

Workshop robot programming - Python

Versie 18 October 2018 EN (based on 14 juni 2017 NL)

Versie 30 August 2025 EN Python version

This document is subject to the “Attribution-NonCommercial-ShareAlike” license (CC BY-NC-SA 4.0).

See <https://creativecommons.org/licenses/by-nc-sa/4.0/> for more information

contact

Auteur: Jeroen Resoort,

E-mail: jeroen@resoort.net

Twitter: [@JeroenResoort](https://twitter.com/JeroenResoort)

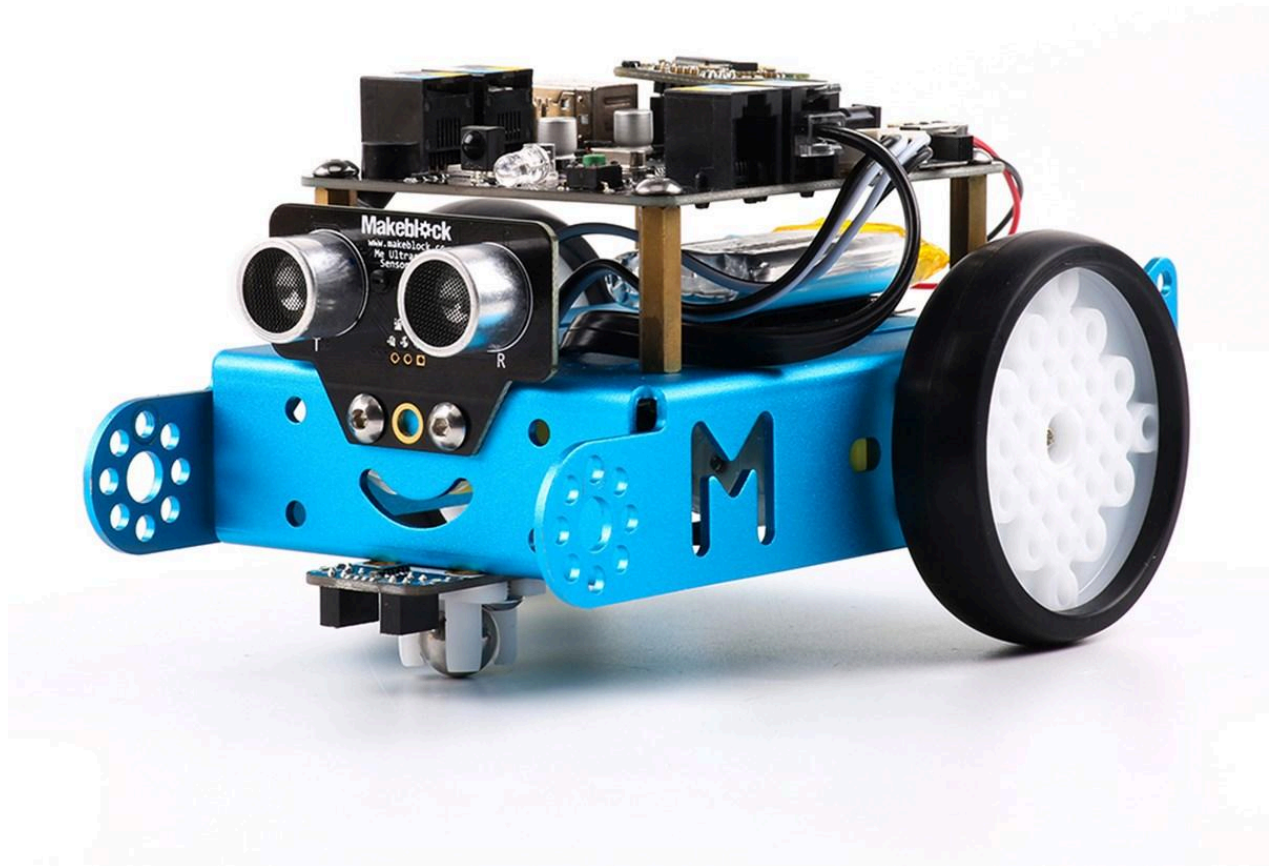
Robot programming

The mBot

The mBot is a little programmable robot. Programmable means you can tell the robot what it should do, using your computer. Today we are going to have the robot do a few assignments in this way.

The mBot has two engines with which we can turn the wheels. In addition, it has sensors at the front to see how far away something is, and a small loudspeaker with which we can make sounds.

Isn't that cool?



Python

To be able to program, we need a computer with programming software. For this we use Python. Python is a programming language that allows you to tell the computer what to do by writing instructions in text.

Assignment 1: Connecting & Flashing LEDs on your mBot

Welcome to your first programming challenge with the mBot!

We'll learn how to **connect to the robot**, **make its lights flash**, and then **change their colors** in a loop.

Open the file **assignment1.py** to use as a starting point.

Step 1: Connect to the mBot

The first thing we need is to connect to the robot.

Use the code below:

```
bot = findMBot()
```

This tells Python: "Find my mBot and let me control it!"

Step 2: Flash the LED Light

The mBot has small RGB LEDs (little lights) on its board.

We can turn them on like this:

```
bot.doRGBLedOnBoard(1, 255, 255, 255)    # White light
sleep(0.5)                                # Wait for half a second
bot.doRGBLedOnBoard(1, 0, 0, 0)           # Turn off
sleep(0.5)
```

This will make the light **flash**.

Now, let's turn on the mBot and run the program to see if it works!

Step 3: Try Different Colors

Change the numbers to make your own colors!

Each color uses three numbers: **Red, Green, Blue (RGB)**.

- The numbers go from **0** to **255**.
- **0** means that color is **off**.
- **255** is the **brightest** for that color.

Examples:

- (255, 0, 0) → Red
- (0, 255, 0) → Green
- (0, 0, 255) → Blue
- You can mix them, e.g. (255, 255, 0) → Yellow

Tip: If you go above 255, the robot won't understand—keep each number between **0** and **255**.

Step 4: Repeat with a While Loop

Instead of writing the same code over and over, we can use a **while loop**:


```
while True:
    bot.doRGBLedOnBoard(1, 255, 0, 0) # Red
    sleep(0.5)
    bot.doRGBLedOnBoard(1, 0, 255, 0) # Green
    sleep(0.5)
    bot.doRGBLedOnBoard(1, 0, 0, 255) # Blue
    sleep(0.5)
```

⚠ In Python, indentation (spaces at the start of the line) shows which lines belong together. Notice how the last 6 lines are indented (moved to the right with 4 spaces). That tells Python: “Do these again and again.”

⚠ Tip: If you need to **stop** the loop, press **Ctrl + C** in your terminal (click first for focus) or press the stop button. If that does not work click the trash can.

Your Task

1. Connect to your mBot.
2. Flash the LEDs **on and off**.

3. Change the colors at least 3 times.
4. Put it all inside a **while loop** so it keeps repeating.
5. Be creative! Try making your own light pattern.
 Tip: There are 2 leds, try to change the first value (left=1 and right=2)

Assignment 2: Driving the mBot Through a Maze

In this challenge, you'll learn how to make your mBot **move around** using the motors. We'll use the command:

```
bot.doMove(left_wheel_speed, right_wheel_speed)
```

- Positive numbers → move forward
- Negative numbers → move backward
- Different numbers for left and right wheels → turn!

Open the file **assignment2.py** to use as a starting point.

Example: Move Forward and Backward

```
bot = findMBot()
try:
    bot.doMove(100, 100)    # Move forward
    sleep(1)               # Wait 1 second

    bot.doMove(-100, -100) # Move backward
    sleep(1)

finally:
    bot.doMove(0, 0)       # Stop motors
```

Step 1: Try Moving Forward

- Use `bot.doMove(100, 100)` to go straight.
 - Try different speeds (like 50 or 200).
-

Step 2: Turn Left and Right

- To **turn left**, make the right wheel go faster than the left, e.g.:

```
bot.doMove(50, 100)
```

- To **turn right**, do the opposite, e.g.:

```
bot.doMove(100, 50)
```

Step 3: Build Your Maze Path

Use a mix of **forward moves** and **turns** to guide your mBot like it's driving through a maze. Think of it like giving the robot driving instructions!

Important: Possible damage!

Please make sure your robot is on the ground when testing your moves, robots can not handle falls very well!

Your Task

1. Fill in the blanks (___) to make your mBot move through a simple maze.
2. Use **forward, backward, left, and right turns**.
3. Adjust the `sleep()` times to change how far the mBot moves.
4. When your mBot finishes the maze, celebrate! 🎉

Assignment 3: Controlling the mBot with Your Keyboard

Now it's time to make your mBot move when **you press keys on the keyboard**!
We'll combine what you already learned about:

- Moving forward, backward, and turning 
- Using loops 

This way, your robot becomes like a little video game character you control!

Open the file **assignment3.py** to use as a starting point.

How Keyboard Control Works

We use a special `Keyboard()` object:

```
keyboard = Keyboard()
```

And can use `key = keyboard.get_key()` to find out which keys are pressed.

- `key` will be the name of the key you pressed ("`UP`", "`DOWN`", "`LEFT`", "`RIGHT`", "`SPACE`", "`ESC`").
 - You can check **which key** was pressed with `if`.
-

Understanding `if` Statements

When we control the mBot with the keyboard, we use `if` statements to check which key was pressed.

Think of it like asking the robot questions:

```
if key == "UP":  
    bot.doMove(200, 200)
```


This means:

➡ *"If the UP key was pressed, then move forward."*

⚠ Important: Indentation Again!

Just like with `while` loops, everything you want the `if` statement to do must be **indented** (moved to the right).

If you forget to indent, Python won't know what belongs to the `if`!

🎯 Your Task

1. Fill in the blanks with the correct numbers to move your robot.
2. Test your robot! Use:
 - **Arrow keys** to drive 🚗
 - **Space** to stop 🛑
 - **ESC** to quit ✖

Assignment 3 Bonus: Fun Functions with Keys

Let's make your robot even cooler! You can **play sounds and flash lights** while driving. For example:

H	Horn sound 🎺
L	Turn lights on/off 💡
P	Police lights + sound 🚓
Reverse (DOWN)	Play a sound while going backward 🗣️

🗣️ Playing sounds with `bot.playTone`

The `bot.playTone(frequency, duration)` command makes your mBot **play a sound**.

- **frequency** → how high or low the sound is (in Hertz, Hz).
 - Low number → low sound (like a bass)
 - High number → high sound (like a whistle)
- **duration** → how long the sound plays (in seconds)

Example:

```
bot.playTone(440, 0.5) # plays a 440 Hz tone for half a second
bot.playTone(330, 1)  # plays a lower tone for 1 second
```

💡 Tip: You can combine tones to make **simple songs or sound effects**!

🎯 Your Task

1. Add **keys** to your program. (Use code like `if key == "H":`)

2. Experiment with **different colors, sounds, and timings**.
3. Be creative — try combining lights and sounds for a mini show! 🎵💡

Assignment 4: Parking Sensor with the Distance Sensor

Now your mBot will act like a **car parking sensor**!

It will measure how far away an object is, and **beep when it gets too close**.

You can combine this with your existing moving robot program and continue adding code in **assignment3.py**

Step 1: The Distance Sensor

We connect the **ultrasonic distance sensor** like this:

```
distance_sensor = DistanceSensor(bot)
```

To read the distance (in **centimeters**):

```
distance = distance_sensor.get_distance()
```

Step 2: Use an **if** Statement

We check the distance:

```
if distance < 20:  
    bot.playTone(440, 0.2) # Beep if closer than 20 cm
```

Step 3: Put It in the existing Loop

We need to keep checking all the time:

```
while True: # this loop already exist  
    distance = distance_sensor.get_distance()  
    if distance < 20:  
        bot.playTone(440, 0.2)
```

Now your robot will keep watching for obstacles like a parking sensor!

Your Task

1. Make your mBot beep when it gets too close to an object.
2. Try different **distance limits** (10 cm, 30 cm, ...).
3. Experiment with different **beep sounds** and **durations**.
4. Extra: Make the beeps **faster** the closer you get to the obstacle, just like a real parking sensor!



Assignment 5: Self-Driving Car with Obstacle Avoidance

Your mBot can now **drive by itself** — just like a real self-driving car!

It will:

- Drive forward when the path is clear
- Detect obstacles with the **distance sensor**
- Turn when something is too close

Continue adding code in **assignment3.py**

Step 1: Keep Driving Forward

We use:

```
bot.doMove(100, 100)  # both wheels forward
```

Step 2: Detect Obstacles

We keep checking the distance:

```
distance = distance_sensor.get_distance()
if distance < 20:
    # too close!
```

Step 3: Avoid Obstacles

When the robot gets close (less than 20 cm):

```
bot.doMove(-200, 200)  # turn on the spot
```

Otherwise, keep going forward.

Your Task

1. Fill in the blanks to make your robot avoid obstacles.
2. Test it — does it stop and turn when something is in the way?
3. Change the **distance limit** (e.g. 10 cm, 30 cm) and see how it drives differently.
4. Challenge: Try different turning moves (left, right, spin, back up).

Assignment 6: Line Follower Robot

In this challenge, your mBot will **follow a black line on the floor** all by itself! It uses special **line sensors** underneath the robot to “see” where the line is.

Open the file **assignment6.py** to use as a starting point.

Step 1: The Line Follower Sensor

The robot can detect:

- **LINE_NONE** → no line detected
- **LINE_LEFT** → line is on the left
- **LINE_RIGHT** → line is on the right
- **LINE_CENTER** → line is under the middle (perfect!)

We check the line like this:

```
line = follower.get_line_status()
```

Step 2: Tell the Robot What to Do

We use **if** statements to make decisions:

```
if line == LINE_NONE:      # no line seen
    Bot.____                # do something
elif line == LINE_RIGHT:   # line to the right
    Bot.____                # do something else
```

Your Task

1. Fill in the blanks with the correct movement values.
2. Place your mBot on a black line (tape or marker line on the floor).

3. Watch it follow the path!
4. Experiment with different **speeds** to make it smoother.