
FLL Programming 101

NXT



June 2008

Your Presenter

Presenters:

- Lamar High School
- CORE

Skills Inventory

- Any programming experience?
- FLL coaching experience?
- NXT or RIS programming experience?
- Have a robot with you?

NXT Programming 101

Objective

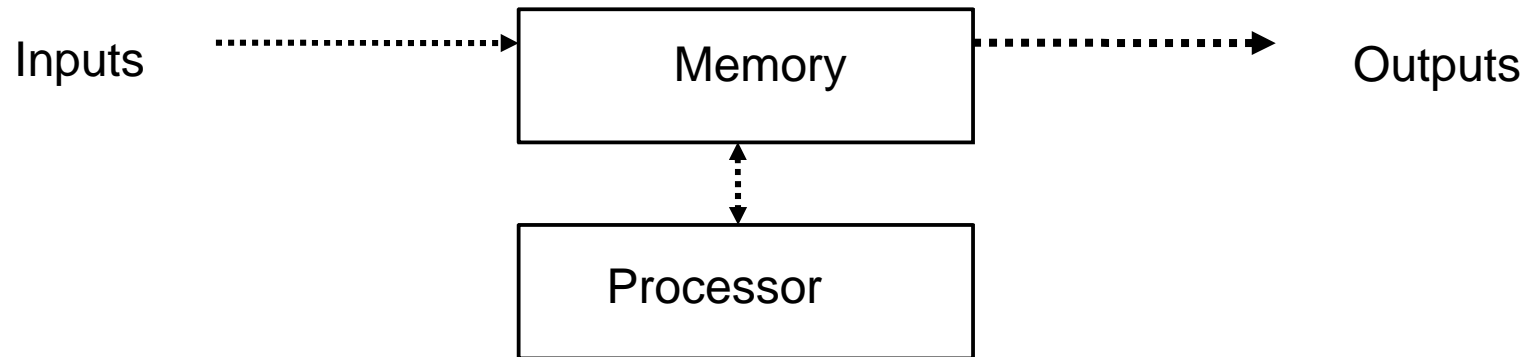
Develop a basic approach to, and understanding of, programming the NXT.

Class Agenda

- Computer Basics
- The Programming Environment
- Basic Programming Steps
- Common Blocks
- 7 Practical Exercises
- Sensors
- Advanced Topics
- Putting It All Together

Computer Basics

The Computer



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

Computer Programs

- Can be used to model a process.
- **KISS=Keep It Silly**
- Rarely perfect.

Writing a Computer Program

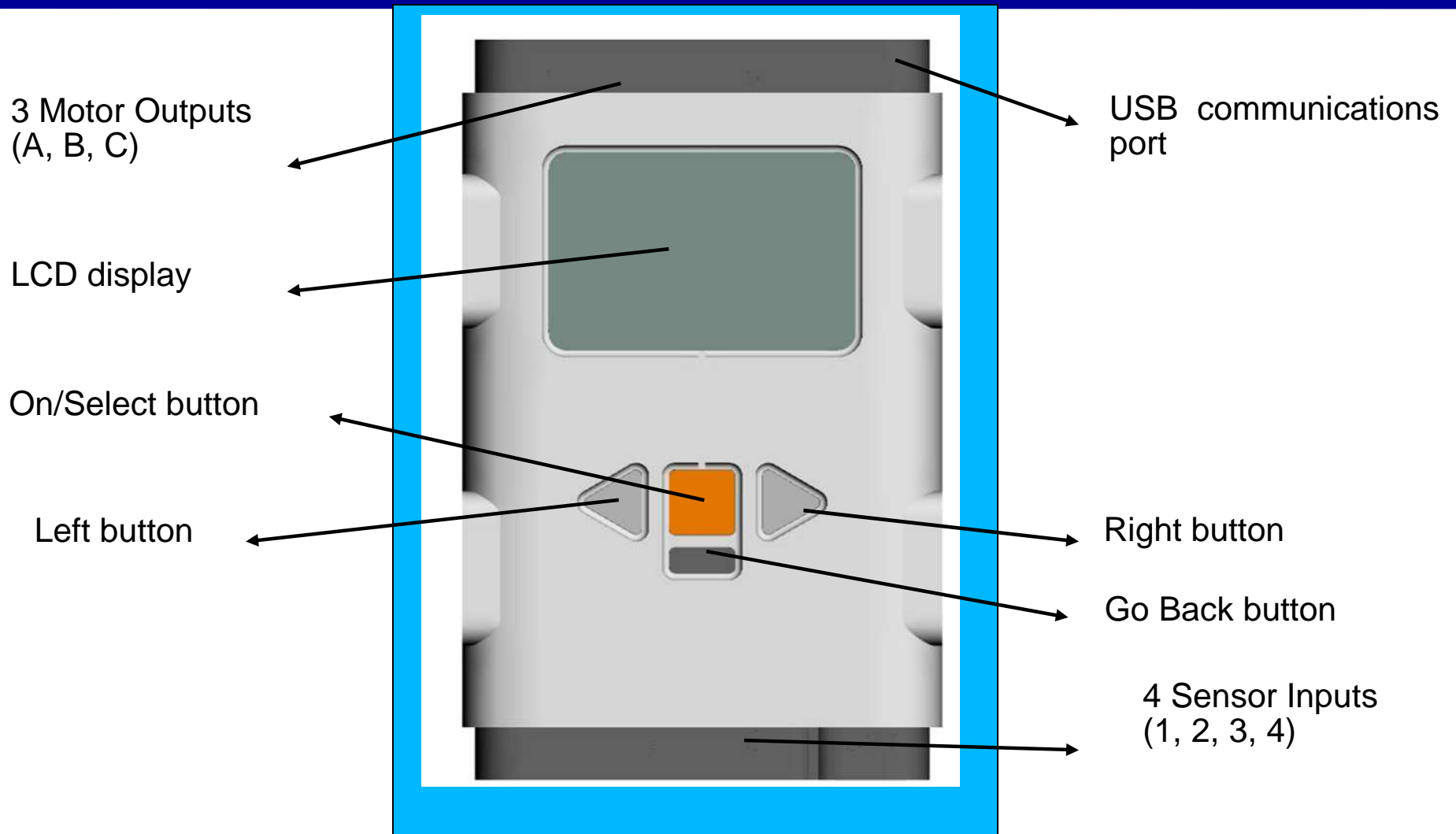
Specify the task

- Inputs to be supplied
- Outputs to be produced

Devise a plan or strategy.

Express that strategy in a computer language.

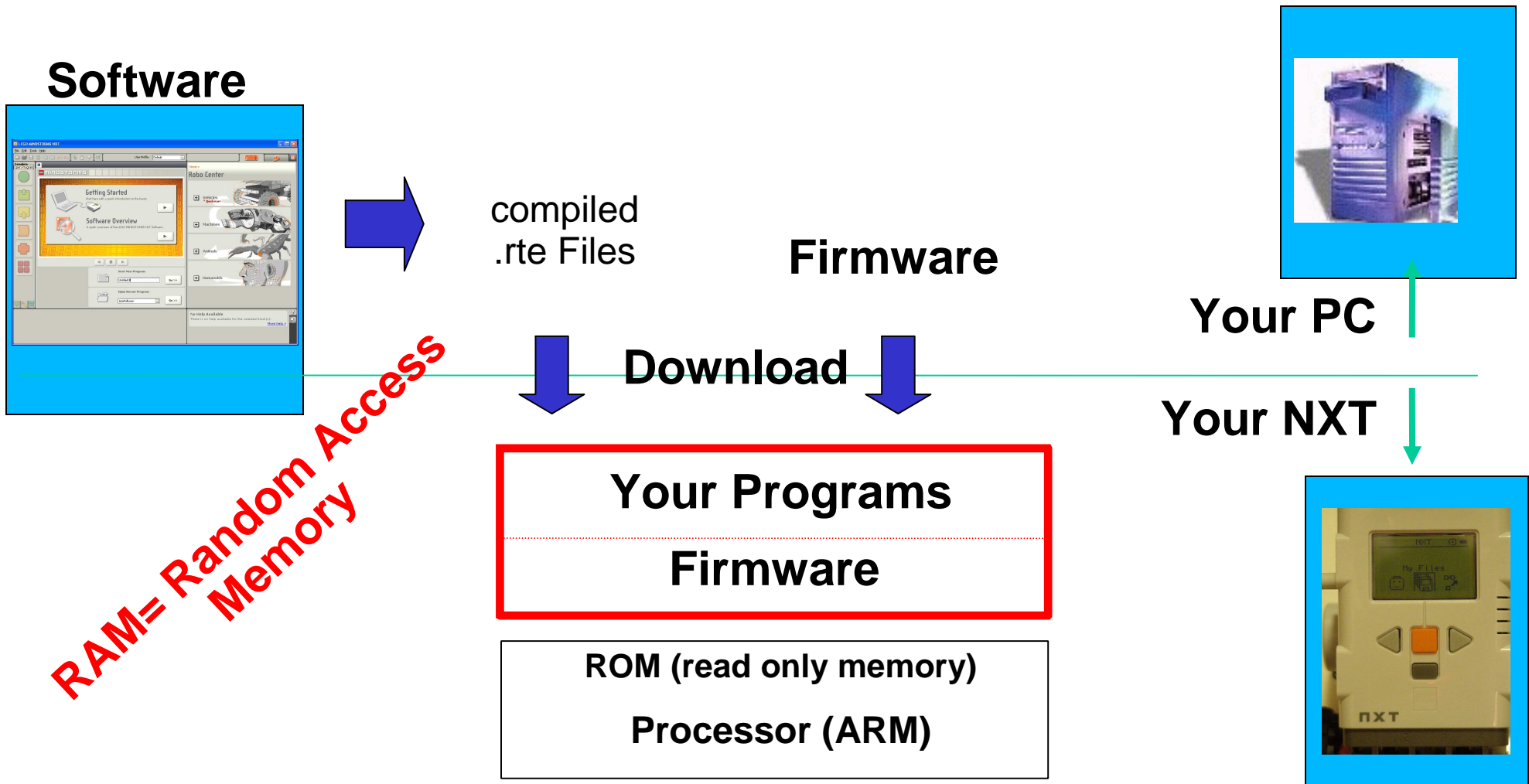
NXT



Processor: 32 bit ARM Atmel AT91SAM256 running at 50 Mhz

Memory: 64K Static RAM, 256K Flash

NXT Firmware



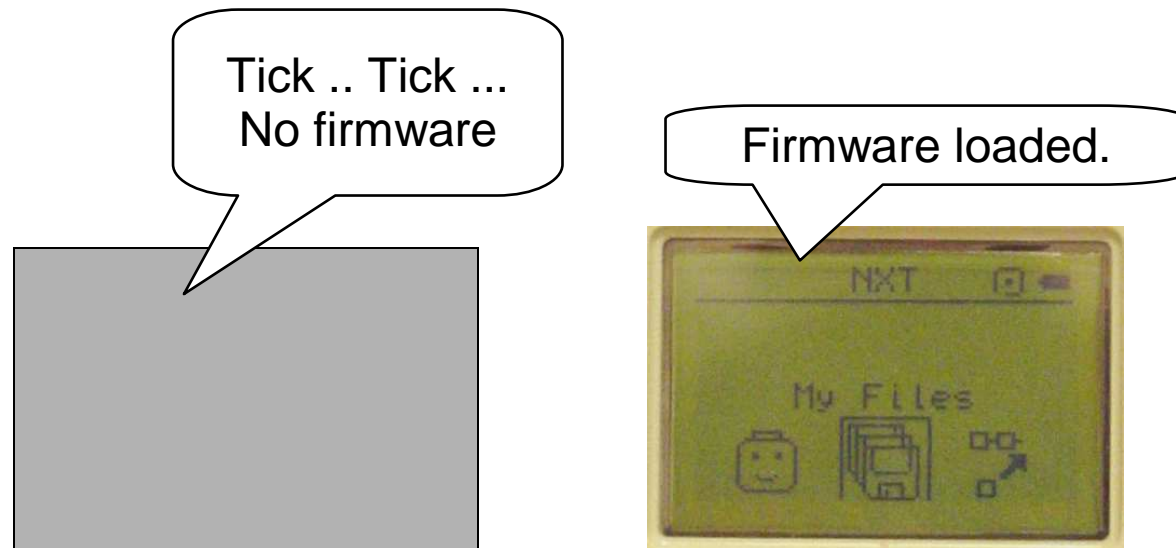
For FLL purposes, think of firmware as the operating system

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.



Mindstorms Languages

NXT

NXT-G Programming Language

- Comes with the NXT version of LEGO Mindstorms
- Runs on PC or a MAC
- Has MyBlocks like Robotics Invention System (RIS)
- Based on LabVIEW like RoboLab

RoboLab 2.9

RCX

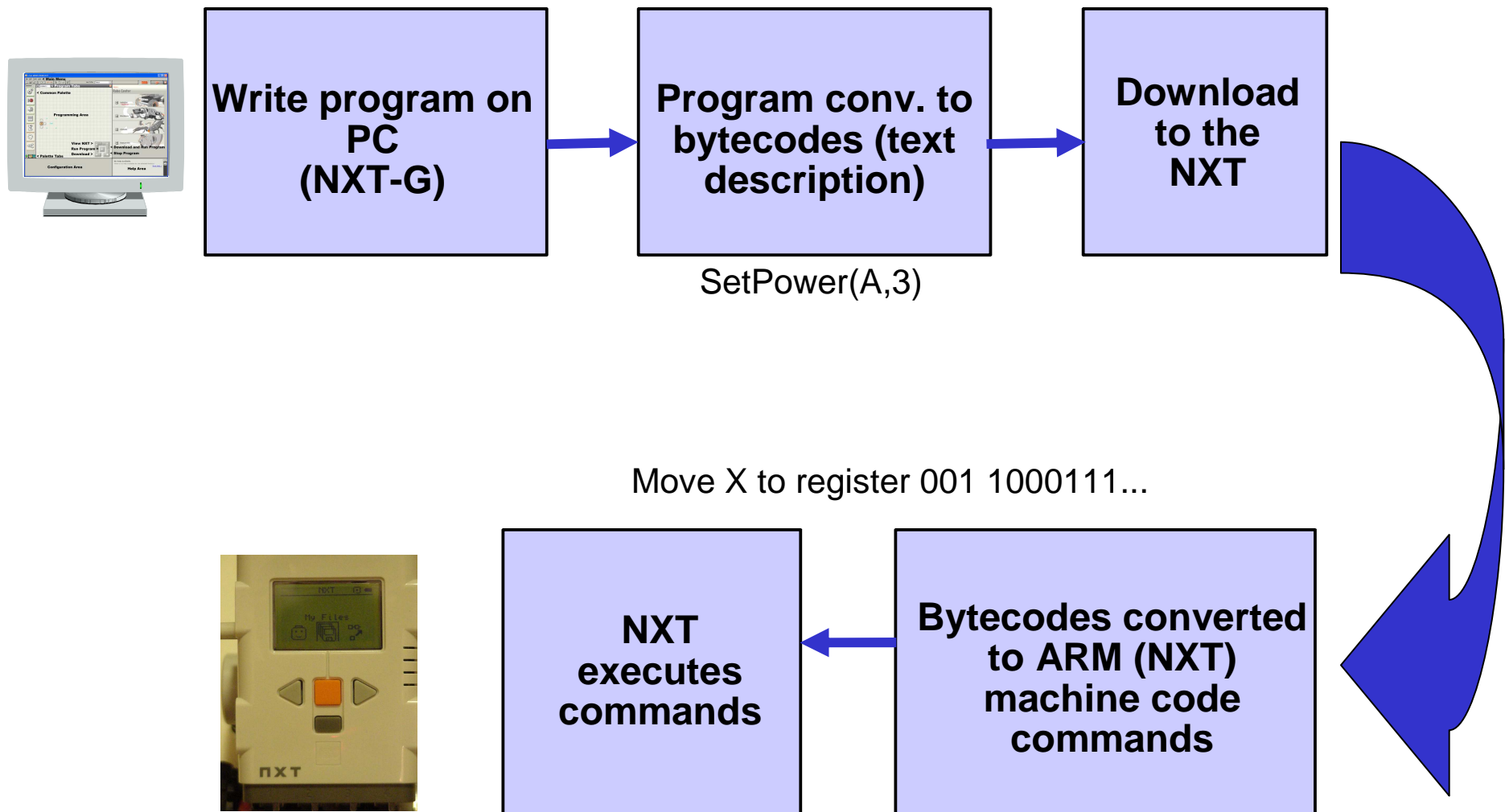
RoboLab

- Runs on MAC or PC.

RIS

- Only runs on a PC

Running a computer program on NXT



About the NXT...

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



- **Bluetooth Communications**
 - The NXT supports Bluetooth communications facilitating wireless program download.
 - Bluetooth should be disabled during competition.

About the NXT...

Batteries

- 6 size 'AA'.
- Can't lose firmware.
- NiMH rechargeable batteries work.
NiCads don't.
- Lithium rechargeables come with the
FLL Mindstorm kits.
- Avoid stalling the motors, it drains batteries.

The Programming Environment

Opening Workspace

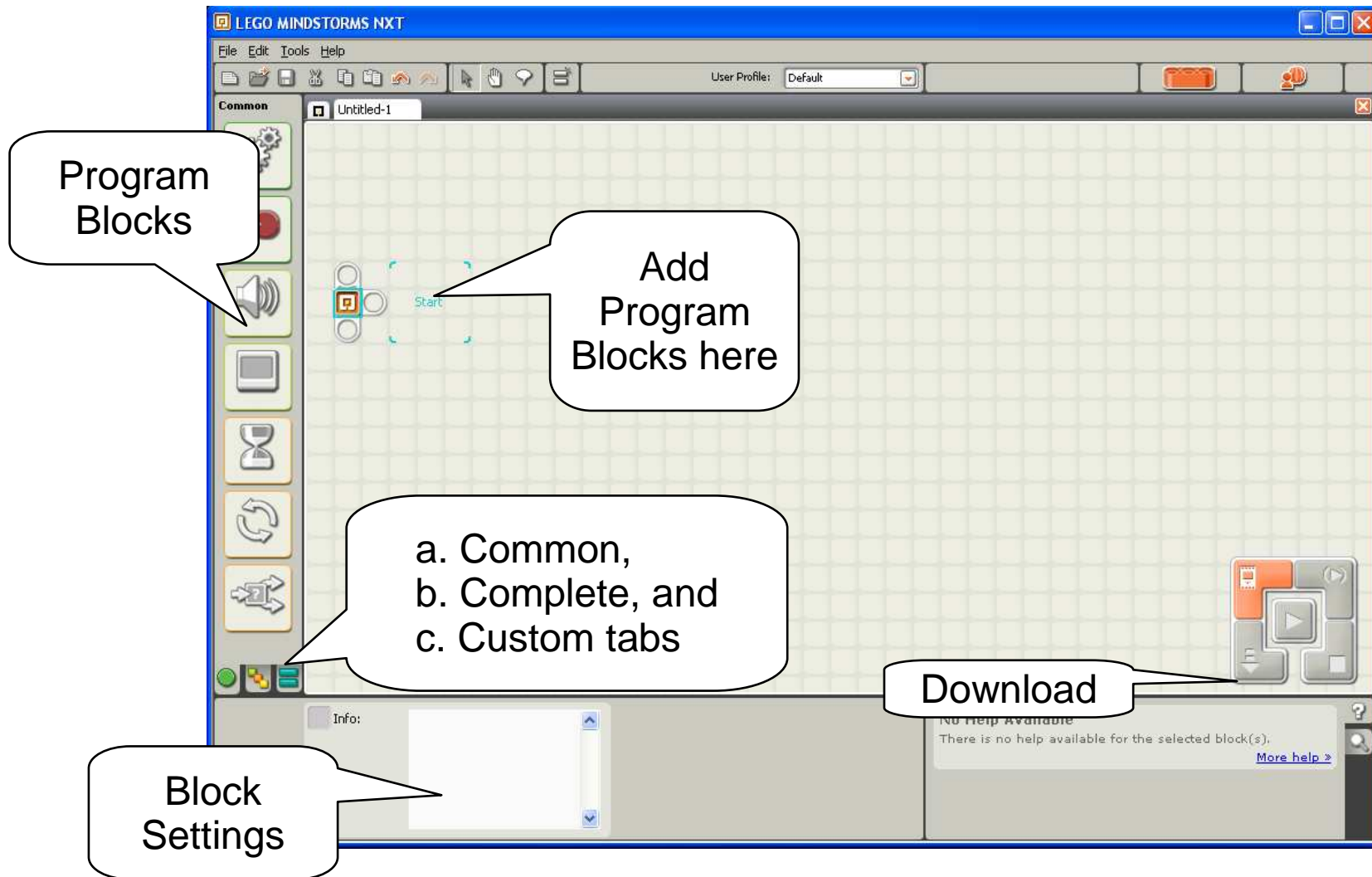
The screenshot shows the LEGO MINDSTORMS NXT software interface. The window title is "LEGO MINDSTORMS NXT". The menu bar includes "File", "Edit", "Tools", and "Help". The toolbar contains various icons for file operations and navigation. A "User Profile" dropdown menu is set to "Default". The main workspace is divided into several panels:

- Left Panel:** A vertical toolbar with icons for "Complete", "Open Program", and other functions.
- Center Panel:** A large area with a yellow background. It features two sections: "Getting Started" (with a video player icon) and "Software Overview" (with a magnifying glass icon). Below these are navigation arrows and buttons for "Start New Program" (with a text input field containing "Untitled-3" and a "Go >>" button) and "Open Recent Program" (with a dropdown menu showing "LineFollower" and a "Go >>" button).
- Right Panel:** A "Robo Center" section with a "Home" link. It lists categories: "Vehicles *Quickstart", "Machines", "Animals", and "Humanoids", each with a corresponding image and a plus sign icon.
- Bottom Right Panel:** A "No Help Available" message: "There is no help available for the selected block(s)." with a "More help >" link.

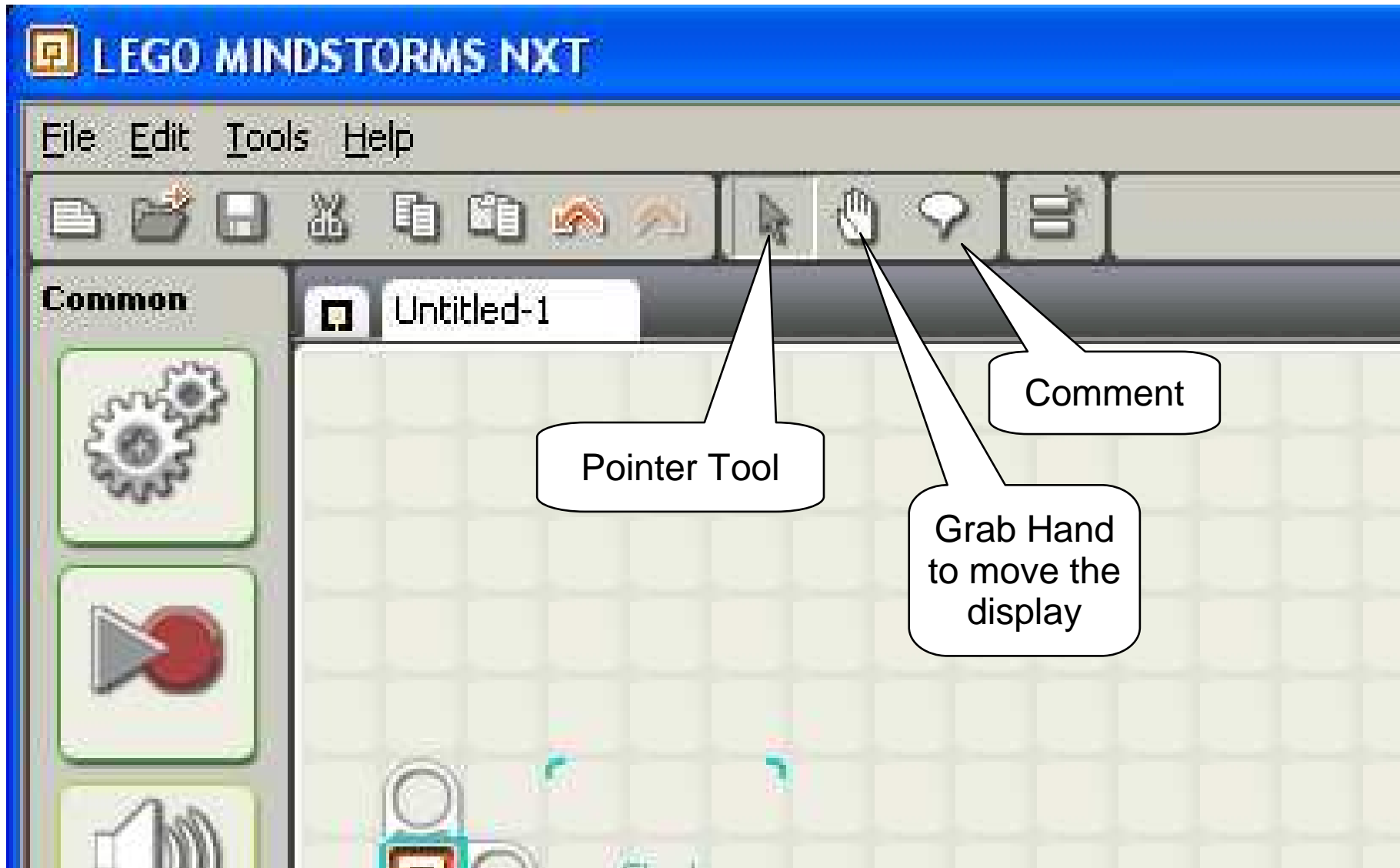
Callouts point to specific features:

- "Profiles" points to the "User Profile" dropdown menu.
- "Pre-built Robots, Programs, and Challenges" points to the "Robo Center" category list.
- "Select a Program Name" points to the "Start New Program" text input field.
- "Help and Zoom Panel" points to the "No Help Available" message.

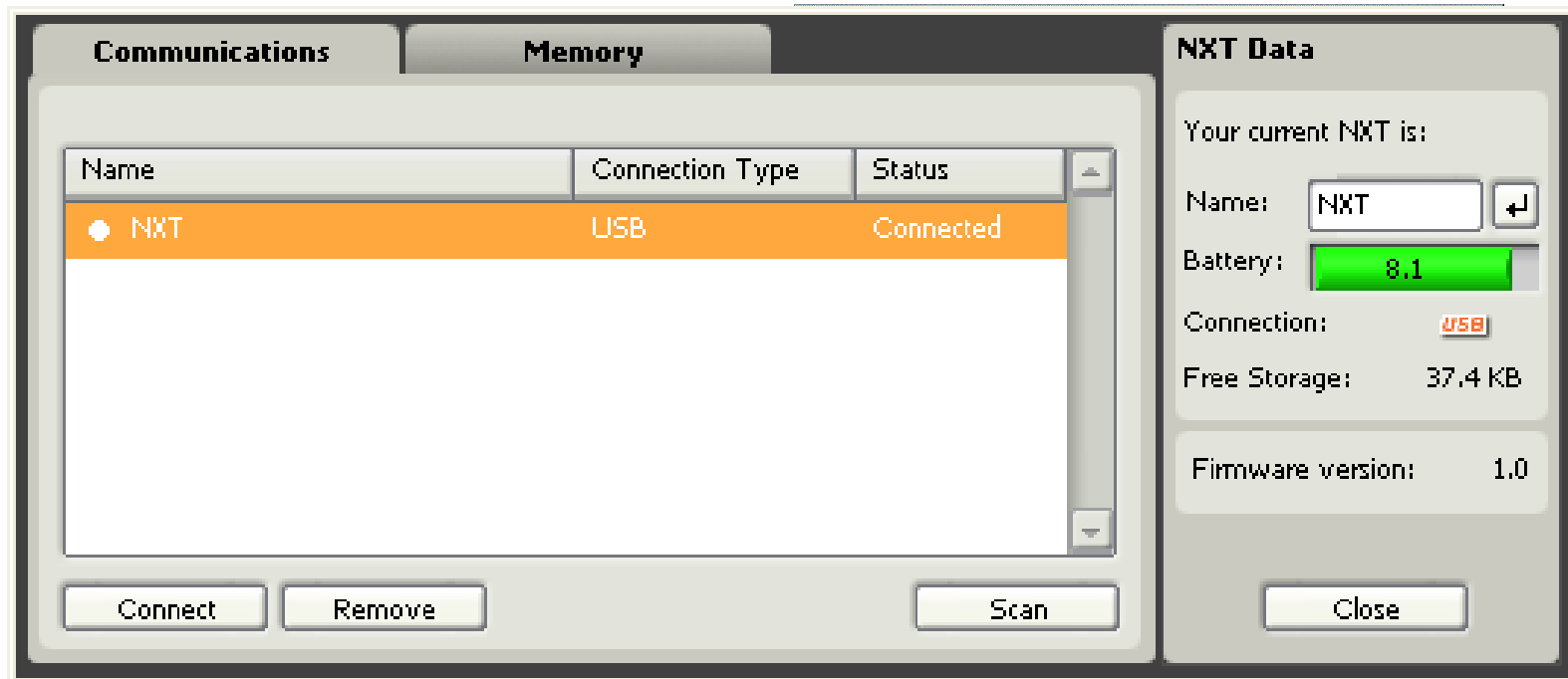
NXT-G Work Space



NXT-G Work Space



Communications



Functions when connected by USB cable or Bluetooth:

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

Memory

The screenshot displays the 'Memory' tab of the NXT software interface. It is divided into three main sections: 'NXT Memory Usage', 'File List', and 'NXT Data'.

NXT Memory Usage: A vertical bar chart shows the distribution of memory. The top portion is labeled 'Unused' (grey), the middle is 'Other' (orange), and the bottom is 'Program' (purple). A 'Delete All' button is located below the chart.

File List: A table with columns 'Name' and 'Size' lists the files on the NXT. The 'TestPivot' file is selected and highlighted in orange.

Name	Size
TestPivot	5.9 KB
MbPivotBTest	5.3 KB
UsRadarCont2	10.0 KB
Nest3	12.8 KB
SensorDisplay3	8.1 KB

Buttons for 'Upload', 'Download', and 'Delete' are located below the file list. A 'Show System Files' checkbox is checked at the top right of the list area.

NXT Data: This section provides system information: 'Your current NXT is:' with a name field containing 'NXT_DJF', a battery level indicator showing '7.4', a connection status icon, 'Free Storage: 87.5 KB', and 'Firmware version: 1.03'. A 'Close' button is at the bottom right.

Functions - Select, then delete Programs, Sounds, Graphics, and Unused files. Up to 130Kb of free of storage is available the NXT.

Basic Programming Steps

Generic Problem Solving Process

1. Define the problem or task to be completed.
2. Brainstorm solutions.
3. Discuss alternative solutions and choose one.
4. Try (implement) one.
5. Evaluate results and modify as necessary.

About Computer Programs...

Computer programs are usually a series of very simple, specific instructions executed in a sequence.

Break your solution down to individual steps.

- Go forward 18 inches.
- Stop for 1 second.
- Turn right 90 degrees.
- Go forward until the wall is reached.
- Stop.
- Raise the accessory arm 90 degrees.

A Program is like a Work Instruction

Imagine telling someone in words how to make a peanut butter and jelly sandwich.

Try it.

Now, how could you make your instructions clearer?

The same happens with computer programs.....

They must be clear.

Simple Ways to Express Programs

In the real programming world there are many ways to do this.

In the FLL world, probably the 3 best ways:

- Simulate the action by manipulating the robot by hand.
- Draw rough diagrams.
- Literally act it out.

Basic Programming Steps

1. Define the basic task to be completed.
2. Create a map of where the robot goes and what it does.
3. Describe, step-by-step, what the program should do.
4. Write the code.
5. Test, and fix, little pieces at a time.

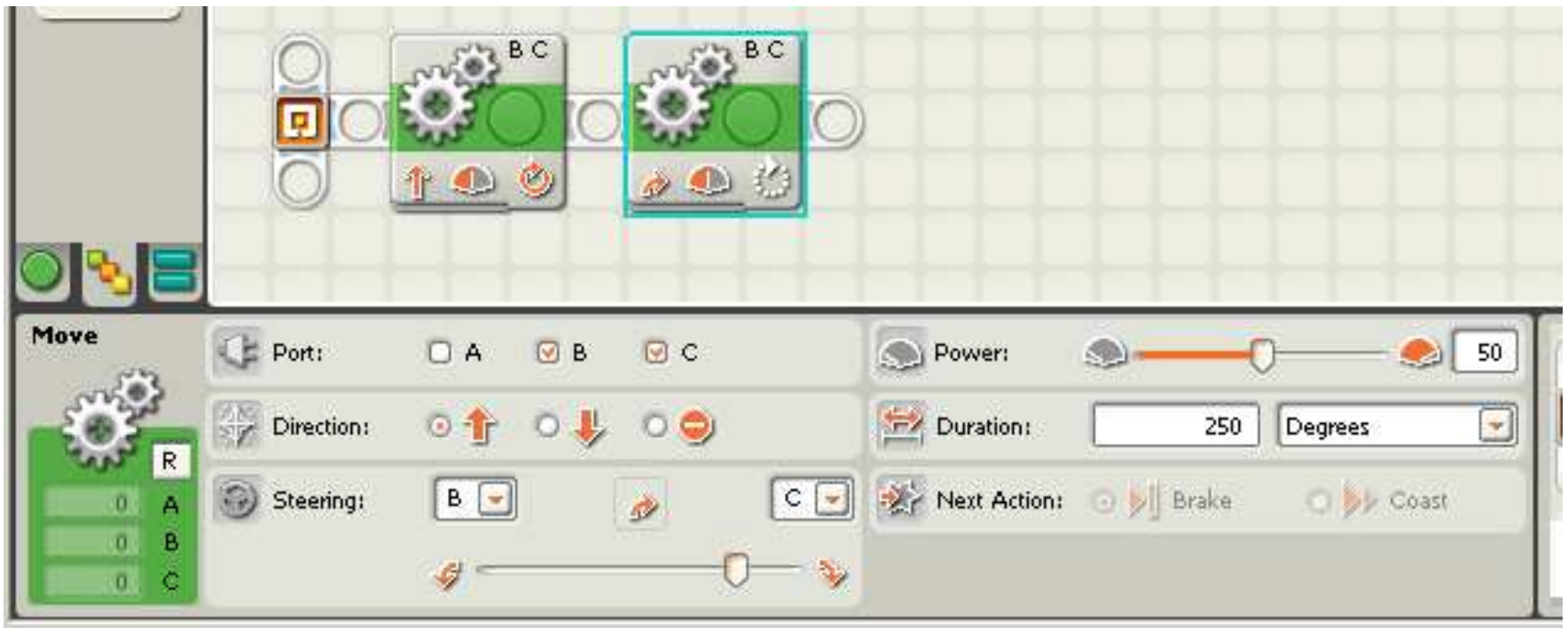
Example

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

1. Set direction and power of motors.
2. Turn on motors.
3. Stop motors, turn right.
 - Run left motor while holding right motor stopped.
 - Run left motor forward and right motor in reverse.
4. Stop motors.

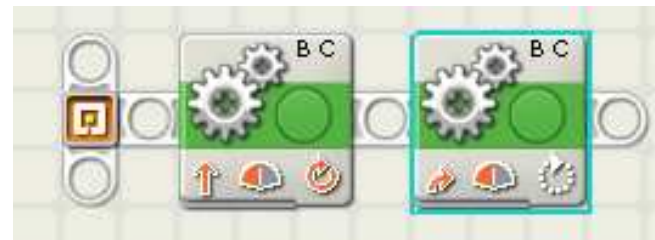
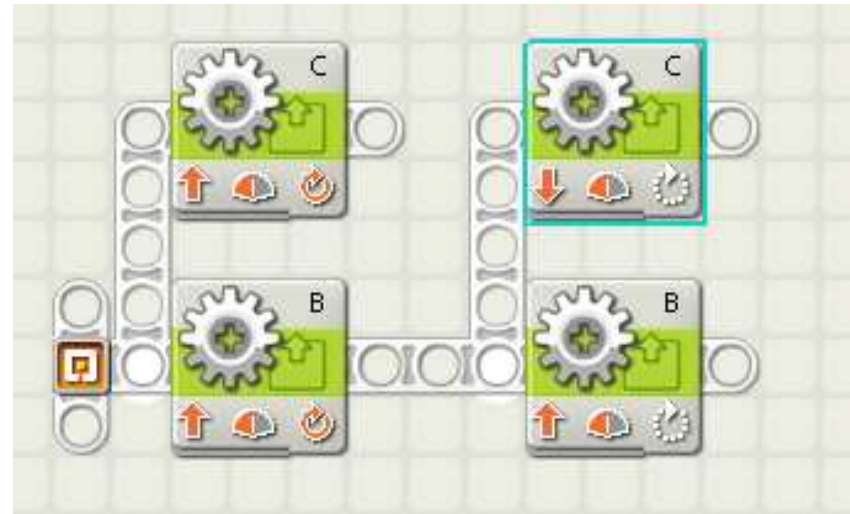
Conversion to a Program

- Forward: Motors A and C forward
- Spin: Motor A 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.

Analysis Tips

- Literally walk through it.
- Ask lots of questions.
- Do little pieces at a time.
- Reuse pieces that work from the current program or other programs.
- Feel confident in your plan before starting to program it.

Common Situations

Not sure which solution is better?

- Try all solutions out.
- Determine the benefits and drawbacks of each.
- Rank and select.

Can't think of all the steps needed?

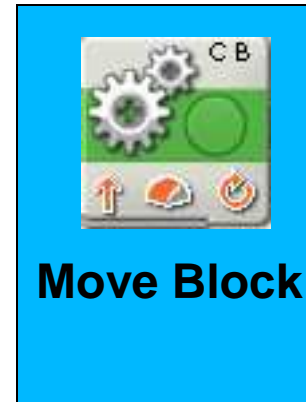
- Program and test the steps you understand.
- Reverse Process Engineer.... i.e. Start at the end and work backwards, step by step, capturing each action and decision.

Common Blocks

Common Blocks

Blocks have many functions:

- Move
- Wait for input
- Display value



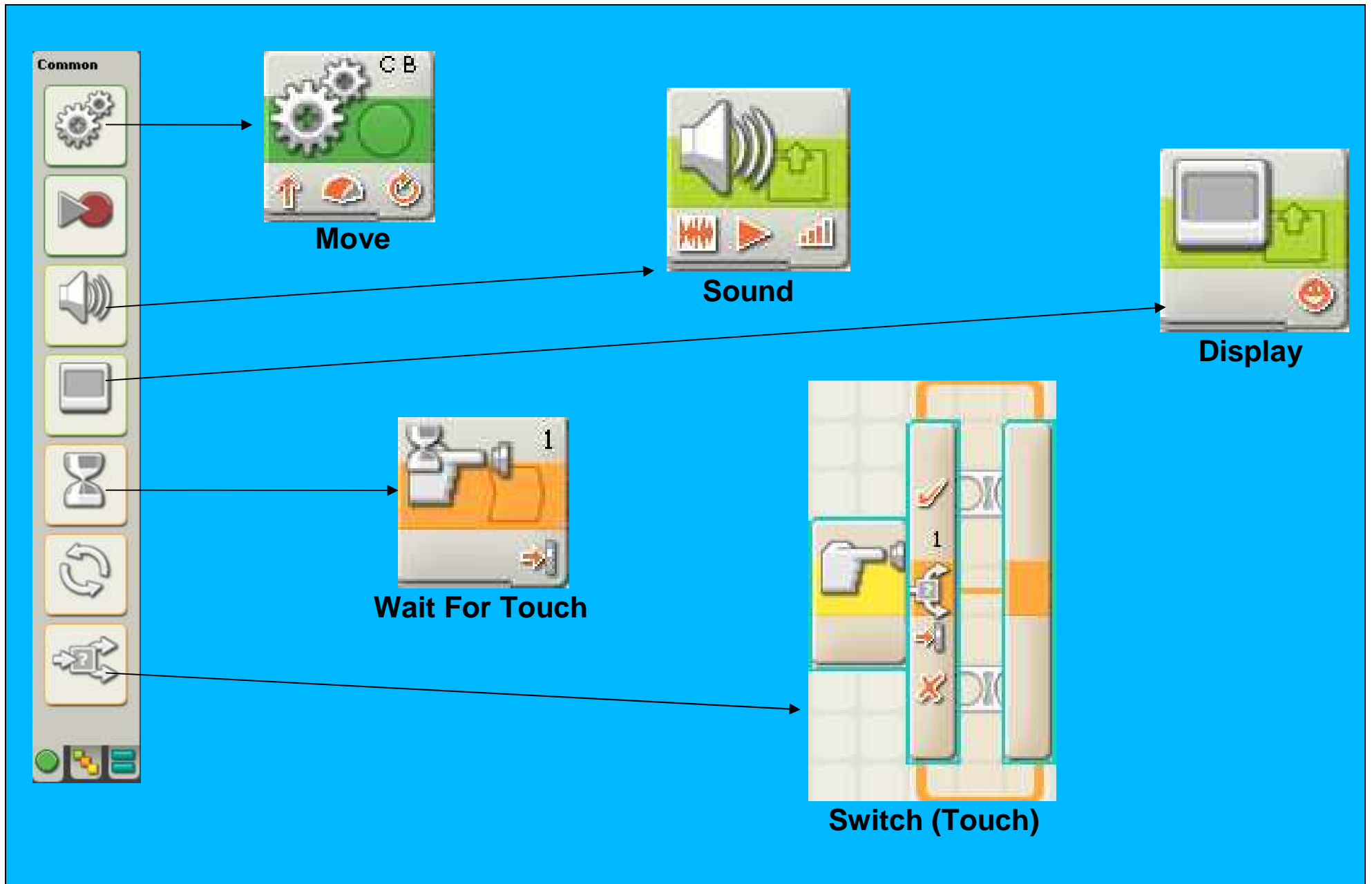
With many modifiers

- Direction, steering, distance, motors used .
- Wait for light sensor, light threshold, sensor port,



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

Port: A B C

Direction:

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

WaitFor Blocks



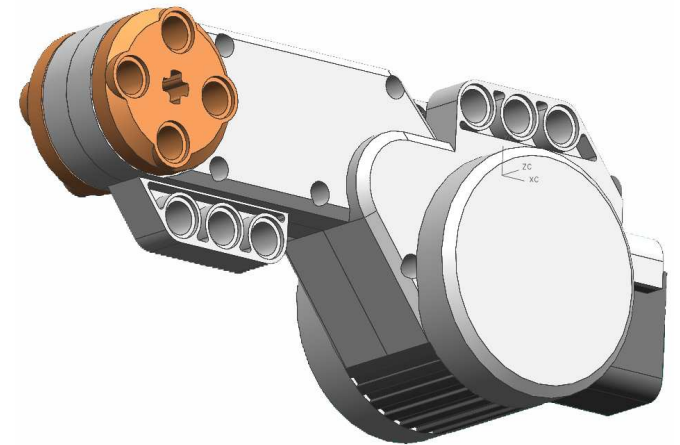
Click on the hourglass.
Click on a *Wait For Block*.
Select:

- *Time*
- *Touch*
- *Light*
- *Sound*
- *Distance (Ultrasonic sensor)*

Motors

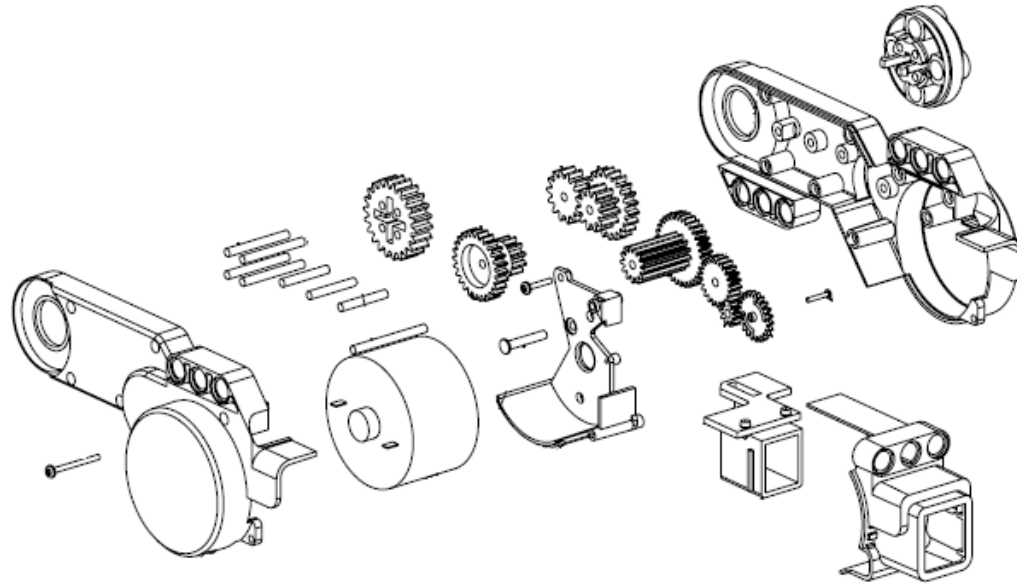
The Lego 9 volt geared motor

- Making the motors turn is the most important **output** of your program.
It makes your creation a robot!
- Without load, motor shaft turns at about 150 rpm.
- Servo sensitive to 1 degree.
- A typical robot runs 3-4 hours on a single set of batteries.



FLL allows up to 3 motors.

Motor Details



Motor can be set to different power settings

- Power levels 1-100
- Power is adjusted by Pulse Width Modulation

Turning the power setting up higher essentially makes the shaft turn faster.

Using the Move Block

Ports A, B, and/or C

Power 0-100%

Duration
Time, Degrees, Rotations, Forever

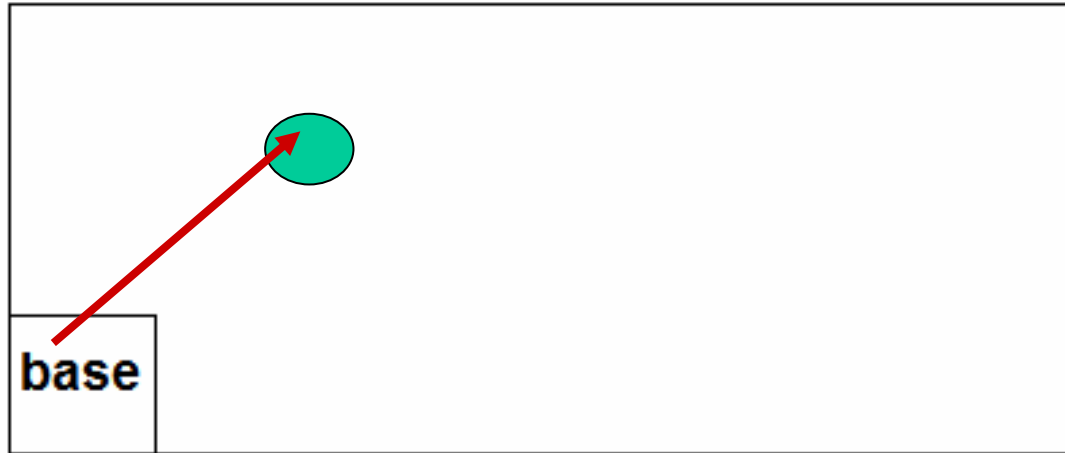
Forward, Backward or Stationary

Steering
Spin, Pivot, Arc, Straight

Brake or Coast

The image shows a screenshot of the LEGO Mindstorms software interface, specifically the 'Move' block. The block is highlighted with a blue border. It features several settings: 'Port' with checkboxes for A, B, and C; 'Direction' with icons for forward, backward, and stationary; 'Steering' with a dropdown menu and a steering wheel icon; 'Power' with a slider set to 75; 'Duration' with a text input field set to 1 and a dropdown menu for 'Rotations'; and 'Next Action' with radio buttons for 'Brake' and 'Coast'. Callout boxes provide detailed explanations for these settings.

Workshop Task #1 – Rotation Sensor Navigation



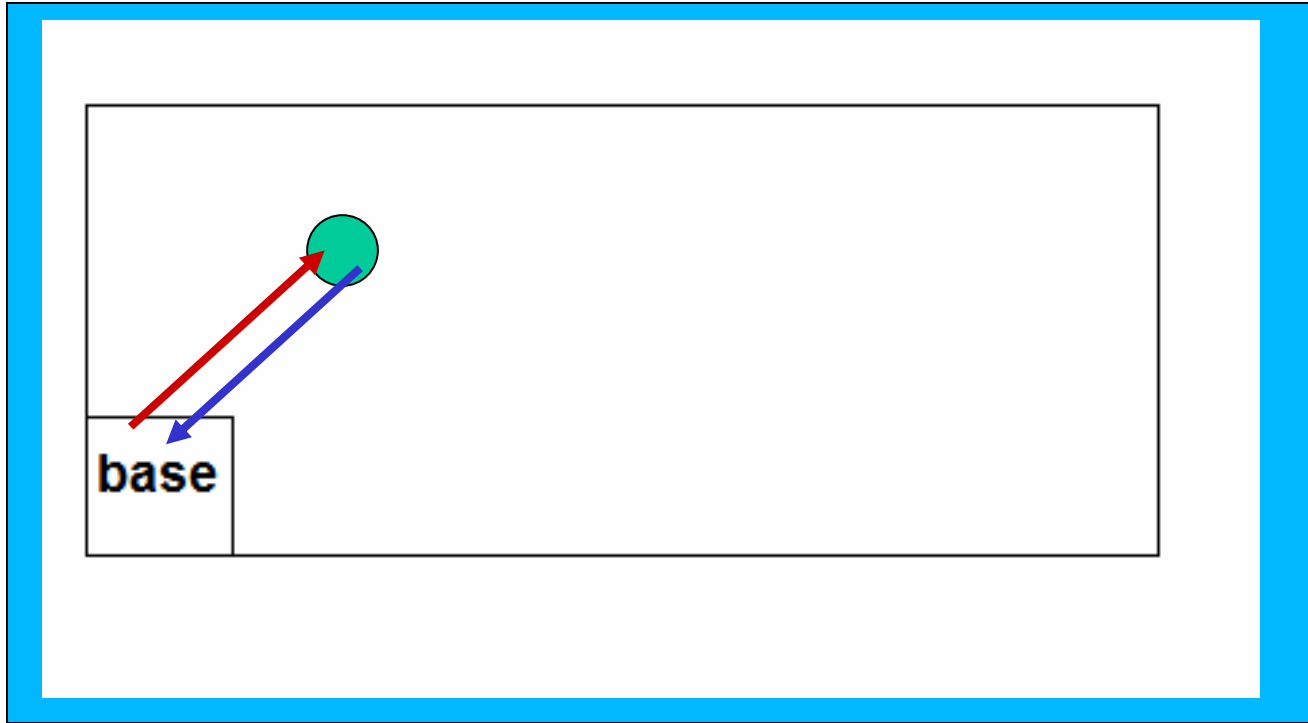
MISSION:

- Navigate robot from base to pizza plate and stop.

Workshop Task #1 - An Answer

[Switch here to show actual code on screen]

Workshop Task #2 – rotation sensor navigation



MISSION:

- Navigate robot from base to pizza plate and stop.
- Rotate robot 180 degrees.
- Return to base and stop.

Workshop Task #2 - An Answer

[Switch here to show actual code on screen]

Sensors

Allow your robot to detect the real world.

- Touch
- Light
- Sound (Microphone)
- Distance (Ultrasonic)
- Rotation
- Time

Touch Sensor



- Detects touching or bumping into something.
- Good for detecting walls and immovable or large objects.

Pressed, Released, Bumped

The image shows a configuration panel for an event system. It is divided into two main sections: "Port" and "Action".

Port: This section contains four circular indicators labeled 1, 2, 3, and 4. Indicator 1 is selected, indicated by a red dot in the center.

Action: This section contains three options, each with a circular indicator and a corresponding icon:

- Pressed:** The indicator is selected (red dot). The icon shows a button being pushed.
- Released:** The indicator is not selected. The icon shows a button being pulled back.
- Bumped:** The indicator is not selected. The icon shows a button being pushed and then pulled back.

A speech bubble points to the "Bumped" option with the text: "Bumped: pressed and released in any order."

Touch Sensor WaitFor Block

Robot waits until sensor responds

Pressed, released, or bumped

Control: Sensor

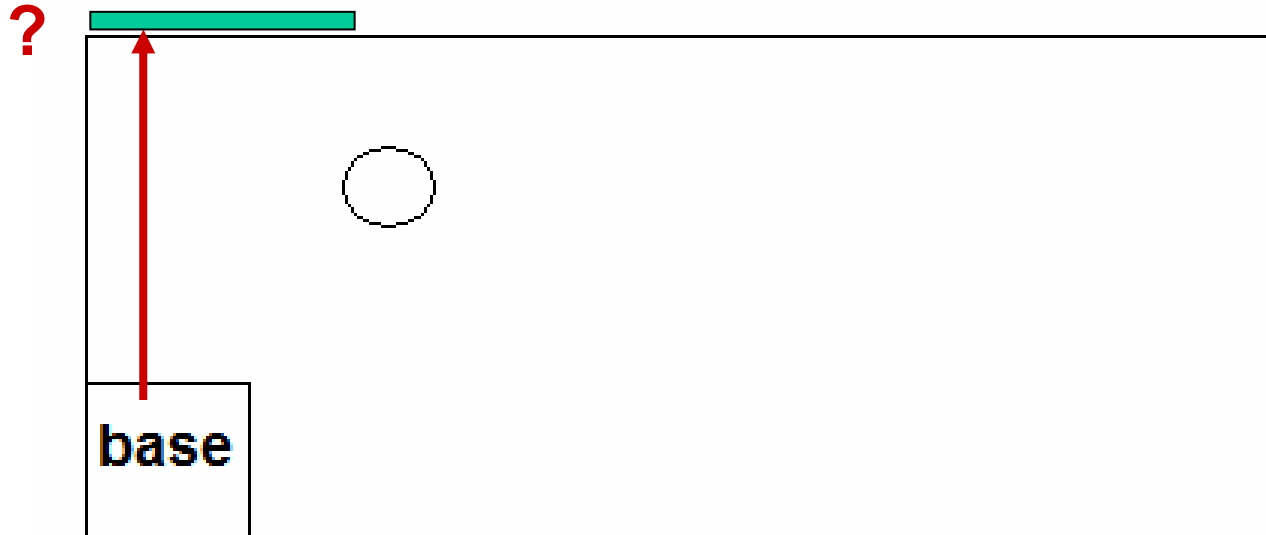
Port: 1 2 3 4

Sensor: Touch Sensor

Action: Pressed Released Bumped

The image shows a programming interface with a 'WaitFor' block. The block is highlighted with a blue border and contains a robot icon and a touch sensor icon. A callout bubble points to the robot icon with the text 'Robot waits until sensor responds'. Another callout bubble points to the 'Action' section of the block with the text 'Pressed, released, or bumped'. The 'Action' section has three radio buttons: 'Pressed' (selected), 'Released', and 'Bumped'. The 'Control' section has a dropdown menu set to 'Sensor'. The 'Port' section has four radio buttons labeled 1, 2, 3, and 4, with port 1 selected. The 'Sensor' section has a dropdown menu set to 'Touch Sensor' and a touch sensor icon.

Workshop Task #3 – Touch Sensor Navigation



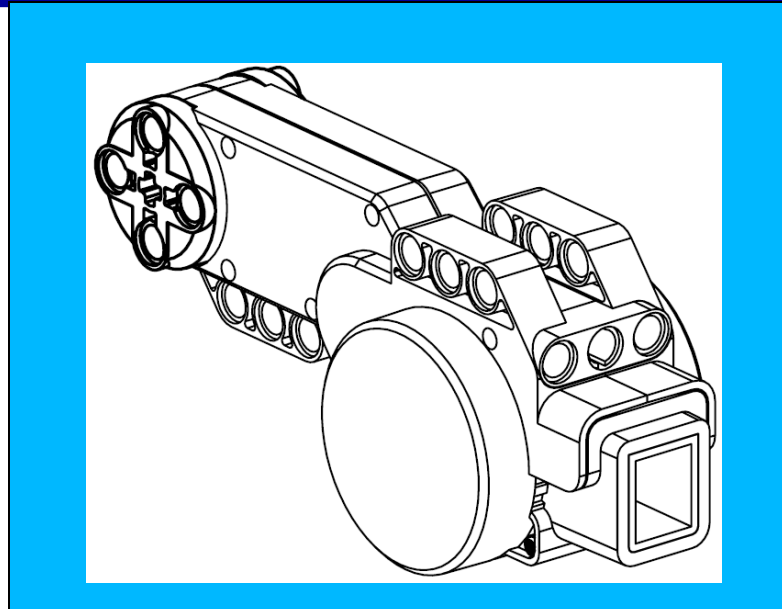
MISSION:

- Navigate to block and stop when touch sensor is depressed. Block may be a variable distance from base.

Workshop Task #3 - An Answer

[Switch here to show actual code on screen]

Rotation Sensor



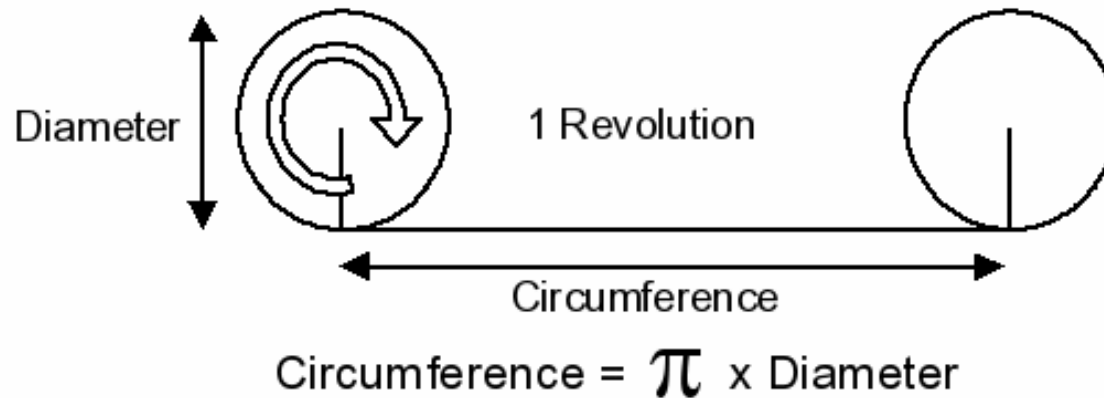
- Measures how far a rotating axle has turned. As the axle turns, a counter in the NXT is incremented or decremented.
- 360 counts per rotation.
- Each motor has an embedded rotation sensor.

Rotation: Move and Motor Blocks



Calculating Distance

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



- 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, gear slippage.

More on Rotation Sensor

- Rotation sensor **counts forward (positive) and backwards (negative).**

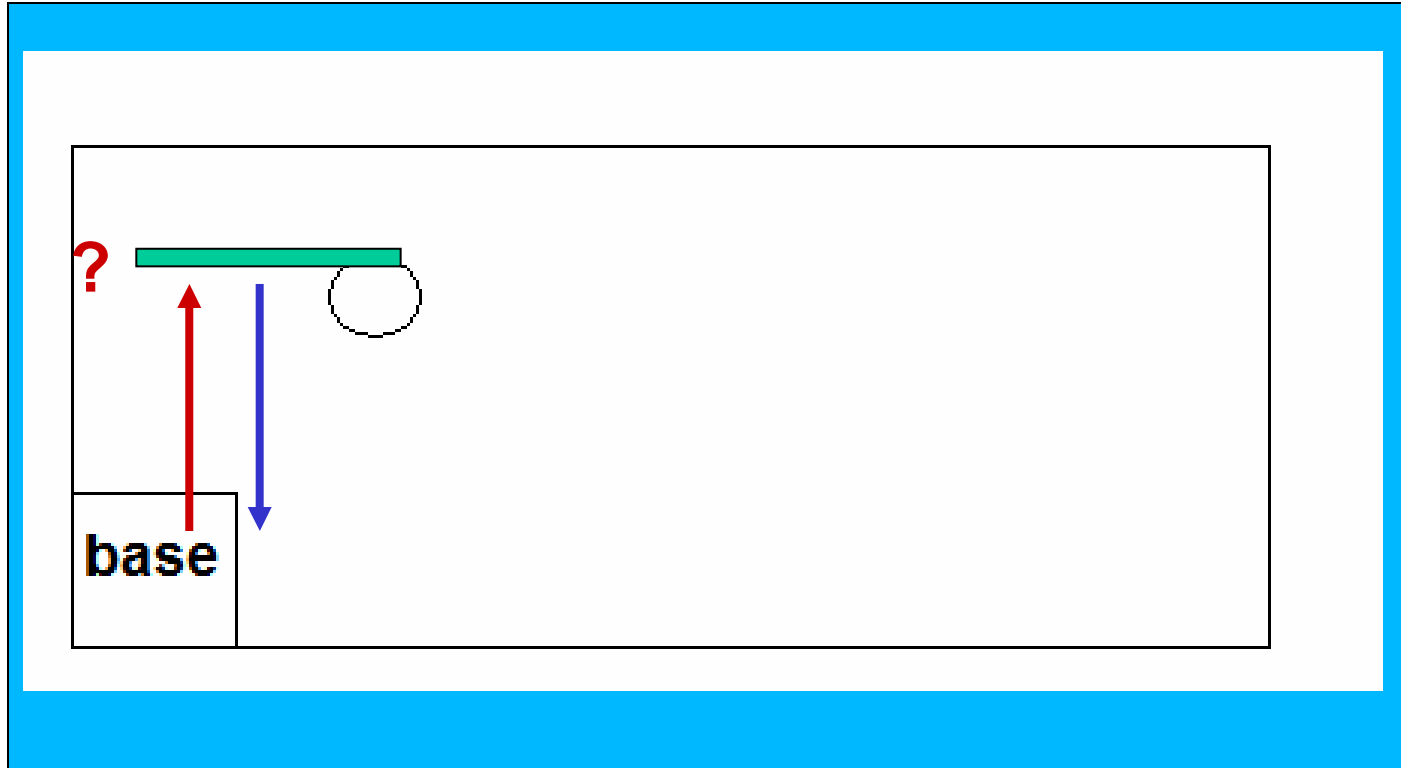
The image shows a screenshot of a programming environment, likely Scratch, with a focus on a rotation sensor block. The main workspace contains a script area with a 'when green flag clicked' block, followed by a 'rotation sensor' block (highlighted with a blue border and a callout bubble that says 'Select the Rotation Sensor'). This is followed by a 'say "a" for 3 sec' block and a 'say "a" for 3 sec' block. The rotation sensor block has a dropdown menu open, showing options for 'degrees' and 'radians'. A callout bubble labeled 'Live updates' points to the rotation sensor block's display area. In the bottom right corner, there is a 'Run' button (a play icon) with a callout bubble labeled 'Run'. The bottom panel shows the 'Rotation Sensor' block's configuration: Port: A B C, Action: Read (selected), Reset, Compare: up/down arrows, a value of 360, and a unit of Degrees. The current rotation value is displayed as 3363.

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.

Workshop Task #4 – touch sensor navigation



MISSION:

- Navigate to block and stop when touch sensor is depressed.
- Return to base and stop.

Workshop Task #4 - An Answer

[Switch here to show actual code on screen]

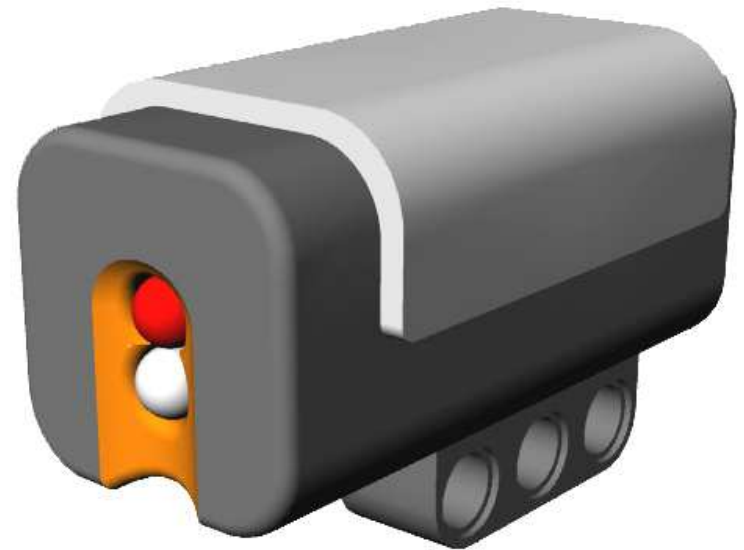
Light Sensor

Operates in "percent" mode

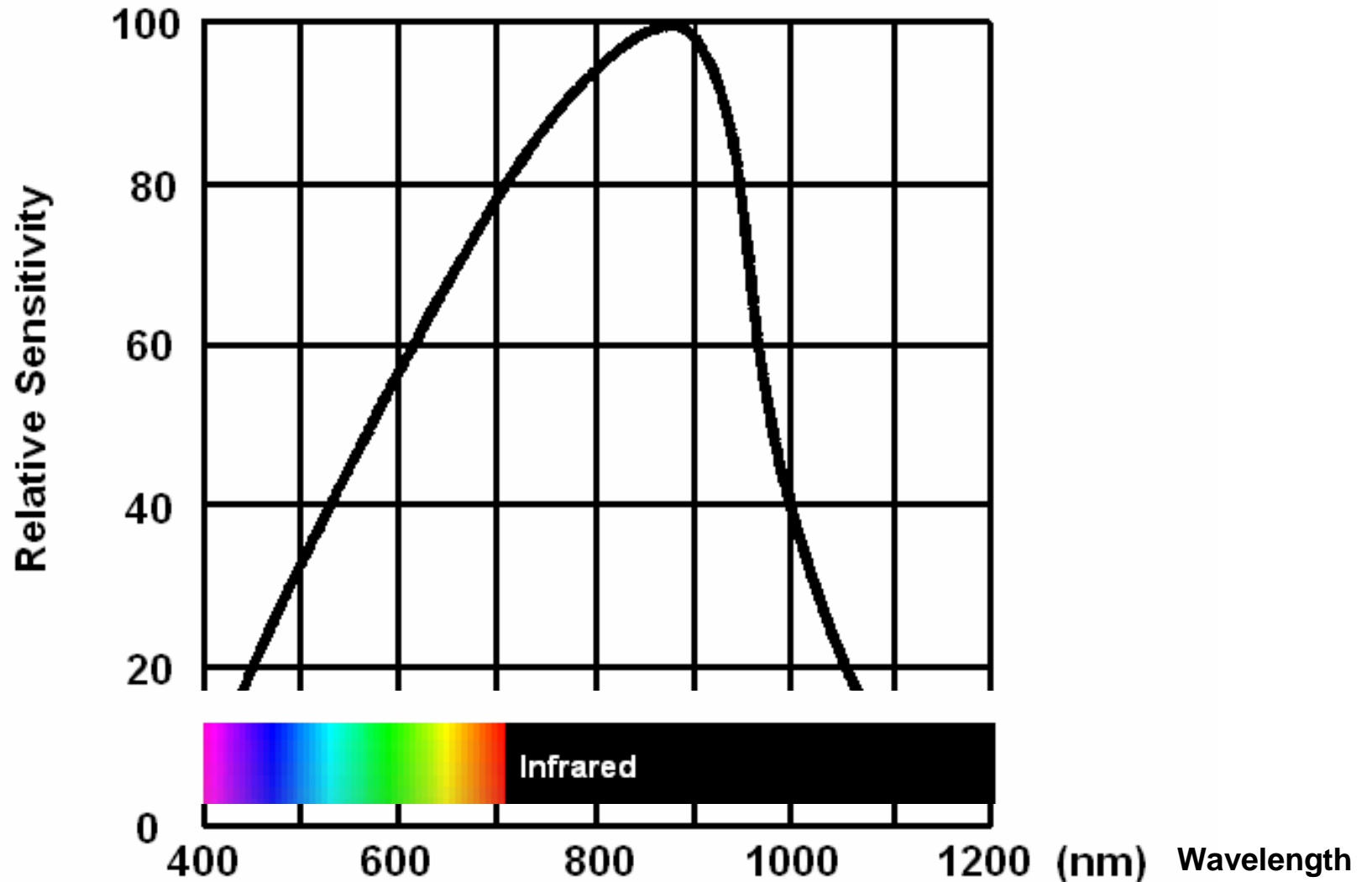
- 0 to 100
- Higher number = more light.
A lighter surface reflects more light.

Light can be turned off.

Shines a red light.



Light Sensor Spectrum

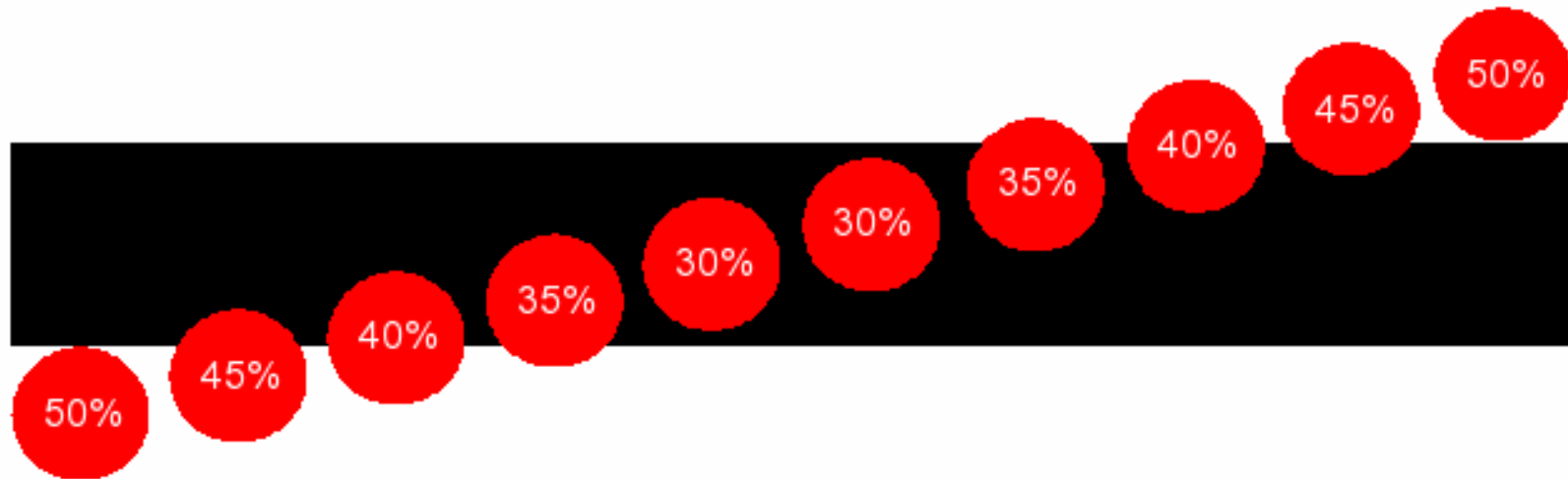


- Most sensitive to red/IR light.

Light Sensor Readings

- Lowest likely reading 5%
- Highest likely reading 100% (pointing at a light)
- Readings also depend on the color of the surface
- 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

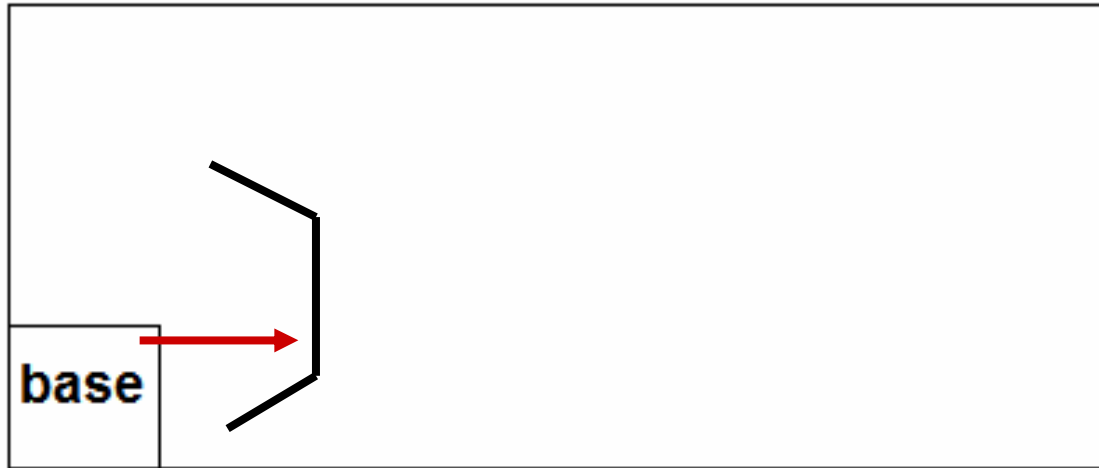
Until: [Slider]

Light: 40

Function: Generate light

Turn light on/off

Workshop Task #5 – light sensor navigation



MISSION:

- Navigate to line and stop.

Workshop Task #5 - An Answer

[Switch here to show actual code on screen]

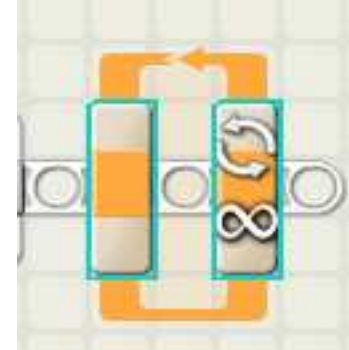
Loops

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.

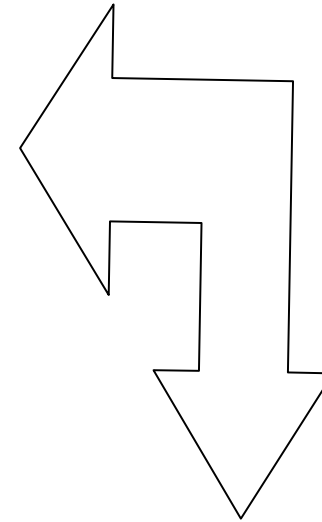
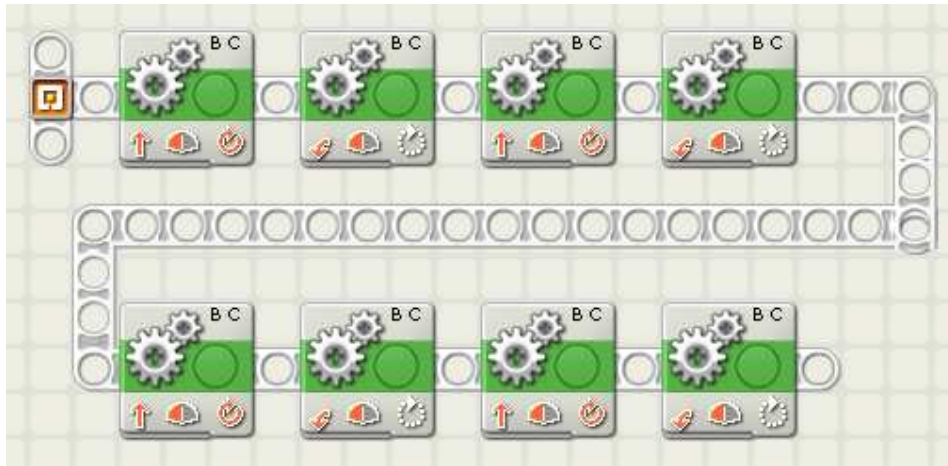


There are loops for:

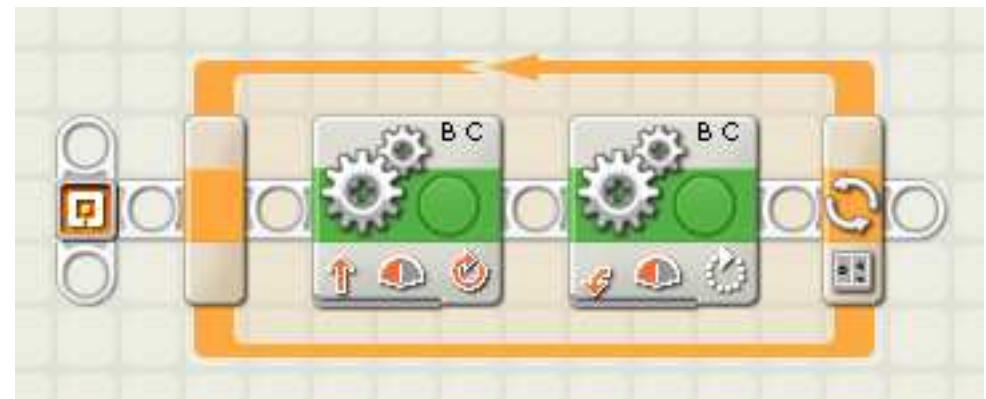
- Forever
- Every sensor (including time)
- Logic
- Count

If your algorithm says something like: “Until the sensor reads x, keep doing this”, use a loop.

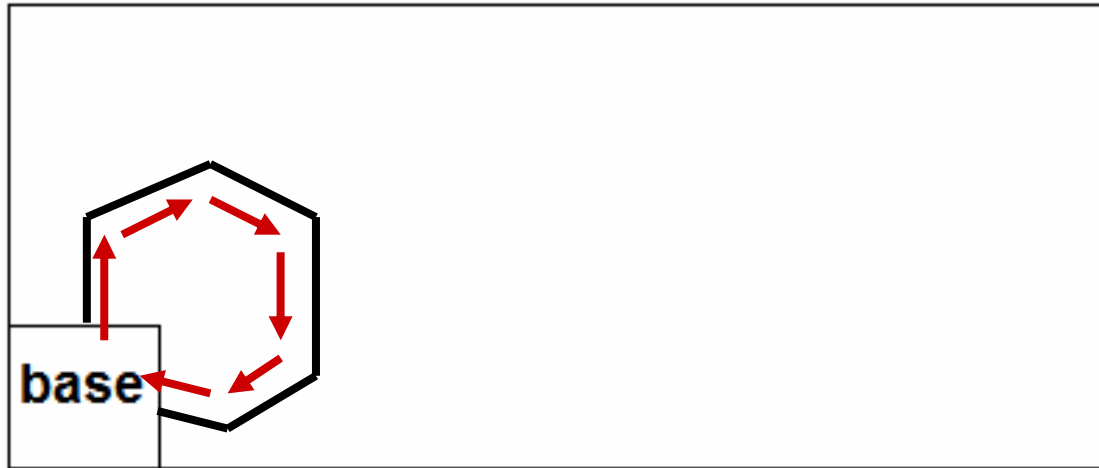
Simple Loop



- Convert this
- to something simpler using a loop



Workshop Task #6 – light sensor navigation



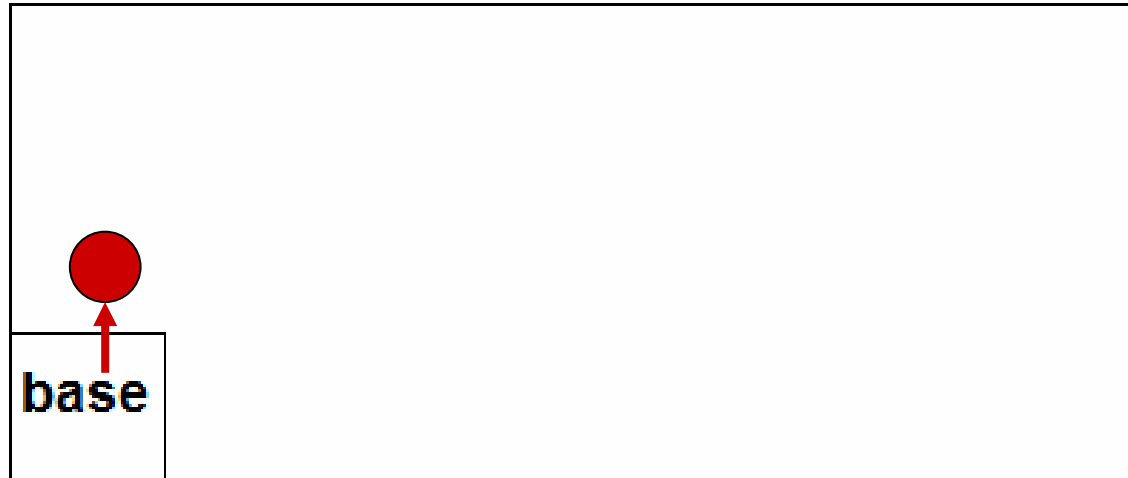
MISSION:

- Navigate around loop using light sensor and return to base.

Workshop Task #6 - An Answer

[Switch here to show actual code on screen]

Workshop Task #7 – manipulating objects



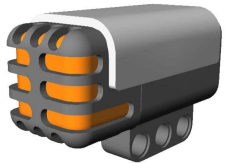
MISSION:

- Object will be positioned 6" from base.
- Robot go forward and pick-up object.

Workshop Task #7 - An Answer

[Switch here to show actual code on screen]

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

Advanced Topics

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 in different places.

- Reuse: put the common code in one place.

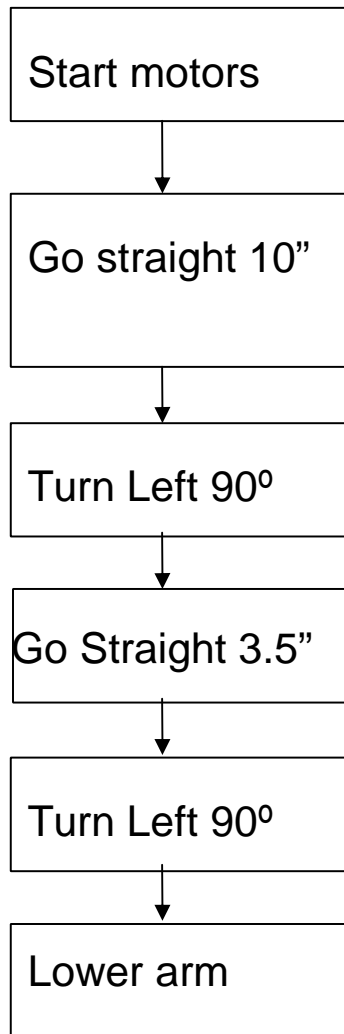
To divide a task into meaningful pieces or modules.

To hide complex details.

Real world example: *'Get ready for bed.'*

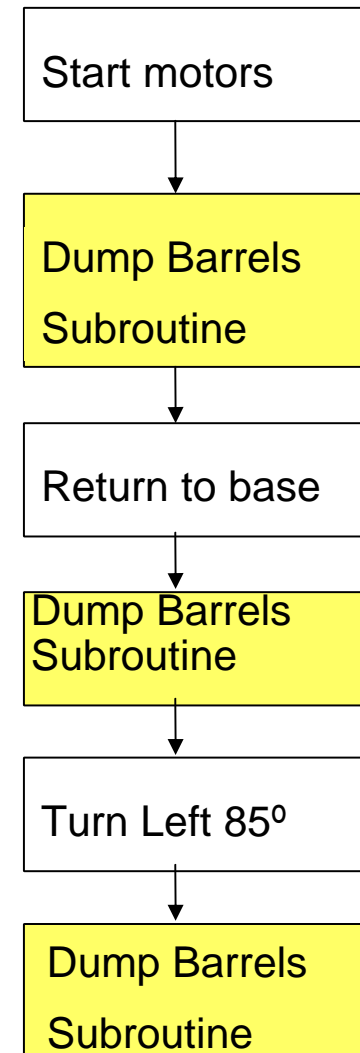
Modular Advantages

Dump Barrels Subroutine

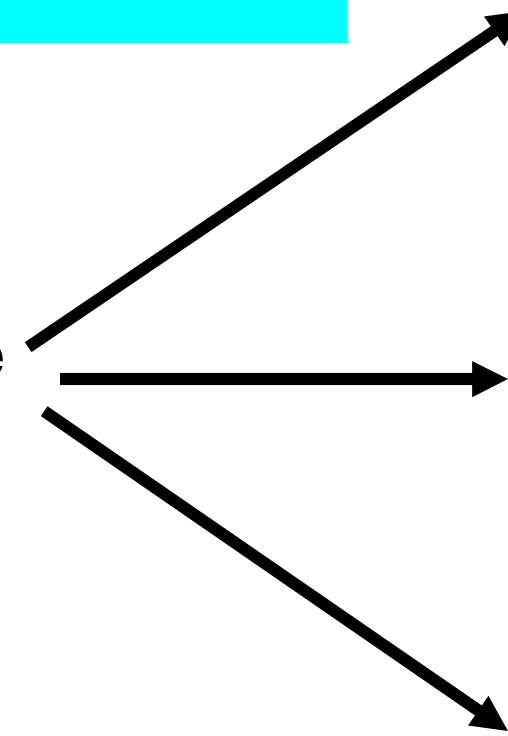


Changes made to subroutines are updated to all other programs where they are used.

Main Program



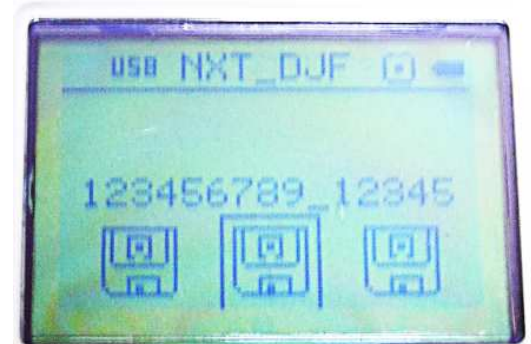
Subroutine



MyBlock Names

Assign useful and informative names.

- 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

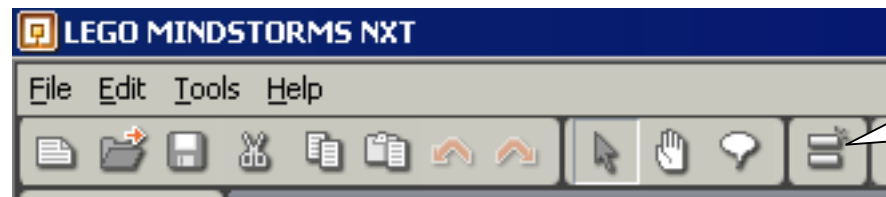


Hide all these blocks inside one MyBlock.



MyBlock Creation 1

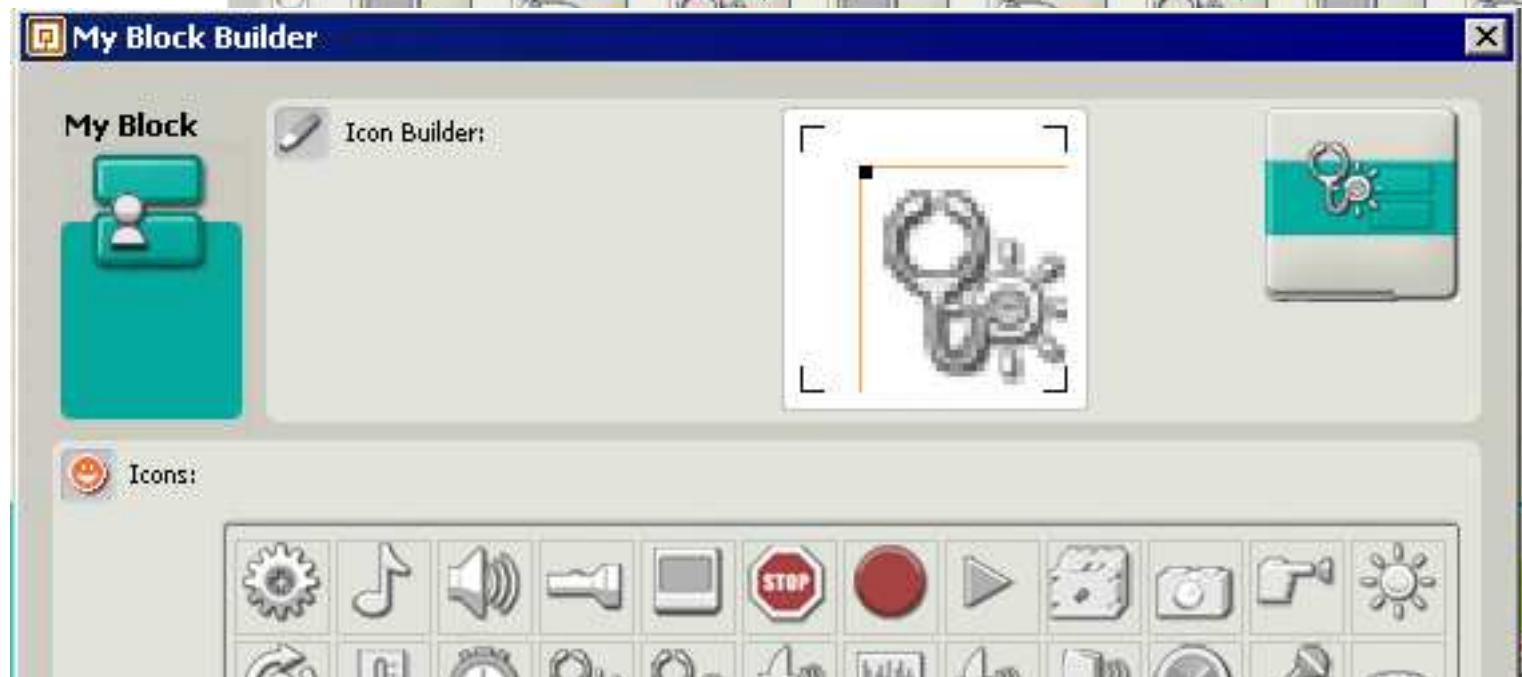
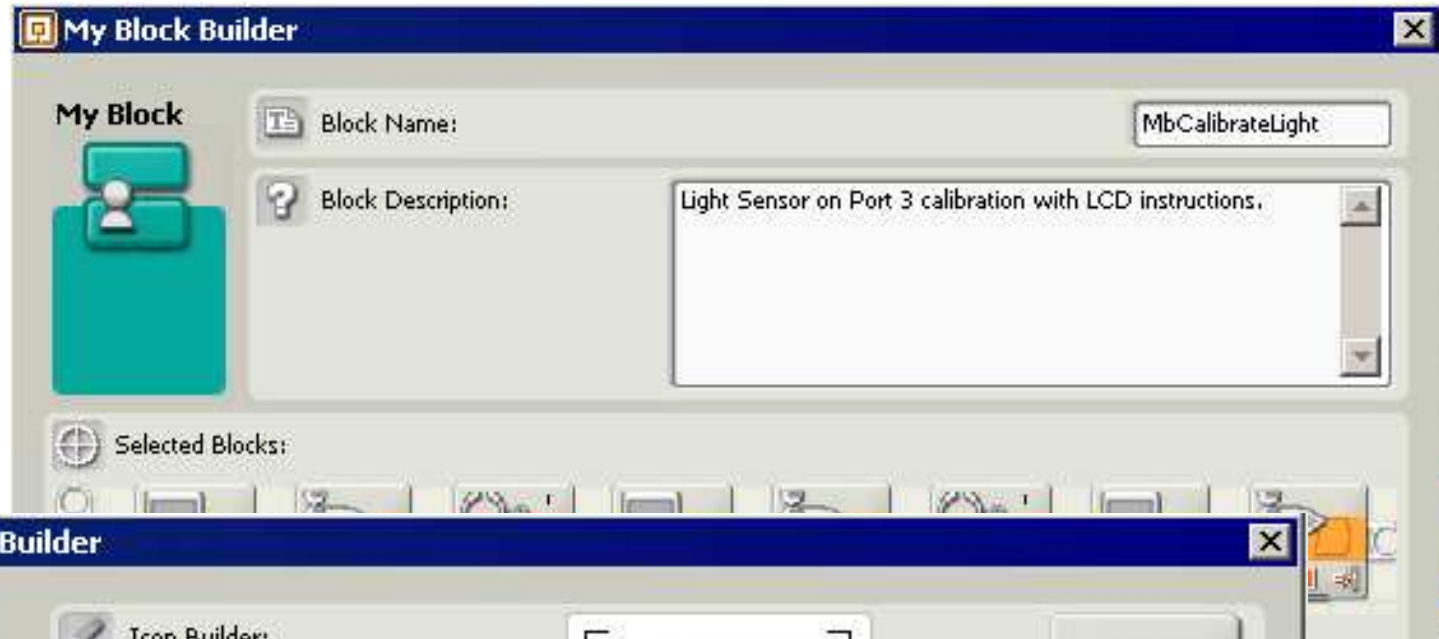
- Start with working code.
- Select blocks to include.
- Click the Create My Block button.



Create My Block button

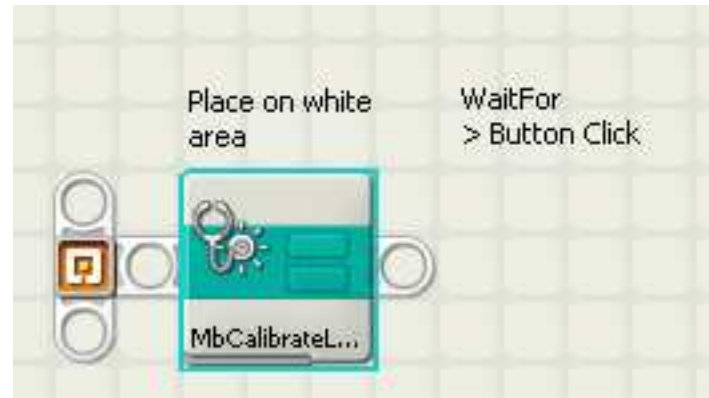
MyBlock Creation 2

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



MyBlock Creation 3

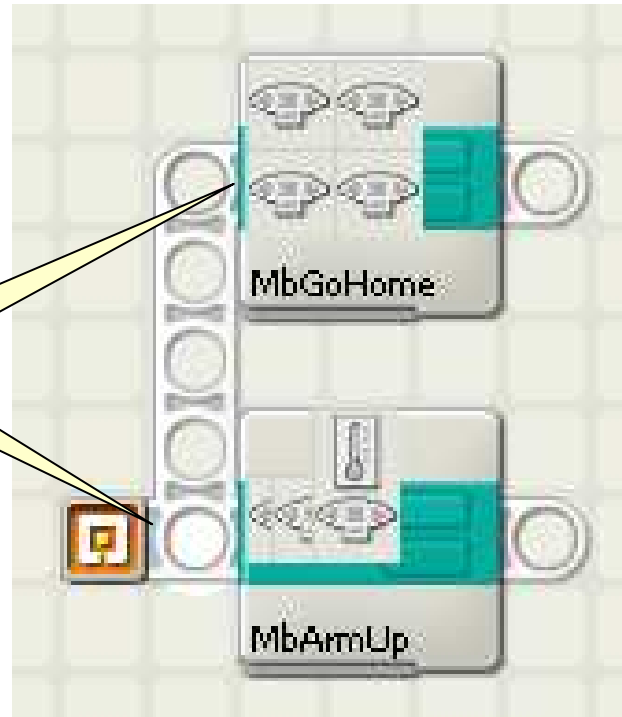
- MyBlock replaces the selected Blocks!
- To add the new MyBlock to a program, select it from the Custom Tab.



Parallel Sequences

Hold cursor here until it changes into a wiring tool.

Click and drag a sequence beam, release here.



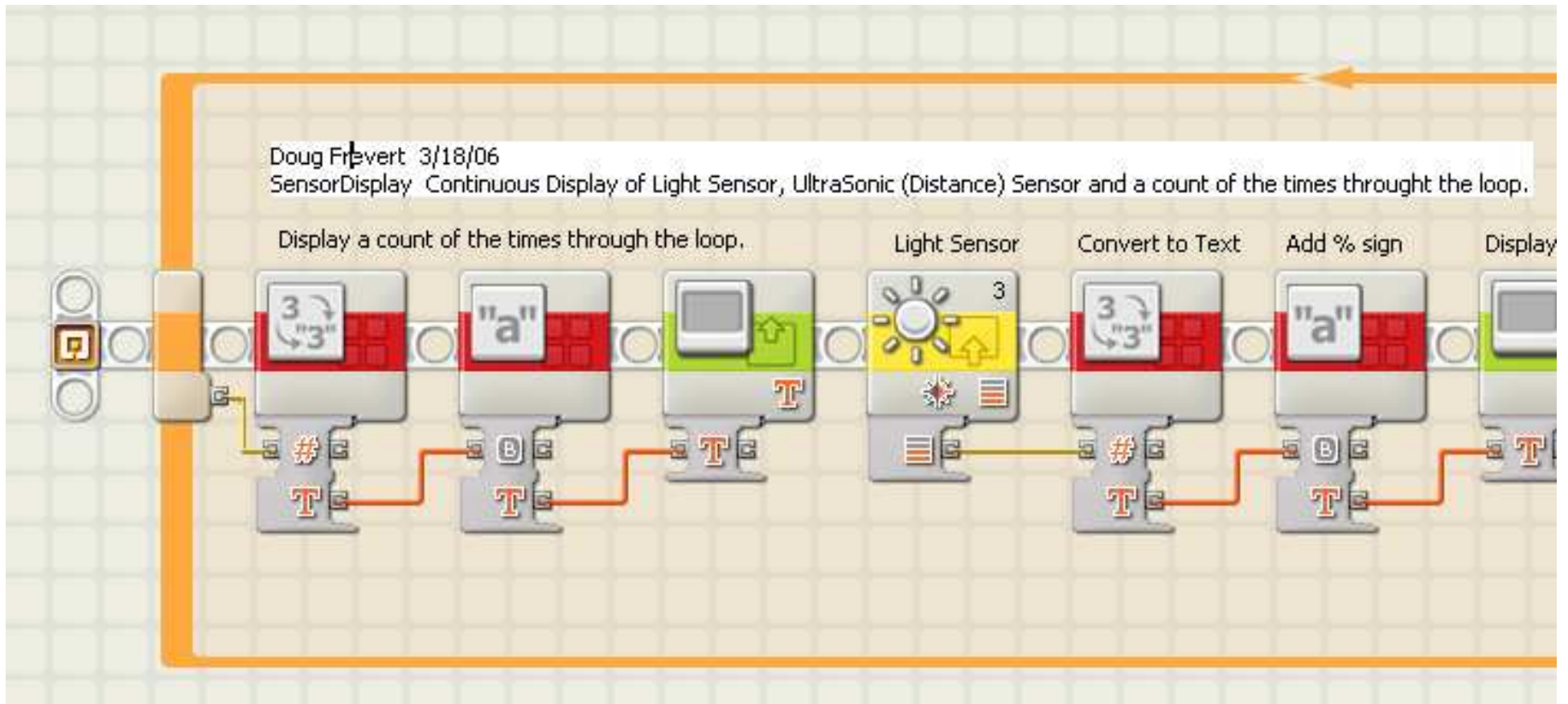
- Make two tasks that run independently. (Can you walk and chew gum?)
- One task lifts the arm. The other task heads for home.

Comments

- Comments explain the program to other programmers.
- In a team process like FLL, comments may be 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.

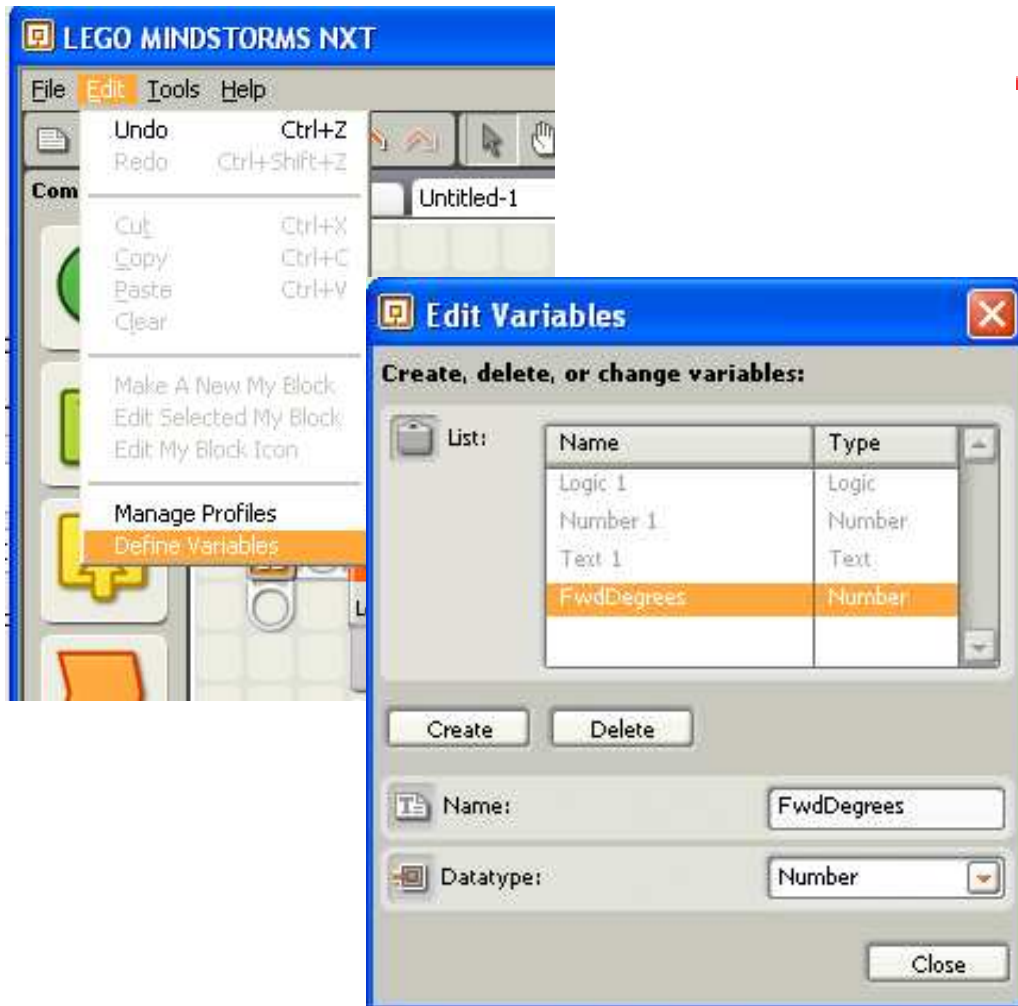
Suggestions

1. Make sure your code works before making it into a MyBlock.
2. Use subroutines to:
 - Hide complexity.
 - Modularize programs.
 - Remove duplication.
3. Use comments liberally.

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

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

Set the value

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

Action: Write

Value: 321

Using a Variable

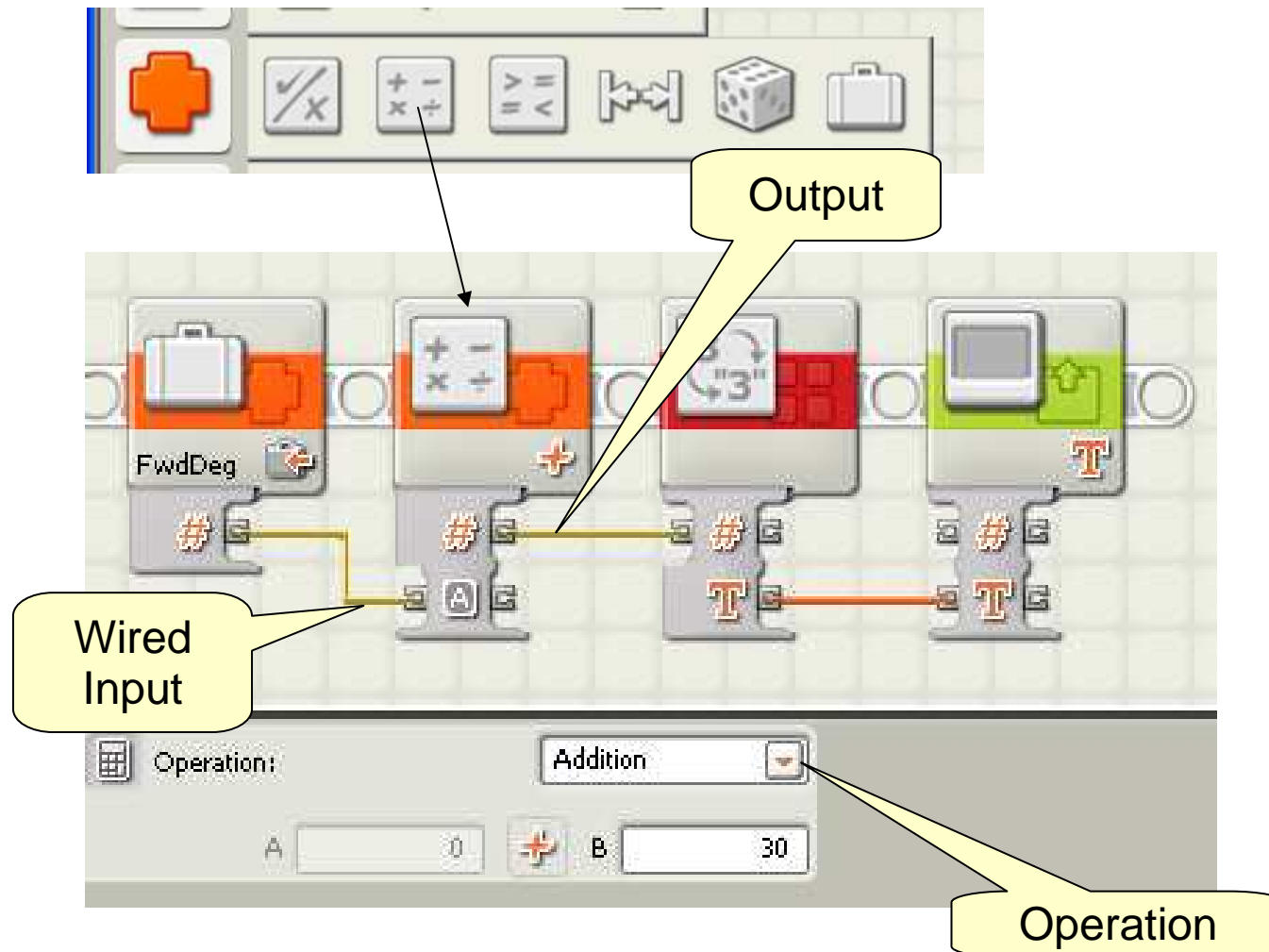
The image shows a logic editor interface with several components and callouts:

- Top Panel:** A toolbar with various icons, including a red cross, a checkmark with an 'X', a plus/minus sign, a greater/less than sign, a key symbol, a cube, and a briefcase icon.
- Main Logic Area:** A sequence of logic bricks on a grid. From left to right: a square button icon, a briefcase icon labeled 'FwdDeg', another briefcase icon labeled 'FwdDeg' (highlighted with a cyan box), and a gear icon labeled 'B'. A yellow wire connects the output of the second 'FwdDeg' brick to the input of the 'B' gear brick.
- Callout 1:** A yellow speech bubble pointing to the cyan box around the second 'FwdDeg' brick, containing the text: "Pick a variable".
- Callout 2:** A yellow speech bubble pointing to the yellow wire, containing the text: "Wire variable value to motor duration".
- Bottom Panel:** A control panel with a table and action options.

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

Below the table, there are radio buttons for "Action:" with "Read" selected and "Write" unselected. To the right is a "Value:" input field with the number "0".
- Callout 3:** A yellow speech bubble pointing to the "Read" radio button, containing the text: "Use the Read action".

Arithmetic



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.

Debugging Tools



Music / Sounds

- Use sounds to identify sections of code.



LCD

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

Workshop Task #8 – manipulating objects



MISSION:

- Manuever from base to wall using touch sensor.
- Turn 90 degrees and go forward to object.
- Pick-up object.
- Turn and return to base.

Workshop Task #8 - An Answer

[Switch here to show actual code on screen]

Putting it All Together

FLL Ace Programmer in 10 Steps

1. Create a sketch 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 plan.

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.

9. Don't be afraid to scrap everything and start over if things get too complex or fragile.

10. If coding/testing/bug fixing is driving you insane, take a break.

Dynamic Blocks

- Download dynamic block update from
 - mindstorms.lego.com/Support/Updates
- From same site also download mini blocks
 - Mini motor and mini move blocks reduce size of programs
- Download "Display Number" block for debugging from
 - www.teamhassenplug.org/NXT

Creating your own blocks with Labview

- Download software from
 - zone.ni.com/devzone/cda/tut/p/id/4435
- Manual is available on same site
- Ultimate way to customize Lego Programming
- Introduces children to a tool that is commonly used in engineering testing and development