

FLL Programming 101 NXT-G



May 2006
Version 1.0

Legal Stuff

© 2006 INSciTE in agreement with, and permission from FIRST and the LEGO Group. This document is developed by INSciTE and is not an official FLL document from FIRST and the LEGO Group. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit

<http://creativecommons.org/licenses/by-nc-sa/2.0/>

or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

LEGO®, ROBOLAB, and MINDSTORMS™ are trademarks of the LEGO Group used here with special permission. FIRST™ LEGO® League is a trademark owned by FIRST (For Inspiration and Recognition of Science and Technology) and the LEGO Group used here with special permission. INSciTE™ is a trademark of Innovations in Science and Technology Education.

INSciTE

PO Box 41221

Plymouth, MN 55441

www.hightechkids.org

Creative Commons License

- High Tech Kids is committed to making the best possible training material. Since HTK has such a dynamic and talented global community, the best training material and processes, will naturally come from a team effort.
- Professionally, the open source software movement has shown that far flung software developers can cooperate to create robust and widely used software. The open source process is a model High Tech Kids wants to emulate for much of the material we develop. The open source software license is a key enabler in this process. That is why we have chosen to make this work available via a Creative Commons license. Your usage rights are summarized below, but please check the complete license at: <http://creativecommons.org/licenses/by-nc-sa/2.0/>.

Credits

This presentation was developed by Doug Frevert. It is based on the work of Fred Rose. The accompanying labs were originally done in RCX Code by Joel Stone and converted to ROBO LAB by Doug Frevert. A portion of the material is taken from *“Building LEGO Robots for FIRST LEGO League”* by Dean Hystad. Amy Harris defined the 10 programming steps. Eric Engstrom, Jen Reichow, and Ted Cochran reviewed ongoing drafts. Eric taught the first class and helped modify the content accordingly.

Computer Programming 101

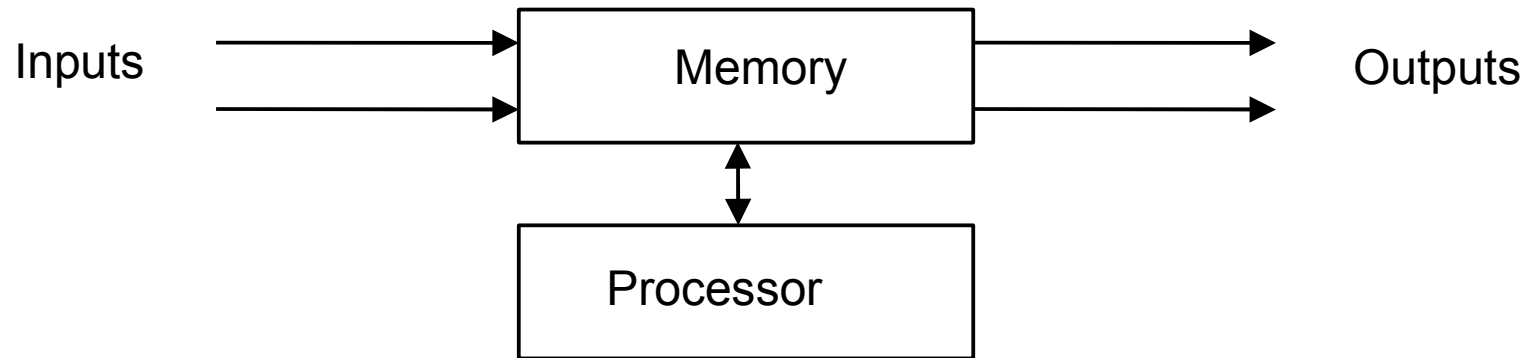
- Objective
 - Develop a basic approach to, and understanding of, programming the NXT
- Structure
 - Theory
 - Examples specific to language
 - Hands-on
- What this class **is**
 - Teach an approach to programming
- What this class **is not**
 - Exhaustive reference on every language command

Class Agenda

- Computer Basics
- The Programming Environment
- Common Blocks
 - Lab #1
- Problem Solving
- Keep It Simple
 - Lab #2
- Sensors
 - Lab #3
- Program Structures
 - Lab #4
- Advanced Topics
- Putting It Altogether

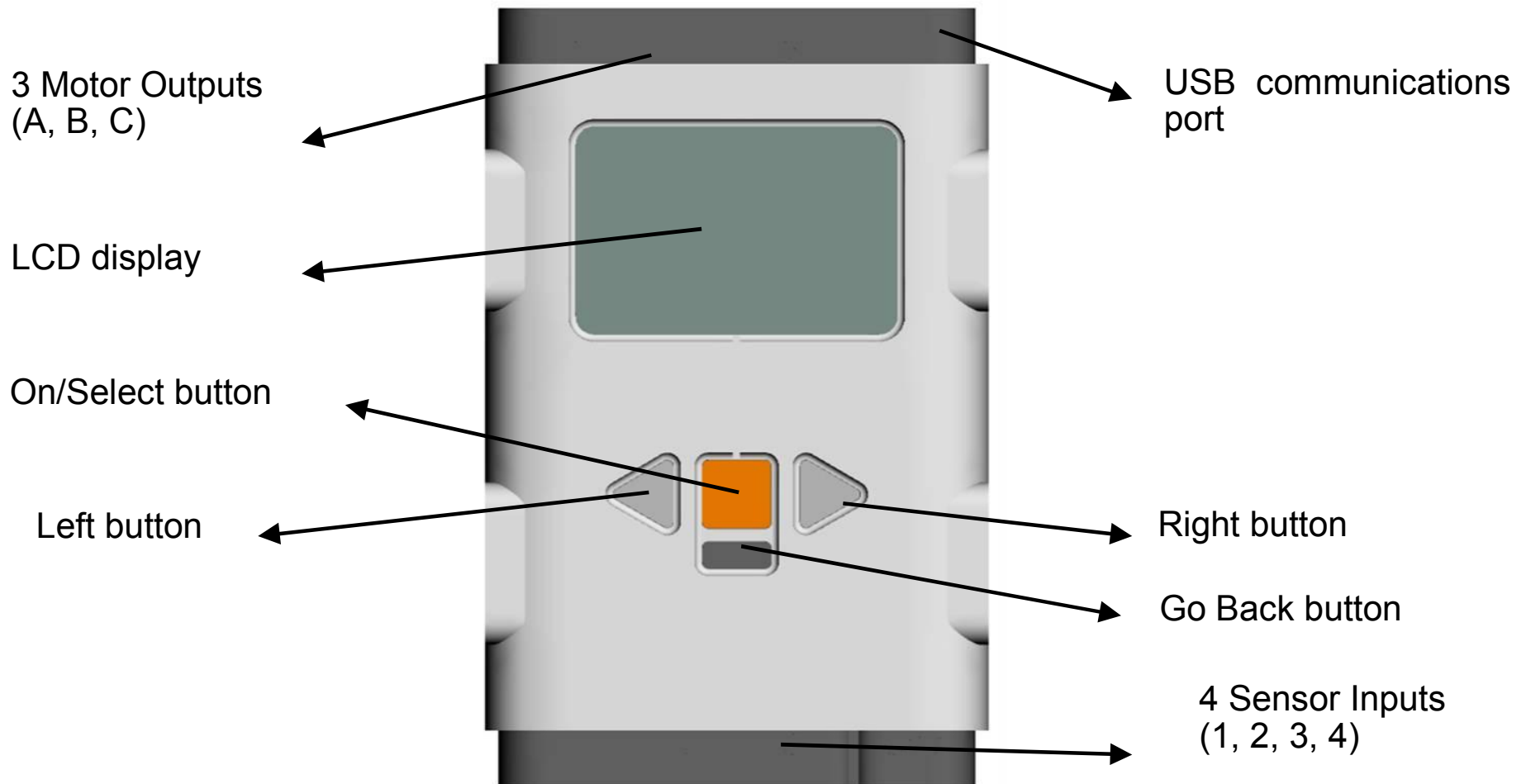
Computer Basics

The Computer (Generic)



- Processor executes commands.
- Memory stores program and data.
- Input devices transfer information from outside world into computer.
- Output devices are vice versa.

NXT

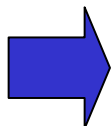
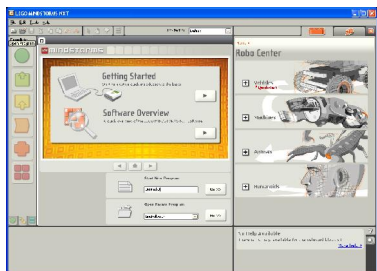


Processor: 32 bit ARM Atmel AT91SAM256 running at 50 Mhz

Memory: 64K Static RAM, 256K Flash

NXT Firmware

Software



compiled
*.rte Files

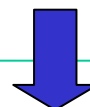
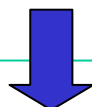
Firmware



Your PC



Download



**Static RAM
(Random Access
Memory)**

*** RAM loses
power, data is
safe!**

**Your Programs
Firmware**

**ROM (read only memory)
Processor (ARM)**

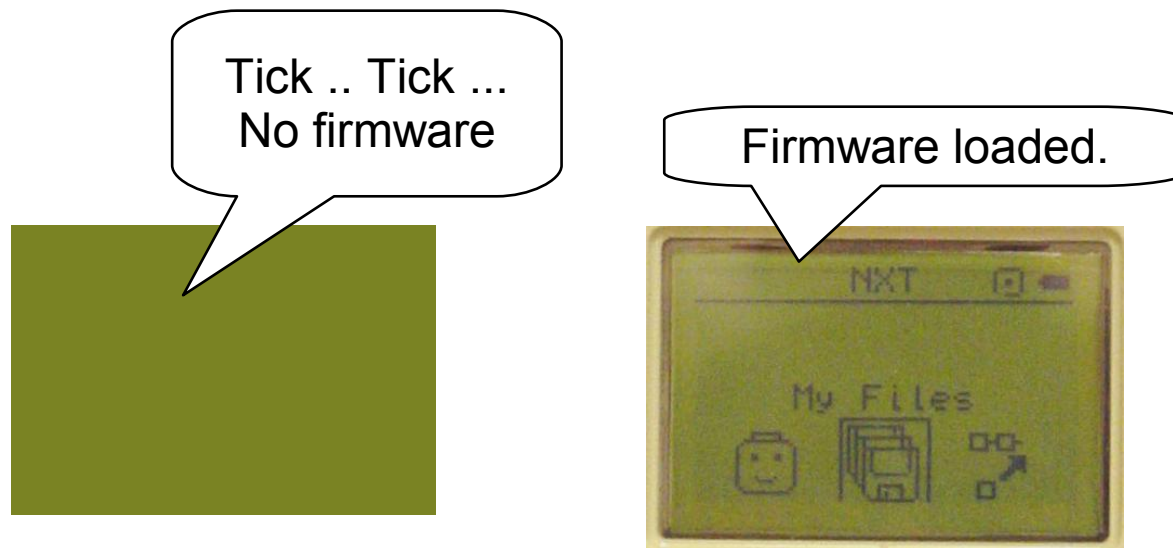
Your NXT



**For FLL purposes, think of firmware as the operating system
(like Windows XP or Mac OSX) for the NXT**

Firmware Loaded?

- Firmware must be downloaded to your NXT so that the NXT can understand your programs.
- Only required to be loaded
 - To install a new firmware release,
 - NXT lost it's firmware for some reason, or
 - NXT starts behaving badly.



Computer Programs

- Model a real or mental process
 - Intricate in detail
 - Only partially understood
 - Rarely modeled to our satisfaction
 - Thus, **programs continually evolve**
-
- The computer is a harsh taskmaster, its programs must be correct and things must be accurate in every detail.

From: Structure and Interpretation of Computer Programs, Abelson, Sussman, and Sussman,

Writing a Computer Program

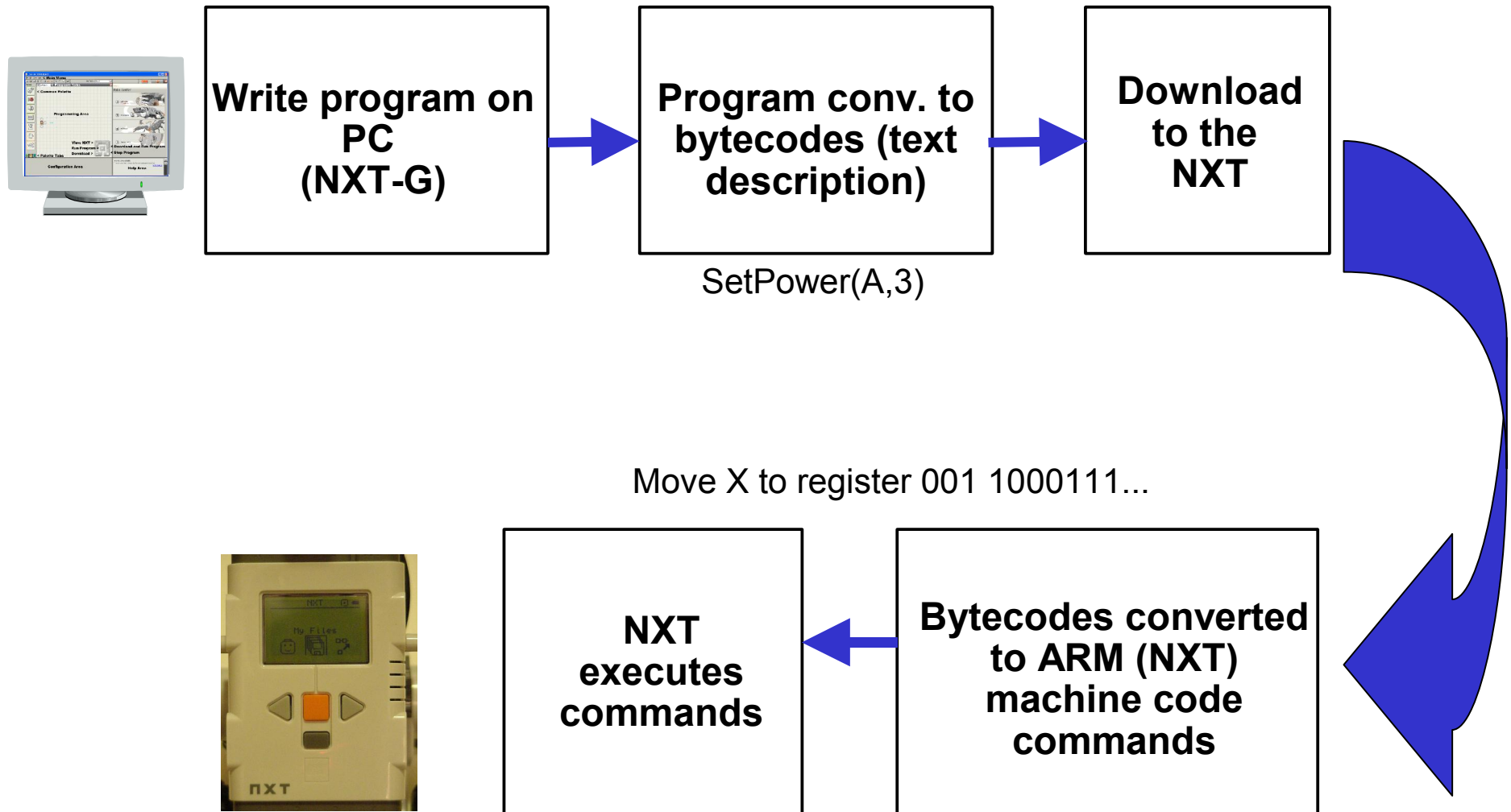
- Specify the task
 - Inputs to be supplied
 - Outputs to be produced
- Devise an algorithm
- Express that algorithm in a computer language

From: Introduction to Pascal, Welsh and Elder

Language Choices

- **NXT**
 - **NXT-G (NXT Graphical programming)**
 - Comes with the NXT version of LEGO Mindstorms
 - PC or a MAC
 - Has MyBlocks like RIS
 - Built on top of LabVIEW like RoboLab
 - **RoboLab 2.9**
- **RCX**
 - **RoboLab**
 - Runs on MAC or PC.
 - **RIS**
 - Only runs on a PC
- **High School FLL: no language restrictions.**

Running a computer program (NXT)



Tips and Tricks (1)

- The NXT has memory to store many programs
 - NXT automatically powers down.

- Bluetooth Communications



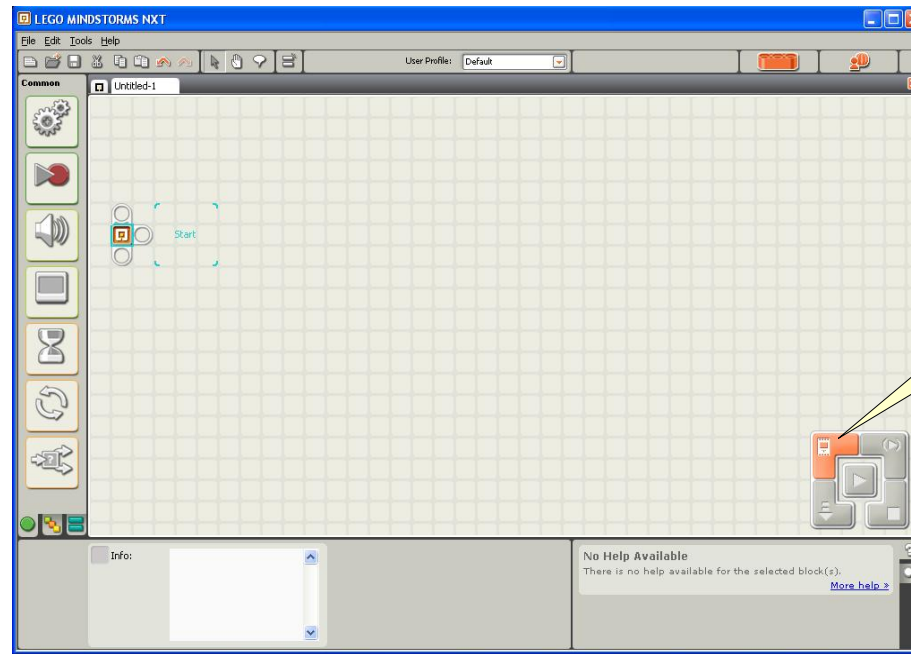
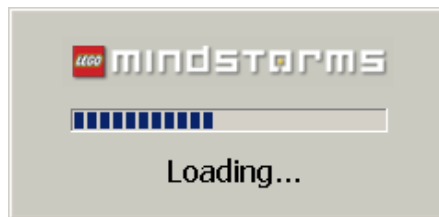
- The NXT has bluetooth communications. If enabled, PCs, NXTs and other bluetooth devices can talk to each other.
 - Disable bluetooth during competition.

Tips and Tricks (2)

- Direction of connecting wires
 - NXT wire connectors only fit one way. Can not be rotated.
- Batteries
 - AA
 - No worry about losing firmware.
 - NiMH rechargeable batteries work. NiCads don't.
 - Lithium rechargeables come with the FLL Mindstorm kits.
 - Avoid stalling the motors, it drains batteries.

NXT-G

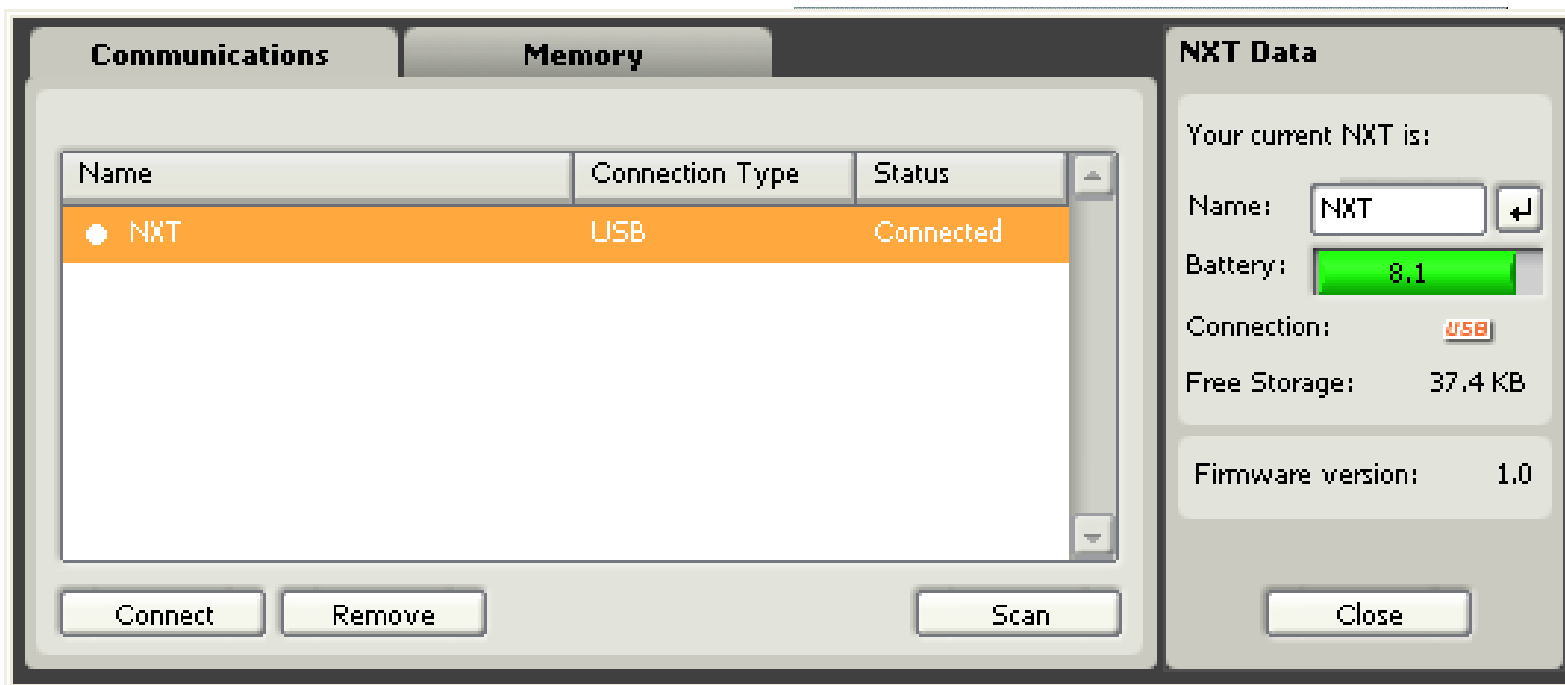
The Programming Environment



Connect your laptop/PC to your NXT and click here for the next slide.

NXT-G to NXT

Communications

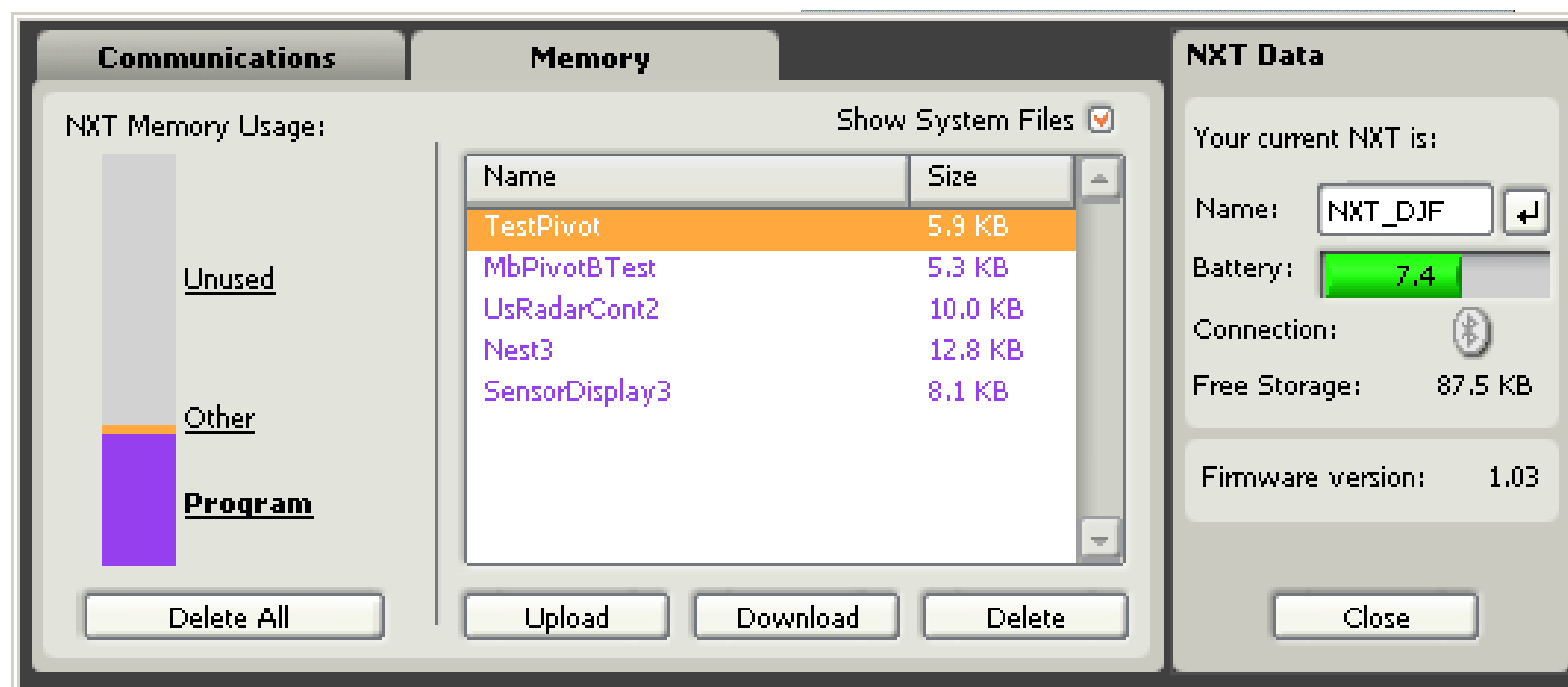


When connected by USB cable or Bluetooth

- Give your NXT a new name
- Check Battery voltage
- View available memory (in KiloBytes)
- Firmware version

NXT-G to NXT

Memory



Select, then delete Programs, Sounds, Graphics, and Unused files.

Can free up to 130Kb of Free Storage on the NXT.

NXT-G Opening Workspace

The screenshot shows the LEGO MINDSTORMS NXT-G software interface. The window title is "LEGO MINDSTORMS NXT". The menu bar includes "File", "Edit", "Tools", and "Help". The toolbar contains various icons for file operations and editing. The "User Profile" dropdown is set to "Default".

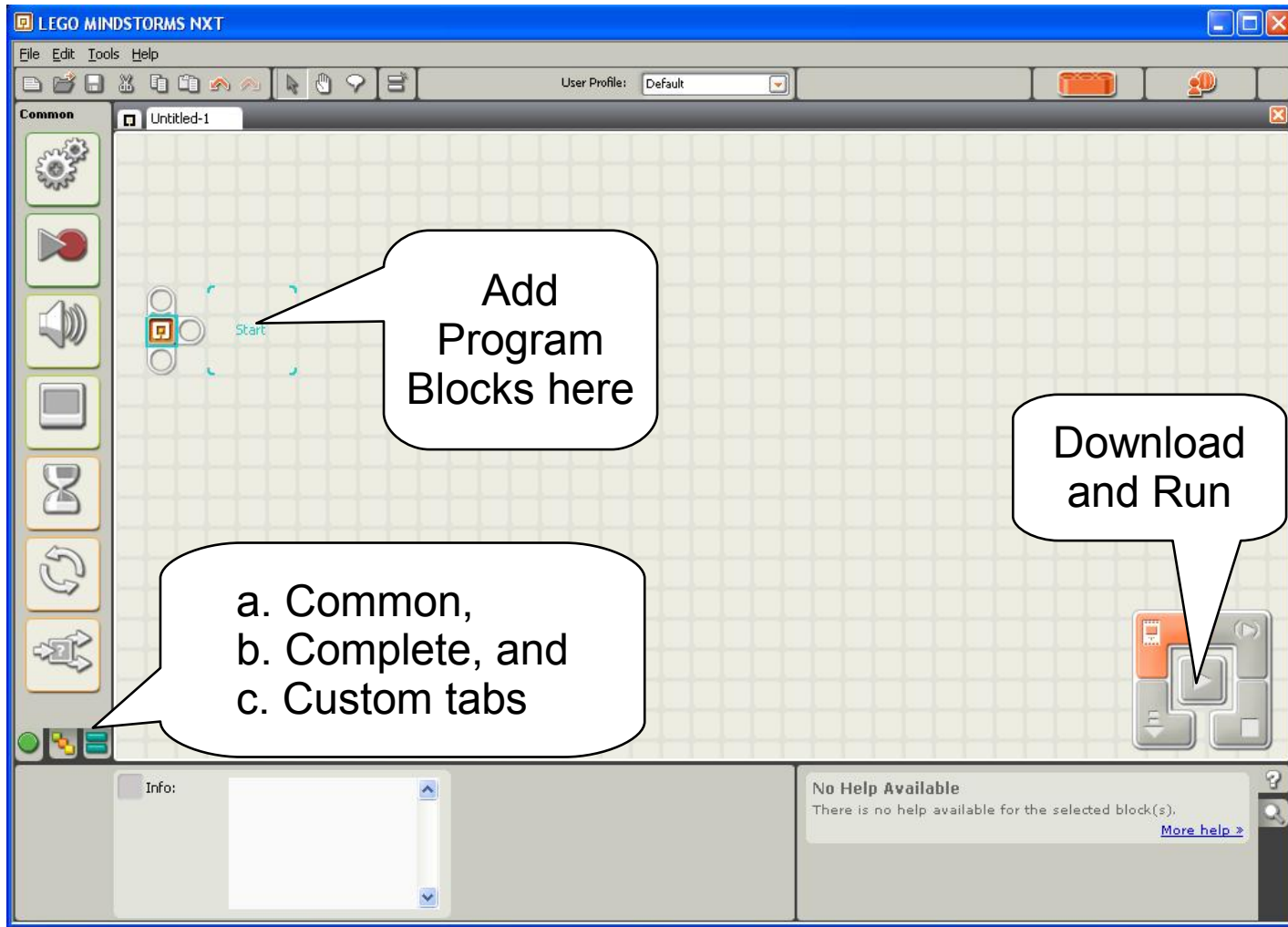
The main workspace is divided into several panels:

- Left Panel:** Contains a vertical stack of icons for "Program Blocks" (green circle, green house, yellow house, orange house, red cross, red plus) and a "Select a Program Name" dialog box.
- Center Panel:** Displays a "Getting Started" screen with a "Software Overview" section. Below this, there are buttons for "Start New Program" (with a text field containing "Untitled-3") and "Open Recent Program" (with a dropdown menu showing "LineFollower").
- Right Panel:** Titled "Robo Center", it lists pre-built robots and programs: "Vehicles * Quickstart", "Machines", "Animals", and "Humanoids".
- Bottom Panel:** A "Block Settings" panel, currently showing "No Help Available" for the selected block(s).

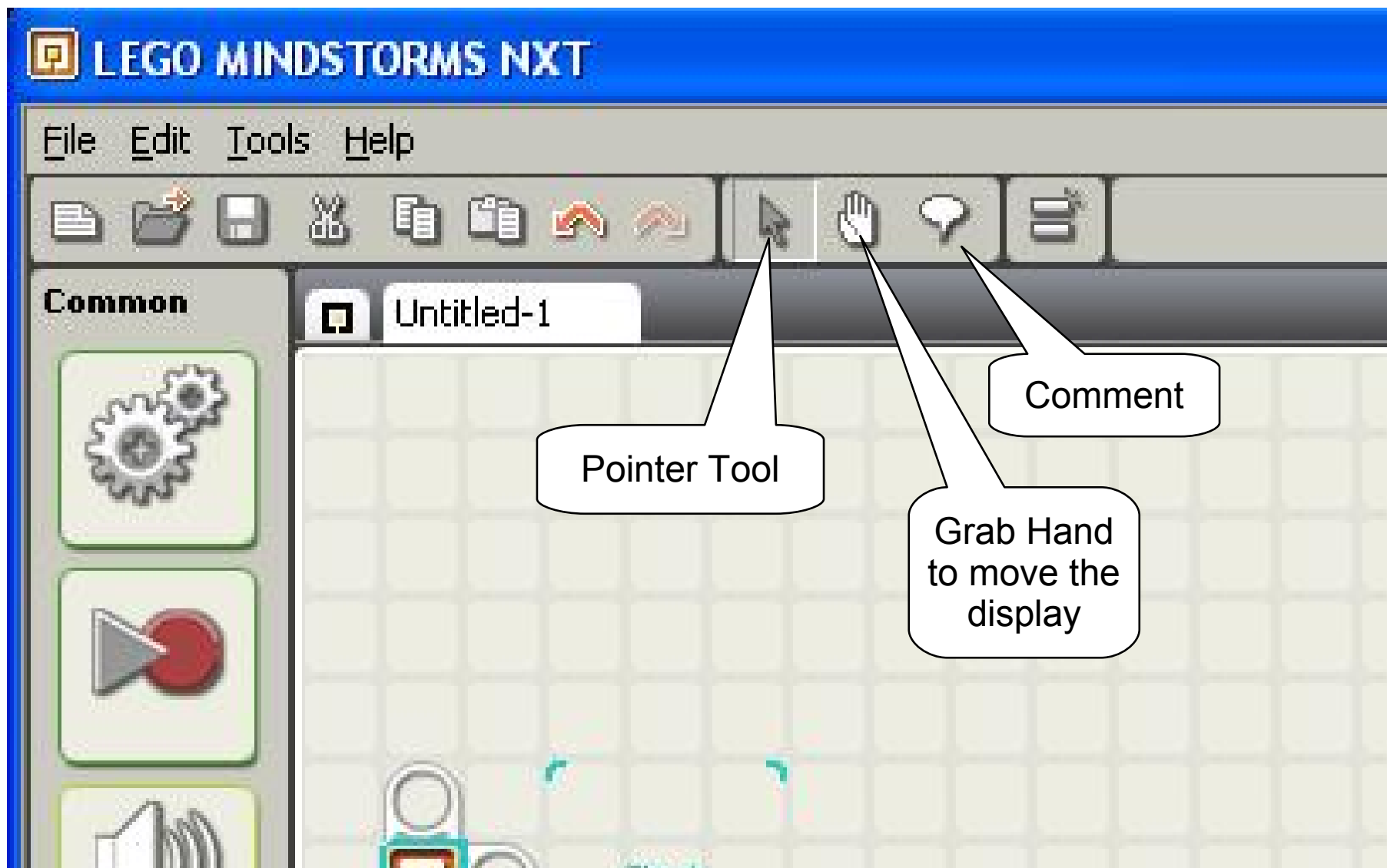
Callout boxes provide additional information:

- Profiles:** Points to the "User Profile" dropdown menu.
- Program Blocks:** Points to the vertical stack of icons on the left.
- Select a Program Name:** Points to the dialog box on the left.
- Block Settings:** Points to the bottom panel.
- Pre-built Robots, Programs, and Challenges:** Points to the "Robo Center" panel on the right.
- Help and Zoom Panel:** Points to the bottom right corner of the interface.

NXT-G Work Space



NXT-G Work Space



Common Blocks

Common Blocks

- Common blocks are full featured actions
 - Like English statements
 - Move
 - Wait for an action
 - Display a value
 - With many modifiers
 - Move direction, steering, distance, motors used . . .
 - Wait for light sensor, light threshold, sensor port, . . .

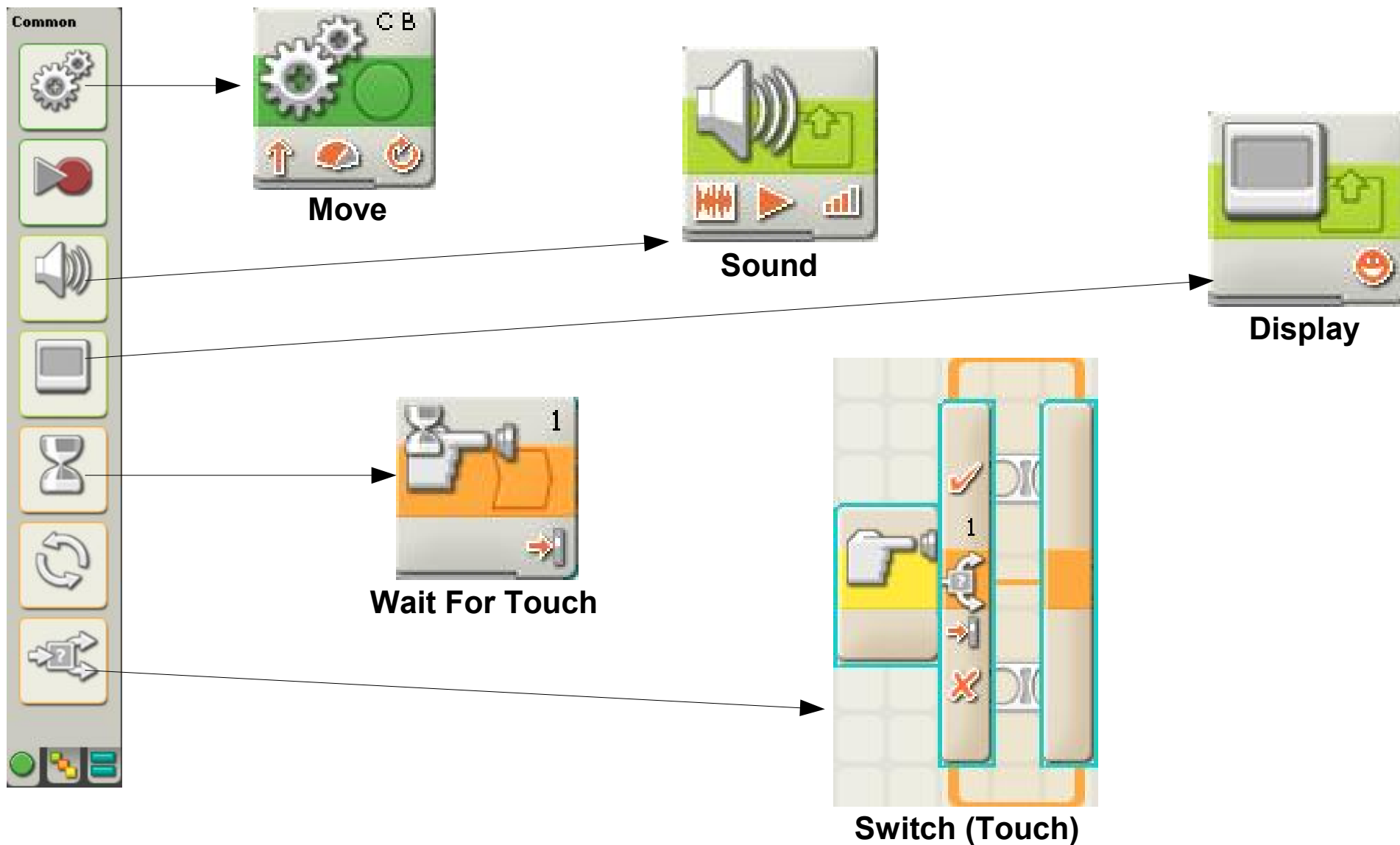


Move Block



Move Block Settings

Common Blocks



Adding a Block to a Program

Click a Block

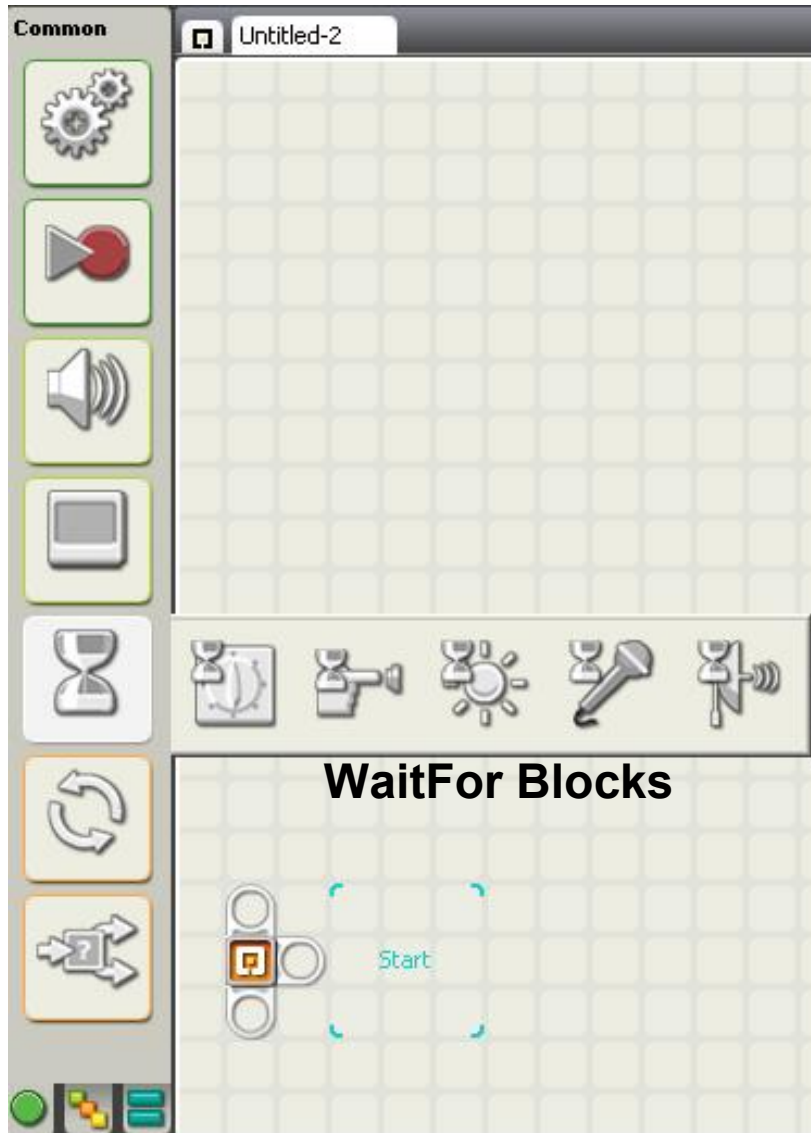
Drag it here

Release when the white position preview marks appear.

Change the settings

- Click on a Block
- Move cursor onto Program and drop it into place. NXT-G will make room.
- Change settings

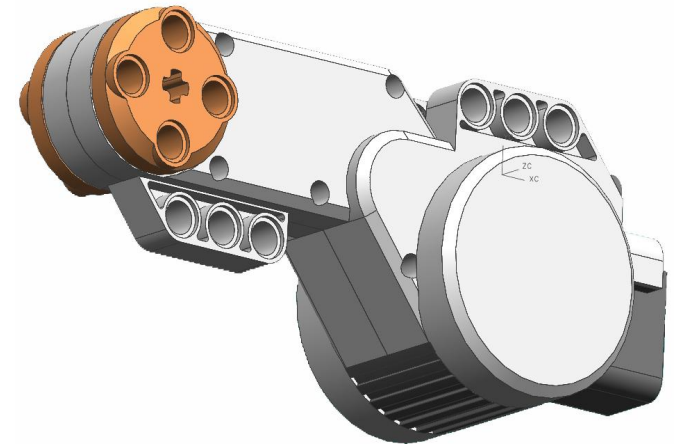
WaitFor Blocks



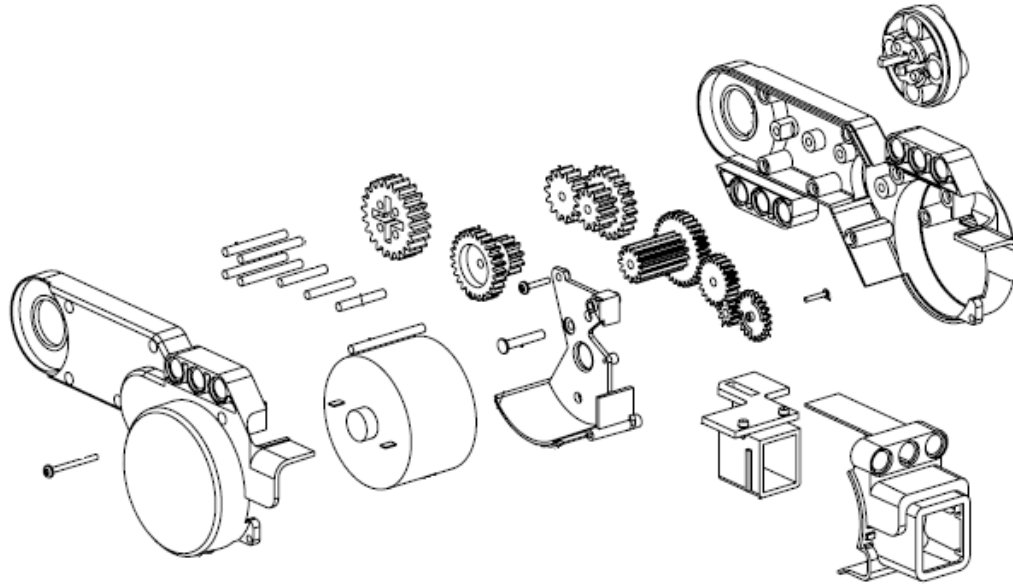
- Click on the hourglass.
- Click on a WaitFor Block
- Time
- Touch
- Light
- Sound
- Distance (Ultrasonic sensor)
- Others (see settings)

Motors

- 9 volt geared motor
 - Making the motors turn is the **output** of your program. It makes your creation a robot!
 - Without load, motor shaft turns at about 150 rpm.
 - Servo sensitive to 1 degree.
 - With a typical robot, 3-4 hours on a set of batteries.
- FLL allows up to 3 motors.



Motor Details



- Motor can be set to different power settings
 - Power levels 0-100
 - Power is adjusted by Pulse Width Modulation
- Turning the power setting up higher essentially makes the shaft turn faster.

Using the Move Block

The image shows a screenshot of a programming environment's 'Move' block. The block is a green rectangle with a gear icon and a 'C B' label. It has several settings and callouts:

- Ports A, B, and/or C:** A callout pointing to the 'Port' section, which has checkboxes for A, B, and C. B and C are checked.
- Forward, Backward or Stationary:** A callout pointing to the 'Direction' section, which has three radio buttons: Forward (selected), Backward, and Stationary.
- Power 0-100%:** A callout pointing to the 'Power' slider, which is set to 75.
- Duration Time, Degrees, Rotations, Forever:** A callout pointing to the 'Duration' section, which has a text input field set to '1' and a 'Rotations' dropdown menu.
- Steering Spin, Pivot, Arc, Straight:** A callout pointing to the 'Steering' section, which has a dropdown menu set to 'C' and a 'Steering' slider.
- Brake or Coast:** A callout pointing to the 'Next Action' section, which has two radio buttons: 'Brake' (selected) and 'Coast'.

Lab One

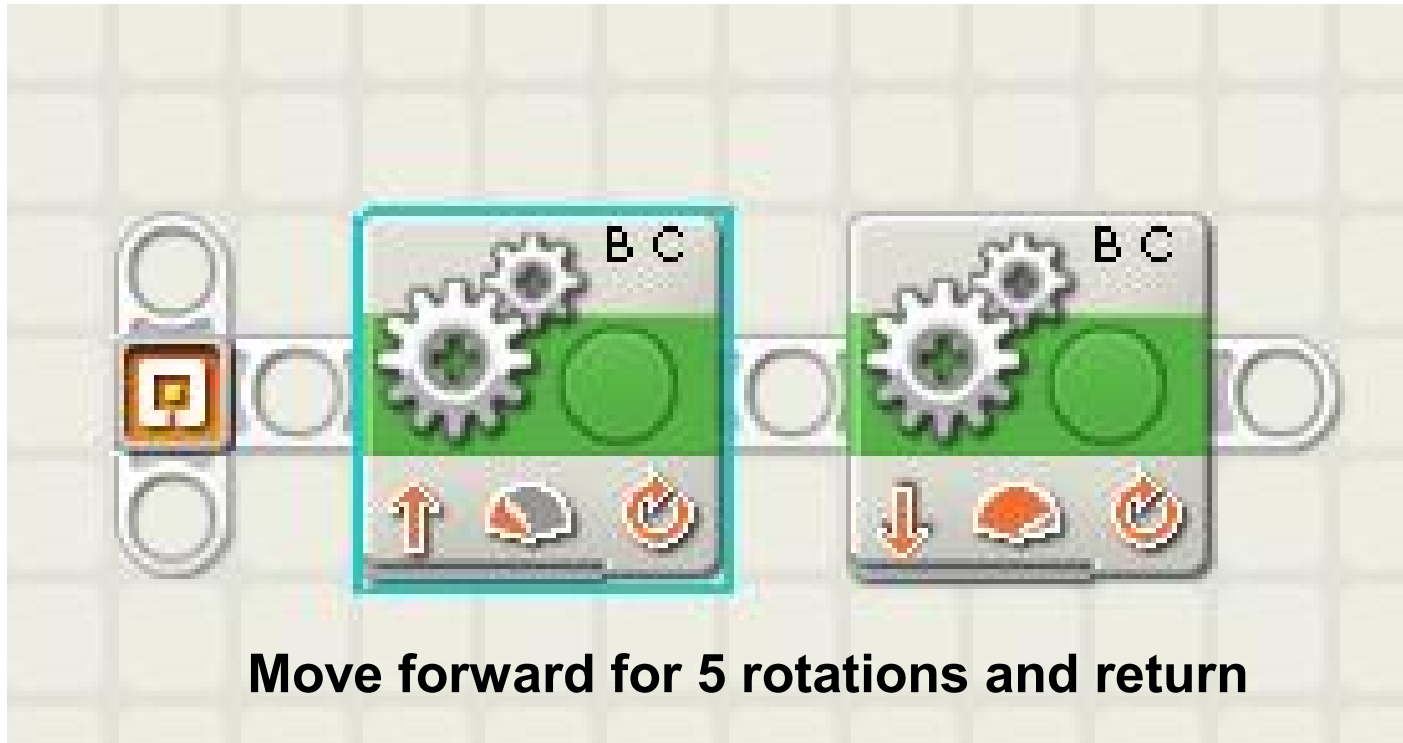
Task:

Move forward for 5 rotations and return

Then try:

Move forward for 5 rotations, turn right 90°

Lab 1 An Answer



Problem Solving

Generic Problem Solving Process

- Define the problem
 - Brainstorm solutions
 - Evaluate solutions Pick one
 - Try (implement) it
 - Evaluate results
-
- Express the solution as an algorithm, then convert it into a computer program.

What's an Algorithm?

- An algorithm (pronounced AL-go-rith-um) is a procedure for solving a problem. The word derives from Mohammed ibn-Musa **Al-Khowarizmi**, a mathematician of the royal court in Baghdad. He lived from about 780 to 850. Al-Khowarizmi's work is the likely source for the word **algebra** as well.
- A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small **procedure that solves a recurrent problem.**

From: <http://whatis.techtarget.com/>

An Algorithm is like a Recipe

Recipe for French Toast

- 8 slices of bread
- 2 eggs
- 1 cup milk
- 1/4 cup flour
- fat or butter
- powdered sugar

Mix eggs, milk and flour and pass through a strainer.

Dip slices of bread into the mixture and drop into a buttered frying pan. Fry both sides.

Before serving, sprinkle with powdered sugar.

How can we make this more “algorithm-like”?

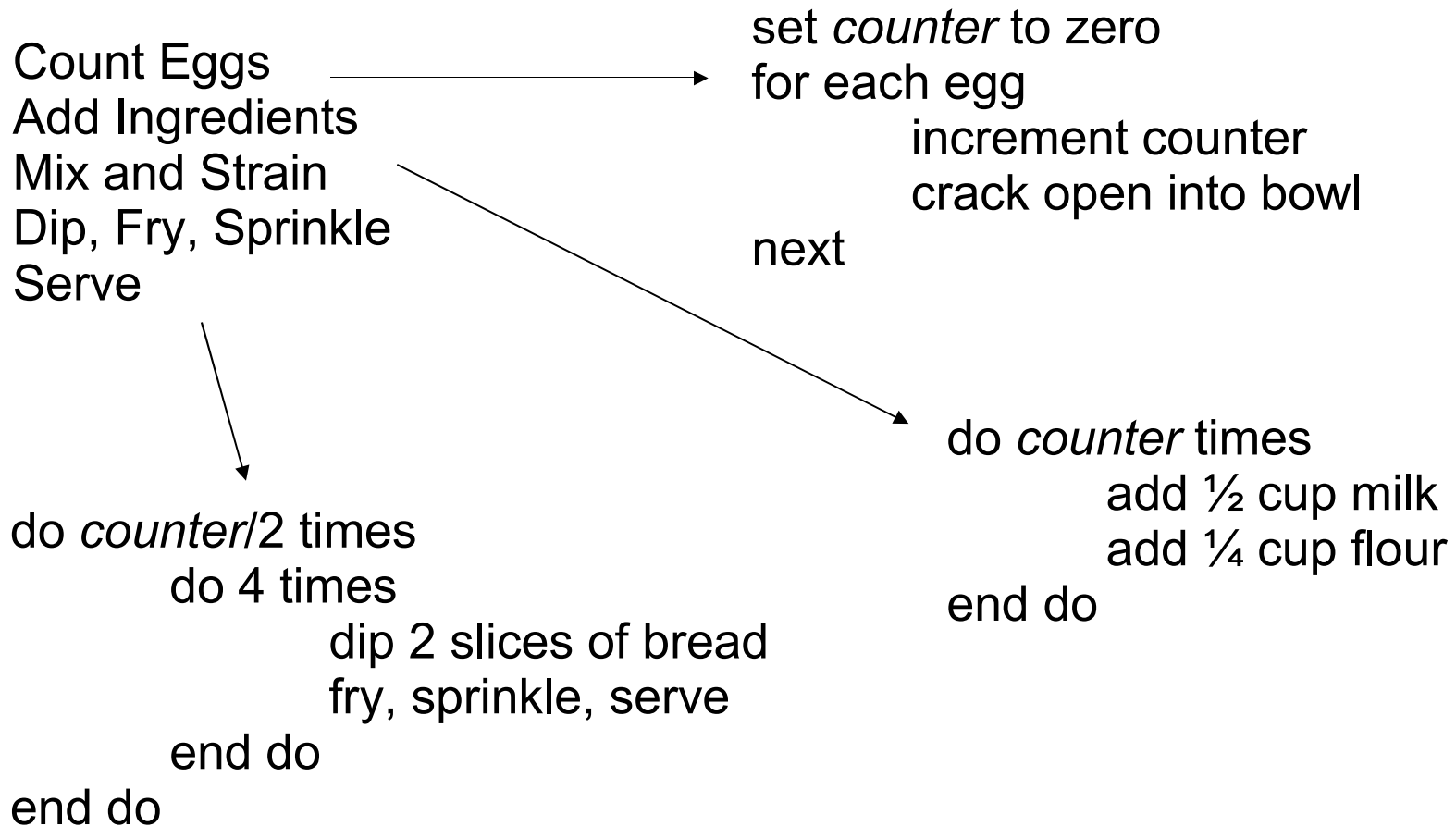
Count eggs, mix eggs, for every 2 eggs add 1 cup milk and 1/4 cup flour, and pass through a strainer. For every 2 eggs, dip 8 slices of bread....

Ways to Express Algorithms

- In the real programming world there are many ways to do this
- In the FLL world, probably the two best ways:
 - Draw block diagrams
 - Literally act it out
- Always talk it out and test it using a team member to walk through it acting like the robot.
 - Keep actions at low level
 - Go Forward 3 steps
 - Stop
 - Motor B forward, C backward

Pseudocode

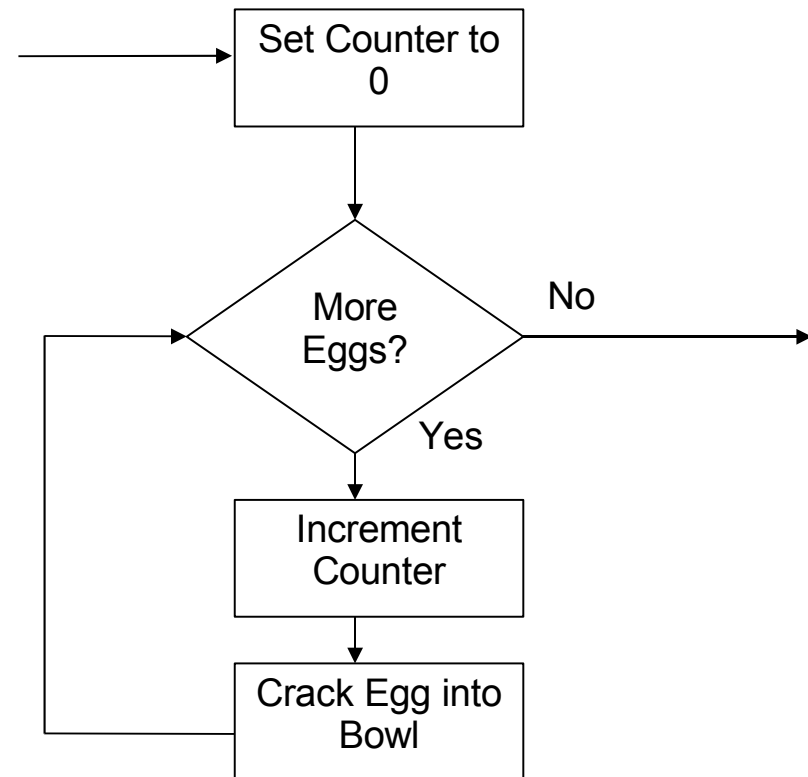
Pseudocode: Not software code, not English, but somewhere in between.



Flowcharts

Flowcharts: A graphic representation of logic.
Convert from pseudocode.
One step closer to software.

set counter to zero
for each egg
 increment counter
 crack into bowl
next



FLL Ace Programmer in 5 Steps

- Map the generic problem solving process to programming
- Create a map of where the robot goes and what it does
- Write what the program should do (your algorithm).
- Code it
- Test, and fix, little pieces at a time

We'll be adding to this process as the class progresses

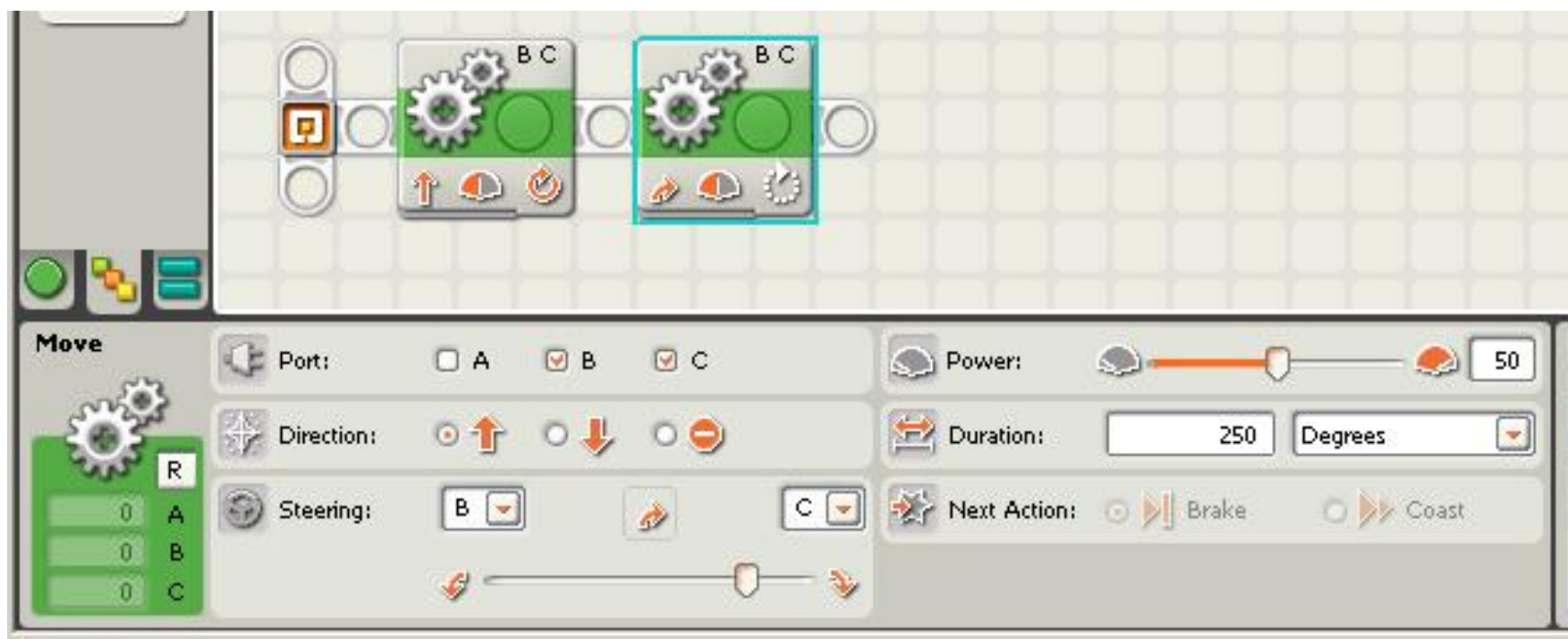
Example Algorithm

Make robot go forward 5 rotations and then turn right 90°

- Set direction and power of motors
- Stop motors, turn right.
 - Turn right by reversing right-side motor
 - Turn motors for ? rotations.
- Stop motors

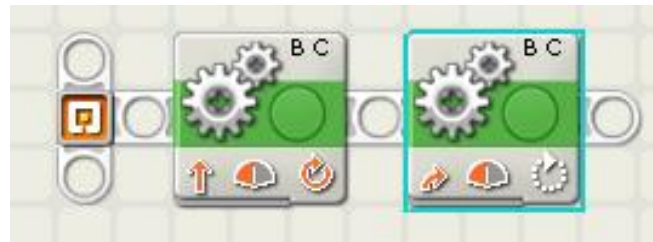
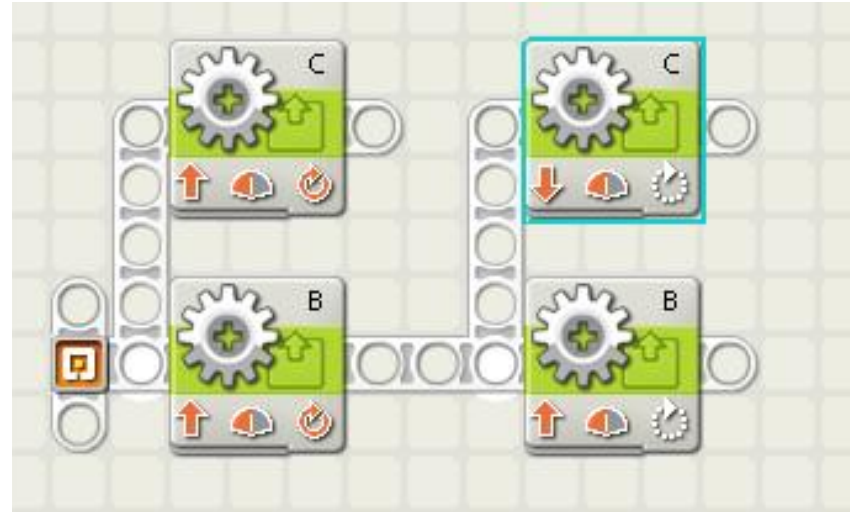
Conversion to a Program

- Forward: Motors B and C forward
- Spin: Motor B forward, C backward,
Power Level 50%
250 degrees
motors brake.



Optimizing Code

- Which is faster?
more reliable?
best?



- Use the one that makes sense to you, the programmer.

Debugging and Analysis

- Literally walk through it
- Ask lots of questions
 - What ifs
- Do little pieces at a time
 - For example, get the robot to where it needs to be first, then work on getting it to do something
- Reuse pieces that work
 - For example, you know how to turn 90°
- Feel confident in your algorithm before starting to code it.

Common Situations

- Not sure which solution is better
 - Try them both, or at least the primary element of each
 - Which is easiest for your team to do?
- Can't think of all the steps needed for the algorithm
 - Program and test the steps you understand.

Keep It Simple Strategies

KISS #1: Subroutines
#2: Comments

KISS #1: Subroutines



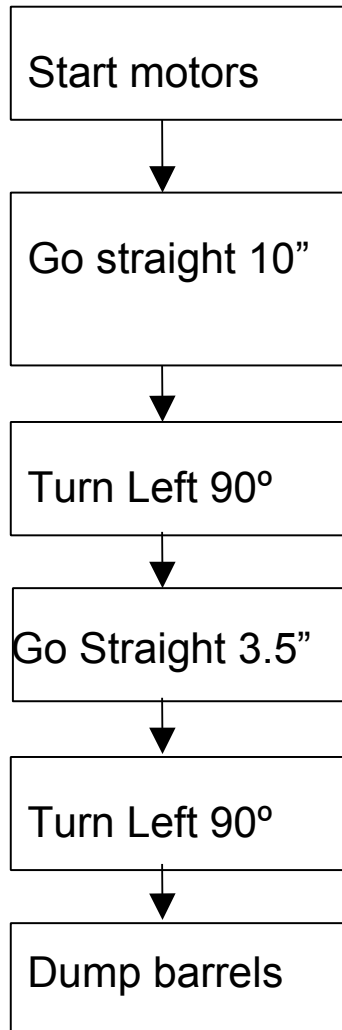
- Wrap a complicated process into a neat and tidy package.
- Once wrapped, just worry about the package.
- In NXT-G, Subroutines are MyBlocks
 - Select from the Custom Tab

Subroutines: When to Use

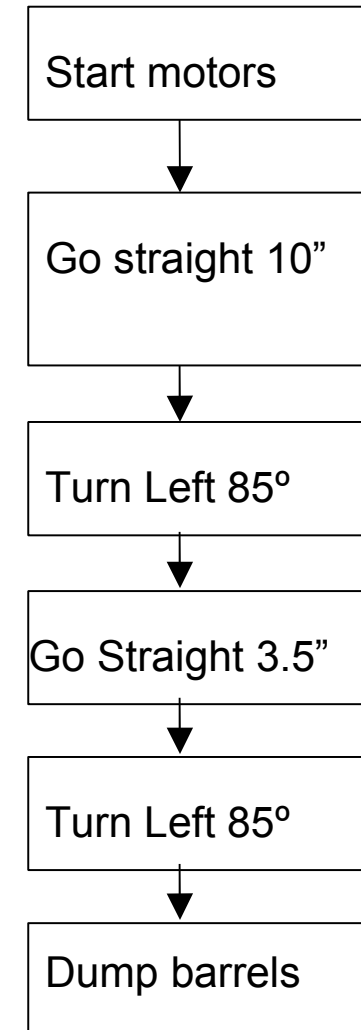
- To do the same thing from different places.
 - Reuse: put the common code in one place.
- To divide a task into meaningful pieces.
 - Modules
- To hide complex details.
- Real world example: Clock
 - User tasks are divided into meaningful pieces: Time, Set Time, Set Alarm, Turn alarm on/off
 - Complicated parts are hidden inside.

Modular Advantages

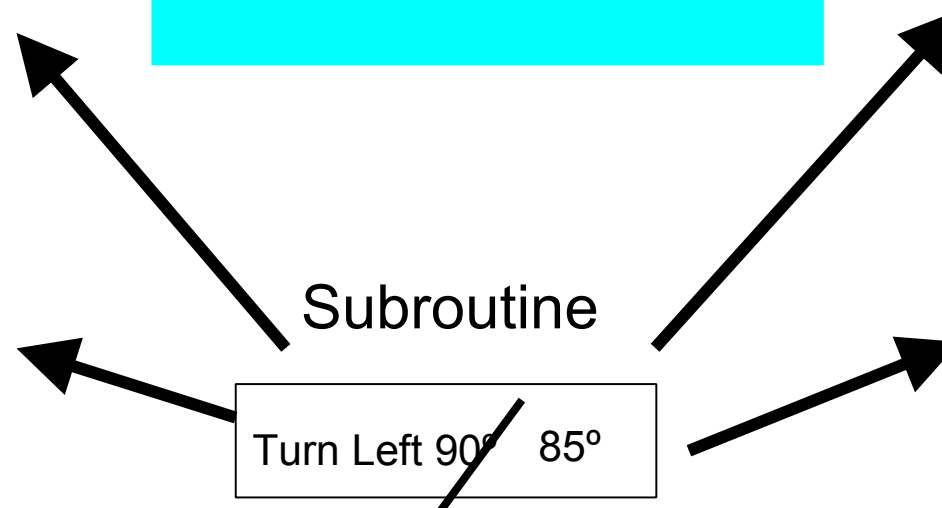
Algorithm - Mission 1



Algorithm Updated- Mission 1



If in the process of testing your program, you realize it is an 85° turn, you only have to change your program in one place, in your subroutine.



MyBlock Names

- Useful and informative
 - ClearSoccerField *not* Csf_amy_3a
 - 12 characters visible on a MyBlock
15 characters visible on the NXT
- Suggest using “action + to + target”:
 - Fwd2Wall or ForwardToWall or Forward_To_Wall
 - FwdDist
 - TurnRight
- Name the task accomplished, not how it was done.
 - FollowLine *not* FollowLine1LightSensor



MyBlock Creation

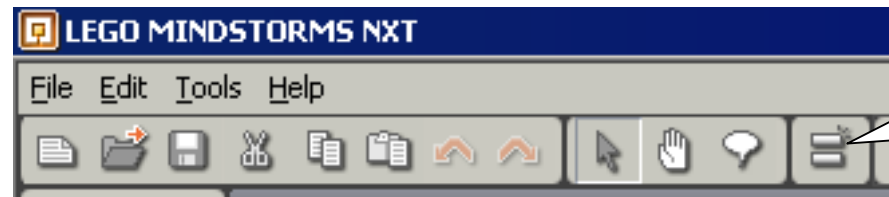


Hide all these blocks inside one MyBlock.



MyBlock Creation 1

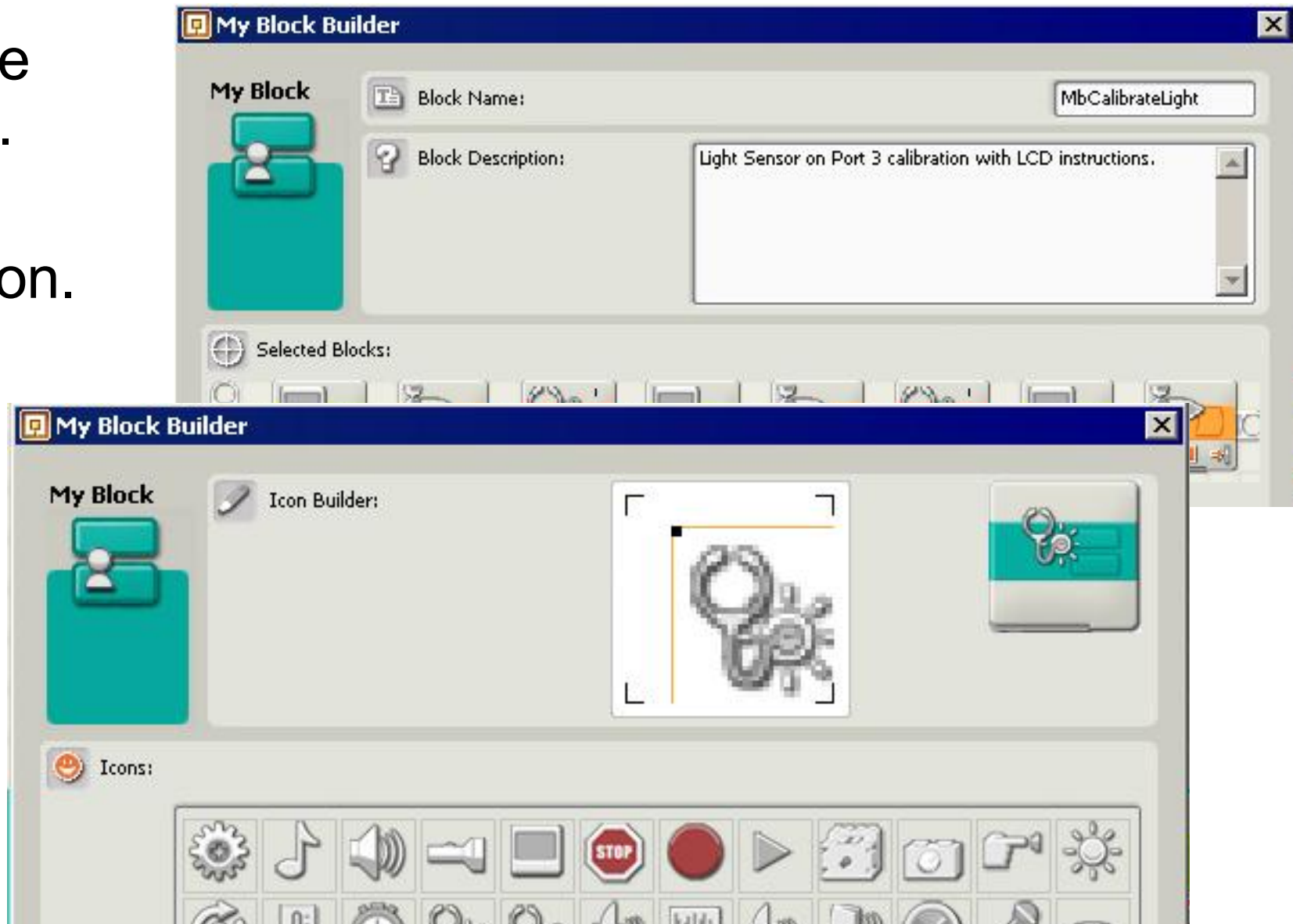
- Start with working code.
- Select blocks to include.
- Click the Create My Block button.



Create My Block button

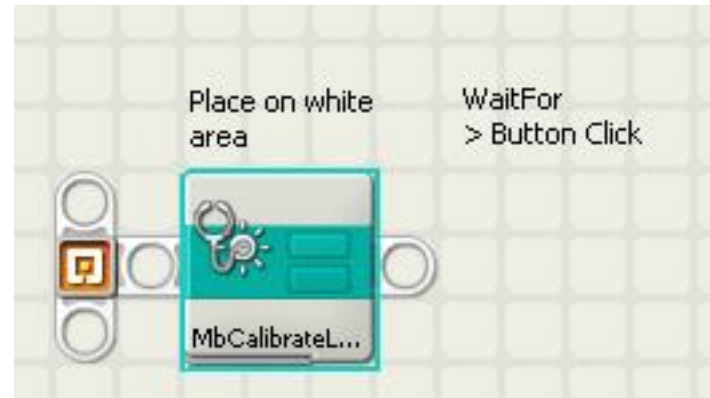
MyBlock Creation 2

- Name the MyBlock.
- Give it a description.
- Add an Icon.



MyBlock Creation 3

- MyBlock replaces the selected Blocks!
- To add the new MyBlock to a program, select it from the Custom Tab.

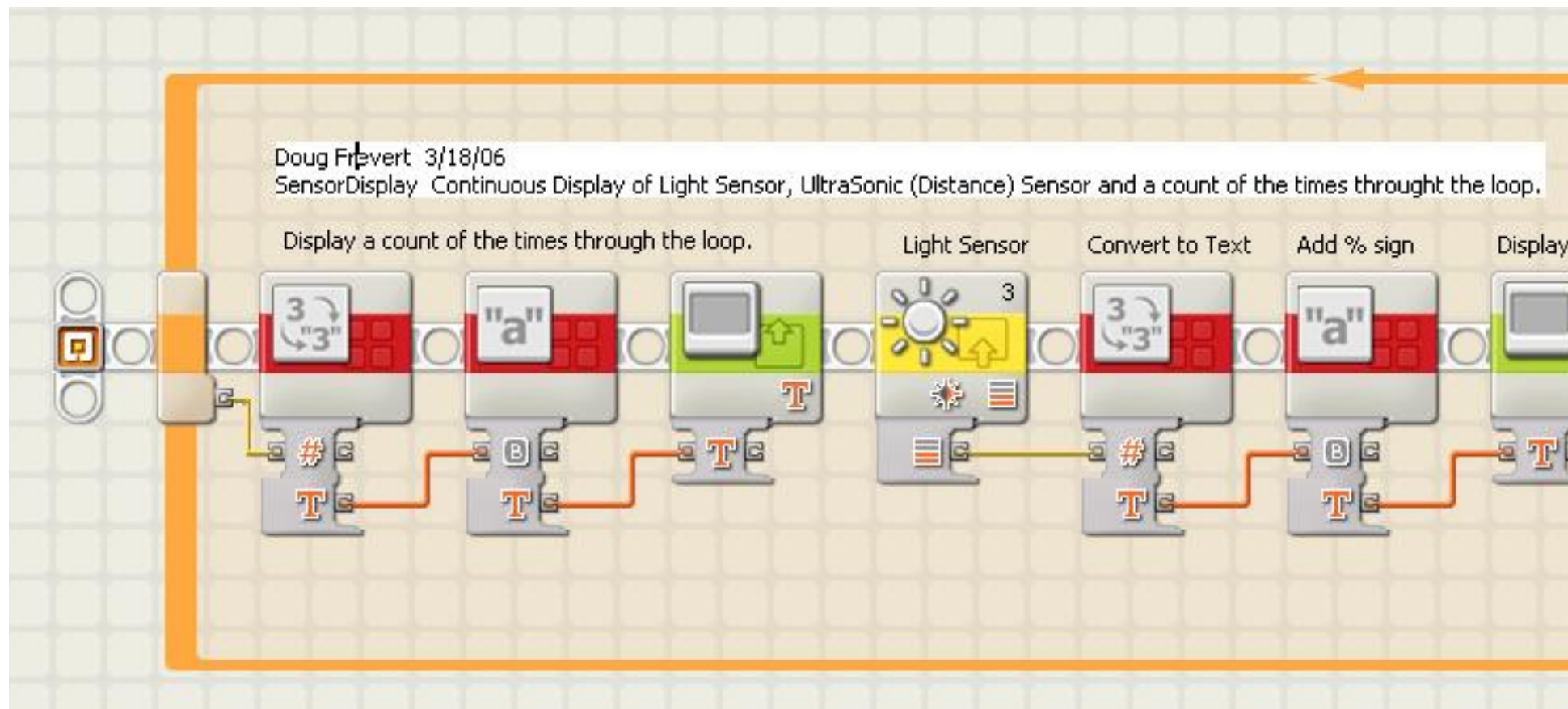


KISS #2: Comments

- Comments explain the program to other programmers.
- **Very** important. Programmers forget.
- In a team process like FLL, comments are especially important as more than one person will be working on the program.



Comment Use



- Add things like who made changes, when, how to use, assumptions, expected results, etc.

Suggestions

- Make sure your code works before making it into a MyBlock.
- Use subroutines to
 - hide complexity,
 - provide structure, and
 - remove duplication.
- Use comments liberally.

Lab Two

Task:

**Make the program from Lab One
(Move forward 5 rotations and turn right 90 degrees)
into a MyBlock**

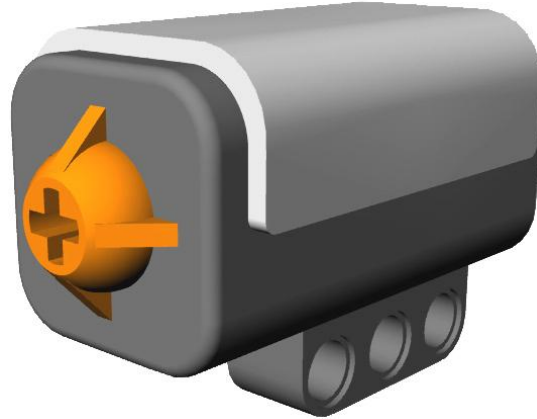
Data Input

Sensors

Sensors

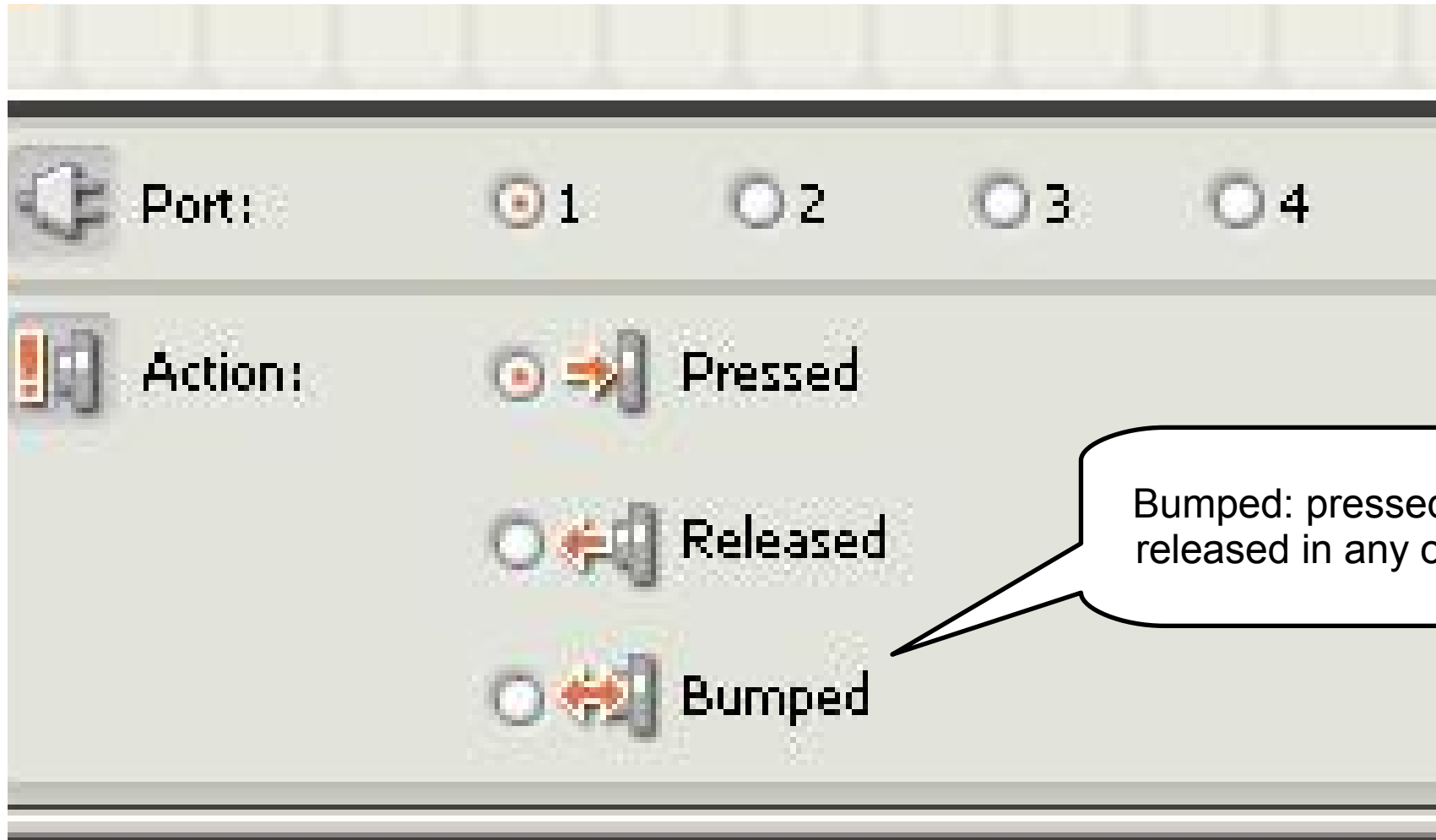
- Allow your robot to detect the real world.
 - Touch
 - Has your robot made contact with something?
 - Light
 - Is the surface light or dark?
 - Sound (Microphone)
 - Ultrasound (Distance)
 - Rotation
 - Embedded in the motors
 - Time
 - Internal sensor, keeps track of time
 - Battery Voltage

Sensor #1: Touch



- To detect touching or bumping into something.
- Good for detecting robot arm movements. The sensor activates when the arm moves far enough to push in the touch sensor.

Pressed, Released, Bumped



Touch Sensor WaitFor Block

Robot waits until sensor responds

Pressed, released, or bumped

Control: Sensor

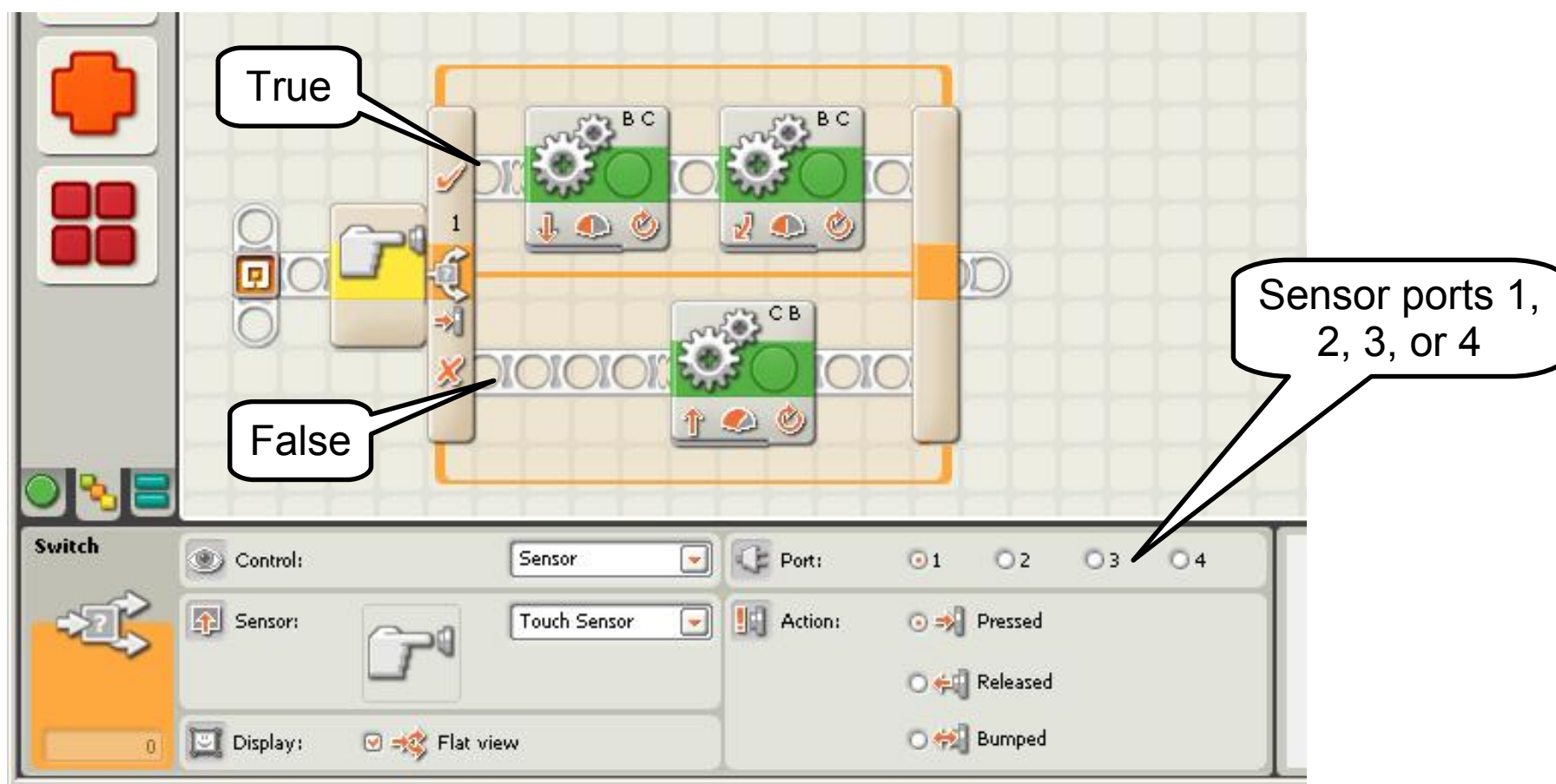
Sensor: Touch Sensor

Port: 1 2 3 4

Action: ☒ Pressed ☐ Released ☐ Bumped

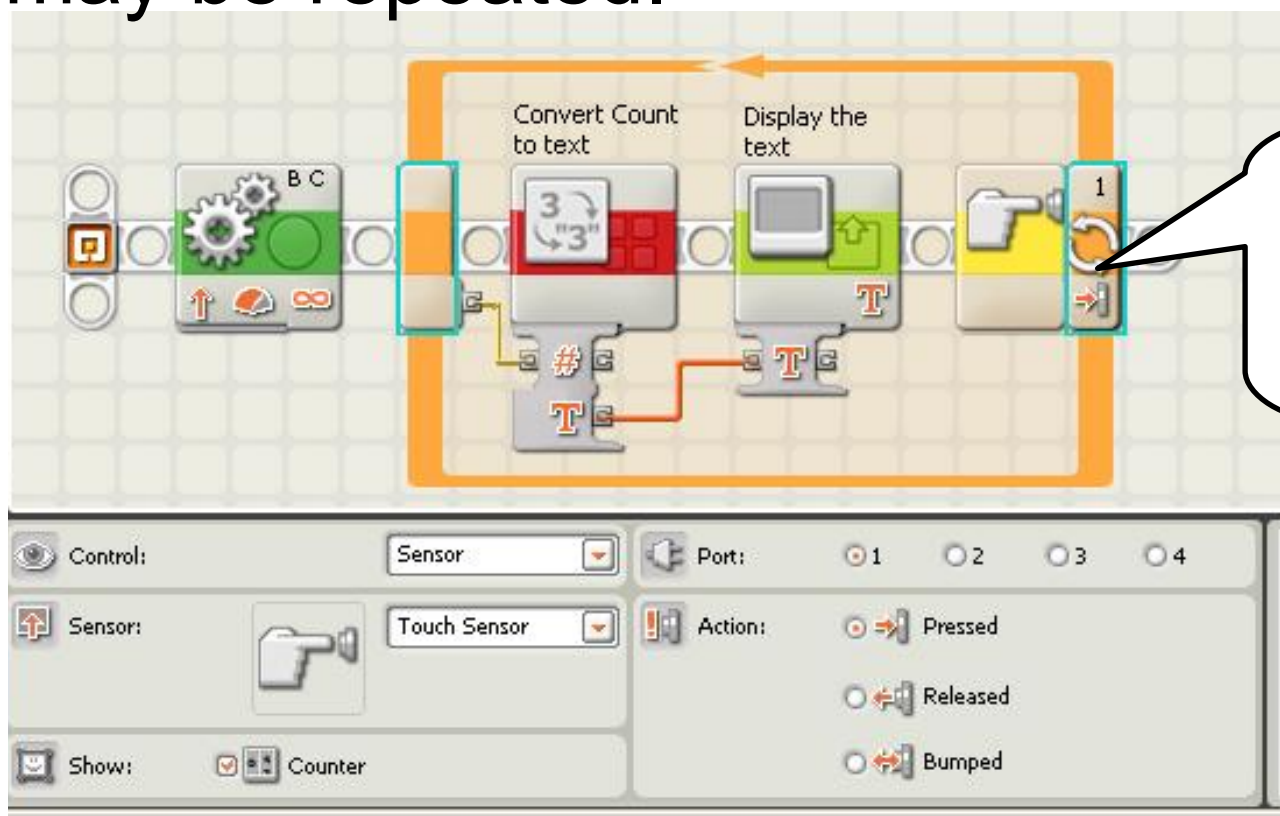
Touch Sensor Switch Block

- Waiting for a touch sensor can be useful, but many times you want to do different things based on the current value.



Touch Sensor Loop Commands

- Loop until a touch sensor is pressed. Useful if the loop contains commands that may be repeated.

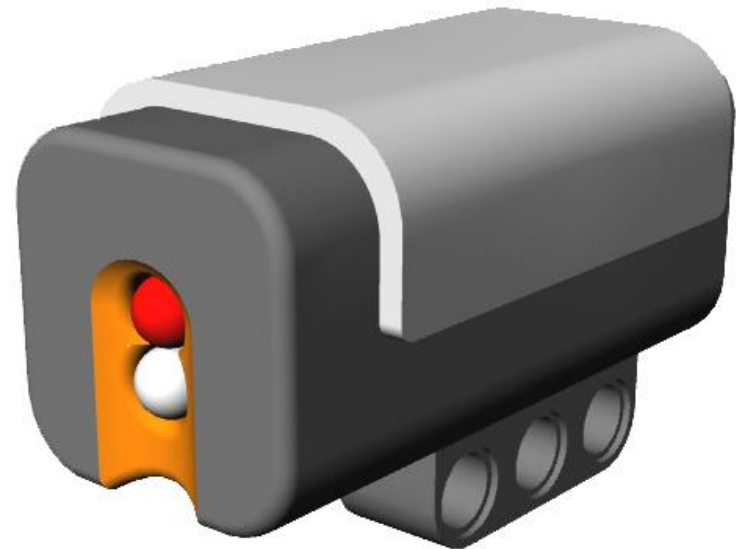


Loop contents
always run at least
once.

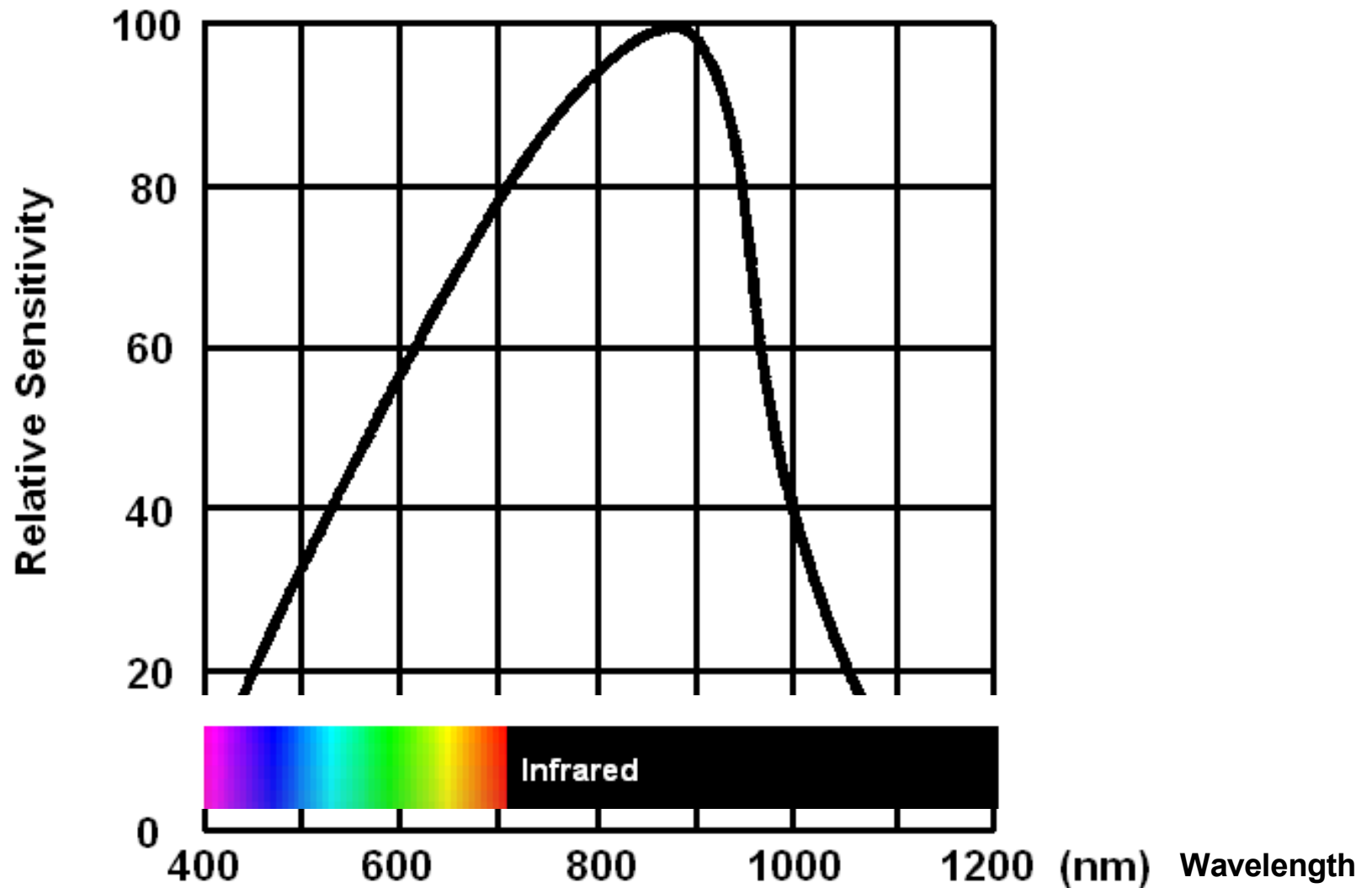
- For instance, a routine that beeps until a bumper hits something.

Sensor #2: Light

- Operates in "percent" mode
 - 0 to 100
 - Higher number = more light.
A lighter surface reflects more light.
- Calibrate the sensor.
- Light can be turned off.
- Shines a red light.



Light Sensor Spectrum

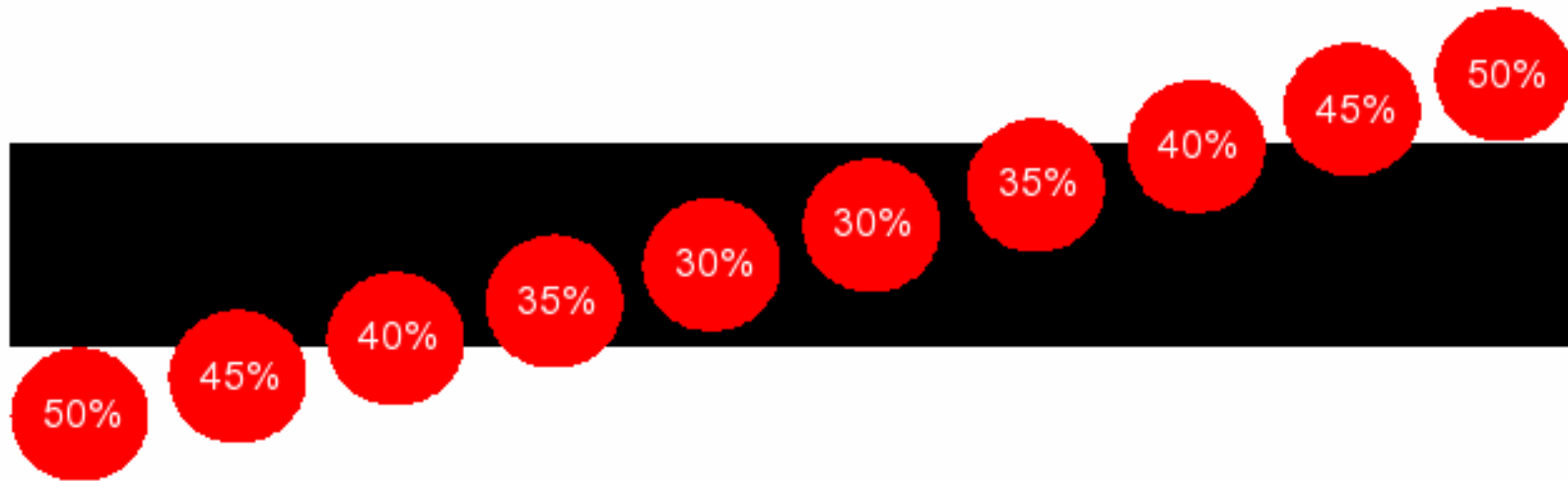


- Most sensitive to red/IR light.

Light Sensor Readings

- Lowest likely reading 5%
- Highest likely reading 100% (pointing at a light)
- Readings also depend on the color of the surface
 - See “Building LEGO Robots for FIRST LEGO League” by Dean Hystad.
- Sensitive to the distance between the sensor and the reflecting surface. Variations can make the readings unusable. Keep the sensor close to the surface, but not too close.
- Shield the sensor from other light sources.

Light Sensor Readings



- The light sensor averages its readings over roughly a circular area.
- Cross a line too fast and you may miss the line.
- Test and recalibrate on competition day.

Light Sensor WaitFor Block

Forward

Right Turn

Use WaitFor Blocks when watching only one sensor.

Control: Sensor

Port: 1 2 3 4

Sensor: Light Sensor

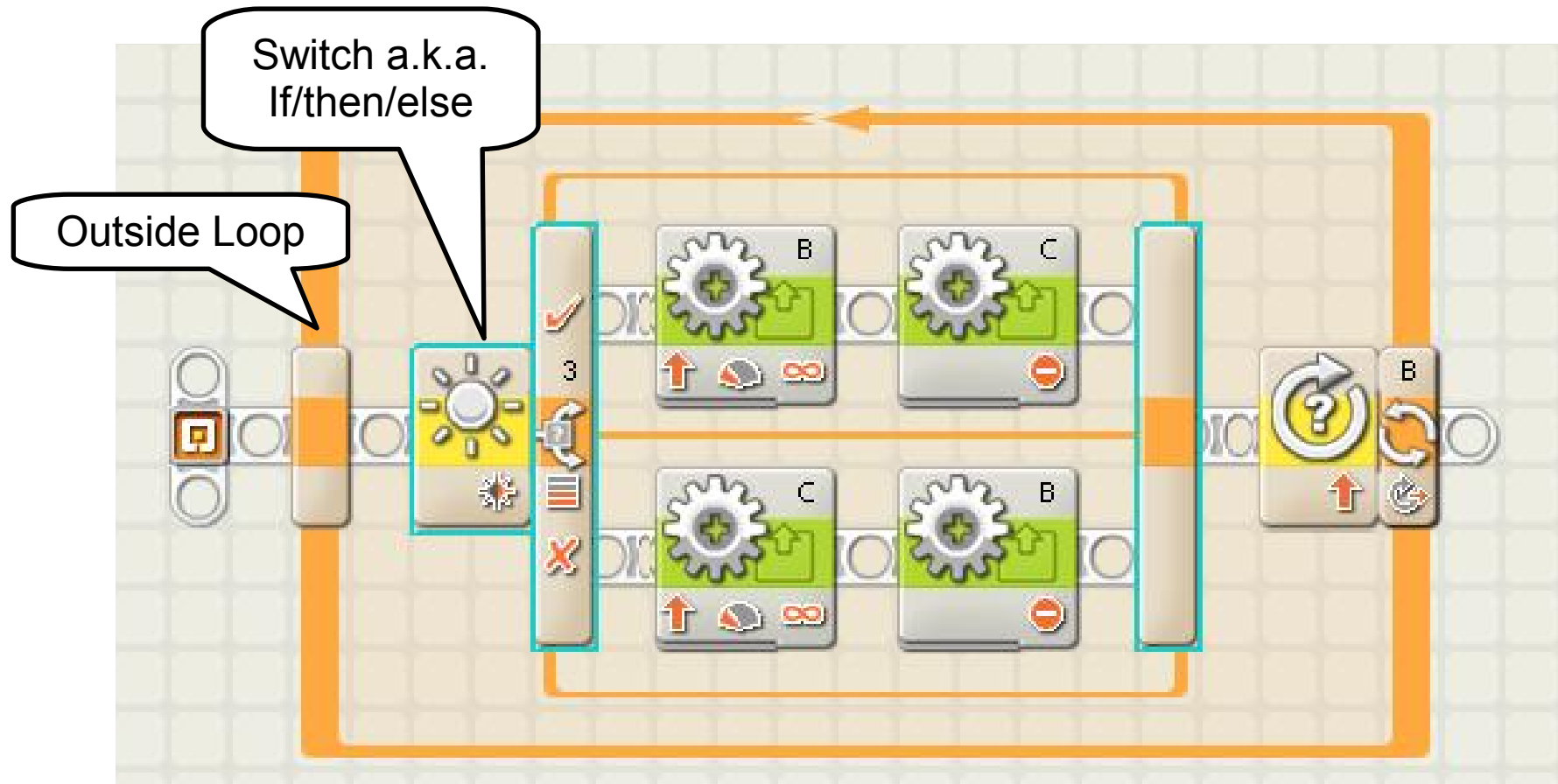
Until: [Slider]

Light: < 40

Function: ☒ Generate light

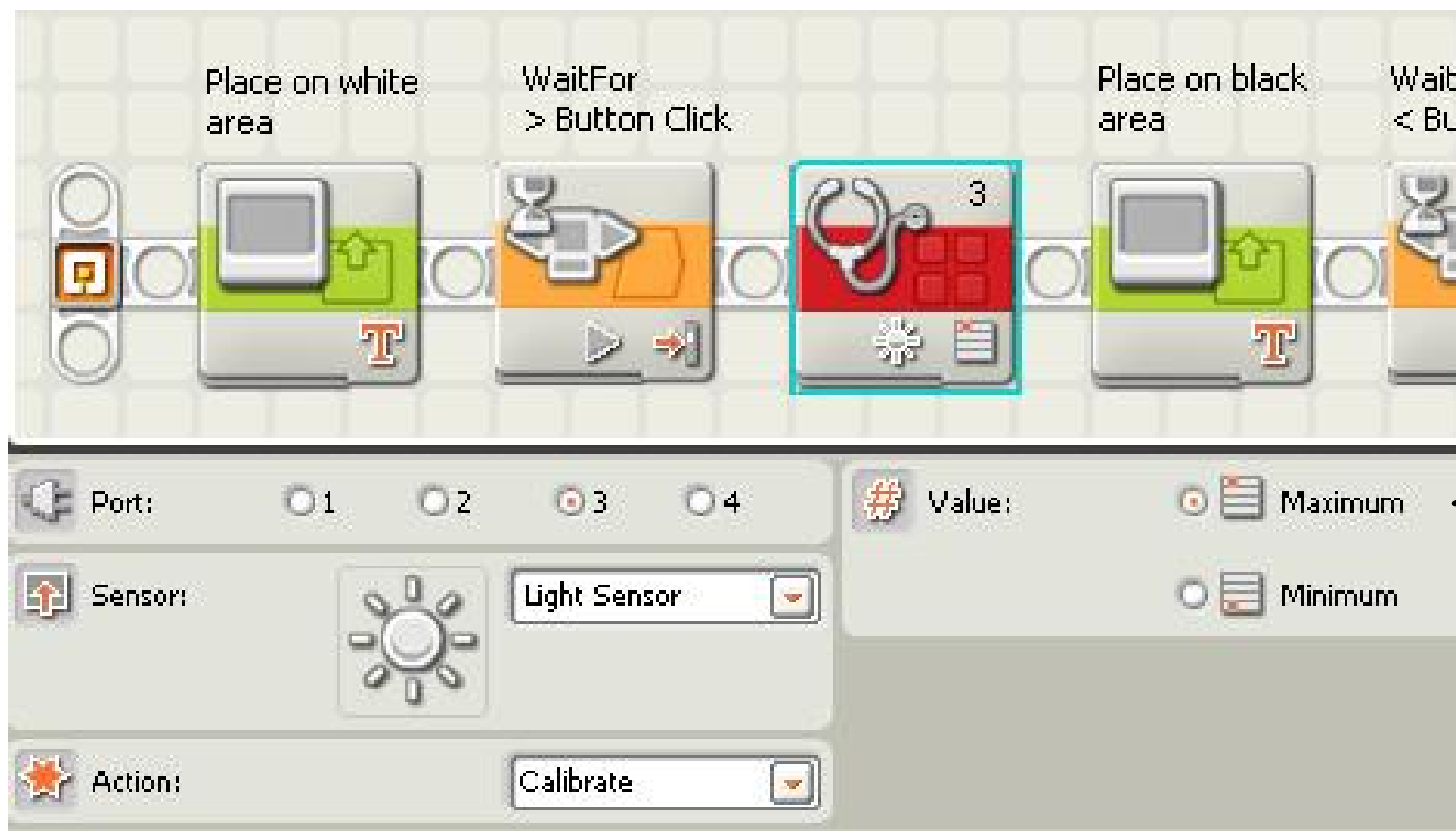
Turn light on/off

Light Sensor Switch Block



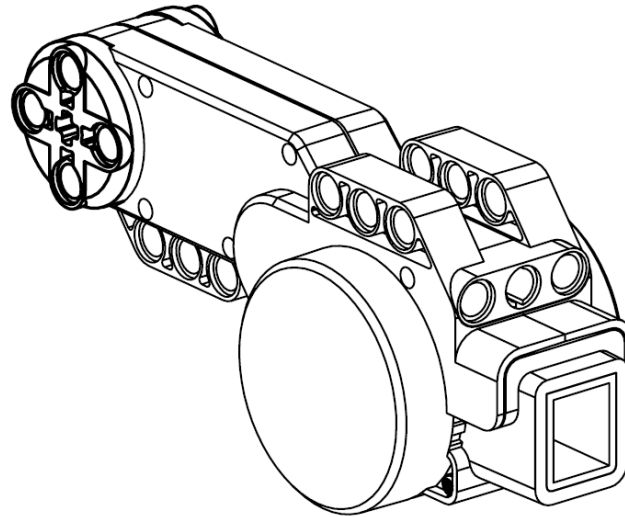
- If brighter than 50, turn motor B on. Stop C.
 - Otherwise, turn motor C on. Stop B.
- What is this?

Calibrate Light Sensor



- Move your robot over light and dark areas.
- Resets light sensor percentages.
- Can also be done on the NXT.

Sensor #3: Rotation



- Measures how far a rotating axle has turned. As the axle turns, a counter in the NXT is incremented or decremented.
- 360 counts per rotation.
- Each motor has an embedded rotation sensor.

Rotation: Move and Motor Blocks



A sensor is built into each motor.

Measure by rotations or degrees.

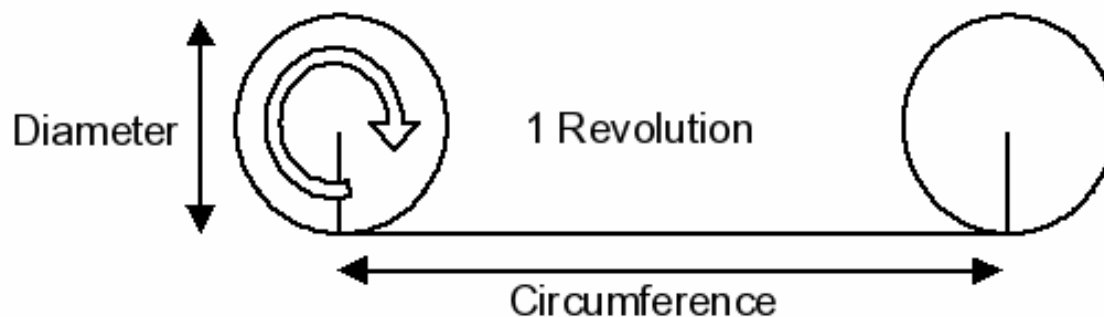
Rotation Sensor Loop Block



- Possible to make a rotation loop.
 - Select any loop. Pick “Sensor” type and then “Rotations.”

Calculating Distance

- The rotation sensor also brings in the possibility of doing some real math!

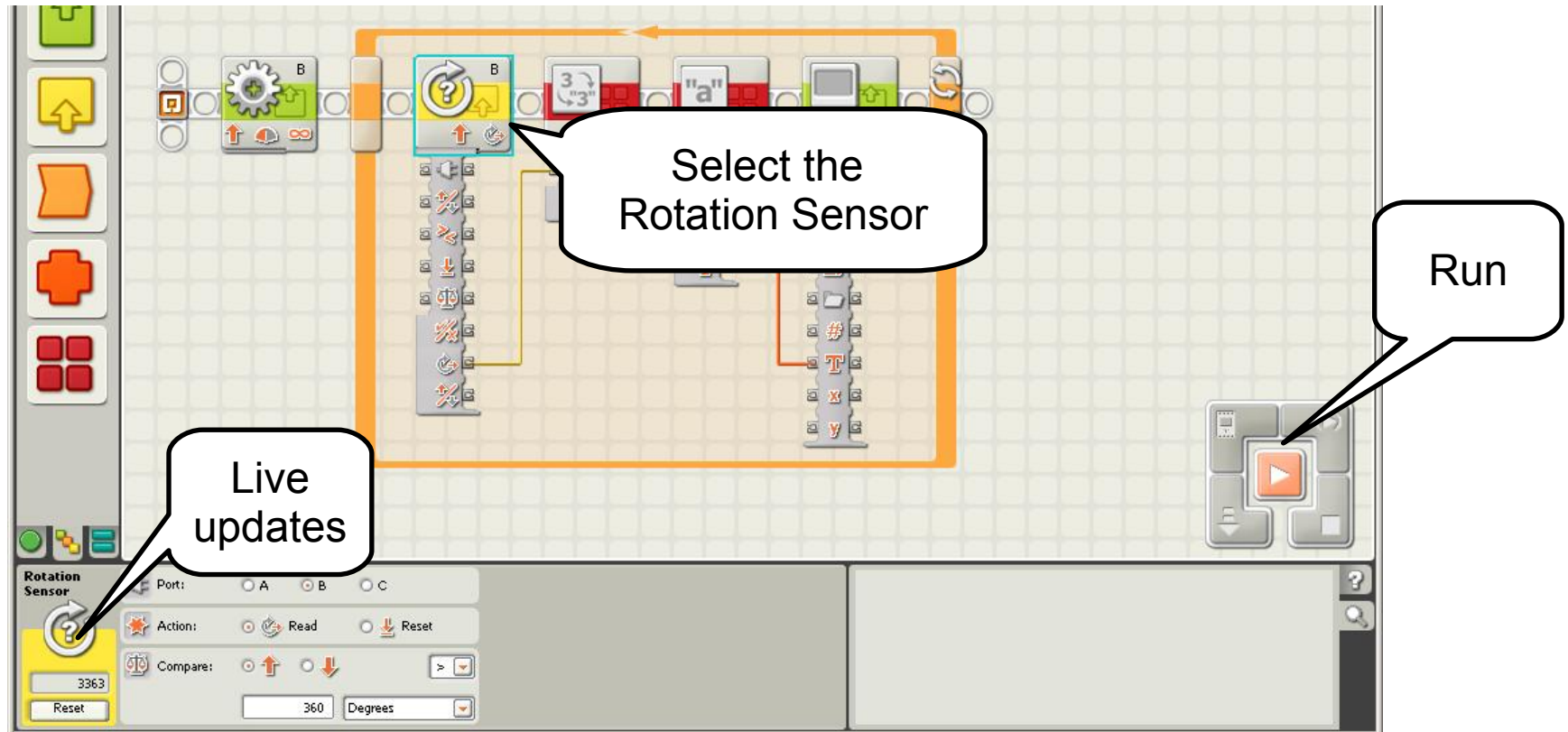


$$\text{Circumference} = \pi \times \text{Diameter}$$

- We'll leave that as an exercise for the reader!
- Of course, trial and error also works.
- Sources of error in calculation - dirt on surface, using a skid rather than a wheel, backlash (poor fitting gears).

More on Rotation Sensor

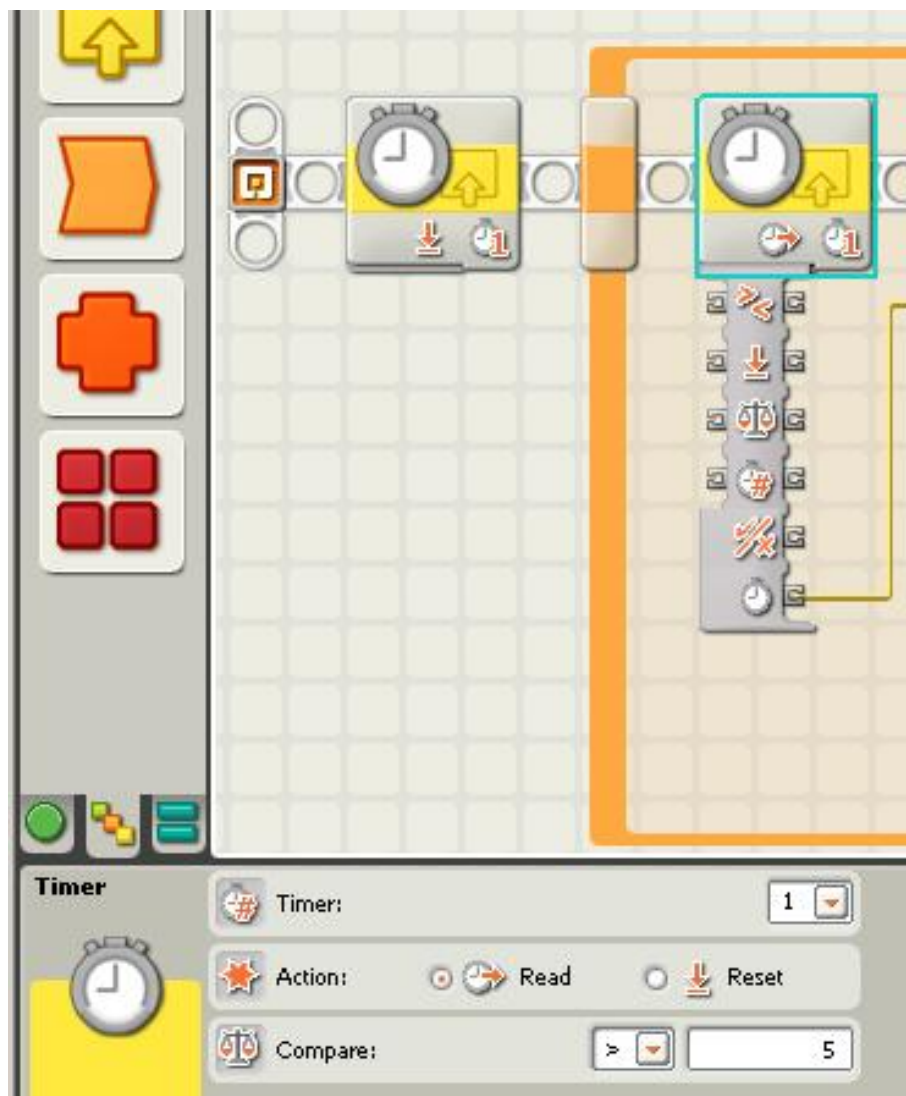
- Rotation sensor **counts forward and backwards.**
- Live updates (Bluetooth or USB connected).



Debugging and Analysis

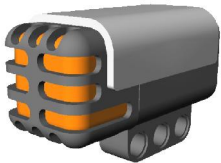
- Common problems
 - Programming: **reset the sensor to zero** before use.
 - Design: inadequate sensor resolution (trying to measure something very accurately, when the sensor is not that accurate).
 - Control: starting, stopping, turning **too fast**.
 - Variations in the initial conditions: not putting everything in the same place before pushing the run button.

Sensor #4: Timer



- 3 Timers
 - Time in thousandths of a second
 - Greater or Less than test
 - Reset
 - Trigger Point
 - Timer Number
 - Yes/No
 - Time as a number

Other Sensors



- Sound (microphone)
 - Is the FLL competition too loud?



- Ultrasonic (distance)
 - Interference with other NXTs?

- NXT Buttons
 - One touch running of the next program.

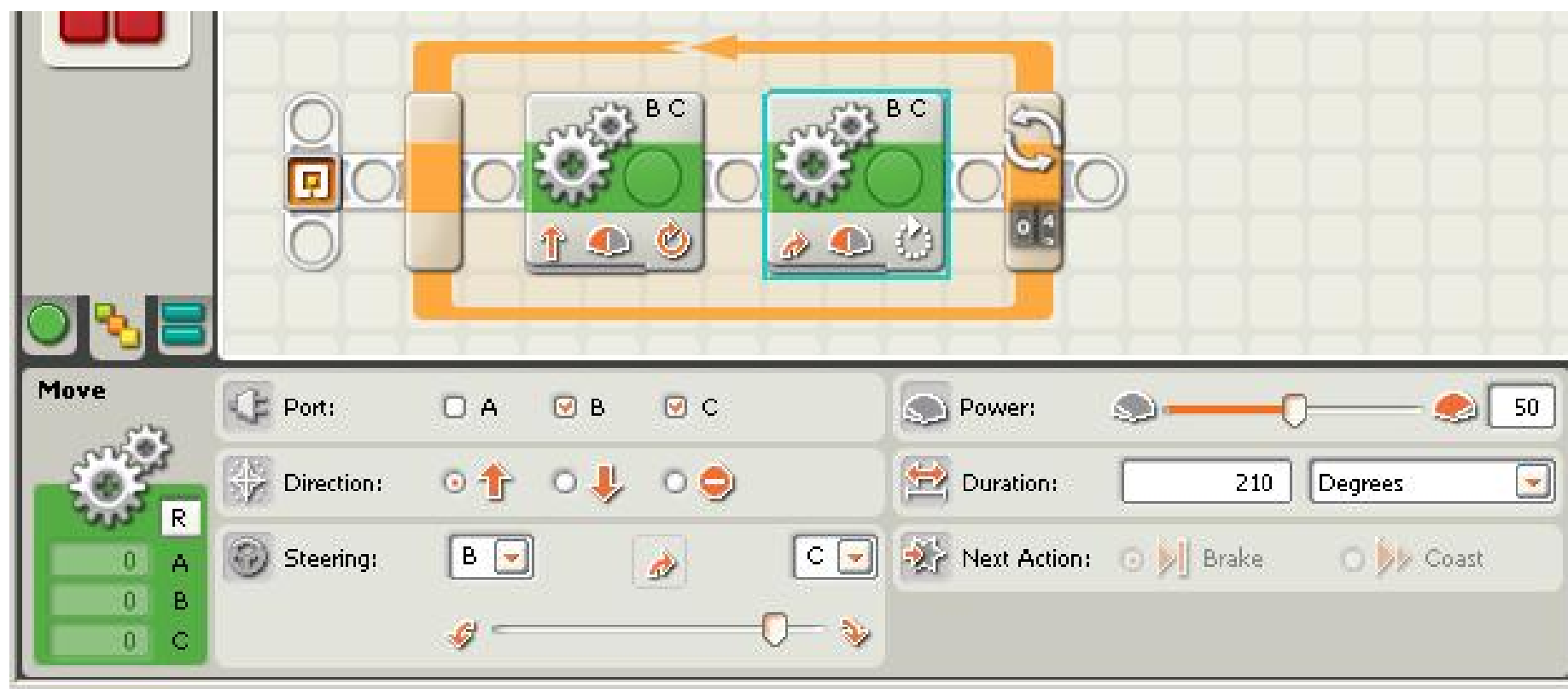
- Received Messages

Lab Three

Task:

**Move forward for 2 feet, turn right 90°
repeat to complete a square path.
End up exactly where you started.**

Lab 3 An Answer



Move forward and turn 4 times to form a square.

Keep it Simple Strategies

KISS #3: Variables

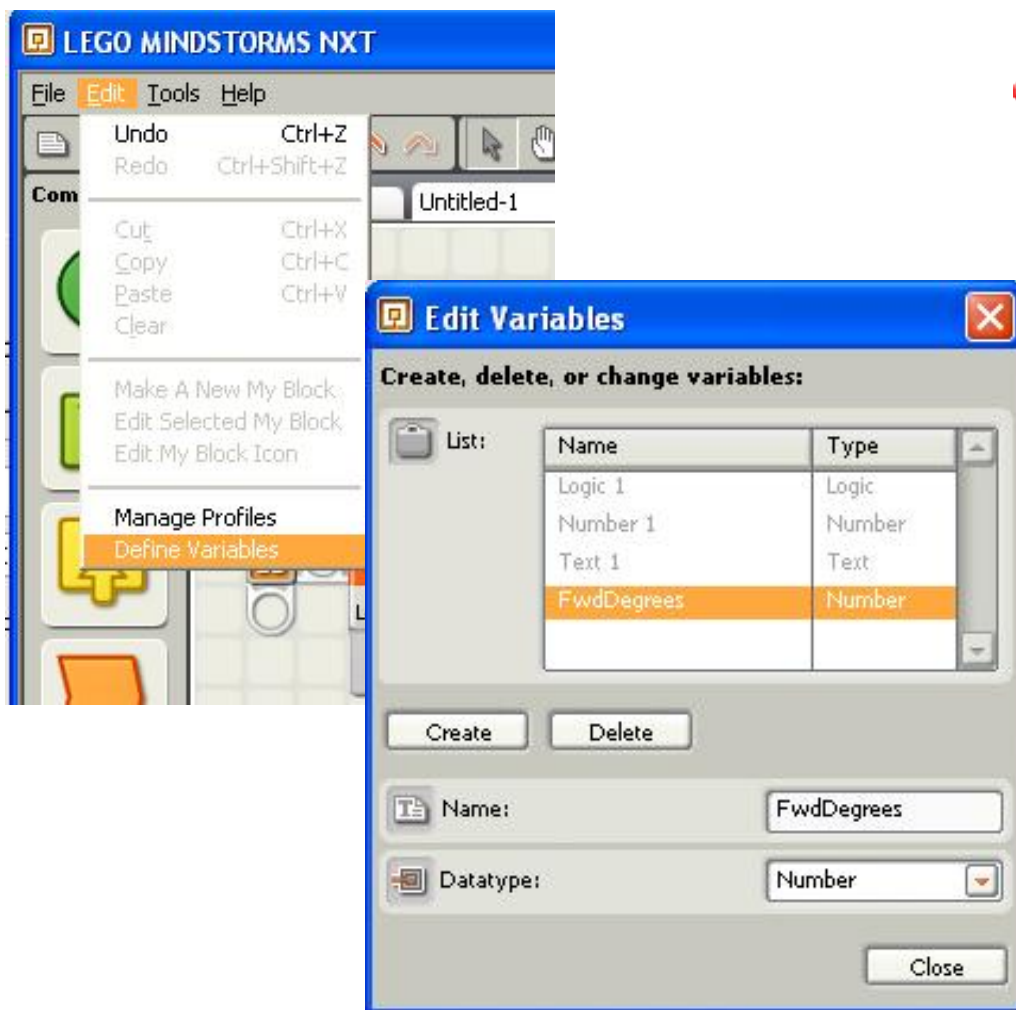
#4: Parallel Sequences

#5: Loops

What's a Variable?

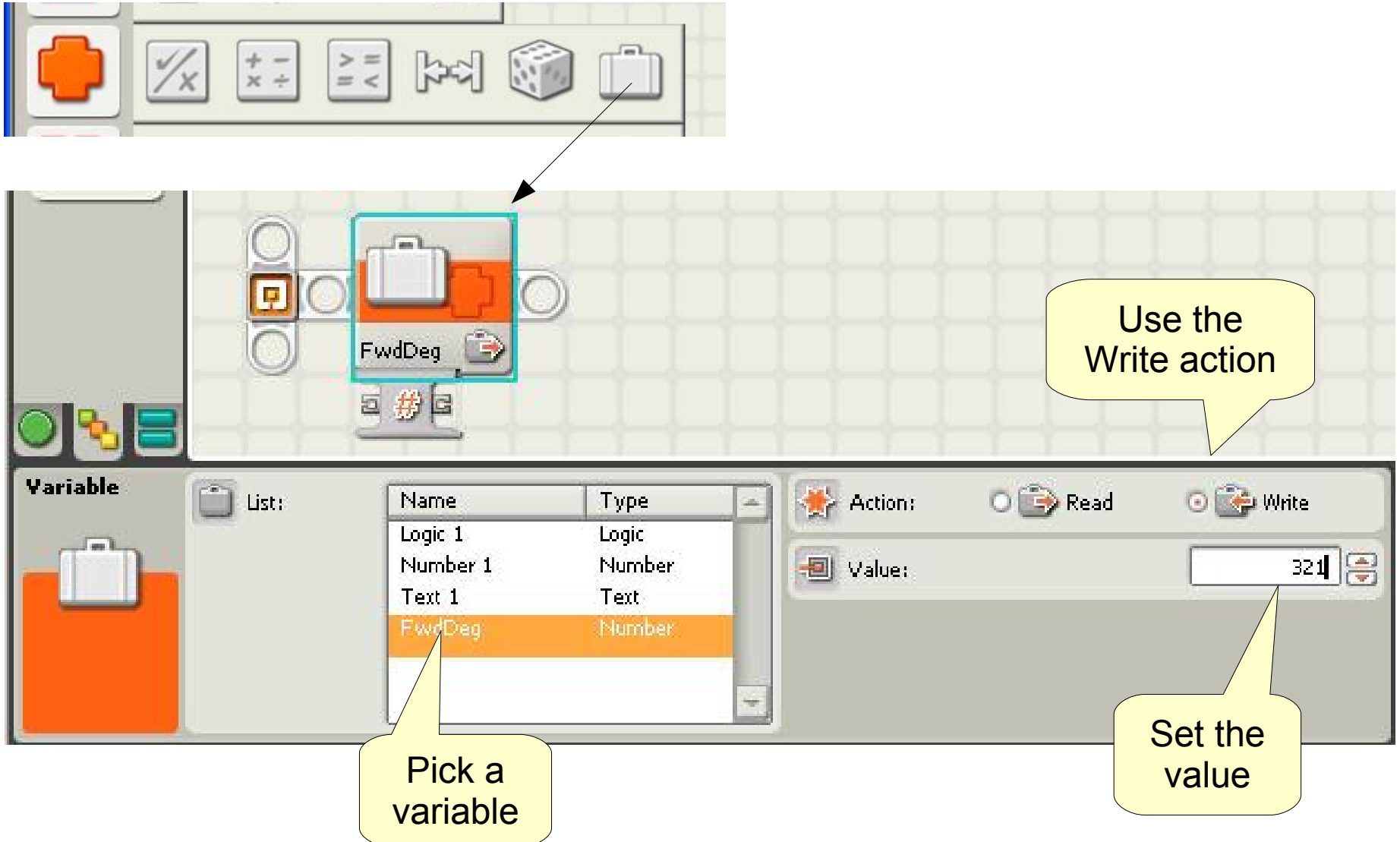
- A value that you can change during your program.
 - This value is “variable”, hence the name.
- For example, your program may store a light sensor reading in a variable called *LightBright*. Use that value later.
- Use a meaningful name.
- Useful to pass values to MyBlocks

KISS #3: Variables



- Creating a variable
 - <Edit><Define Variables>
 - Create
 - Name the variable
 - Select a datatype
 - Number
 - Text
 - Logic

Using a Variable



Use the Write action

Pick a variable

Name	Type
Logic 1	Logic
Number 1	Number
Text 1	Text
FwdDeg	Number

Set the value

Value: 321

Using a Variable

Wire variable value to motor duration

Pick a variable

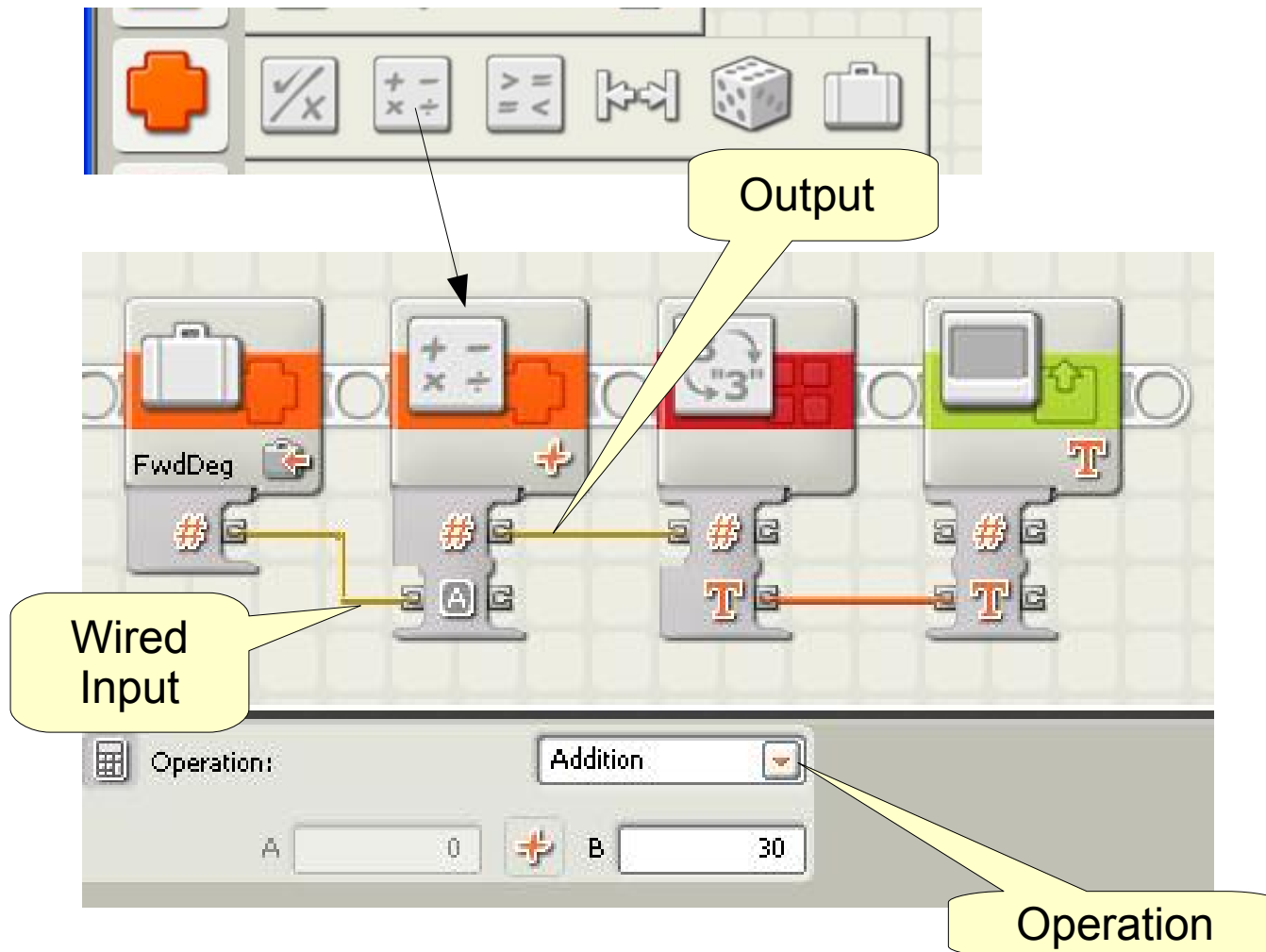
Use the Read action

Name	Type
Logic 1	Logic
Number 1	Number
Text 1	Text
FwdDeg	Number

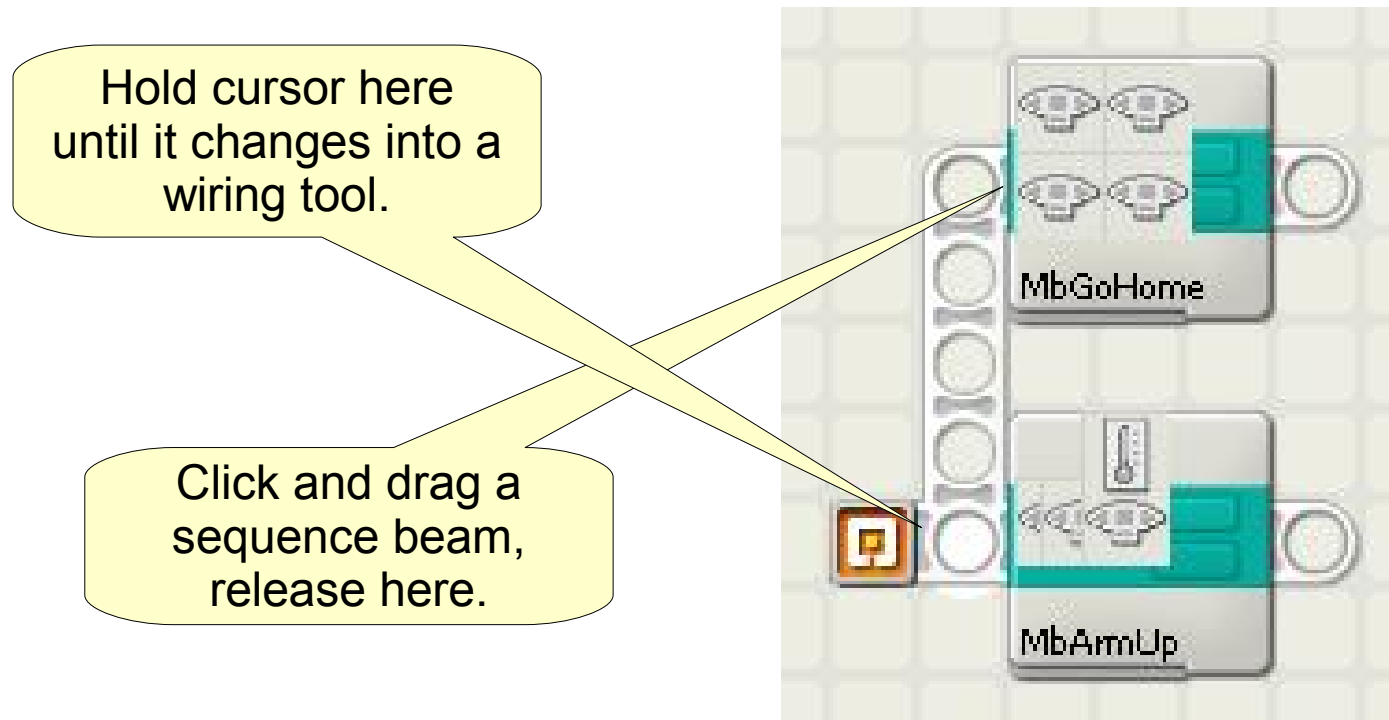
Action: ☒ Read ☐ Write

Value:

Arithmetic



KISS #4: Parallel Sequences



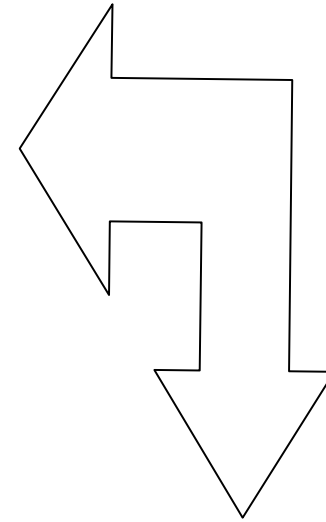
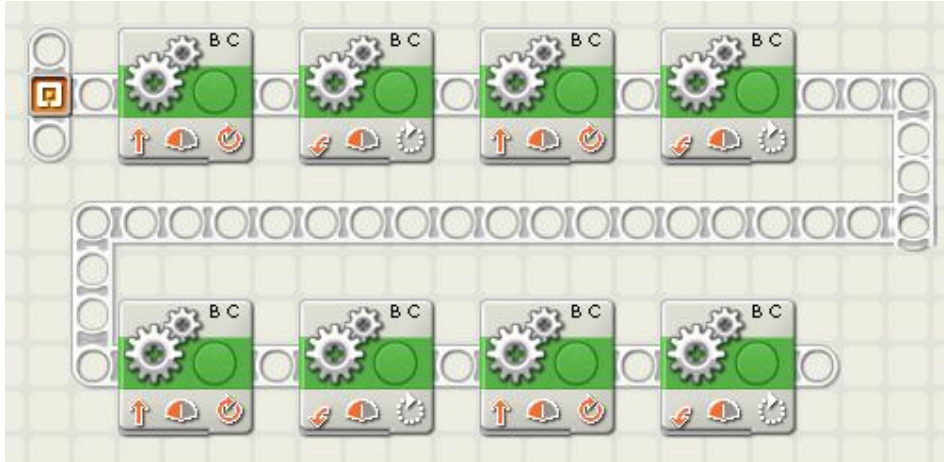
- Make two tasks that run independently. (Can you walk and chew gum?)
- One task lifts the arm. The other task heads for home.

KISS #5: Loops

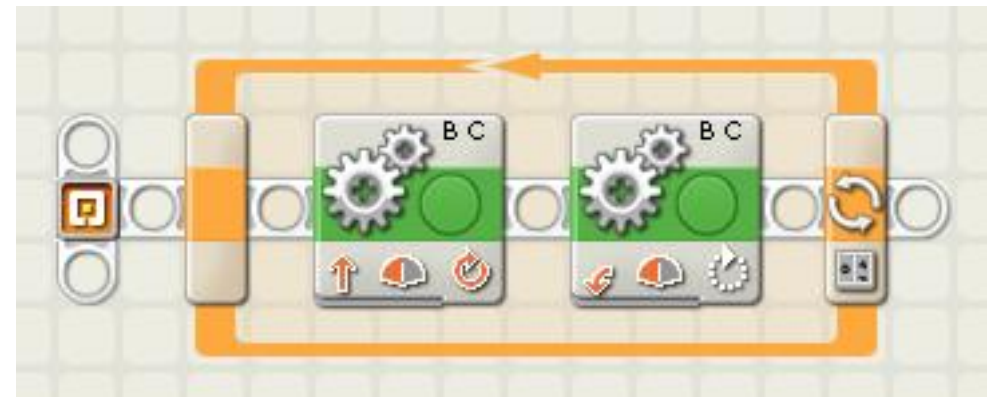
- Loops are a control structure
 - In other programming languages:
 - For ... Next Do loop n times
 - Do ... Until Do it. Unless some test, do it again.
- There are loops for
 - Forever
 - Every sensor (including time)
 - Logic
 - Count
- If your algorithm says something like: “Until the sensor reads x, keep doing this”, use a loop.



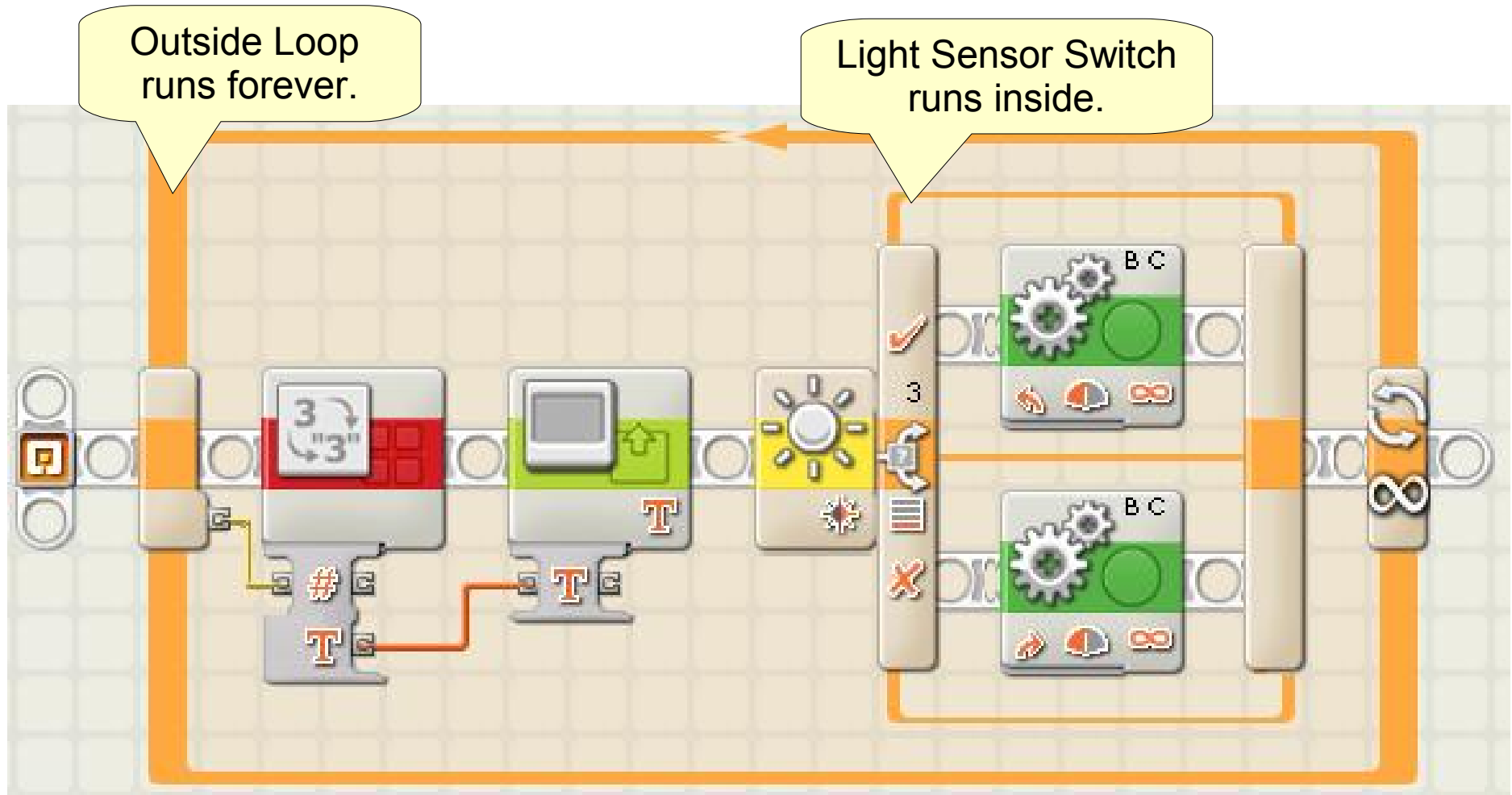
Simple Loop



- Convert this
- to something simpler using a loop

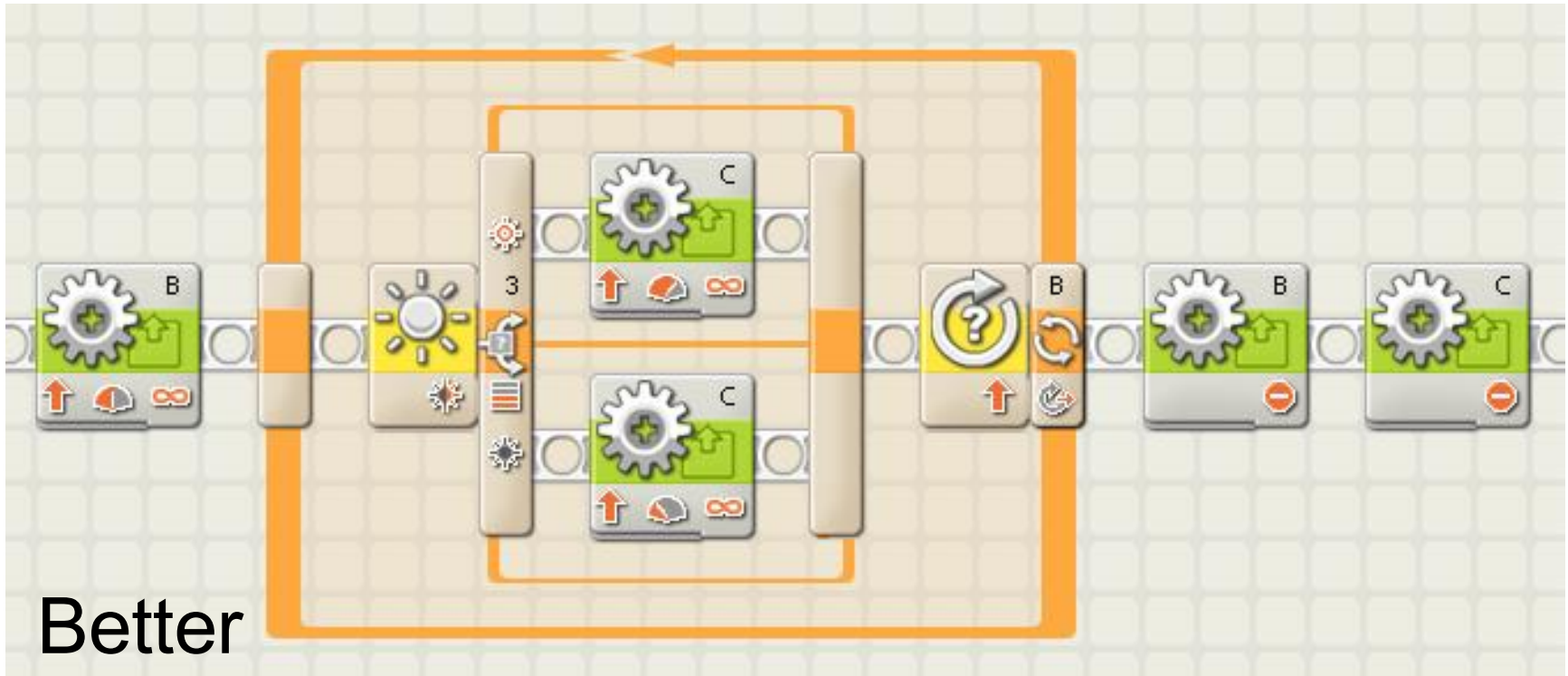


Combining Structures



Comparing Algorithms

- Compare that line follower to this one:

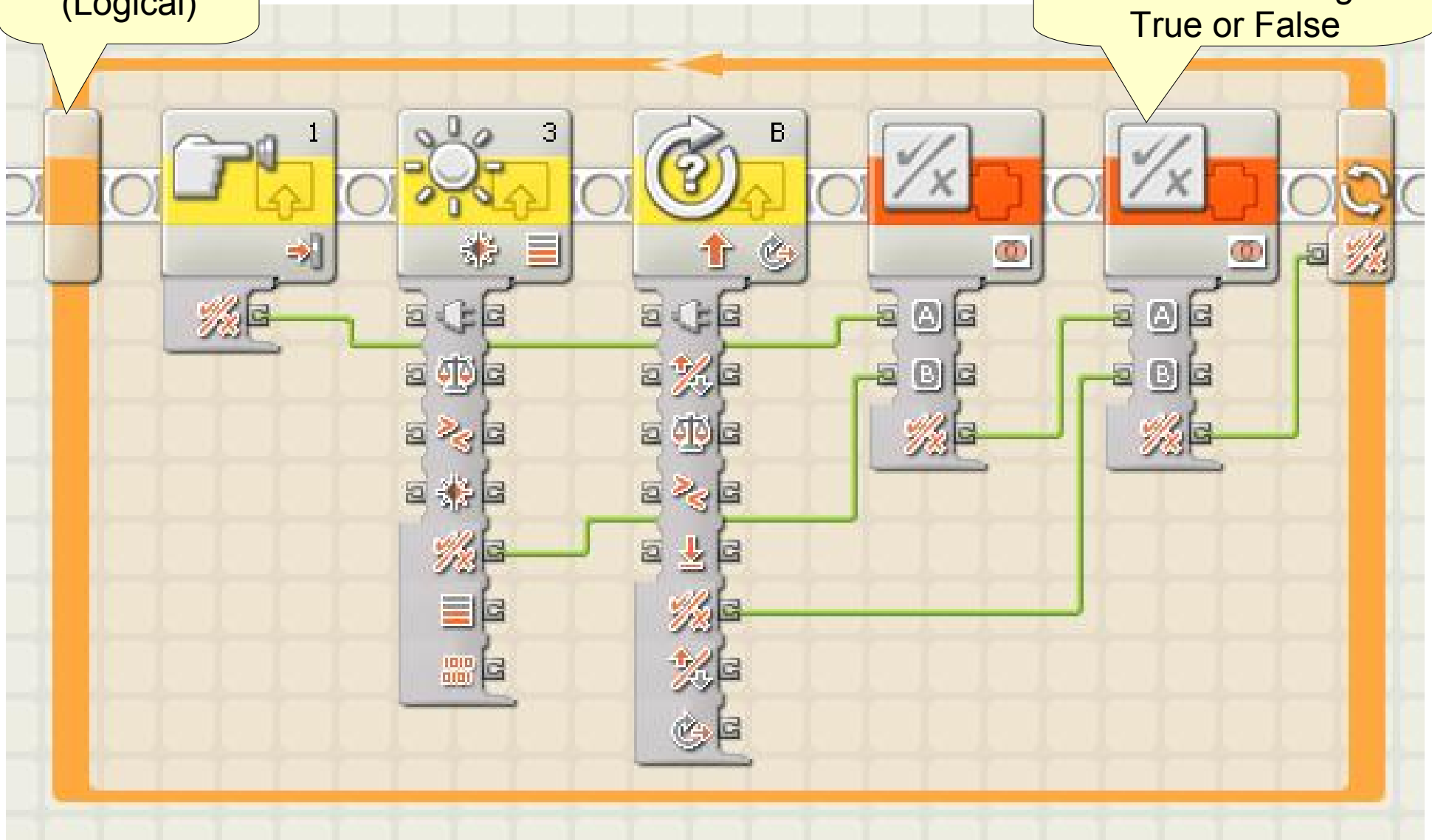


- Better
 - Simpler. One motor changes speed. Faster.
- Worse
 - Tight corners?

Watching 3 Sensors

Outside Loop
(Logical)

Logical ORs combine
results into a single
True or False



Comparing Structures

- Rework your algorithm to fit the structures.
- Loop
 - Execute something until an event
- WaitFor
 - Wait for one event, then continue.
- Switch
 - Make a choice based on a sensor value at a given point in time.
 - Be careful to make sure you will be watching for the event at the right time

Tricks and Tips

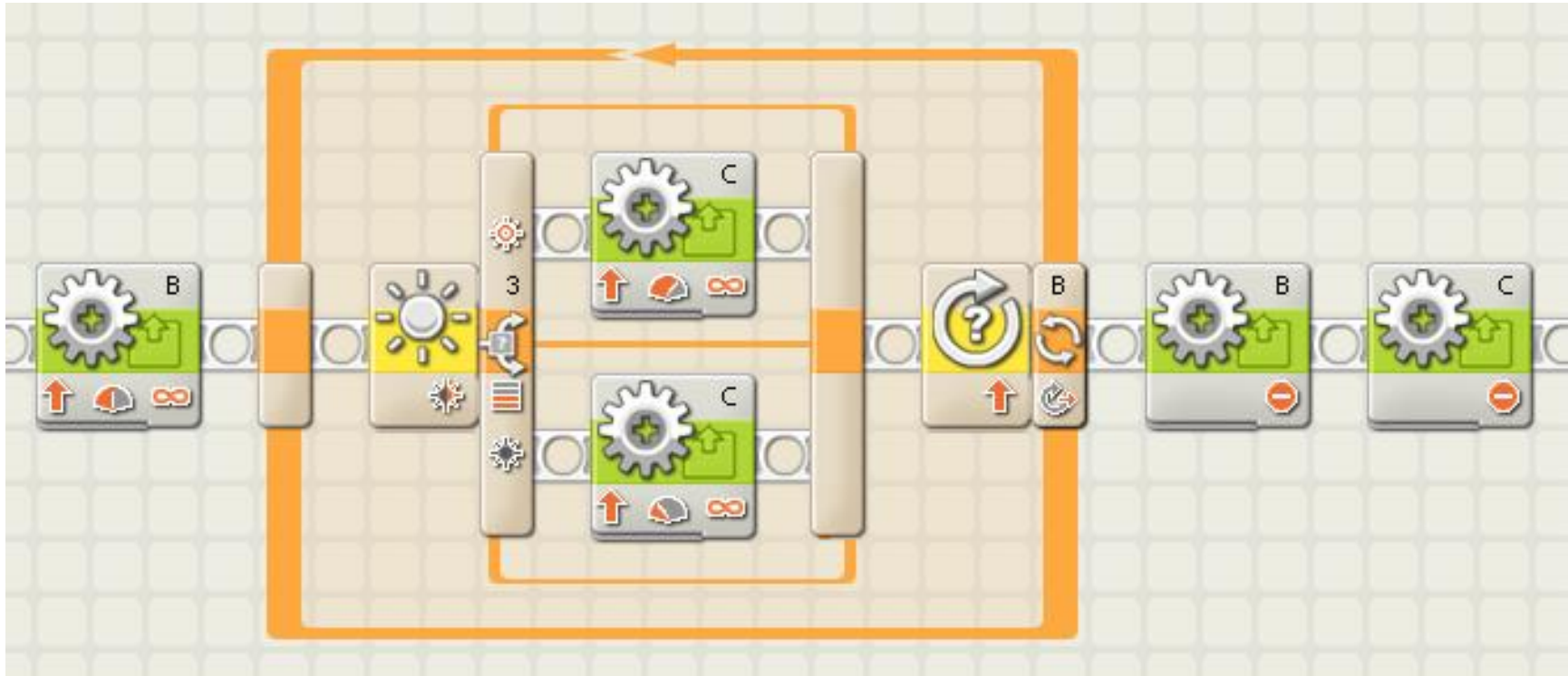
- Never wait for a sensor to be an exact value, always greater than or less than
 - You could miss the event due to sampling rate
 - For example, wait until rotation greater than 64 not equal to 64.

Lab Four

Task:

**Move exactly one lap around an oval.
(Black 2cm line on white paper)**

Lab 4 An Answer



- Did you start by copying? Why not?
- Is building a robot like taking a test?

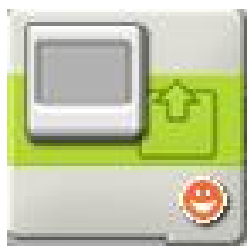
Advanced Topics

Debugging Tools
Additional Resources

Debugging Tools



- Music
 - Use music to identify sections of code.
 - One or two quick notes, a good ear can hear the difference.



- LCD
 - Write text, even graphics, to the NXT LCD panel.



- NXT Live updates
 - From your PC (or MAC), use NXT-G to view the values of variables and sensors.

Switch Between Cases

One switch block with 3 tabs, one for each case.

Click here to add cases.

Text or number type.

Must not use flat view

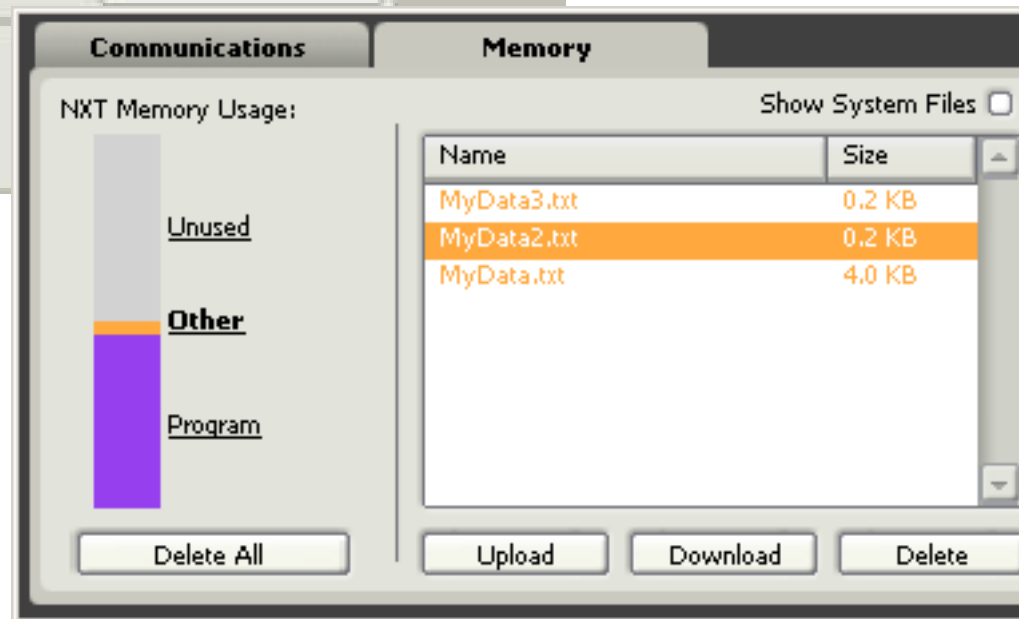
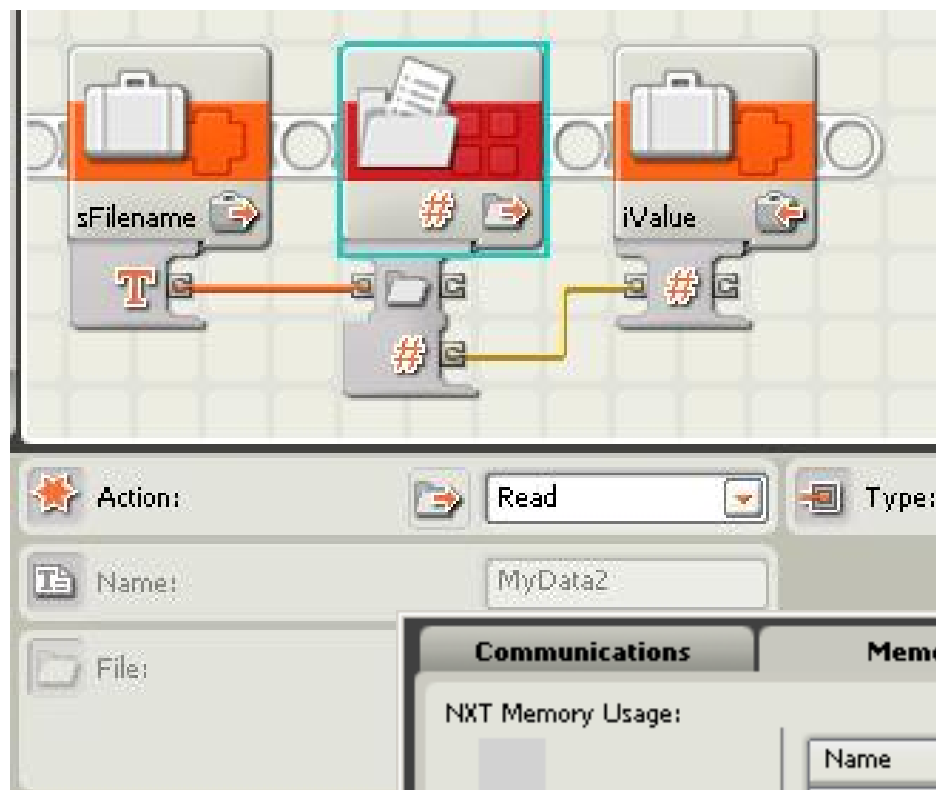
Control: Value
Type: Text
Display: ☒ Flat view

Case	Value
1.	dark
2.	grey
✓ 3.	light

light

File Operations

- Open
 - Read
 - Write
 - Close
 - Delete
-
- Upload/Download



Additional Resources

- Internet:
 - www.hightechkids.org (this presentation)
 - www.lugnet.org
 - mindstorms.lego.com
 - www.firstlegoleague.org
 - www.ni.com (LabVIEW™)
- Books
 - ???

Putting it All Together

How to Become an FLL Ace Programmer in
10 Easy Steps

FLL Ace Programmer in 10 Steps

- 1. Create a map of where the robot goes and what it does. These are your Requirements.
- 2. Use the Requirements to further examine the problem
 - What tasks can go in the same program?
 - Any actions we do in multiple places? (good candidates for subroutines)
 - Will using variables help?
- 3. Write out your algorithm

FLL Ace Programmer in 10 Steps

- 4. How it could fail. How can you recover?
 - For example, the robot hits a wall it shouldn't have. What can you do to allow it to recover?
- 5. How are you going to test and debug it?
 - Perhaps use a series of beeps in the program to tell you where the program is.
- 6. Have a system for versions.
 - Put comments at the beginning of the program
 - Always save a working version in a file with a name that makes sense (like date, etc.).

FLL Ace Programmer in 10 Steps

- 7. Write the code using above information
 - Code little parts and test them.
 - Name subroutines and files with descriptive names. Right_turn is better than Rturn.
 - Think about how readable the code is. Make it less confusing.
 - Use a lot of comments.

FLL Ace Programmer in 10 Steps

- 8. Fix bugs in a stepwise manner
 - Fix the bug, test it, then test other things related to it to make sure they weren't broken by your fix.
 - 80% of the bugs come from 20% of the code.
- 9. Don't be afraid to scrap everything and start over if things are getting complex and fragile.
- 10. If coding/testing/bug fixing is driving you insane, go have an ice cream cone! Take a break, have a friend look at your code, come back another day.

Summary

- Remember the problem solving process
- Create the algorithm before writing your code
- Use subroutines to keep things cleaner
- Be careful in using sensor commands.
- Use comments liberally
- Program and test little pieces at a time
- Have fun!!

