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 type of robot)

Deploys the program to the robot

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

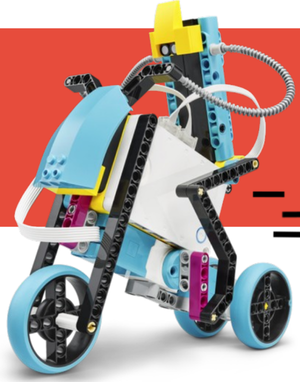
Debug the program by stepping/tracing it in the simulator

Visual interface to the robot configuration details

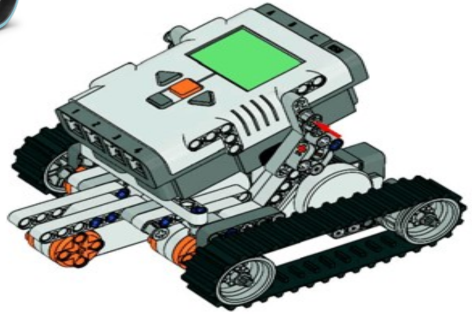
Motors, sensors, wheels geometry, LCD displays, LEDs, ports, shields

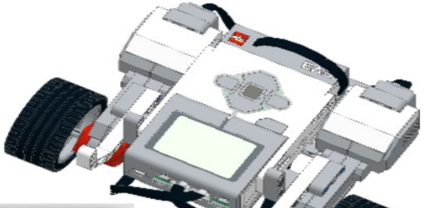
WIKI: https://jira.iais.fraunhofer.de/wiki/display/ORInfo

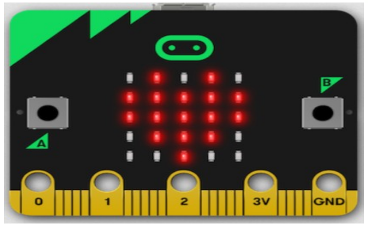
Open Roberta:
many robots and embedded systems

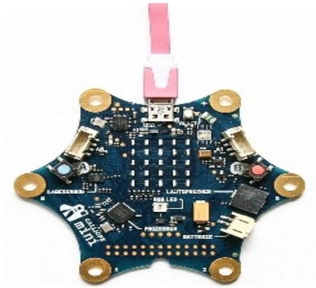


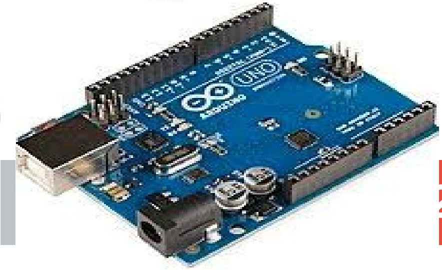
NAO, BOB3, Lego WeDo2/EV3/NXT/Spike, Robotino
Bot’n Roll, Calliope Mini, Micro:bit, Arduino,
mBot, senseBox









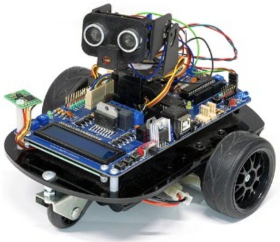


New robots! (+ neural networks!)

Open Roberta xNN

Thymio

micro:bit Joy-Car



Bot'n Roll

Bionic Flower

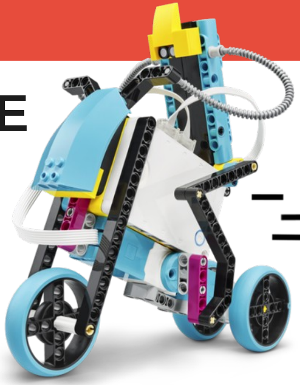
Spike Prime / Robot Inventor

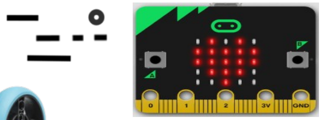
mBot 2

Bionics Kit

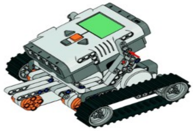
Many generated languages

Python: Lego EV3 & SPIKE micro:bit NAO robotino





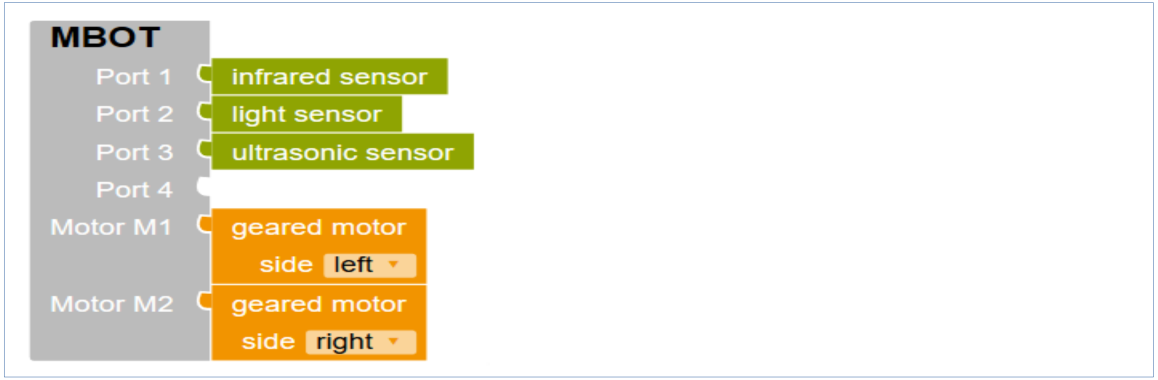
C/C++: Arduino, Bot'n roll, Lego NXT/EV3, BOB3, SenseBox, mBot, Calliope



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



MBOT

Port 1 infrared sensor

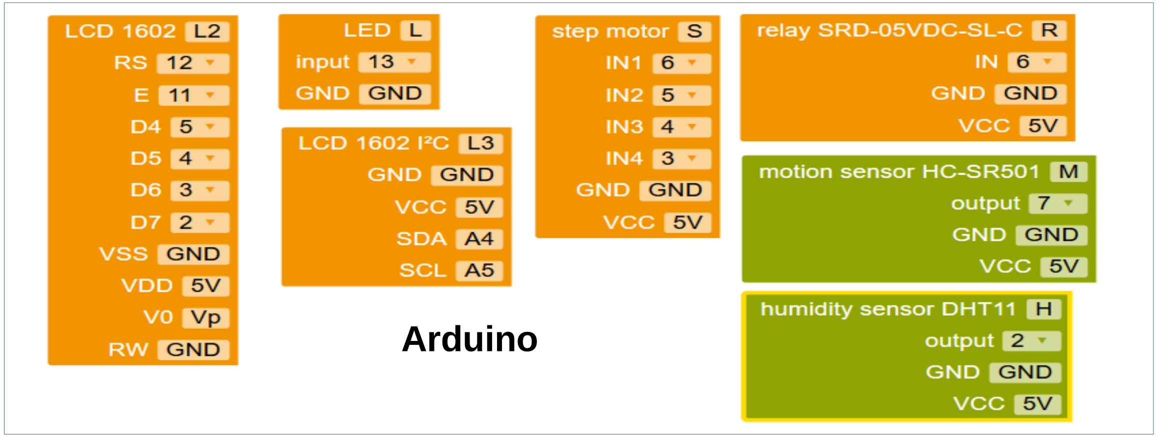
Port 2 light sensor

Port 3 ultrasonic sensor

Port 4

Motor M1 geared motor
side left

Motor M2 geared motor
side right



LCD 1602 L2
RS 12
E 11
D4 5
D5 4
D6 3
D7 2
VSS GND
VDD 5V
V0 Vp
RW GND

LED L
input 13
GND GND

LCD 1602 I²C L3
GND GND
VCC 5V
SDA A4
SCL A5

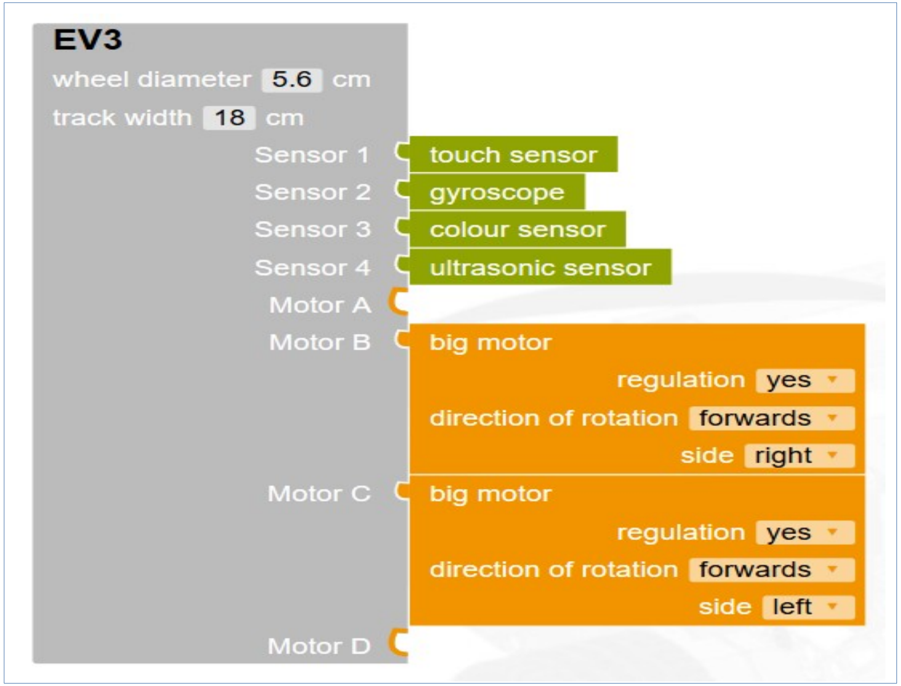
step motor S
IN1 6
IN2 5
IN3 4
IN4 3
GND GND
VCC 5V

relay SRD-05VDC-SL-C R
IN 6
GND GND
VCC 5V

motion sensor HC-SR501 M
output 7
GND GND
VCC 5V

humidity sensor DHT11 H
output 2
GND GND
VCC 5V

Arduino



EV3

wheel diameter 5.6 cm

track width 18 cm

Sensor 1 touch sensor

Sensor 2 gyroscope

Sensor 3 colour sensor

Sensor 4 ultrasonic sensor

Motor A

Motor B big motor
regulation yes
direction of rotation forwards
side right

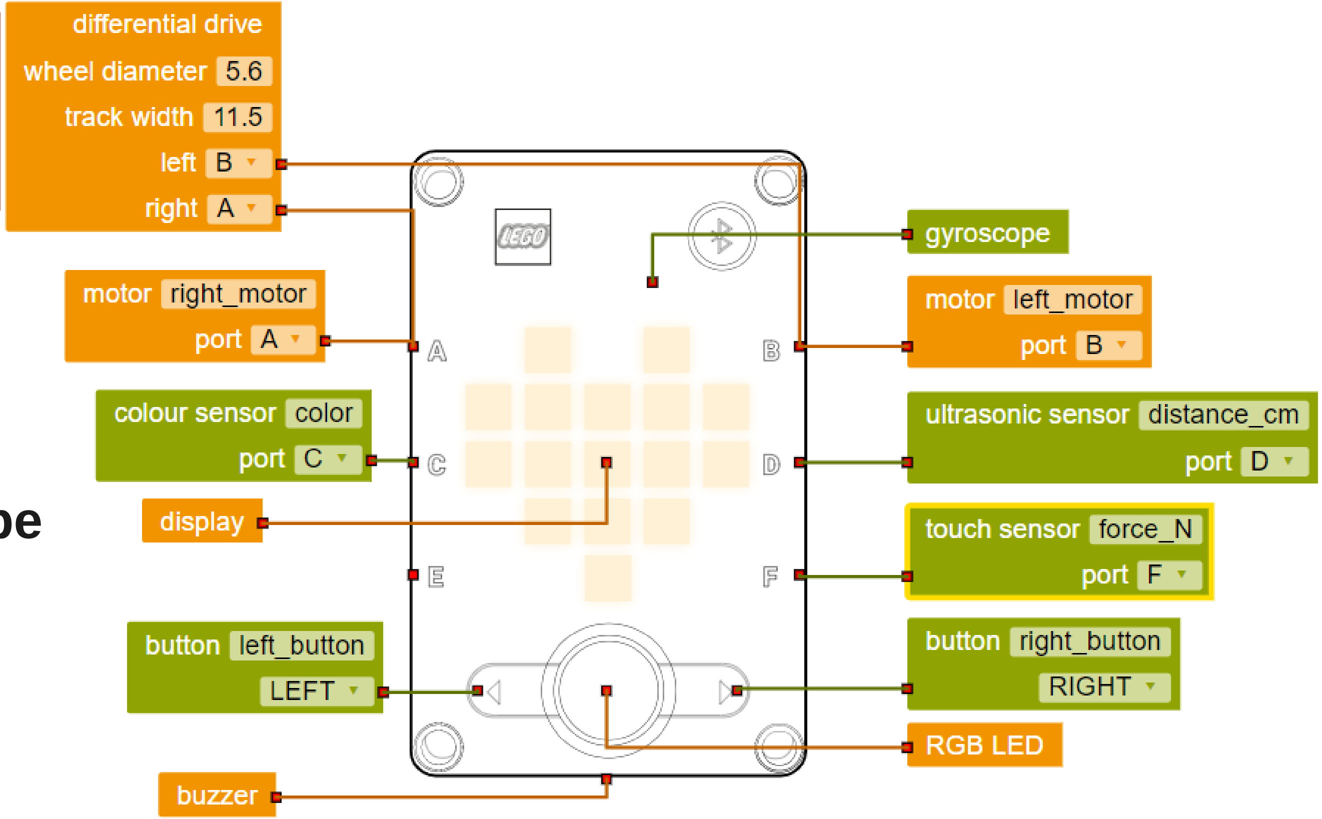
Motor C big motor
regulation yes
direction of rotation forwards
side left

Motor D

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



differential drive
wheel diameter 5.6
track width 11.5
left B
right A

motor right_motor
port A

colour sensor color
port C

display

button left_button
LEFT

buzzer

gyroscope

motor left_motor
port B

ultrasonic sensor distance_cm
port D

touch sensor force_N
port F

button right_button
RIGHT

RGB LED

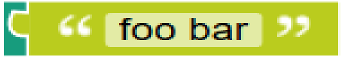
You can rename sensors or motors for better code readability

Producing var names containing 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

tring


Colour

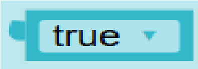
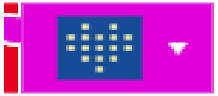
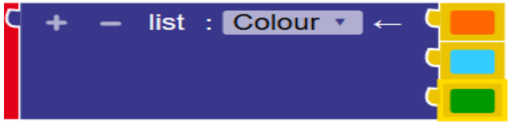
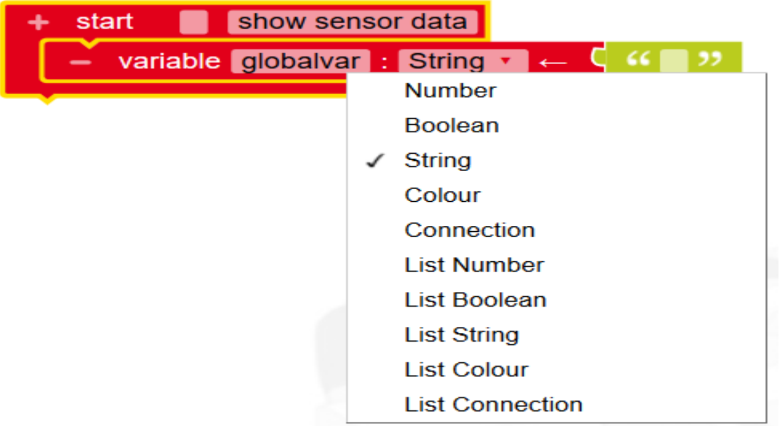
Boolean


Image


Connection

List of <T>


(<u>same type elements</u>)



Variables and arguments are <u>visually typed</u>

(the connector is coloured)

Data types are <u>visually enforced</u>

(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

<u>Continue, break</u>

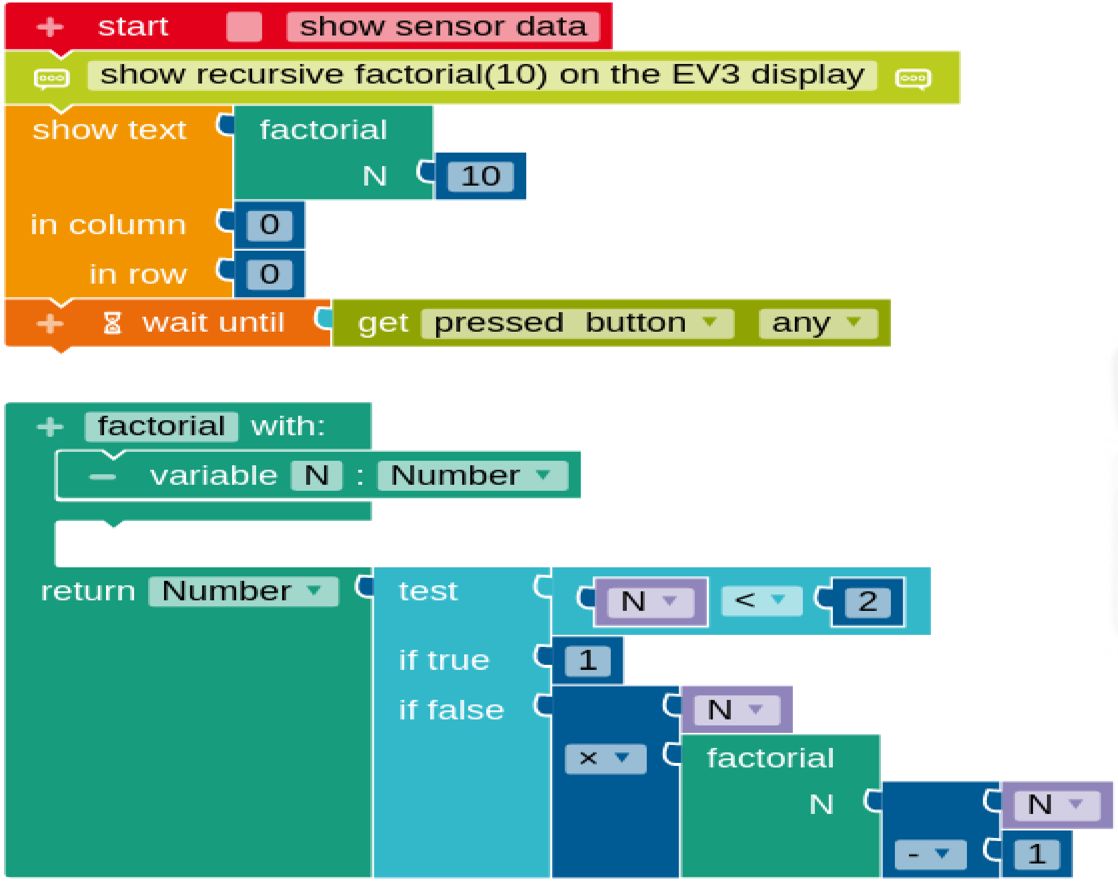
Wait N ms,
Wait until condition …
or other condition … or else

If, if-else, if-elif-…-else

Constrain value between

Recursion? YES

Local variables as arguments(!)

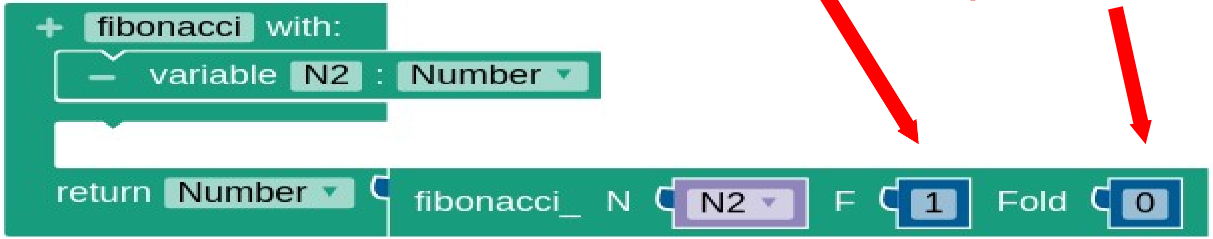


```
+ start     show sensor data
  show recursive factorial(10) on the EV3 display
  show text   factorial
                      N   10
  in column   0
     in row   0
+ wait until  get  pressed  button ▾   any ▾

+ factorial with:
    − variable N : Number ▾

  return Number ▾   test       N ▾  < ▾  2
                    if true    1
                    if false        N ▾
                               × ▾  factorial
                                            N      N ▾
                                              - ▾  1
```

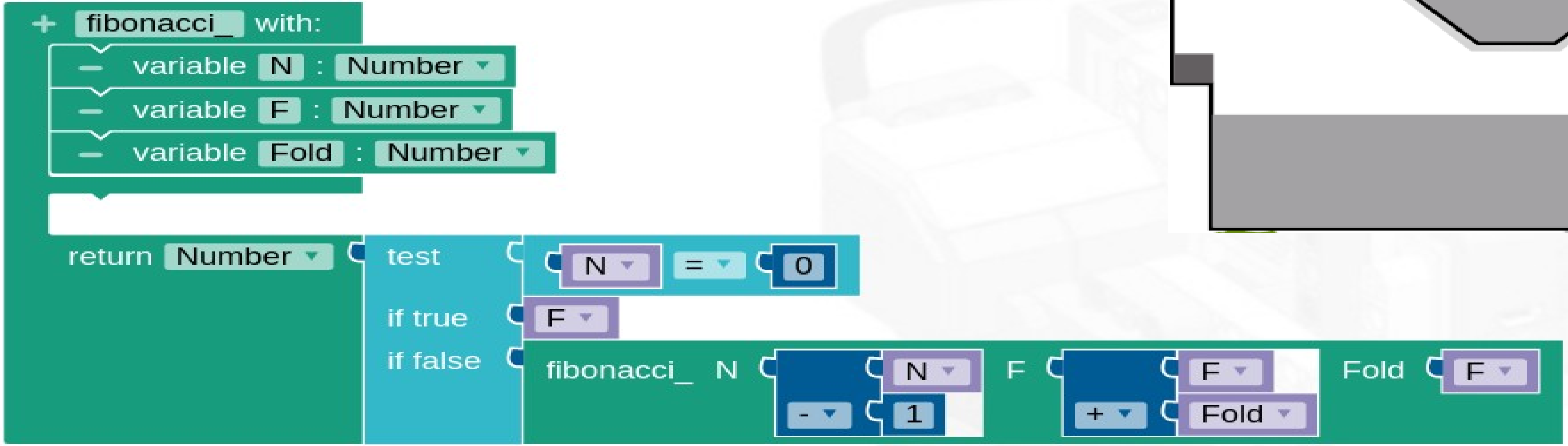
Example: efficient recursive Fibonacci (forward loop simulation)

Fibonacci(0) value

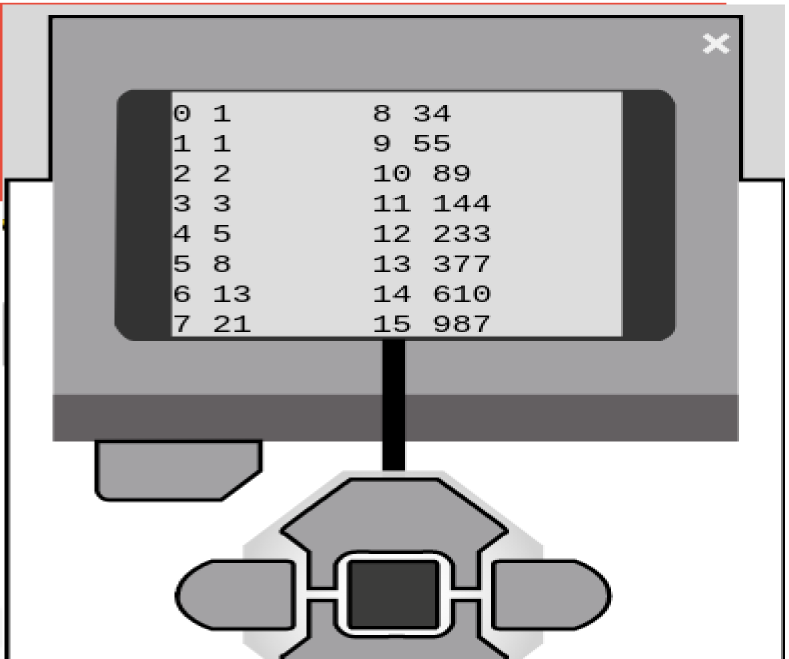
previous value



```
+ fibonacci with:
    − variable N2 : Number
  return Number    fibonacci_  N  N2   F  1   Fold  0
```

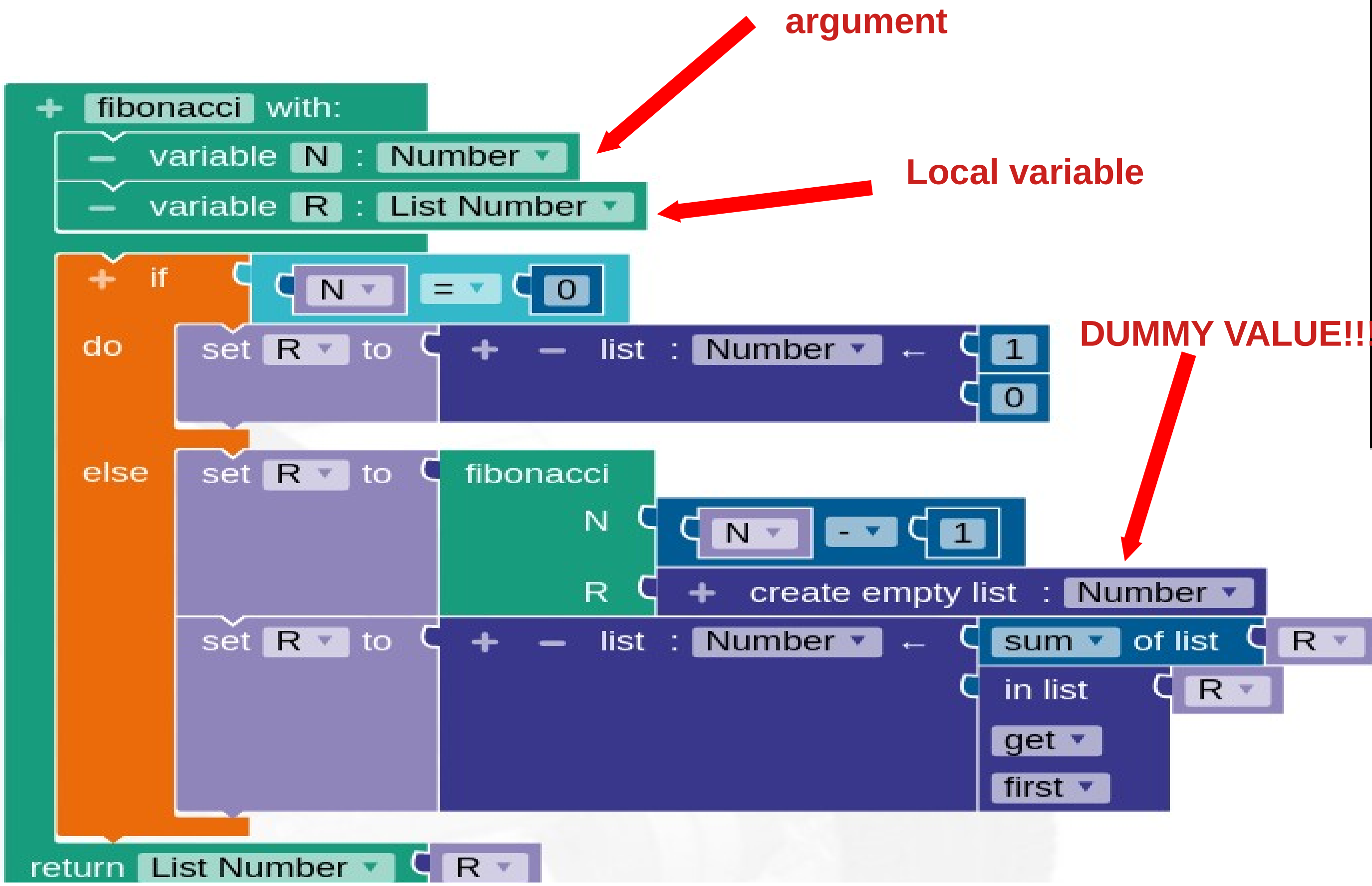


```
+ fibonacci_ with:
    − variable N : Number
    − variable F : Number
    − variable Fold : Number
  return Number  test      N = 0
                 if true   F
                 if false  fibonacci_  N  - N      F  + F       Fold  F
                                           1          Fold
```



```
0 1         8 34
1 1         9 55
2 2         10 89
3 3         11 144
4 5         12 233
5 8         13 377
6 13        14 610
7 21        15 987
```

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

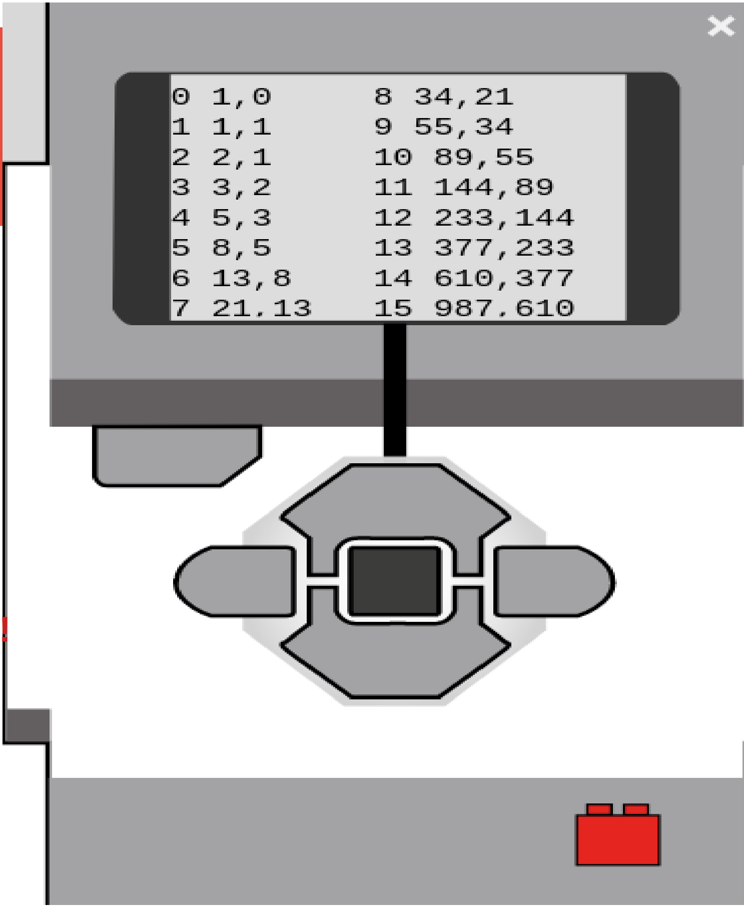


argument

Local variable

DUMMY VALUE!!!

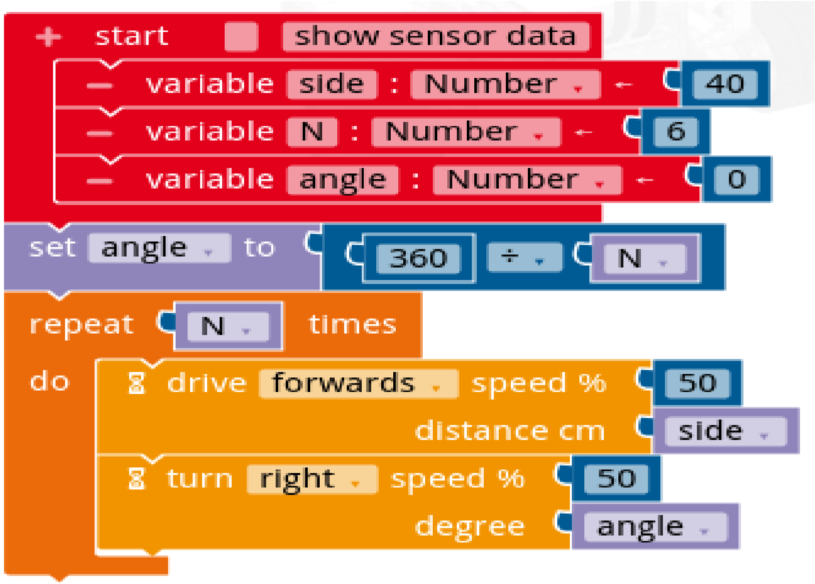
```
+ fibonacci with:
    - variable N : Number
    - variable R : List Number
  + if  N = 0
    do    set R to  + - list : Number ← 1
                                        0
    else  set R to  fibonacci
                      N  N - 1
                      R  + create empty list : Number
          set R to  + - list : Number ← sum of list R
                                        in list R
                                        get
                                        first
  return List Number  R
```



```
0 1,0       8 34,21
1 1,1       9 55,34
2 2,1       10 89,55
3 3,2       11 144,89
4 5,3       12 233,144
5 8,5       13 377,233
6 13,8      14 610,377
7 21,13     15 987,610
```

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);
        }
    }
}
```



```
start   show sensor data
    variable side : Number ← 40
    variable N : Number ← 6
    variable angle : Number ← 0
set angle to 360 ÷ N
repeat N times
do  drive forwards speed % 50
              distance cm side
    turn right speed % 50
              degree angle
```

Our experience:
10 lessons for 9 and 10 y/o students in K4 and 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 your laptop)
- loose wires in the robot that raise 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 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

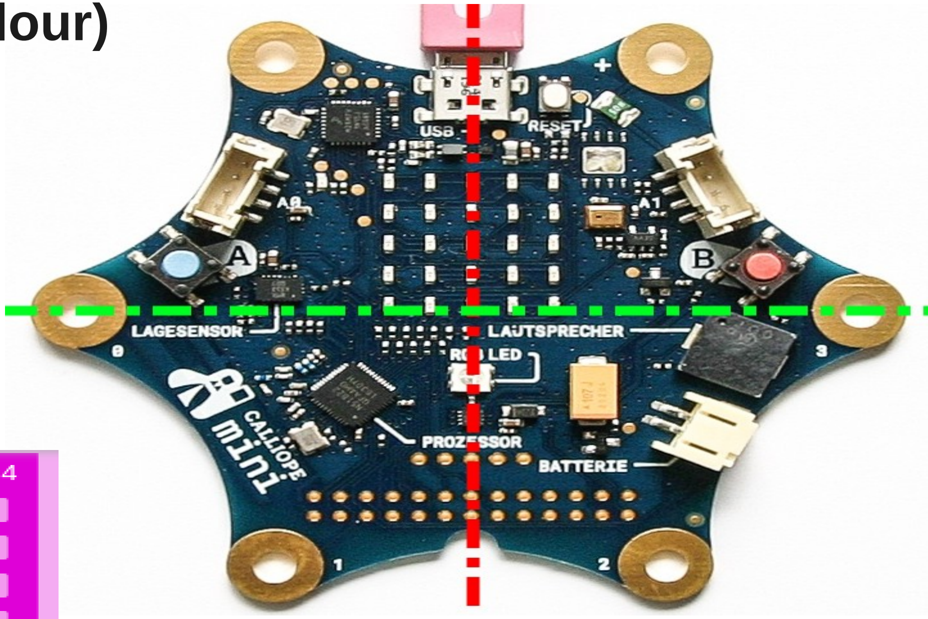
Then all Robots and PC browsers in the class are connect by wifi to your laptop

(Available also for Android)

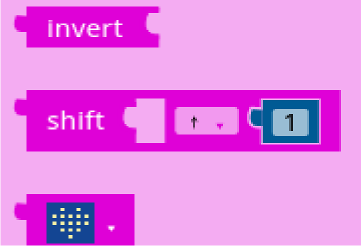
Microcontrollers:
Calliope mini - a lot of sensors

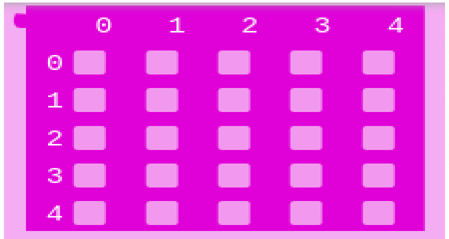
Sensors: buttons, tilt, compass, temperature, light, sound intensity, gyroscope, accelerometer, humidity, ultrasound, external analogue sensors (e.g. colour)

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)

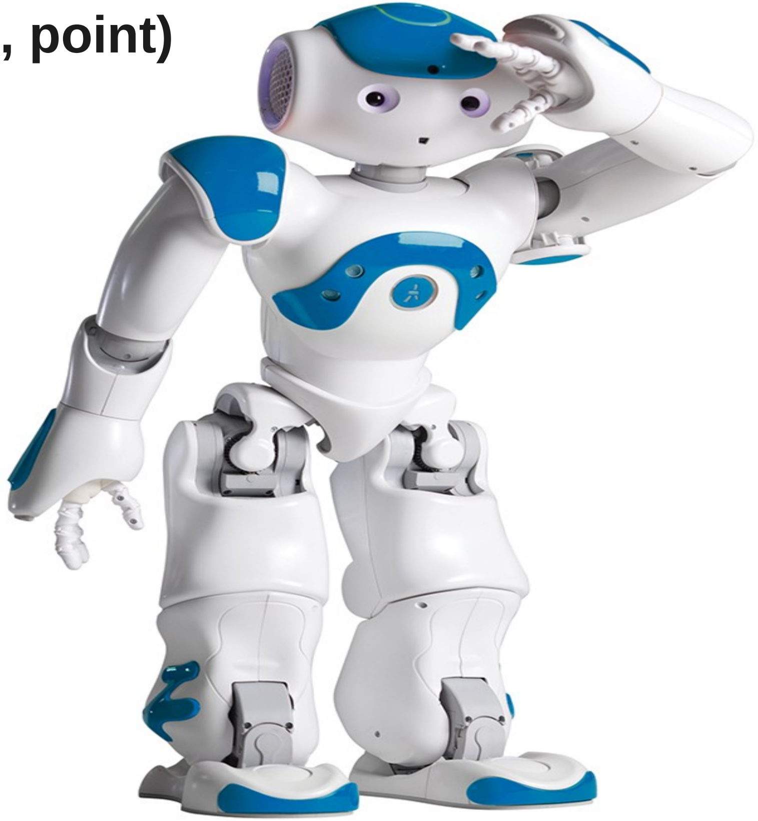
Walk to, hand movements in space, …

Can record a video or picture

Can remember/recognize a face

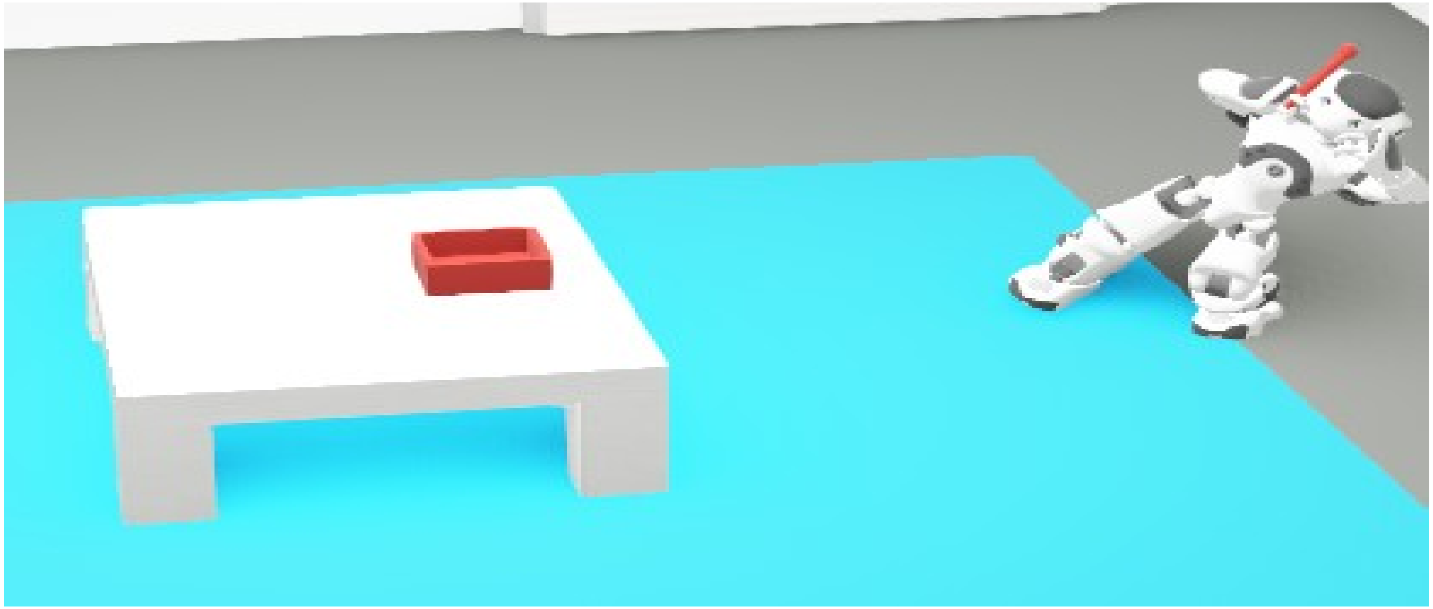
Play sounds, speak (text to speech)

Programmed in Python



3D simulation in browser

E.G. making a Tai chi move



NEW! Neural Networks!!! (for Lego EV3)

NeuralNetwork editor/simulator and trainer

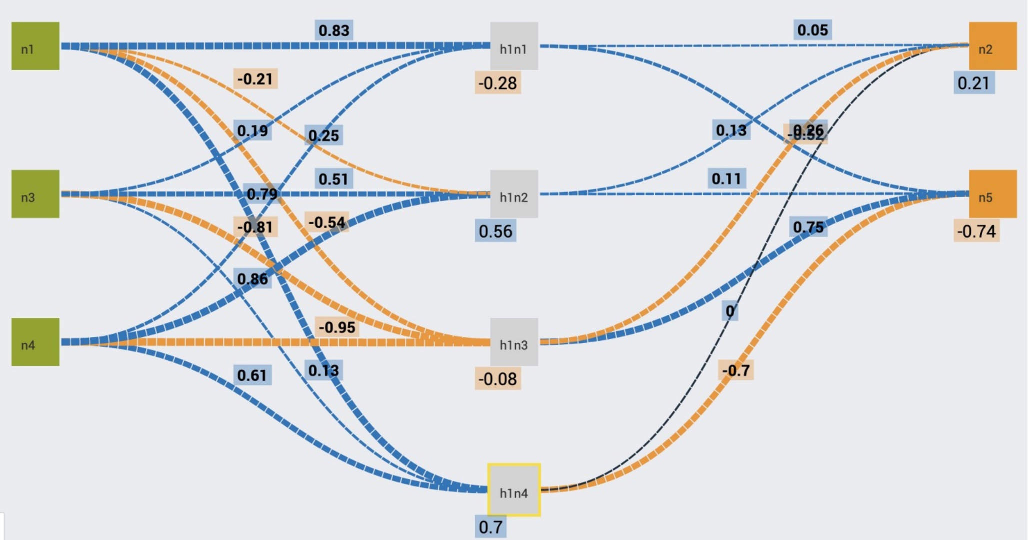
- number of neurons in each level
- activation function of the neurons: Linear, ReLU, Sigmoid, Tanh, Bool

Neural network simulator

- Forward propagation:
  full, by layer, by neuron
- import training data from file

Neural network trainer

- learning rate, epochs
- training: complete, one epoch,
  one line of training data



Demo

Demo