

# EduRob Open Roberta Lab Tutorial Series

*EduRob Summer School, Technical University Graz*

*version 2, 2024/08/08,*

*Pavel Petrovič, Comenius University, [pavel.petrovic@uniba.sk](mailto:pavel.petrovic@uniba.sk)*

[robotika.sk/roberta/](http://robotika.sk/roberta/)

## Table of Contents

Lesson 01 .....	1
1 HELLO ROBOT! .....	1
2 PLAY A ROBOT .....	1
3 MOVE THE ROBOT .....	2
4 SQUARE.....	3
Lesson 02 .....	4
1 Reach the target .....	5
2 Design a new maze .....	6
3 Robot solves the maze!.....	6
4 More maze challenges for the robot .....	7
Lesson 03 .....	8
1 Letters .....	8
2 Maze solver.....	9
Lesson 04 .....	9
1 How robot moves a certain distance .....	10
2 Moving forward while doing something else .....	11
3 Meet the touch sensor! .....	11
4 Light sensor.....	12
5 Distance sensor.....	13
6 Robot explorer .....	13
Lesson 05 .....	13
1 Ferry .....	14
2 Robot parking .....	14
Lesson 06 .....	15
1 RECAP.....	16
2 Play a robot! .....	17
3 Explorer with a colour detector.....	18
4 A friendly agile dog .....	18
5 A friendly phlegmatic dog.....	19
Lesson 07 .....	19
1 Following a loop track.....	20
2 Track with left turns .....	21
3 SOLUTION1: Follow edge (1/3) .....	21
4 SOLUTION1: Follow edge (2/3) .....	22
5 SOLUTION1: Follow edge (3/3) .....	22

Lesson 08 .....	23
1 Line-following: SOLUTION1 with wait-until .....	23
2 Line-following: SOLUTION2: zig-zag with wait-until .....	25
3 Line-following: SOLUTION3: P-controller (1/2).....	26
4 Line-following: SOLUTION3: P-controller (2/2).....	26
5 Line-following: SOLUTION4: two sensors .....	27
Lesson 09 .....	29
1 Detect lines .....	30
2 Counting.....	30
3 Route to the station.....	32
Lesson 10 .....	33
1 A sequence of actions.....	33
2 Replaying the learned sequence.....	34
3 Recording melody and rhythm .....	37
Lesson 11 .....	38
1 Multiple robot demo .....	39
2 Duck parade.....	43
3 Your turn .....	47
Lesson 12 .....	48
1 Task description .....	48
2 Overall program structure .....	50
3 recordPattern algorithm .....	51
4 recordPattern function .....	51
5 comparePatterns algorithm.....	52
6 comparePatterns function .....	53
7 moreSimilar algorithm.....	53
8 moreSimilar function .....	54
9 main program algorithm.....	55
10 main program implementation .....	55

## Lesson 01

### Hello Robot!

Welcome to EduRob OpenRoberta Tutorial Series! 🤖

We invite you to a fascinating journey to meet robots and to program simulated robots in the following 12 lessons! 🤖

We hope you will have a lot of fun! 😊

If you are ready, start working and learning now! 👍

**Lernziel:** You will meet robots and Open Roberta Lab Simulation!

**Vorkenntnisse:** No previous knowledge is assumed

### 1 HELLO ROBOT!

Discuss in pairs or groups:

What are robots?

Why are they cool?

What are they able to do/ what not?

You can watch these videos to help you answer these questions:

[Evolution of Boston Dynamic's Robots \[1992-2023\]](#)

[Evolution of Boston Dynamics since 2012](#)

Have you heard about RoboCup?

[RoboCupRescue - long version](#)

### 2 PLAY A ROBOT

Volunteer for playing a robot, why not you?

The robot player will understand a set of basic commands. Which commands? Discuss with your class. Decide on short names for these commands. For example:

MOVE 1 STEP FORWARD	(STEP)
MOVE 5 STEPS FORWARD	(FIVE STEPS)
TURN-RIGHT	(RIGHT)
TURN-LEFT	(LEFT)

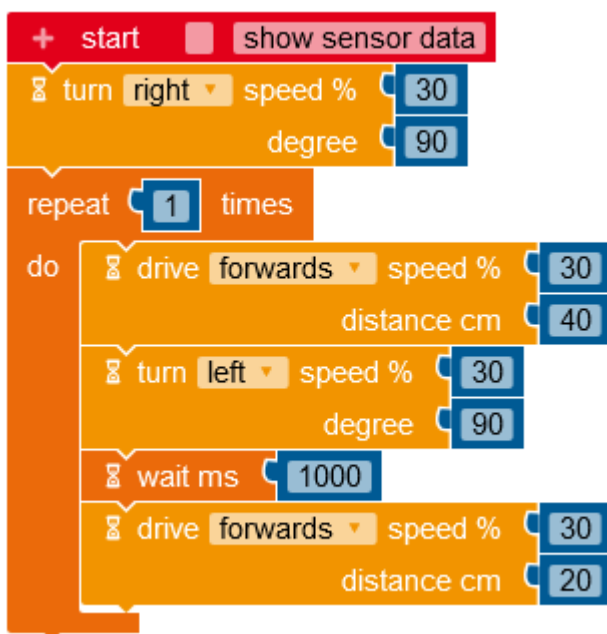
TURN-RIGHT-A-BIT (BIT RIGHT)  
TURN-LEFT-A-BIT (BIT LEFT)

You can shout out these commands and the robot player should perform them. Make him or her go in a circle. Make him or her go in a square. Make him or her reach the exit of the room from a special place in your classroom.

### 3 MOVE THE ROBOT

Now you solve the same task, but instead of controlling the robot player, you are going to control the robot on the screen!

Click the button at the bottom of this page to see the program that draws a shape of the letter L.



Copy the program, be careful to make an exact copy, all numbers should be the same! We want you to copy the program so that you learn how writing programs works here.

Remember you can always come back to this tutorial from the simulation by pressing the first button with the academic hat, which is currently selected and highlighted in the menu.



Now test the program in simulation:

1. click the SIM button



2. select the empty background image by clicking on the first button in the toolbar



3. switch on drawing of robot trajectory so that we can observe the path the robot travelled (enable robot draw trail)



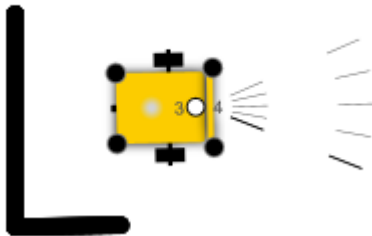
4. start the program by pressing the PLAY button underneath the simulation arena.



Are you able to see the robot moves in the shape of letter L?

You can drag the robot sideways with mouse/finger to see the full letter it has drawn.

If you are not satisfied with the result, you can reset the position of the robot with the last button in the bottom toolbar, make some changes in the program and try again.



**TASK:** Experiment with the program! Can you make the shape of the letter L a bit higher? Can you make it a bit wider? Do you understand all the blocks that form the program? Do you understand the meaning of all the values in those blocks? If not, try changing them to figure out! Draw several different shapes of letter L and share the result with your friend or the teacher.

If you are ready and fast:

- Can you make a program that will draw several letters L next to each other without starting the program more than once?

#### 4 SQUARE

**TASK:** Now change the program to make the robot go in a square.



Notice that the square will not be 100% perfect. That is because robots are normally operating in a real-world, where the conditions are not 100% under control. Sometimes the floor is a little bit uneven at some place, or a bit more slippery, also the wheels can be a bit dirty, and many other small details may vary from place to place. In this simulator, real conditions are simulated as well!

Your simulated robot is not 100% perfect exactly for the same reason as why the real robot is also not 100% perfect.

In this step of the tutorial, you now have access to all the blocks in the beginner mode. If you cannot find the right blocks, move to the previous step of the tutorial to see where to find them.



Always share your success with your friend(s) or the teacher!

If you are fast and ready, try also:

- moving in a smaller square first, then larger square, even larger, and larger... how many squares can you draw?
- moving in the shape of a rectangle, triangle, hexagon, octagon, ...
- drawing a grid of A x B squares

**Congratulations!**

*You have reached the end of the first EduRob Open Roberta Lab tutorial.*

Remember, this is only the beginning.

You are now in the beginning your journey through amazing world of Robotics, when ready, open the next tutorial.

You can also come back here to review the material. Always formulate your own ideas, goals, challenges, never stop and never give up! Good luck!

**THANK YOU!**

## Lesson 02

### **Maze navigation**

Welcome to the second tutorial in the EduRob series!

Have you ever visited a cave?

Or some labyrinth in an entertainment park or somewhere else?

We will program the robot to find the golden treasure in a maze.

**Lernziel:** You learn about how a robot can navigate a maze

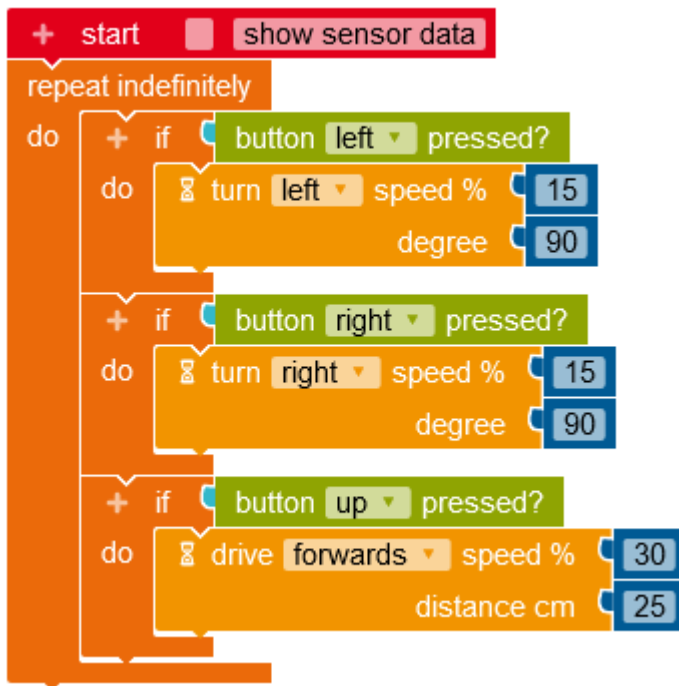
**Vorkenntnisse:** You should already know some basic robot movements

## 1 Reach the target

Open the simulation window and load the maze background image (lesson2\_maze\_background.png) using the load background button in the top simulation window toolbar.



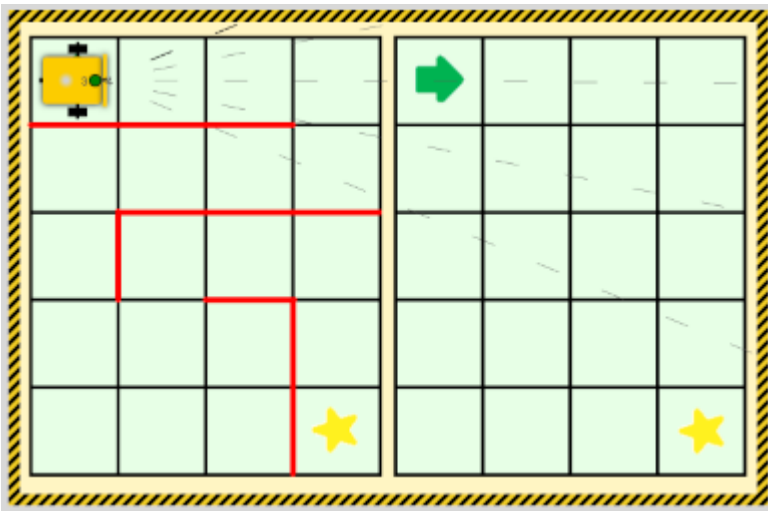
Look at the program on the left (*in case you do not see it, import the program button\_robot\_navigation, or create the program on your own*), it should look like this:



Place the robot at the green arrow, open the robot view using the button at the bottom toolbar, move it on the side so that you see the simulated arena. Please also enable robot draw trail, and run the program.







**TASK:** Bring the robot from the start of the maze to the end by pressing the arrow buttons in the robot view window.

If you cannot bring the robot cleanly to the end the first time, reset the simulation view with the button in the bottom toolbar, move it to the start and try again.

Hint: at the end, disable drawing the trail and move the robot away to see the whole path of the robot.

## 2 Design a new maze

Let your friend to create a new maze for you in the right half of the background image.

Let him or her use the tool in the simulation window to draw coloured area: mark the unpassable walls with red colour.

**TASK:** Place the robot at the start of the second maze, enable drawing of the trail, the robot view, and solve this maze by using the arrow buttons in the robot view.

If you are ready and fast:

In addition to coloured walls, have your friend to place some obstacles in the maze as well, and try to solve a maze with both walls and obstacles.

Discuss with your friend or in the class:

- which mazes are more easy and which mazes are more difficult?
- can you make a maze that cannot be solved?
- what do all the mazes that can be solved have in common?
- what do all the mazes that cannot be solved have in common?

## 3 Robot solves the maze!

Place the robot at the start of the first maze.

One square of the maze measures 25 cm.

The following program moves the robot to the end of the first alley, turns right (90 degrees), follows one cell forward, turns right again, and moves one cell forward again:

```
graph TD; Start[+ start] --> Show[show sensor data]; Show --> Drive1[drive till the end of the first alley]; Drive1 --> Drive1a[drive forwards speed % 30 distance cm 75]; Drive1a --> Turn1[turn right South]; Turn1 --> Turn1a[turn right speed % 15 degree 90]; Turn1a --> Drive2[drive one cell forward South]; Drive2 --> Drive2a[drive forwards speed % 30 distance cm 25]; Drive2a --> Turn2[turn right West]; Turn2 --> Turn2a[turn right speed % 15 degree 90]; Turn2a --> Drive3[drive one cell forward West]; Drive3 --> Drive3a[drive forwards speed % 30 distance cm 25];
```

**TASK:** finish this program so that the robot can reach its target.

If you close this tutorial, you can import the program from the main menu Edit - Import program... select the program maze\_navigation\_stub. Test the program first. When you are done, you come back here. It is a good opportunity to practice navigating between the programming mode and the tutorial, let's go!

#### 4 More maze challenges for the robot

Use the maze created by your friend in the right half of the maze background image.

**TASK:** create a new program that can solve the maze created by your friend.

If you are ready and fast:

- Let your friend to leave some obstacles in the maze, change the program so that the robot does not crash to the obstacles, does not pass through any wall, and reaches the target. Be creative in setting up the rules of your game.

Wonderful, you have reached the end of the maze tutorial.



Did you like it?

When you have time, you can proceed further, we will practice the robot movements yet a bit more so that we are all confident about how to let our robot move around!

## Lesson 03

### Interesting shapes and movements

Welcome to the third tutorial in the EduRob series!

Have you seen some engineering drawings of buildings, or products engineers design?

Do you know anyone in your circles who can program plotter machines?

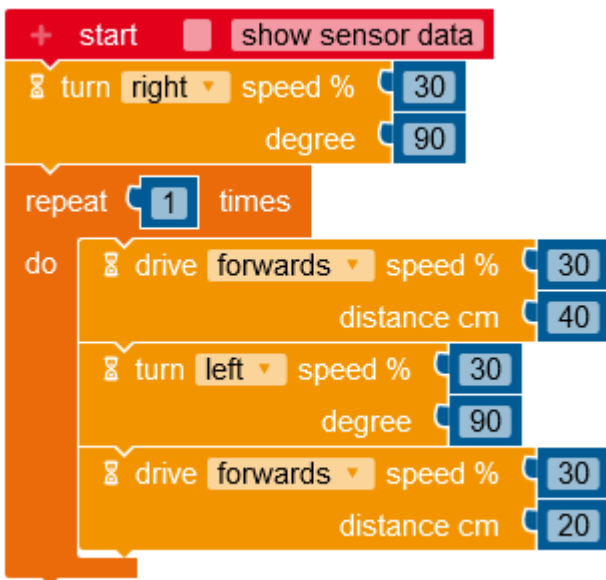
We will program the robot to draw some more interesting pictures in the simulation area.

**Lernziel:** You will recap the basic robot movements so that you are confident in using them

**Vorkenntnisse:** You should already know how basic robot movements work

### 1 Letters

Open an empty background image. Recall the program that draws the letter L from the first tutorial (you can import the program draw\_L).



**TASK:** modify this program to draw different letters. Select three different letters and share your program with others!

Example letters to start with: V, U, Z, T, P, N, M, H, K...

If you are ready and fast:

- Can you draw a shape of some word, LOL? LOVE? COOL? :-)
- Have the robot wait at the end of each letter before it moves to the beginning of another one, where it will again wait for a while. This will allow you to disable and re-enable drawing of trail.

## 2 Maze solver

Open the empty simulation background image.

Ask your friend to place two to four obstacles in the simulated environment, mark the start with the green rectangle and the end with the red rectangle.

**TASK:** make a program for your robot starting from the green rectangle moving around the environment without touching any obstacle, and reaching the target red rectangle.

Now the background image does not have a grid. You need to estimate the proper distances in some other way.

If you are ready and fast:

- Let your friend create a new maze for you.

Discuss with your friend or class:

- Can you design a situation that is impossible to solve?
- Which situations are difficult to solve and which situations are easy?

Wonderful, you have reached the end of the maze tutorial.

Did you like it?

When you have time, you can proceed further, we will now meet very important building blocks/electronic elements that robots use to get information from their environment. Do you know what those are?

## Lesson 04

**Robot, look around!**

Welcome to another EduRob robot tutorial.

Until now we were only programming robots to perform fixed pre-programmed actions.

Robot could not sense - could not read, hear, touch, smell, in fact it had absolutely no idea what was around it.

It was the programmer who knew how the environment looked like, and programmed the robot in such a way that it performed proper movements in that particular environment.

Such robots have a very limited use. They can only be used in situations that are always the same. They are not flexible to adjust to changing conditions.

Real world always changes, it is often unpredictable, and therefore, robots will usually have some way how to "look around". They use sensors!

In this lesson, we are going to use sensors in simple tasks.

If you are ready, let's move to the first step.

**Lernziel:** You will learn how robot interacts with its environment

**Vorkenntnisse:** You should already know how to control basic robot movements

## 1 How robot moves a certain distance

The simulated robot that we are using here is already equipped with several different types of sensors.

First, we are going to use a "bumper" - a touch sensor that has two states "pressed" or "released". When the robot runs into an obstacle, the bumper becomes pressed and so the robot program can detect that and react with some action.

Let's open an empty room background image and create a program that the robot will run until it will hit a wall (or another obstacle), and then it will back-up 10 cm, to get away from the obstacle.

So first we need to get the robot to start moving forward. However, the blocks that we were using until now will not help us...!



Please recall how this block works: place the block right after the start of the program and test it in the simulation and then come back here.

This block made the robot move forward. However, it required a specific distance - in this case 20 cm to be entered. This block will make the execution of the program to pause at this block until the robot travels 20 cm forward, and only after the robot will reach the final location, and stop, then the execution will continue to the next block of the program. Obviously, this is not what we want, because **we do not know how far is the obstacle**, and we want to detect it at any time while moving forward! With this block, if the obstacle is on the way, the robot will crash to the obstacle, its wheels will be spinning, rubbing the floor while still pushing forward and not moving anywhere. Switch to the simulation and try it out!

Fortunately, this is not the only way how to make the robot drive forward...

## 2 Moving forward while doing something else

We start with studying the following example. The robot will start moving, and then while it will be moving forward for several seconds, it will be changing the color of its light. After some time, it will finally stop.

Enter the following program and test it (program name: `move_and_change_color`).

```
graph TD
    Start[+] --> Show[show sensor data]
    Show --> Drive[drive forwards speed % 30]
    Drive --> Repeat[repeat 4 times]
    subgraph Loop [ ]
        direction TB
        Green[turn brick light colour green on]
        Wait1[wait ms 400]
        Orange[turn brick light colour orange on]
        Wait2[wait ms 400]
        Red[turn brick light colour red on]
        Wait3[wait ms 400]
    end
    Repeat --> Green
    Green --> Wait1
    Wait1 --> Orange
    Orange --> Wait2
    Wait2 --> Red
    Red --> Wait3
    Wait3 --> Off[turn brick light off]
    Off --> Stop[stop]
```

Notice the difference in the drive block - the first command after the start. Here, the block does not contain information about **how much** it should travel forward. The robot will be moving until the program will tell the robot to stop.

## 3 Meet the touch sensor!

Now we will modify this program a little bit: we remove the *repeat loop* part completely, and replace it with a command that will monitor the bumper **touch sensor** and wait until the bumper will be pressed. After that, the program will proceed to the next block and stop the robot.

```
graph TD
    Start[+] --> Show[show sensor data]
    Show --> Drive[drive forwards speed % 30]
    Drive --> Wait[wait until get pressed touch sensor Port 1]
    Wait --> Stop[stop]
```

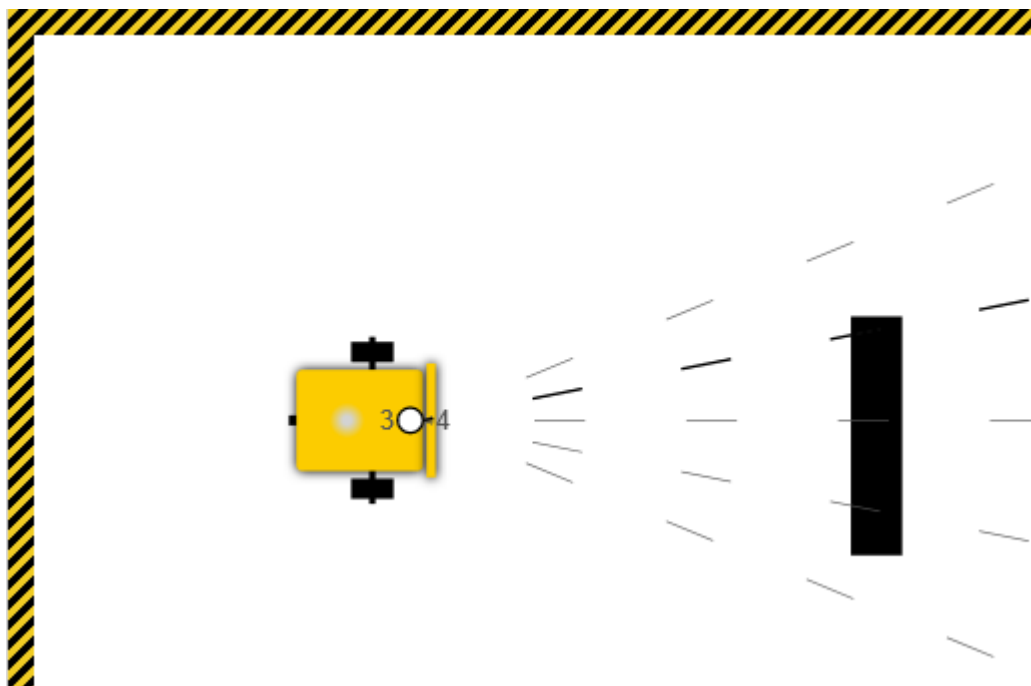
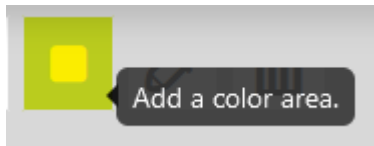
Place an obstacle at some distance in front of the robot and try to run this program. Describe what happened, and which blocks in the program made it happen.

**TASK:** modify (extend) the program so that when the robot hits the obstacle, it will back up 10 cm, and then stop.

#### 4 Light sensor

Another sensor that our robot has, is a **colour sensor**. Among other things, it can detect the colour of the floor that the front part of the robot is driving over.

Open the empty background image and create a black line in some distance from the robot.



Replace the "get pressed touch sensor" block with a colour comparison command, which looks as follows:

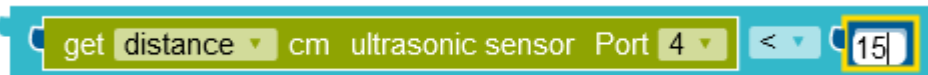


Test your program, that it stops on the line, and then backs-up 10 cm.

## 5 Distance sensor.

Finally, the robot also has a distance **ultrasonic sensor** that is transmitting an ultrasonic wave and measuring its reflection from various surfaces. In this way, the sensor is capable of detecting obstacles even without running into them.

Replace the condition in the wait until block with:



Then run the program and observe what happens.

Did it hit the obstacle?

## 6 Robot explorer

Modify the program with the distance sensor so that after the robot detects the obstacle and backs-up, it will turn at least 100 degrees, and starts moving forward again, until will detect an obstacle...

The program will repeat this sequence in an infinite loop. It will run forever, unless you will only stop it manually using the red stop button.

Hint: use the "repeat indefinitely" block.

If you are ready and fast:

- modify the program so that it does not always turn the same angle, but generates the amount of turning randomly.
- replace the condition waiting for the distance sensor with a color sensor and place the robot in a closed polygonal area, which it will never quit. Share your solution with your friends and the class.

Congrats! You have just completed another EduRob tutorial that introduced the use of sensors.

We cannot wait to meet you in the next one!



## Lesson 05

**Useful robots!**



Welcome to the next lesson of the EduRob tutorial!

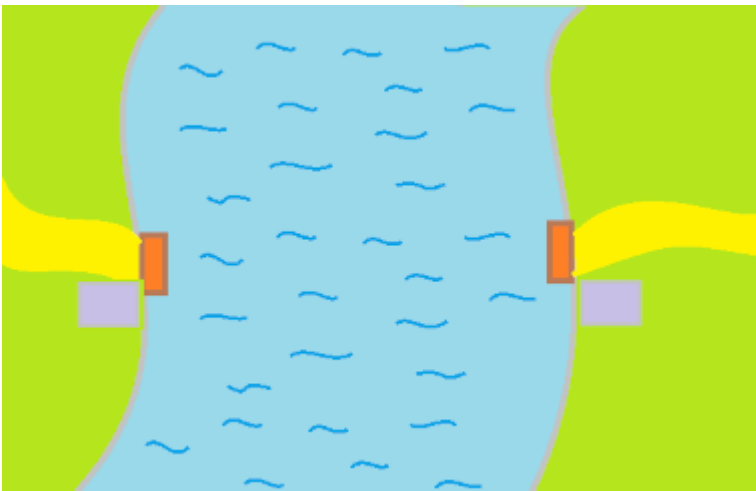
Have you ever had to cross a river using a ferry?

**Lernziel:** You will learn how to use sensors to program interesting useful robots

**Vorkenntnisse:** You should have a basic understanding of robot movements and sensors

## 1 Ferry

Open the background image of the river (lesson5\_river\_bacground.png). Place the robot on one of the pier pointed in the direction across the river. Place some obstacles on the grey-marked locations so that the robot will be able to detect them using the ultrasonic distance sensor.



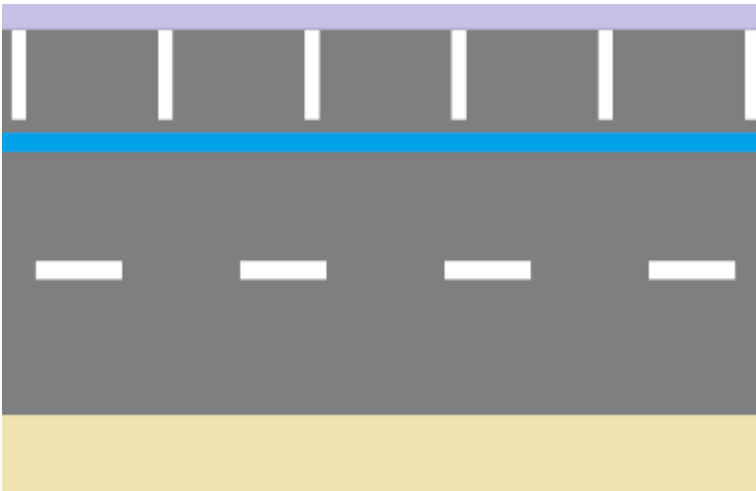
**TASK:** implement a program for the ferry. The robot waits parked next to a pier in the water until "the passangers get on", then sails in a straight move to the other bank, turns 180 degrees, waits for the passangers to get off and get on the boat, and the process is repeated.

If you are ready and fast:

- Have the ferry announce each arrival to the bank by showing its name on the display, and making a sound.
- Have the ferry wait with the departure until it receives a signal. The signal may be moving an obstacle from the enbankment further away - or you can come up with another interesting way.

## 2 Robot parking

Load the road with parking slots background image (lesson5\_parking\_bacground.png). Place the robot on the road. Fill all the parking places except of one with obstacles that are blocking them, representing parked cars.



**TASK:** Write a program for the robot that will park to an empty spot. It should always turn 90 degrees in the direction of the parking place to check if it is empty. If it is empty, it will stop searching and park, otherwise, it will move to the next parking stop. Use the *repeat until* block. The length of each parking location is about 41 cm.

Congratulations! You have completed yet another EduRob Open Roberta Lab tutorial!



It is exciting to follow you on your journey. Keep it on!

## Lesson 06

### Sit down

You already know how to use sensors to detect some situation.

This allows us to wait until something happens - for example robot ferry reaches the peer of the port, or a parking place is free.

However, in some situations, we do not just want the robot to wait until something happens.

We want the robot to "look" what is the situation now, and depending on what the robot learns (detects), it should react either with one or another response.

In addition to the very useful blocks *wait until* or *repeat until* that we used in the previous tutorials, we will now learn a new way of how to control the program flow depending on the sensor readings.

**Lernziel:** In this lesson you will learn how to run different parts of program depending on some condition

**Vorkenntnisse:** You should know the basic control commands for loops, movements, and sensor readings

## 1 RECAP

To recap what we already know:

We know how to make the robot drive forward or backward a certain distance at a specified speed:



or turn a specified angle at a specified speed:



Both of them block the execution of the program until the movement is finished.

We also know the blocks that cause the robot to start moving or start turning, but without blocking the program, continuing to the next commands in the program immediately, and keeping the robot in this movement until another drive or stop block:



We also know how to make some part of the program to repeat a specified number of times, infinitely or until some condition gets satisfied:



```

repeat indefinitely
do
  turn brick light colour green
  on
  wait ms 500
  turn brick light off
  wait ms 100

```

```

repeat until
  red = get colour colour sensor Port 3
do
  show picture eyes closed
  wait ms 100
  show picture eyes open
  wait ms 100

```

A similar command that waits for something to happen, but does not allow anything to be happening while the program waits is the *wait until* command:

```

+ wait until
  get pressed touch sensor Port 1

```

Our programs were always running in the prescribed order and sequence, the conditions were only deciding how long something will be repeated or not.

In this tutorial, we are going to use **conditions** to "branch" the program execution: depending on some situation / condition, the program will continue with a different path.

## 2 Play a robot!

In this exercise, we will again need a volunteer.

Go ahead and be the volunteer and have some fun!

Now the rest of us are going to decide on several rules that the robot player will follow. When you shout start, the player will start his or her performance, moving slowly and making sure that all rules are followed.

Example rules:

- IF you see the bin THEN turn right
- IF you see the notice board THEN move forward
- IF you are near something THEN turn left
- IF you are near a chair
  - THEN sit down
  - ELSE say "HELP"

After you create a set of rules that will make the volunteer sit down at his or her place, you can select another volunteer.

### 3 Explorer with a colour detector

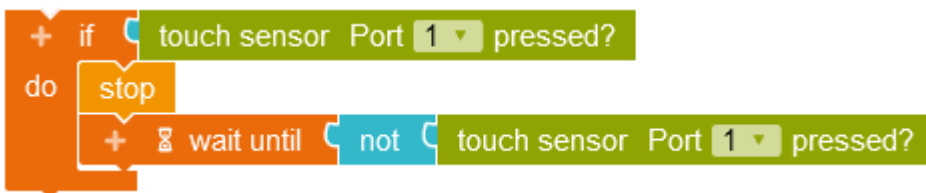
Recall the task of robot explorer where the robot drives until the next obstacle, bounces from it (backs-up, turns, and starts moving in a different direction).

Load the program again, and test it.

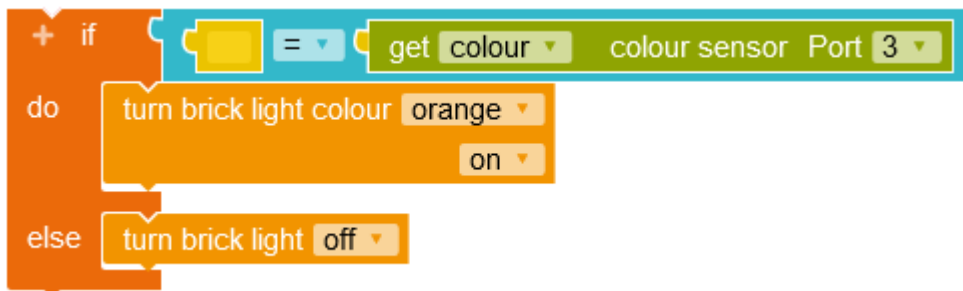
Note: Remember to always save your programs. If you did not, you can import it from *explorer\_with\_ultrasonic* or *explorer\_with\_bumper*.

**TASK:** modify the behaviour of the robot to respond both to the obstacles using ultrasonic sensor or a bumper, but also to a coloured areas using the colour sensor. If the robot is passing over a coloured area, it should emit sound (beeping). Notice, the obstacle can be placed also inside of a coloured area and the robot should still bounce off that obstacle.

Use one of the two branching blocks *if ... do* or *if ... do ... else*:



```
if touch sensor Port 1 pressed?  
do  
  stop  
  wait until not touch sensor Port 1 pressed?
```



```
if get colour colour sensor Port 3 = yellow  
do  
  turn brick light colour orange on  
else  
  turn brick light off
```

If you are ready and fast:

- Use the version of the explorer program that is using *colour sensor* and stays inside of the coloured polygon. Modify it so that when the robot meets an obstacle, it stops, produces sound, and waits until the obstacle is removed before it continues.

### 4 A friendly agile dog

Think of the robot as being a friendly pet dog, which likes his owner so much that it wants to always stay in his or her vicinity. The dog is permanently observing its owner and reacting to his or her actions:

- when the owner is coming closer than certain critical distance (say 30 cm) to the dog, the dog backs-up

- when the owner is moving away from the robot, further than some critical distance (say 50 cm), the dog follows

**TASK:** implement the program for the robot dog, use one loop and two decision blocks **do ... if**.

If you are ready and fast:

- Make the robot produce sounds, and show images on the display depending on the situation

## 5 A friendly phlegmatic dog

The program for the dog we constructed in the previous step is probably making the dog a bit neurotic, it never sits, it always runs - forward, or backward. Such dog would get exhausted very soon, and it would also be a bit annoying for the owner, if his or her dog is always moving.

**TASK:** change the program so that the robot behaviour will match a bit modified set of rules:

- when the owner is coming closer than certain critical distance (say 30 cm) to the dog, the dog backs-up
- when the owner is moving away from the robot, further than some critical distance (say 50 cm), the dog follows
- *new:* when the distance to the owner is between 30 and 50 cm, the robot just sits down where it is, waits, and observes what happens.

Hint: Use two **if ... do ... else** blocks and one loop block.

# Congratulations!

Now you can not only use the **loops**, but also the **if - then** and **if - then - else** conditions!

**Conditions are very useful control structures.**



**See you in the next tutorial!**

## Lesson 07

### Line Following

**Welcome back to another in the EduRob series tutorial!**

Today, we open a very useful topic: **robot line following**.



This skill will allow us to use the robot in many situations, solve many tasks, participate in competitions, and of course have a lot of fun!

Are you ready?

Let's go!



**Lernziel:** You learn how a robot can follow a line with one sensor

**Vorkenntnisse:** You need to know the basics about robot movements and sensors

### 1 Following a loop track

Recall how the colour sensor works. It returns the colour that the sensor sees at that very moment:

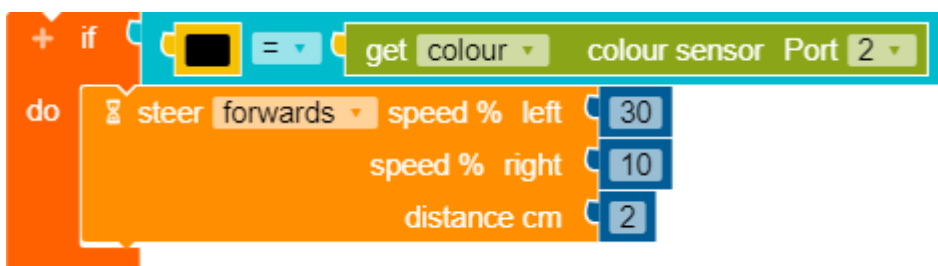


We can compare it using the compare block to any of the basic colours, for example to black:

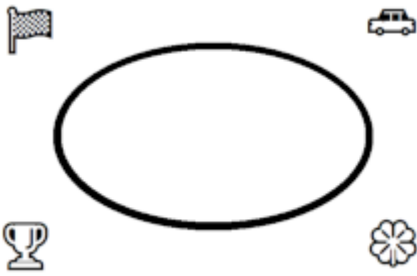


This blocks tell us "yes, this is true", or "no, that is not true" - depending whether the sensor sees black or not.

We can use this logical yes/no answer in the IF condition that allows us to execute some commands only if the condition is satisfied:



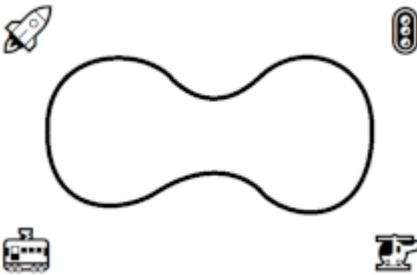
**TASK:** Now try to make a complete program that follows a looping track. Load the background image from the file `lesson7_oval_track.png`.



**HINT:** when the sensor sees the black line, the robot should move forward, when it does not see the black line, it moves forward while turning right to get back on the line.

## 2 Track with left turns

Try to run your program on a track that contains both left and right turns  
(`level7_intermediate_track.png`):



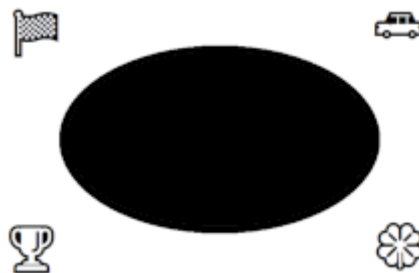
The robot keeps following the line while it is straight and contains right turns.

But it fails to follow it when it encounters a left turn!

Discuss together how could we tackle this problem - there are many ways what can be done, try to find as many as possible!

## 3 SOLUTION1: Follow edge (1/3)

In our first solution to the line-following problem, we will change the track: instead of following a line track, we will try to follow the contour of a large filled area (`lesson7_filled_oval_track.png`):



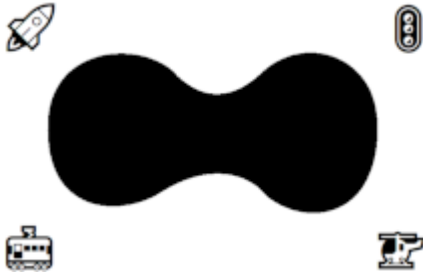
Try your original program that could follow the looping track on this filled area track. Does it still work? Why?



If you are ready, do not wait and proceed to the next step!

#### 4 SOLUTION1: Follow edge (2/3)

Let's try to change the shape of our filled area a little bit (lesson7\_filled\_intermediate\_track.png):

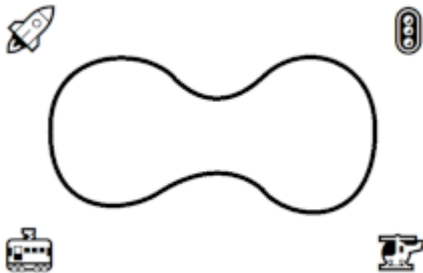


Does the original program still follow the contour of the filled area? If not, why? What happens?

Can you propose a new version of the program that will follow the contour of the intermediate track shape correctly?

#### 5 SOLUTION1: Follow edge (3/3)

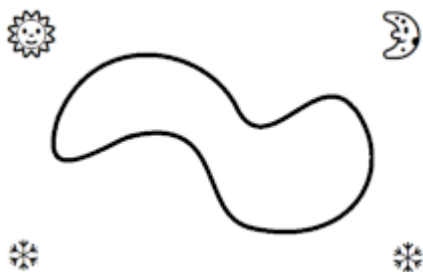
Now, let's change the area again, and make it a looping track with both left and right turns (lesson7\_intermediate\_track.png):



Does the robot follow the line of this track properly?

If you are ready and fast:

- try also the following track (lesson7\_difficult\_track.png):



Did your program solution require any modifications?

Share what you have learned with your friend and your class.

# Congratulations!



You should now have a very good idea about what it means to follow a line with a single colour sensor!

We will come back to this interesting topic also in the next tutorial, stay tuned!

## Lesson 08

### More Line Following

In the previous tutorial, we learned how to follow a line in one particular way - by following an edge.



Today, we are going to see some other interesting ways how to implement line-following. They may be more preferred in certain situations.

We hope you are as curious as we are!

# Let's go!

**Lernziel:** You learn how a robot can follow line in different ways.

**Vorkenntnisse:** You should have completed the tutorial about basic line following.

## 1 Line-following: SOLUTION1 with wait-until

In the previous tutorial, we have considered the problem of following a line, and we found a solution that is based on following an edge of that line. Thus the algorithm was as follows:

1. start on the line
2. while you are on the line, move forward left, until you leave the line
3. while you are outside of the line, move forward right, until you enter the line again
4. continue from step 2

To implement this algorithm, we have used the **if ... do ... else** condition block.

```
+ start
show sensor data
drive forwards speed % 30
repeat indefinitely
do
+ if
= get colour colour sensor Port 3
do
steer forwards speed % left 10
speed % right 50
else
steer forwards speed % left 50
speed % right 10
```

Load one of the more advanced tracks and test the program again to make sure it works.

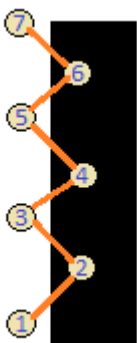
Now. If we look at the 4-step algorithm above, we see that its structure more resembles the **wait ... until** block than the conditional block in the program above here.

**TASK:** Rewrite this algorithm to work exactly the same, but instead of **if ... do ... else** block it will use the **wait ... until** block.

Let's look at the following sequence of observations that describe the process of this edge following:

*light, left of the line*  
*dark, on the line*  
*light, left of the line*  
*dark, on the line*  
*light, left of the line*  
...  
*etc.*

The following picture shows possible positions where the sensor would be reading the colour of the floor, always causing the steering to navigate towards the other side of the edge:



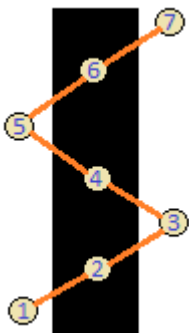
Yes, this is edge-following. How would this pattern look like for a line following? If the robot sensor would try to trace the full line, not just the edge? Alternately leaving the line on the left edge, and on the right edge? Write it down and discuss it with your your friend and in the class. Only then move to the next step.

## 2 Line-following: SOLUTION2: zig-zag with wait-until

OK, so if the robot would be following a line, not an edge, the sequence could look like this:

*light, left of the line*  
*dark, on the line*  
*light, right of the line*  
*dark, on the line*  
*light, left of the line*  
*dark, on the line*  
*light, right of the line*  
*dark, on the line*  
...  
*etc.*

The trajectory of the robot sensor above the line then could look like this:



Notice the robot would now be changing the direction of its movements less frequently. It waits until it gets into the line, does not change its direction, but waits until it gets out of the line (on the right side), then...

1. start on the left of the line
2. while you are outside of the line, move forward right
3. while you are inside of the line, keep moving forward right
4. change the direction of the movement to forward left
5. while you are outside of the line keep moving
6. while you are inside of the line, keep moving
7. change the direction of the movement to forward right
8. and continue from the step 2

**TASK:** write a program that implements this kind of line-following. Try to do it as simple as possible by modifying the previous program with adding two **wait...until** blocks by simply duplicating them and arranging them in the proper order.

If you are ready and fast:

- Modify one of the programs in such a way that the robot will stop following the line, if it meets an obstacle. Which program is easier or more reliable to modify and why? Share your findings with your friend and the class.

### 3 Line-following: SOLUTION3: P-controller (1/2)

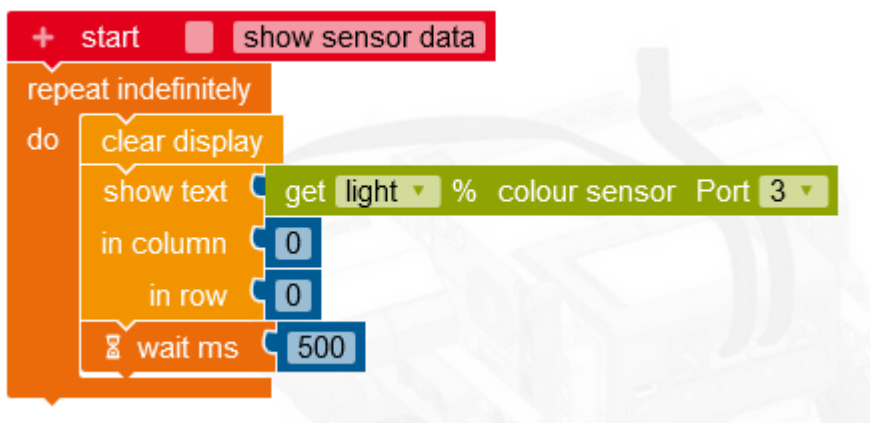
We continue our journey with a different and surprisingly efficient idea for line-following algorithm using a single sensor. But let's introduce this story properly first.

Can you recall that the colour sensor that we used to detect colours (and thus lines on the floor) can also report another type of information?

Anyone remembers what was that? What else apart from the colour can it detect?

Yes! It can detect the amount (percentage) of reflected light. And this signal is much more useful than just a colour: black or not black. Here, the sensor can read the amount of brightness (light colour) of that particular spot illuminated by the sensor.

Construct the following program and run it in an arena with a black line.



Make the EV3 display visible - the value read by the sensor will be shown on the display. Manually move the robot over the edge of the line using the mouse or a finger. What values can you see? How many different values can you make the display show?

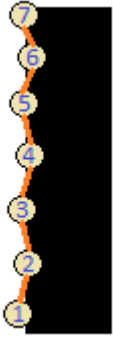
Thinking about what you have just seen, how could we use this method to make the robot follow the line - properly turning left and right as the line indicates?

Discuss with your friend or class and then move on to the next step.

### 4 Line-following: SOLUTION3: P-controller (2/2)

Great, so you already have some idea...

The algorithm is based on an idea that could be explained by the following picture:



As the sensor enters a little bit "into the line", the amount of reflected light is decreased. As it starts to leave the line, the amount of reflected light is increased. If the sensor is close to the ideal edge center, it should only steer a tiny little bit - gently left or right to return to the edge. If the sensor is further away from the edge, it should probably steer a little bit more. So we could say that in order to stay exactly on the edge the robot should steer left or right with the magnitude proportional to the difference from the reading in the center of the edge.

**TASK:** Create a program that can follow the edge using the algorithm/idea described above!

You may need to use some calculations, which you find in the Math block palette, for example:



Try to come up with your own solution. If you cannot get it working, see the help below.

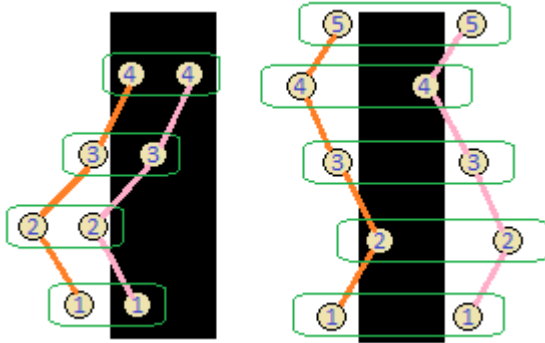
*A side comment: Using the colour sensor in this reflectance mode is often a better idea than using it in colour mode. Colours are sometimes unreliable: reflections from the lights around, or mechanical properties of the surface can easily confuse the sensor to report a different colour. Using the reflected light is more robust and brings more information. We typically use the colour-mode of the sensor only when we really need to detect colours. For example, if a dog sees a yellow brick, it should bark, if it sees a red brick, it should howl, etc.*

If you are ready and fast:

- Experiment with your solution on different tracks. Can it follow all the tracks in all directions? What needs to be changed in order to succeed? Share your findings with your friend or the class.

#### 5 Line-following: SOLUTION4: two sensors

Finally, if we can offer two sensors for the task of line-following, it becomes a bit more convenient. Yet, we need to compare the width of the line with the distance of the two sensors we are going to use. Look at the following picture, and try to formulate how the width of the line influences the strategy and what should the efficient strategy to follow a line with two sensors look like:



Try to fill the following table and give a clear direction of the movement the robot should take in each situation:

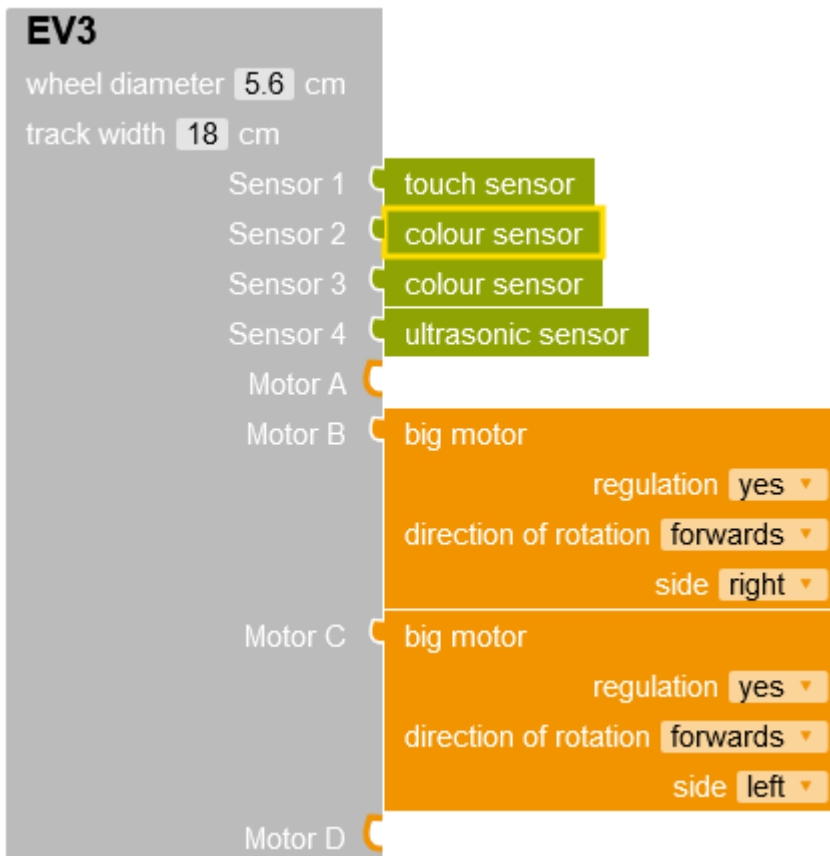
situation	direction
<input type="checkbox"/> <input type="checkbox"/>	
<input type="checkbox"/> <input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/> <input type="checkbox"/>	
<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	

**TASK:** Implement a program that follows the line using the two light sensors.

Note: You may need to change the configuration of your robot, go to:

**ROBOT CONFIGURATION EV3basis**

and replace the gyro with another light sensor - it will be automatically mounted on the front, next to the one on port3. You can also experiment with adding a third light sensor and use the two which are on the borders of this 3-sensor group.



If you are ready and fast:

- Try to implement the ideas from the previous tutorial step (P-controller) with two sensors. How could that be done? Share your findings with your friend and the class.
- Can you think about yet another way to follow a line using one or two sensors? Can you demonstrate that?

## Congratulations!

**Now you are an expert on line-following algorithms!**

There is much more to explore about programming simulated robots  
and we can't wait to see you in the next tutorial!



Lesson 09

**Storing information**



Welcome to another in the series of the EduRob Roberta tutorials!

Today, we will start working with data. The robot will not only react to what is around in the environment.



It will also be able to **remember** something!

If you are ready,

let's go!

**Lernziel:** You learn how to use variables

**Vorkenntnisse:** You should have some elementary robot programming experience

## 1 Detect lines

In the previous tutorial, we have already worked with numbers. When the amount of steering was proportional to the difference between the reflected light returned by the sensor and the optimal value on the edge - the robot program performed some calculations... In this tutorial, we are going to work with numbers a little bit more, and it will allow us to program more interesting tasks.

But let's start with something simple first.

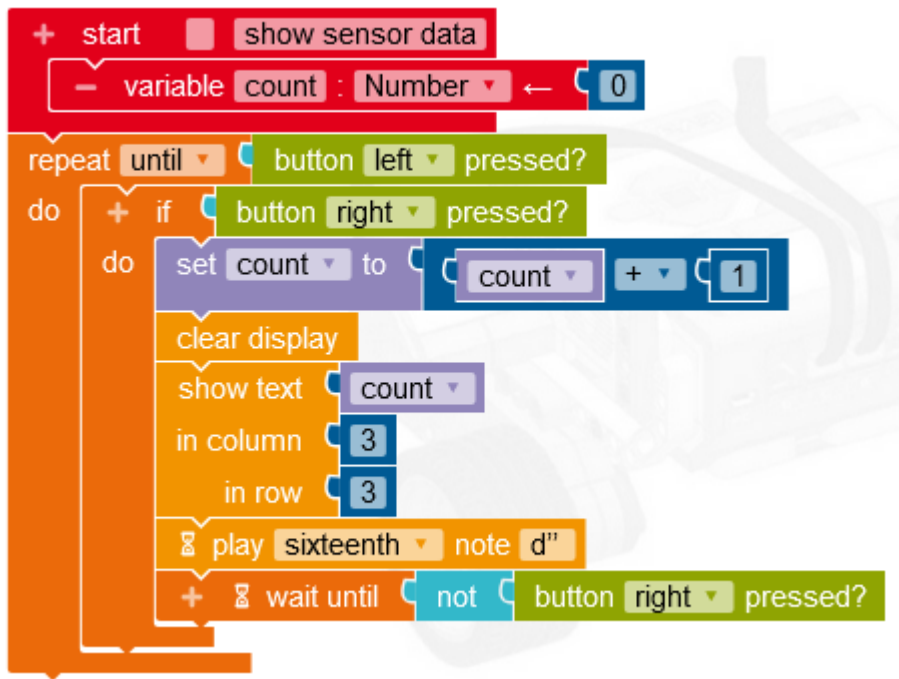
Load the [lesson9\\_line\\_patterns.png](#) background image and place the robot on one of the green arrows, or alternately further to the right to skip some of the lines, and place an obstacle at the marked position.

**TASK:** Create a program for the robot so that the robot will start moving straight towards the obstacle and each time it will pass over a black line, it will produce a sound. When the robot will reach the obstacle, it will stop, and the program will finish. The program should work from any location on the tracks, not only from the green arrows.

**HINT:** use the **repeat ... until** and the **if ... do** blocks.

## 2 Counting

We will now see how to use variables. Enter the following program (alternately import it from [count\\_right\\_arrows](#)), open the EV3 brick view and test it.



The program will beep each time you press the right arrow button, it will count the number of times you press it, and it will show that number on the robot display.

The program uses the variable **count** which is increased every time the button is pressed. The variable can be used as any other number, so we can show it on the display with **show text** block.

Notice that we also need to wait until the user will release the button to prevent increasing the variable many times for a single press of the button. Try removing the **wait until** block and see that now the button presses are not always counted properly.

**TASK:** Modify the program from the previous step (driving over the lines) so that when the robot will reach the obstacle and stop, it will make as many beeps as is the number of lines it drove over.

Test the program from different starting locations. Does the speed of the robot influence its capability to produce the correct result? How about the width of the line? Try to cover part of some lines with a white color area. Are they still always detected? Discuss your findings with your friend and the class.

If you are ready and fast:

- Can you write a program that will help the robot to return to approximately the same place as where it has started by going backwards and counting the lines? The robot will first travel over the lines to the obstacle, do its beeping, and then travel back to where it has started.
- Add some more lines with different colours. Create a program that will count both colours separately. It will first beep one colour, and then the other.
- Combine the two tasks above: two colours + returning back. Some new blocks from the Logic palette might be useful!

Share your findings with your friend or the class.

### 3 Route to the station

In this step, we will combine different parts we already know in a small project.

Robot is located at crossroads. It would like to reach the railway station. However, the robot does not know the right way - which turn should it take? Therefore, it will ask a local villain for the help:

**Robot**> *Hello my friend, how can I get to the railway station?*

**Villain**> *Take this smaller road, then turn to the right, follow the path and then you get to the station.*

**Robot**> *Thank you, and which turning should I take?*

**Villain**> *Let me press your right button several times to indicate to you which turn you should take.*

**Robot**> *Thank you, please do!*

**TASK:** Load the background image [lesson9\\_route\\_to\\_station.png](#). Place the robot at the green arrow. Write a program that will wait for several presses (let's say K) of the right arrow button followed by a single left arrow press. Then the robot will travel by following the black line until it will reach K-th right turning, where it will take that turning, and continue travelling by following the black line until it will see an obstacle in front of it (the railway station), when it will stop.



If you are ready and fast:

- Change the program so that the villain will not have to press the right arrow button, but it will wave its hand several times in front of the ultrasonic sensor.
- Modify the program so that the villain will not have to press the left arrow button to finish the sequence, but after it will stop pressing/waving for some time, the program will automatically understand that the sequence has finished. Hint: use the timer.



## Congratulations!

You should now understand what is **a variable** and how a variable can be used to **remember some value** - for example a number, how it can be **set to some value, changed to another value**, and **used** at any place, where a constant value (= a number) could be used instead.

We will work more with variables in the remaining tutorials, stay tuned!

## Lesson 10

### Remembering more data

Welcome again in the Roberta EduRob tutorial series!

As we are getting familiar with programming the robots, we can start making them more intelligent.



We can program the robot to remember a sequence of things happening around!

If you are curious how this can be done,

**just come in and start!**

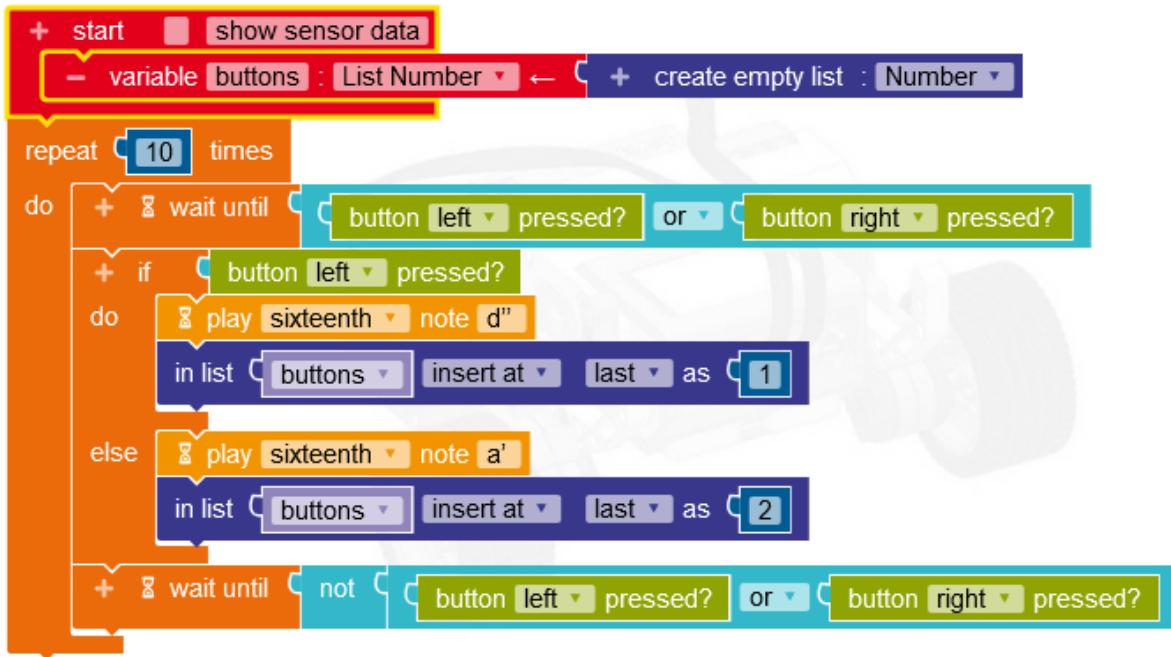
**Lernziel:** You learn how to work with the lists

**Vorkenntnisse:** You should know basic decision and loop blocks, and what is a variable

### 1 A sequence of actions

In the previous tutorial, we have seen how a robot can remember and manipulate some numerical value for later use. Sometimes we need to remember more than just one single number. Sometimes we would like to remember a whole list of them!

Look at and study the following program ([sequence\\_of\\_actions](#)):



Can you figure out what does it do?

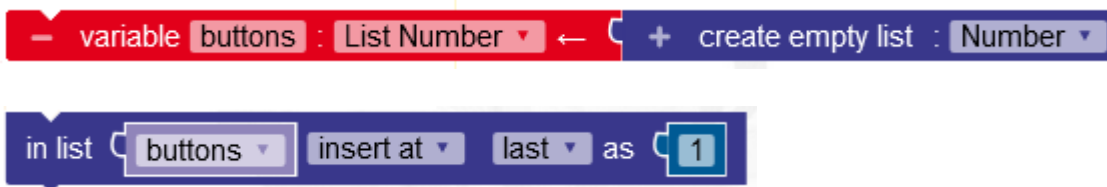
You can try to run it - open the "robots view" in the simulation view, run the program, and press the left or right arrows on the brick for up to 10-times before the program terminates. What happens when you press those arrows?

Try it out, and then proceed to the next step.

## 2 Replaying the learned sequence

Yes, the program reacts with the sound depending whether we press the left or the right arrow. Nothing special, right?

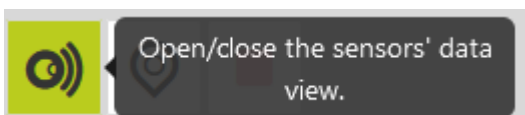
But the dark blue blocks do something more.



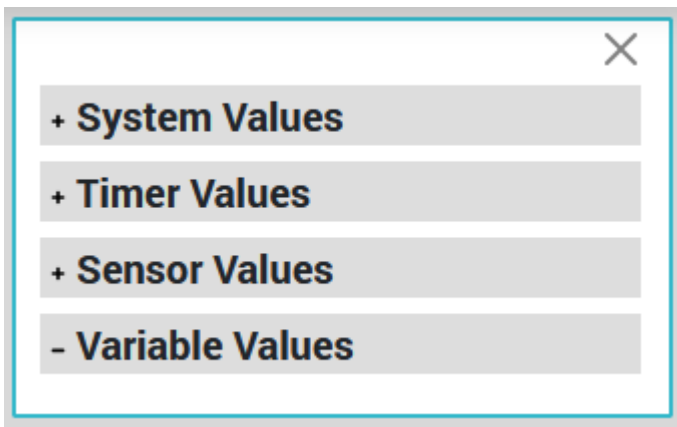
Let's have a more detailed look what exactly they do. Open the simulation view in debug mode.



Then, in the bottom toolbar open the sensors/data view.



And finally make sure the variables are shown (there is a minus sign next to the variables section of the view).



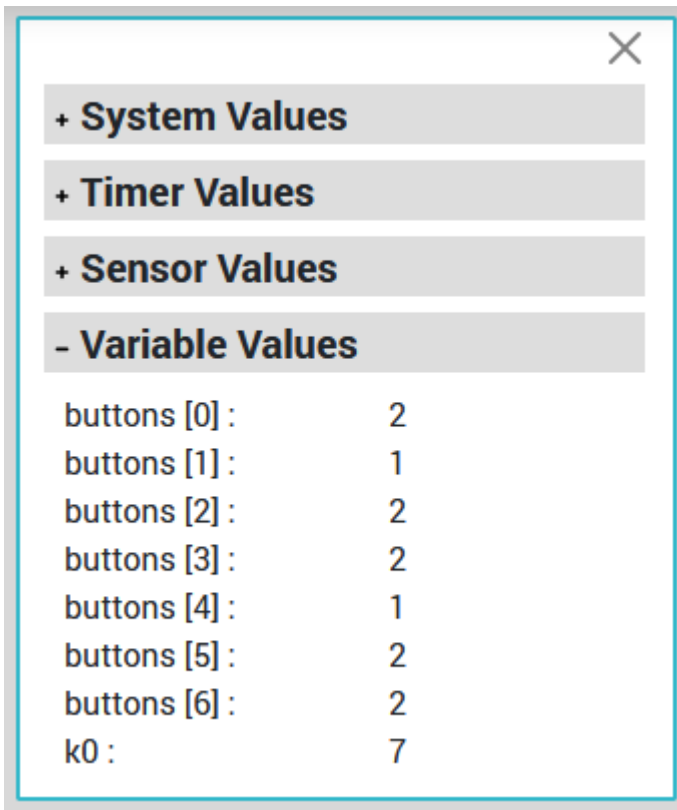
This allows us to see the values of the variables as the program runs in the debug mode. Before running the program open the robot's view,



and finally run the program in this debug mode.



Now you can press the left and right arrows and observe what happens in the variable values...



Oh, I see... Every time you press the left arrow button, the number 1 is appended at the end of the list, and every time you press the right arrow button, the number 2 is appended. Observe also the path of the execution in the program, the debugger nicely highlights the blocks that are being executed. You can also click at any of the blocks in the program to mark it red (setup a breakpoint), and the debugger will pause the program when the block will be reached. Then you can continue the program step by step, block by block. Please try it out, it is a very useful feature!

*Notice that you can use the sensor/data view also when running the program in the usual simulation view (without debugger). We just wanted to show you the debug mode, which also shows you what the program is doing.*

Great, but why is this list useful at all?

**TASK:** complete this program. After 10 presses of the buttons, the program should replay the same "melody": playing the *note d*" for the left presses, and *note a'* for right presses. However, in the second round, the user will not be pressing the buttons. Instead, the robot will use the values stored in the list *buttons*, and replay the same sequence from its memory.

*Note: it can be done in many different ways, but one way to do it is to use the following two blocks:*



If you are ready and fast:

- Change the program so that it will replay the learned sequence more than once.
- Try to find at least three different ways to solve this extra task.
- Modify the program to do something else - be creative!

Share your findings with your friend or in the class.

### 3 Recording melody and rhythm

Great, and now we will make the program even more fun - it will not only remember the melody, but it will also remember the rhythm!

To achieve that, we will need to store in a second list, the time intervals that pass between the individual button presses. How to measure time?

Look at the following program (import `measure_time` program):

```

+ start
- variable duration : Number ← 0

repeat indefinitely
do
+ wait until button right pressed?
reset timer 1
play sixteenth note d"
+ wait until not button right pressed?
+ wait until button right pressed?
set duration to get value ms timer 1
play sixteenth note d"
wait ms 2000
play sixteenth note d"
reset timer 1
wait ms duration
play sixteenth note d"

```

You can also test it and see what it does.

Let's read it together:

1. The program uses one numeric variable *duration*.
2. It first waits for the right button to be pressed (and released).
3. After the press, it resets the *timer1* (sets it to 0 value), timer is always running...
4. It also plays a short sound after the first press.
5. Then it waits for the second press of the right button.
6. After the second press, it reads the value of the *timer1*, and stores it to the variable *duration*.



7. It also plays a short sound after the second press so that we can hear how long was the interval between the two button presses - exactly the same as the time between the two sounds.
8. Then it waits 2 seconds to separate the first time interval with its repetition.
9. Finally it proceeds with the repetition: beeps, waits the same amount of time (reading the variable *duration*) and then beeps again.
10. (the reset timer here is there just to make the program execution be the same as in the first case to make the timing more precise, but this block can be safely removed).

Alright, hopefully now we see how measuring the time works!

**TASK:** Update the program you created in the previous step to not only record the melody, but also the rhythm!

HINT: use a second list, which will store the time elapsed between the individual button presses as measured by the timer.



## Congratulations!

You have completed another tutorial.

Now you should have a good idea about what **lists** are good for and how to use them!

See you soon in the next tutorial!

### Lesson 11

#### Multiple robots

Welcome to another Roberta EduRob tutorial!



Today, we are going to play with more than one robot at a time!

We believe you will love this exciting feature of the Open Roberta Lab simulation.

# Try it out!

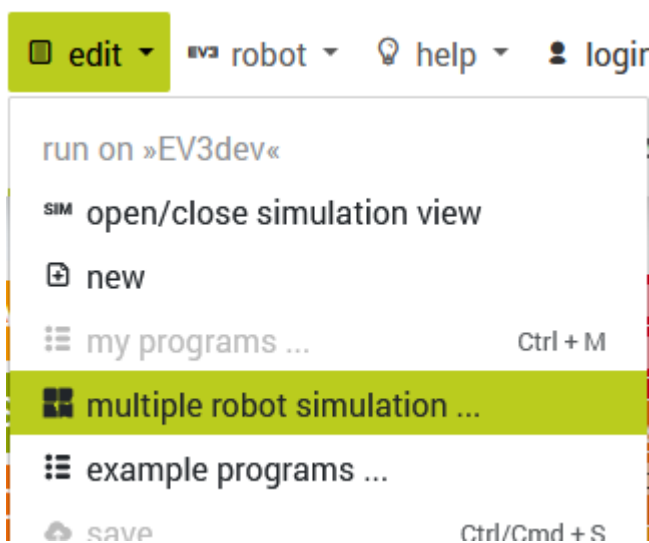
**Lernziel:** You learn to program multiple robots

**Vorkenntnisse:** You should already understand programming a single robot

## 1 Multiple robot demo

We are going to have some fun first. You will need to close this tutorial and then open it again, but do not worry, the steps are very simple, even though the program you see on the left might seem a bit complex.

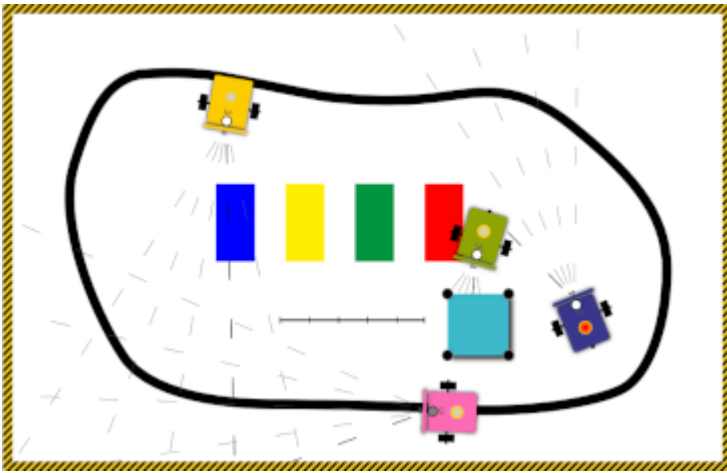
When you close the tutorial, go to main menu, and select *Edit - Multiple robot simulation*:



And then use the [ + ] button to specify the number of robots that will be run. After that, open the simulation view with the button in the right



select the usual background with the line track, place the robots randomly, and start the simulation.



What you see is just a random demo of multiple robots working together in the same environment, all running the same program and showing different behaviors.

**TASK:** identify as many as you can different behaviors/activities the robots are displaying. Try to find different parts of the program that are responsible for each particular behavior. Can you notice some anomalies/abnormalities in the behaviors? Can you figure out why do they occur? The teacher will limit the time for your work for about 5-10 minutes, so be efficient. After the time for this activity is over, share your findings with your friend and the class.

If you are ready and fast:

- modify the program to alter some of the behaviors, or introduce a new one

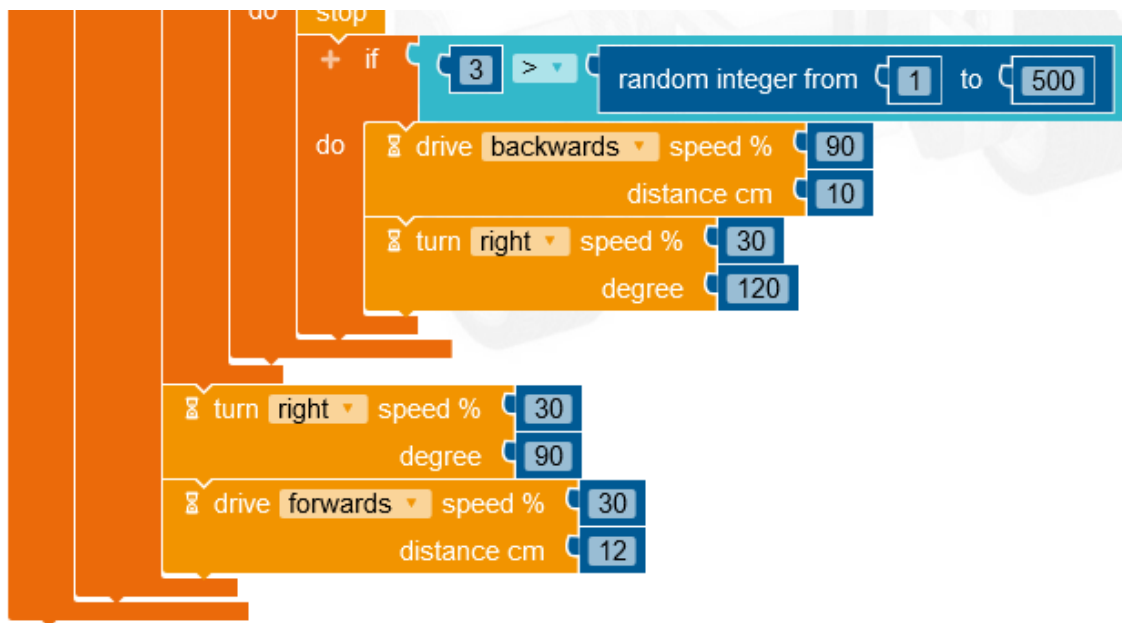
```
+ start  show sensor data
- variable r : Number ← 0

repeat indefinitely
do
  repeat until  get light % colour sensor Port 3 < 20
  do
    drive forwards speed % 30
    + - if  get colour colour sensor Port 3 = 
    do
      turn brick light colour green on
    else if  get colour colour sensor Port 3 = 
    do
      turn brick light colour orange on
    else if  get colour colour sensor Port 3 = 
    do
      turn brick light colour red on
    else
      turn brick light off
  repeat until  get distance cm ultrasonic sensor Port 4 > 12
  do
    stop
    + if  3 > random integer from 1 to 500
    do
      ⌚ drive backwards speed % 90 distance cm 10
      ⌚ turn right speed % 30 degree 120
  set r to random integer from 1 to 3
```

```

set r to random integer from 1 to 3
+ - if 1 = r
do
  drive backwards speed % 60
  distance cm 10
  turn right speed % 30
  degree 20 + random integer from 0 to 190
else if 2 = r
do
  drive backwards speed % 60
  distance cm 10
  turn left speed % 30
  degree 20 + random integer from 0 to 190
else
  drive forwards speed % 30
  distance cm 6
  turn right speed % 30
  + wait until get light % colour sensor Port 3 < 20
  repeat random integer from 100 to 300 times
  do
    + if get light % colour sensor Port 3 < 20
    do
      steer forwards speed % left 10
      speed % right 30
    else
      steer forwards speed % left 30
      speed % right 10
    wait ms 20
  repeat until get distance cm ultrasonic sensor Port 4 > 12
  do
    stop

```



## 2 Duck parade

Thanks for coming back. Let's have some more fun. You will again need to close the tutorial to open the multiple-robot simulation view, but please first read through the whole step. You will import the program '**ducks**'. Configure the environment for 4 robots (or whatever number you want) running the same program. When you open the simulation area, select the large area (where the robots appear small). Have one small obstacle ready and create one colored area - black line that can be dragged around with the mouse. When you start the program, the robots will start rotating until they will see an obstacle, then they will wait until they will see the black line. Have them aligned heading the same direction, and then slowly slide the black line from the back underneath all of them - from the last robot to the first. If they were already aligned, just place the obstacle in front of the first one and then slide the black line underneath them. Turn on the trajectory drawing - but only after you have arranged them and are ready to slide the black line. The first duck will become a leader, and the remaining "**ducks**" should follow up. It may look something like this:





**TASK:** Study the behavior of the robots. Spend some minutes observing, and playing around. Try to describe using plain words what is going on so that your friend and class will understand you. Discuss with the others. Then, study the program. Try to identify the parts of the program that are responsible for the individual behaviors. Discuss with others. What are the variables used for? What do those numbers mean?

After you will have completed the step, open this tutorial for the last step.

```

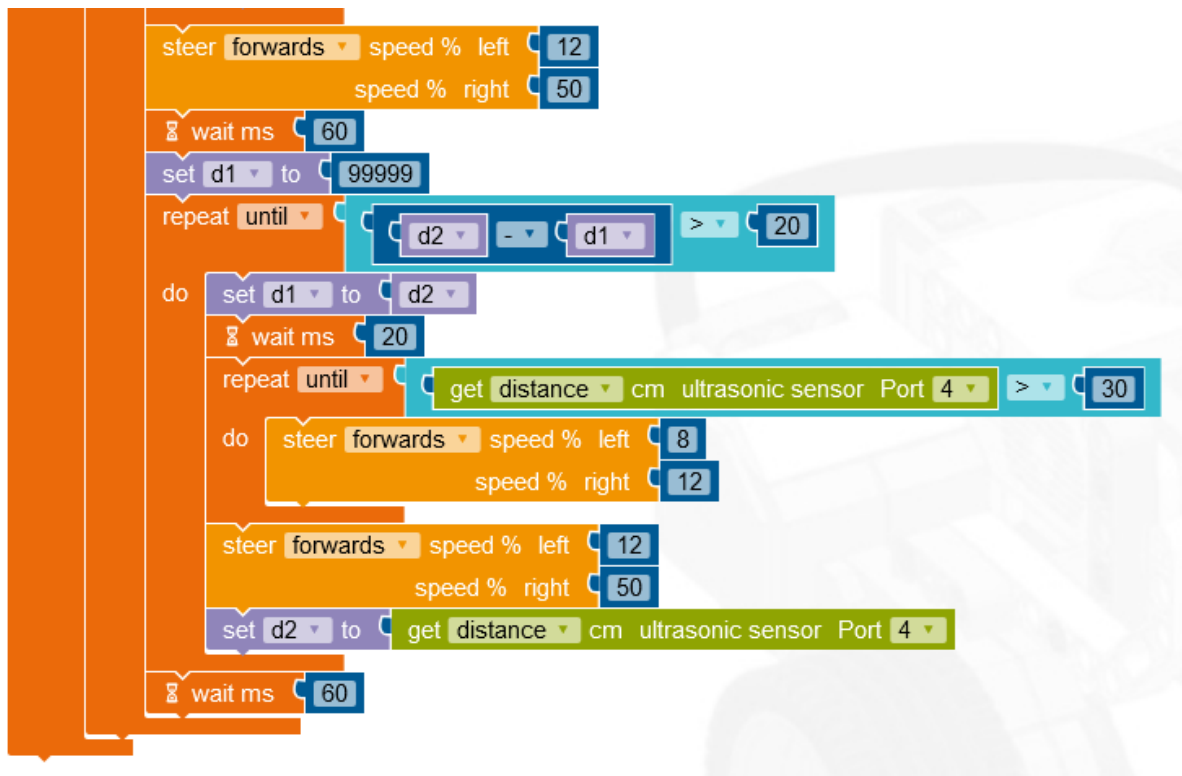
+ start
  show sensor data
  - variable leader : Boolean ← true
  - variable d1 : Number ← 0
  - variable d2 : Number ← 0
  - variable s : Number ← 0
  turn right speed % 10
  + wait until get distance cm ultrasonic sensor Port 4 < 30
  stop
  + wait until get light % colour sensor Port 3 < 50
  + if get distance cm ultrasonic sensor Port 4 < 30
  do set leader to false
  steer forwards speed % left 30
  speed % right 30
  + if leader
  do turn brick light colour red
  on
  repeat indefinitely
  do + if get distance cm ultrasonic sensor Port 4 < 120
  do
    turn right speed % 100
    degree 10
    set d1 to get distance cm ultrasonic sensor Port 4
    turn left speed % 100
    degree 20
    set d2 to get distance cm ultrasonic sensor Port 4
    turn right speed % 100
    degree 10
    + if d1 < d2
  
```



```

do
  ⚙️ steer forwards speed % left 22
      speed % right 35
      distance cm 67
else
  ⚙️ steer forwards speed % left 35
      speed % right 22
      distance cm 67
else
  set s to random integer from 20 to 40
  ⚙️ steer forwards speed % left s
      speed % right 60 - s
      distance cm 7
else
  turn brick light colour green
  on
  repeat indefinitely
  do
    ⚙️ steer forwards speed % left 50
        speed % right 12
    set d1 to 99999
    set d2 to get distance cm ultrasonic sensor Port 4
    repeat until d2 - d1 > 20
    do
      set d1 to d2
      ⚙️ wait ms 20
      repeat until get distance cm ultrasonic sensor Port 4 > 30
      do
        ⚙️ steer forwards speed % left 12
            speed % right 8
      ⚙️ steer forwards speed % left 50
          speed % right 12
      set d2 to get distance cm ultrasonic sensor Port 4

```



### 3 Your turn

Thanks for coming back again! We are now finally going to construct our own programs for a joint behavior for several robots working in the same environment.

Select one of the following tasks and create a program that solves it and then share the result and your experience with others!

1. Line following with avoidance. The robots are following the line in any direction with the same speed. When the robots meet - detect each other with the bumper, they back up a little bit, avoid each other by a round detour to the right, passing the other robot on the left-hand side, and then they both continue following the line.
2. Exploring with avoidance. The robots are exploring the area that is surrounded with a black line: they continue forward until they reach the line, then they back-up, turn, and continue exploration. However, when the robots meet - they detect each other with the bumper, they back up a little bit, avoid each other by a round detour to the right, passing the other robot on the left-hand side, and then they both continue exploring the area. If you are ready and fast, try to avoid an accidental escape from the area during a robot collision / avoidance.
3. Robots are located at the opposite corners of a rectangle with contours marked with black line. They will run the same program, but one robot starts on black line, the other on white floor right in front of the line. The task for the robot is to fill the area with their trajectory line. One of them will start from the middle, the other from its corner. When they are done, they both stop.
4. Organize a tournament for at least three different line-following strategies that will all use the same speed setting, but different algorithms. Load the [lesson11\\_race.png](#) background, place one robot at the mark of each track, use the sound signal to denote when the robot has driven one full lap.

Good luck, have fun, and remember to share!

# Hi again!



You have reached the end of this Open Roberta Lab EduRob tutorial on multiple robots!

We hope you have enjoyed it and we see you again soon!

Lesson 12

**Pattern Recognition**

**Welcome!**



You have reached the final tutorial in the EduRob Open Roberta Lab Tutorial series!

And that means you are ready for a larger project.

We hope you can solve it on your own, but in case you need help, we will be here for you.

## Get ready...

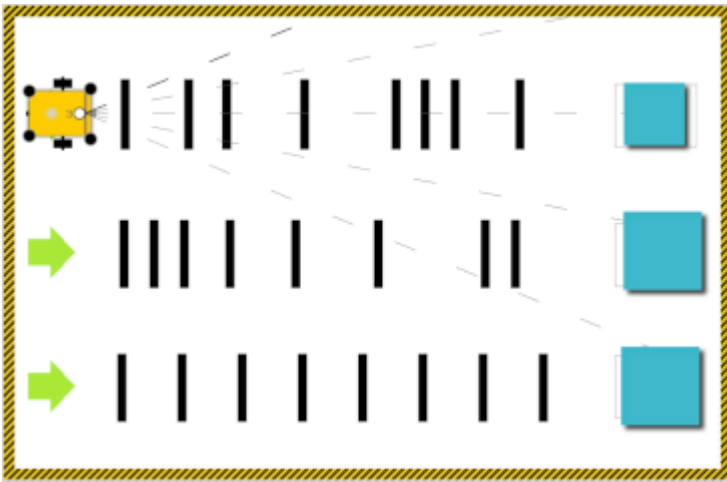
# Let's go!

**Lernziel:** You learn to store and compare patterns

**Vorkenntnisse:** You should understand all features of robot programming

1 Task description

In this project, we are going to use the line-pattern background image ([lesson9\\_line\\_patterns.png](#))



**TASK:** Create a program that will memorize two line patterns. It will then keep reading a new pattern and recognizing it: the program will beep once, if the new pattern is similar to the first memorized pattern and display number 1, and it will beep twice and display number 2, if the new pattern is more alike the second memorized pattern.

The operation of the program should be as follows:

1. Before starting the program, place the robot at the beginning of the first pattern
2. When the program is started, the robot will drive over the line pattern and store the distances between the lines into a list. The pattern is terminated by an obstacle (or wall), robot will stop. Only the distances between the lines should be recorded - the leading distance to the first line, and the distance from the last line to the wall should be ignored.
3. Now the user moves the robot by hand to the start of the second pattern to memorize and presses the right arrow button on the robot. This will put the robot into motion and it will drive over the second pattern, and memorize the distances between the lines into a different list.
4. The rest of the program will be repeated indefinitely:
  1. After pressing the right-arrow button, the robot will read a new pattern (number 3) into yet another list.
  2. It will then compare the first pattern with the new one, as well as the second pattern with the new one, meaning it will calculate the sum of absolute differences of the lengths of the corresponding gaps between the lines for both pattern pairs.
  3. If the error sum for the first pattern was smaller, and this error sum (the total difference) is lower than some threshold (for example 100 degrees in total), then the program will beep once and display number 1.
  4. If the second pattern is more similar and still within the tolerance threshold, it will beep twice and display number 2.
  5. Otherwise, it will not beep at all, and show number zero.

For measuring the length of the gaps, we recommend to use the light sensor, and the motor encoders. Each time robot will reach a new line, it will read the value of the motor encoder:

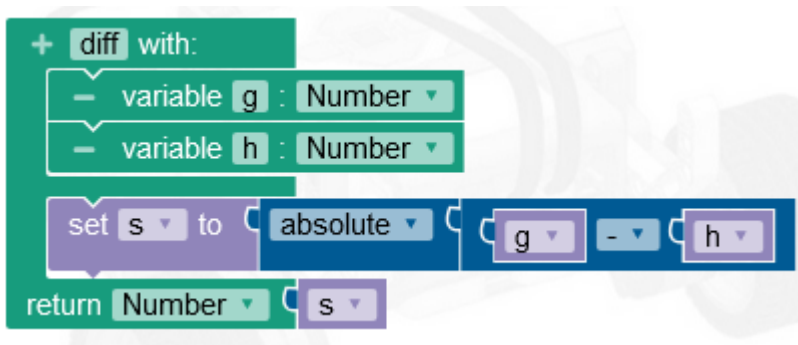
```
get degree ° encoder B
```

and then immediately it will also reset the encoder so that the new gap will be properly measured

reset encoder B ▾

Since the calculation of the differences will be performed more than once, it will be very convenient to use *functions*. Functions are pieces of code that can be called from the main program (or other functions) from several different places. This makes the program shorter and avoids duplicity - which is always bad (*avoiding duplicity is one of the main principle in computer programming*). Functions can take parameters and return values. We can thus create a function that can take a list, fill it with the measured gaps, and return a list full of the values. Or, we can also create a function that can calculate the total difference between gaps stored in two lists, and return the resulting number.

Here is an example of a function that takes two values **g**, **h** and returns the absolute value of their differences.



Notice the function is using the variable **s** that needs to be defined in the main program and the function should not interfere with other uses of the same variable.

We believe you should now be able to solve this project on your own. Good luck!

In case you won't be able to move on after a while, you can move to the next step that gives you some hints about how to structure the whole program. But please try on your own first!

## 2 Overall program structure

We could structure the program in many different ways. One way could be like this:

The main program will define three lists **p1**, **p2**, and **p3**.

We will implement a function **recordPattern**, which will drive over a line pattern and store to the list it will receive on the input the gaps between the lines, all the way until it will reach the obstacle, when it will stop.

We can also implement a function **comparePatterns**, which will receive two lists **u**, **v** as its arguments, it will pass through the two lists (if they do not have the same length, it will stop when the first end is reached). It will calculate the sum of differences of gap lengths recorded in those lists and return this total sum.

Finally, to make the main program more concise, we can implement a function **moreSimilar**, which will take three lists **a**, **b**, **c**, and it will calculate the differences between **a** and **c** and **b** and **c** using the function **comparePatterns**. It will then compare the results also wrt the maximum allowed

difference threshold, and it will return the number **1**, **2**, or **0** depending which of the patterns **a**, **b** or none of them are more similar to the pattern **c**.

The main program will then mostly handle showing information on the display, waiting for pressing the buttons, calling the **recordPattern** for the patterns **p1**, and **p2**, and then in an infinite loop repeating calling the **recordPattern** to **p3**, and then evaluating it by calling the function **moreSimilar**.

We hope if you follow this structure, you will be able to implement the full program on your own. Please try hard, and if you can't proceed, you can move on to the next step for more details on how to implement these functions.

### 3 recordPattern algorithm

Let's start with the function **recordPattern**.

The function will receive one argument **outputList** of a type **List Number**, and it will return the same list at the end.

It should perform the following steps:

1. start moving forward
2. wait until the first line
3. reset the encoder
4. repeat the following steps until the obstacle is reached:
  1. if next line is detected:
    1. append the value of the encoder to the end of the list
    2. reset the encoder
    3. wait a little bit to leave the current line (otherwise it could be detected again as another line)
5. finally stop
6. and return the list with the measured gaps

If you haven't done so, implement the function now.

You can then compare your result with our version of the function in the next step.

### 4 recordPattern function

Here is our version of the **recordPattern** function:

```

+ recordPattern with:
- variable outputList : List Number
  turn brick light colour green
  on
  drive forwards speed % 30
  + wait until get light % colour sensor Port 3 < 50
  reset encoder B
  play sixteenth note a'
  wait ms 200
  repeat until get distance cm ultrasonic sensor Port 4 < 15
  do
  + if get light % colour sensor Port 3 < 50
  do
  in list outputList insert at last as get degree ° encoder B
  reset encoder B
  turn brick light colour red
  on
  play sixteenth note a'
  wait ms 200
  turn brick light colour green
  on
  stop
  turn brick light colour orange
  on
  return List Number outputList

```

To help the user, we also use the brick light and sound to indicate correctly detecting the lines.

In the next step, we will discuss the details of the `comparePatterns` function.

### 5 comparePatterns algorithm

The `comparePatterns` function should receive two lists `u` and `v` (possibly of different length) and calculate the sum of absolute differences of the valid list elements.

Here are the steps of the function:

1. Initialize the resulting sum to `0` - we will use a global variable `s`
2. Initialize the index (position in the lists) to `0` - we will use a global variable `i`
3. Compare the lengths of the lists, and take the shorter one into a global variable `m`
4. Repeat `m`-times:
  1. Take the `i`-th element of list `u`
  2. Take the `i`-th element of list `v`

3. Subtract them
  4. Calculate absolute value of the result
  5. Add that value to the sum **s**
  6. Increase the index **i** by **1**
5. Return the sum **s**

Now it should be straight-forward to implement this function. Next step of the tutorial shows our implementation.

## 6 comparePatterns function

This is our implementation of the function **comparePatterns**:

```

+ comparePatterns with:
- variable u : List Number
- variable v : List Number

set s to 0
set i to 0
set m to length of u
+ if length of v < m
do set m to length of v
repeat m times
do
change s by absolute (in list u get # i) - (in list v get # i)
change i by 1
return Number s
  
```

Notice the variable **i** is used for the position in the list. This is usually called an *index*, and that is why the variable named **i** is used for this purpose most of the times.

In the next step, we will discuss an auxiliary (helping) function **moreSimilar** that helps us to have the main program a bit shorter. Even though it will only be called from one place, it is usually convenient to break program into several independent pieces (functions). It helps readability and maintainability of the program.

## 7 moreSimilar algorithm

The **moreSimilar** function will take three lists **a**, **b**, and **c** as its arguments. It will perform the following steps:

1. call the **comparePatterns** with lists **a** and **c** and store the result to global variable **c1**
2. call the **comparePatterns** with lists **b** and **c** and store the result to global variable **c2**
3. if **c1** is a smaller number than **c2** (patterns **a** and **c** are more similar then **b** and **c**):
  1. check if **c1** does not exceed the maximum allowed difference threshold
    1. if it does not, set value **1** to variable **i** (the result will be number **1**)
    2. if it does, set value **0** to variable **i** (none of the two patterns is similar, the result will be **0**)
  2. otherwise (if **c1** is not smaller than **c2**), check if **c2** does not exceed the maximum allowed difference threshold



1. if it does not, set value **2** to variable **i** (the result will be number **2**)
  2. if it does, set value **0** to variable **i** (none of the two patterns is similar, the result will be **0**)
4. return the resulting number **i**

The next step will disclose our implementation of this function.

### 8 moreSimilar function

Here is our implementation of the **moreSimilar** function:

```

+ moreSimilar with:
- variable a : List Number
- variable b : List Number
- variable c : List Number

set c1 to comparePatterns
  u a
  v c
set c2 to comparePatterns
  u b
  v c

if c1 < c2
do
+ if c1 < 100
do
set i to 1
else
set i to 0
else
+ if c2 < 100
do
set i to 2
else
set i to 0

return Number i

```

Now we are almost ready to write the main function. We will have to summarize which global variables did we require to be already defined in all the functions we wrote and define them in the very beginning of the program. We will also need to carefully take care of displaying information for the user and waiting for pressing the buttons.

## 9 main program algorithm

Here is the idea for the structure of the main program:

Variables that we have used in other functions:

**i, s, c1, c2, m** - all are numbers

Variables that we will need now:

**p1, p2, p3** - all are number lists

The steps of the function:

1. Call the **recordPattern** function and store the result into list **p1**
2. Wait until the user [moves the robot and] presses the right arrow button
3. Call the **recordPattern** function and store the result into list **p2**
4. The rest of the program will be repeated indefinitely:
  1. Wait until the user [moves the robot and] presses the right arrow button
  2. Call the **recordPattern** function and store the result into list **p3**
  3. Compare the patterns and calculate the result **0, 1** or **2** into variable **i** by calling the **moreSimilar** function with the lists **p1, p2, and p3**
  4. Show the result on the display and produce **i** beeps

The final step of the tutorial shows the implementation of the main program.

## 10 main program implementation

Here is the main program:

*(for better use of space we split it into two parts, but it is only one program)*

```

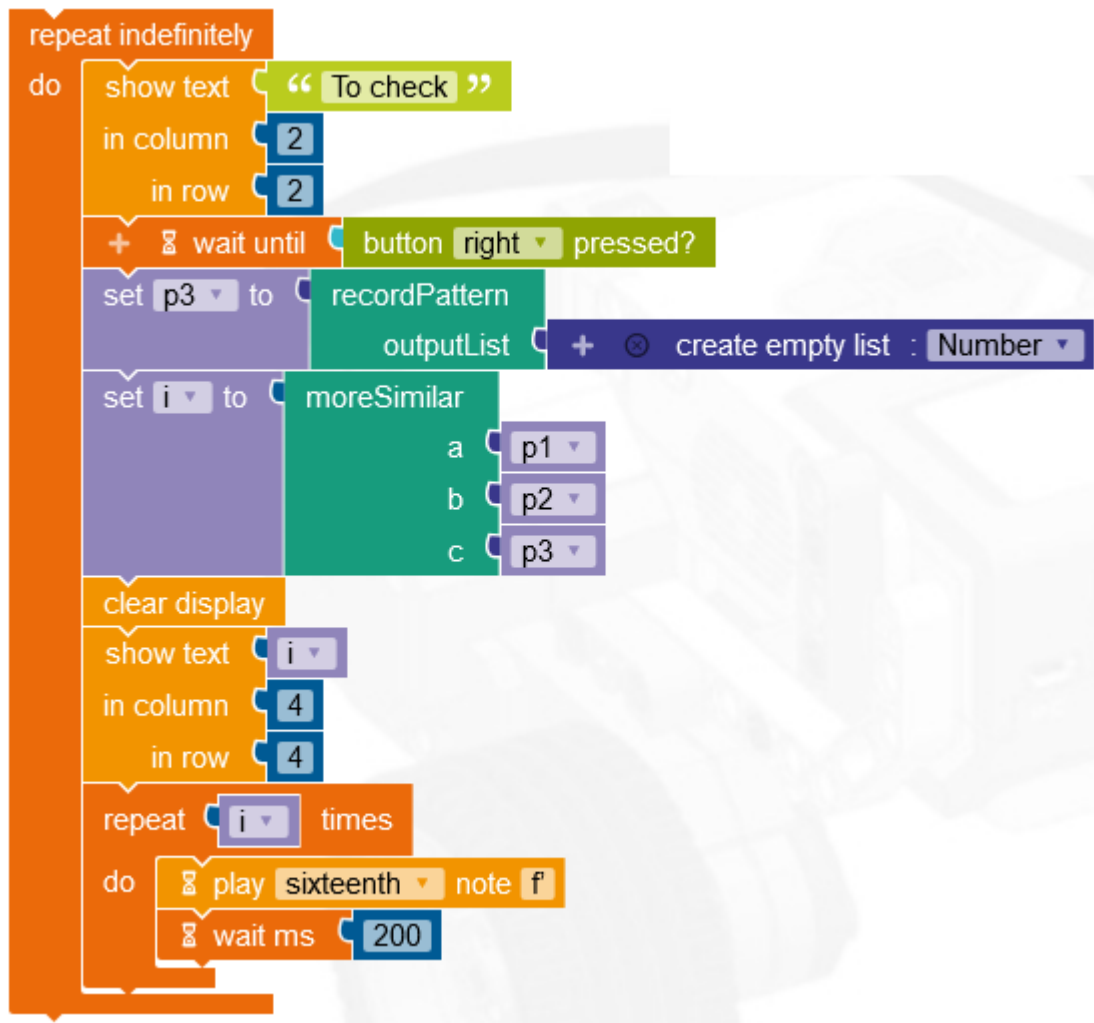
+ start
  show sensor data
  - variable p1 : List Number ← + create empty list : Number
  - variable p2 : List Number ← + create empty list : Number
  - variable p3 : List Number ← + create empty list : Number
  - variable i : Number ← 0
  - variable s : Number ← 0
  - variable c1 : Number ← 0
  - variable c2 : Number ← 0
  - variable m : Number ← 0

clear display
show text " First to learn "
in column 2
in row 2
set p1 to recordPattern
outputList p1

clear display
show text " Second to learn "
in column 2
in row 2
+ wait until button right pressed?
set p2 to recordPattern
outputList p2

clear display

```



Now it is definitely the time to test the program and share your experience with your friend and the class.

If you are ready and fast:

- Modify the program so that it can learn three instead of two patterns.
- Modify the program so that even if it will only see a part of the pattern (not necessarily from the first line), it will still be able to correctly recognize it.

 **Congratulations!**

You have completed all the tutorials in the EduRob Open Roberta Lab Tutorial series!

**Remarkable!**

**We hope you enjoyed Roberta! You are now ready to design your own challenges to solve!**

Have a lot of fun with the robots! 