# Open Roberta (Blockly-based)



Andrea Sterbini – sterbini@di.uniroma1.it

# Open Roberta
# Simple visual robot/microcontroller programming

## Built with Blockly

lab.open-roberta.org

Transforms visual programs to Python/Java/C/C++ (depending on which type of robot)

Deploys the program to the robot

Runs the program on the robot (or a browser-based simulation)

Debug the program by stepping/tracing it

Visual interface to the robot configuration details

Motors, sensors, wheels geometry, LCD displays, LEDs, ports, shields

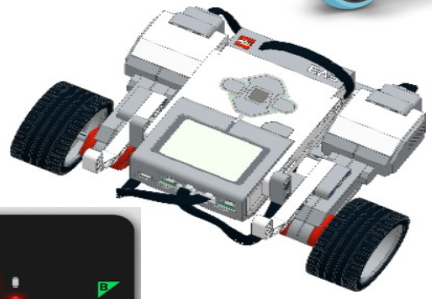
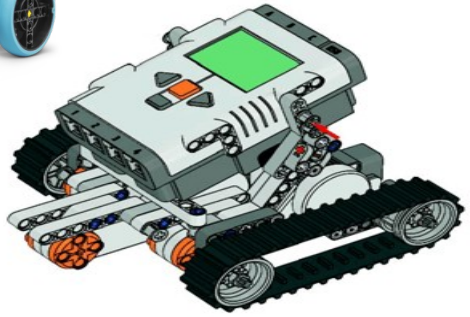
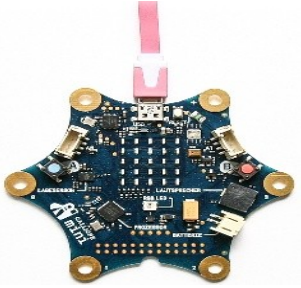
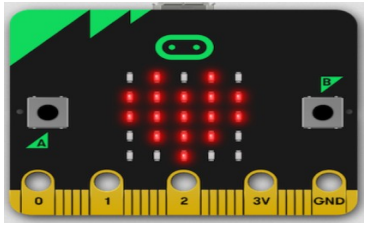
**WIKI:** https://jira.iais.fraunhofer.de/wiki/display/ORInfo

# Open Roberta
## Many robots and embedded systems supported

NAO, BOB3, Lego WeDo2/EV3/NXT/Spike, Robotino
Bot'n Roll, Calliope Mini, Micro:bit, Arduino, mBot, senseBox

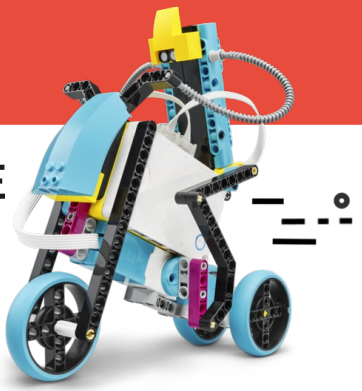
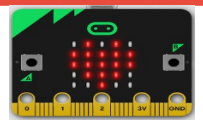


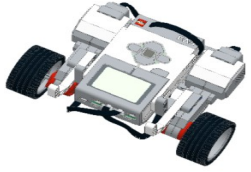




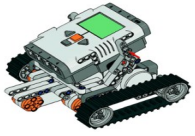

# Many generated languages

Python: Lego EV3 & SPIKE  micro:bit  NAO robotino

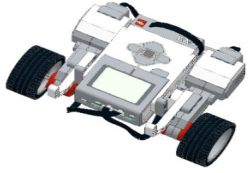


C/C++: Arduino, Bot'n roll, Lego NXT and EV3, BOB3, SenseBox, mBot, Calliope



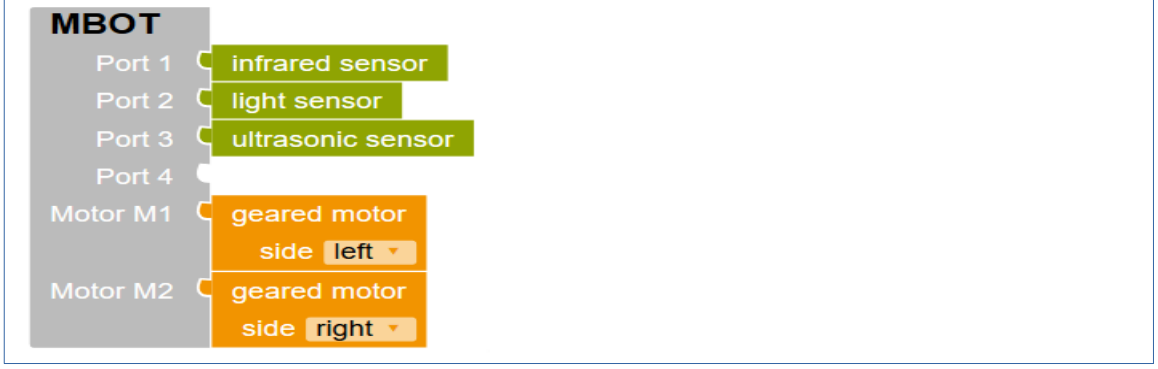


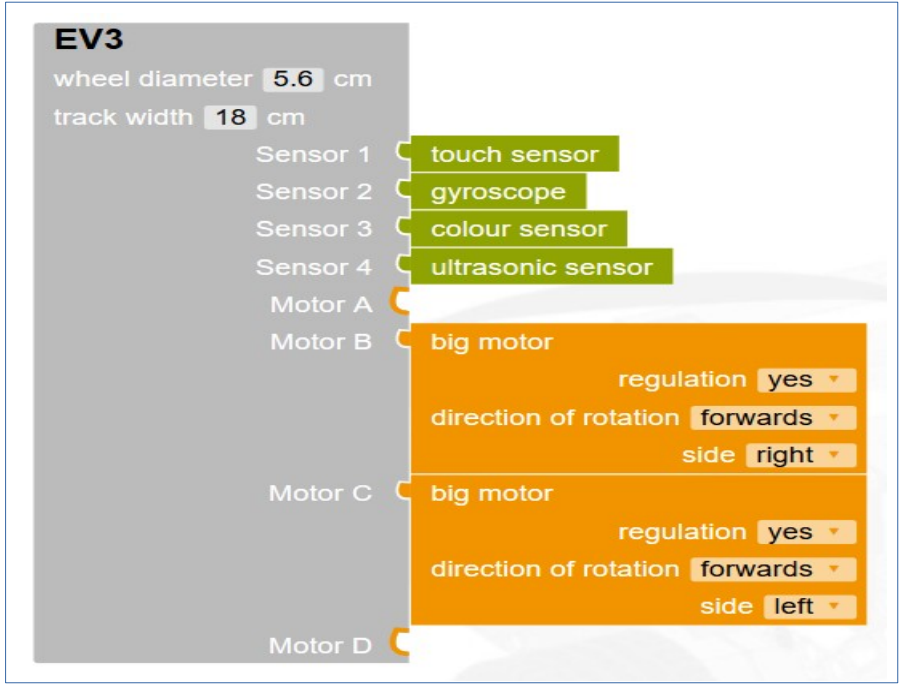


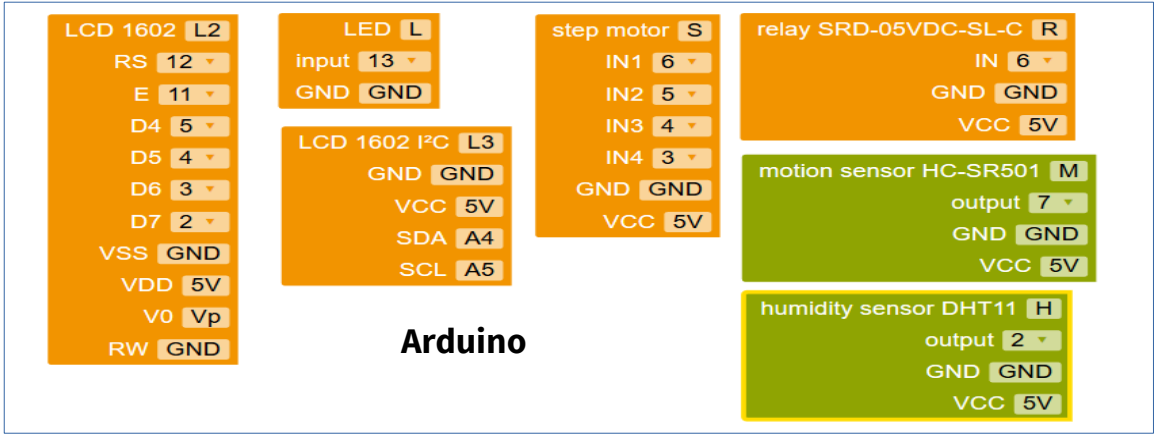

Java: Lego EV3 + Java firmware 

Json: Lego WeDo (runs in the browser)

# Visual configuration of what sensors/actuators are connected (and where) to the Robot/Microcontroller



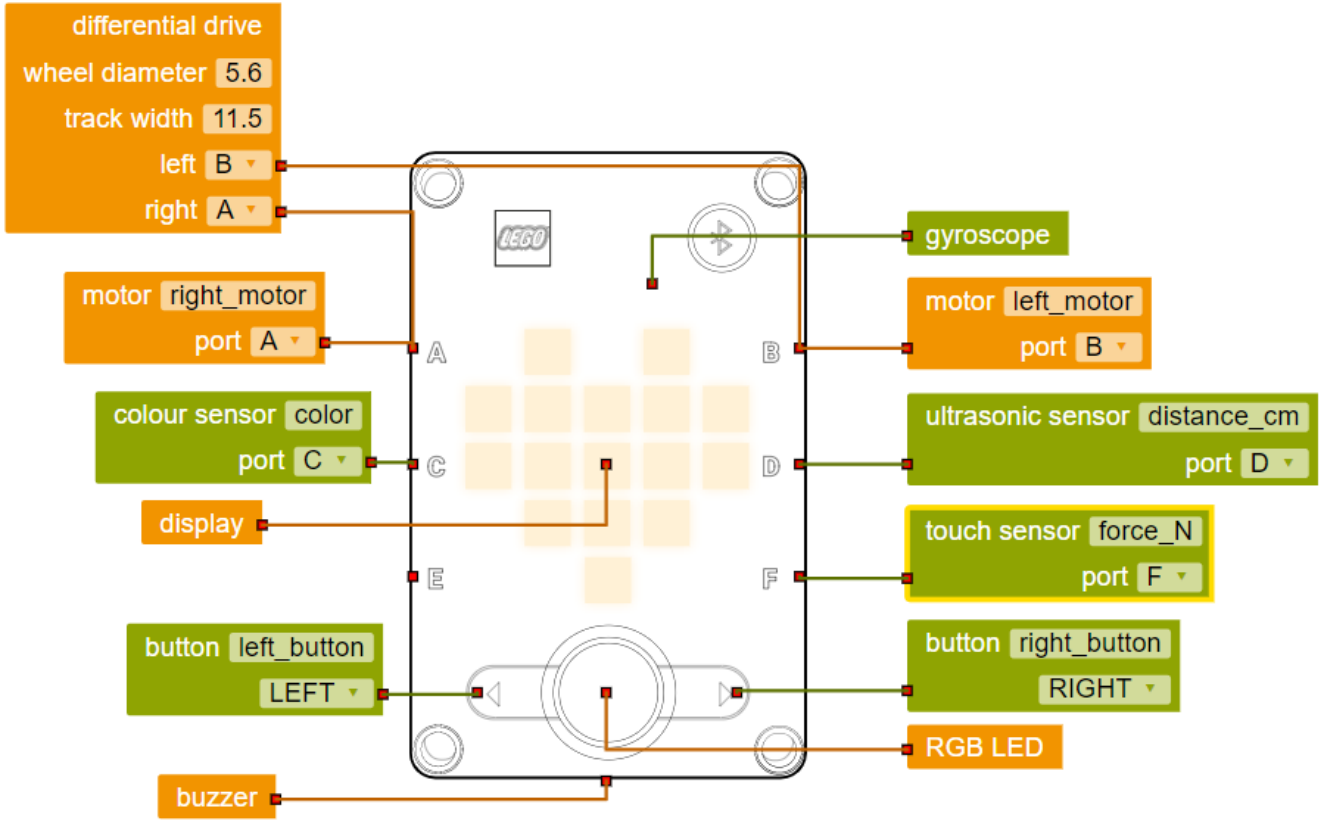



# E.G. the Java configuration for EV3 + Lejos firmware

```
public class NEPOprog {
  private static Configuration brickConfiguration;
  private Set<UsedSensor> usedSensors = new LinkedHashSet<UsedSensor>();
  private Hal hal = new Hal(brickConfiguration, usedSensors);
  public static void main(String[] args) {
    try {
      brickConfiguration = new EV3Configuration.Builder()
        .setWheelDiameter(5.6)
        .setTrackWidth(18.0)
        .addActor(ActorPort.B, new Actor(ActorType.LARGE, true,
                                         DriveDirection.FOREWARD, MotorSide.RIGHT))
        .addActor(ActorPort.C, new Actor(ActorType.LARGE, true,
                                         DriveDirection.FOREWARD, MotorSide.LEFT))
        .build();
...
```

# E.G. Lego SPIKE config. in MicroPython



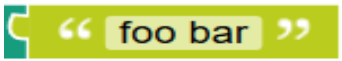

You can rename sensors or motors for better code readability

Producing var names that contain both the sensor type and the given name)

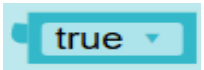
```
import spike
touch_sensor_force_N = spike.ForceSensor('F')
ultrasonic_sensor_distance_cm = spike.DistanceSensor('D')
color_sensor_color = spike.ColorSensor('C')
```

# Data types: statically typed vars/args

Number 

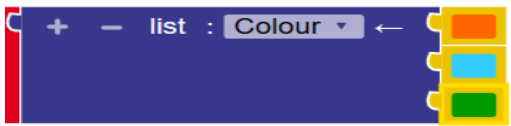

String 


Colour 

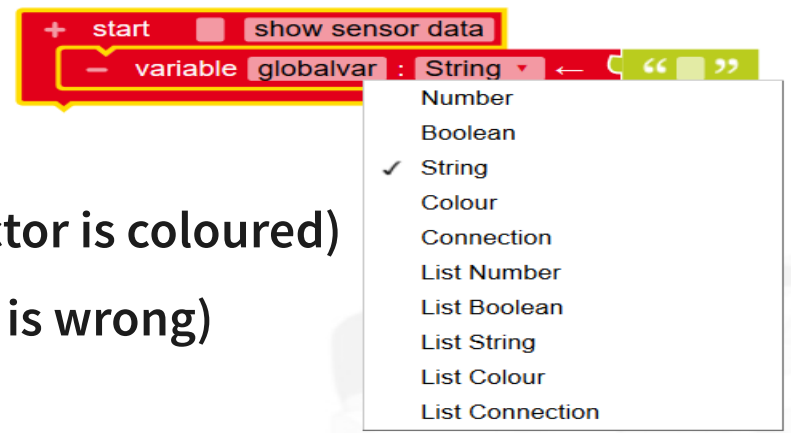
Boolean 


Image 

Connection 


List of <T> 


(with all elements of the same type)




Variables and arguments are visually typed (the connector is coloured)

Data types are visually enforced (cannot join if the type is wrong)

# Execution model: single thread

Single thread of execution (main program/main loop)

| | | |
|---|---|---|
| New Functions? | YES | |
| Global variables? | YES | (defined only at main level) |
| Local variables? | YES? | (must be defined as function's arguments) |
| Messages? | NO? | (but some robots can communicate over BT or serial) |
| Events? | NO | |

Events must be simulated by polling the sensors + "when"

Lego EV3 robots can connect via BT and exchange text messages

Other robots can communicate over serial wires

# “Advanced-enough” programming

Counted Loops, Foreach,
Repeat until, Repeat while

Continue, break

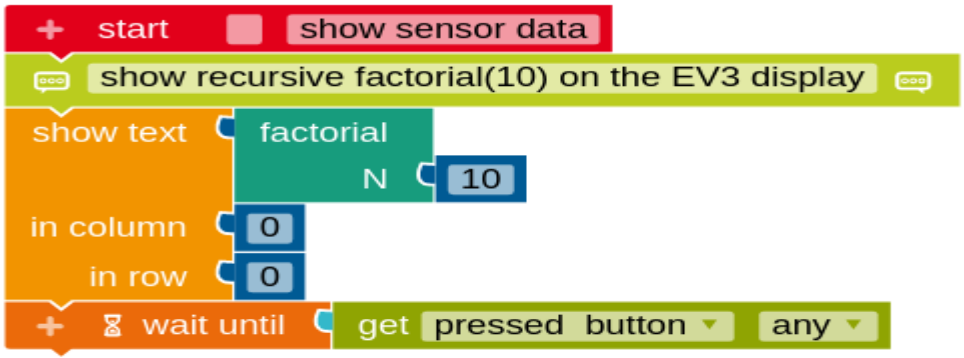
Wait N ms,
Wait until condition …
or other condition … or else

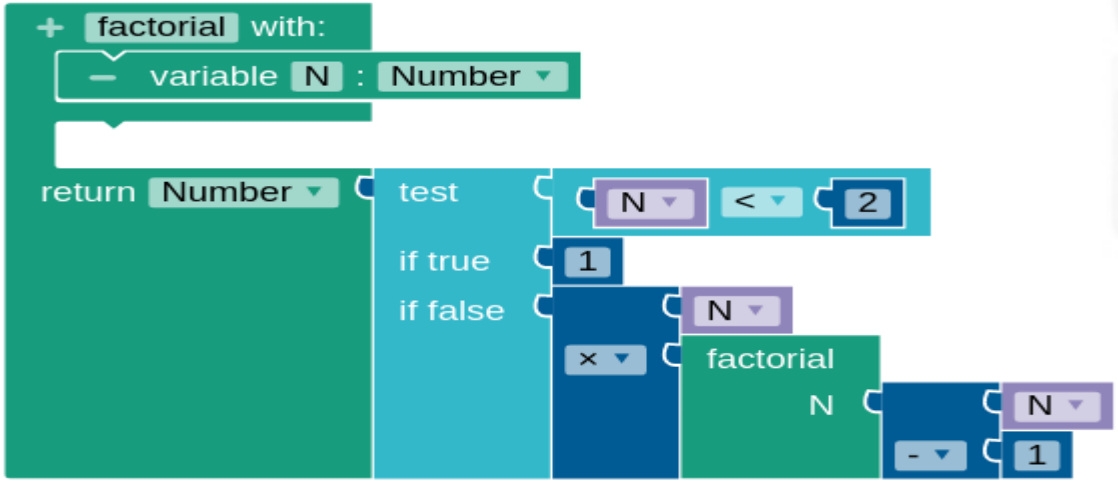
If, if-else, if-elif-…-else

Constrain value between

Recursion? YES

Local variables as arguments(!)



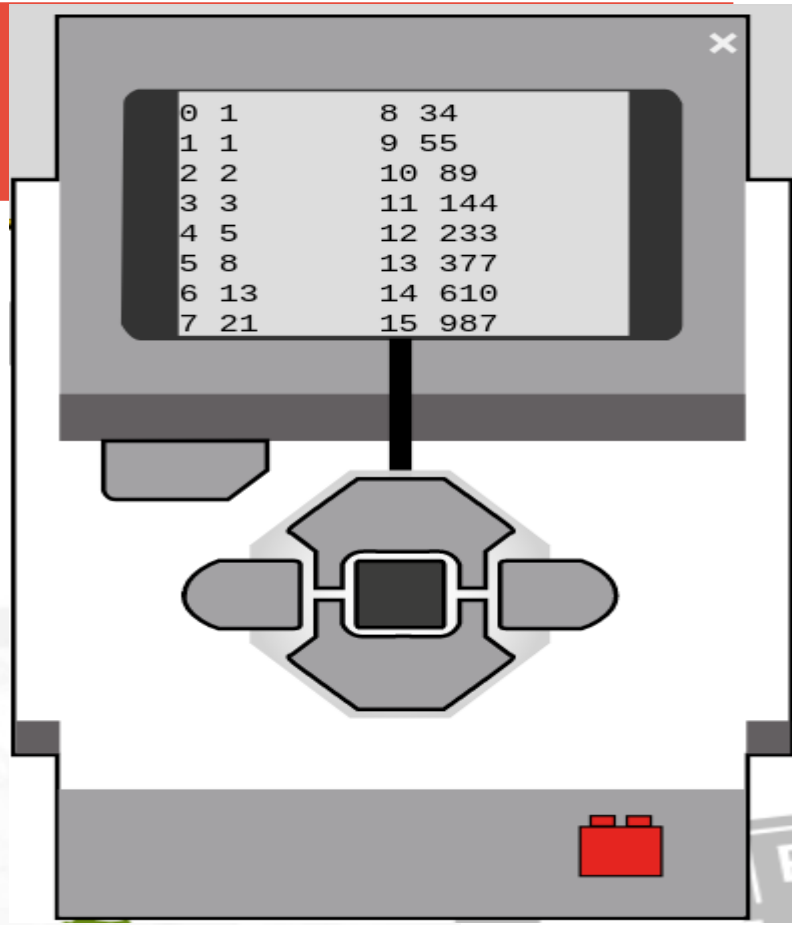
# Example: efficient recursive Fibonacci (forward loop simulation)

Fibonacci(0) value

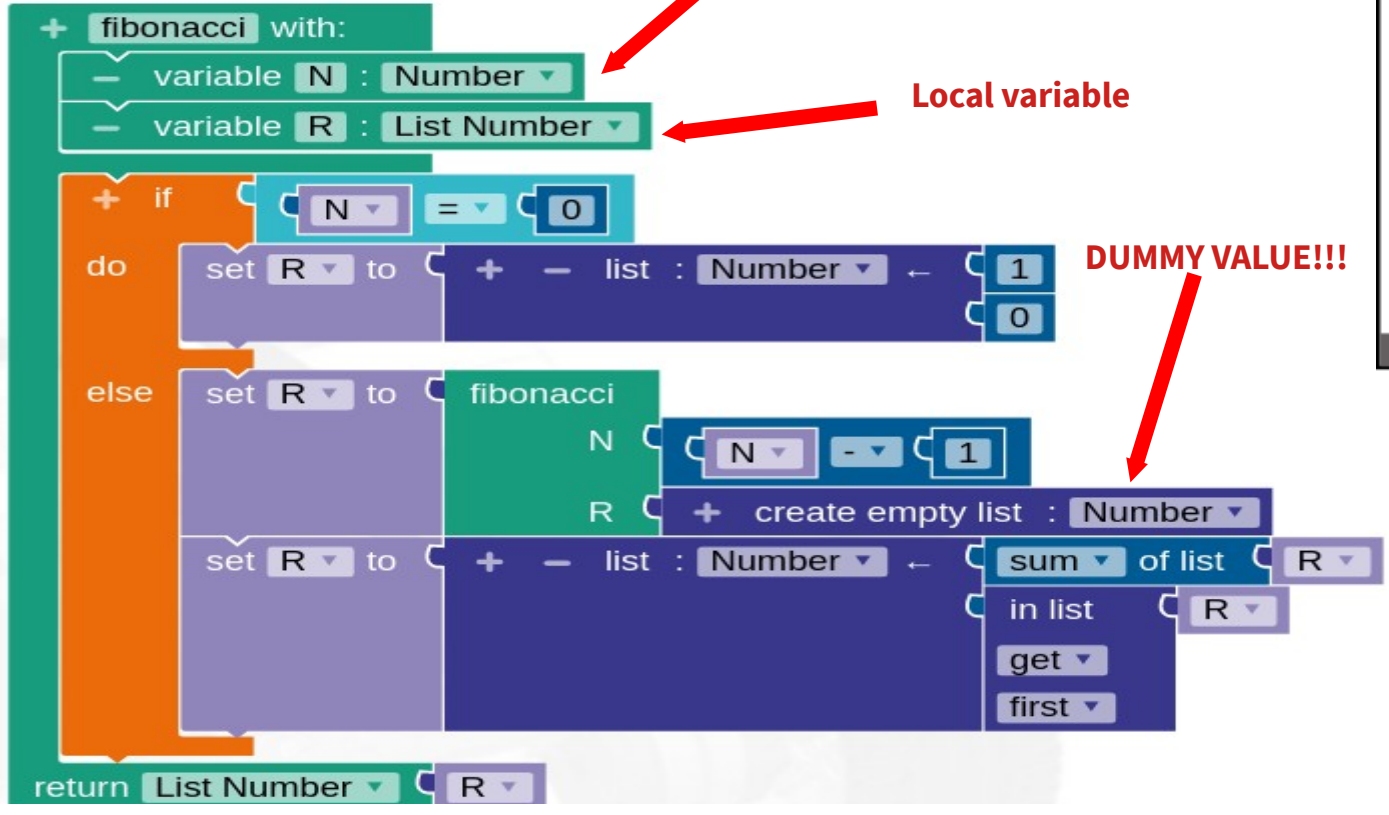
previous value

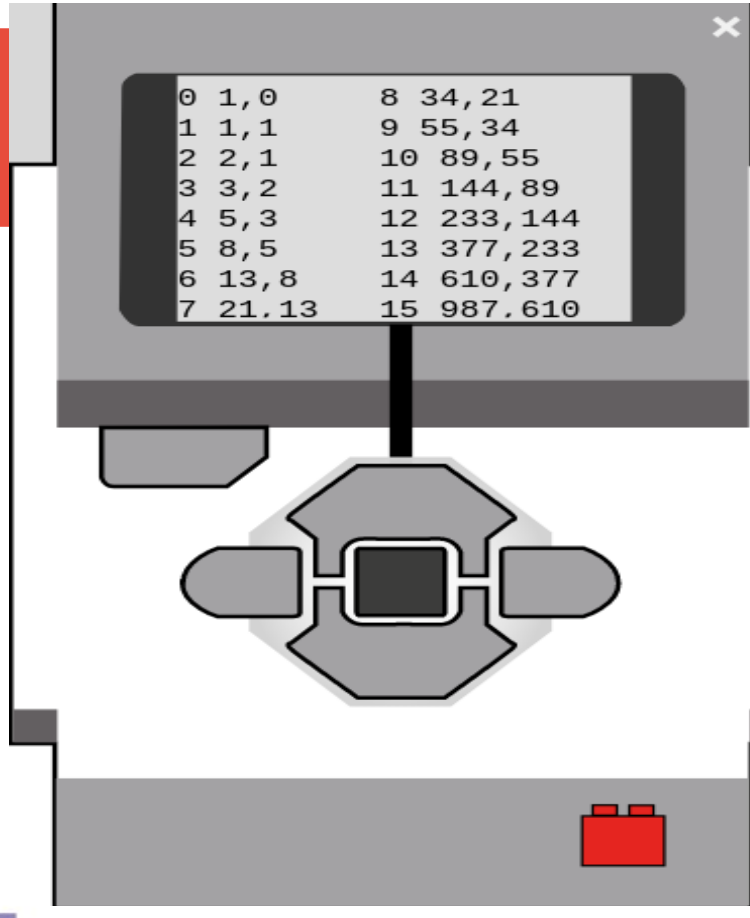
```
+ fibonacci with:
    – variable N2 : Number

  return Number   fibonacci_ N N2 F 1 Fold 0
```

```
+ fibonacci_ with:
    – variable N : Number
    – variable F : Number
    – variable Fold : Number

  return Number   test N = 0
                  if true F
                  if false fibonacci_ N N - 1 F F + Fold Fold F
```



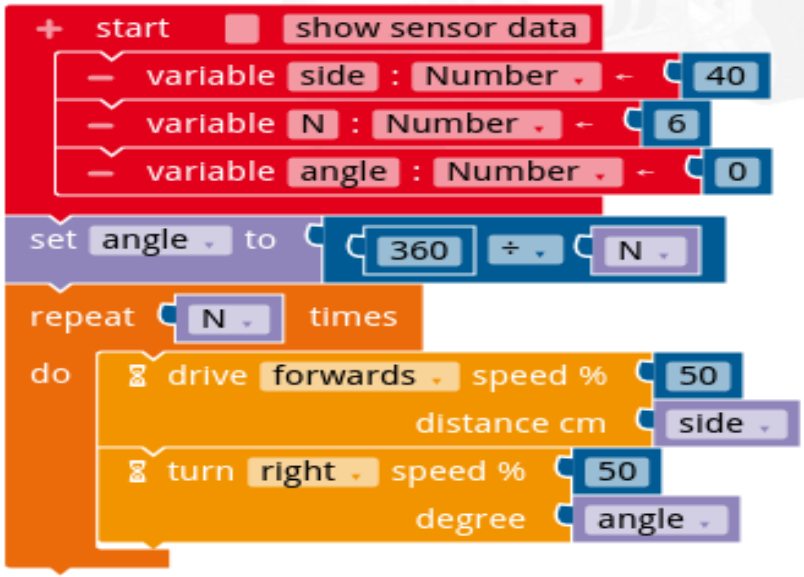
# Example 2: efficient recursive Fibonacci (backward loop simulation returning a pair)



# Example: polygon movement in C++

```
// MAIN code
   float ___side = 40;
   float ___N = 6;
   float ___angle = 0;

   public void run() throws Exception {
      ___angle = 360 / ((float) ___N);
      for ( float ___k0 = 0; ___k0< ___N; ___k0+= 1 ) {
         hal.driveDistance(DriveDirection.FOREWARD, 50, ___side);
         hal.rotateDirectionAngle(TurnDirection.RIGHT, 50, ___angle);
      }
   }
}
```



# Our experience: 10 lessons for 9 and 10 y/o students in K4 nd K5

**Phase 1) Role play on a grid + instructions with arrows, repetitions and conditions**

Algorithm as a sequence of instructions with conditional paths

**Phase 2) small programs on Scratch with turtle graphics**

Variables and turtle graphic

**Phase 3) small programs with Lego EV3 robots in Open Roberta**

Robots in class moving around, calibration, sensor polling while moving

**We had to pay attention to:**

**- Network connectivity (if possible install the software locally or on teacher's laptop)**
**- loose wires in the robot that raise strange exceptions for disconnected sensors**
**- Bluetooth was a mess (use wifi, it's more stable and supported)**
**- local teachers that don't know how to help (prepare your helpers on the lesson and tools)**

# When possible use a local installation (for a better network access)

OpenRoberta is Open source

Available on https://github.com/OpenRoberta/openroberta-lab

Java based, built with Maven

You can enable/disable separately each module/robot to fit your available robots

You can run the server on your laptop in class and share your wifi

Then all Robots and PC browsers in the class are connect by wifi to your laptop

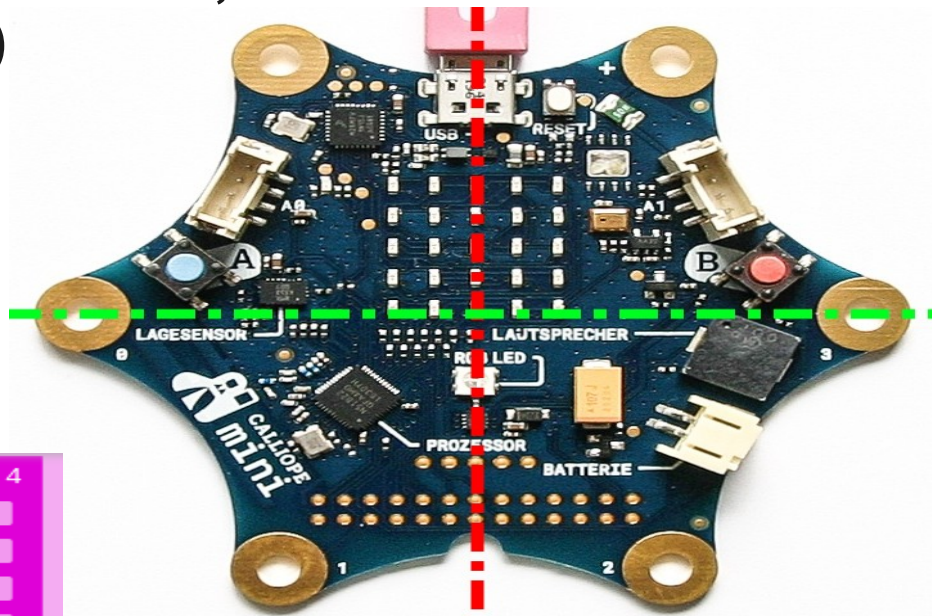
(Available also for Android)

# Microcontrollers:

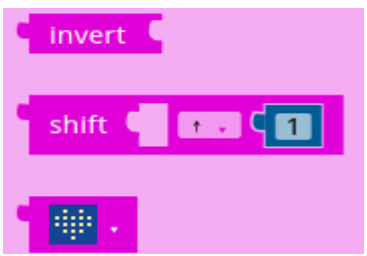
## Calliope mini - a lot of sensors

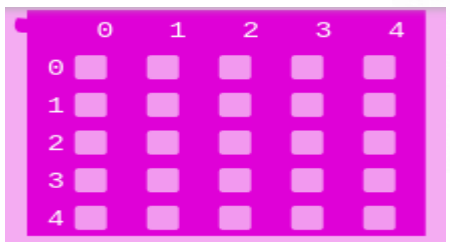
Sensors: buttons, tilt, compass, temperature, light, sound intensity, gyroscope, accelerometer, humidity, ultrasound, external analogue sensors (e.g. colour)

Actuators: 5 x 5 LED matrix
external 4-digits display
serial port to terminal
external motor controllers




Special blocks for 5x5 LED matrix



# NAO: a small “dancing” robot

Predefined complex movements (tai chi, wave, blink, point)

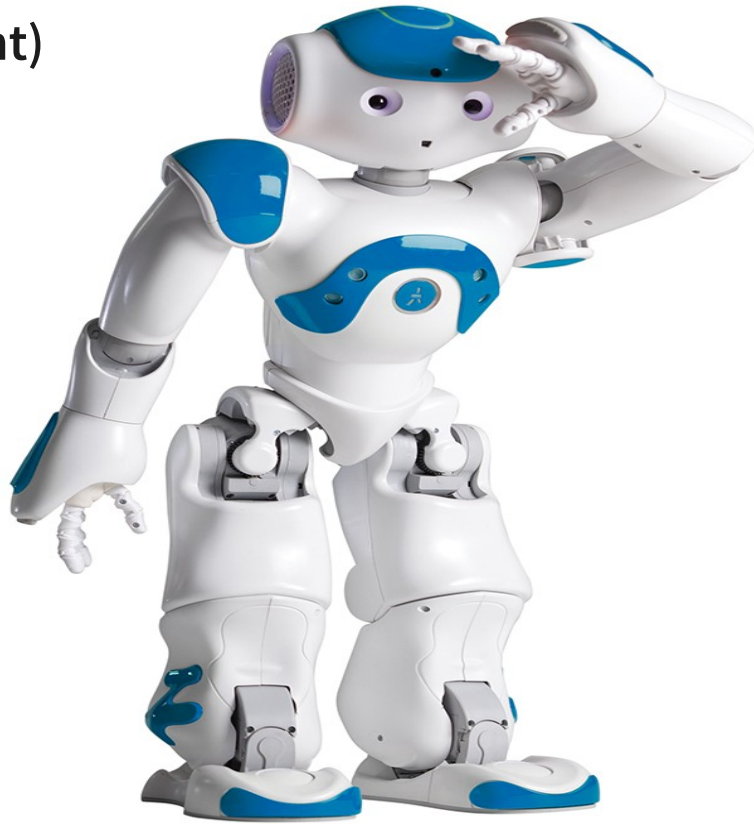
Walk to, hand movements in space, …

Can record a video or picture

Can remember/recognize a face

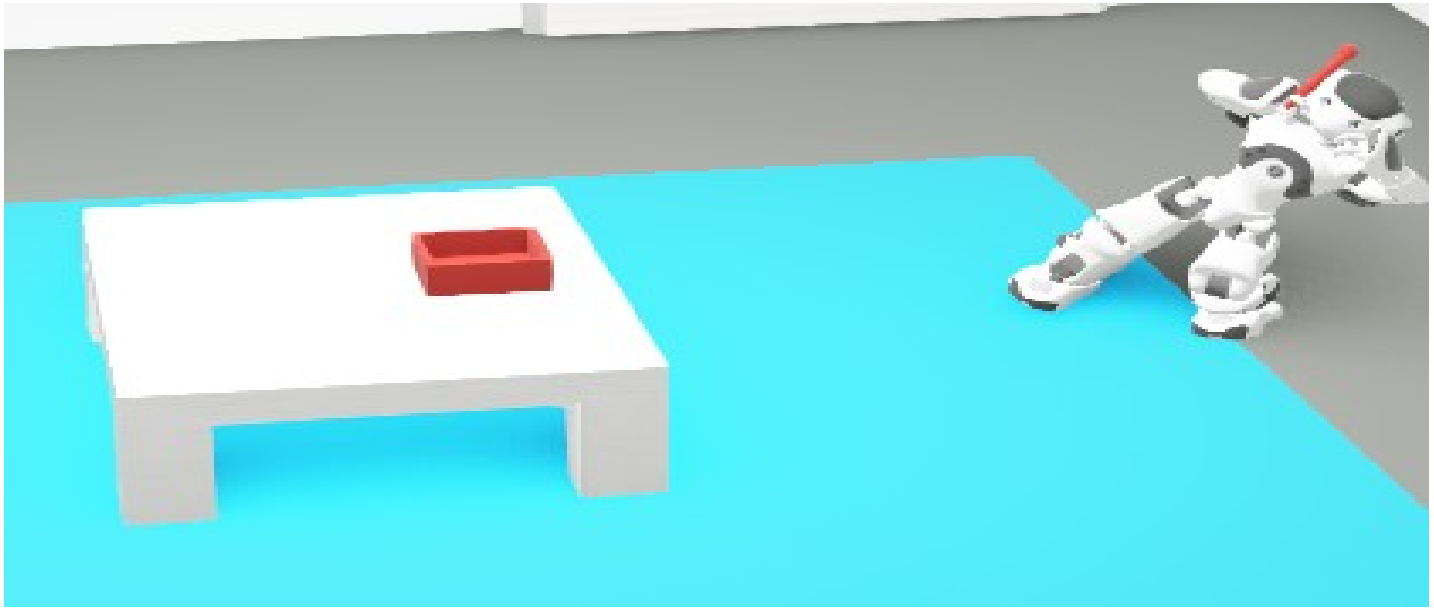
Play sounds, speak (text to speech)

Programmed in Python



# 3D simulation in browser

E.G. making a Tai chi move



# Demo

Demo