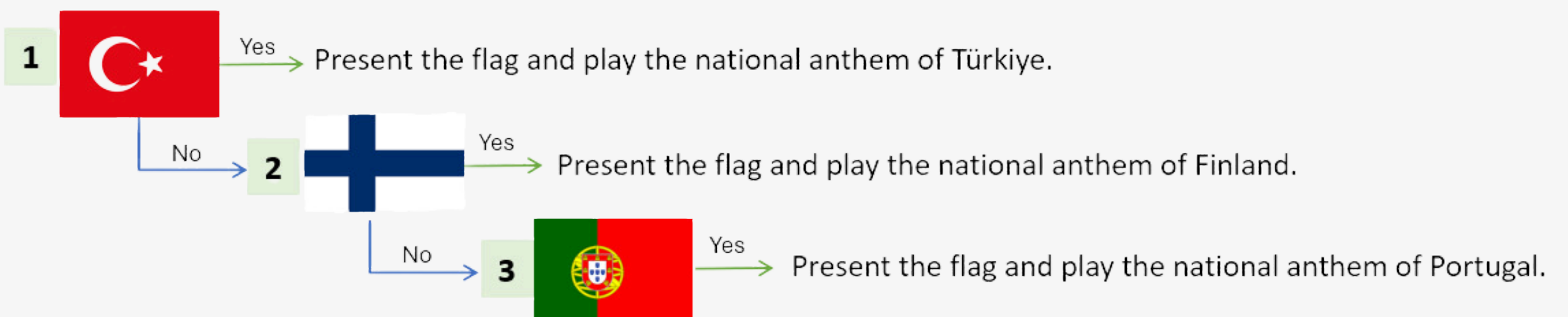
To place images of flags, you must create a new Notes page for each flag in the same document.

Each flag must also be assigned a variable, which must be given a name, by naming the photo. This can be done using **ctrl** + **menu** and then give the image a name (just do **ctrl** + **C** followed by **ctrl** + **V**):



Once we already have the anthems and flags, the idea consists of the following algorithm:

Which country is the winner?



# Activity 4

## m&m - micro:bit and music



### 4 What national anthem to play?

The algorithm basis is as follows:

**If** the winner is Türkiye , i.e. country number 1 **then:**

Make the image of the flag appear Play the anthem at maximum volume.

The anthem will be played with a certain rhythm: following a rhythm of 4 beats per minute where each beat consists of 110 ticks (pulsations).

Finally, the command that allows the anthem to play is placed and the program will continue to execute the next instructions given to it, such as removing the flag from the display.

**But if** the winner is Finland, i.e. country number 2 **then:**

....

That is, in Python language it is:

```
if ctry == 1:
    flag=load.image('bTR')
    flag.show_image(0,0)
    set_volume(255)
    music.set_tempo(4,110)
    music.play(anthem_TR,wait=False)
elif ctry == 2:
    flag=load.image('bFIN')
    flag.show_image(0,0)
    set_volume(255)
    music.set_tempo(4.75)
    music.play(anthem_FIN,wait=False)
elif ctry == 3:
    bandeira=load.image('bPT')
    bandeira.show_image(0,0)
    set_volume(255)
    music.set_tempo(4,100)
    music.play(anthem_PT,wait=False)
```

If “wait = False”, the hymn will start playing and the program will immediately continue executing the next instructions. If “wait = True”, the song will play until the end before the next line of code is executed.



# Activity 4

## m&m - micro:bit and music



### 4 What national anthem to play?

The final code:

```
from ti_image import *
from ti_draw import *
from microbit import *

print("_"*20)
print("1 - Turkey")
print("2 - Finland")
print("3 - Portugal")
print("_"*20)

ctry=int(input("Qual o país vencedor?"))

anthem_PT=["Bb4:1","G:3","Eb:1","C5:8",
"Bb4:7","C5:1","Bb4:4","Ab4:4","G4:3","Eb4:1",
"D4:3","Eb4:1","G4:8","A4:3","Bb4:1","C5:3",
"D5:1","Bb4:10","R:2","Bb4:3","D5:1","F5:8",
"D5:4","Bb4:3","Ab4:1","Ab4:8","G4:3","Bb4:3",
"G4:1","Eb4:8","Bb3:1","Eb4:3","F4:1","G4:3",
"Bb4:1","F4:12","R:4","F4:7","G4:1","Eb4:4",
"Eb4:4","Ab4:7","Bb4:1","G4:3","G4:3","Bb4:1",
"Eb5:8","Bb4:4","Db5:3","C5:1","Bb4:4","F4:9",
"R:2","F4:1","Gb4:4","Bb4:4","Bb4:4","Ab4:3",
"Gb4:1","F4:9","R:2","F4:1","Gb4:4","Bb4:4",
"Bb4:4","Ab4:3","Gb4:1","Bb4:9","R:4","Bb4:1",
"Eb5:8","D5:7","Bb4:1","C5:8","Bb4:3",
"R:1","Eb4:3","F4:1","G4:4","G4:4","Bb4:", "G4:4",
"F4:12","R:3","Bb4:1","Eb5:8","D5:7","Bb4:1",
"C5:8","Bb4:3","R:1","Bb4:3","Eb5:1",
"G5:8","F5:8","Eb5:9","G4:2","Ab4:2","Bb4:2",
"C5:4","C5:4","Bb4:5","R:1","Bb4:1","Eb5:9"]

anthem_TR=["B3:5", "B3:1", "B3", "B3:5",
"B3:1", "B3", "B3:2", "B", "B", "B", "B", "R:2",
"B2:4", "E3", "F#3", "G", "D#:3", "F#:1", "E:11",
"R:1", "E3:4", "A", "B", "C4:3", "B3:1", "G#:3",
"B:1", "A:12", "B:2", "A#:1", "B", "F#:4", "F#:7",
"A:1", "G:3", "D#:1", "E:3", "F#:1", "G:3",
"A:1", "B:3", "C4:1", "D:3", "E:1", "D:4",
"D3:2", "C#:1", "D", "B:4", "A", "G:12",
"B2:2", "A#:1", "B", "F#3:4", "B2", "B3:3",
"A:1", "G:2", "F#:1", "G:1", "E:4", "E4:7",
"D:1", "C:3", "B3:1", "A:3", "G:1", "F#3:3",
"E:1", "B:4", "B2", "E3:12", "B2:4",
"E3", "F#3", "G", "D#:3", "F#3:1", "E:12", "E:4",
"A", "B", "C4:3", "B3:1", "G#:3", "B:1",
"A:12", "B:2", "A#:1", "B", "F#3:4", "F#:8",
"A:1", "G:3", "D#:1", "E:3", "F#3:1", "G:3",
"A:1", "B:3", "C4:1", "D:3", "E:1", "D:4",
"D3:2", "C#:1", "D", "B:4", "A", "G:12",
"B2:2", "A#:1", "B", "F#3:4", "B2", "B3:3",
"A:1", "G:2", "F#3:1", "G:1", "E:4", "E4:7",
"D:1", "C:3", "B3:1", "A:3", "G:1", "F#3:3",
"E:1", "B:4", "B2", "E3:12", "B2:4"]
```

```
anthem_FIN=["F3:2","Bb:3","F:1","D:2","F:2",
"Bb:2","C4:2","D:8","Bb3:4","G:3","C4:1","Bb3:4",
"A:4","Bb:4","R:4","F3:2","D:2","Eb:2","F:6","Bb:2",
"C4:3","F3:1","D4:8","Bb3:4","G:3","C4:1","Bb3:4",
"A:4","Bb:5","R:3","F3:4","C4:3","Bb3:1","A:2","G:2",
"F:2","Eb:2","D:2","G:2","F:4","R:2","F3:4",
"C4:3","Bb3:1","A:2","G:2","F:2","Eb:2","D:2",
"G:2","F:4","R:2","F3:2","Bb:3","F:1","D:2","F:2",
"Bb:2","C4:2","D:8","Bb3:4","G:3","C4:1","Bb3:4",
"A:4","Bb:5","R:3","F3:4","C4:3","Bb3:1","A:2",
"G:2","F:2","Eb:2","D:2","G:2","F:4","R:2","F3:4",
"C4:3","Bb3:1","A:2","G:2","F:2","Eb:2","D:2",
"G:2","F:4","R:2","F3:2","Bb:3","F:1","D:2","F:2",
"Bb:2","C4:2","D:8","Bb3:4","G:3","C4:1","Bb3:4",
"A:4","Bb:5","R:3","F3:4","C4:3","Bb3:1","A:2",
"G:2","F:2","Eb:2","D:2","G:2","F:4","R:2","F3:4",
"C4:3","Bb3:1","A:2","G:2","F:2","Eb:2","D:2",
"G:2","F:4","R:2","F3:2","Bb:3","F:1","D:2","F:2",
"Bb:2","C4:2","D:8","Bb3:4","G:3","C4:1","Bb3:4",
"A:4","Bb:5","R:3","F3:4","C4:3","Bb3:1","A:2",
"C4:3","F3:1","D4:8","Bb3:4","G:3","C4:1","Bb3:4","A:4",
"Bb:5","R:3","F3:4","C4:3","Bb3:1","A:2","G:2","F3:2",
"Eb3:2","D:2","G:2","F:4","R:2","F3:4","C4:3","Bb3:1",
"A:2","G:2","F:2","Eb3:2","D:2","G:2","F:4","R:2",
"F3:2","Bb:3","F:1","D:2","F:2","Bb:2","C4:2","D:8",
"Bb3:5","G:5","C4:2","Bb3:5","A:5","Bb:8"]

if ctry==1:
    bandeira=load_image("bTR")
    bandeira.show_image(0,0)
    set_volume(255)
    music.set_tempo(4,110)
    music.play(anthem_TR,wait=False)
elif ctry==3:
    bandeira=load_image("bFIN")
    bandeira.show_image(0,0)
    set_volume(255)
    music.set_tempo(4,75)
    music.play(anthem_FIN,wait=False)
elif ctry==5:
    bandeira=load_image("bPT")
    bandeira.show_image(0,0)
    set_volume(255)
    music.set_tempo(4,100)
    music.play(anthem_PT,wait=False)
```

### Challenge:

Three more countries entered the championship: Greece, Ireland and Slovakia. The challenge is to obtain the anthems of these three countries and complete the code for the six countries.



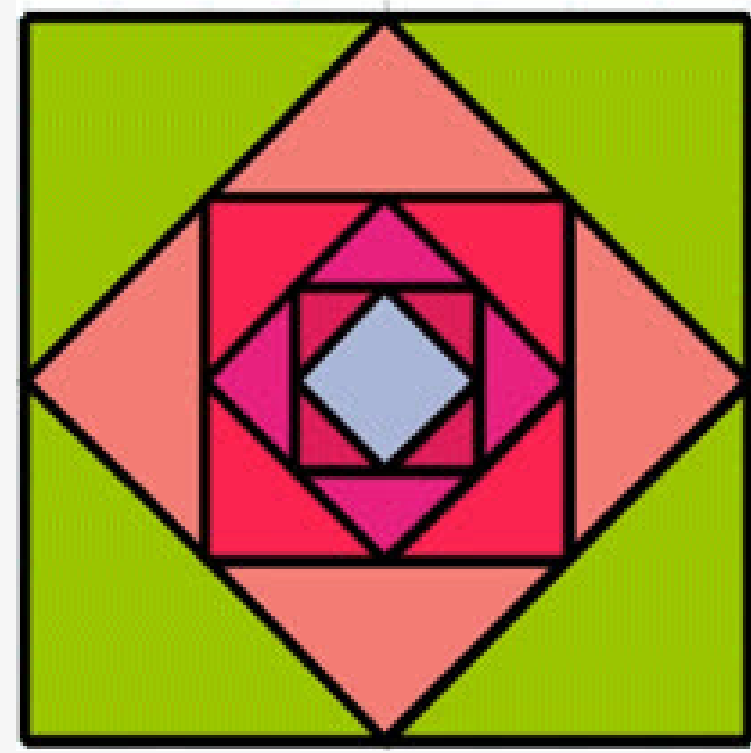
By accessing the **A4.tns** file you can see all the programs analyzed here, either by reading the QR Code on the side or by clicking on it:





# Activity 5

## What a beautiful work of art!



Python offers a module, the Turtle module, which allows you to create, with simple commands, patterns with different complexities, geometric shapes and other appealing visual effects, in an educational environment.

### How is it installed?

By clicking or reading the QR Code on the side, you can access the file necessary to download to the calculator and a Getting Started guide for installing the module on it.



With the module installed, continue being artists and start with a very simple work of art: a square.

### 1 Draw a square.

First of all, you must start by importing the Turtle library into the first line of the program.

To do this, go to: **menu** > 9: More modules > 7: Turtle Graphics > 1: from turtle import\*

The **from turtle import\*** statement appears, but also the **t=Turtle()** statement in the line of code, which creates the Turtle object with the name **t**.

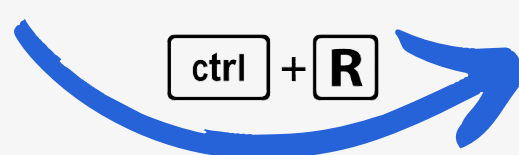
```
1.1 1.2 *Doc RAD X
*quad.py 3/3
from turtle import *
t=Turtle()
```

The turtle draws with the trail of its movement. When you “ask” it to make a square, you have to think about how the turtle should “walk”:

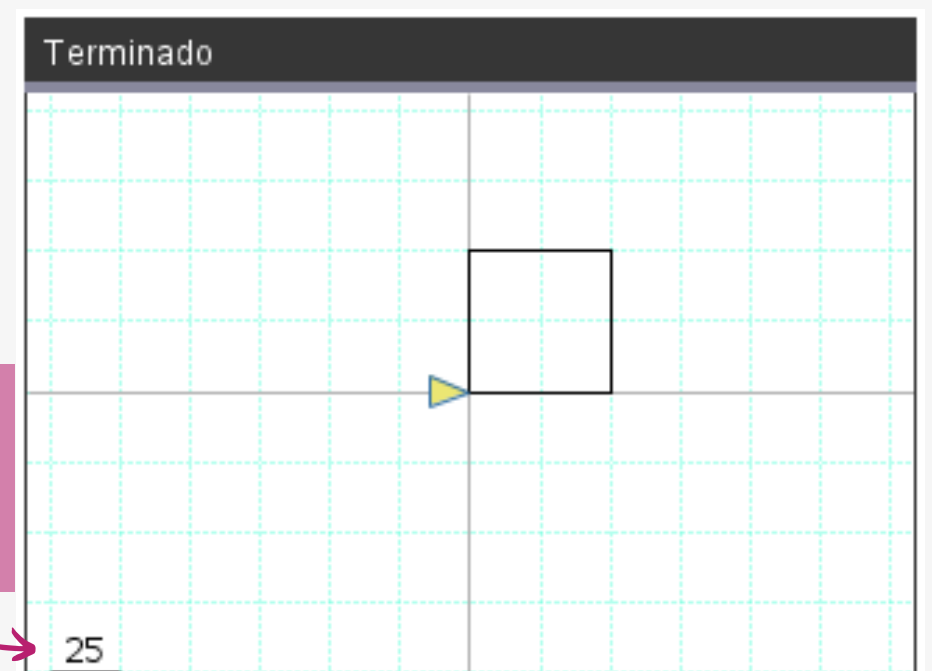
At first, the turtle moves forward, for example, 50 pixels. [**t.forward(50)**] Then, to turn the first corner, with 90°, you have to turn 90°, to the left or to the right. [**t.left(90)** or **t.right(90)**, respectively]. This procedure has to be done 3 more times (to complete the square), after which you return to the point where you started. [**for i in range(4):**]

```
1.1 1.2 1.3 *Atividade 5 RAD X
quad.py 1/7
from turtle import *
t=Turtle()

for i in range(4):
    t.forward(50)
    t.left(90)
```

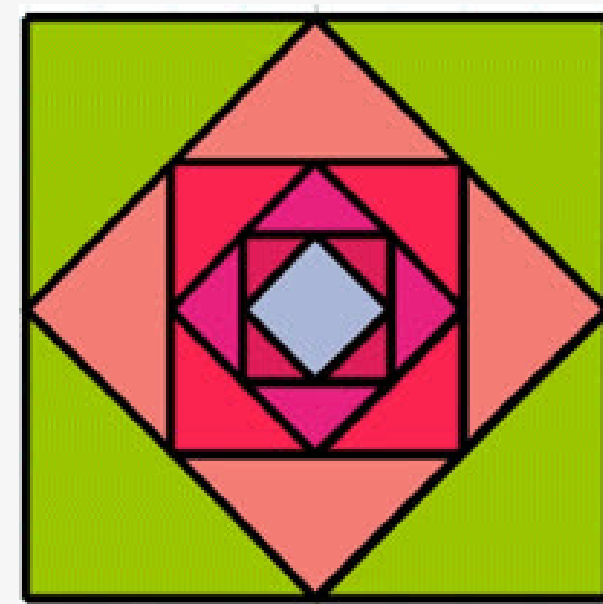


The grid lines are separated by 25 pixels, as can be seen in the legend in the lower left corner of the screen.



# Activity 5

## What a beautiful work of art!



### 2 Draw a regular polygon with $n$ sides.

Based on the previous program, the user is expected to enter the number of sides of the regular polygon for the turtle to draw.

To do this, it is necessary, after **from turtle** **import \*** and before the `t=Turtle()` instruction, to place the instruction `n=int(input("How many sides?"))` so that the question is asked before the drawing screen appears.

Sometimes there are more complex designs that can take longer. But, here you can change the speed of the turtle, making it faster or even instantaneous. To do this, enter the command **t.speed(vel)**, an instruction that can be found in the Settings submenu ( `> 9: More modules > 7: Turtle Graphics > 5: Settings` ).

The **vel** argument in parentheses is a value between 0 and 10, with 0 corresponding to the fastest movement.

The **range(n)** command indicates that the indented action will be repeated  $n$  times. When explained, it varies from 0 to  $n-1$ , with increments of 1 unit.

```
*Activity 5
RAD
11/17
*poligono_n.py
from turtle import *
n=int(input("How many sides?"))
t=Turtle()
```

```
*Activity 5
RAD
5/10
*poligono_n.py
from turtle import *
n=int(input("How many sides?"))
t=Turtle()
t.speed(9)
for i in range(n):
    t.forward(50)
    t.left(360/n)
```

Pay attention to the rotation angle that varies according to the number of sides of the regular polygon!!!

### But what if we want to paint the polygon? It's possible?

**Yes! It's possible.** There are two ways to assign color to a figure:

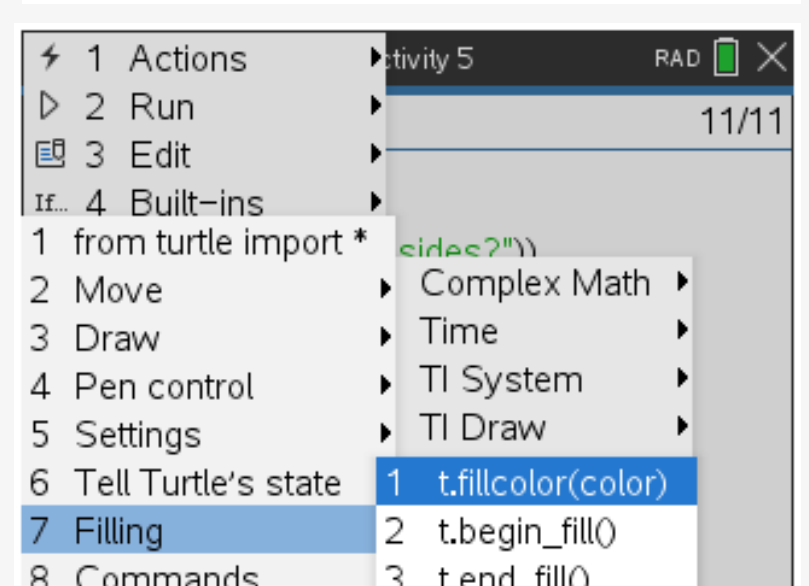
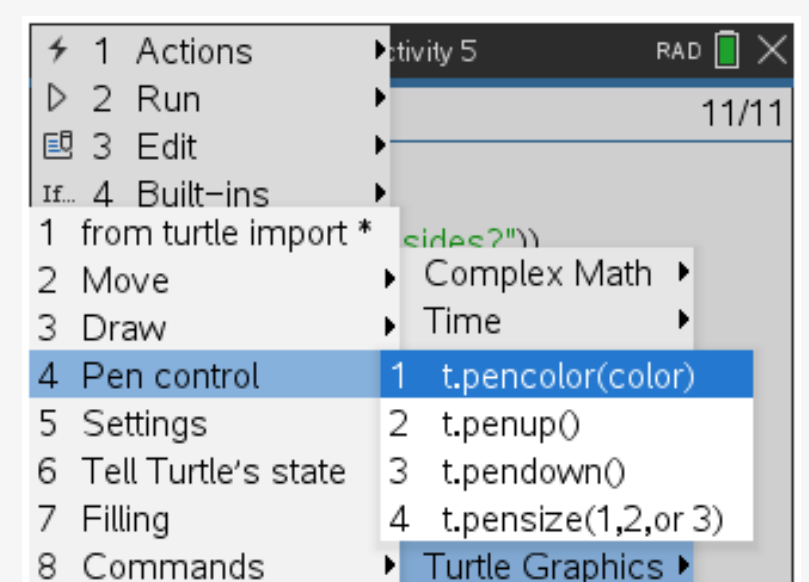
- **t.pencolor(r,g,b)** - accessing the submenu **4: Pen control**

This command allows you to color lines, in this case the sides of the polygon.

If the instruction is in the repeat cycle and the RGB code is automatically changed, it can have one side of each color.

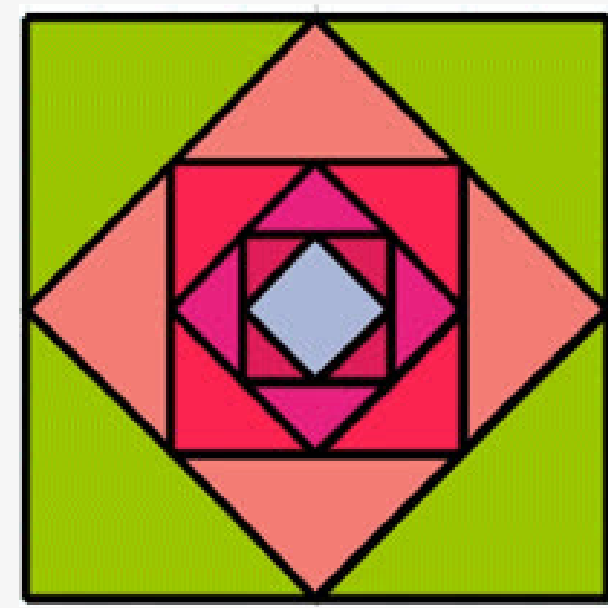
- **t.fillcolor(r,g,b)** - accessing the submenu **7: Filling**

This command allows you to fill the figure with the color of the RGB code considered or the expression relating to the color in a specific menu.



# Activity 5

## What a beautiful work of art!



### 2 Draw a regular polygon with $n$ sides.

Code R-G-B

In this system, each color is defined by the amount of red, green and blue that make it up. Thus, a palette of colors can be obtained, each one as the result of the combination of the three aforementioned colors. To know the quantities of each color, use integers between 0 and 255 to specify them. The number 0 indicates no intensity and the number 255 indicates maximum intensity.

To obtain a pentagon with a colored interior, before drawing it you must choose the color, with the instruction **t.fillcolor(x,y,z)** from submenu **7: Filling**.

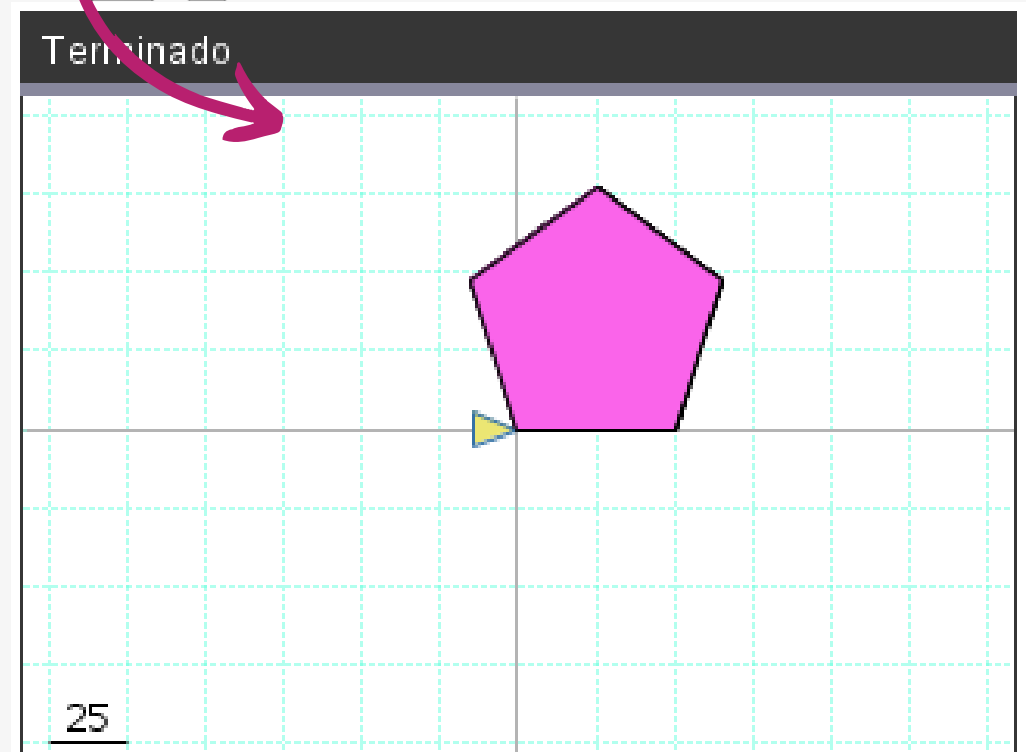
Integers from 0 to 255 are placed in **x**, **y** and **z** to obtain a color.

Then, the order must be given to start filling with the **t.begin\_fill()** instruction, from the same submenu. This instruction must be placed before drawing a pattern.

In the end, to observe the polygon with a filled interior, you must insert the **t.end\_fill()** instruction.

```
*Activity 5
4/10
*color_polig.py
from turtle import *
n=int(input("How many sides?"))
t=Turtle()
t.speed(9)
t.fillcolor(250,100,234)
t.begin_fill()
for i in range(n):
    t.forward(50)
    t.left(360/n)
t.end_fill()
```

ctrl + R n=5

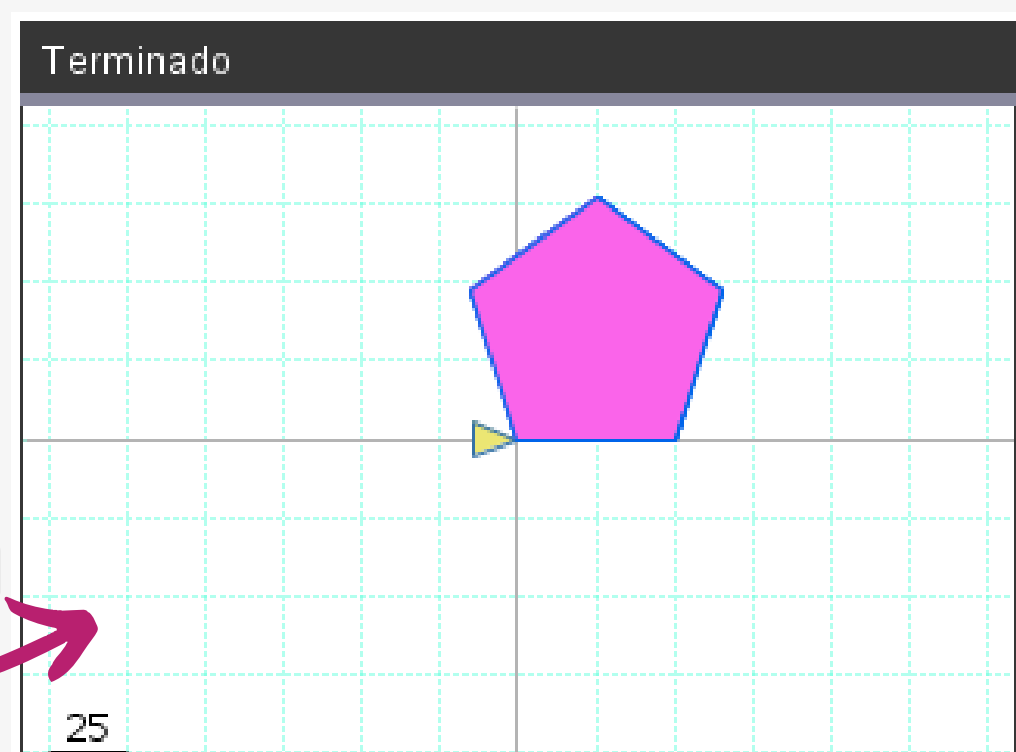


If you want to color the polygon line in another color, add **t.pencolor(a,b,c)** (from submenu **5: Pen control**) before the **t.fillcolor(x,y,z)** command.

```
*Activity 5
11/11
color_polig.py
from turtle import *
n=int(input("How many sides?"))
t=Turtle()
t.speed(9)
t.pencolor(0,100,234)
t.fillcolor(250,100,234)
t.begin_fill()
for i in range(n):
    t.forward(50)
    t.left(360/n)
t.end_fill()
```

ctrl + R

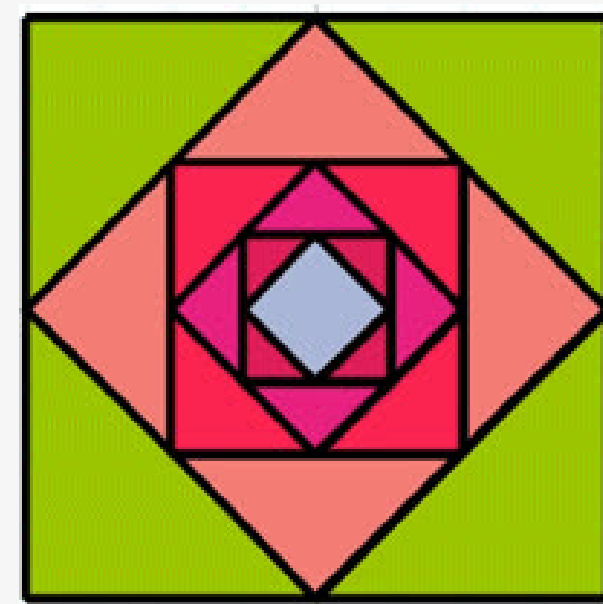
n=5





# Activity 5

## What a beautiful work of art!



### 2 Draw a regular polygon with n sides.

On the calculator, when you access the `t.fillcolor` or `t.pencolor` command, a list of colors appears by default from which you can choose one.

```
*color red 6/12
ffrom turtle green
n=int(input("How many sides?"))
t=Turtle()
t.speed(9)
t.pencolor("blue")
t.fillcolor("yellow")
#.fillcolor("cyan")
t.begin_fill()
for i in range(n):
    t.forward(50)
    t.left(360/n)
```

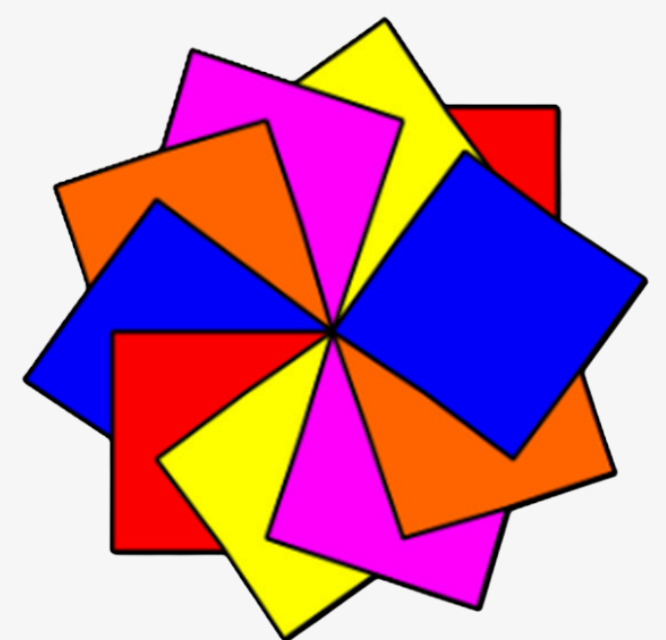
```
*color_polig.py 6/12
ffrom turtle import *
n=int(input("How many sides?"))
t=Turtle()
t.speed(9)
t.pencolor(0,100,234)
t.fillcolor("yellow")
#.fillcolor(250,100,234)
t.begin_fill()
for i in range(n):
    t.forward(50)
    t.left(360/n)
```

There are two ways to color a figure here.

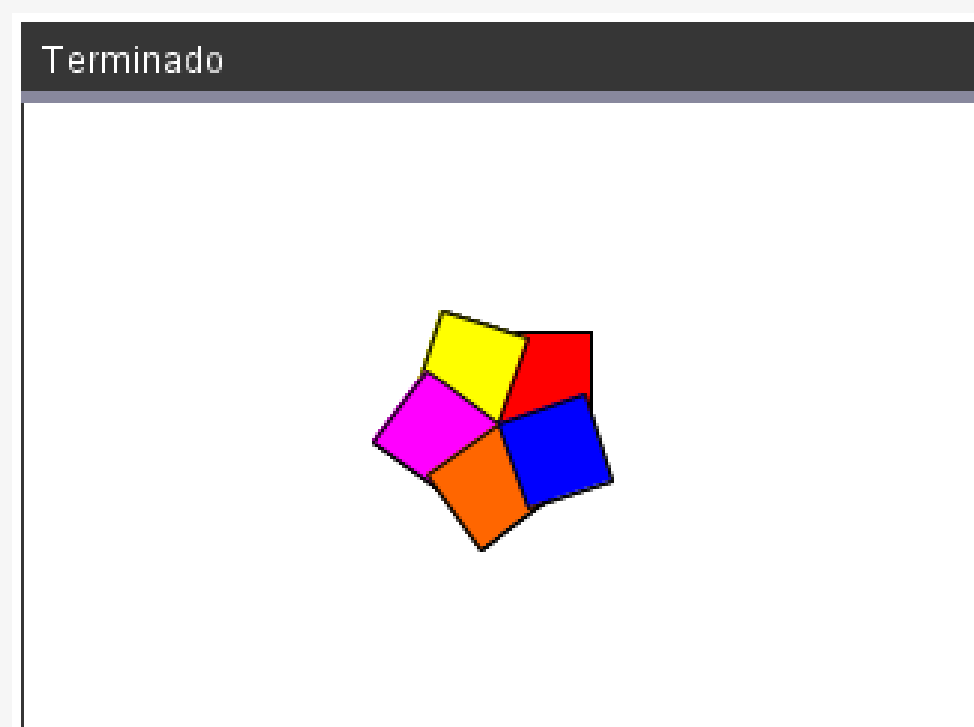
### 3 Several squares...

With the bases obtained, you can now create a more artistic figure, like the one to the side.

Next, there is a modification in the execution of the program to build an analogous figure, where the user will have the freedom to choose whether they want larger or smaller squares and their quantity.



```
*Atividade 5 6/6
Python Shell
>>>#Running color_polig_round.py
>>>from color_polig_round import *
>>>#Running color_polig_round.py
>>>from color_polig_round import *
How many squares?5
How big is each square?30
```



As the objective is for the user to have freedom in choosing the number and size of the squares, it is necessary to insert the instructions:

```
q=int(input("How many squares?"))
l=int(input("How long is each square?"))
```

Furthermore, we want to build the figure without having the turtle and the grid visible, so we ask to hide the two elements using `t.hideturtle()` and `t.hidegrid()`. These two commands can be found in submenu 5: Settings.

```
*Atividade 5 10/22
*color_polig_round.py
from turtle import *
q=int(input("How many squares?"))
l=int(input("How big is each square?"))

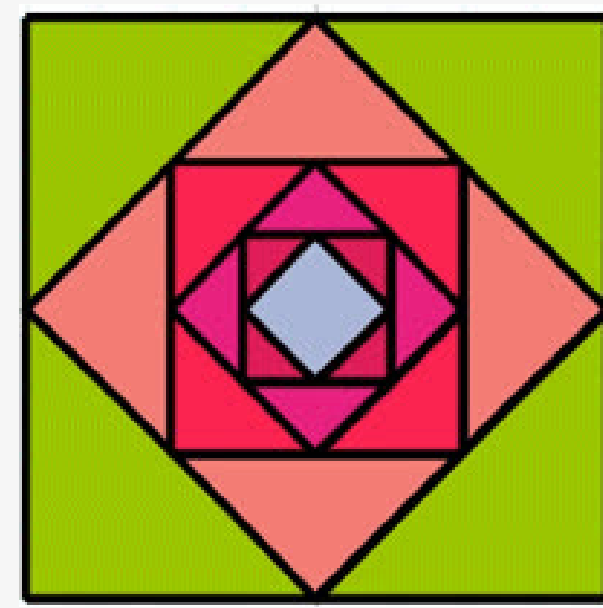
t=Turtle();t.hideturtle();t.hidegrid();t.speed(10)

colors=["red","yellow","magenta","orange","blue"]
```

The design has squares with different colors that alternate to insert a list of colors.

# Activity 5

## What a beautiful work of art!



### 3 Several squares...

For the program to have a more simplified code, you can start by using a Python function to construct a square. Consider the name **quad(x)** for this function, where x is the length, in pixels, of the sides of the square.

```
1.5 1.6 1.7 *Activity 5 RAD 1/18
sq_col_round.py
from turtle import *
q=int(input("How many squares?"))
l=int(input("How big is each square?"))

t=Turtle();t.hideturtle();t.hidegrid();t.speed(10)
colors=["red","yellow","magenta","orange","blue"]

def quad(a):
    for i in range(4):
        t.forward(a)
        t.left(90)
```

### How does square coloring work?

As already written at the beginning, you have to have a predefined color palette to then color the squares. This palette is named "colors".

The coloring follows a pattern in which the first square is colored with the first color in the list, the second square is colored with the second color in the list and so on...

**But** the palette only has 5 colors and if the user wants a design with 10 squares, how will it be done?

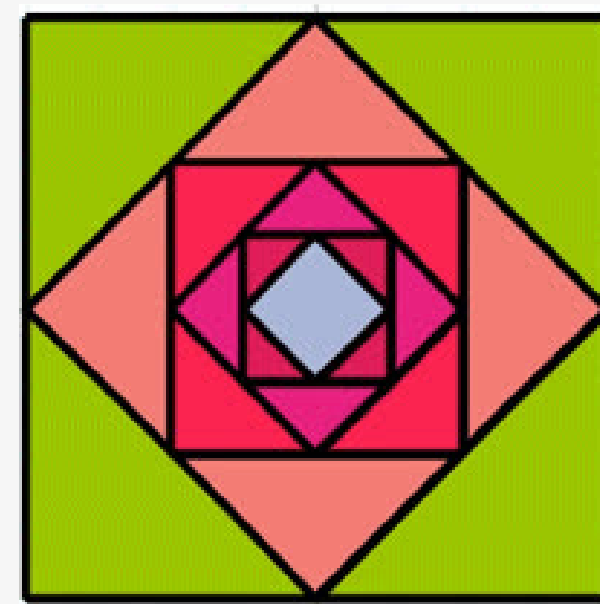
The idea is to use the remainder of the integer division of i by 5, which in Python translates as **i%5**. This operation returns the remainder of the division of the left operand by the right operand, that is, for example, 6%5=1 because when dividing 6 by 5, the remainder is 1.

Before continuing, it is important to remember that the first element of a list is indexed as 0. Therefore, if **i** is the square number **i+1**, we have the following:

|                                 |      |   |
|---------------------------------|------|---|
| <b>1st Square</b>               | i=0  | 0%5=0<br>1st color in the palette: colors[0]  |
| <b>2nd Square</b>               | i=1  | 1%5=1<br>2nd color of the palette: colors[1]  |
| <b>3rd Square</b>               | i=3  | 3%5=3<br>4th color of the palette: colors[3]  |
| <b>4th Square</b>               | i=6  | 6%5=1<br>2nd color of the palette: colors[1]  |
| <b>12nd Square</b>              | i=12 | 12%5=2<br>3rd color of the palette: colors[2] |
| <b>(i+1)<sup>o</sup> Square</b> | i+1  | <b>colors[i%5]</b>                            |

## Activity 5

# What a beautiful work of art!

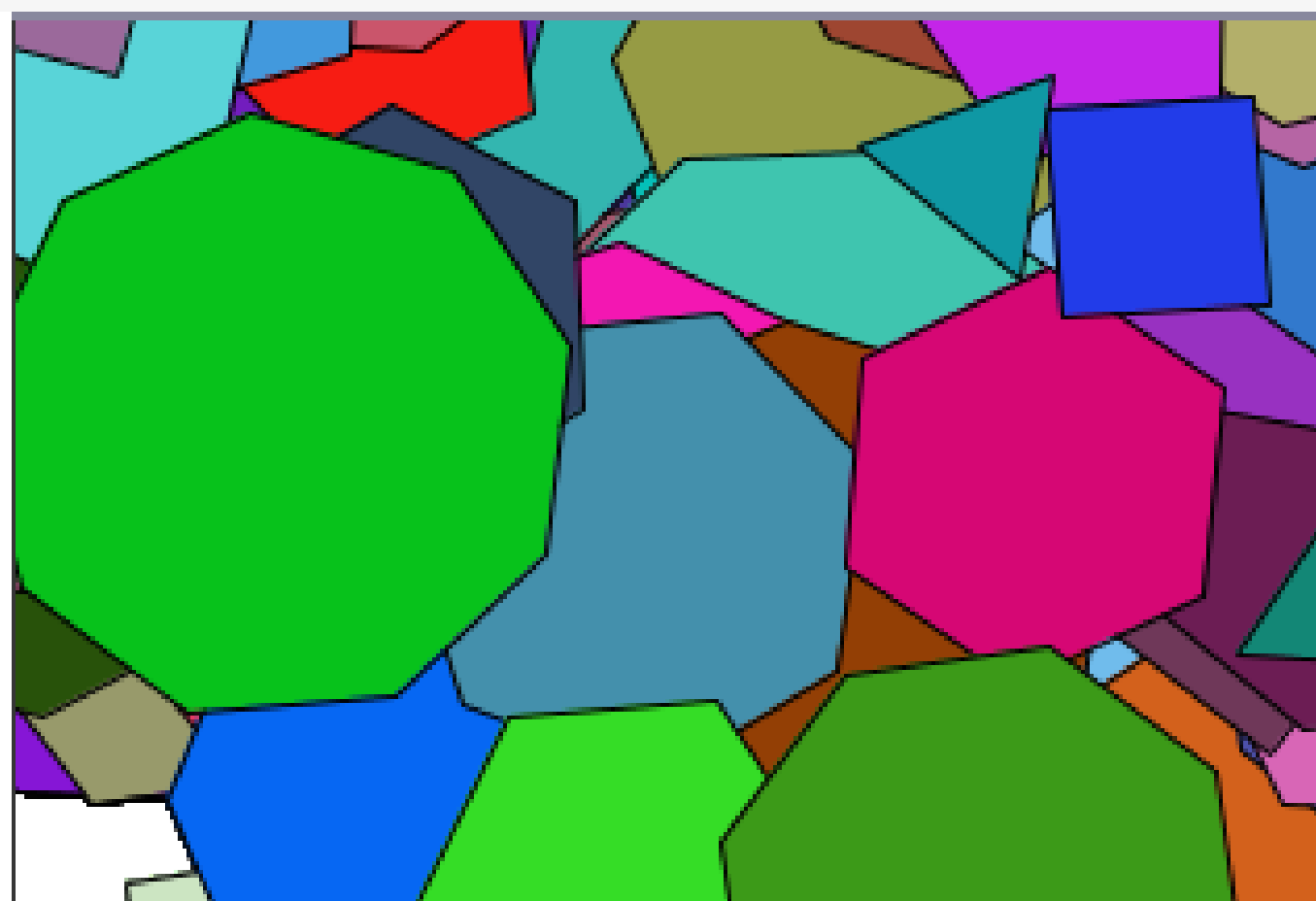


The design in question is obtained through  $q$  rotations of  $360/q$  degrees of a square, which is done using  $q$  times the repetition cycle: `for i in range(q)`

```
*Activity 5
sq_col_round.py 18/18
def quad(a):
    for i in range(4):
        t.forward(a)
        t.left(90)
for i in range(q): #36xq=360
    t.fillcolor(colors[i%5])
    t.begin_fill()
    quad(50)
    t.end_fill()
    t.left(360/q)
```

### 4 A “cubist” work...

What we now want is a program that produces an image identical to the one illustrated alongside, which is made up of regular polygons in a random manner, whether in relation to the number of sides, the color or the place where the drawing starts.



As you will need to generate random numbers, in addition to importing the Turtle module, you must also import the Random module.

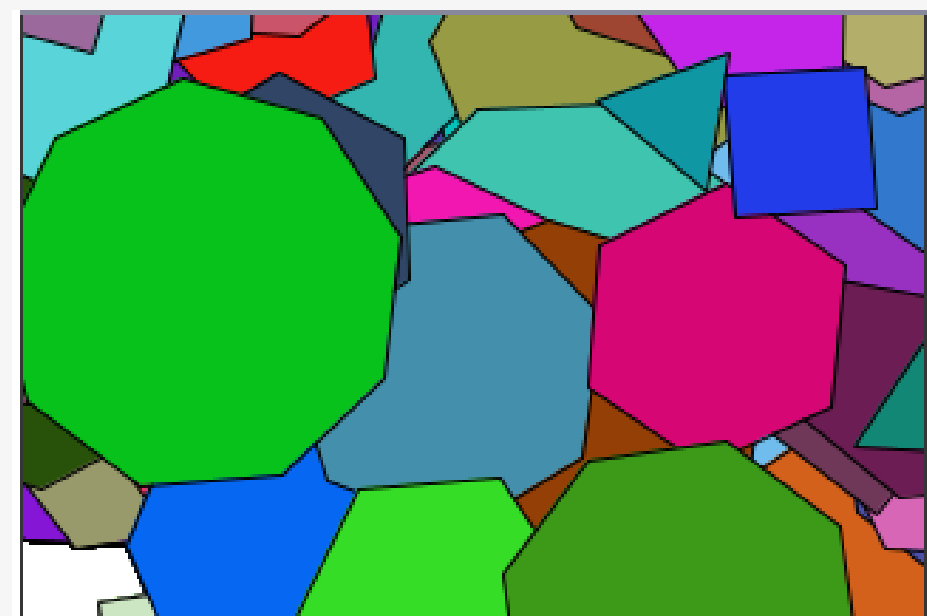
To improve the appearance of the drawing, use the `hideturtle()` and `t.hidewindow()` commands to hide the turtle and the grid, respectively. In addition, `t.speed(0)` is used to make the construction of the work of art faster!

```
*Atividade 5
*Poligonos.py 1/23
from turtle import *
from random import *
t=Turtle()
t.hidewindow(); t.hideturtle(); t.speed(0)
```



# Activity 5

## What a beautiful work of art!



### 4 A "cubist" work...

First, use the command

```
while get_key()!="esc"
```

in the main repeat cycle to stop program execution by pressing the key `esc`.

Then, you have to define the number of sides of each polygon, which, being a random number, is obtained with the command `n=randint(3,9)` (generates random integers from 3 to 9).

Next, place:

- **t.penup()** - to lift the turtle off the screen and when it moves it will not draw. It is important when you want not to show the trace of the movement of an object made by the following command. This command can be found in submenu **4: Pen control**.

- **t.goto(x,y)** - to move the turtle to a point on the screen with coordinates (x,y). If the turtle has not been "lifted", a linear trail will be drawn to the destination (x,y). In this work of art, these coordinates are intended to be random, which is why the command

```
t.goto(randint(-105,105),randint(-150,150))
```

**Note** that the display has dimensions of 318 x 212 pixels.

This command can be found in submenu **2: Move**.

- **t.setheading(α)** - to set the turtle's orientation for each angle **α**. Paste **t.setheading(-90,90)** onto the work of art. This command can be found in submenu **2: Move**.

- **t.pendown()** - to make the turtle reappear. This command can be found in submenu **4: Pen control**.

Finally, the idea is similar to that already seen in the programs presented previously. Note that, as we want a work of art where the polygons are colored with random colors, then the command

```
t.fillcolor(date(0.255),date(0.255),date(0.255))
```

```
1.5 1.6 1.7 *Atividade 5 RAD [battery icon] [close icon]
*Poligonos.py 12/18
from random import *
t=Turtle()
t.hidegrid(); t.hideturtle(); t.speed(0)
while get_key()!="esc":
    ♦♦n=randint(3,9)
    ♦♦t.penup()
    ♦♦t.goto(randint(-150,150),randint(-105,105))
    ♦♦t.pendown()
    ♦♦t.setheading(randint(-90,90))
```

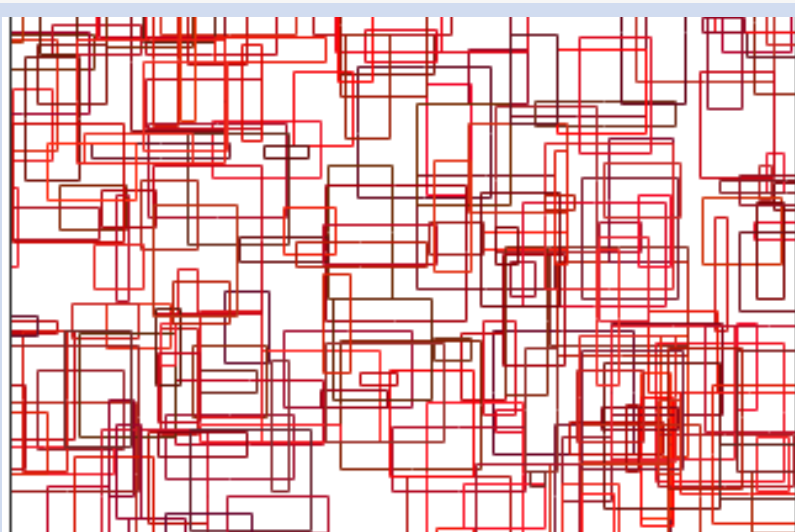
```
1.5 1.6 1.7 *Atividade 5 RAD [battery icon] [close icon]
*Poligonos.py 17/17
♦♦t.penup()
♦♦t.goto(randint(-150,150),randint(-105,105))
♦♦t.pendown()
♦♦t.setheading(randint(-90,90))

♦♦t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
♦♦t.begin_fill()
♦♦for i in range(n):
    ♦♦♦♦t.forward(50)
    ♦♦♦♦t.left(360/n)
♦♦t.end_fill()
```

### Challenge:

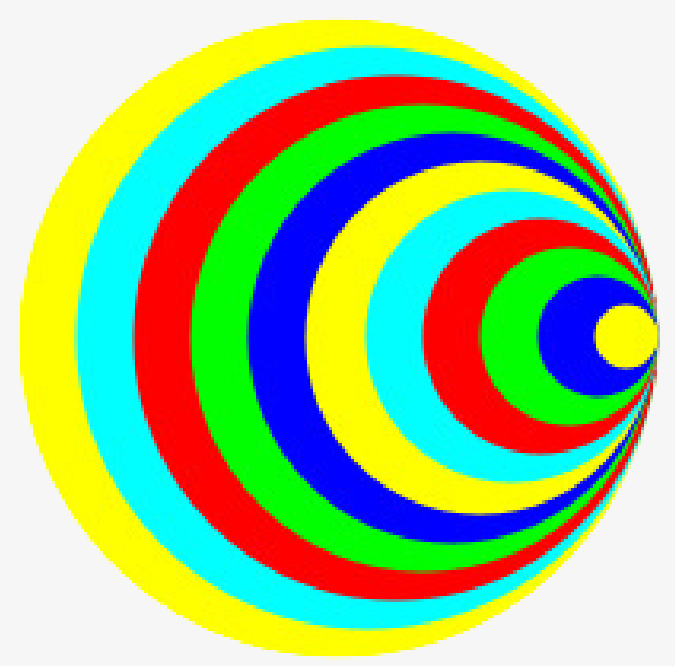
With what has been done so far it is possible to build many other abstract works of art!

The challenge is to create a code that allows you to obtain a figure identical to the one on the side, made up of several random rectangles. Please note that they are not filled in!



# Activity 5

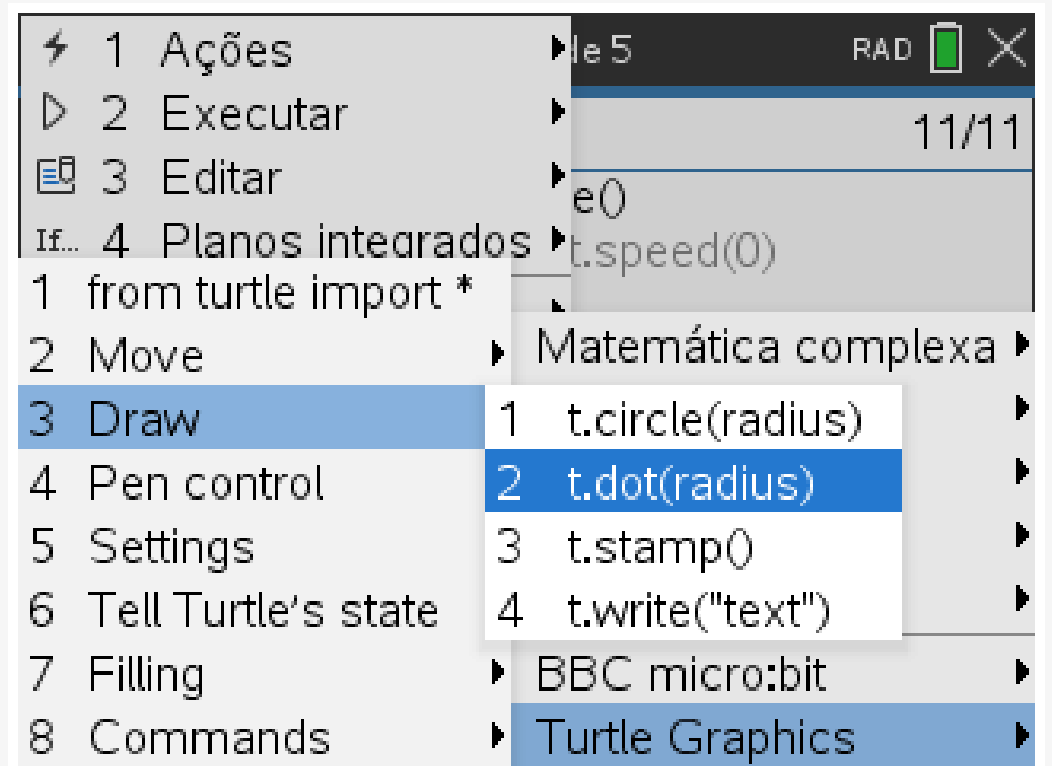
## What a beautiful work of art!



### 5 Special rainbow.

Now a new topic will be introduced in this Turtle world: **dot**.

To program the figure illustrated in the upper right corner, you will use this tool, which is in submenu **3: Draw**.

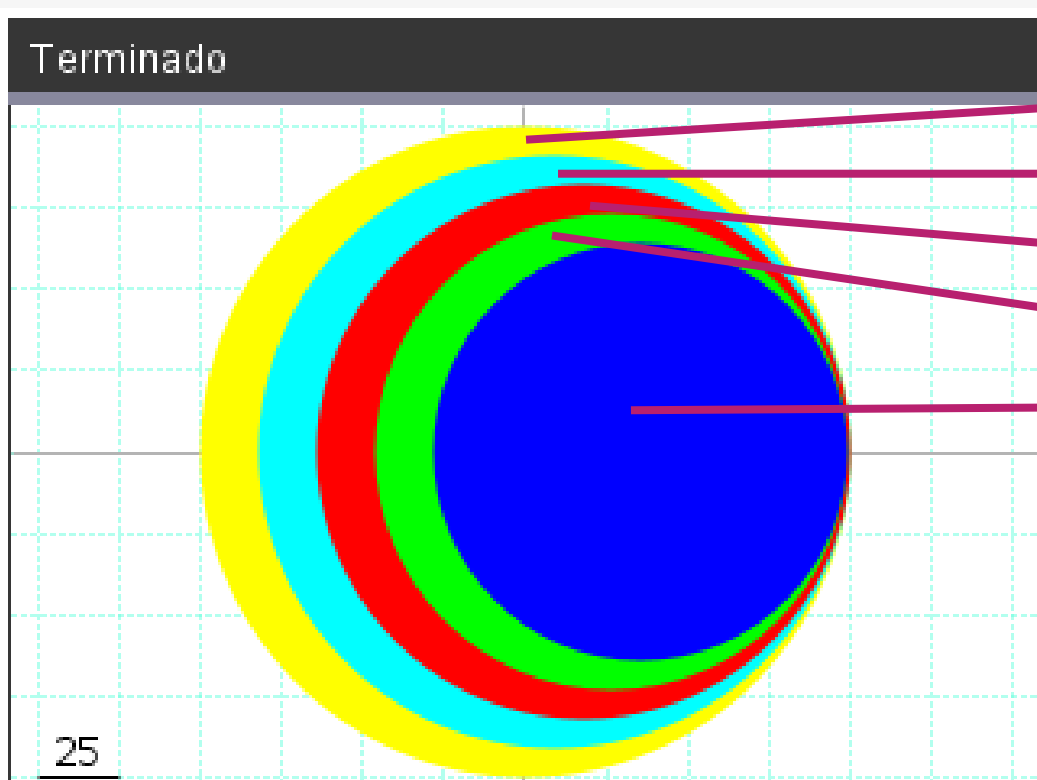
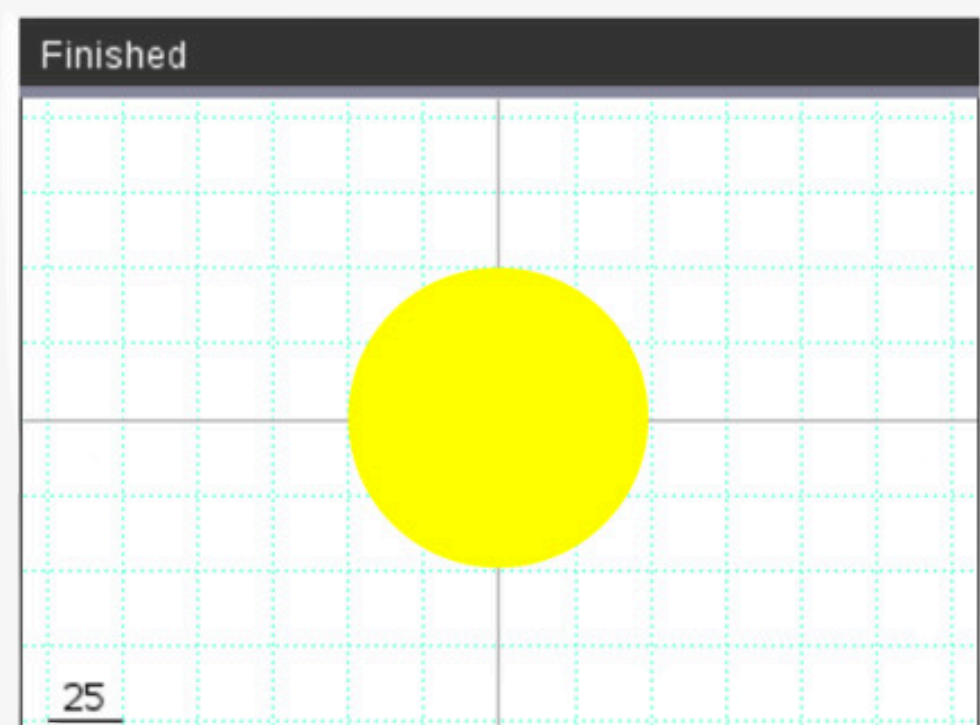
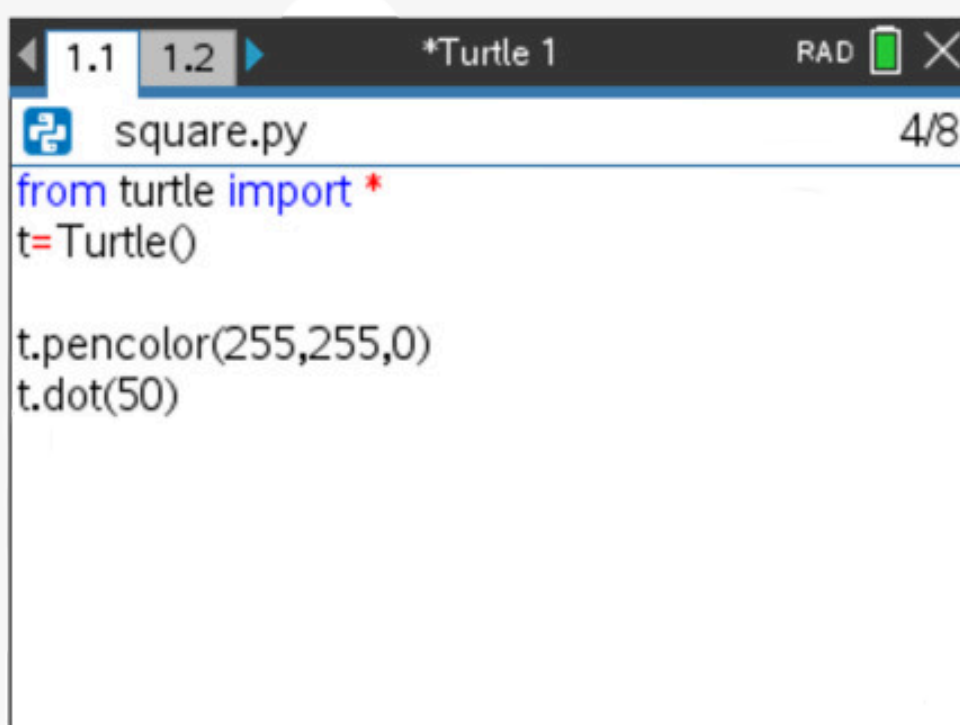


As you can see in the figure, the **t.circle(radius)** instruction also appears, can't you use it?

Now, the **t.circle()** instruction only draws a circle and not its interior.

Therefore, for the drawing that will be made, the command **t.dot(radius)** must be used.

Below is an example of a "dot" painted yellow whose radius is 50 pixels.



- t.dot(100)
- t.dot(91)
- t.dot(82)
- t.dot(73)
- t.dot(67)
- ⋮

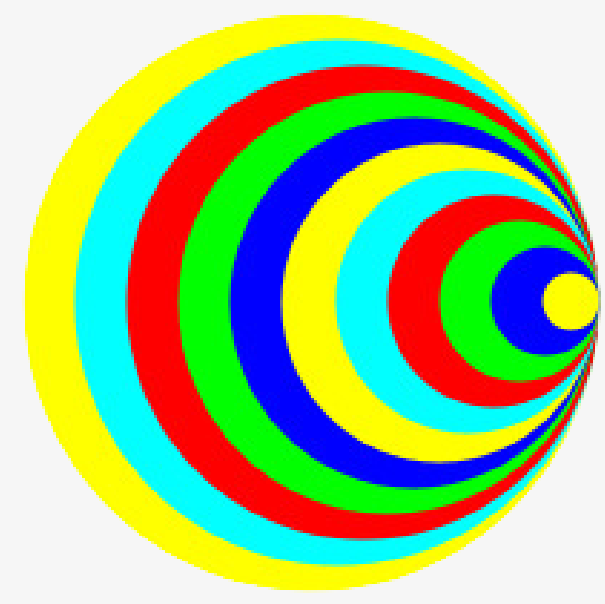
To achieve a descending effect in the drawing, at each point the radius is placed smaller and smaller and the center coordinates are translated.

In other words, in the dot that will be drawn next, the radius decreases by 9 units and its center moves 9 units to the right.



# Activity 5

## What a beautiful work of art!



### 5 Special rainbow.

So, the program to obtain this work of art is what is next.

The coloring process is similar to what has been done previously.

Inside the **for** loop you will find the radius **r**, which will be modified in each iteration, and the instruction that paints the points.

In addition, here you will also find instructions for moving the center of each “point” and, respectively, drawing them.

Please note that the **range** command argument must respect the fact that **r > 0**, that is, the radius of the dot is positive. If you enter a number greater than 11, the program will give an error:

```
1.9 1.10 1.11 Activity 5 RAD 1/12
arcoirisred.py
from turtle import *; t=Turtle()
#.hidegrid(); t.hideturtle(); t.speed(0)

cores=["yellow","cyan","red","green","blue"]

for i in range(12):
    r=100-9*i
    t.pencolor(cores[i%5])
    t.penup()
    t.goto(9*i,0)
    t.dot(r)
```

```
1.9 1.10 1.11 *Activity 5 RAD 3/3
Shell Python
>>>#Running arcoirisred.py
>>>from arcoirisred import *
>>>
Error
Radius cannot be negative
OK
```

### 6 Borders and signs...

A border is a line that territorially delimits a State, establishing its extension. To know which country you are entering in Europe, there is usually a sign, like the one illustrated on the side, which indicates which European country you are entering.

The aim here is to replicate the Portuguese border sign, like the one next door.



```
from turtle import *

t=Turtle()
t.hidegrid(); t.hideturtle(); t.speed(0)
```

```
Blue = (8, 92, 208)
```

```
def quad(a):
    t.penup()
    t.goto(-a/2,-a/2)
    t.pendown()
    t.begin_fill()
    for i in range(4):
        t.forward(a)
        t.left(90)
    t.end_fill()
```

As always, you must import the Turtle module. Furthermore, you must start by hiding the grid and the turtle and start drawing it in quick mode.

The blue color of the plate is defined according to the RGB system

Here, as previously seen, a function is defined that makes the turtle draw a square with the side measuring **a**.

The command **t.goto (-a/2,-a/2)** positions the center of the square at the origin (0, 0). Divide **a** by **2** to ensure the square is centered.



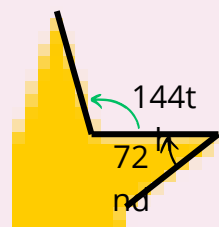
# Activity 5

## What a beautiful work of art!



### 6 Borders and signs...

```
def estr(a):  
    d=1.902*a  
    t.penup()  
    t.setheading(54)  
    t.forward(a)  
    t.left(54)  
    t.fillcolor("yellow")  
    t.pendown()  
    t.begin_fill()  
    for i in range(5):  
        t.forward(d)  
        t.left(144)  
        t.forward(d)  
        t.right(72)  
    t.end_fill()
```



This function makes the turtle draw a star.

Start by multiplying **a** by **1.902** to obtain **d**, which is the length of each segment of the line that will form the star.

This value is a calculated factor to ensure that the star's proportions are correct.

Then, to adjust the turtle's direction to 54 degrees, use the **t.setheading(54)** command to guide the turtle to the correct angle from which it will start drawing the star.

After this, you have to place the turtle in the initial position to start drawing the star, therefore, placing the following command lines:

**t.forward(a)** - the turtle moves forward "a" units

**t.left(54)** - the turtle turns 54° to the left.

The program continues in a similar way to what has been seen previously.

```
def stars():  
    for i in range(12):  
        t.penup()  
        t.home()  
        t.setheading(30*i)  
        t.forward(40)  
        estr(2.2)
```

As you don't just want to build one star, but 12 in circumference. In this way, a **for** cycle is made:

- The turtle moves without drawing, always returning to the initial position (center of the screen) so that each star will be equally distanced from the center. This is done with the **t.penup()** and **t.home()** instructions, respectively.

- Next, you set the turtle's direction to a specific angle, with the instruction **t.setheading(30\*i)**. Multiply **i** by **30°**, because **360°** (complete turn) **divided by 12** (number of stars) **is equal to 30°**. This places each star at an equal angular distance from the others.
- In order to move the turtle to the position where the star should be drawn in relation to the center, that is, at a distance of 40 pixels, the instruction **t.forward(40)** is placed.
- Finally, the previously created function "**estr**" where the argument is **2.2**, is called to draw and paint a star with that size in the turtle's current position.

# Activity 5

## What a beautiful work of art!



### 6 Borders and signs...

#### To end:

```
t.pencolor(Blue)
t.fillcolor(Blue)
quad(120)
t.fillcolor("white")
quad(117)
t.fillcolor(Blue)
quad(112)
stars()
t.pencolor("white")
t.penup()
t.goto(-28,-12)
t.pendown()
t.write("Portugal")
```

- The blue plate is designed with the blue border
- A white square smaller than the first drawn is drawn and superimposed on it.
- Another blue square smaller than the previous one is drawn, which is superimposed on this one.
- When calling the stars() function, the turtle is asked to draw the 12 stars.
- To write the word "Portugal", you must be aware that the turtle must be hidden, that the word is written in white and must be centered on the plate.

### 7 More challenges with the Turtle module!

(1) Draw the flag of Portugal, starting without the central area of the coat of arms and arms, or of another country of your choice.

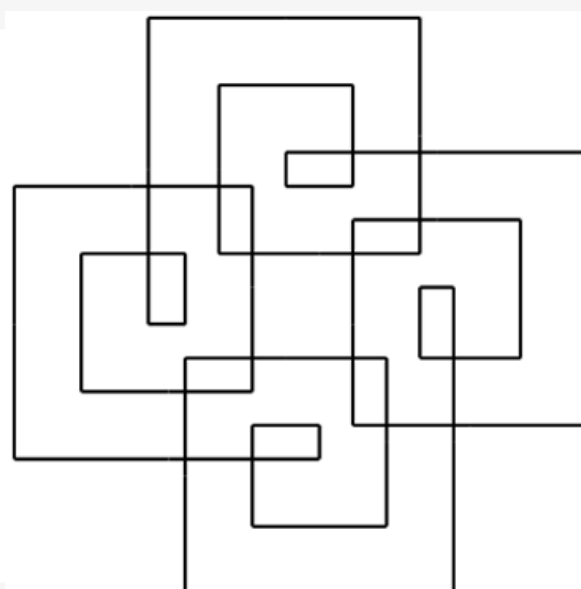


(2) Design the Mitsubishi logo or another car logo of your choice!

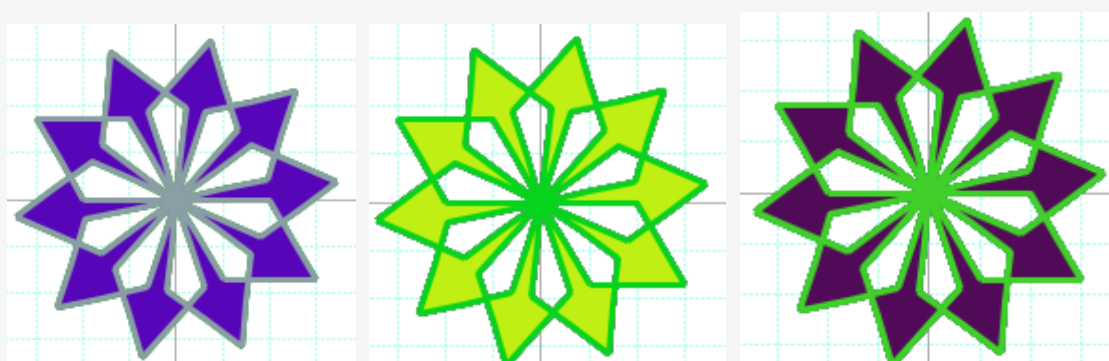


(3) Try making art with spirolaterals!

```
from turtle import *
t=Turtle()
t.speed(10);t.hideturtle();t.hidegrid()
passos=[1,2,3,4,5,6,7,8,9]
n=0
while n<4:
    for i in passos:
        t.forward(i*10)
        t.left(90)
    n=n+1
```



(4) Write a code that results in a design like the one shown in the image below. The colors of the drawing alternate.



By accessing the **A5.tns** file you can see all the programs analyzed here and a proposed resolution of the Challenges, either by reading the QR Code on the side or by clicking on it:





# Activity 6

## “The” irrigation system!



Adapted by Alexandre Gomes.

Imagine that we have a small garden and we want to install a smart irrigation system, so it only waters in two situations:

- **Ideal Watering:** the ambient temperature is below 25°C, there is little natural light, the relative ambient humidity is above 80% and the soil humidity is below 50%.
- **Emergency watering:** the ambient temperature is below 25°C, there is little natural light, the soil humidity is very low, that is, less than 10% (regardless of the environmental conditions).

Continue building, with Python programming to measure environmental variables (such as temperature, humidity, among others) to control a water pump that simulates the smart irrigation system!

### 1 What materials are needed for construction?

- TI-Nspire CX II-T Calculator
- TI-Innovator™ Hub
- Smart Irrigation System Kit, consisting of:
  - Grove Temperature & Humidity sensor
  - Grove Moisture sensor
  - Grove Light sensor
  - Grove MOSFET and 4 AA battery holder
  - 4 AA batteries
  - Water pump
- For the garden prototype:
  - Plastic tray
  - Glass
  - Scotch tape
  - Straws
  - Earth
  - Plants
- rega.tns file

### 2 First... open a Python page, pre-configured for projects with TI Innovator™ Hub.

```
*rega.py
# Hub Project
#=====
from ti_hub import *
from math import *
from time import sleep
from ti_plotlib import text_at,cls
from ti_system import get_key
#=====
```



# Activity 6

## “The” irrigation system!

Adapted from Alexandre Gomes



### 3 Configure and test the water pump!

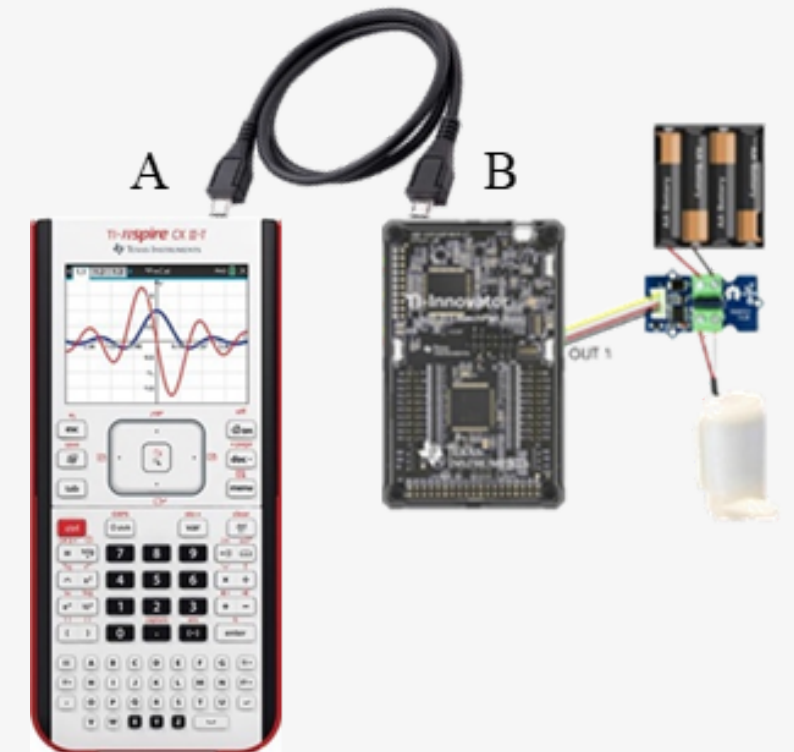
Before building and putting the entire irrigation system into operation, we must test the water pump.

#### How do you connect the water pump to the calculator?

Through the TI-Innovator™ HUB.

#### As?

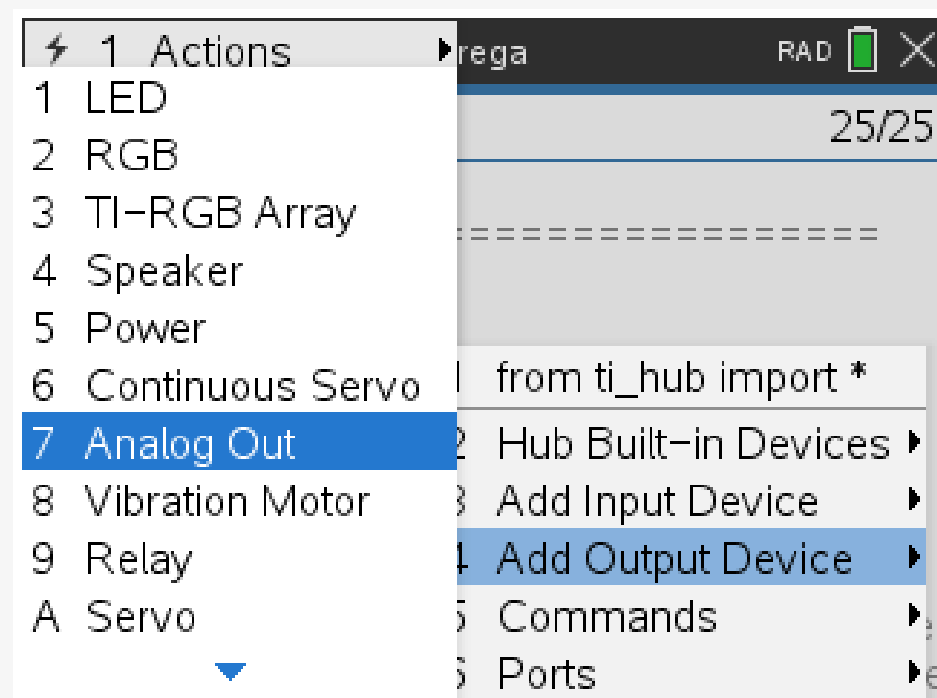
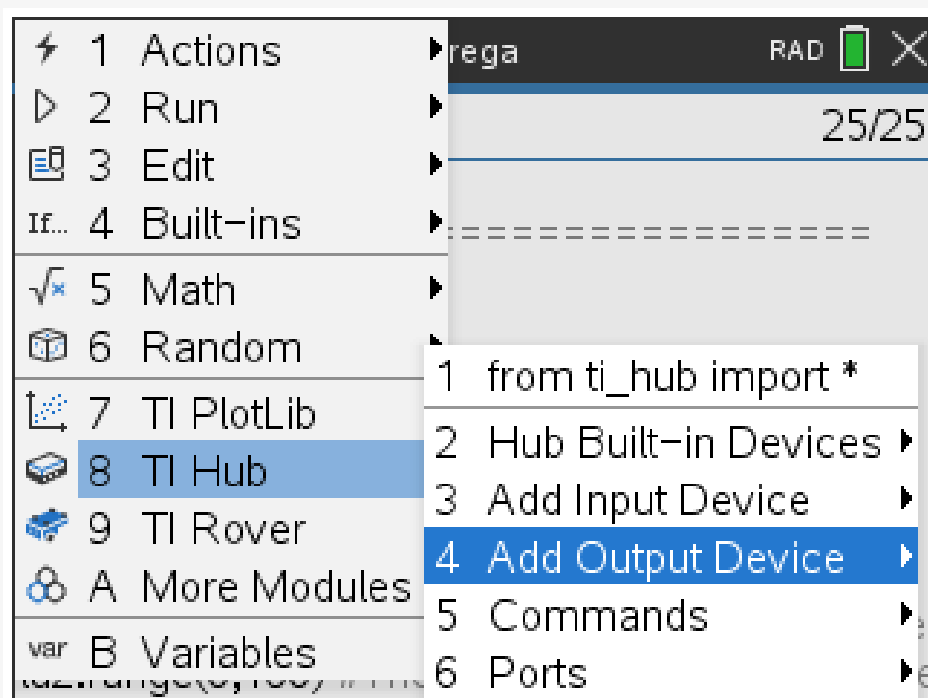
1. Using the mini-USB cable, connect the TI-Innovator™ HUB to the calculator, taking care to use end B to connect to the **Data** input on the TI-Innovator™ HUB, and end **A** to the calculator.
2. Connect the MOSFET to the **OUT 1** output of the TI-Innovator™ HUB, using one of the colored cables (note that they are all similar).



#### How to configure and test the pump?

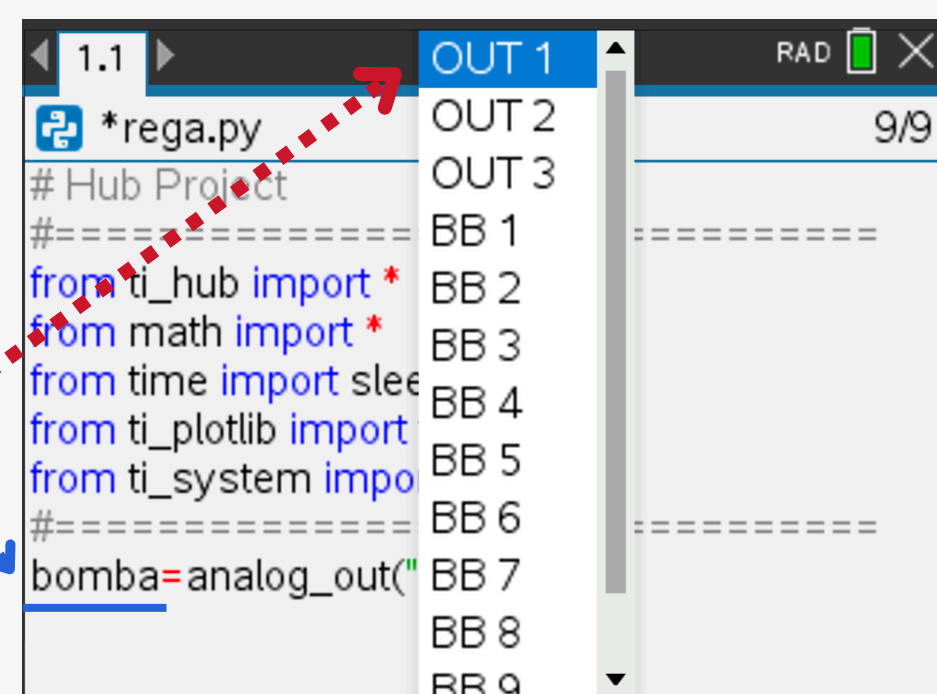
1. Select the analog output with which the pump will be configured

`menu` > 8: TI Hub > 4: Add output device > 7: Analog Out



2. Assign a name to the object, replacing var with the chosen name (**bomba** in this case)

3. Select, with the cursor positioned over (“port”), the OUT 1 output



# Activity 6

## "The" irrigation system!



4. To see all commands for the bomba object, press `.` after entering its name.

In this case, the `on()` option is used to turn on and `off()` to turn off the pump.

```

1.1 *Doc RAD 10/10
*rega.py
# Hub Project
#-----
from ti_hub import *
from math import *
from time import sleep
from ti_set(value) text_at,cls
from ti_off()      port get_key
#-----
bomba=analog_out("OUT 1")
bomba.
    
```

5. After having entered the commands to turn the pump on and off, in a similar way to what is shown in the figure to the side, you must add a waiting time between turning the pump on and off.

To do this, use the `sleep()` command, accessible through

`menu` > 8: THE Hub > 5: Commands > 1: `sleep(seconds)`.

The program should be similar to the one shown below (for the pump to run for 2 seconds).

To test: `ctrl` + `R`

```

1.1 *Doc CAPS RAD 10/10
rega.py guardado com sucesso
#-----
from ti_hub import *
from math import *
from time import sleep
from ti_plotlib import text_at,cls
from ti_system import get_key
#-----
bomba=analog_out("OUT 1")
bomba.on()

bomba.off()
    
```

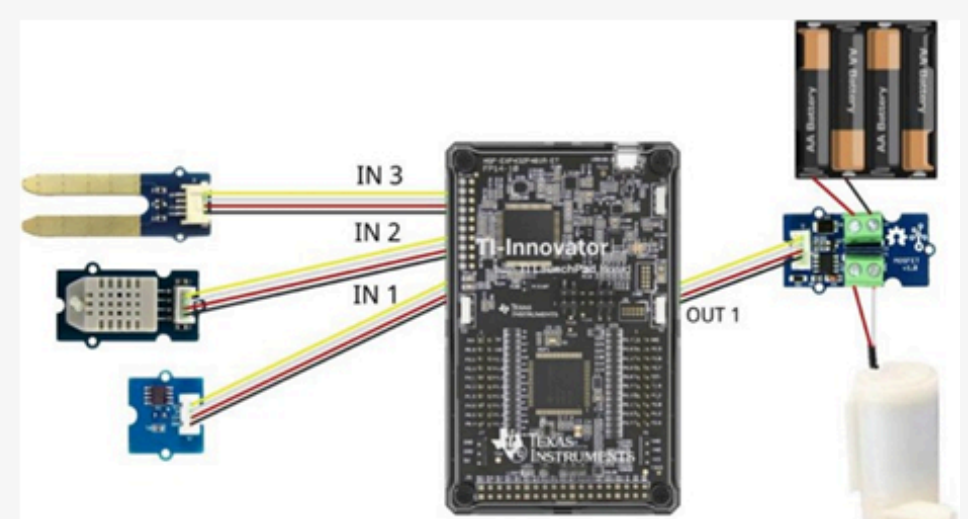
```

1.1 1.2 1.3 *rega RAD 13/14
bomba.py
from ti_hub import *
from math import *
from time import sleep
from ti_plotlib import text_at,cls
from ti_system import get_key
#-----

bomba=analog_out("OUT 1")
bomba.on()
sleep(2)
bomba.off()
    
```

### 4 Connect all sensors to the system.

At this stage, we can now connect all sensors to the IN 1, IN 2 and IN 3 inputs of the Hub, as shown in the following figure:





# Activity 6

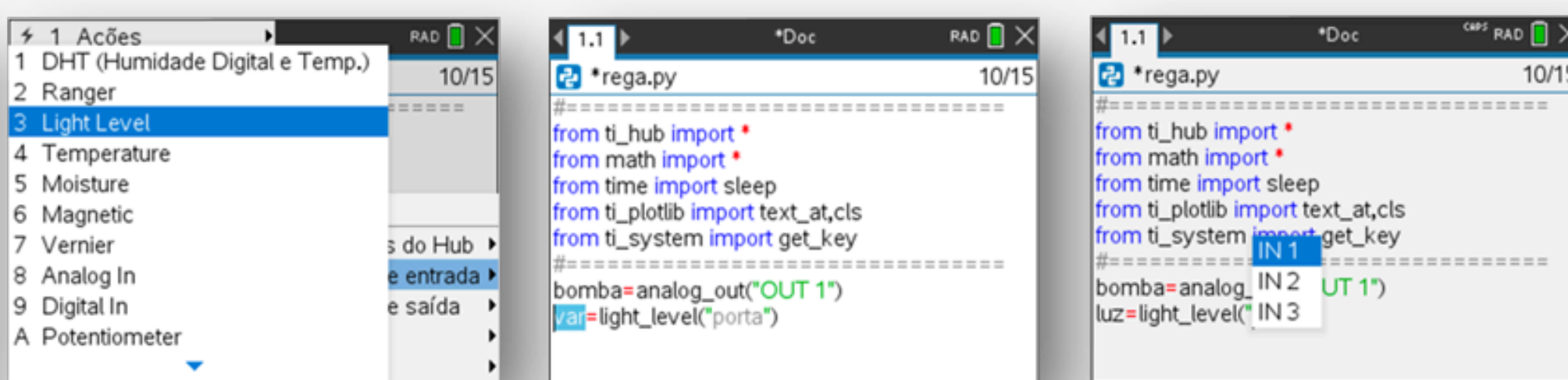
## "The" irrigation system!



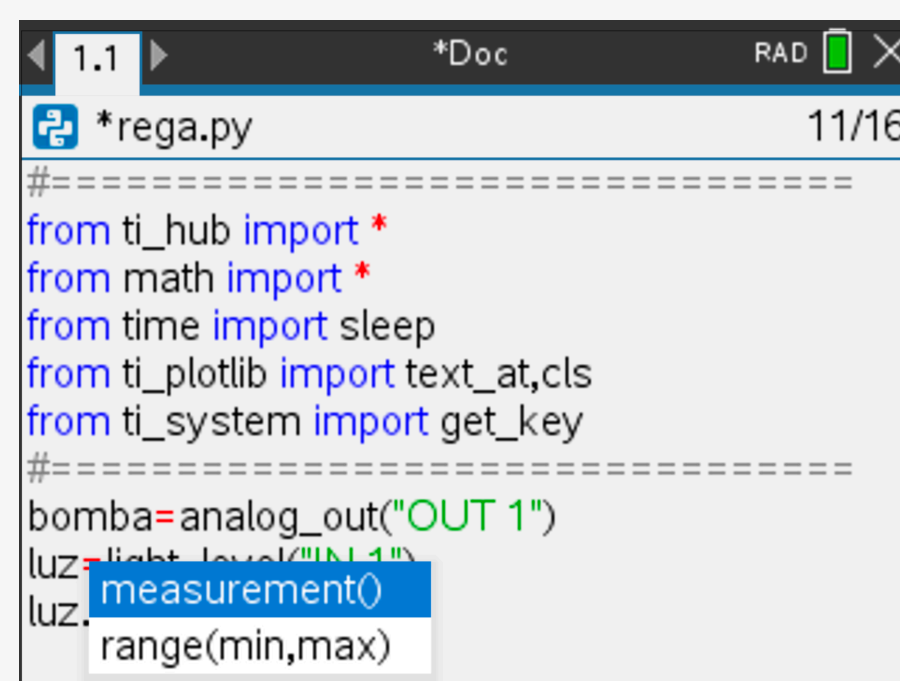
### 5 The automation...

Before being imported, for the desired program, you must start by configuring the luminosity, soil humidity and ambient temperature and relative humidity sensors. To do this select the...

- ... sensor **Light Sensor** com
  - ◻ menu > 8: TI Hub > 3: Add input device > 3: Light Level
  - Give the object a name, replacing **var** with the chosen name (**luz** in this case)
  - Select, with the cursor, the **IN 1** input:

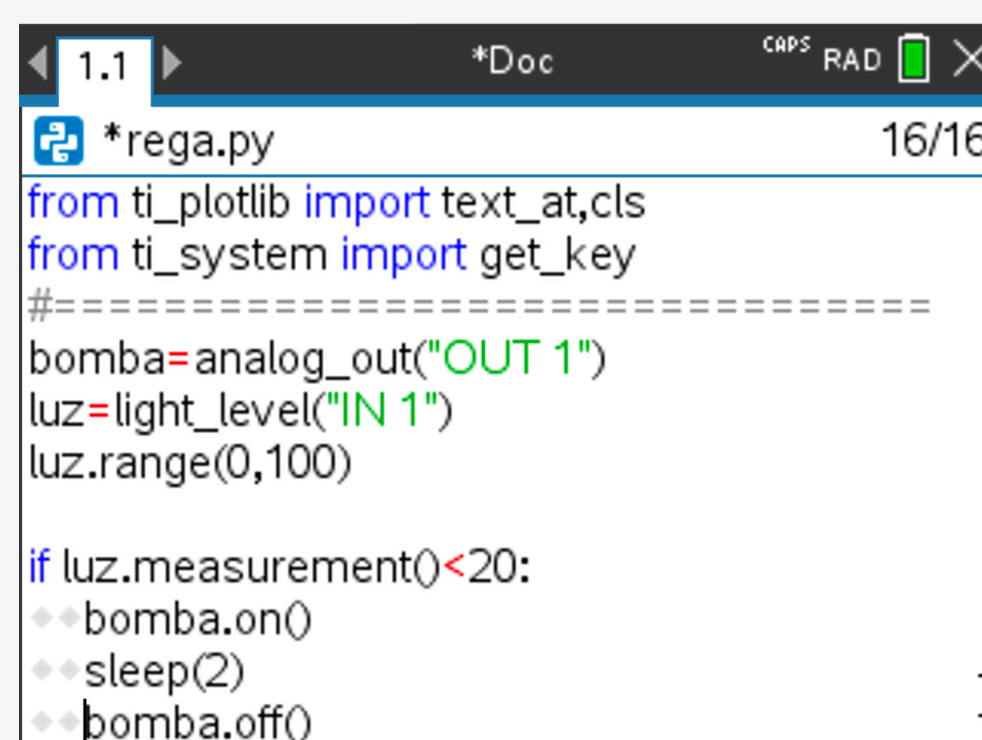


- To see all the object **luz** commands, press ◻ after entering its name. In this case, use the options **range(min, max)**, to reset the sensor scale and **measurement()** to obtain the luminosity reading.



- For the light sensor to consider values more predictable brightness levels, regardless of where we are, we can redefine its scale to vary between 0 and 100, recording values as if they were a percentage. To do this, we add the line **luz.range(0, 100)** right after the line that contains the sensor connection.
- So that the execution of the program is conditioned to the brightness value, a selection structure is added, which can be typed directly on the keyboard, or accessed from ◻ menu > 4: Integrated plans > 2: control > 1: if...

- The **ExprBoolean** argument will be replaced by the condition **light.measurement()<20**. All the commands that will be associated with this structure must have the same alignment in front, of 2 spaces (called indentation).





# Activity 6


## "The" irrigation system!



### 5 The automation...

The program only works once, it is not very useful as an automatic irrigation system.

To begin with, an infinite repetition structure is added, so that the program continually evaluates environmental conditions and decides whether or not the pump should be activated. To do this, the command **while get\_key() != "esc"** is placed at the beginning and serves exactly this purpose.

The program runs continuously until the key  is pressed, and the pump alternates between the active and inactive state by simply covering or uncovering the light sensor successively.

```
1.2 1.3 1.4 *rega RAD 18/20
* auto1.py
bomba=analog_out("OUT 1")
luz=light_level("IN 1")
luz.range(0,100)
while get_key() != "esc":
    if luz.measurement() < 20:
        bomba.on()
        sleep(.1)
        bomba.off()
```

Simply configure the remaining 2 sensors in a similar way to that done for the brightness sensor and add the conditions of the algorithm defined in the selection structure already created.

To make the program more responsive, the time allocated to the **sleep** command can be reduced to, for example, **0.1** seconds.

Pay attention to the respective indentations of each command line.

Then, the program already developed follows.

You can access the **A6.tns** file, either by reading the QR Code on the side or by clicking on it.



```
# Hub Project
#=====
from ti_hub import *
from math import *
from time import sleep
from ti_plotlib import text_at,cls
from ti_system import get_key
#=====
```

```
luz=light_level("IN 1") #The light sensor is connected to the Hub via port IN1.
luz.range(0,100) #The brightness scale varies between 0 and 100.
dht=dht("IN 2") #The temperature and relative humidity sensor is connected to the Hub via port IN2.
moist=moisture("IN 3") #The soil moisture sensor is connected to the Hub via port IN 3.
moist.range(0,100) #The humidity scale varies between 0 and 100.
```

```
bomba=analog_out("OUT 1")
```

**while get\_key() != "esc":**  **The second line observed continues after "80" in the first line.**

```
    if (luz.measurement() < 20 and dht.temp_measurement()<25 and dht.humidity_measurement()>80
and moist.measurement()<60) or moist.measurement()<10:
        bomba.on()
        sleep(.1)
        bomba.off()
```