Introductory scenarios

Programming classes

Lesson scenarios introducing the basics of programming

Author: Photon Education

Programming – the basic level

Photon dances the waltz

programming • technology

Duration time:	Robots:	Programming interface:	Accessories:
90 min.	×1	Photon Blocks	-

Goals (for the children):

- To understand the basic concepts of programming and coding.
- To create a program for a Photon Robot using a visual programming interface (Photon Blocks).
- To be able to identify activities being performed by the robot
- To gain practical knowledge of using programming loops.

Required items:

- This scenario does not require using additional resources / aids.

Type of exercise:

- individual
- group

Preparations:

You can use one or several robots to conduct a lesson based on this scenario.

If you only have one robot and the Photon Magic Dongle, we recommend using an interactive whiteboard to benefit the entire group. In order to do this, install the Photon Magic Bridge app on your computer.

Please remember to charge your robots and tablets before the class.

Lesson scenario:

1. **Introduction**

- Explain to the children that they will have to prepare the simplest dance choreography for the waltz. The robot's 'dance' will include the backward, forward, right turn moves. If time permits, you can have additional elements in this choreography.
- Explain that work on the choreography should be divided into several steps.

2. **Learning activity**

Step 1. Simulation

- Select two students from the group. The first student's task is to simulate the robot's moves, and the other student should write down these movements on the board (in a simplified way using just arrows).
- Ask the first student to present a choreography so that they end up in the same place and facing the same direction they started:

  - Backward, forward, right turn.
  - Backward, forward, right turn.
  - Backward, forward, right turn.
  - *Backward, forward, right turn.*

- At the same time, the other student should write down these moves on the board. Pay particular attention to the form of writing "right turn". A right arrow (→) could suggest taking a step to the right instead of a turn.

  - The expected moves combination: ↓↑⤵, ↓↑⤵, ↓↑⤵, ↓↑⤵.

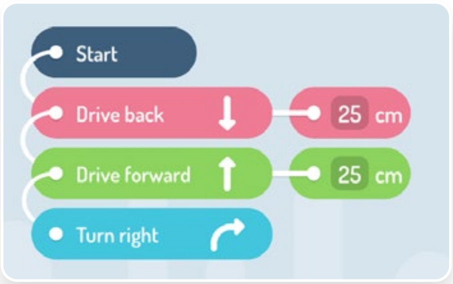
- Fantastic! You have just created instructions that the Photon Robot needs to perform your choreography.

Step 2. Programming

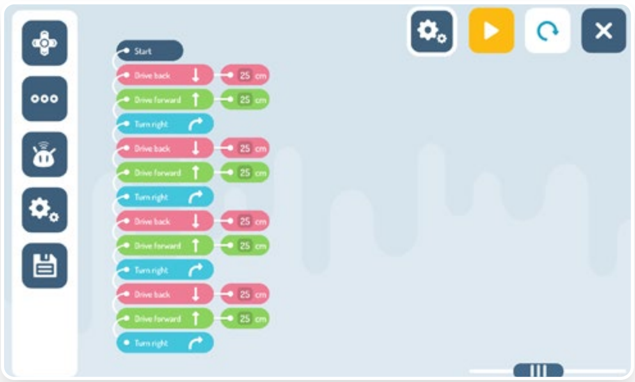
Depending on the number of robots and tablets you have available, divide your students into groups or work on this activity as a group exercise. When working with one robot and the whole class, you can use an interactive whiteboard to display the programming interface on it. You can also use your TV to display the image from the tablet.

- Launch the Photon Blocks programming interface (in the Photon EDU app for tablets or the Photon Magic Bridge app for computers).
- Start by programming the basic steps. In the toolbox on the left, select the Movement category and drag out the following blocks:

  - *Drive back* – set it to 25 cm.
  - Drive forward – set it to 25 cm.
  - *Turn right* (optionally, you can use the *Turn* block and define the direction and angles).



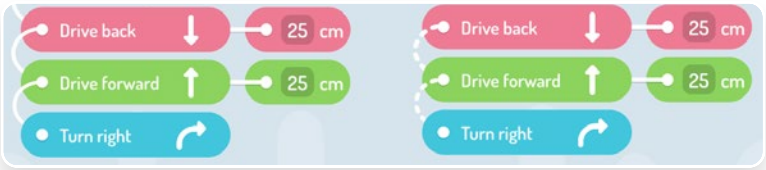
- Run the program to show the results to the students. Once the robot turns, it stops, so you can manually return it to its original position.
- Continue creating the program by duplicating the blocks and arranging them in the correct order to complete the choreography (the robot should stop in its starting position).



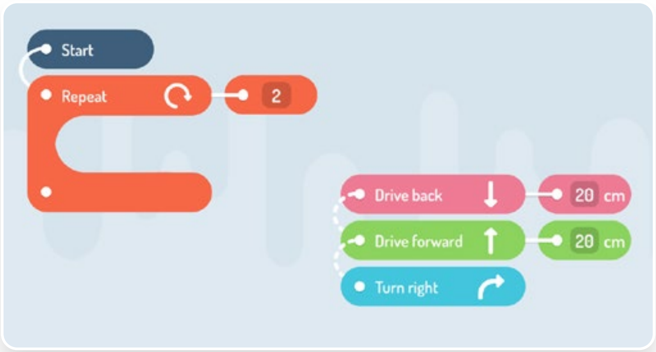
- Run the program to show the results to the students.

Step 3. Loops

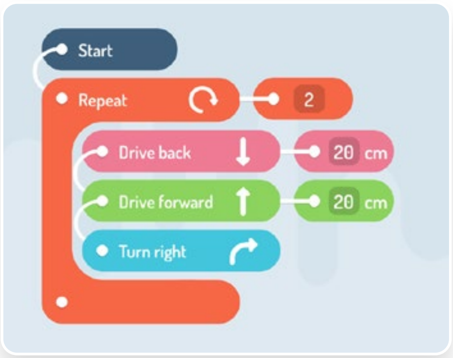
- To help the children understand the possibilities of programming and common mistakes in programs, you can introduce the concept of loops.
- However, before using this feature, ask the students to make a specific change in their choreography. Instead of 25 cm forward and backward move, make the robot move by 20 cm.



- Ask for a volunteer to make this change in all eight movement blocks.
- It might take a while. Once done, emphasize that this program is really short. Let them imagine making this change to choreography compromised of many more instructions or steps. How much longer would it take? Too long.
- Explain that one of the primary uses of programming is the automation of repetitive tasks. Review the created program together and indicate a repetitive part (*Drive back, Drive forward, Turn right*).
- Move this part of the program, consisting of the three repeating blocks, to a side programming window, and discard the remaining blocks. Go to the third category in the toolbox on the left. Drag and drop the Repeat block and place it in the main program window (the default setting is two repeats – leave it as is).



- Then drop the repetitive part inside the *Repeat* block. Run the program to show the result – the robot will stop in the middle of the choreography.
- Change the number of repetitions to 4 by clicking on the number on the *Repeat* block. Run the program again to show that the robot has repeated this sequence four times and performed the entire choreography.
- Ask students: *What has to be done to make the robot dance a waltz until we tell it to stop?* The application allows you to set the *Repeat* block an infinite number of times. This way, the robot stops when you press the *Stop* button only.



- Finally, remind students how time-consuming it was to change the length of travel in all choreography sequences. Ask a volunteer to make this change again, i.e., modify the backward-and-forward section of the movement block to 25 cm again. As they will see, this time, it only takes a moment! You only had to make two changes.

Step 4. Experiments (optional)

If you have some spare time left, ask your students to experiment more by changing the choreography. Here are some inspiring challenges:

- **Challenge #1:** Create a more advanced choreography to make the robot move around the square (repeat: backward, forward, turn, forward).
- **Challenge #2:** Make the choreography more complex so that the robot rotates by an angle different than the standard 90˚ (replace the *Turn right* block with *Turn...*).
- **Challenge #3:** Try to program a choreography for two robots (if you have more than one) to make them dance as a pair or in the same way.

Author: Photon Education

Programming – the basic level

Controlling the robot by willpower

programming • technology

Duration time:	**Robots:**	**Programming interface:**	**Accessories:**
45 min.	×1	Photon Blocks	-

Goals (for the children):

- To understand the basic concepts of programming and coding.
- To create a program for a Photon Robot using a visual programming interface (Photon Blocks).
- To be able to identify activities being performed by the robot.
- To learn how to use conditional statements and loops.
- To learn how to use the robot's built-in distance sensor

Required items:

- This scenario does not require using additional resources / aids.

Type of exercise:

- group
- individual

Preparations:

You can use one or several robots to conduct a lesson based on this scenario.

If you only have one robot and the Photon Magic Dongle, we recommend using an interactive whiteboard to benefit the entire group. In order to do this, install the Photon Magic Bridge app on your computer.

Please remember to charge your robots and tablets before the class.

Lesson scenario:

1. Introduction

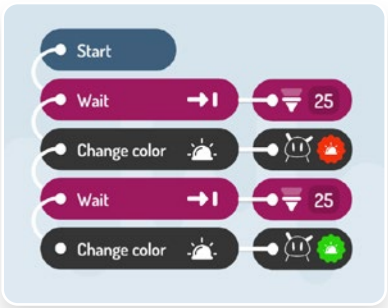
- Tell the children that they will create a program that will allow them to control the robot using willpower – pull it back and push it away like a wizard or a Jedi from Star Wars movies.
- Explain that work on the choreography should be divided into several steps.

2. Learning activity

Depending on the number of robots and tablets you have available, divide your students into groups or work on this activity as a group exercise. When working with only one robot as the whole class, you might want to use an interactive whiteboard to display the programming window. You can also use your TV to display the image from the tablet.

Step 1. Determining the distance – color signaling

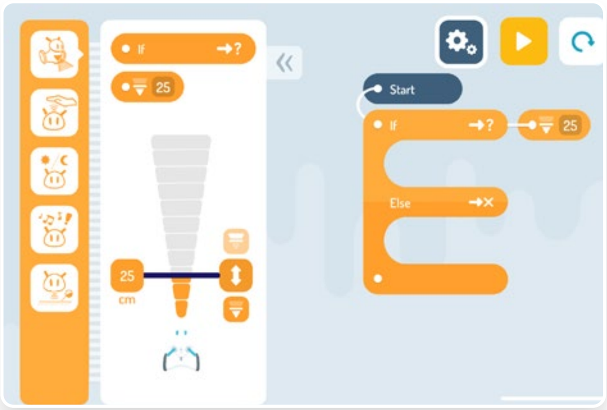
- Present to your students two situations that will follow one another:
    - When the robot detects an obstacle (a hand) within 25 cm, it turns red.
    - When the robot detects an obstacle further than 25 cm away, it turns green.
- Launch the Photon Blocks interface (in the Photon EDU app for tablets or in the Photon Magic Bridge app for desktop computers).
- Create a program based on four instructions: the robot waits for an obstacle closer than 25 cm, then it turns red, then it waits for an obstacle located more than 25 cm away, and it turns green.
- The *Wait* block is available in the third from the top category in the toolbox, while the *Change Color* block is in the second category.



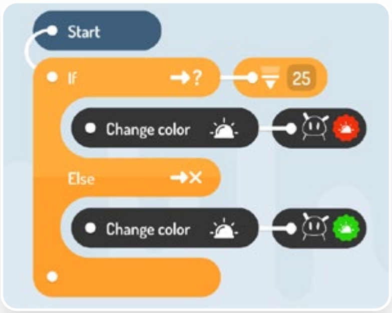
- Once done, run the program to show the results.

Step 2. Introduction of conditional statements

- In the next step of creating the program, explain to the students that the program you have just created dictates in advance what should happen. The robot makes no decisions – it waits and follows instructions. To create a smarter program that allows the robot to respond to any event, add the following conditional statement to the program: "*If / Else*".
- Move the currently arranged blocks to the side of the workspace. From the third category of the toolbox on the left take out the double "*If / Else*" block and arrange it using the program. Set the parameter to *Distance less* than 25 cm.



- In the next step, complete the remaining program instructions according to the logic: if the robot detects a hand at a distance of less than 25 cm, it will turn red; if not (i.e.,if it doesn't detect a hand at a distance of less than 25 cm) – it will turn green.



- Explain to the students the logic of negation:

  NO < 25 cm means **> 25 cm**.

- Once done, run the program to show the results.

⚠ **Important!**

If nothing is placed next to the robot, the robot most likely turns green and does not respond to subsequent forwarding or retracting your hand. This is the correct behavior of the program.

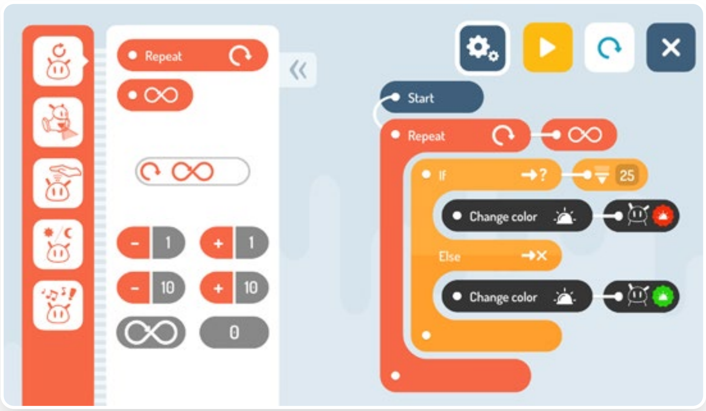
- Inspect this behavior together: when you click the Start button, the program goes to the conditional block If. Then it checks if the hand is less than 25 cm away and makes a decision:

  - If this is the case – it moves to the first section and turns red.
  - If this is not the case – it moves to the second section and turns green.

  Then the robot moves on to the next part of the program, i.e., blocks directly below the *If / Else* block. There is nothing else in our program, so the program will terminate after this step.
- Run the program several times, each time placing your hand or any object in front of the robot at a different distance. Observe the results.

Step 3. Introduction of the Infinity loop

- The next step is to expand our program so that the robot responds to the hand when we run the program in continuous mode.
- To achieve this, simply add the *Repeat* block to the program (located in the third category in the toolbox on the left) and set it to *Infinite*.
- The completed *If / Else* block should be placed inside the *Repeat* block.



- Run the program and test with students how the robot responds to the changing distance of the hand being placed in front of it. Follow the program's execution step by step.

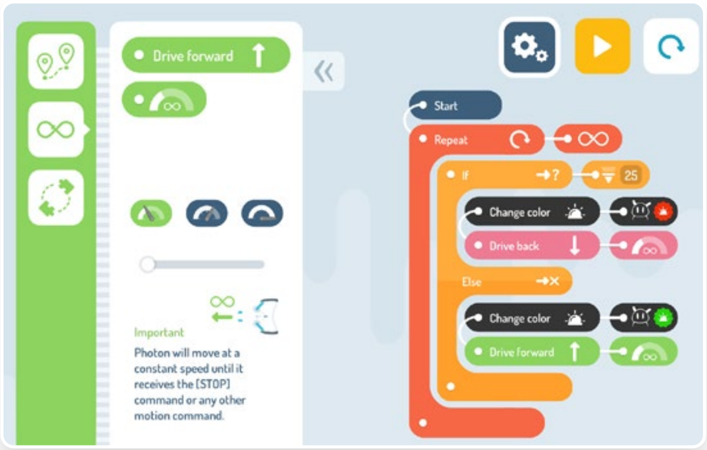
Step 4. True willpower!

Now you will work on the final part of the program. The only thing that you need to program now is a way of controlling the forward and backward moves of the robot in response to the hand movement.

- Make sure that when you put your hand close to the robot (< 25 cm), the robot is pushed (drives backward), and when you move your hand away from the robot (> 25 cm) the robot is pulled (drives forward).
- Add the following blocks to the program:
    - To the *If* section, add a block *Drive back* and set its second parameter (Infinity) at low speed.
    - To the *Else* section, add a *Drive forward* block with the second parameter set to (*Infinity*) at low speed.

⚠ **Important!**

It is also possible to set the second parameter for the *Drive forward* or *Drive back* block as a distance. Its length must be expressed in centimeters. With this option the robot will move intermittently as per the set travel distance. In the *Infinity* option, the movement will be smooth.

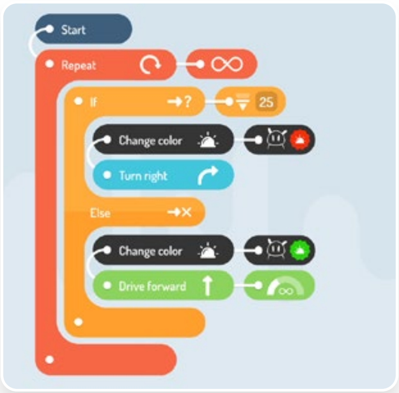

Drive forward
Important
Photon will move at a constant speed until it receives the [STOP] command or any other motion command.
Start
Repeat
If
25
Change color
Drive back
Else
Change color
Drive forward

- To make sure you do not damage the robot, put it on the floor.
- Run the program and test your magical ability to push and pull the robot with your willpower.

Step 5. Experiments (optional)

If you have some spare time left, ask your students to experiment more by changing the way the robot is programmed. Here are some inspiring challenges:

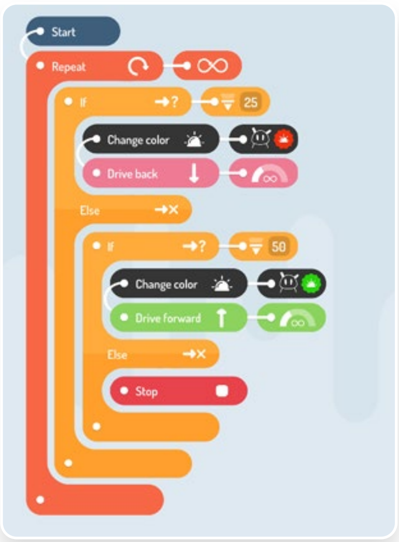
- **Challenge #1:** Change one instruction to make the Photon act like a simple autonomous robot vacuum cleaner that goes straight until it detects an obstacle. Solution: Replace the *Drive back* block with, for example, *Turn right*.



- **Challenge #2:** Expand the program created during the class to make the robot:

  - Reverse (be pushed away) when the hand is closer than 25 cm.
  - Drive forward (be pulled) when the hand is closer than 50 cm.
  - Stop in place when no hand is detected (> 50 cm).

  Solution: In the *Else* section of the *If / Else* block, place another *If / Else* block to use the so called “nesting”. It may seem difficult to understand at first, but once the robot performs it several times, it becomes simple, logical, and understandable for everyone



- **Challenge #3:** Rearrange the program you created up to the end of Step 3 (without the forward/reverse instructions) and change the decisive condition in the *If / Else* block. Test with the students the robot’s response to touch or sound detection.

Author: Photon Education

Programming – the basic level

Robot the Guardian

programming • technology

Duration time:	Robots:	Programming interface:	Accessories:
90 min.	×1	Scratch 3.0	-

Goals (for the children):

- To understand the basic concepts of programming and coding.
- To create a program for a Photon Robot in a programming interface (Scratch).
- To be able to identify activities being performed by the robot.
- To gain practical knowledge of using programming loops.

Required items:

- This scenario does not require using additional resources / aids.

Type of exercise:

- individual
- group

Preparations:

You can use one or several robots to conduct a lesson based on this scenario.

If you only have one robot and the Photon Magic Dongle, we recommend using an interactive whiteboard to benefit the entire group. In order to do this, install the Photon Magic Bridge app on your computer.

Please remember to charge your robots and tablets before the class.

Lesson scenario:

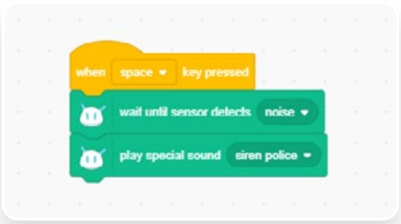
1. Introduction

- Explain that the children need to create an advanced program focused on the Photon Robot listening to loud noises. In response to hearing a loud sound, it has to scan the closest environment and react appropriately to anything it notices.
- Explain that work on this program is divided into several steps.

Step 1. Responding to a sound

The program is a combination of several related activities. The first one is detecting a loud sound.

- Launch the Scratch interface (in the Photon Magic Bridge app on your computer or in the Photon EDU app on your mobile device) and make sure it is set to your preferred language. If not, click the globe icon in the upper left corner and select your language.
- In order to define how to run the program once it is ready, find the *Events* category in the toolbox on the left and select the block *When space key pressed* – drag it to the workspace.
- Then, find the *Photon* section in the toolbox, select the *Wait* for block and drop it below the first block. Select Noise from the drop-down list.
- Go back to the *Photon* section in the toolbox, insert *Special Sound* and set it to *Police Siren*.



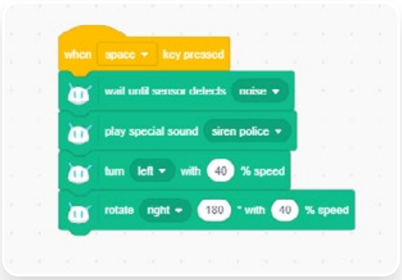
```
when space key pressed
wait until sensor detects noise
play special sound siren police
```

- To demonstrate the result, run the program using the space bar. When you clap your hands, the robot will make a police siren sound!

Step 2. Looking around

Great, so the first part of your program is ready. We established that when the Photon Robot detects a sound, it should scan the area around it.

- Before you move on to scanning, create a movement pattern. Make sure that at first, the robot looks to the left, and then, looking carefully, turns to the right.
- This is how you could do it:
    - Add a block *Rotate left* at 40% speed to the program.
    - Then – *Turn right* 180 degrees.



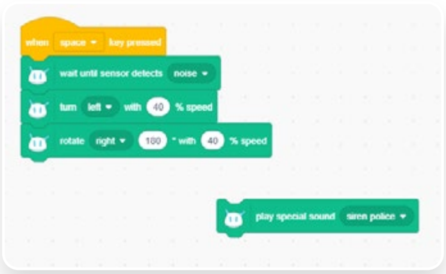
```
when [space ▾] key pressed
wait until sensor detects [noise ▾]
play special sound [siren police ▾]
turn [left ▾] with (40) % speed
rotate [right ▾] (180) ° with (40) % speed
```

- Run the program (by pressing the space bar) and see what happens when the robot detects a loud sound.

Step 2. Dividing the surrounding space into zones (loops)

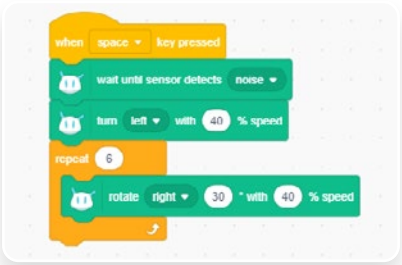
At this step, the robot looks around but does not scan the space it is looking at. In order to increase the complexity of your program, you need to add loops and conditionals.

- Before proceeding further, move the block responsible for the special sound to the side. This way, the block will not disturb you or distract the children.



```
when [space ▾] key pressed
wait until sensor detects [noise ▾]
turn [left ▾] with (40) % speed
rotate [right ▾] (180) ° with (40) % speed

play special sound [siren police ▾]
```

- Start by dividing the area the robot needs to scan. After turning left, the robot should start scanning a space divided into six zones.
- To achieve this, you must ensure the robot makes six turns by 30 degrees (180 divided by 6) instead of one 180-degree turn. Use the *Repeat block* from the *Control* section of the toolbox. Add it to your program, and place the *Turn right* instruction inside. Change the rotation block parameters accordingly to 30 degrees and the *Repeat* to 6 times.



```
when space ▾ key pressed
wait until sensor detects noise ▾
turn left ▾ with 40 % speed
repeat 6
    rotate right ▾ 30 ° with 40 % speed
```

- Run the program to see the results. You should have noticed that the robot made six tiny stops.

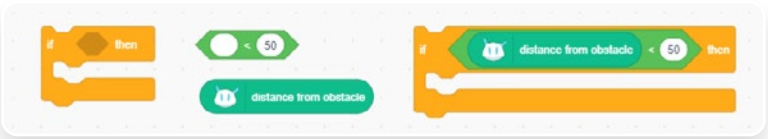
Step 3. Scanning (conditional statements)

In the next step, you will add the scanning feature to your program. You want the robot to check if there is any obstacle within the distance of 50 cm after each of the six rotations.

- To do that, you have to use a combination of three blocks. From the toolbox on the left please drag out:

  - a square block *If* (from the *Control* section),
  - a triangular block *( ) < 50* (from the *Expressions* section),
  - a circular block *Distance from* obstacle (from the *Photon* section).

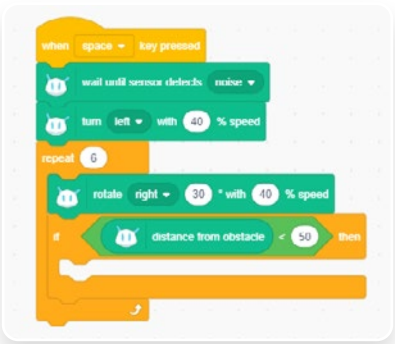
- Connect them according to their matching shapes.



```
if ⬡ then

( ) < 50

distance from obstacle

if distance from obstacle < 50 then
```

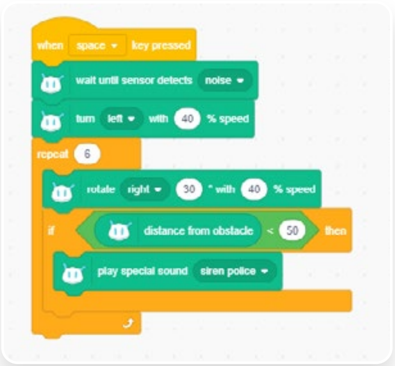
The block works so that The block works so that during the execution of the program, at the specific moment where the block is placed, the robot measures its distance from an obstacle (object). If it is less than 50 cm, the instructions placed inside the orange block become executed. If the distance is greater than 50 cm, the instructions inside the orange block are skipped, and the program continues from the next block below – the *If* block.

- This combination of blocks should be placed below the instruction for turning by 30 degrees to the right.



```
when space ▾ key pressed
wait until sensor detects noise ▾
turn left ▾ with 40 % speed
repeat 6
    rotate right ▾ 30 ° with 40 % speed
    if distance from obstacle < 50 then
```

- At this point, the robot is able to detect any object within a 50 cm range present at the time of scanning.
- To test this, place the *Special Sound* block (previously set aside) inside the *If* block.

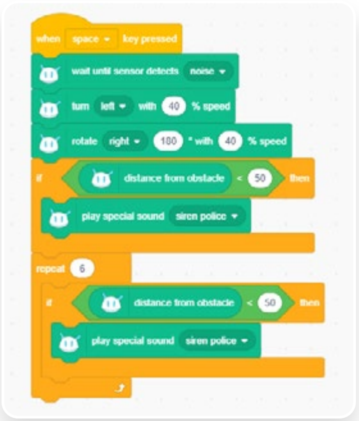


```
when space ▾ key pressed
wait until sensor detects noise ▾
turn left ▾ with 40 % speed
repeat 6
    rotate right ▾ 30 ° with 40 % speed
    if distance from obstacle < 50 then
        play special sound siren police ▾
```

- Run the program to check the results.
- As you can see, the program meets its objectives, although it is not perfect. How could you improve it?

  - As you have probably noticed, the robot always stops at a different position than the original start position. So, you can add one more instruction *Turn left* at the end of the program to ensure the Photon Robot returns to the base position after scanning.
  - The robot always scans the surroundings after a turn to the right, but does it perform a scan after the first turn to the left? No! So, for example, if you have a thief standing to the robot's left, the Photon Robot will miss this threat. To fix this issue, you can add the entire *If* block (with all the blocks inside) just after the first turn to the left (before the *Repeat* block).

- The complete and revised program looks as follows:



```
when space key pressed
wait until sensor detects noise
turn left with 40 % speed
rotate right 180 ° with 40 % speed
if distance from obstacle < 50 then
    play special sound siren police
repeat 6
    if distance from obstacle < 50 then
        play special sound siren police
```

Note

It might be a good idea to explain to the children why you placed it before the Repeat block and not inside in the first position. If you placed it inside, the robot would scan the space before and after the right turn each time. The result would be as follows: the Photon Robot scans – turns – scans, scans – turns – scans, scans, etc. This is not a desired result. Would the children want to make their bed twice or brush their teeth again if they just did it?

Step 4. Experiments (optional)

If you have some spare time left, ask your students to experiment more by changing the way the robot is programmed. Here are some inspiring challenges:

- **Challenge #1:** Re-program your robot to scan the whole space around (360 degrees). You can keep the six zones you have and change the rotation angles only, or you can set up as many new scan zones as you like and help your students determine the rotation angles (360 degrees divided by the number of new zones).
- **Challenge #2:** Using the knowledge from this class, come up with a more creative work pattern for this guarding robot. For example: make the robot stand still and guard a specific object, like a pencil case. If someone takes it away or steals the robot, the Photon Robot should sound an alarm.
- **Challenge #3:** Using the knowledge from this class, come up with a more creative work pattern for this guarding robot. For example: the robot stands by a closed door or in a dark room. If someone opens a door or turns on the light, the Photon Robot should sound an alarm.

Author: Mateusz Blaszkiewicz

Programming – advanced level

To the rescue!

social development • computer science • programming

Duration time:	**Robots:**	**Programming interface:**	**Accessories:**
45 min.	×1	Scratch 3.0	Magic Dongle

Reference to the core curriculum:

- **IT Grade 4–6:** I.2.3, II.1, IV.4,
- **Computer Science 6–8:** II.1, II.2.

Goals (for the children):

- To learn about the different uses of variables in programming
- To learn how to create a program and – as more tasks are received – to develop it accordingly

Required items:

- an obstacle for the robot to locate, e.g., a book, a teddy bear
- desktop computer

Type of exercise:

- individual
- in pairs

Attachments:

https://photon.education/eko/2

Lesson scenario:

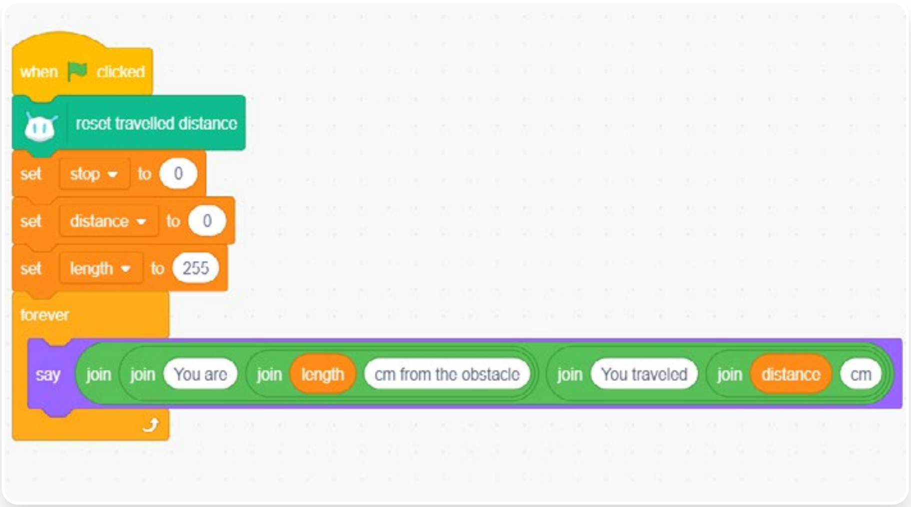
Ask the children what variables are, how they can be used in different programs and what they could "remember" in the case of the Photon Robot. Tell them that their task will be to create a Rescue Robot that:

- **Should start from the base and** reach the destination by emitting light and sound signals
- **Should stop** in front of the target and provide assistance
- **Should inform** the Emergency Call Center about the length to the target and the distance traveled
- **It should return to the base.**

Task 1

Interface – creating an Emergency Call Center in Scratch

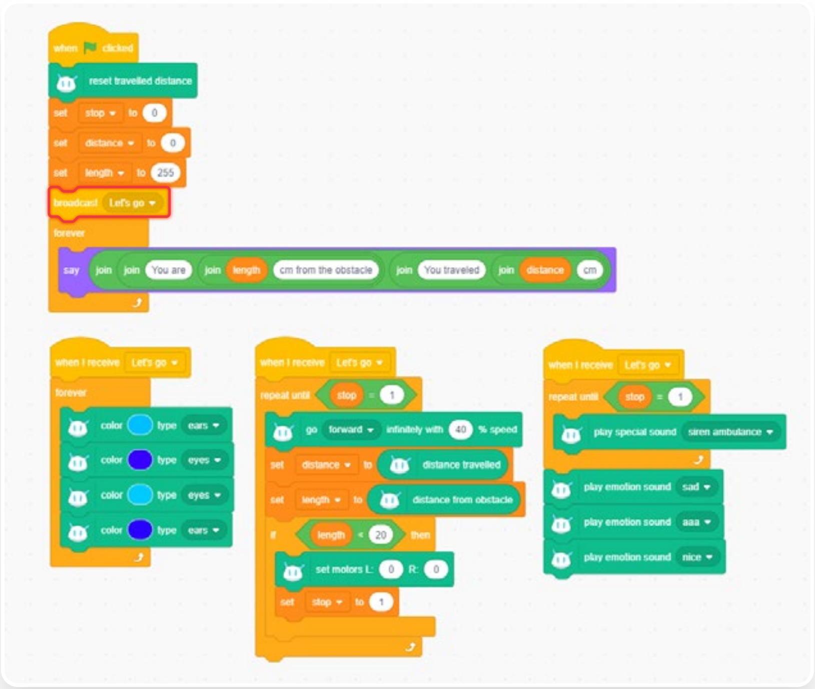
The children are to create a scene, choose an appropriate background and Sprite of their choice. Ask them to create three variables named: distance, length, and stop. Ask them if they understand the difference between the words *distance* and *length*. The “stop” variable will determine whether the robot is standing still or if it has reached its destination. Using the appropriate loop, the Sprite should inform how far the obstacle is and the distance traveled. It is important to reset the distance before each start of the robot.



```
when [flag] clicked
reset travelled distance
set stop to 0
set distance to 0
set length to 255
forever
    say join (join You are (join length cm from the obstacle)) (join You traveled (join distance cm))
```

Light and sound signals – we're on our way

The children are to create a script responsible for giving visible light signals and launching the robot towards the target. Ask the children to use appropriate messages to create each group of blocks separately. The robot's ears and eyes are supposed to blink alternately, and the alarm is supposed to sound only when the robot is moving. When the robot arrives at its destination, turn the alarm off.



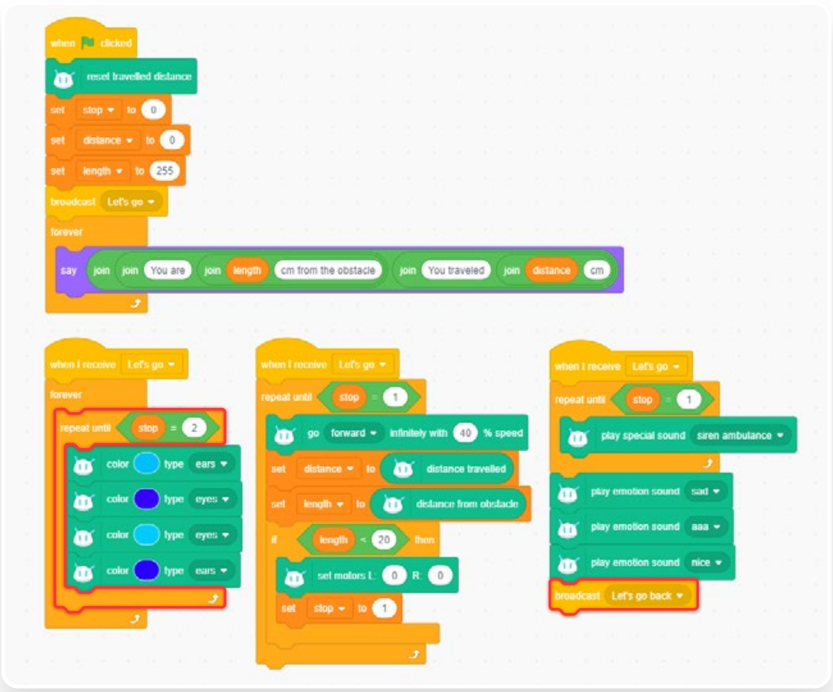
```
when 🏁 clicked
reset travelled distance
set stop to 0
set distance to 0
set length to 255
broadcast Let's go
forever
    say join join You are join length cm from the obstacle join You traveled join distance cm

when I receive Let's go
forever
    color ● type ears
    color ● type eyes
    color ● type eyes
    color ● type ears

when I receive Let's go
repeat until stop = 1
    go forward infinitely with 40 % speed
    set distance to distance travelled
    set length to distance from obstacle
    if length < 20 then
        set motors L: 0 R: 0
        set stop to 1

when I receive Let's go
repeat until stop = 1
    play special sound siren ambulance
play emotion sound sad
play emotion sound aaa
play emotion sound nice
```

Sample solution (fragments changed in relation to the previous script in the last task are flagged).

Task 3

We provide help and return to the base

Ask the children to select the appropriate emotions that may be felt during the robot's assistance. Then, among the participants, collect any ideas on how the robot can return to the base – the place where it took off.



```
when 🏳 clicked
reset travelled distance
set stop to 0
set distance to 0
set length to 255
broadcast Let's go
forever
    say join join You are join length cm from the obstacle join You traveled join distance cm
```

```
when I receive Let's go
forever
    repeat until stop = 2
        color 🔵 type ears
        color 🔵 type eyes
        color 🔵 type eyes
        color 🔵 type ears
```

```
when I receive Let's go
repeat until stop = 1
    go forward infinitely with 40 % speed
    set distance to distance travelled
    set length to distance from obstacle
    if length < 20 then
        set motors L: 0 R: 0
        set stop to 1
```

```
when I receive Let's go
repeat until stop = 1
    play special sound siren ambulance
play emotion sound sad
play emotion sound aaa
play emotion sound nice
broadcast Let's go back
```

Sample solution:

Lesson Summary:

In our program, we used variables in three different applications:

- *If* (a “length” variable) – as part of an expression in a conditional statement.
- *Repeat until* (a “stop” variable) – as an indication of a condition that ends the loop.
- A “distance” variable – as a variable that stores the distance traveled by the robot.

Now let the children play by modifying the programs and creating other units, for example, fire department, police department. If your class consists of a larger group of children, they can simultaneously send their fire department/police units to the specified location on a specific signal, for example, after turning of the lights, making noise, etc.

Interesting facts / Opening questions:

- What emergency services do the children know?
- Do the children know how the procedures for sending an ambulance, fire brigade, or police department work and who is responsible?
- The Emergency Call Center operator is responsible for dispatching services after reporting to 112.

Related resources:

- A video showing an execution of the program

https://portal.photon.education/pl/wideo/73

Author: Mateusz Chmielewski and Michał Nowak

Programming – advanced level

Programming the Photon Robot in Python – Getting to know Photon and Python!

Computer Science • Programming

Duration time:	**Robots:**	**Programming interface:**	**Accessories:**
45 min.	×1	Photon Blocks Python	Magic Dongle

Goals (for the children):

- To learn the basics of the Python programming language syntax and how to use it with the Photon Robot
- To learn how the Photon Robot works and to explore its main features
- To prepare a simple programming code in Python (using motion and sound)

Required items:

- one PC per team
- computers/smartphones with the Internet access (to watch videos)

Attachments:

https://photon.education/eko/1

Teaching methods:

- experiments
- group tasks
- discussion

Type of exercise:

- group

Lesson scenario:

1. **Introduce students to the subject of robotics**

   Ask the students to provide examples of robot implementations nowadays. Divide your students into several groups. Ask each group to watch a short video showing:

- An industrial robot – manipulator
- Melson - a humanoid robot
- Mars Rover

   Take a moment to talk about what do they all have in common, i.e., we communicate with them in the same way.

2. **Principles of working with a robot**

   Please tell the groups that in a moment they will receive a robot. But first, explain the rules that apply to all the Photon Robot classes. If you already have them, explain the rules or invite students to create a list together.

3. **Testing the robot's capabilities**

- Launch the Photon Magic Bridge application to connect and control the robot from the computer.
- Test the robot using the Photon Blocks interface. Let students explore its functions freely and experiment with them for 10 minutes. Make sure that everyone in the group has a chance to test the robot.

4. Programming in Python – an introduction

- Move on to programming in Python. Ask students: In your opinion, why do we learn more complex ways of programming if we can easily program the robot in Scratch or another simple visual language?
- Point out that simplifications limit our possibilities, while "mature" programming languages are more versatile and are used more often in real life
- Explain how the programming environment is set up.

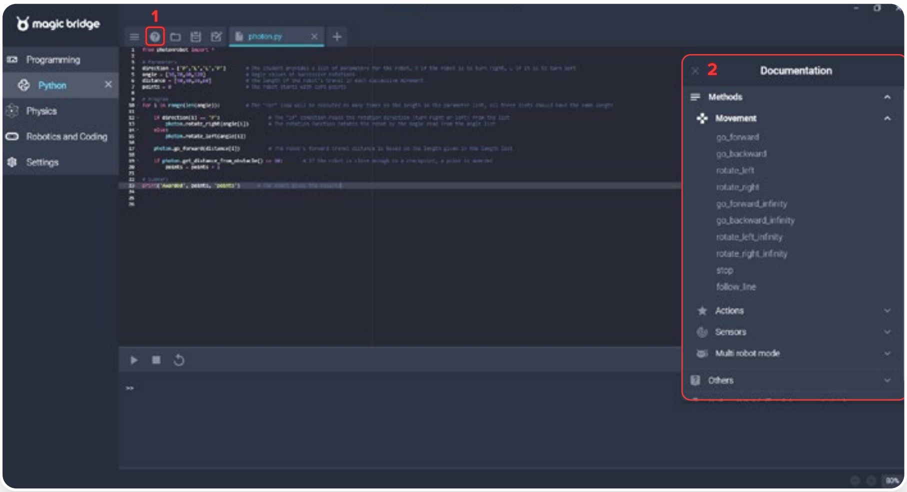
  - Individual commands, same as individual blocks in Scratch, are called “Methods” in Python.
  - Indicate where students can review Methods in regards to Photon Robot. To open the documentation (a list of all Methods for Photon), click the question mark icon on the upper menu bar (see below).

Python is a very popular programming language with many extensions and add-ons. Using it is like speaking in English, we use it every day, but we still don't know all the words – sometimes we have to Google them up or look them up in a dictionary. The same goes for programming. It’s a bit like someone telling you at the beginning of a conversation that there will be a lot of specific medical terminology, and to understand this conversation, you will need to use a dictionary! Because at first, the programming environment runs on only the basic Python commands and “methods” that we need, it runs faster. In this project, we will use all Methods from the “photonrobot” library.

  - Please pay attention to the internal comments. Please ignore parts of the code starting with the "#" – these are internal comments for developers verifying the code.

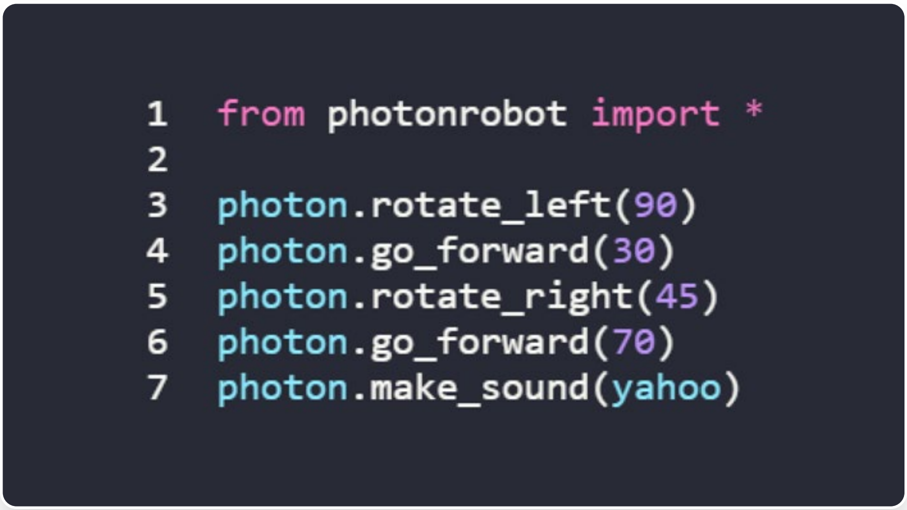
5. The First Program

Select and assign each group a specific destination (target) point. The programmed robot must reach the target and, once there, play a sound. Students must use the correct algorithm composed of several methods and test it.



```
from photonrobot import *

# Parameters
direction = ['R','L','L','R']      # [illegible]
angle = [illegible]                # [illegible]
distance = [illegible]             # [illegible]
points = 0                         # [illegible]

# Program
for i in range(len(angle)):        # [illegible]

    if direction[i] == 'R':            # [illegible]
        photon.rotate_right(angle[i])  # [illegible]
    else:
        photon.rotate_left(angle[i])

    photon.go_forward(distance[i])     # [illegible]

    if photon.get_distance_from_obstacle() [illegible]:     # [illegible]
        points = points + 1

# Summary
print('Awarded', points, 'points')      # [illegible]
```

List (Python methods for the Photon Robot) in the Photon Magic Bridge app



```
from photonrobot import *

photon.rotate_left(90)
photon.go_forward(30)
photon.rotate_right(45)
photon.go_forward(70)
photon.make_sound(yahoo)
```

Sample code to visualize the task at hand

Lesson Summary:

Summarize this lesson and allocate some time to answer questions – most likely, the students will get to know answers to their questions in one of the following classes.

Interesting facts / Opening questions:

- Polish universities always take top places in annual international Martian Robot challenges for young inventors of Mars Rovers.
- Robotic arms (manipulators) are most often used in the automotive industry, medical procedures, laboratory research, and space and ocean floor explorations.
- Where else could we use robots?

Author: Mateusz Chmielewski and Michał Nowak

Programming – advanced level

Programming the Photon Robot in Python – Going from point to point

computer science • programming

Duration time:	Robots:	Programming interface:	Accessories:
45 min.	×1	Python	Magic Dongle

Goals (for the children):

- To learn how to solve complex problems independently
- To learn complex functions that refer to lists
- To learn how to create code that allows passing through checkpoints as efficiently as possible
- To improve team work skills

Required items:

- obstacle markers – 7 pcs
- computer – 1 per group

Attachments:

https://photon.education/eko/4

Teaching methods:

- activities
- experiments

Type of exercise:

- group

Preparations:

In our fifth session of the series, you need to present an issue that has to be solved entirely by your students. As this is the halfway point of the course, it’s time to test the knowledge and skills acquired so far in practice. Ask the group to program a route leading the robot through several checkpoints placed on the floor. The developed program should implement a variable, a number of collected points, that increase with each obstacle passed.

Prepare several obstacles. The checkpoints could be anything from a paper cone made of sheets of paper to small boxes or any other appropriate objects.

Lesson scenario:

1. Start with a knowledge revision from the last few classes.
2. Tell the students the main objective of the class. There are several obstacles placed on the floor. Each group that is programming a robot should create a code that will make the robot approach each obstacle and receive a point for doing so. At the end of the whole route, the program should give us the total number of points scored.
3. Please remember to use resources made available to you with a detailed overview of all the methods that apply to the Photon Robot programming with examples of their use. Also, formulate a list of best practices to make the work easier:

- All variables, lists, and functions should be defined at the top of the code. These elements are fixed, and ideally, they should be next to each other. This strategy ensures that no crucial code parameter is deleted or changed if any part of the code below is changed.
- Each line should have a comment at the end, starting with # describing the elements students used in their code. It makes it easier for people verifying the code to understand its logic and find any possible errors.

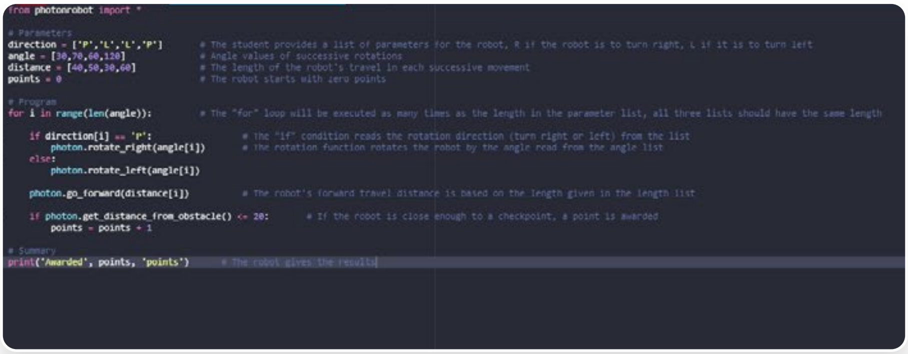
- This way, the code is divided into sections, descriptions in the comments, and the definitions that are separated from the actual programming code. All the individual loops, conditions, and other independent parts of the algorithm can be separated with empty lines to make the program more readable.

4. Ask the students to independently develop a code that meets the above conditions and test it.
5. Ask them to describe any problems they have encountered. In the end, try to come to a solution together.
6. The difficulty is that variables work differently in Python and in Scratch. These are not saved as a set code, and they function only within a given section of the code (you could try to move them to another section, but it's quite difficult).

- Your students need to create a code that loops in the following sequence:
  - getting to a checkpoint (an obstacle)
  - measuring distance
  - awarding a point if the robot is close enough to the obstacle.
- However, a simple loop cannot be used here because the movement and distance are slightly different in each repetition. This could be done if the obstacles were evenly spaced out, such as at each corner of a square or other symmetrical figure.
- In this situation, the loops can be programmed on the basis of a list of set parameters. Therefore, you need to create a list of directions, angles, and distances. In the first repetition of the loop, the robot should rotate in the first given direction by the first given angle and move by the first specified length (in centimeters) on the list. The lists must contain the same number of entries.
- In the *for* and *in range* loops the parameters specified in parentheses (len(angle)) mean that there will be as many repetitions as there are values in the "angle" list. You could put the name of any list in the parentheses. In this example, we chose "angle".
- The direction of rotation is determined using the *if else* loop, as there are only two directions to choose from. The program checks if the direction in the list is indicated as R and if so, the robot will rotate to the Right, otherwise it will rotate to the left. This is simpler than creating a second separate *if* loop to check if the robot should turn left.

- **i** is the number of repetitions in the program running in the loop. This designation comes from the word *iteration*. In combination with “the list name” and square brackets [], for example, direction[i], it means the first item on the first iteration in the “direction” list, the second item on the second iteration in the “direction” list and so on.
- You might also want to explain the meaning of symbol ==, which has a different function than = in many programming languages. The double equals sign does not precede a value, it just describes it in a mathematical sense. That’s why it is used in equations.
- We confirm the task as complete with the familiar *if* loop. A point is awarded if the distance is less than or equal to 20 cm (8 inches) to a checkpoint at measurement time. As a result, the variable aka “points” increases. We use the following formula for that, points = points + 1. Mathematically this formula is incorrect, but in programming, it is used often. The equal sign gives a new value rather than describes an existing value.

7. Test the program you all prepared together and summarize the activity. Focus on any new elements introduced by your students. Praise all the students who independently came up with solutions to the discussed problem.



```
from photonrobot import *

# Parameters
direction = ['P','L','L','P']       # The student provides a list of parameters for the robot, R if the robot is to turn right, L if it is to turn left
angle = [30,70,60,120]              # Angle values of successive rotations
distance = [40,50,30,60]            # The length of the robot's travel in each successive movement
points = 0                          # The robot starts with zero points

# Program
for i in range(len(angle)):         # The "for" loop will be executed as many times as the length in the parameter list, all three lists should have the same length

    if direction[i] == 'P':                 # The "if" condition reads the rotation direction (turn right or left) from the list
        photon.rotate_right(angle[i])       # The rotation function rotates the robot by the angle read from the angle list
    else:
        photon.rotate_left(angle[i])

    photon.go_forward(distance[i])          # The robot's forward travel distance is based on the length given in the length list

    if photon.get_distance_from_obstacle() <= 20:       # If the robot is close enough to a checkpoint, a point is awarded
        points = points + 1

# Summary
print('Awarded', points, 'points')      # The robot gives the results
```

Sample code to visualize the task at hand

Interesting facts / Opening questions:

What is the most challenging thing about orienteering? Finding designated points and proper logistics, i.e., finding the shortest route connecting all the points.