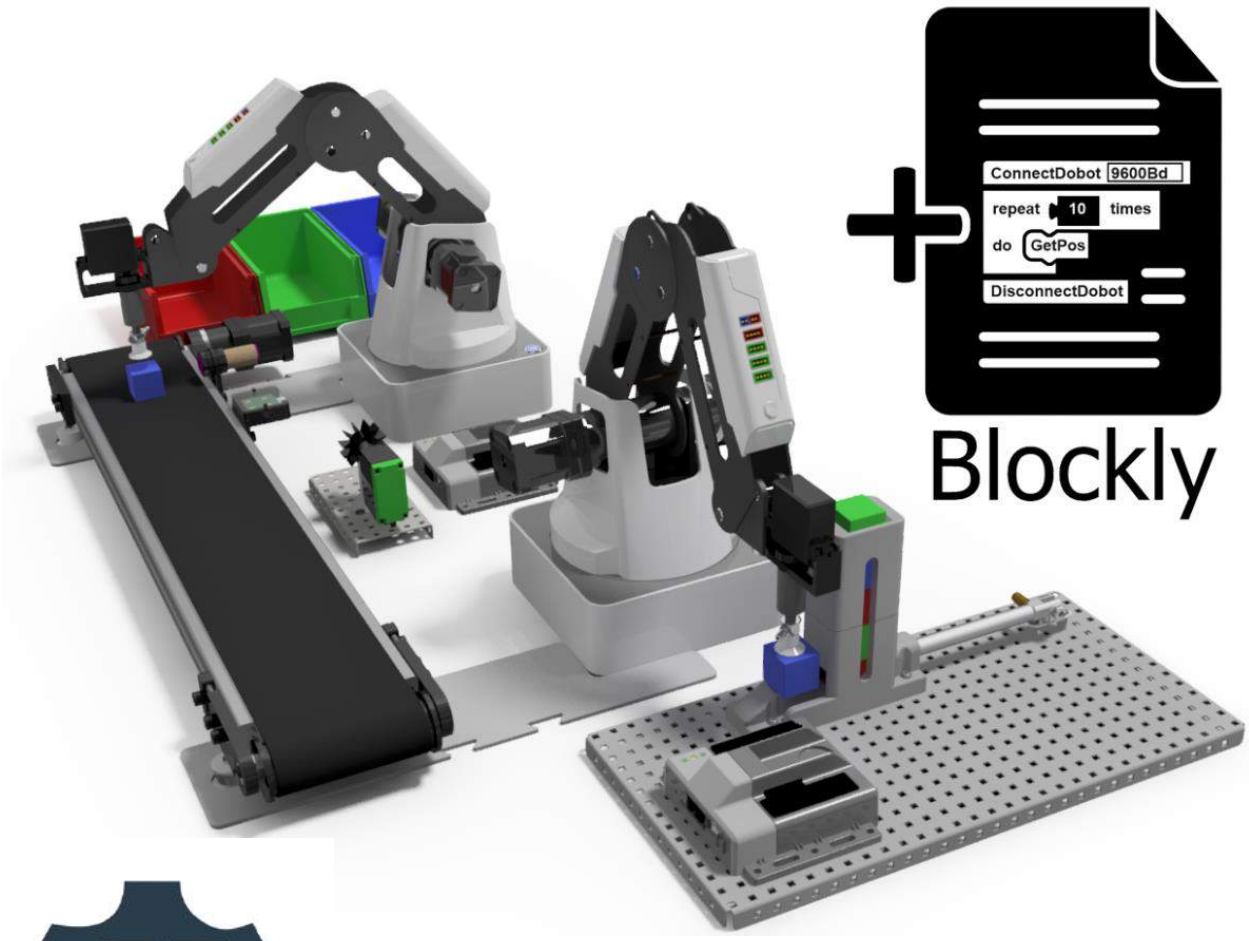
DOBOT



www.ChrisandJimCIM.com

Programming in DobotStudio with Blockly

Jim Hanson & Chris Hurd

Introduction: The Curriculum

This curriculum was designed to teach high school and college level students the basics of robotics, as used in industry, using the Dobot Magician, DobotStudio software, and the blockly programming language.

Through these activities, you will also be able to make the robot interact with other devices including, but not limited to:

- Arduino microcontrollers
- Color sensors
- Conveyor belts
- Other outputs like motors and LED's
- VEX Cortex microcontroller
- Infrared sensors
- Other robots
- Other inputs like microswitches and sensors

Introduction: Defining Artificial Intelligence

A lot is being reported about artificial intelligence and robotics in industry, and it is probably one of the most controversial issues surrounding robots… If you do not understand it. The way that we wish to address it in this document depends on how it is defined. For our use here Jim and I will define it as such:

Artificial Intelligence(AI): Using computers, microcontrollers, and other electronic devices to replicate intelligent behavior to automate tasks and make manufacturing more efficient.

We would like to look at AI from a practical standpoint. How AI helps us in industrial robotics and automation is what intrigues us the most and what we are most passionate about. In that vein, the next question to answer then is: "How does this curriculum embody artificial intelligence?"

With the above definition of AI, isn't a lot of automation & robotics considered artificial intelligence? Take these instances of what students will be able to do with a dobot and this curriculum:

- Make a motor run forward or backwards depending on what time it is.
- Make a light bulb light up when you want it to. Better yet, make different color lights light dependent upon what you want.
- Determine what color an object is and then decide where to put it.
- Change the speed of a motor dependent upon where an object is.
- Mathematically calculate where to put the next box on a pallet, or to stack objects perfectly.
- Make a robot talk to another robot and decide when to perform certain actions.
- Make a robot talk to another device to perform a myriad of automation and manufacturing tasks.
- Make a robot 3D print a necessary part for you, or laser engrave a barcode on each passing part of an assembly line.

All of these are possible with only a Dobot Magician, A microcontroller, this curriculum, a little determination, and a lot of curiosity.

Introduction: Defining Industry 4.0

Here's another term that is being widely used in Industry, and being touted as the next greatest thing in manufacturing. What does it mean? Again, we have to define it for ourselves so that we can move forward, and possible embrace it. From our limited research and knowledge of the topic we would like to define it this way:

Industry 4.0: The 4th industrial revolution where manufacturing facilities employ computers, machines, and technology, that have inputs and outputs that allow them to wirelessly connect to ever larger manufacturing systems.

No more is a drill press just a drill press. It may be a CNC machine that has a vision system that knows where a hole has to go as well as what size it is. Also this machine can be programmed on the fly to change rapidly if a different order comes in from a different vendor. It'll even tell the customer when the part will be done, and in some instances, some factories will even let customers watch their parts being produced via webcam. The list of tasks above in the definition of AI are all within reach of high school and college students alike, and aren't these tasks all a part of Industry 4.0?

With a Dobot Magician, this curriculum, and a bunch of spare parts, computers, and some ingenuity, students will definitely be headed in the right direction towards being a part of the future of Industry; no matter what it's called when they graduate.

Table of Contents (List of Activities & Presentations)

Curriculum Standards ..... **Page 8**

Presentation: Dobot Blockly-Programming Commands ..... **Page 13**

1 Blockly - Pick and Place ..... **Page 40**

Students learn about how to complete basic pick and place routines using the blockly coding technique.

Essential questions answered in this activity include:
- How do I use blockly to move the robot?
- How can I create a delay?
- What method is used to turn the suction on and off?
- How do I use other end effectors?
- What are some of the basic Dobot configurations?

2 Blockly - PnP with Jumps & Loops ..... **Page 51**

Students learn about how to use jumps and loops in blockly to make the robot move and repeat tasks.

Essential questions answered in this activity include:
- What's a Jump, and why is it important?
- How to make the robot repeat a motion or a task?
- When would I use a While statement?
- How do I use an Until statement?
- How do I set jump height?

3 Blockly - Pick and Place With Digital Inputs ..... **Page 59**

Students learn about what it means to be a digital input, how it works, and how to code a switch in blockly to act as an input device.

Essential questions answered in this activity include:
- What's the difference between an input and an output in robotics?
- What are some examples of digital inputs?
- How do I code a digital switch as an input in blockly?

4 Blockly - Developing a Cube Matrix …….. Page 68

Students learn about the use of variables and functions when programming a robotic arm. This activity uses mathematical concepts to de-palletize a set of blocks.

Essential questions answered in this activity include:
- Why are variables used?
- How do I use variables in blockly?
- How can I use math to palletize?
- What are some of the blockly commands needed to do this?
- Why is palletization and de-palletization important in industry?
-

5 Blockly - Using the Color Sensor …….. Page 87

Students learn about how to make the robot interact with the color sensor as an input.

Essential questions answered in this activity include:
- What's the difference between digital and analog?
- What colors can i sense with the color sensor?
- How does the color sensor work?
- How do I wire the components together?
- How do I get a value from the color sensor?
- How do I print to a log?
- How do I sort items by color with a robotic arm?

6 Blockly - Starting & Stopping the Conveyor …….. Page 105

Students learn about the basics of conveyor belts, and how to interface one with the Dobot Magician. Students will also learn how to use the infrared sensor to stop and start the conveyor.

Essential questions answered in this activity include:
- Is the conveyor an input or an output to the robot?
- Is the Infrared sensor an input or an output?
- How do I wire the components together?
- What can I do with the infrared sensor?
- How do I code the conveyor in blockly?
- How do I code the infrared sensor in blockly?

Presentation: Dobot Blockly-Hardware Connections …….. Page 119

7 Activity: Dobot to Dobot Handshaking ..... Page 132

Students develop an understanding of how robots can communicate with one another through the use of inputs and Outputs. Students use two Dobot Magicians to complete a two robot operation without timing.

Essential questions answered in this activity include:
- How do I make a robot send a signal?
- How do I get a robot to receive a signal?
- How is this done in Dobot Studio Software?
- How do I make two robots talk to one another?
-

8 Blockly - Handshaking - Arduino to Arduino ..... Page 143

Students develop an understanding of how important it is to be able to make machines talk to one another. In this activity students learn how to make Arduino based microcontrollers talk to one another.

Essential questions answered in this activity include:
- How do I make my microcontroller send and receive signals?
- What kind of software is necessary to communicate?
- How do I wire the hardware to communicate?
- How do I troubleshoot a complex system?

9 Blockly - Handshaking - VEX to VEX ..... Page 154

Students develop an understanding of how important it is to be able to make machines talk to one another. In this activity students learn how to make VEX based microcontrollers talk to one another using RobotC.

Essential questions answered in this activity include:
- How do I make my microcontroller send and receive signals?
- What kind of software is necessary to communicate?
- How do I wire the hardware to communicate?
- How do I troubleshoot a complex system?

10 Activity: Handshaking - Dobot to VEX ……..….. **Page 163**

Students develop an understanding of how robots can communicate with other devices, like microcontrollers, through the use of inputs and outputs.

Essential questions answered in this activity include:

- How do I get a robot to send and receive a signal?
- How do I make my microcontroller send and receive signals?
- How is this done in Dobot Studio Software?
- How do I wire the hardware to make this happen?
- How do I troubleshoot a complex robotic system?

11 Activity: Workcell Design ……..……. **Page 169**

Students develop an understanding of workcells and the interaction of different machines to complete a manufacturing process.

Essential questions answered in this activity include:

- How do you integrate robots and other parts of a work cell to complete a given task?
- How do you safely communicate between a microcontroller and a robot?
- What are the different types and styles of inputs and outputs needed to complete your given tasks?
- Which end of arm tooling is most appropriate for your workcell?
- Where is it appropriate to use the jump command within my workcell?
- Where would it be appropriate in your programming to use either variables or functions while programming?
- What components of Blockly did you need to complete this task?
- What other software & hardware will I need to complete this task?

Appendix A: Blockly Glossary ……. **Page 173**

Appendix B: Input/Output Guide ……. **Page 176**

Appendix C: Field Diagram Pallet & Dip Tank…….. **Page 181**

Appendix D: Field Diagram Blank …….. **Page 182**

Curriculum Standards

The standards defined below are far reaching and very large in scope. They include the following standards:

- Standards for Technological Literacy
  https://www.iteea.org/File.aspx?id=67767
- Next Generation Science Standards
  http://www.nextgenscience.org/ pg 102
- Common Core State Standards for Math
  http://www.corestandards.org/Math/Practice/

All the activities in this curriculum, once completed, will cover all of the standards outlined below, and most of them multiple times.

Standards for Technological Literacy	
The Standards for Technological Literacy (STL) were developed by the International Technology and Engineering Educators Association (ITEEA) and are available as a complete download for free here: https://www.iteea.org/File.aspx?id=67767	
2-W	*Standard:* Students will develop an understanding of the core concepts of technology. *Benchmark:* Systems thinking applies logic and creativity with appropriate compromises in complex real-life problems.
2-Z	*Standard:* Students will develop an understanding of the core concepts of technology. *Benchmark:* Selecting resources involves trade-offs between competing values, such as availability, cost, desirability, and waste.
2-AA	*Standard:* Students will develop an understanding of the core concepts of technology. *Benchmark:* Requirements involve the identification of the criteria and constraints of a product or system and the determination of how they affect the final design and development.
2-BB	*Standard:* Students will develop an understanding of the core concepts of technology. *Benchmark:* Optimization is an ongoing process or methodology of designing or making a product and is dependent on criteria and constraints.
8-H	*Standard:* Students will develop an understanding of the attributes of design. *Benchmark:* The design process includes defining a problem, brainstorming, researching and generating ideas, identifying criteria and specifying constraints, exploring possibilities, selecting an approach, developing a design proposal, making a model or prototype.

8-I	*Standard:* Students will develop an understanding of the attributes of design. *Benchmark:* Design problems are seldom presented in a clearly defined form.
8-J	*Standard:* Students will develop an understanding of the attributes of design. *Benchmark:* The design needs to be continually checked and critiqued, and the ideas of the design must be redefined and improved.
8-K	*Standard:* Students will develop an understanding of the attributes of design. *Benchmark:* Requirements of a design, such as criteria, constraints, and efficiency, sometimes compete with each other.
9-I	*Standard:* Students will develop an understanding of engineering design. *Benchmark:* Established design principles are used to evaluate existing designs, to collect data, and to guide the design process.
9-J	*Standard:* Students will develop an understanding of engineering design. *Benchmark:* Engineering design is influenced by personal characteristics, such as creativity, resourcefulness, and the ability to visualize and think abstractly.
9-K	*Standard:* Students will develop an understanding of engineering design. *Benchmark:* A prototype is a working model used to test a design concept by making actual observations and necessary adjustments.
9-L	*Standard:* Students will develop an understanding of engineering design. *Benchmark:* The process of engineering design takes into account a number of factors.
11-N	*Standard:* Students will develop the abilities to apply the design process. *Benchmark:* Identify criteria and constraints and determine how these will affect the design process.
11-O	*Standard:* Students will develop the abilities to apply the design process. *Benchmark:* Refine a design by using prototypes and modeling to ensure quality, efficiency, and productivity of the final product.
11-P	*Standard:* Students will develop the abilities to apply the design process. *Benchmark:* Evaluate the design solution using conceptual, physical, and mathematical models at various intervals of the design process in order to check for proper design and to note areas where improvements are needed.
11-Q	*Standard:* Students will develop the abilities to apply the design process. *Benchmark:* Develop and produce a product or system using a design process.
11-R	*Standard:* Students will develop the abilities to apply the design process. *Benchmark:* R. Evaluate final solutions and communicate observation, processes, and results of the entire design process, using verbal, graphic, quantitative, virtual, and written means, in addition to three-dimensional models.

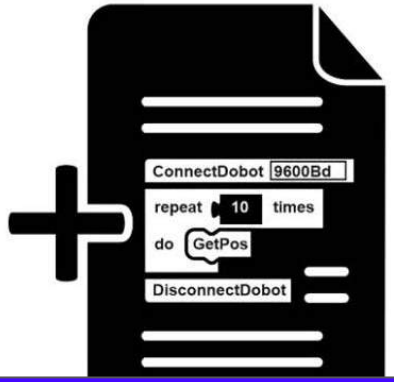
12-P	*Standard:* Students will develop the abilities to use and maintain technological products and systems. *Benchmark:* Use computers and calculators to access, retrieve, organize, process, maintain, interpret, and evaluate data and information in order to communicate.
Next Generation Science Standards	
The **Next Generation Science Standards** is a multi-state effort to create new education **standards** that are "rich in content and practice, arranged in a coherent manner across disciplines and grades to provide all students an internationally benchmarked **science** education." More information can be found here: http://www.nextgenscience.org/ pg 102	
HS.ETS1.2	Engineering Design
	Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.
HS.ETS1.3	Engineering Design
	Evaluate a solution to a complex real-world problem based on prioritized criteria and trade-offs that account for a range of constraints, including cost, safety, reliability, and aesthetics, as well as possible social, cultural, and environmental impacts.
DCI - ETS1.B	Engineering Design - Developing Possible Solutions
	When evaluating solutions, it is important to take into account a range of constraints, including cost, safety , reliability , and aesthetics, and to consider social, cultural, and environmental impacts. (HS-ETS1-3)
DCI - ETS1.C	Engineering Design - Optimizing the Design Solution
	Criteria may need to be broken down into simpler ones that can be approached systematically, and decisions about the priority of certain criteria over others (tradeoffs) may be needed. (secondary to HS-PS1-6)
	Science and Engineering Practice - Planning and Carrying Out Investigations
	Plan and conduct an investigation or test a design solution in a safe and ethical manner including considerations of environmental, social, and personal impacts.
	Science and Engineering Practice - Using Mathematics and Computational Thinking
	Create and/or revise a computational model or simulation of a phenomenon, designed device, process, or system.

	Crosscutting Concepts - Systems and System Models
	• A system is an organized group of related objects or components; models can be used for understanding and predicting the behavior of systems. • Systems can be designed to do specific tasks. • When investigating or describing a system, the boundaries and initial conditions of the system need to be defined and their inputs and outputs analyzed and described using models. • Models (e.g., physical, mathematical, computer models) can be used to simulate systems and interactions—including energy, matter, and information flows—within and between systems at different scales. • Models can be used to predict the behavior of a system, but these predictions have limited precision and reliability due to the assumptions and approximations inherent in models.
	Connections to Engineering, Technology, and Applications of Science
	Influence of Science, Engineering, and Technology on Society and the Natural World New technologies can have deep impacts on society and the environment, including some that were not anticipated. Analysis of costs and benefits is a critical aspect of decisions about technology. (HS-ETS1-1) (HSETS1-3)

Common Core State Standards for Mathematics	
The Standards for Mathematical Practice describe varieties of expertise that mathematics educators at all levels should seek to develop in their students. These practices rest on important "processes and proficiencies" with longstanding importance in mathematics education. More information here: http://www.corestandards.org/Math/Practice/	
N.Q.1	**Quantities** Use units as a way to understand problems and to guide the solution of multistep problems; choose and interpret units consistently in formulas; choose and interpret the scale and the origin in graphs and data displays.
N.Q.3	**Quantities** Choose a level of accuracy appropriate to limitations on measurement when reporting quantities.
N.Q.4	**Vector and Matrix Quantities** (+) Add and subtract vectors.
A.SSE.1.A	**Seeing Structure in Expressions** Interpret parts of an expression, such as terms, factors, and coefficients.

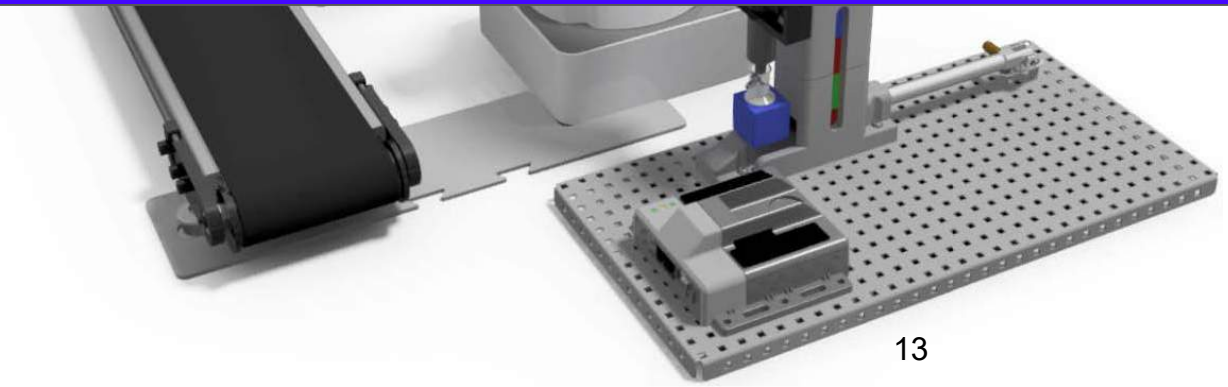
A.CED.4	**Creating Equations** Rearrange formulas to highlight a quantity of interest, using the same reasoning as in solving equations. For example, rearrange Ohm's law V = IR to highlight resistance R.
A.REI.1	**Reasoning with Equations and Inequalities** Explain each step in solving a simple equation as following from the equality of numbers asserted at the previous step, starting from the assumption that the original equation has a solution. Construct a viable argument to justify a solution method.
A.REI.3	**Reasoning with Equations and Inequalities** Solve linear equations and inequalities in one variable, including equations with coefficients represented by letters.
G.SRT.8	**Similarity, Right Triangles, and Trigonometry** Use trigonometric ratios and the Pythagorean theorem to solve right triangles in applied problems.
G.MG.1	**Modeling with Geometry** Use geometric shapes, their measures, and their properties to describe objects (e.g., modeling a tree trunk or a human torso as a cylinder).
G.MG.3	**Modeling with Geometry** Apply geometric methods to solve design problems (e.g., designing an object or structure to satisfy physical constraints or minimize cost; working with typographic grid systems based on ratios).

DOBOT



Blockly & Dobot

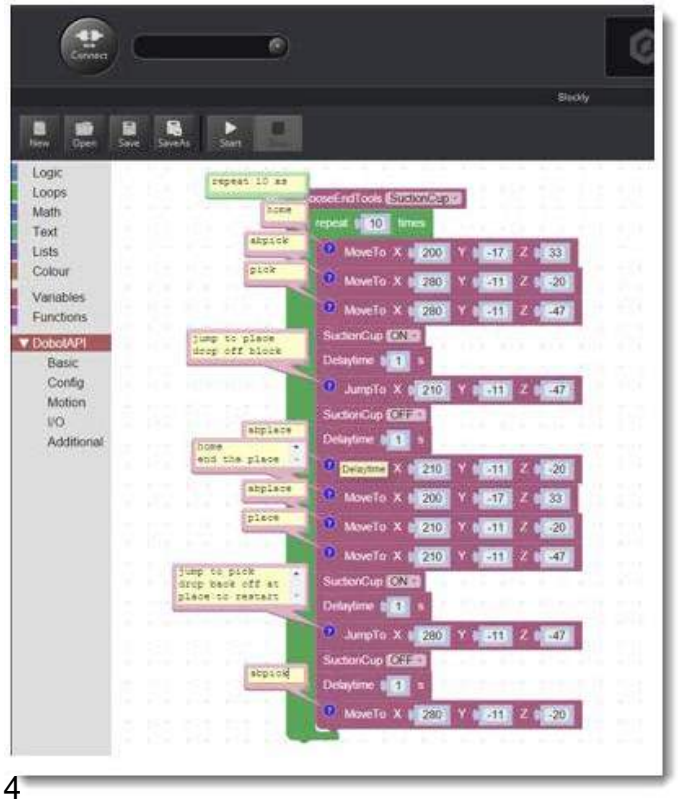
Programming Commands



Blockly Commands for the Dobot Magician

Blockly Definition:

A programming language used to program the Dobot Magician. Lines of complex code are represented by simple “blocks” that fit together to form a program. **Blockly** is a graphical programming method rather than text based.

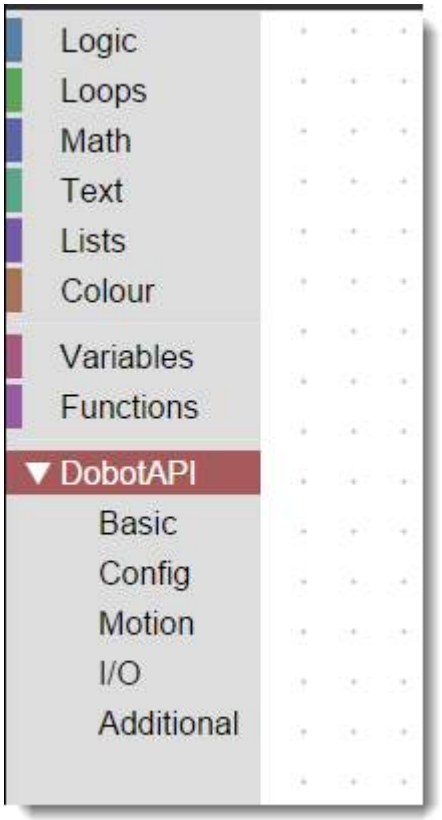


Blockly Commands for the Dobot Magician

Types of Commands in Blockly

In DobotStudio, Blockly commands are broken up into nine different categories with one category, **DobotAPI**, broken up into 5 subcategories.

Each of these have similar commands grouped together, and this presentation will describe and define some of the most common blocks that are used in programming the Dobot Magician.



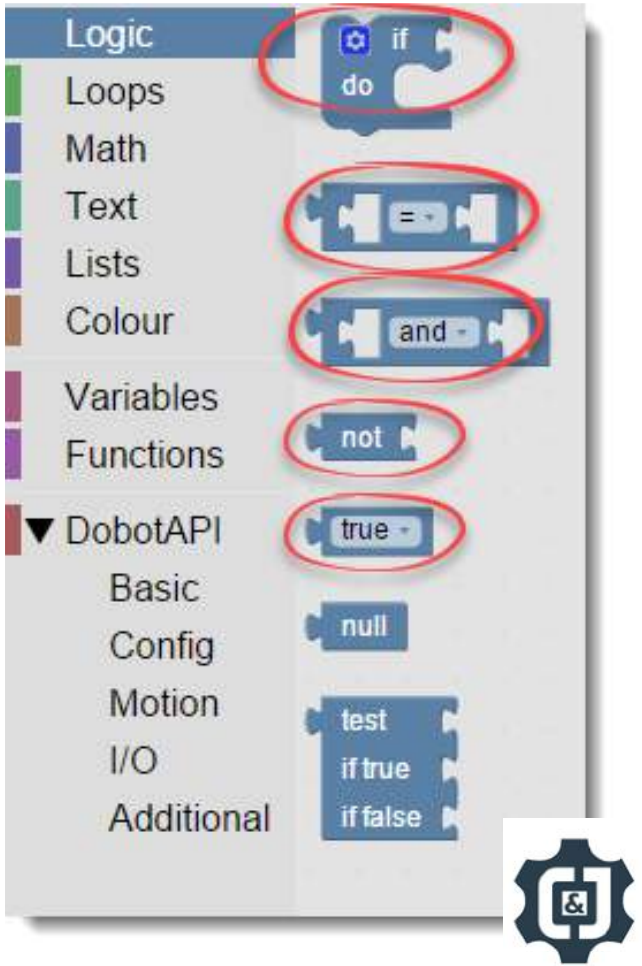
Blockly Commands: Logic

Logic

Logic commands allow you to use Boolean operands to make your robot complete complex operations.

Some of the important tasks that these blocks will allow you to do are:

- ***If Else** statements*
- *Set two programming elements to <,>,=*
- *Use **AND**, **OR** and **NOT***
- *Set something to **TRUE** or **FALSE***

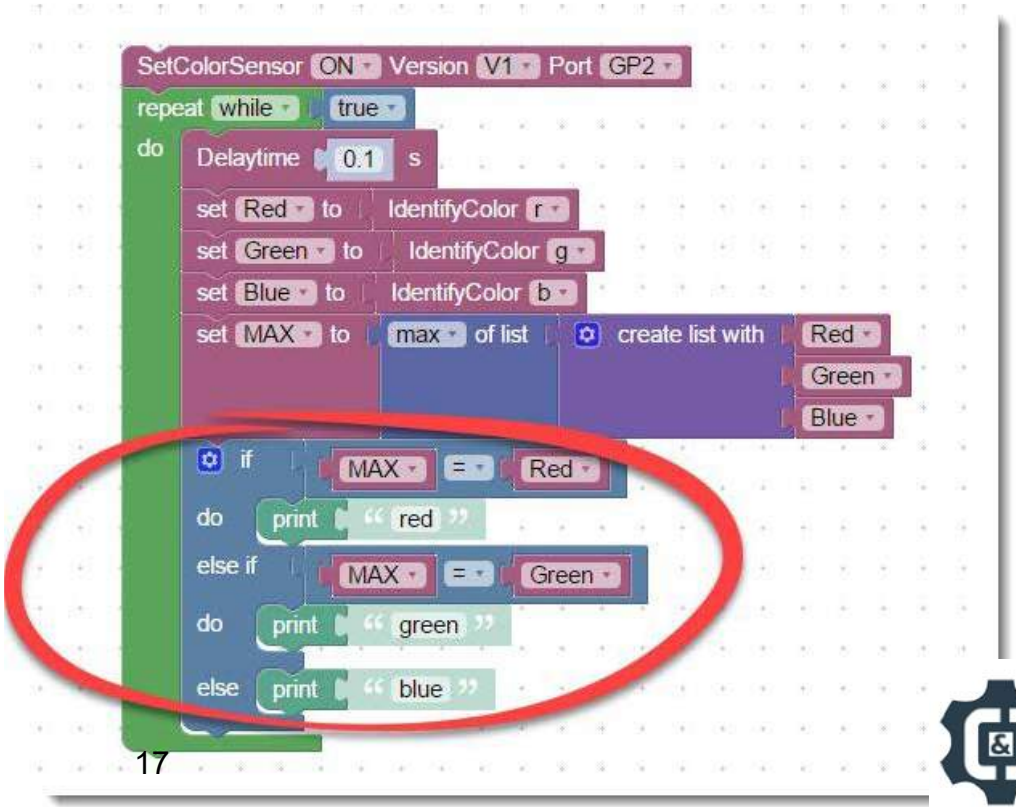


Blockly Commands: Logic

Logic

*In this example an **If Else** statement is used to make a color sensor print the color of the part being sensed.*

Notice that there are actually three different conditions.



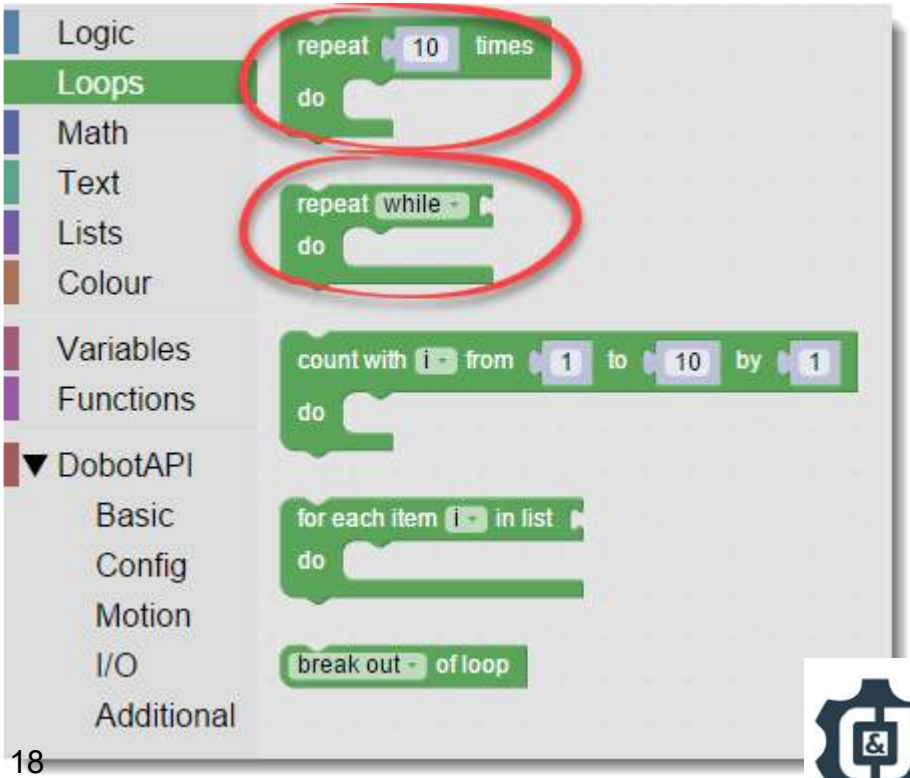
Blockly Commands: Loops

Loops

Logic commands that will allow you to repeat actions within a program.

Some of the important tasks that these blocks will allow you to do are:

- ***Repeat** a number of times*
- ***Repeat while** something is happening*
- ***Repeat forever***

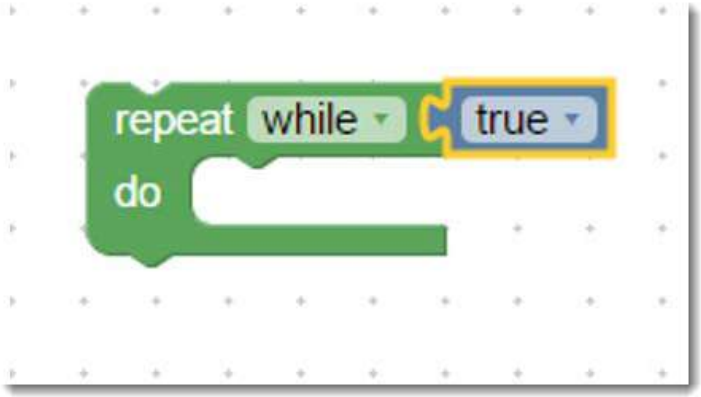


Blockly Commands: Loops

Repeat forever

*Blockly logic commands that allows you to use **TRUE** with the repeat command to continually complete an action or set of actions*

*In this example you can use this block with a **TRUE** block and make a block, or group of blocks run continuously. This could be used when you want to continuously look for an input.*



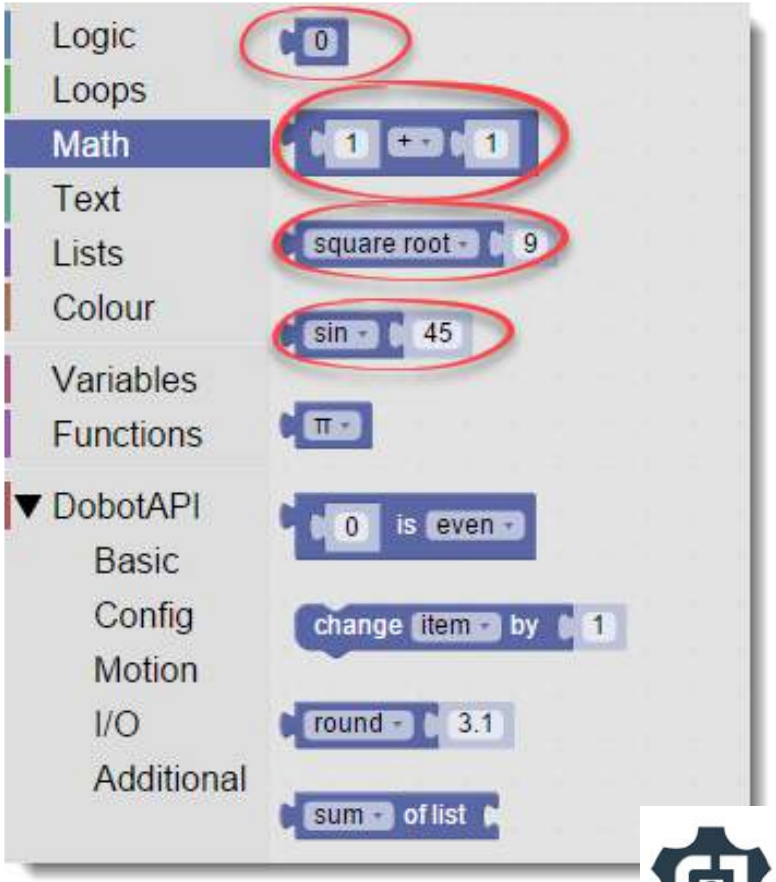
Blockly Commands: Math

Math

Logic commands that allow you to use mathematics on numbers in your program. These are all very self explanatory.

Some of the important tasks that these blocks will allow you to do are:

- *Return a number of your choice*
- *Return a SUM of two numbers*
- *Return the sine/cosine/tangent of a number*
- *Return the square root of a number*

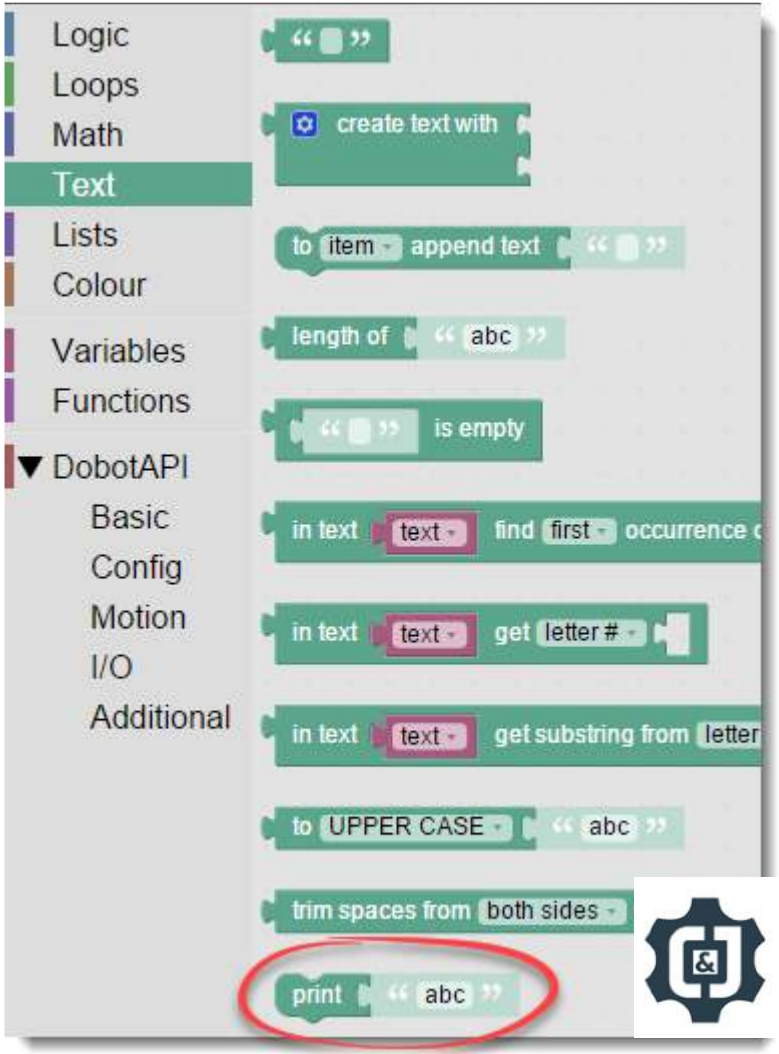


Blockly Commands: Text

Text

*Logic commands that will allow you to "Print" **text** and other programming elements to the running log so that you can see what is happening in your program in real time.*

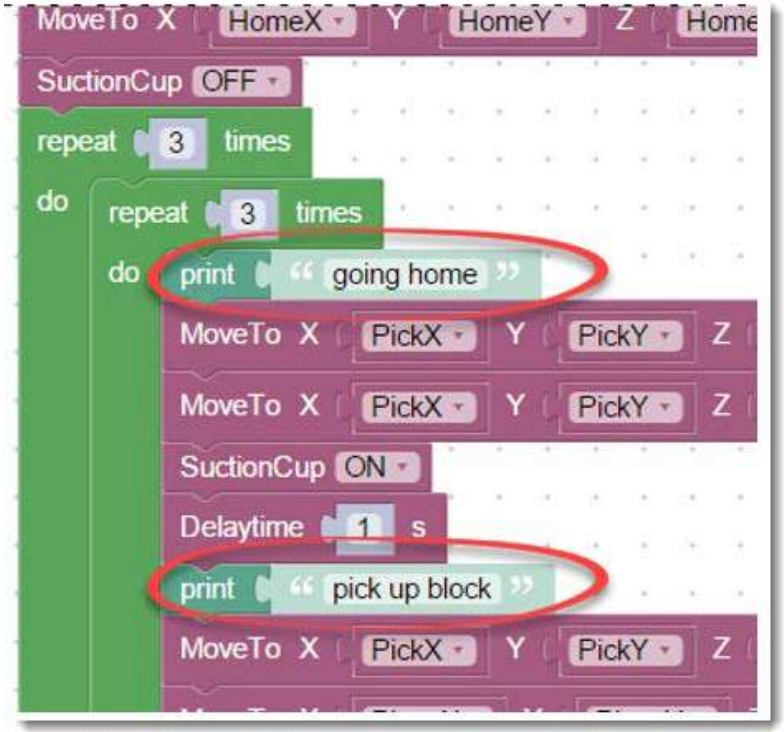
This is a great way to troubleshoot a complex program.



Blockly Commands: Text

Text

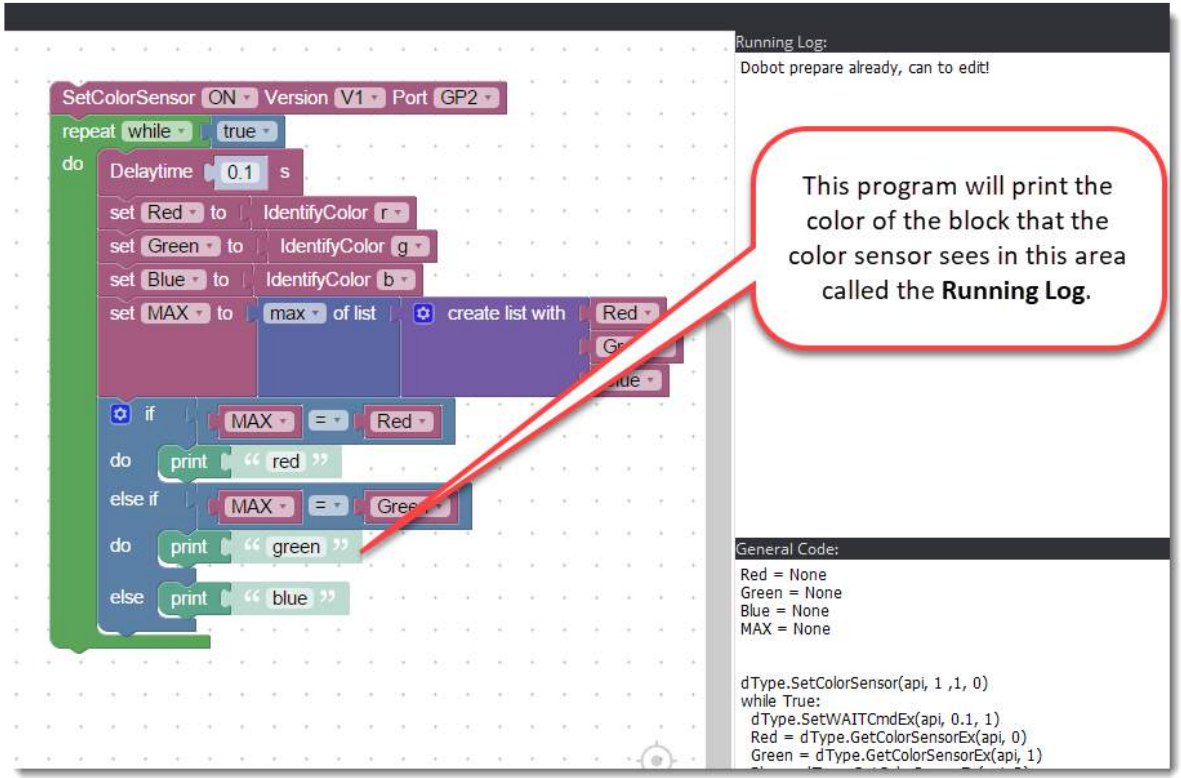
*In this example, it will print "**going home**" in the running log while the robot is moving to a home position and then "**pick up block**" when going to the pick position.*



Blockly Commands: Text

Text

*The **Running Log** appears to the right of the programming window in Dobot Studio and runs constantly.*

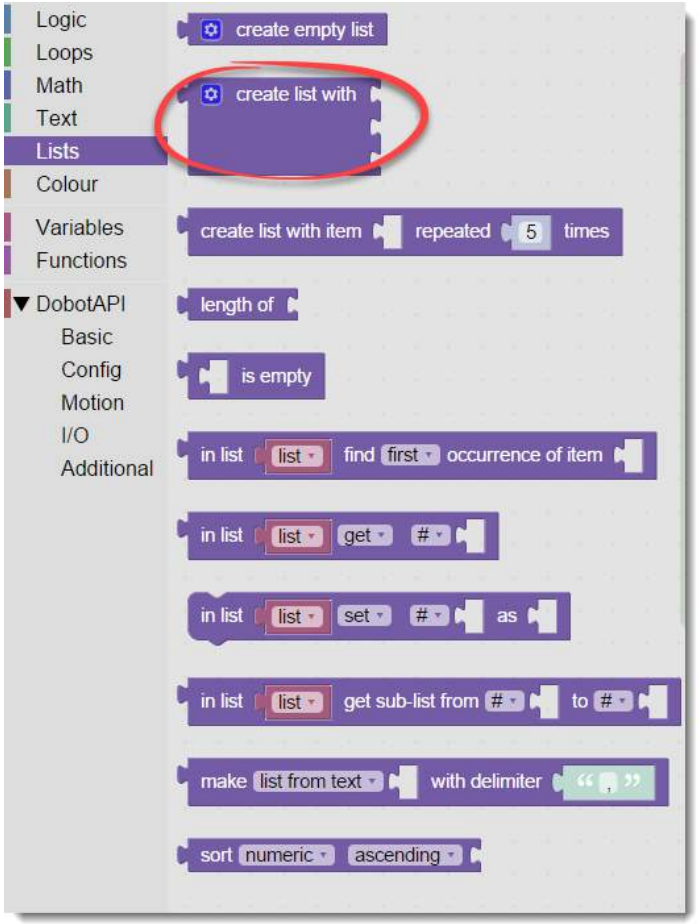


Blockly Commands: Lists

Lists

*Logic commands that will allow you to build and deal with lists. A **list** is an ordered set of items that can be used by the rest of your program.*

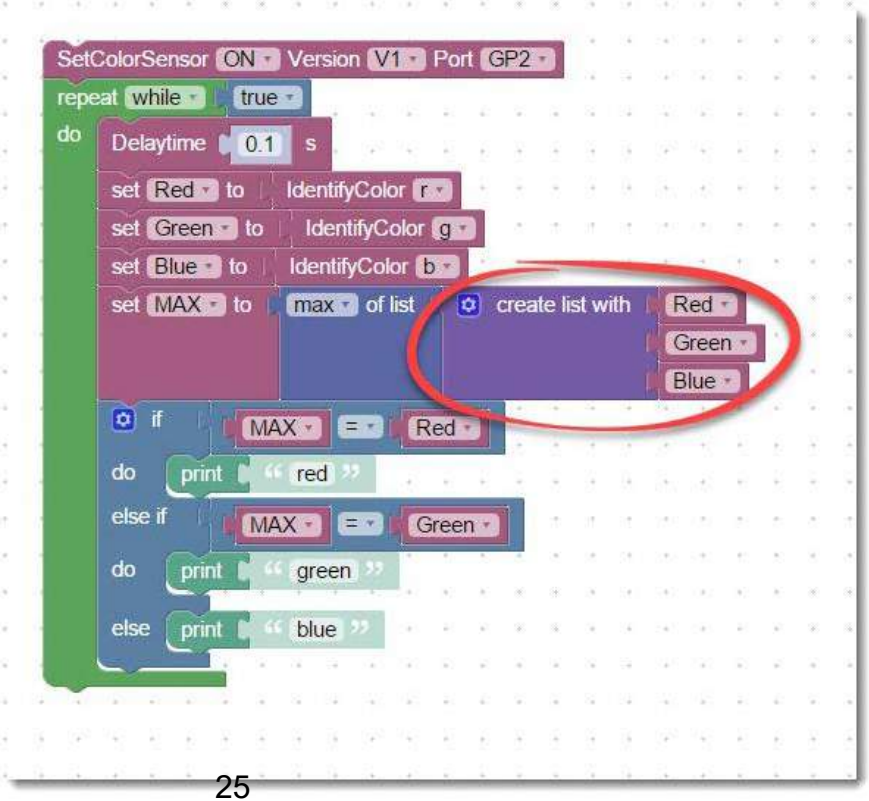
One of the important tasks that these blocks will allow you to do is to build a list when sorting colors with a color sensor.



Blockly Commands: Lists

Lists

*In this example program a **list** is used when a color sensor checks to see what color a block is. It then **prints** to to the running log the value: Red, Green, or Blue.*

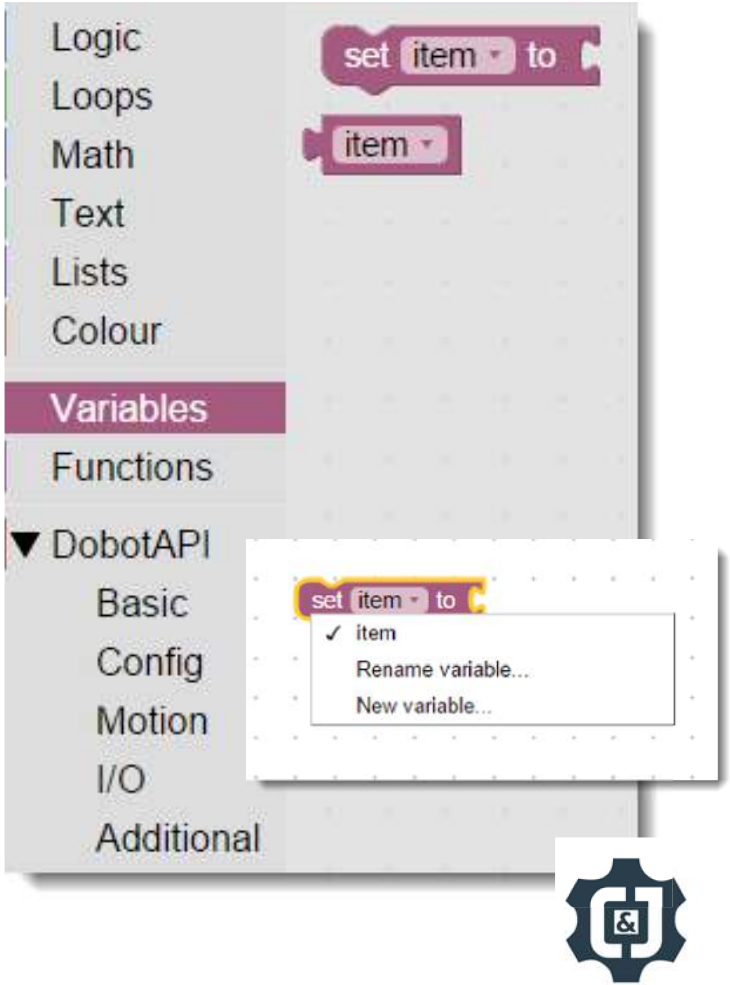


Blockly Commands: Variables

Variables

Logic commands that will allow you to set variables in a program and call them out for use when needed.

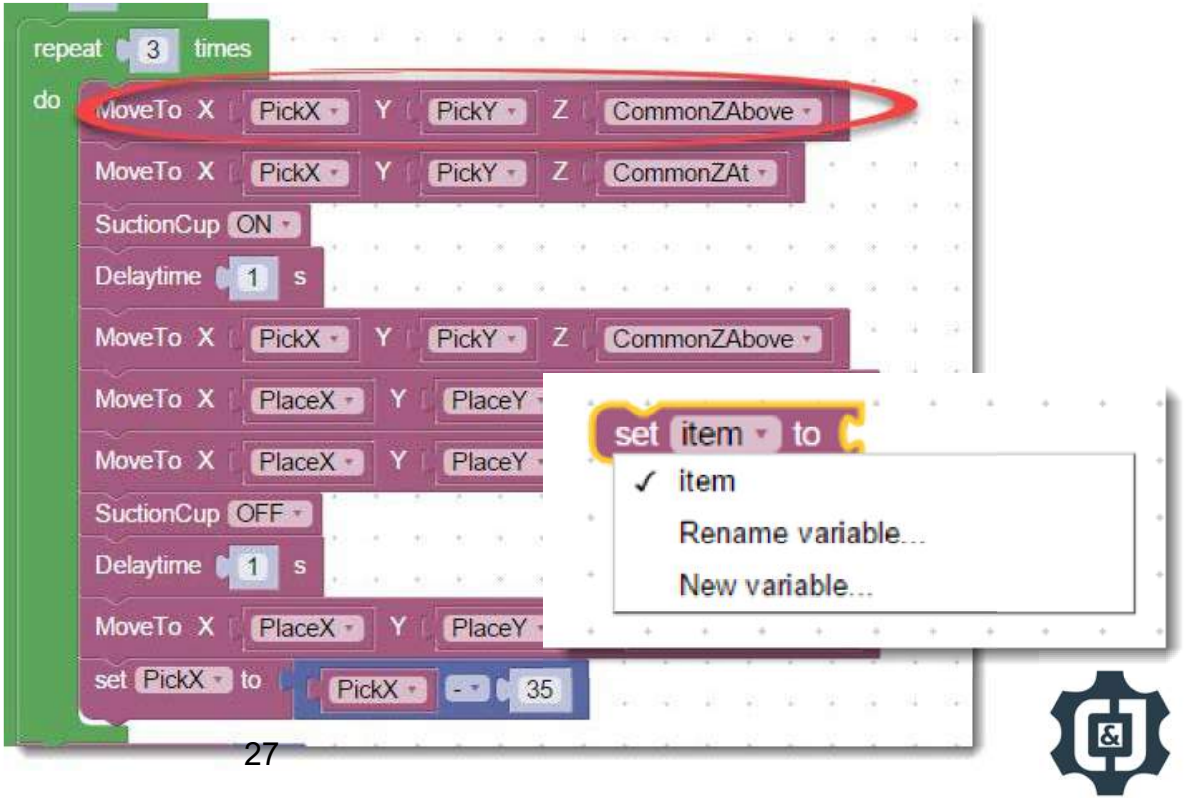
*In this example, you can click on "item", make a new **variable**, then use that **variable** in multiple places in the program. This makes it easy to make a change in a program. Change the **variable** once, and it changes it everywhere.*



Blockly Commands: Variables

Variables

*In this example program you can can see that **variables** were used to set the Pick X, Y, and Z values as well as the Place X, Y, and Z values.*

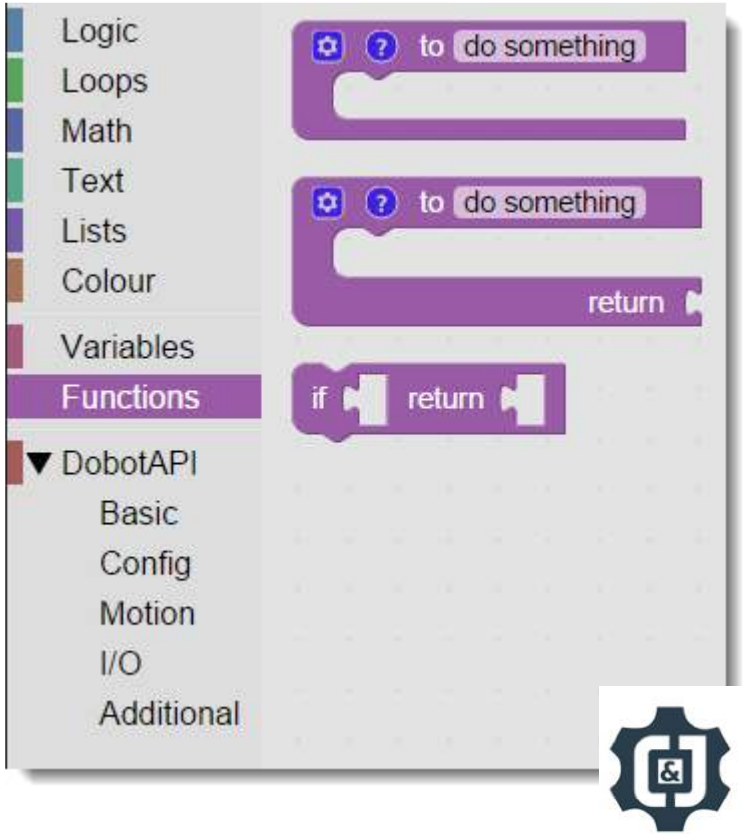


Blockly Commands: Functions

Functions

Logic commands that allow you to name a section of a program and then use it repeatedly, simplifying it for the programmer and end user. ***Functions*** *can also be called voids.*

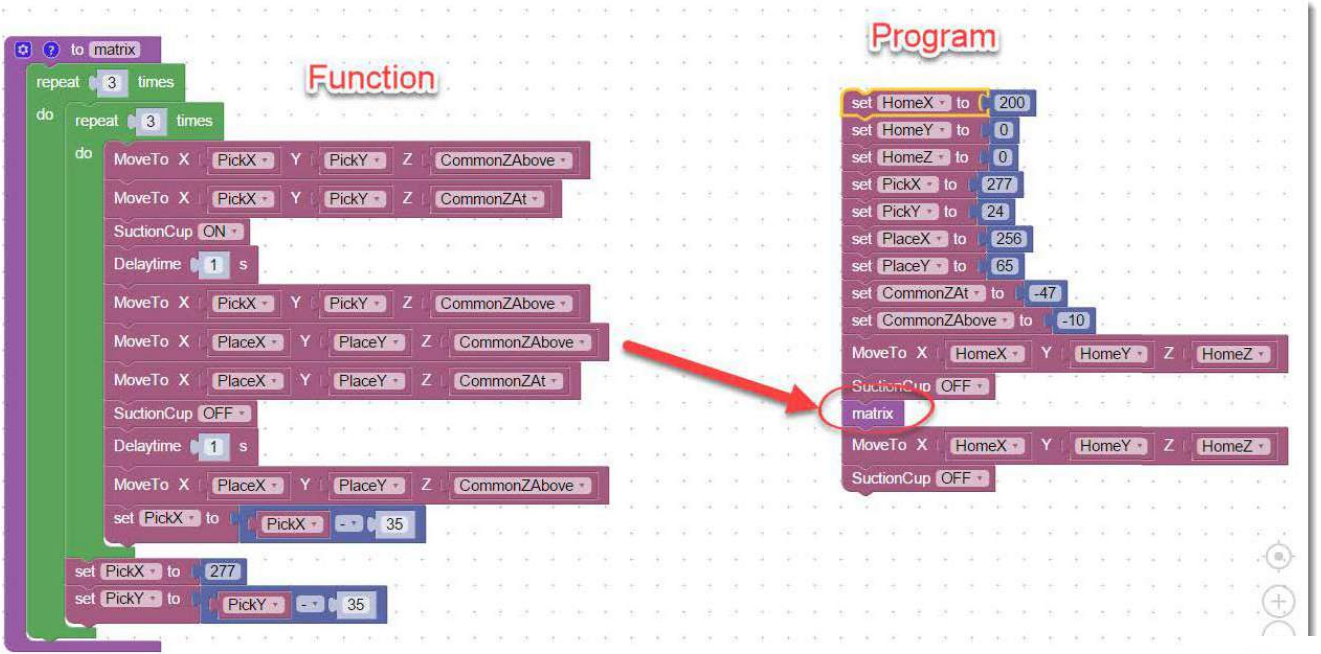
Click on "do something", name your function, then drag what you want it to do into the block.



Blockly Commands: Functions

Functions

*In this example program, the **Function** is on the left, and the program where it is called out is on the right.*

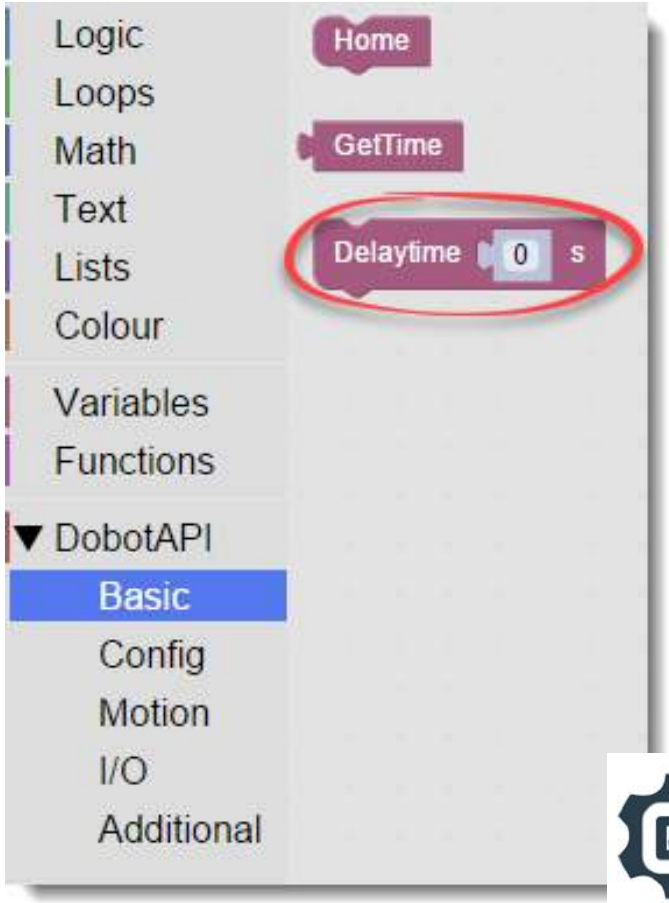


Blockly Commands: Basic

Dobot API - Basic

*The most important command in this section is the **Delaytime** command.*

This allows you to set a delay time within a program between steps when timing is critical. It is measured in seconds and decimal seconds can be used.

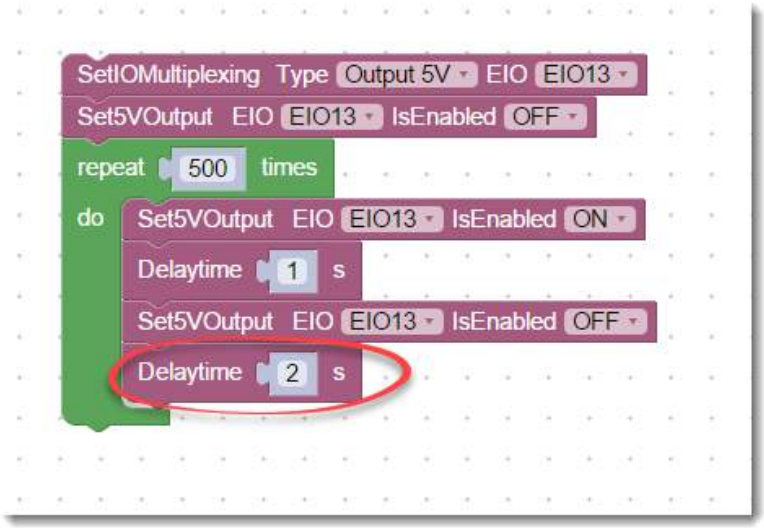


Blockly Commands: Basic

Dobot API - Basic

In this example program a robot is sending a signal to another device to test the connection. It is set to do this 500 times.

*The **Delaytime** after the output is turned off must be greater than 1 second otherwise the other machine does not have time to complete its process.*



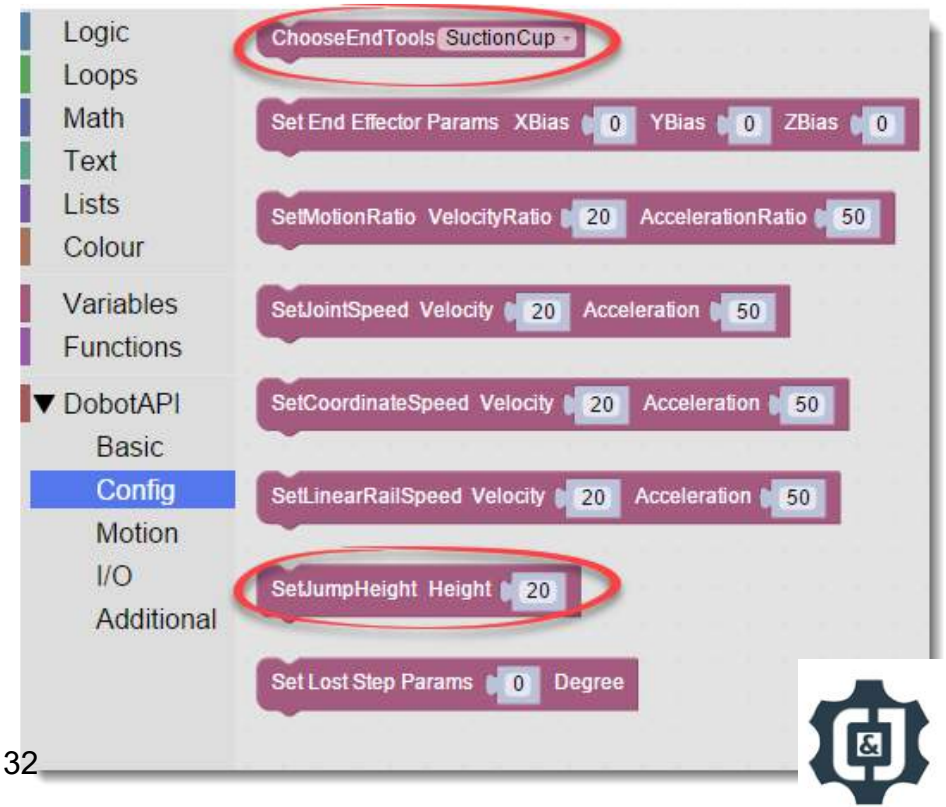
Blockly Commands: Config

Dobot API - Config

Logic commands that allow to configure certain items in the program.

The two most important tasks are the

- ***ChooseEndTools** – allows you to choose your end effector*
- ***SetJumpHeight**-allows you to choose the height of a jump move.*



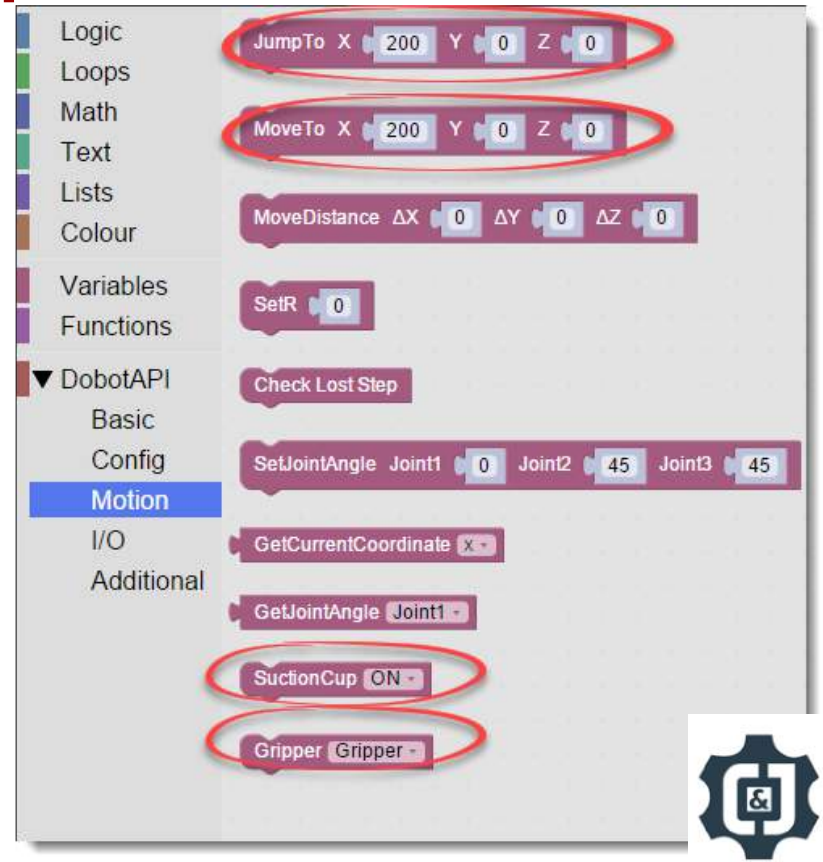
Blockly Commands: Motion

Dobot API - Motion

Logic commands that control the motion of the robot arm.

Some of the important tasks that these will allow you to do are:

- ***JumpTo** a cartesian coordinate*
- ***MoveTo** a cartesian coordinate*
- *Turn the Suction Cup on or off*
- *Open and close the gripper*

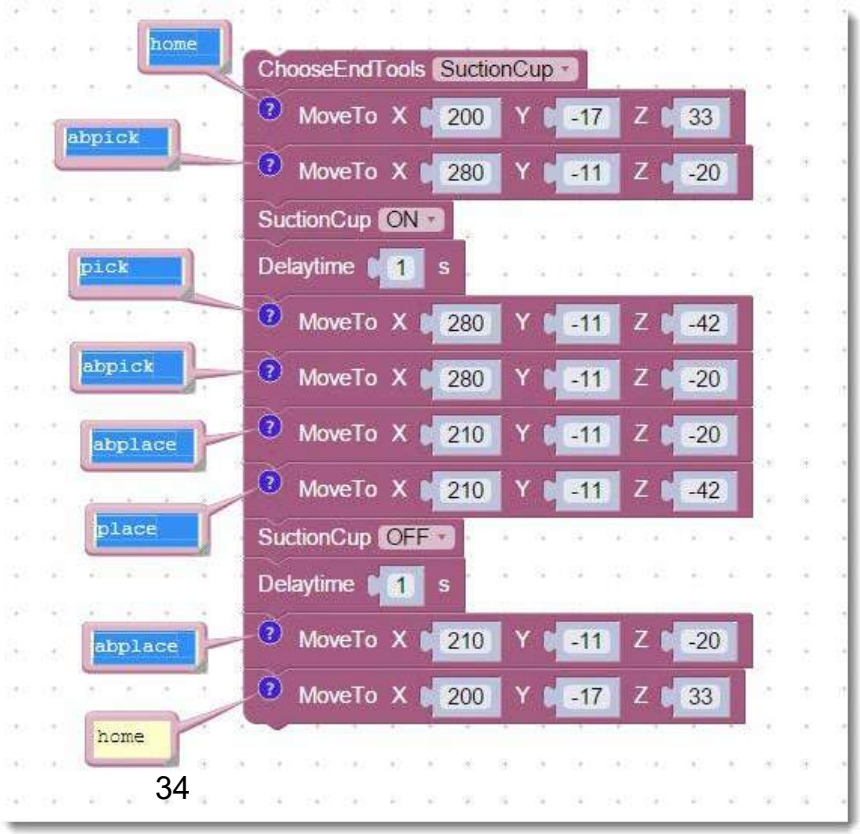


Blockly Commands: Motion

Dobot API - Motion

This example program completes a pick and place of an object in a workcell.

*Notice how **MoveTo** was used to move the robot between points.*



Blockly Commands: I/O

Dobot API – I/O

Logic commands that deal with Inputs and outputs

Some of the important ones are:

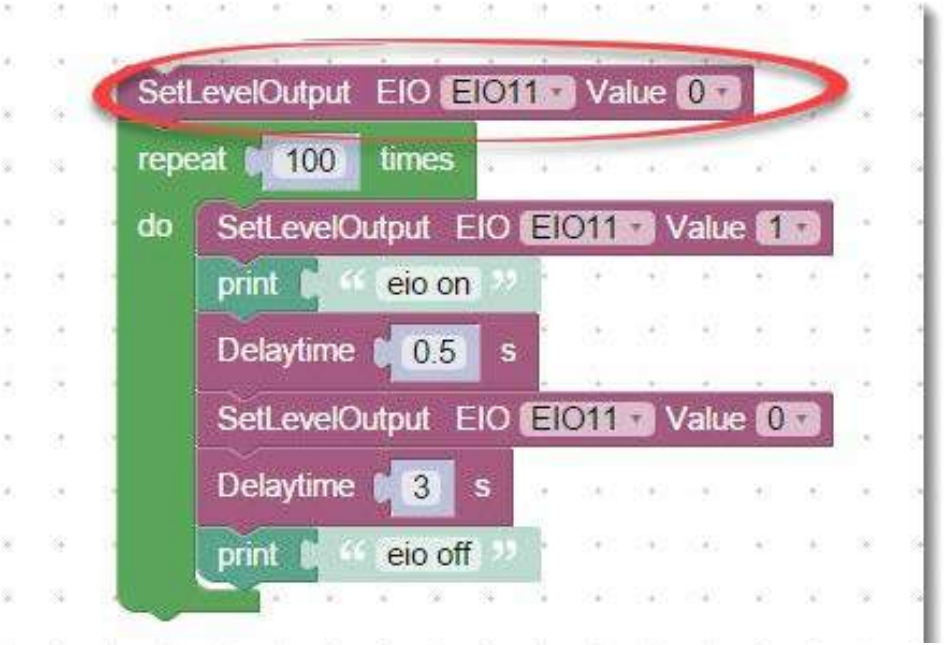
- *Set an input type and choose the port*
- *Check the level of an input*
- *Set a 3.3, 5, or 12v output*

Blockly Commands: I/O

Dobot API – SetLevelOutput

This example program was written to test an output by turning it on and off 100 times

Notice how the value of the output was set at 0 to start, then in the loop it uses a “1” to turn it on and “0” to turn it off.



Blockly Commands: Additional

Dobot API – Additional

Logic commands that deal mainly with sensors and outputs built specifically for the Magician.

Some of the important ones are:

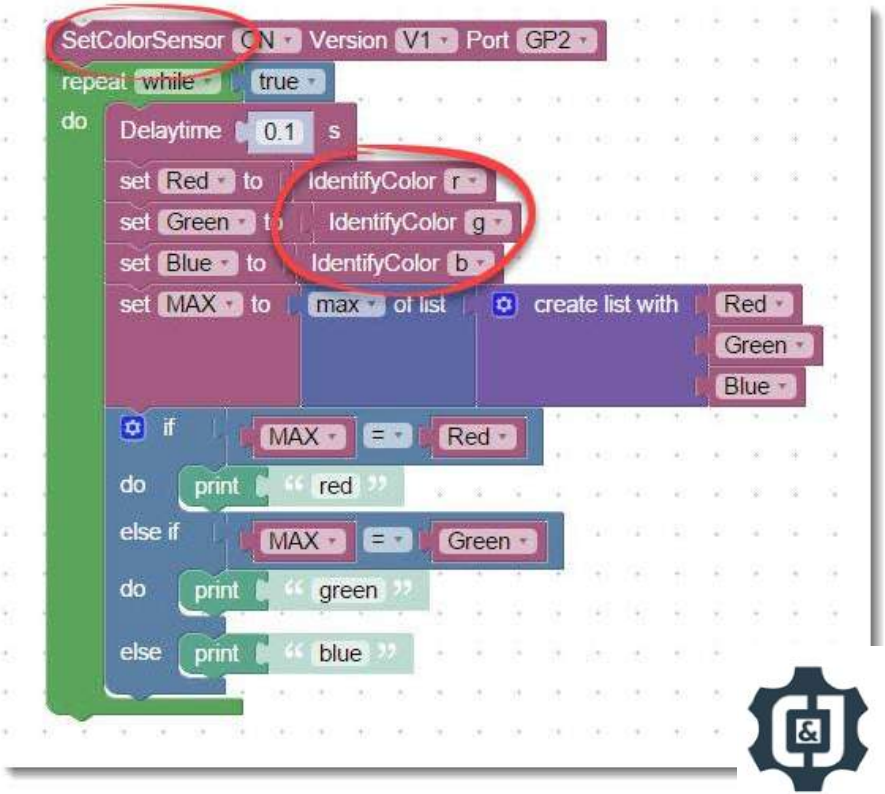
- ***SetPhotoSensor**, **GetPhotoSensor***
- ***SetColorSensor***
- ***IdentifyColor***
- ***SetConveyor** Motor port & Speed*

Blockly Commands: Additional

Dobot API – Additional

*In this example program **SetColorSensor** tells the program to turn it on and that you are using a V1 color sensor on port GP2.*

*The **IdentifyColor** block is then used to identify the color of an object in front of the sensor where variables are used to name the values “Red”, “Blue”, and “Green”, then the names are printed to the Running Log.*



Resources

All photos, graphics, images & icons included in this presentation are the intellectual property of ChrisandJimCIM.com.

1 Blockly - Pick and Place

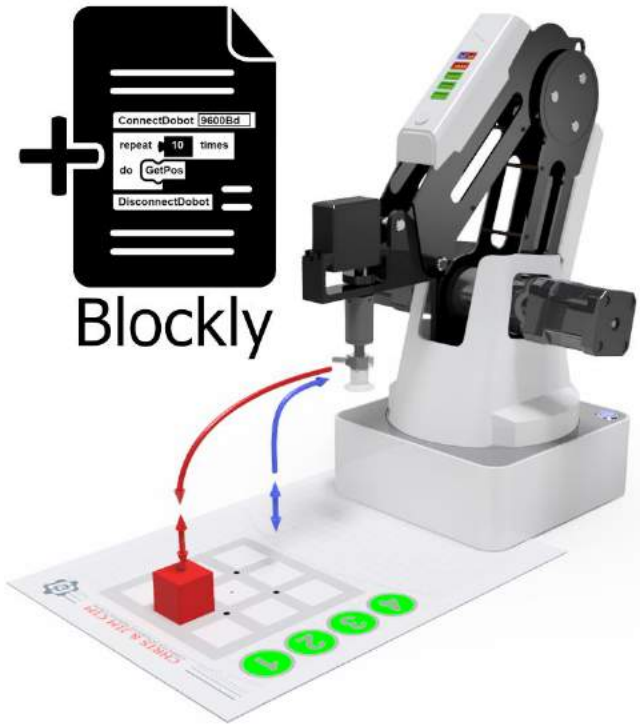
NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Robotic arms are excellent for performing pick and place operations such as placing small electronic components on circuit boards, as well as large boxes on pallets. A pick and place operation will require at least 5 points:

1. A home or safe location
2. A position above the object
3. A position at the object
4. A position above the drop off
5. A position at the drop off

In this activity you will learn how to make a basic *Pick and Place* operation in blockly. Through this activity you will learn how to program the robot to move and turn on it's suction cup in blockly



Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

KEY VOCABULARY

- MoveTo
- Placeholder
- Delay time
- Suction Cup
- Blockly Programming
- Pick and Place

All blockly commands have been put into a separate document called ***Blockly Vocabulary***, and can be referred to at any time throughout all of these activities.

EQUIPMENT & SUPPLIES

- Robot Magician
- Dobot Blockly - Pick and Place Field Diagram
- 1” cubes or cylinders
- DobotStudio software
- Suction Cup Gripper

ESSENTIAL QUESTIONS

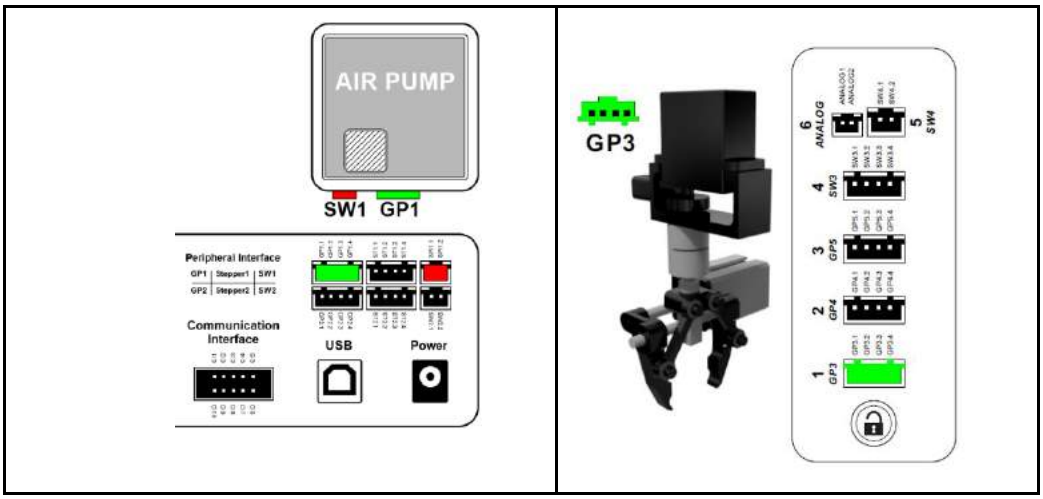
Essential questions answered in this activity include:

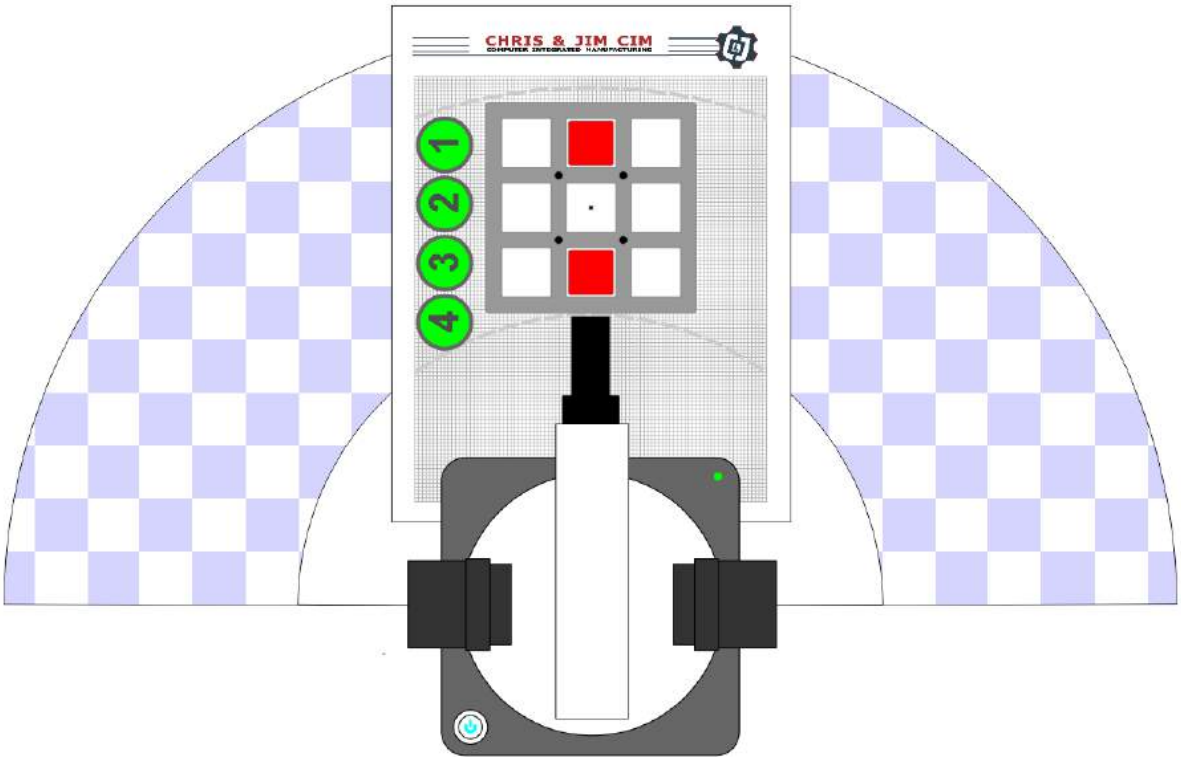
- How do I use blockly to move the robot?
- How can I create a *delay*?
- What method is used to turn the *suction on* and *off*?
- How do I use other end effectors?
- What are some of the basic Dobot configurations?

PROCEDURE

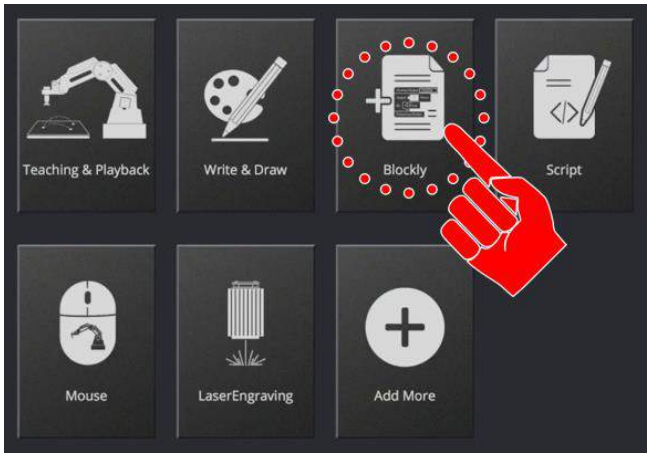
Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

1. Print the Blockly - Pick and Place Field Diagram
2. Set up the robot with a suction cup and Air pump and place a cube in one of the red squares on the field diagram provided.





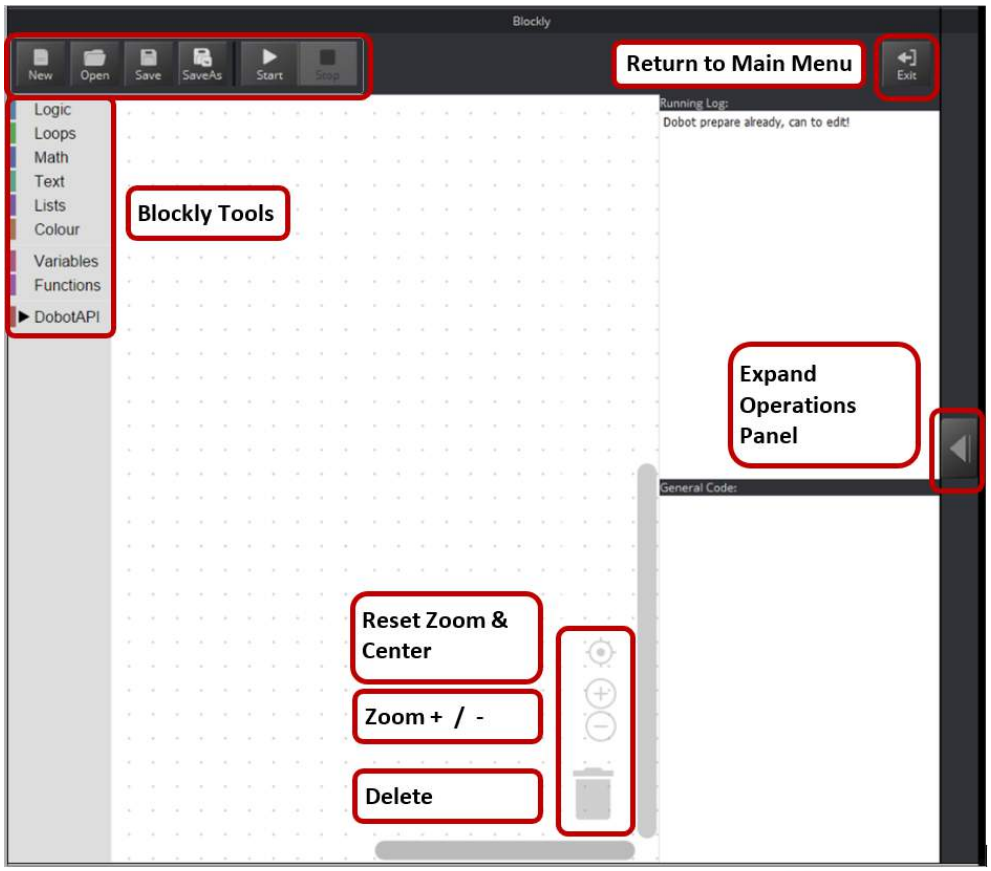
3. Open up Blockly in the software



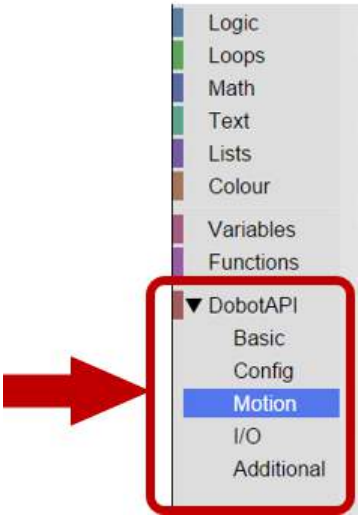
When DobotStudio is closed with a file open, it will reopen with the last file used. Insure the file open is the one you want to edit, if it is not, you may end up overwriting another program

Shown below is the main menu for the Blockly programming section of DobotStudio. On the left are categories and tools that each block of code is sorted into to. On the right are controls for zooming around the program as you develop it as well as a trash can for any code element(s) you wish to delete. The scroll wheel on the mouse can also be used to zoom in and out of the program. In order to pan up and down or left and right you may either use the scroll bars shown or click and drag on an empty space in the program.

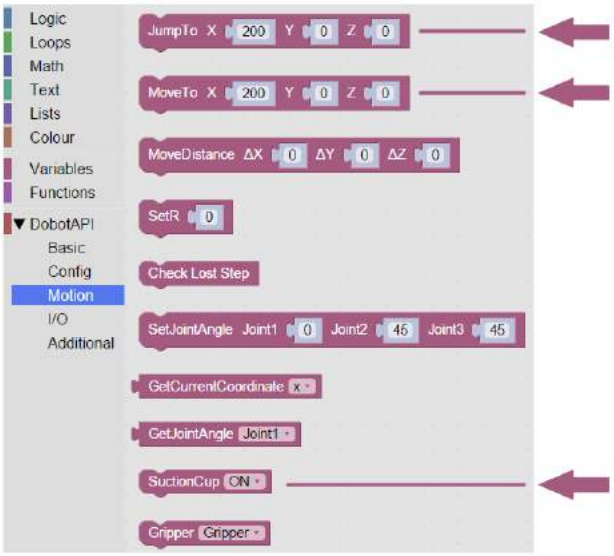
 The Crosshair Icon can be used to reset the zoom level back to default and center your code in the middle of the field.



In this program we will be using the **DobotAPI-Motion** sections in order to create a simple version of a Pick and Place program



In this section there are blocks of code that allow you to move the Dobot ranging from the arm itself to the manipulator. For this program, we will need the ***MoveTo, JumpTo and SuctionCup[ON]*** blocks.

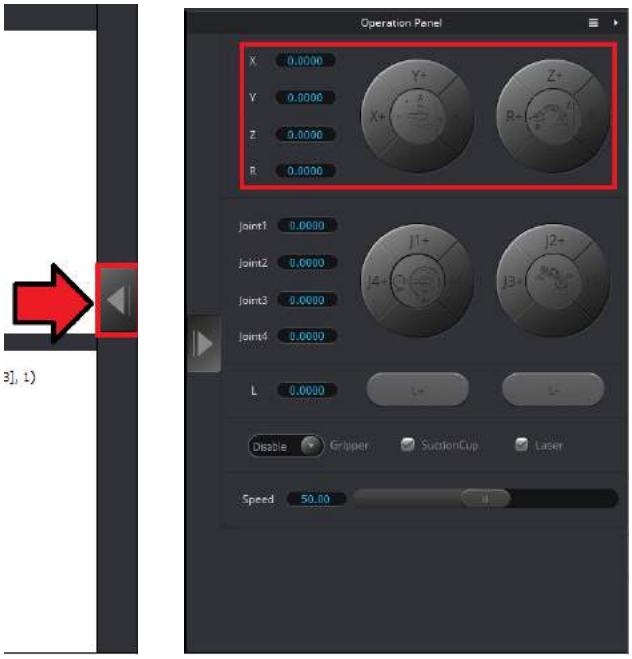


HELPFUL TIPS

For helpful definitions of all the important blockly commands, please see the glossary.

The first step in the Blockly Pick and Place process will be to define the X, Y, and Z coordinates for each step.

Expand the *Operation Panel*.

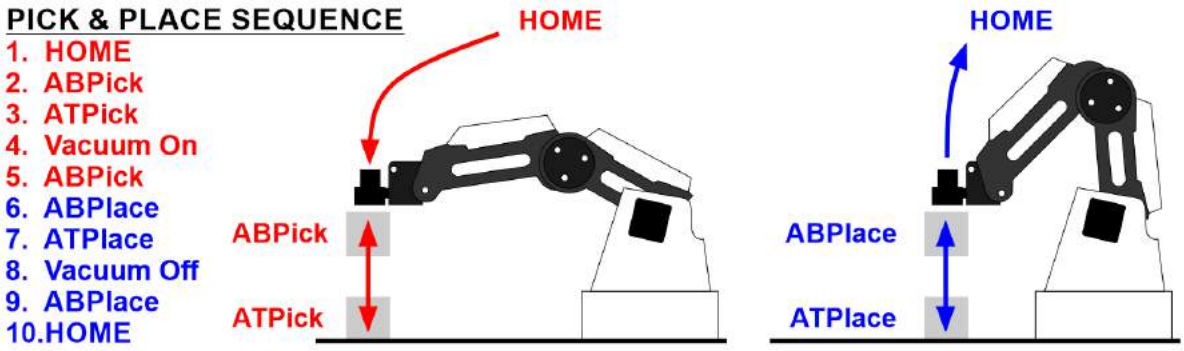


Use the X, Y, and Z Jog controls to locate the coordinates for each position and record them in the chart below. You can also use the Lock button to manually locate positions as we did in the Teach and Playback programming.

You may also use the lock button, and then look at the coordinates in this window to see what the XYZ values of the points are.

HELPFUL TIPS

Touch up the points so that all of the corresponding positions xyz values are aligned.



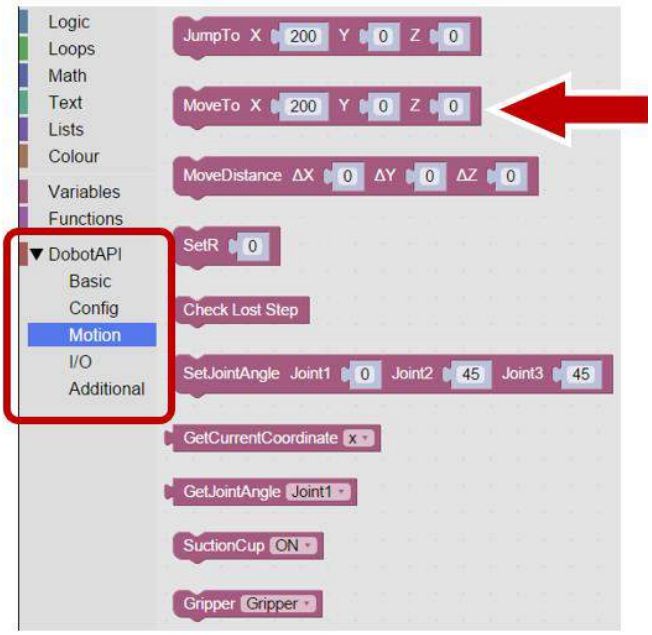
Complete the table below with all of the XYZ coordinates needed.

	X	Y	Z
1. Home			
2. Above Pick			
3. At Pick			
5. Above Pick			
6. Above Place			
7. At Place			
9. Above Place			
10. Home			

Now that all of the positions have been documented, we will transfer their locations in ***MoveTo*** blocks.

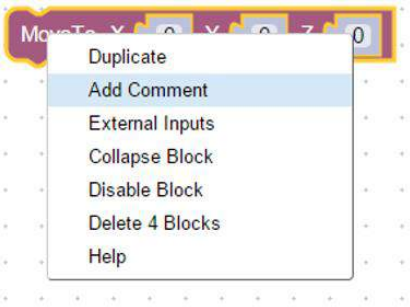
Drag over a ***MoveTo*** block from the *DobotAPI/Motion* Tool Box in the programming field.

This first block is going to be the HOME or start of the program. For this move, we want the robot to always go to a safe or reset position before continuing on with the program. This coordinate or point will be called home and it does not necessarily mean the position the robot is at when the home button is clicked.



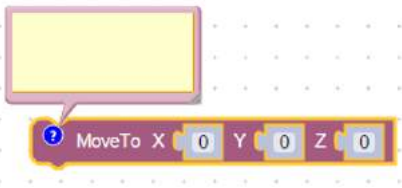
The Home position should be a position above the environment in which the robot is working, and safely away from any objects.

Once the coordinates for the home position have been recorded in the ***MoveTo*** command, right click on the command and select *Add Comment*.



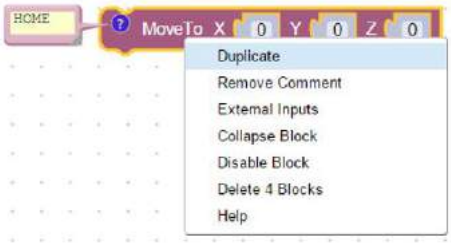
> Please note the values in the images are placeholders, fake values to be changed later on, and that you will need to replace them with your own values that you get from your positions.

Click on the Question Mark that appears. This will allow you to name the position.



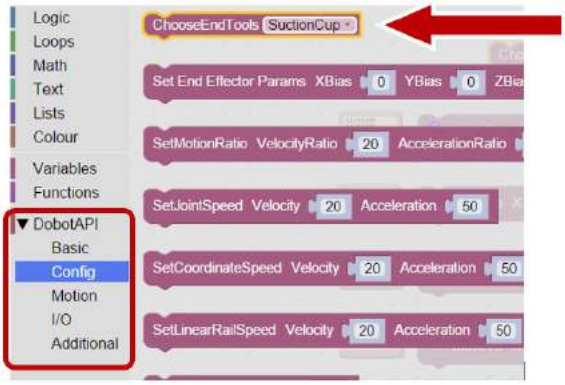
The Text Box that appears can be relocated and resized as needed. Clicking on the Question Mark again will collapse the call out box

Repeat the process for the remaining steps in the program. Ignore steps 4 and 8 (Vacuum On and Off) for now.



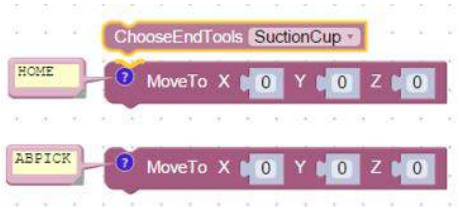
You can also right click on a step and select Duplicate to create a copy of the step. This may speed up the process for steps that have similar coordinates

We will now assign the *End of Arm Tooling (EoAT)* as the Suction Cup

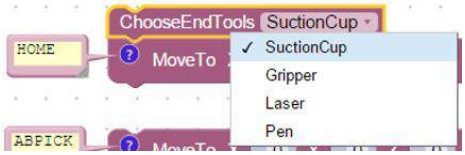


Drag the ***ChooseEndTools*** from the DobotAPI/Config Tool Box over to the programming field.

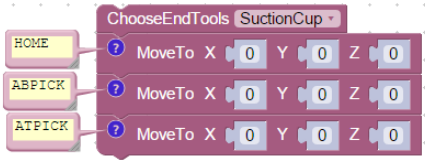
Drag the ***ChooseEndTools*** over to the 1st Home position until the links on both lines of programming turn orange and release the block code.



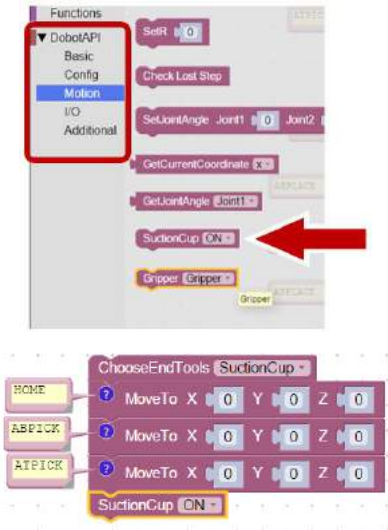
The ***SuctionCup*** should already be selected. If it is not, please select it from the drop down menu.



Link the next lines of code just before Step 4 - Turn on Vacuum

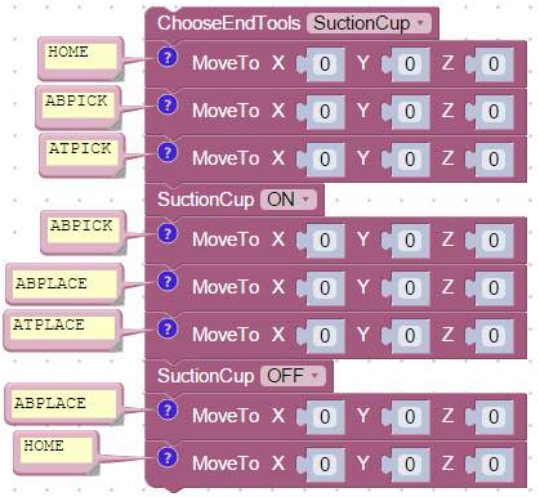


Drag the ***SuctionCup*** over and link it to the bottom of the Main block grouping.

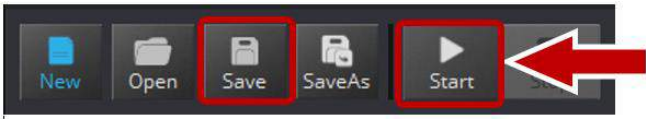


When dragging around blocks of code, if you select the top block, it will allow you to drag around that block and all of the blocks connected below it. If you selected a block of code from the middle of a string of blocks, it will disconnect that block and all of the blocks below it.

Complete the remaining portion of the program. Remember to add one additional ***SuctionCup*** command to turn off the Vacuum after placing the block down.

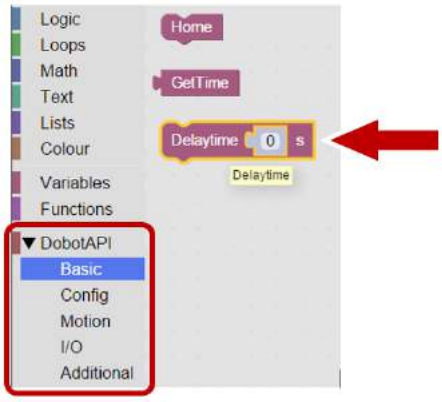


Save your work and select *Start* to run your current program.



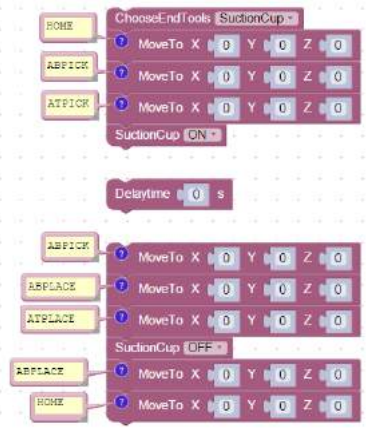
You should notice that we have the same issue in *Blockly* that we had in *Teach and Playback* where the suction cup has not fully engaged as the arm moves away from the ATPick location. We need to solve this problem in the same manner by adding a pause in the line of code to allow the vacuum to build up.

Drag and drop the ***Delaytime*** block over the the programming field.

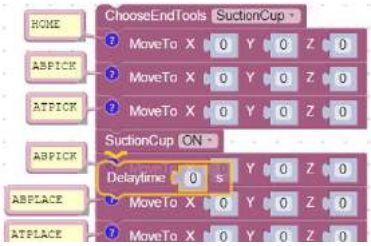


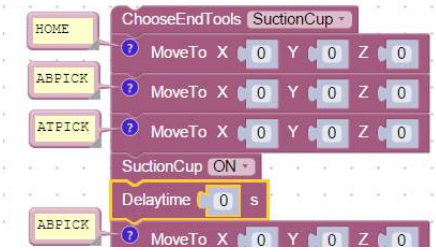
The block of code can be added in one of two different ways.

Option 1: Separate the lines of code selecting the ABPick, below the ***SuctionCup On*** command, and dragging it down. The ***Delaytime*** can now be added and the bottom of the code reassembled.

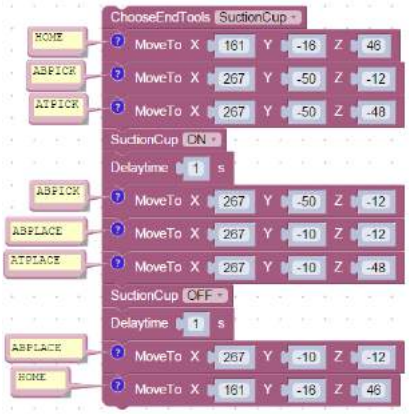


Option 2: Drag the ***Delaytime*** over to its desired location. Wait for the connections to turn orange and release the code. This will automatically insert the line of code and shift the remaining code down.





Add an additional ***Delaytime*** after the vacuum is turned off. Again, Test your Code. The Delay time should be adjusted appropriately.



If your set up did not work correctly the first time, what did you have to do to make it work?

CONCLUSION

1. *What are the five needed positions for a pick and place operation?*

2. *Explain in your own words why wit was necessary to add delay times into the program in the space below.*

3. *What is the purpose of the safe positions that are programmed above the object before it is picked up.*

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

_____ 1. Reverse the process so that at the end the robot puts the cube/cylinder back in its original position

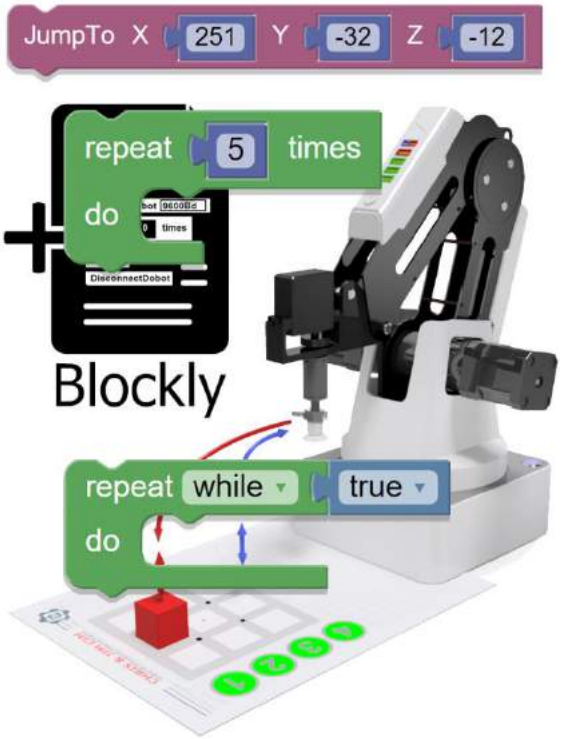
_____ 2. Try picking and placing the object on locations that are not directly in front of the robot. What does this change? Can this be corrected in Blockly as it was in Teach and Playback?

2 Blockly - PnP with Jumps and Loops

NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

When programming a robotic arm, it often becomes necessary to repeat operations a set number of times or indefinitely. This can be accomplished by adding different styles of loops to our program. It is also a good programming habit to optimize or reduce your lines of code when appropriate.



In this activity you will learn one way to simplify your code as well as add different styles of loops to a basic Pick and Place operation in blockly.

The two main types of loops are:

- Forever loops
- While loops
- Repeat

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

KEY VOCABULARY

- Forever Loop
- While Loop
- Repeat
- True
- Jump
- Placeholder
- Condition

All blockly commands have been put into a separate document called *Blockly Vocabulary*, and can be referred to at any time throughout all of these activities.

EQUIPMENT & SUPPLIES

- Robot Magician
- Dobot Field Diagram
- 1” cubes or cylinders.
- DobotStudio software
- Suction Cup Gripper

ESSENTIAL QUESTIONS

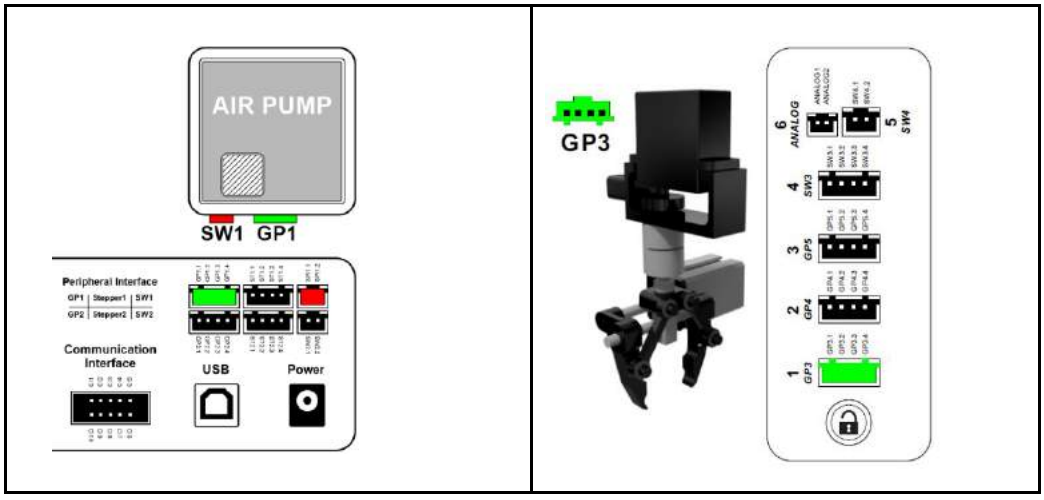
Essential questions answered in this activity include:

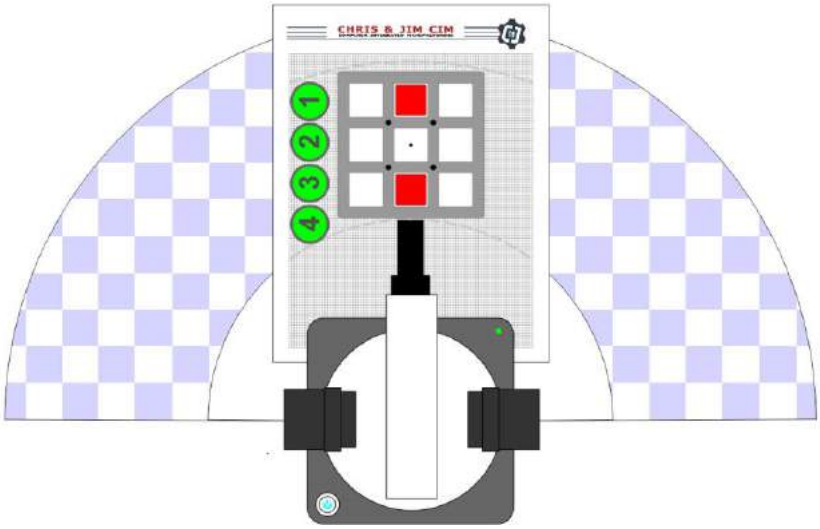
- What’s a *Jump*, and why is it important?
- How to make the robot repeat a motion or a task?
- When would I use a *While* statement?
- How do I use an *Until* statement?
- How do I set jump height?

PROCEDURE

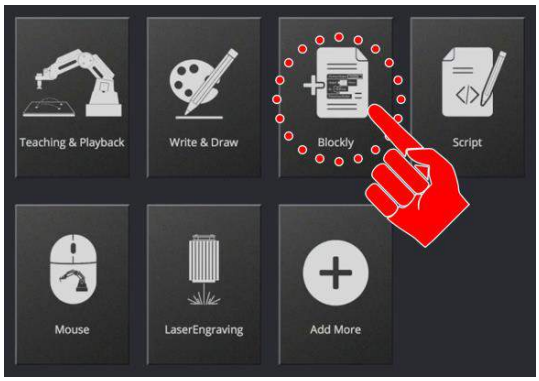
Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

1. Print the Blockly - Pick and Place Field Diagram
2. Set up the robot with a suction cup and place a cube in one of the red squares on the field diagram provided





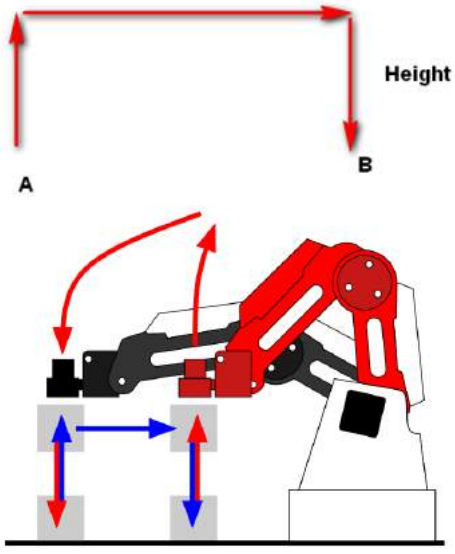
3. Open up Blockly in the software



When DobotStudio is closed with a file open, it will reopen with the last file used. Insure the file open is the one you want to edit, if it does not, you may end up overwriting another program

Open your Pick and Place Blockly Activity.

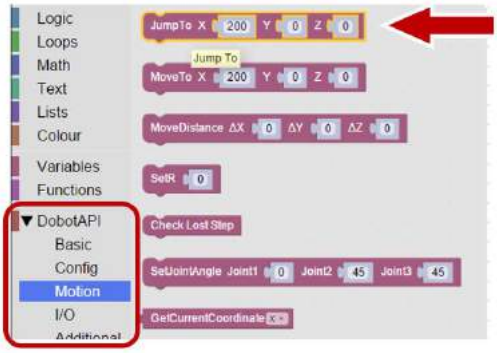
In this activity, we will add a *JUMP* movement as well as a *LOOP*. The JUMP command in blockly works the same as it does in Teach and Playback. This will allow us to remove a few lines of code create the same operation is less steps.



JUMP Movement - A *JUMP* movement combines three steps into one. It combines the raise up, over, and back down to a new point. This type of movement simplifies repetitive movements such as a dipping operation or soldering operation. The Z Height is defined in the JUMP parameters in the settings menu.

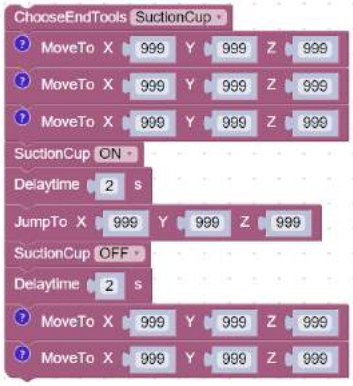
A *JUMP* movement does not replace the initial ABPick to ATPick as well as the finial ABPlace

Remove the lines of code from your previous activity and replace them with a ***JumpTo*** command. The JumpTo command is found in the same DobotAPI/Motion Tool box



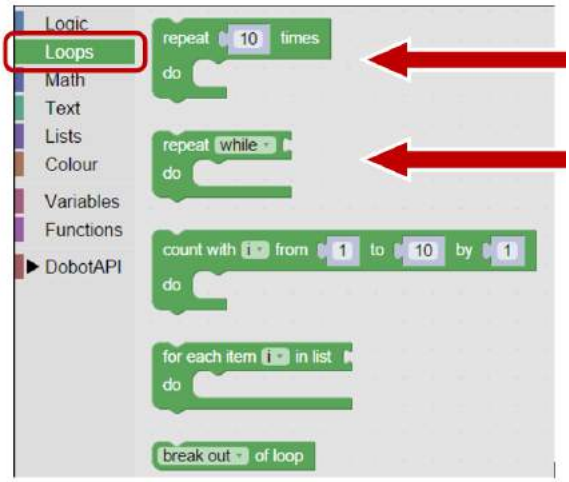
Adjust the X, Y, and Z movements as needed

Run your code to ensure proper operation.

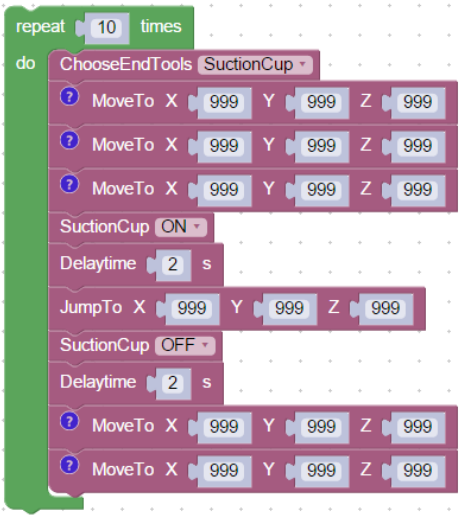


If your set up did not work correctly the first time, what did you have to do to make it work?

Next we are now going to make the program loop, or restart, for a certain number of times. To do this we will be using the ***repeat*** block and the *while* loop, both can be found in the *Loops* section on the left.



The ***repeat*** block is very easy to use. Drag the ***repeat*** block into a program environment. It can be dragged into the empty space or dragged directly onto the top connection for the ***ChooseEndTools***. If the Repeat loop is dragged onto the empty space, grab the top line of the existing program and pull it into the repeat loop. The ***repeat*** loop will automatically expand to encompass the entire code. The program will now run for the set number of times specified in the repeat.



Run the program again. To ensure the *Repeat Loop* works as designed. You will need to manually replace the cube or cylinder at the start of each of the loops.

If your set up did not work correctly the first time, what did you have to do to make it work?

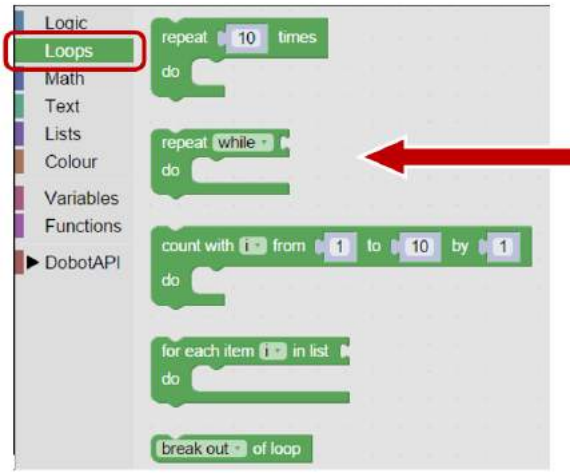
Many times in programming, we need to add a *CONDITION* to our loops. A condition is the scenario that must be met to start a loop. Many times we need something to keep repeating until we stop the program, this type of loop is considered a forever loop and can be made using *while loops.* To create a forever loop for this program, we will add a loop that needs a condition instead of just having it repeat a set number of times.

In order to do this, first drag the code out of the ***repeat*** block and then delete it.

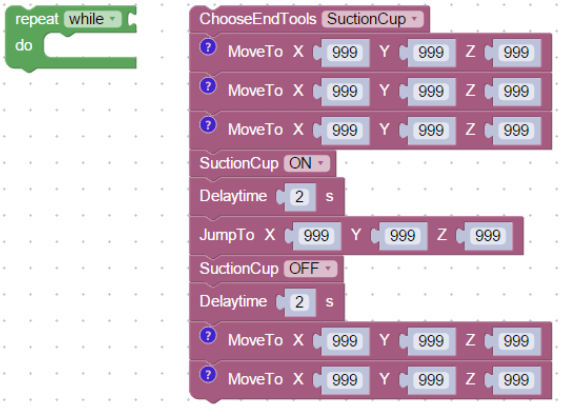
HELPFUL TIPS

*If you select the **Repeat** Loop to be deleted before the program section is pulled out of the loop, it will delete the loop and everything it contains.*

Go to the *Loops Section* and drag the ***Repeat While*** loop over and drag the rest of the program into it.

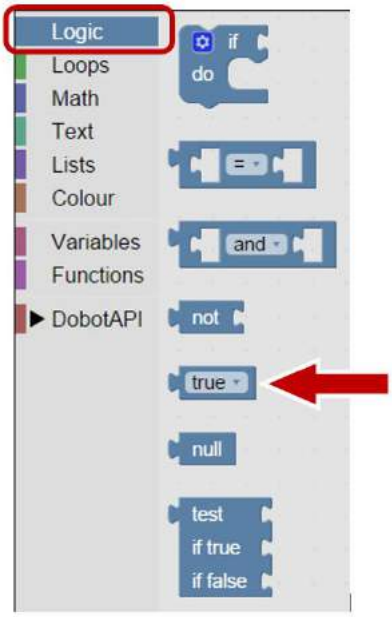


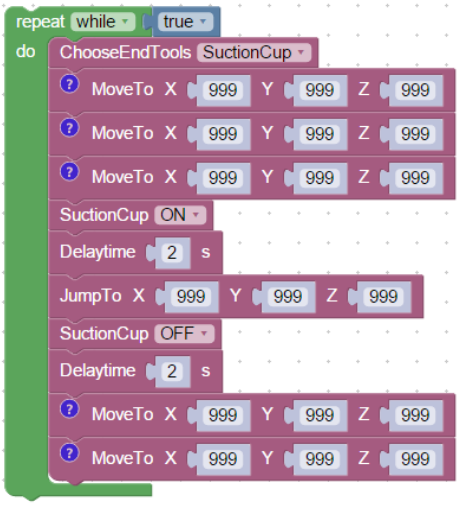
Repeat While statements are different than ***Repeat*** a number of times as a Repeat while requires a condition to be true or false to determine if they run or not, and the Repeat Times only requires a number of times to repeat/loop; no conditions must be met.



A *while loop* will run until it's condition is false so in order to run something forever, the statement must always be true. For example, 1 = 1, 4<5, 100>3, 1 is not 2. These statements will always be true no matter what happens and we could use these for our while loop but there is an easier way. We can simply have the statement be *True*, as True can never be False.

In the *Logic Section*, you will see a ***True*** block. Drag this over to the puzzle link next to the top of the while loop and it should snap into place. This will cause the loop to run forever as statement can never be false





Once again, run the program and you should see that once the program drops off the block and returns to it's safe position, it goes back to the start of the program to try to pick up another block and that this will continue repeating indefinitely until you hit the stop button or the robot is turned off.

If your set up did not work correctly the first time, what did you have to do to make it work?

CONCLUSION

1. *What is the difference between a forever loop and a repeat loop?*

2. *Can a loop be placed inside another loop? Give an example of how you might use this when programming in Blockly.*

3. *Describe in your own words how a JumpTo command works. Why is it good programming practice to use JumpTo's?*

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

_____ 1. Nesting Loops

Produce 2 different pick and place routines

Routine 1 will repeat three times

Then routine 2 will repeat twice

This process will loop routine 1 and then 2 forever

_____ 2. Produce a repeat loop for a dipping operation. Use the JumpTo when appropriate

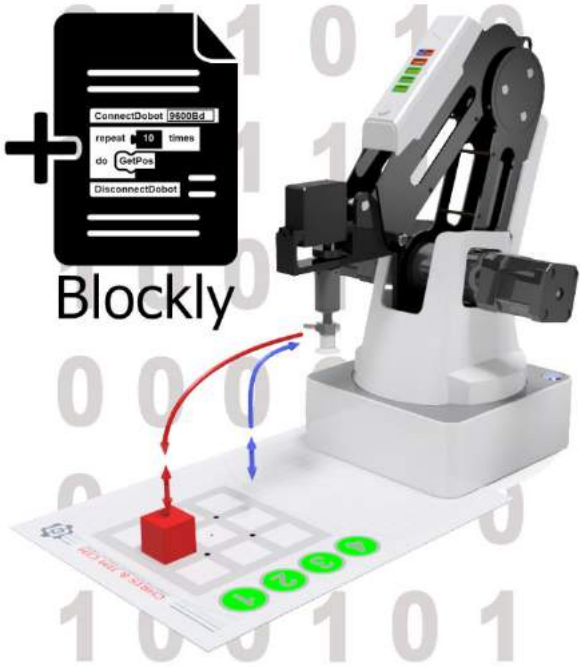
3 Blockly - Pick and Place with Inputs

NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Often it is necessary for a robotic arm to wait for another machine or process to finish before moving with its program. This can be done by adding the ability for a robot to read *input* values.

In this activity you will learn how to add an input to a basic Pick and Place operation in blockly. Through this you will learn how to program the robot to move, turn on it's suction cup, and how to set up *Inputs* in Blockly.



Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

KEY VOCABULARY

- Forever Loop
- While Loop
- Placeholder
- Multiplexing
- Inputs
-

EQUIPMENT & SUPPLIES

- Robot Magician
- Dobot Field Diagram
- 1" cubes or cylinders
- DobotStudio software
- Suction Cup Gripper
- *Dobot Input/Output Guide*

ESSENTIAL QUESTIONS

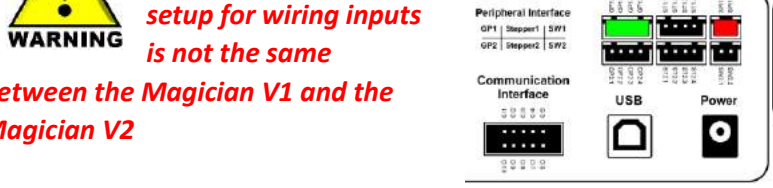
Essential questions answered in this activity include:

- What's the difference between an *input* and an *output* in robotics?
- What are some examples of *digital inputs*?
- How do I code a digital switch as an *input* in blockly?

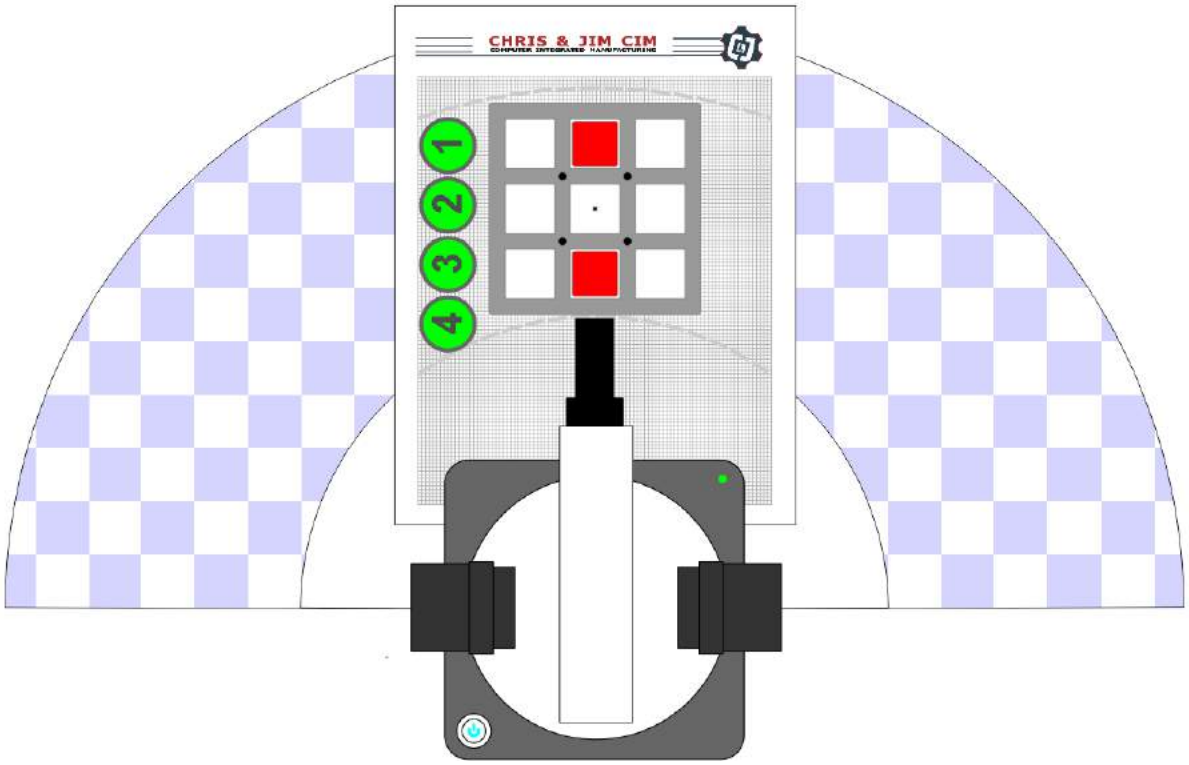
PROCEDURE

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

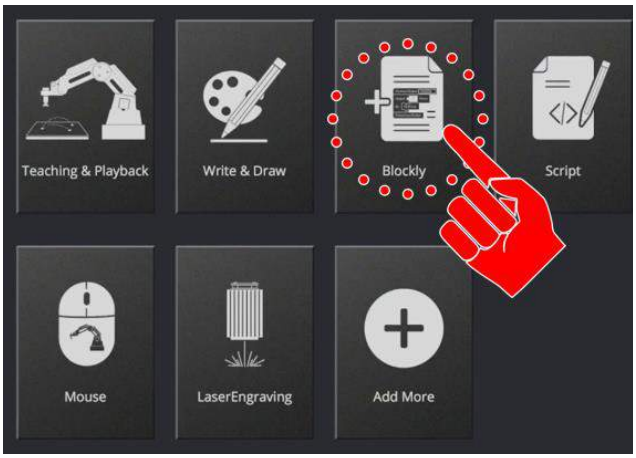
1. Print the Blockly - Pick and Place Field Diagram
2. Set up the robot with a suction cup and place a cube in one of the red squares on the field diagram provided
3. Wire the robot such that the LIMIT SWITCH is plugged into port GP5.



Caution: Please Refer to the Dobot Input/Output Guide for wiring your input. The setup for wiring inputs is not the same between the Magician V1 and the Magician V2



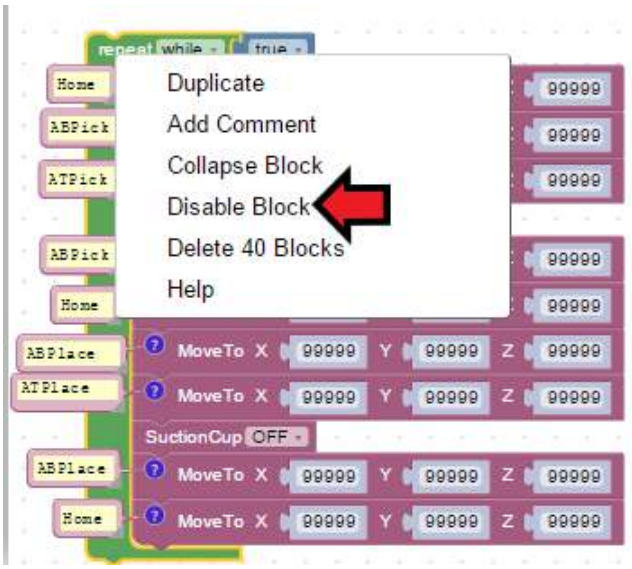
4. Open up Blockly in the software and Open your Blockly - Pick and Place file from the previous activity.



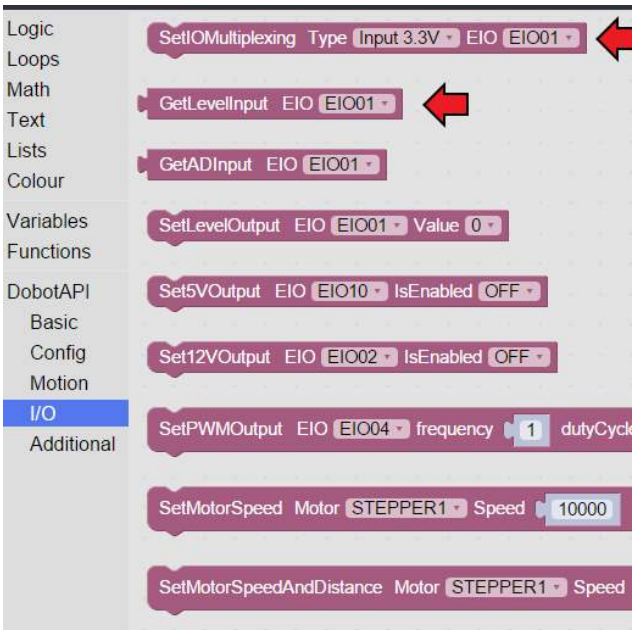
HELPFUL TIPS

When you re-open this program check that the name of the file on top matches the code in the file, if it does not, you may end up overwriting another program

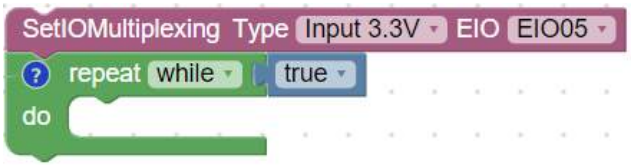
In this activity we are going to add the ability for the program to wait for an input and then have the program run whenever a limit switch is pressed. To do this, you must prepare a limit switch that can work on the Dobot, as done in Activity 5, and plug it into GP5 (PORT3). **Make sure the robot is off before you plug or unplug anything from the robot.** After that, we are going to right click our current while loop and click *Disable Block.* What this does is that it grays out the current selected block and any blocks inside of it, making the program not read it but still having it there incase you want it back later on.



We are now going to setup a program to make sure the dobot is reading the input signal from switch. This is done by having the Dobot print the value of the switch to the *Running Log* on the right of the screen. First though, we must tell the robot that there is an input we are expecting and we do this by using the block ***SetIOMultiplexing***. To see the value we use the *GetLevelInput block*

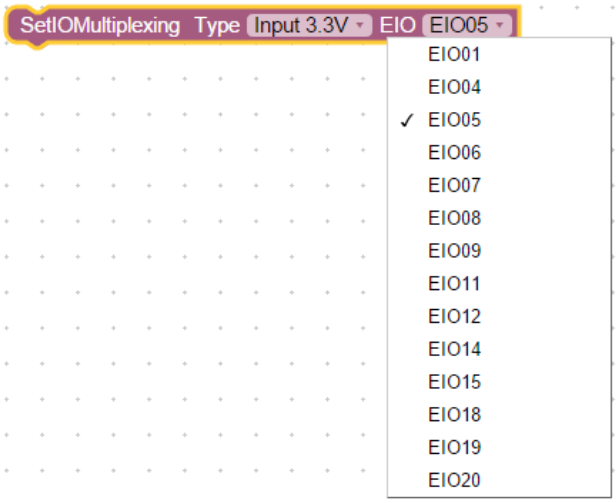


Drag a ***SetIOMultiplexing*** block over and it will be the start of this program. Change the EIO port to EIO05 which is equivalent to GP5 (PORT3), you may use another port but make sure to consult the *Dobot Input/Output Guide* as some ports may damage the Dobot or other equipment. Then create a Forever loop below it.

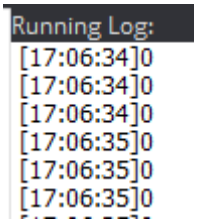


Be sure to consult the Dobot Input/Output Guide if you want to use other inputs and outputs, as damage to your robot or your other equipment may result.

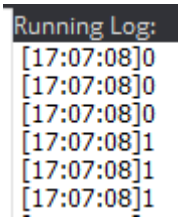
Next we are going to have the program print the status of the switch. To do this we are going to need the ***print*** block, this can be found in the *text* section near the bottom. Drag the ***print*** block into the Forever loop and then drag the ***GetLevelInput*** block, that was indicated before, and drag it into the right side of the ***print*** block. Make sure to change it to the correct port in order for it to work.



Now run the program. In the running log you should see it printing 0's or 1's while the switch is not pressed and it should swap printing to either a 1 or a 0 while it is pressed.



N.O. - If you see 0's that change to 1's, your switch or the Dobot is wired N.O. - Normally Open. The input will read Low or False until the switch is pressed.
N.C. - If you see 1's that change to 0's, your switch or the Dobot is wired N.C. - Normally Closed. The input will read high or true until the switch is pressed.



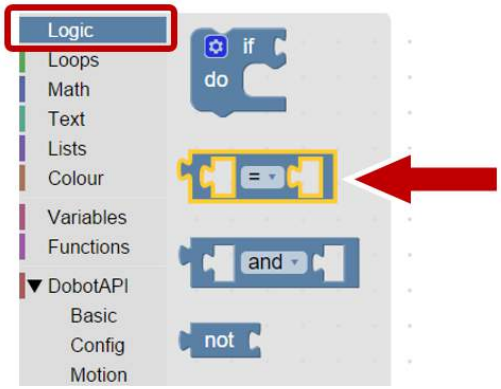
If this is not the case, you may be plugged into the wrong port, have the wrong port selected for the IO setup, or have the wiring for the switch incorrect. Please refer to the Dobot Input/Output Guide. for your setup and try again. ***Remember to turn the power off to the robot if you switch any wires.***

If your setup did not work correctly the first time, what did you have to do to make it work?

Now that we are able to see the switch, get rid of the ***Print*** block. We are now going to make a while loop that will run as long as the switch is pressed. Drag another ***while*** loop into the *Forever loop* but instead of dragging in a *True* block as we did previously, we are going to go to the *Logic* section and drag over a ***[blank] = [blank]*** block. This block is how we check if an item is equal, greater, lesser, or not equal to another item. In the left side of the block, drag in the *GetLevelInput* block that we just used. We then are going to go to the *Math* Section and grab a ***number*** block, and drag that into the right side of the block and change it into a 1.

Create the *Nested Loop* as seen above (Steps provided below). A *nested loop* is a loop within another loop. We will place the robot's pick and place movement commands inside the second loop that is only called to start if the robot sees a change in the input value on input EIO05. Refer to your testing to whether the robot should start the movement on a 1 or a 0.

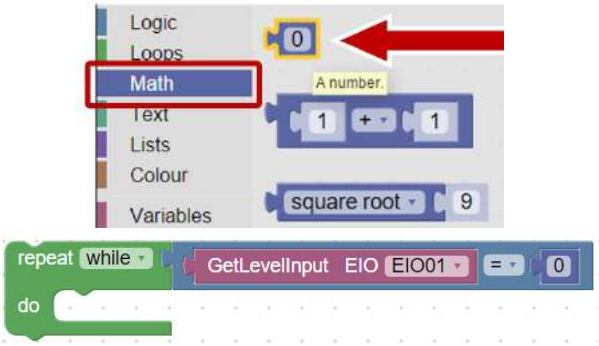
Attach a ***ReturnTrue*** from the *Logic* section. This condition will return a true value if both inputs are equal as the condition for the second while loop



Insert a ***GetLevelInput*** from the *DobotAPI* section and add that as the first condition for the ***ReturnTrue*** statement. Change the Input value to EIO05.

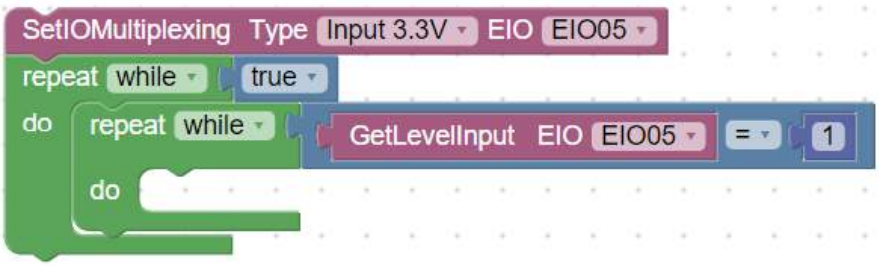
Insert an ***AddNumber*** condition as the second part of the ***ReturnTrue*** statement.

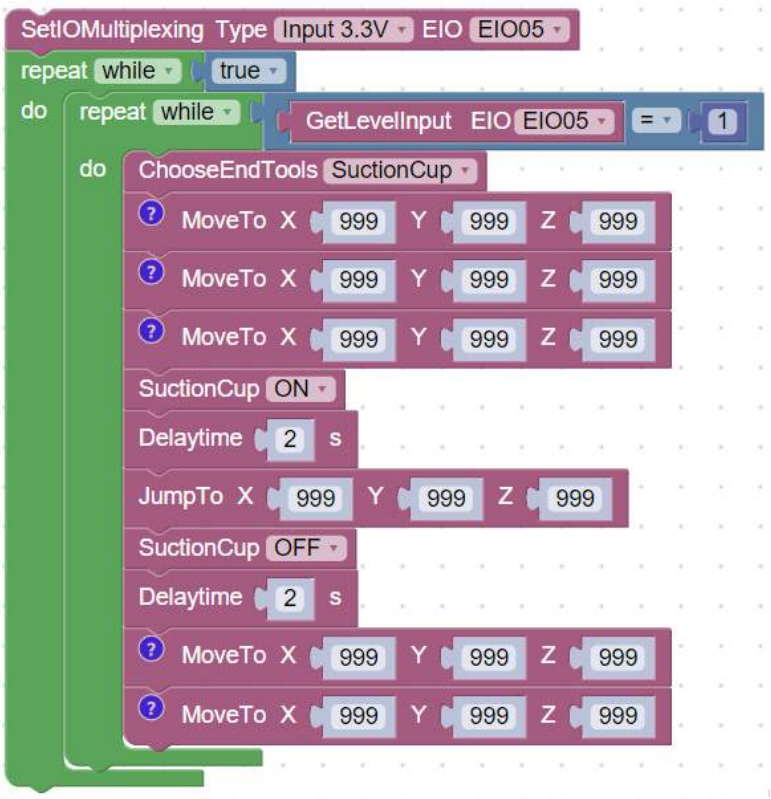
Change the value of the ***AddNumber*** to the value that is needed to start your loop (0 or 1)



Once a loop has started, it will continue to finish ALL of the steps in the loop even if the condition that started the loop becomes false. The condition to start the loop is only evaluated at the beginning of each loop

Now that the *nested loop* is setup, we want to re-enable our previous code for the pick and place routine. To do this, right click on the loop again and click *enable*. Once it is enabled, drag the movement code from the old loop into the new loop and then delete or disable the old loop. In the end it should like this but with real values instead of placeholders.





Once the program is written, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

If your set up did not work correctly the first time, what did you have to do to make it work?

CONCLUSION

1. *What would happen if the loop detecting the switch wasn't in a forever loop? Try it and describe what happens. (Hint: Make sure to be holding the switch when you start the program)*

2. What happens to the Pick and Place process when the switch is released (The loop condition *becomes false)?*

3. *In your own words, define a nested loop.*

4. *In your own words, define a condition.*

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

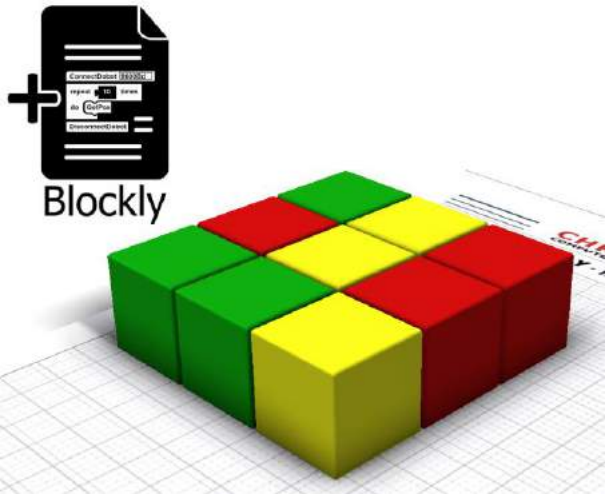
_____ 1. Break the pick and place activity into multiple sections that will wait for the limit switch before performing each step.

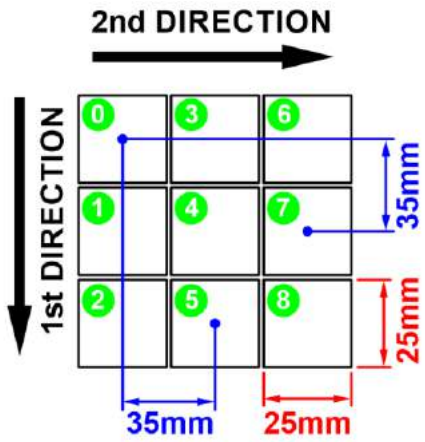
4 Blockly - Developing a Cube Matrix

NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Matrices come in many different sizes throughout industry and can be used in other areas besides robotics. Many companies will use matrices in order to efficiently store products and materials, so it is important for you to know how they work and how to program one yourself. It is especially helpful for palletizing routines when placing boxes on a pallet efficiently.





Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

KEY VOCABULARY

- Forever Loop
- Repeat Loop
- Matrix
- SuctionCup
- Delaytime
- ReturnSum Math Block
- Function
- User-Defined Function
- Inputs
- Variable
- MoveTo
- NumberBlock

EQUIPMENT & SUPPLIES

- Robot Magician
- Dobot Field Diagram
- 1” cylinders or cubes
- DobotStudio software
- Suction Cup Gripper
- *Dobot Input/Output Guide*

ESSENTIAL QUESTIONS

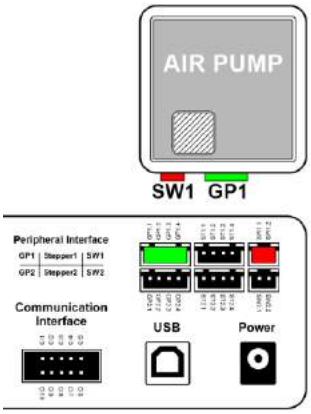
Essential questions answered in this activity include:

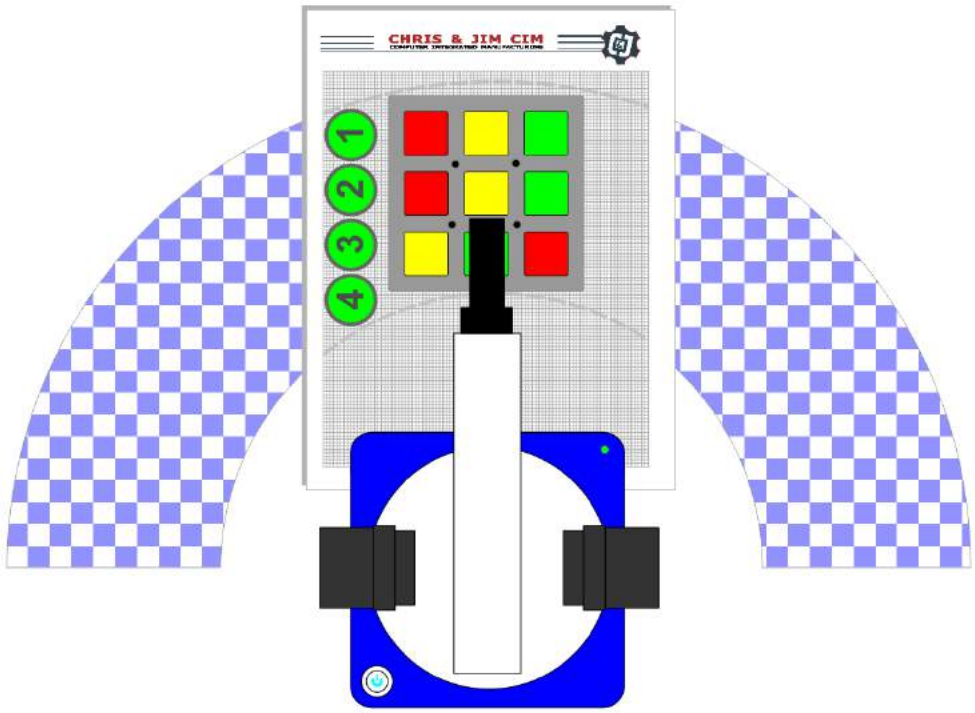
- Why are variables used?
- How do I use variables in blockly?
- How can I use math to palletize?
- What are some of the blockly commands needed to do this?
- Why is palletization and de-palletization important in industry?

PROCEDURE

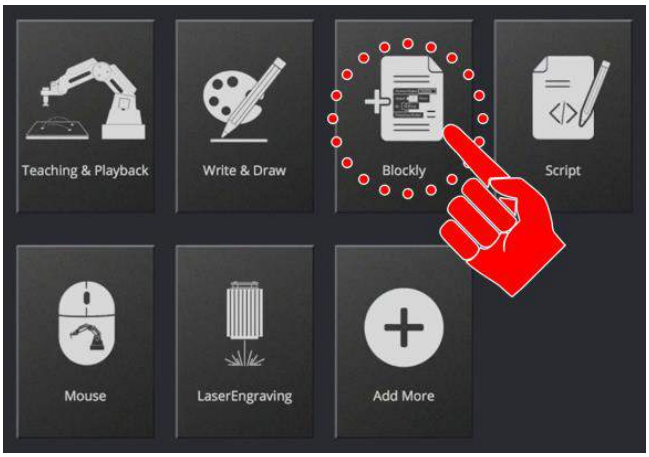
Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

1. Print the Dip Tank Field Diagram
2. Set up the robot with a suction cup and place a cube on all of the empty squares on the field diagram provided. *(Cube color does not matter for this Activity)*





3. Open up Blockly in the software and start a new program.



HELPFUL TIPS

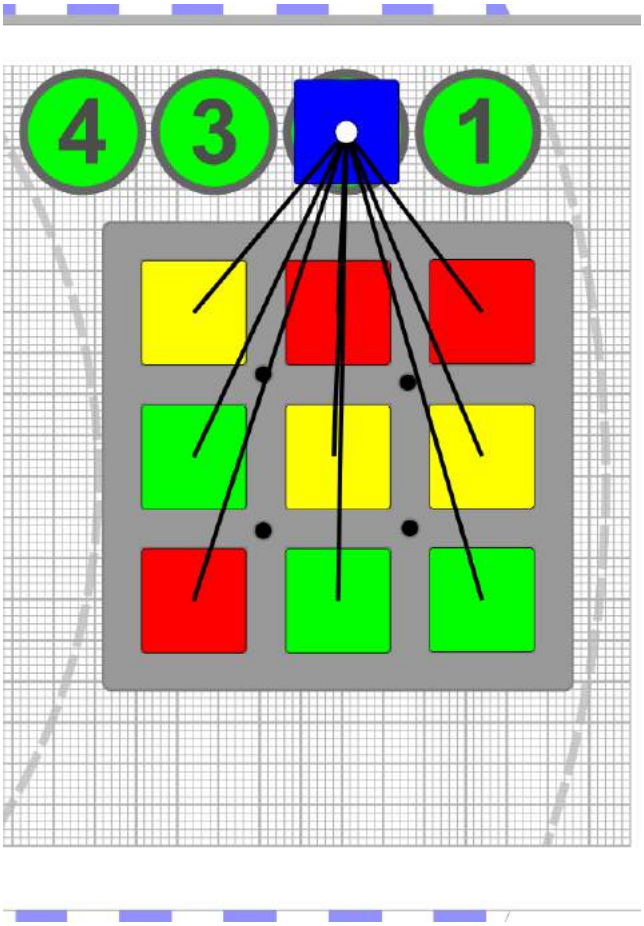
When you re-open this program check that the name of the file on top matches the code in the file, if it does not, you may end up overwriting another program

Programming the robot to pick up each cube from a matrix of cubes and drop them all off at a common location is not necessarily hard to do, just tedious, time consuming, and numerous lines of code that are repetitive motions. This same process can be done in reverse; such as when you move parts from a feeder and place them in a matrix. This is called *palletization*.

From this activity we can learn how to condense and simplify our program when we have things in common or repeated.

In this exercise we have the following commonalities:

- Above Z locations are the same
- At Z locations are the same
- Place X and Y locations are the same
- They are all EVENLY spaced in a regular matrix.



The only thing that keeps changing is the X and Y coordinates of the cubes in the matrix field.

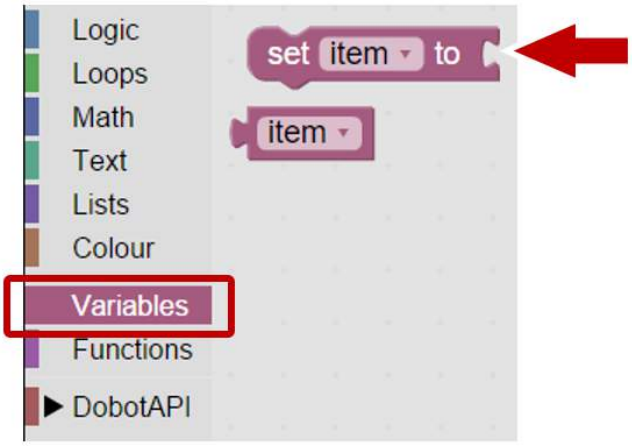
We can use these common factors to simplify our operations and our code. We will do this with user defined *variables* and *functions*.

Functions are a named section of a program that performs a task that will be repeated over and over again in our program. This is a procedure or a routine that will greatly simplify our otherwise complicated program.

Functions:

- **Group of Code:** Can be several lines of code that is needed to complete a single task or operation. Allows a programmer to group thoughts or actions.
- **Use Multiple Times:** Functions can be called to run multiple times throughout a program.
- **Simplifies Programming:** Allows repeated code to be condensed it a single line of code throughout the main program.
- **Simplifies Editing.** Code that repeats itself multiple times throughout a program now only needs to be edited once.

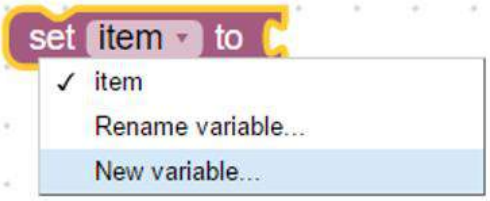
We will start our program by defining a set of variables. *Variables* are a changeable quantity in a program that can be represented by a word or a letter. Variables can be assigned, changed, or referenced throughout a program.



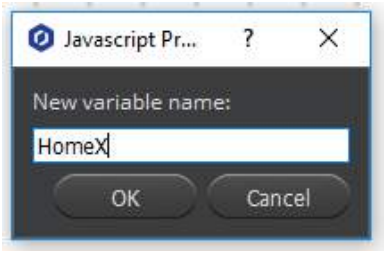
List of variables to define

- HOME X Value
- HOME Y Value
- HOME Z Value
- PICK X Value
- PICK Y Value
- PLACE X Value
- PLACE Y Value
- COMMON Z ABOVE Value
- COMMON Z AT Value

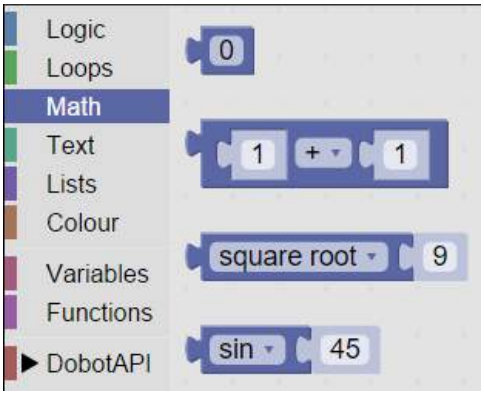
Drag out a ***set item to variable*** from the *Variables* section.



Click on the "item" portion of the variable tool and select *New variable*. This will be our HomeX value.



Next, from the *Math* section, drag over a ***Number*** block and attach it to the end of our variable.



The next step is to start finding each of our starting values.

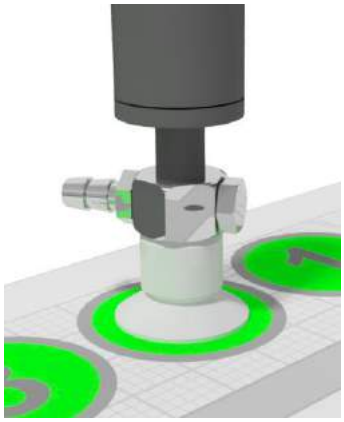
Expand the Operation Panel

Using either the Jog Controls, or the Lock Button find the values for the table below.

It makes it easier to use the Lock Button to get close to the desired position and then use the Jog Controls for smaller movements.

HELPFUL TIPS

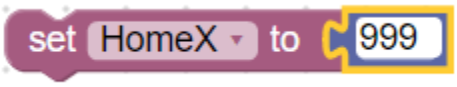
- ***Use Cube '0', from the diagram on page 1, as the X and Y location for the Pick Location.***
- ***Round all values to the nearest whole number***
- ***Align the center of the vacuum gripper to the center of the cube***
- ***Use the location of dip dank 2 for the common drop off point***
- ***Ensure the CommonZAbove positions is high enough to clear the other blocks***
- ***Write the values in the table below or in your notebook for future reference.***



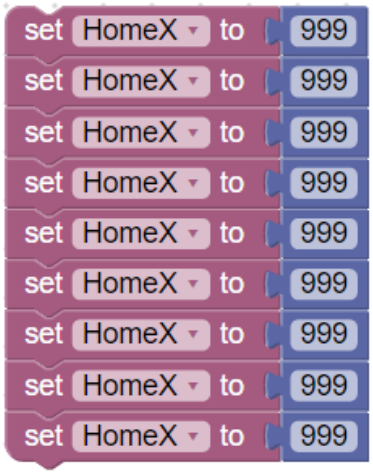
HomeX	
HomeY	
HomeZ	
PickX	
PickY	
PlaceX	
PlaceY	
CommonZAt	
CommonZAbove	

Part 1: Setting Variables

Start by recording the HomeX position in the number window of the block we created



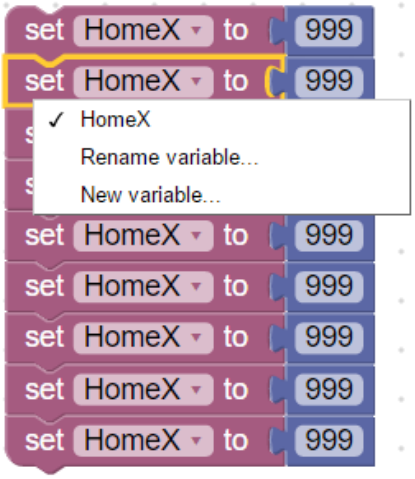
Duplicate the HomeX block until you have 9 variables total.



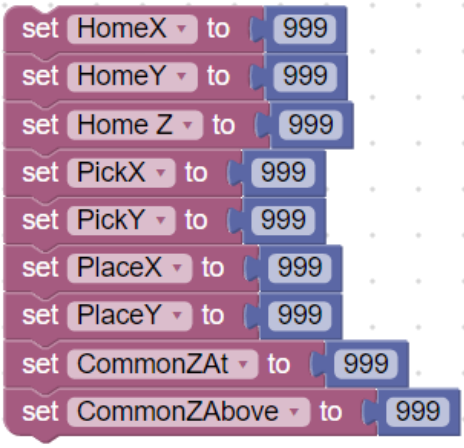
Left Click on the next variable name and select New Variable.

Create New Variables for each block and fill in their values.

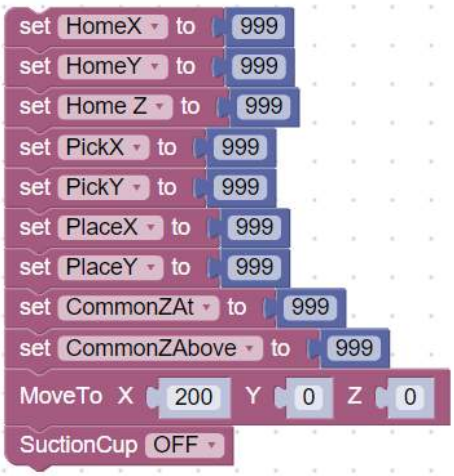
If you select Rename Variable, it will only Rename that value and every where it has been used in the program



The order in which we set the *variables* does not matter in this program.



The first actual movement is for the robot to always go home and turn off the vacuum. Please drag and add these blocks to the program.

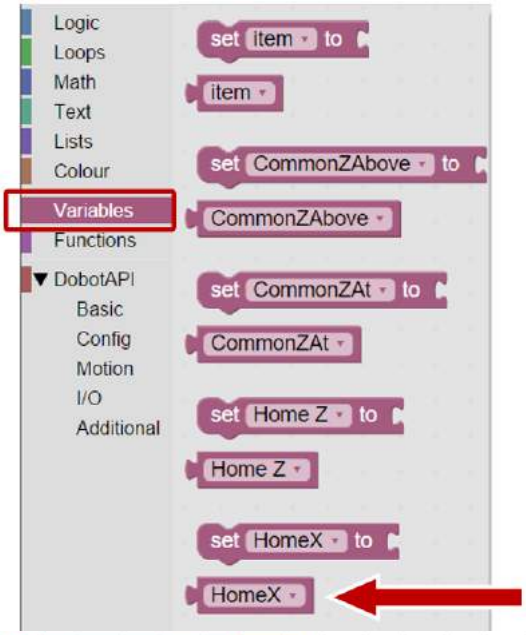


HELPFUL TIPS

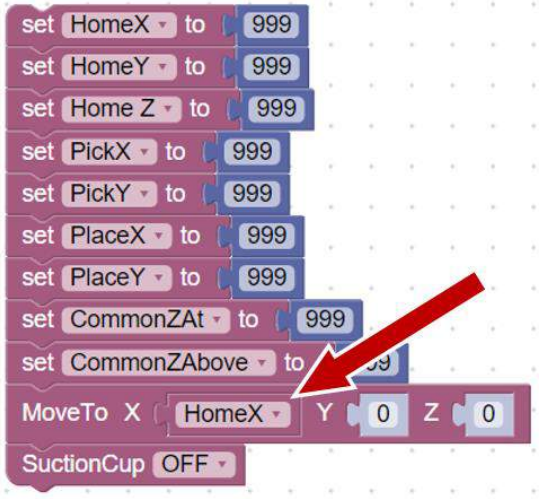
Remember that these tools come from the DobotAPI/Motion section.

We will now use some of the user defined *variables* that we just created.

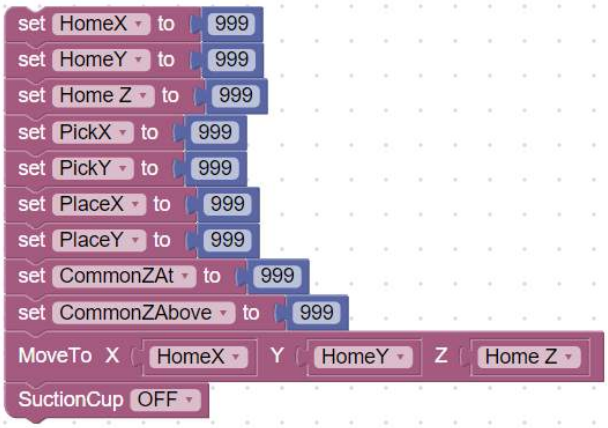
From the Variable section, you can now see that all of the variables that we have created are now options to use.



Select the ***HomeX Variable*** (*not the* ***set HomeX variable***) and drag it into the X position for the ***MoveTo*** step.



Fill in the remaining positions for the MoveTo Home Block as shown.

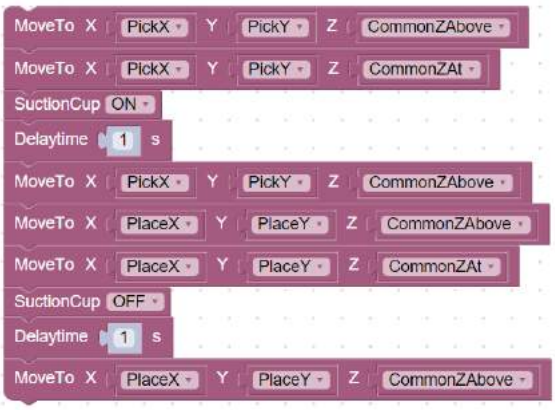


Assigning variables really helps when XYZ coordinates in a program are going to have to change multiple times or is going to be used as a template for multiple programs. The can be altered once at the beginning of the program and that will update everywhere they were used in the program. It also makes reading the program easier as the user now sees words in place of number values.

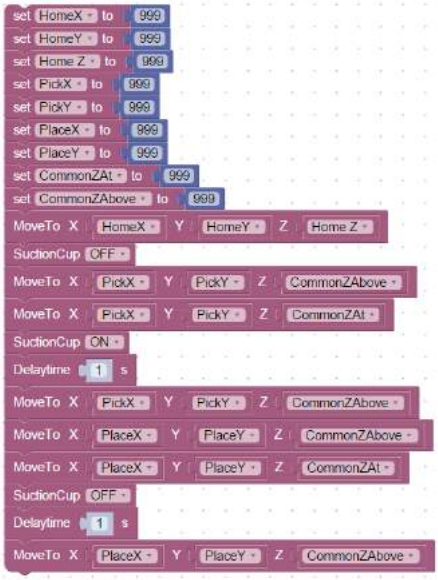
We will now add all the steps needed to go get the first block '0' from the assigned pick location and move it to the place location.

Making duplicates of the existing ***MoveTo Home*** position we just made will create the program faster.

Duplicate and alter other blocks as they are created to save time.



Link both sections together.



Once the program is written, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

Remember: if you hit a limit with the robot at any time, or you hear a clicking or a grinding noise, it's always a good idea to home the robot again. Also, if the robot does not return to the same position, just re-home it.

If your set up did not work correctly the first time, what did you have to do to make it work?

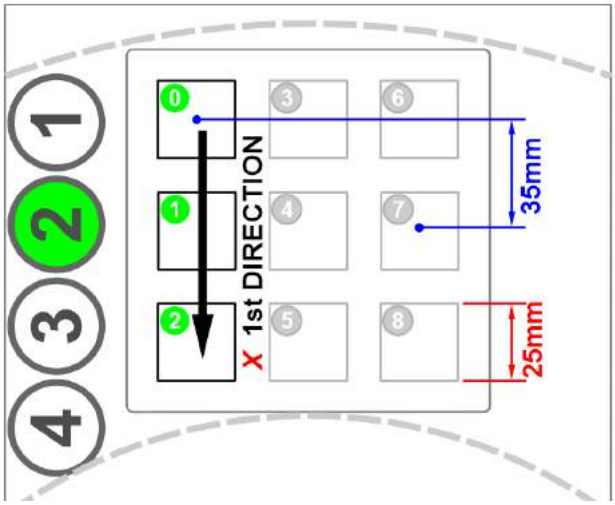
ALL NEW - THE TRICKY PART!!!

Part 2: Adjusting Variables

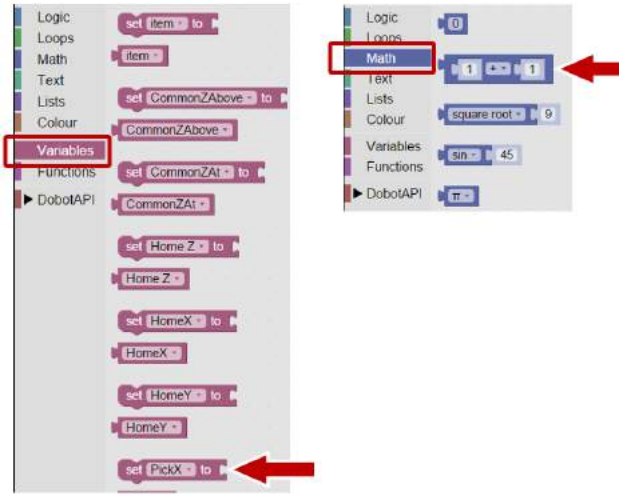
Now it's time for the second cube.

The only difference in the programming for this cube should be the X Position.

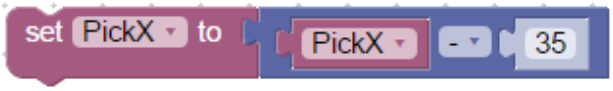
From the diagram shown, the blocks are 35mm apart and that the block is moving *toward* the robot. Moving toward the robot is shown mathematically by subtracting 35mm from the current X position value for block number 2 and again for block number 3.



In order to adjust the X position, set up a block (***Return the sum of two numbers***) from the Math section, and attach it to the ***SetVariable*** block. Attach the two blocks together.

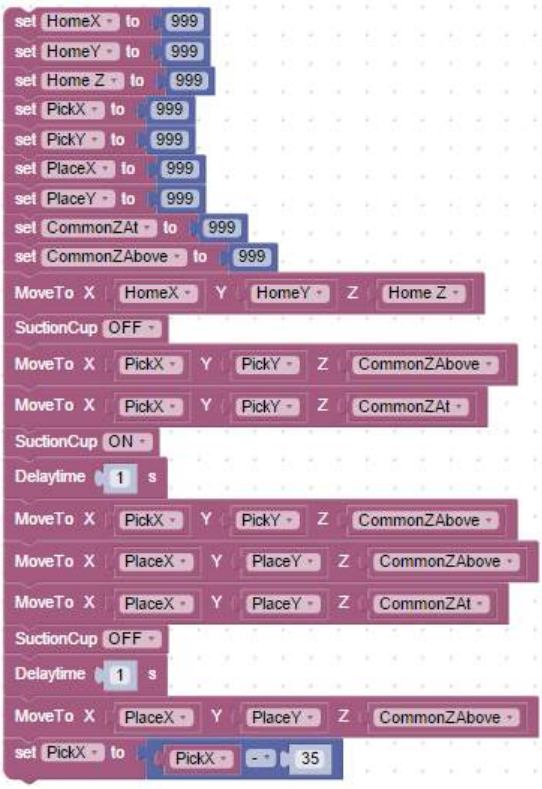


In order to add 35mm to the current PickX value, and assign that value to the old PickX value, replace the first value with PickX and the second value with 35.



Current X value equals itself plus a constant of 35

Mathematically: PickX = PickX - 35



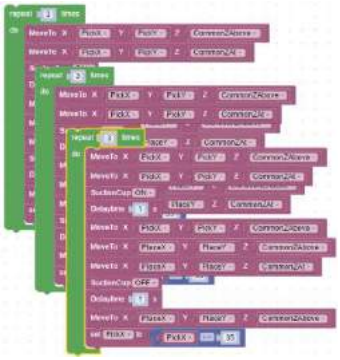
Grab the PickX Variable from the Variables section or duplicate it from the main program

Add this block line to the bottom of your program

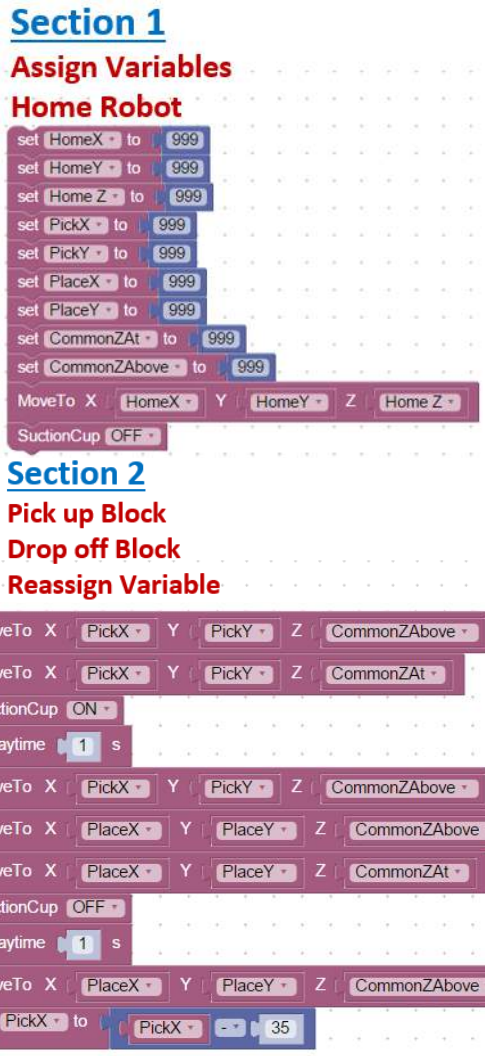
Now that the PickX value has been adjusted, repeat the entire process of picking up the second and third cubes and dropping them off at the place location.

This can be done by recreating all of Section 2 of the program two more times. There is an easier way of duplicating groups of code without actually grouping them. If Section 2 is pulled away from Section 1 (Grab the 1st *MoveTo X*), it can be put into a loop:

If the loop is duplicated, it will duplicate everything inside the loop.



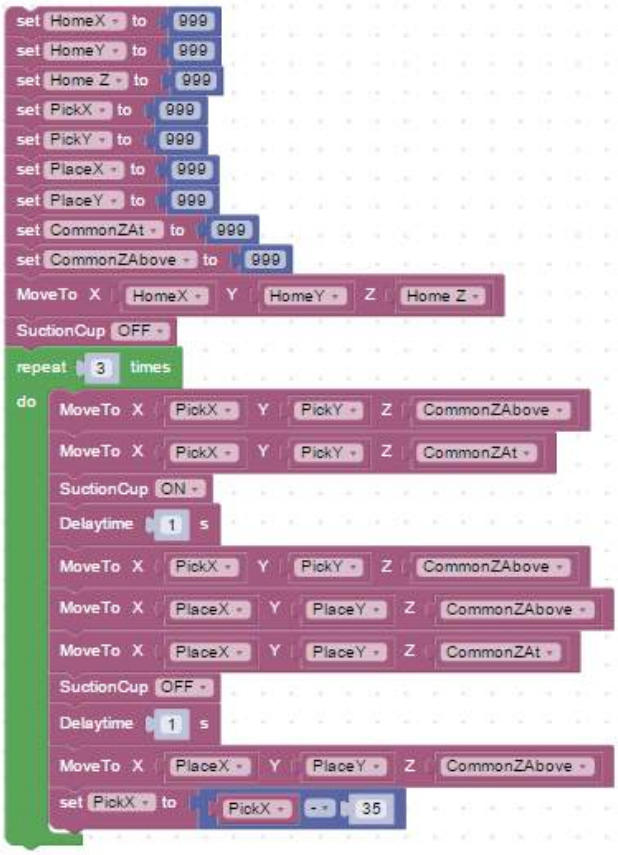
If a loop is deleted before pulling the group of code out of the loop, it will delete the loop and its content.



Using this method the program becomes very long, and we would have to do a lot of scrolling to move around in our program.

Notice too that the same thing is happening three times. Instead, just assign a loop and *repeat* it three times.

Make it repeat three times as shown and try it out to make sure it works with the blocks in spaces 0, 1, and 2.



Since one full column of blocks is completed, just repeat this again for the next two columns.

This can be done by resetting the X value to what it was in the code before and move the Y value over to the next column.

Repeat this block of code with those two changes two more times to complete the entire matrix.

Part 3: Simplification

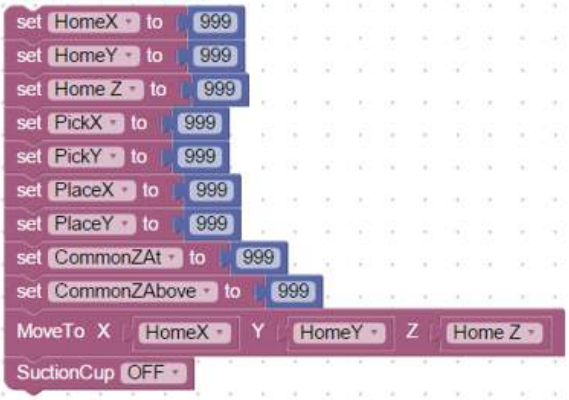
Break your code up into 3 separate parts:

1. Assign all of the original values needed as variables and send the robot to a safe/above home position
2. Pick up the first object using predefined location points from the list of variables. Add or subtract whatever value is needed to go from the center of the start object to the center of the next object and repeat this process in a loop for however many objects are in that specific column.
3. Reassign the start value that was altered in section 2 back to the original value. Add or subtract whatever value is needed to go from the center of the start object in column one to the center of the next object in column 2.

Section 1

Assign Variables

Home Robot

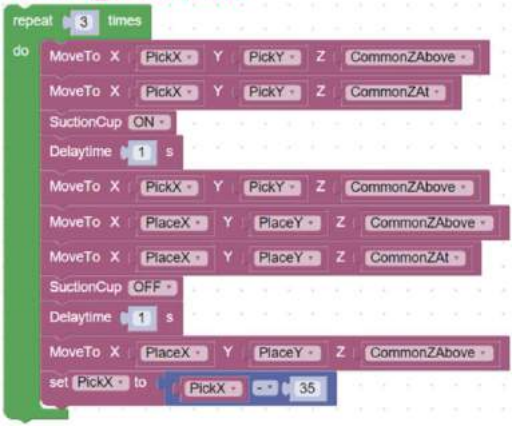


Section 2

Pick up Block

Drop off Block

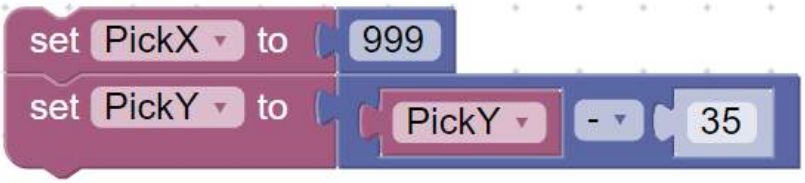
Reassign Variable



Section 3

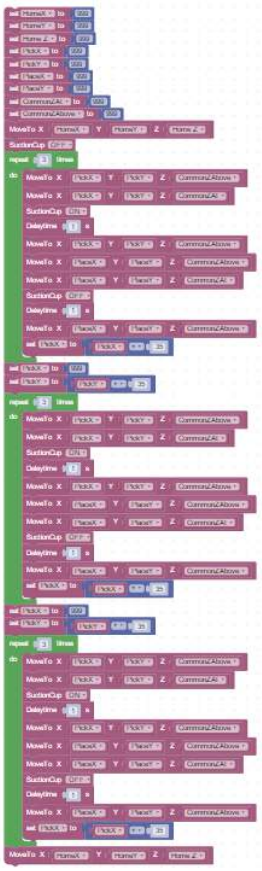
Reassign PickX to the Original Value

Assign PickY to PickY +/- 35



Rather than create a really long code that looks like:

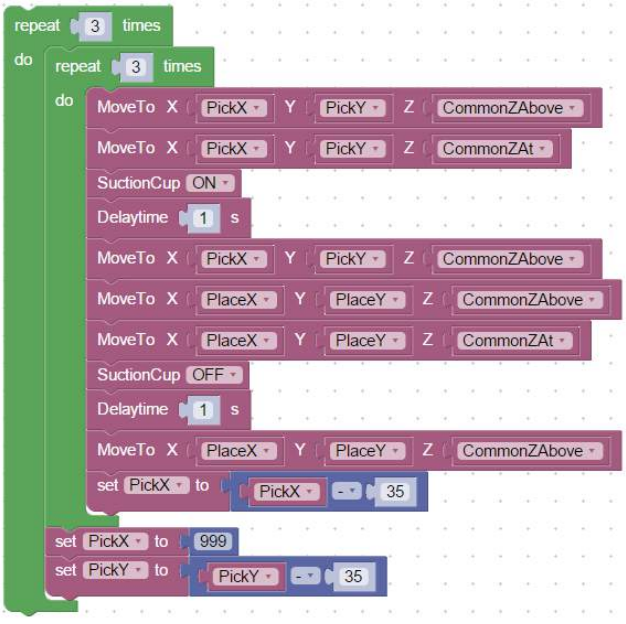
Section 1
Section 2
Section 3
Section 2
Section 3
Section 2
Section 3
Send Robot to Home



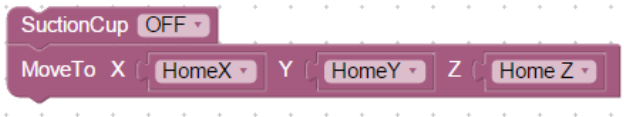
Look to the right to see how long it actually makes our program. This would make it really hard to work with in the programming window as a lot of scrolling would have to be done in order to make any edits.

Simplify it with another loop.

Be sure to add section 3 from above, and reset the variable PickX to its original value, and then subtract 35 from PickY to move it over.



We can then duplicate and add one final SuctionCup Off and a final MoveTo Home to end the code.



Add this to the program, then try to run it.

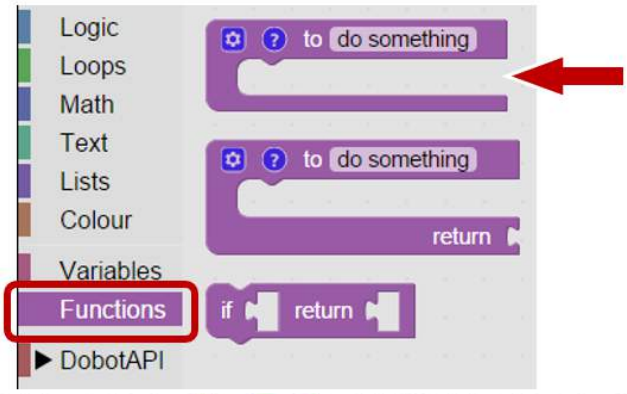
Part 4: Adding Functions

The program is still very long even though variables were used to make it shorter and simpler.

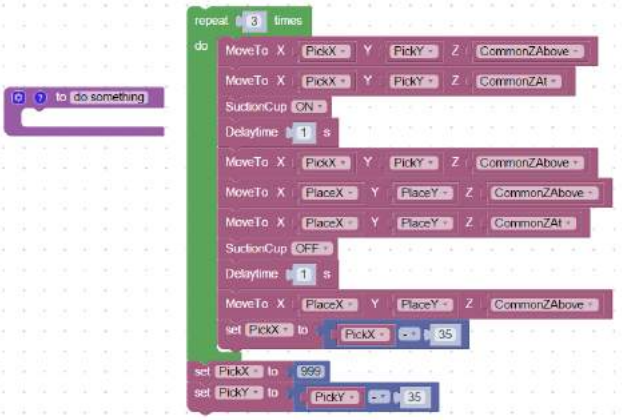
The next step is to simplify it even further with a block of code called a function. A *function* is a named section of a program that performs a task. It can also be considered a procedure or a routine and greatly simplifies otherwise complicated programs.

When a program calls a function to run, the program pauses, runs the function, and then returns to the program where it left off.

Drag over a ***Function*** block with no Output .

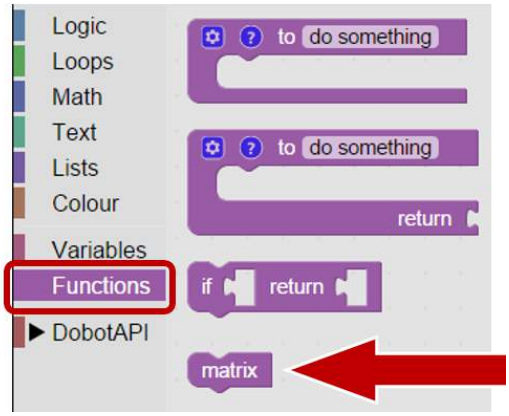


We can pull the ENTIRE matrix looping process into the Function



Give the *Function* a name that describes the process it contains… like Matrix

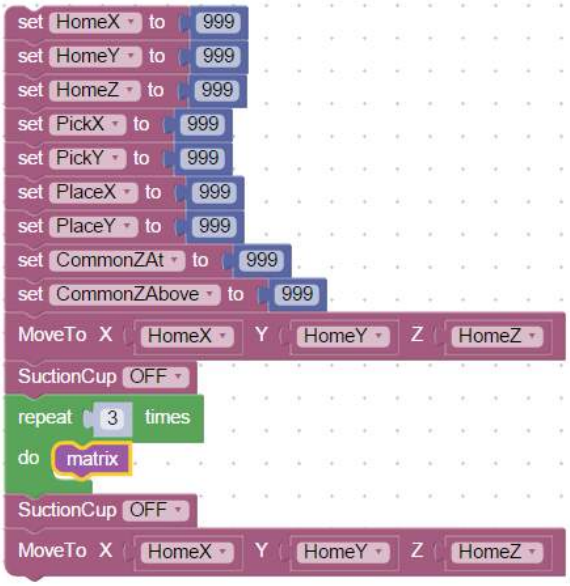
The Function that was just created is now available to be used in the program as many times and wherever needed.

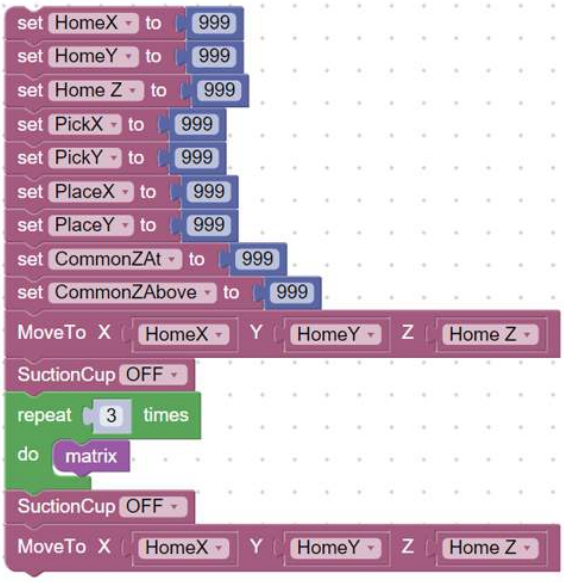


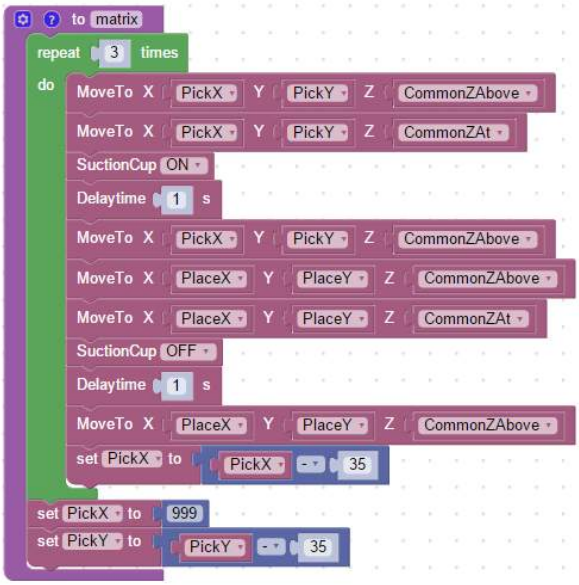
Drag the new *Function* into the Repeat Loop.

The Function just floats in the programming environment unattached to the main program. They are essentially separate mini programs to be called by the main program.

Developing Functions can not only produce groups of code to be used multiple times, but can also help a programmer organize their program into mini programs or separate thoughts.







Once the program is written, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

If your set up did not work correctly the first time, what did you have to do to make it work?

CONCLUSION

1. *In your own words, define a variable.*

2. *In your own words, define a Function.*

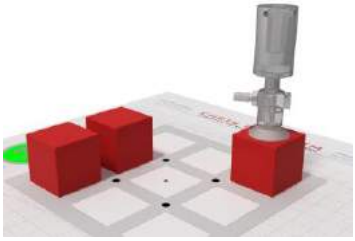
3. *Explain what would have to be done to palletize two layers using bullet points or a step by step list below.*

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

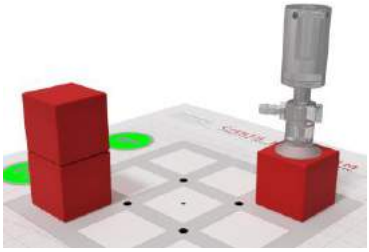
_____ 1. Use a switch as an input to make the robot wait until you move the block from the Place position and hit the switch.

_____ 2. Move a row of three cubes to another row (Matrix to Matrix)



_____ 3. Develop a program that will run the entire process in reverse. Take a block from a common location and distribute them into a 3x3 matrix. This process is referred to as palletizing.

_____ 4. Take the cubes from a common location and stack them in a column vertically.



5 Blockly - Using the Color Sensor

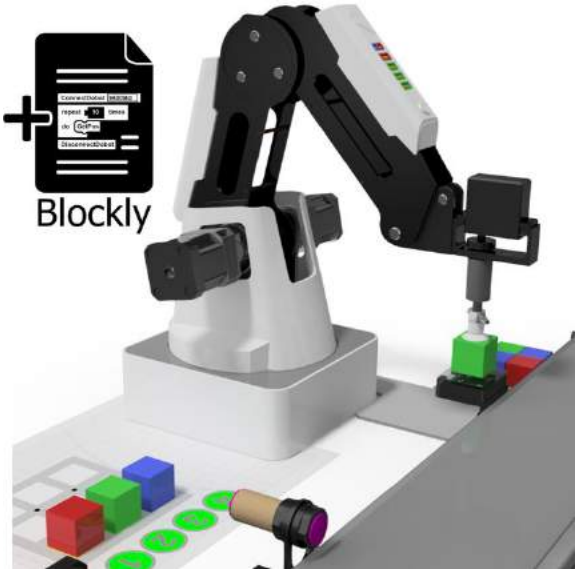
NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Sensors are often added to industrial robots in order for them to perform specific tasks. These sensors can be as simple as a color detecting sensor, or as complex as a full vision system that will allow a robot to be aware of its surroundings, or find a part and determine its location and orientation.

In this activity you will learn how to use and program the color sensor in blocky.

The dobot will pick up a part and move it above the color sensor. The dobot will then check the part's color and place it in a specific location for that specific color part. The robot will repeat the process each time a limit switch is pressed.



Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

KEY VOCABULARY

- Color Sensor
- If / Else If / Else Statement
- List
- Return True
- Function / Voids
- Identify Color
- Sum of List

EQUIPMENT & SUPPLIES

- Robot Magician
- Dobot Field Diagram
- 1" cylinders or 1" cubes (RED, BLUE, GREEN)
- Dobot Color Sensor
- DobotStudio software
- Suction Cup Gripper
- *Dobot Input/Output Guide*
- 3 Small Containers

ESSENTIAL QUESTIONS

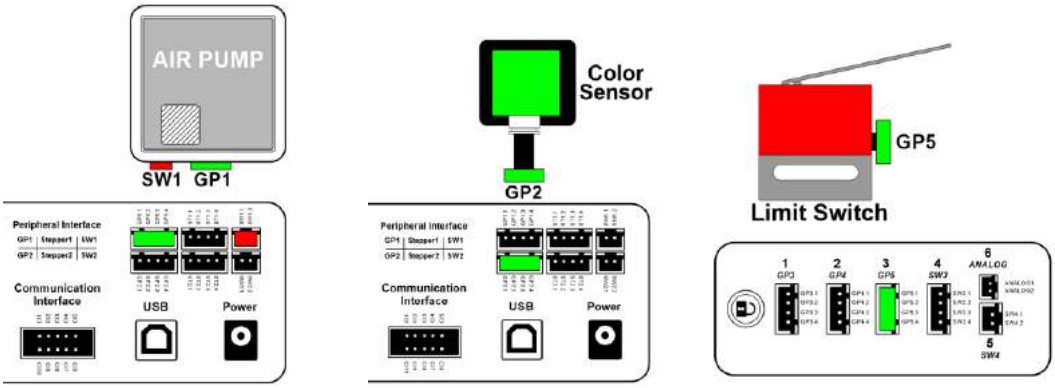
Essential questions answered in this activity include:

- What's the difference between digital and analog?
- What colors can I sense with the color sensor?
- How does the color sensor work?
- How do I wire the components together?
- How do I get a value from the color sensor?
- How do I print to a log?
- How do I sort items by color with a robotic arm?

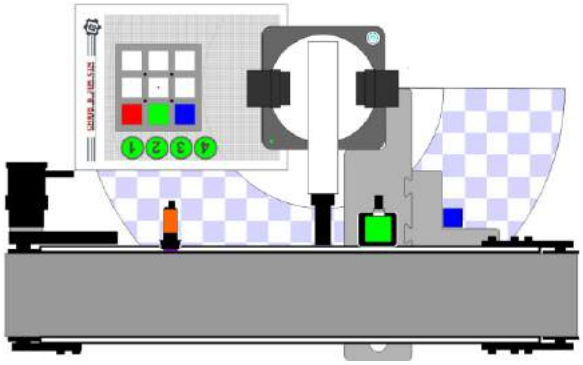
PROCEDURE

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

1. Set up the robot with a suction cup - ***GP1 & SW1***
2. Wire the robot with the Color Sensor plugged into port ***GP2***
3. Wire the robot with the Limit Switch plugged into port ***GP5 - EI05***



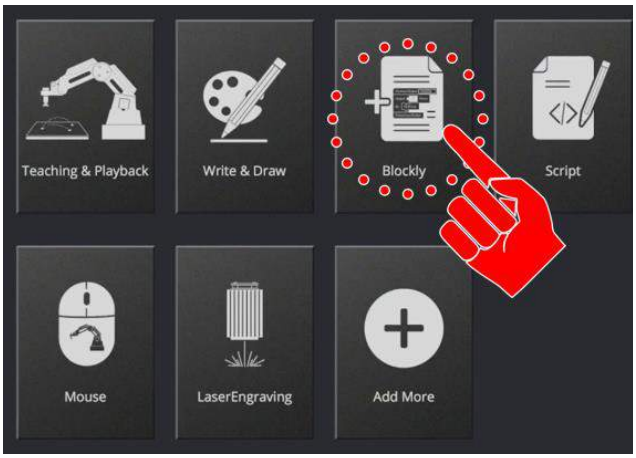
Set up the robot, conveyor, and color sensor as shown in the diagram below:



Order of operations

- Start with the robot a safe/home position.
- Place a single cube, one at a time for the robot to pick up from a common location.
- A limit switch will be used to call for the robot to come and get the block for evaluation.
- After determining its color, drop the cube off at a specific location for that color.
- Manually remove the cube once it has been placed and send the robot to its home position.

Open up Blockly in the software



HELPFUL TIPS

When you re-open this program check that the name of the file on top matches the code in the file, if it does not, you may end up overwriting another program.

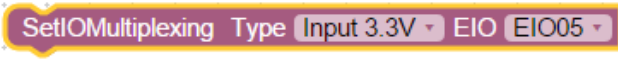
The first step in programming this activity is to set up the inputs.

Drag over the ***SetColorSensor*** from the *DobotAPI/Additional* tool box

Set the sensor to ON / V2 / GP2

Next, set up the limit switch as we have done in past activities.

Input 3.3V GP5 EI05

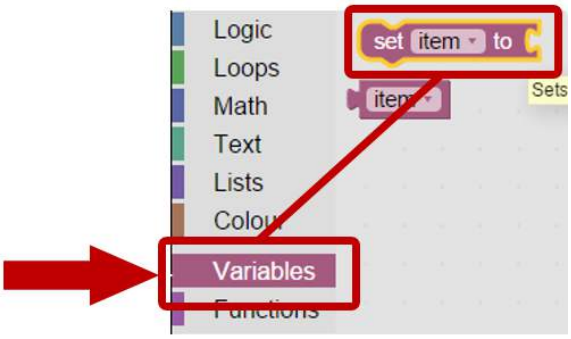


COLOR SENSOR SETUP

The next set of commands are needed to get the values from the color sensor to report the correct values to our program/

Step 1

Bring over and link together three ***Set Variables to Input*** blocks



Select *item* for each block and create a New Variable.

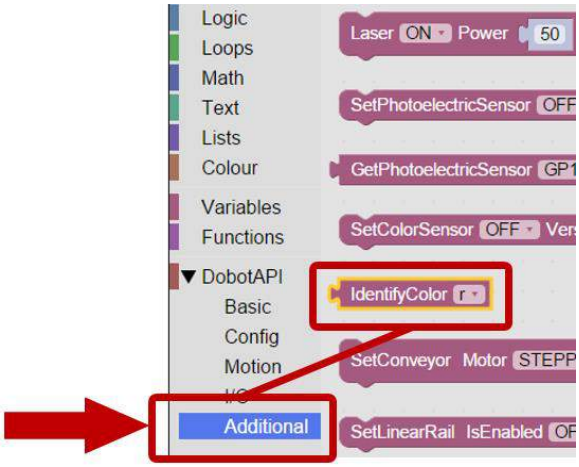
We need to create a *variable* for each color that we will use in our program

Create *variables* named:
RED
GREEN
BLUE

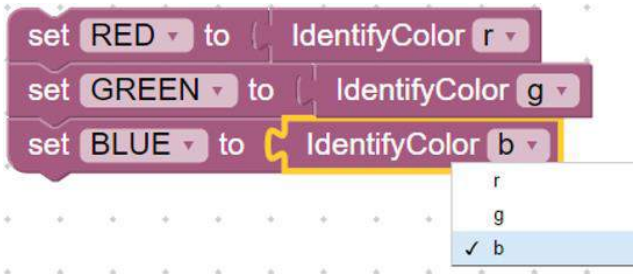
Step 2

The next step is to set each individual variable to a pre-set range collected from the Color Sensor

The ***IdentifyColor*** command is used to identify three basic colors: Blue, Green, & Red. This block returns a true or 1 value when the specific color is identified and a false or 0 when it is not



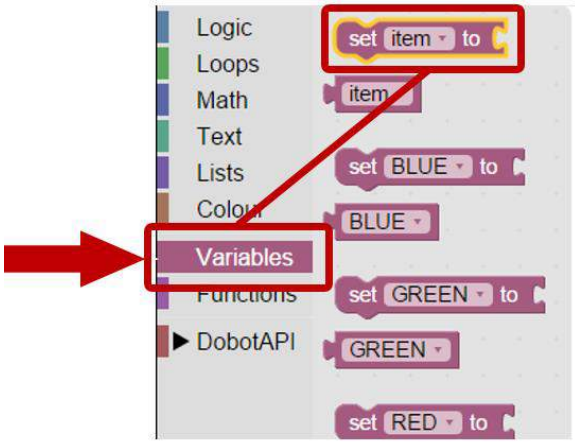
Attach an ***IdentifyColor*** block to each of your ***Variables*** and select the corresponding color letter for each block.

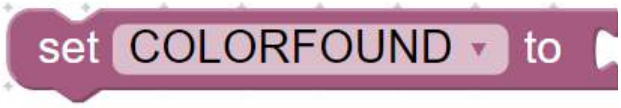


Step 3

Group or consolidate all three color possibilities into one variable that can be used in the program.

Create one last variable called COLORFOUND.



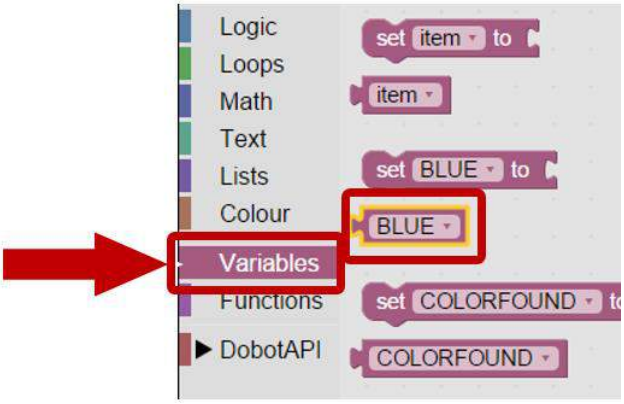


Step 4

Attach each variable (RED, GREEN, and BLUE) to a *LIST*. The *LIST* will allow us to verify that a color is being read. From this command we will not capture which color is being read, we will just verify that a valid color from the list is being read.

Select the ***Create List With*** block

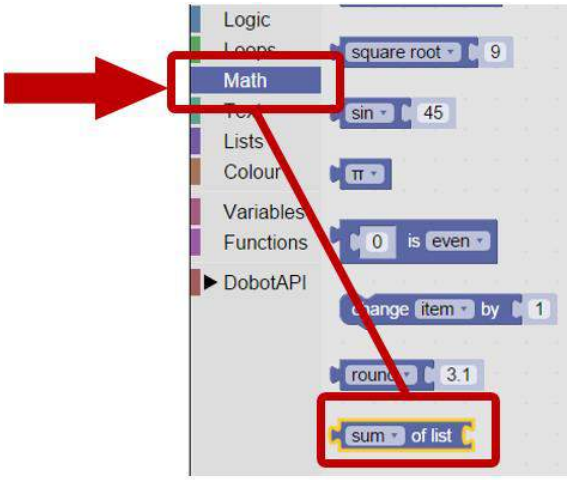
In order to put each variable's value into the list, we need to attach each variable to the list.



Attach the variable to all three empty links on the ***create list with*** block.

Drag over the ***Sum of List*** block. This block is a *RETURN DATA TYPE* command. It will allow us to look statistically at the data from the list.

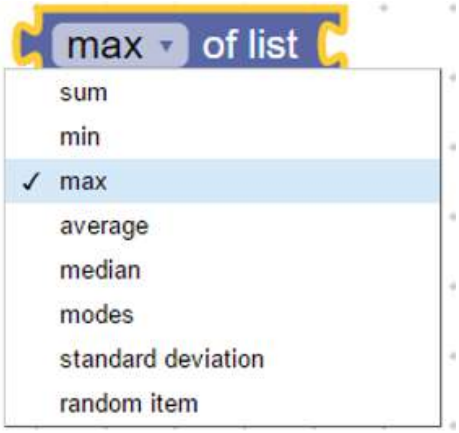
Using the blocks drop down, change *SUM* to *MAX*. Each time a color from the list is read, they will report a 1 to the *RETURN DATA TYPE* block (for this activity, the color variables will only return a 1 or 0). This will allow it to detect if any color is read and write either a 1 or 0 to our COLORFOUND variable.



Any Color Read, MAX value = 1

No Color Read, MAX value = 0

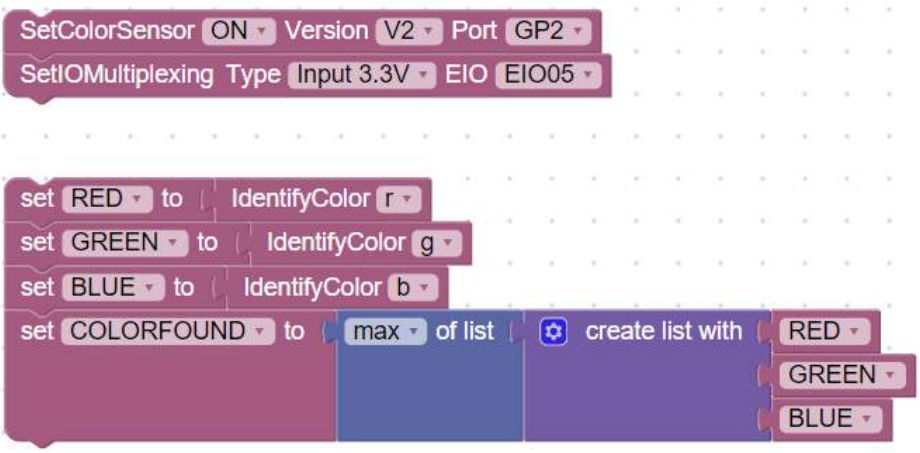
The last step is to attach the *LIST* and the *RETURN DATA* block to the variable COLORFOUND.



Connect all three elements together as seen below

Final Variable - Max Value from the List - Individual Color Variables

We can now link all of our variables together in one grouping as seen below



Now that the sensor is setup, it needs to be tested to make sure everything is reading correctly. The color sensor needs to report what color it is currently reading and is setup in Blockly to only read RED, BLUE, and GREEN.

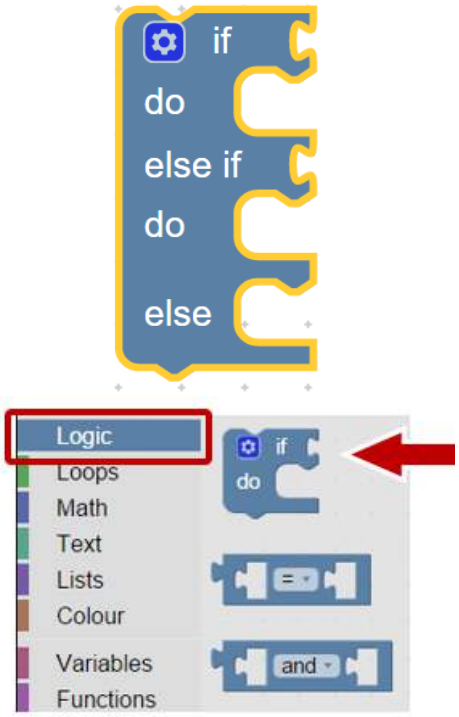
A loop needs to be created that will show in the Running Log the following information:

IF the color is RED

Or IF the color is GREEN

Or IF the color is BLUE

This can be done by creating an ***IF/ELSE IF/ELSE Loop***



There are two new commands that will be needed to check the cube's color and decide which location to take the cube to.

1. ***If/Else If/Else Statement***
2. ***IdentifyColor*** - Uses the current value from the color sensor

"If" statements are used when 2 or more conditions need to be evaluated. A three part "If" statement needs to be set up.

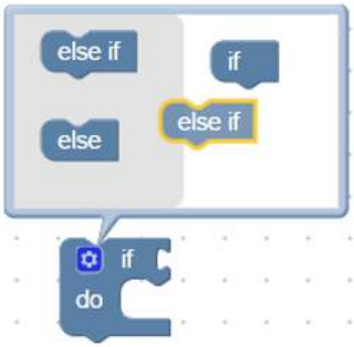
IF the value is Identified as RED, do this

ELSEIF the value is GREEN, do this

ELSE do this (Else will be read as anything other than the RED or GREEN cube which will include our BLUE cube.

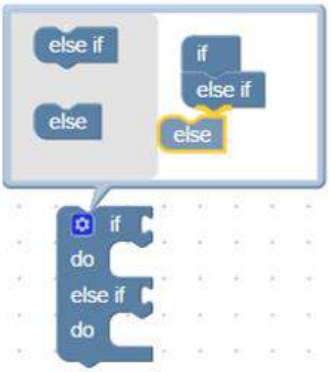
*The **If** structure used in this example will allow a NO CUBE or INVALID COLOR to be read as BLUE since BLUE is setup as the **else** condition. If your program is only reading BLUE, it may not be reading a color at all.*

In order to get the Logic If statement to expand to the three conditions we need, click on the gear next to the word if.



Drag the *Else If* over to the bottom pin connection of the *If* in the expansion window.

Do the same for the *else*.



This will build the three part IF statement that we need.

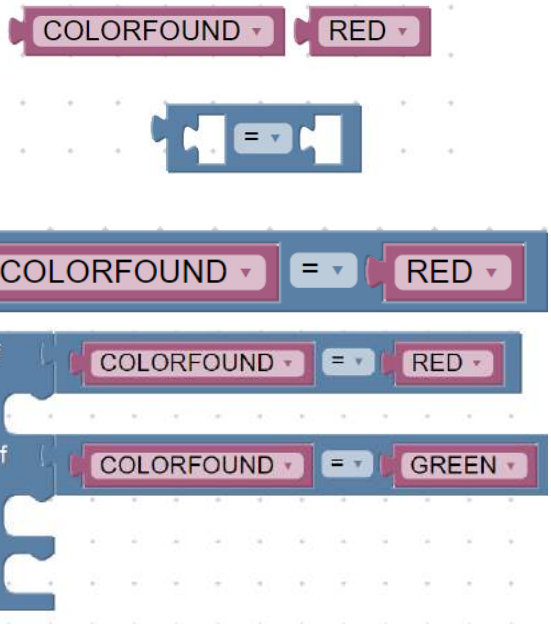
Conditions now must be made for when each level of the IF structure runs.

Drag over the ***Return True*** block from the logic section.

This block will allow us to compare the variable value *COLORFOUND* to each individual color variable.

When these two match (1 is equal to 1), that level of the IF structure will be ran.

Place the variable *COLORFOUND* and the variable *RED* inside the ***Return True*** block.



We now need to fill in our ***If/Else If/Else*** Statements.

If it is RED, stop evaluating and DO the statements included with the IF

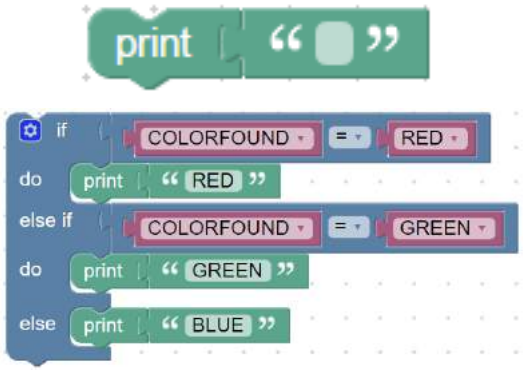
If it is not RED, it must be GREEN or BLUE

If it is GREEN, stop evaluating and DO the statements included with the ELSE IF

If it is not GREEN, it must be BLUE. Stop evaluating and DO the statements included with the ELSE

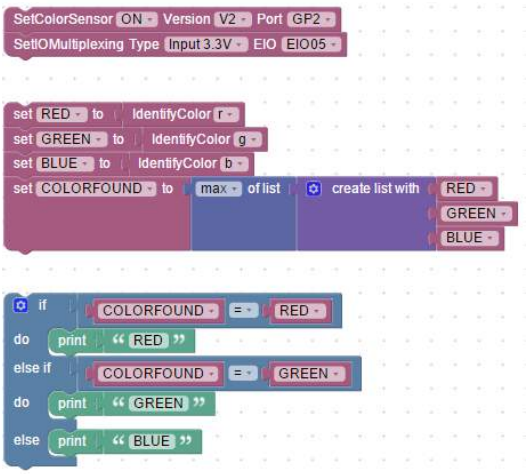
As our programs start getting more complex, it may help to start adding print commands to the code in order to make it easier to troubleshoot.

Add the ***Print*** block into each statement. These values will report to the Running Log so that you can see what is going on in the program.

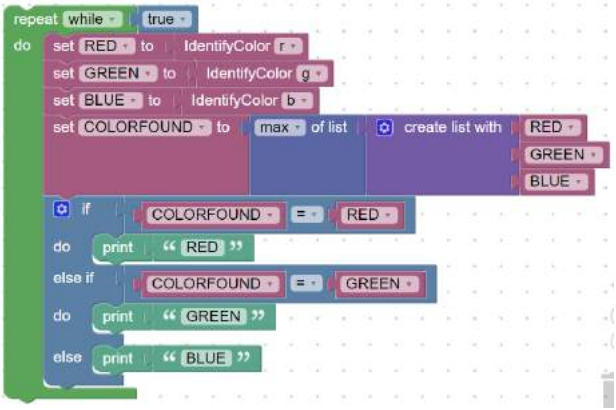


We now have three separate groups of code

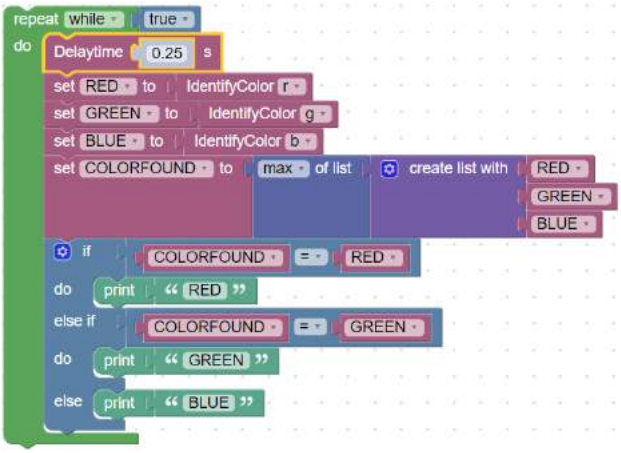
1. Setting Up Inputs and Outputs
2. Setting Up Variables
3. If Structure



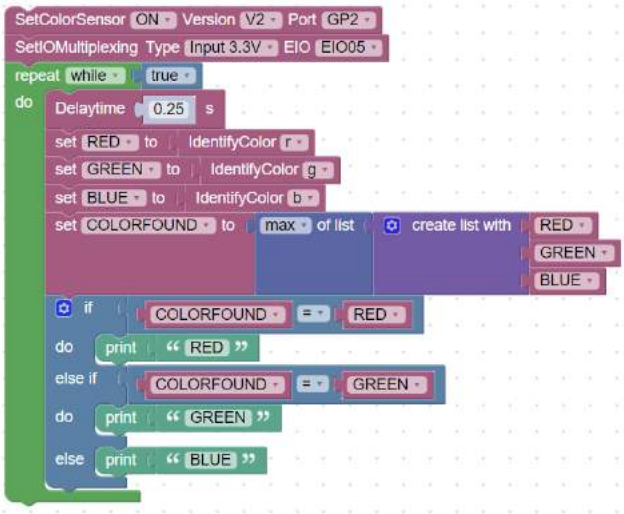
In order to get the If statements to constantly look for new values, both the *variables* and the If structure need to be put into a Forever Loop



We will also add a small delay *(from the DobotAPI/Basic section)* to slow down the looping process at the beginning of each loop.



Add together our header code and the color checking code so that it looks like the diagram here.



Once the program is completed, run it and see if it works correctly. Every time you put a colored block in front of the sensor, you should see the correct color reported in the running log. If it does not work, troubleshoot it until it does.

Points for discussion:

What does the sensor reads when there is no cube placed on the sensor?

What happens if you put the yellow cube on the sensor?

What type of reading do you get as you raise the cube up and away from the sensor?

Sometimes you will get a single missed, or incorrect reading as it changes between colors. Why?

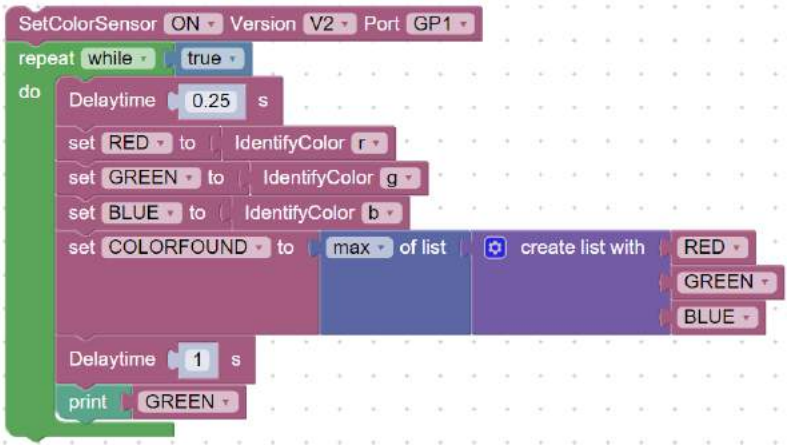
If your set up did not work correctly the first time, what did you have to do to make it work?

TROUBLESHOOTING

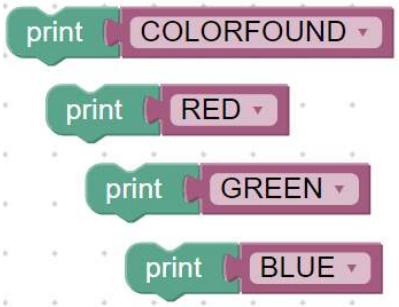
Checking Individual Variables

Develop a code similar to the one on the right.

The print can be replaced with each variable to isolate which ones are ready correctly, and which ones are not.



Each variable should read as 1 when they are true and 0 when they are false. The COLORFOUND will read as a 1 if it detects RED, GREEN, or BLUE (YELLOW reads as GREEN).



HELPFUL TIPS

The color sensor works best if the object is held at a consistent distance from the sensor Dobot suggested distance = (5mm to 10mm) or (¼ to ⅜)

HELPFUL TIPS

The different versions of the color sensors can come with different shades of colored cubes. The V1 sensors tend to come with a lighter green and blue than the V2 sensors. This can cause an issue with the sensors reading correctly. The V2 sensor tend to be more forgiving when it comes to shades of colors.

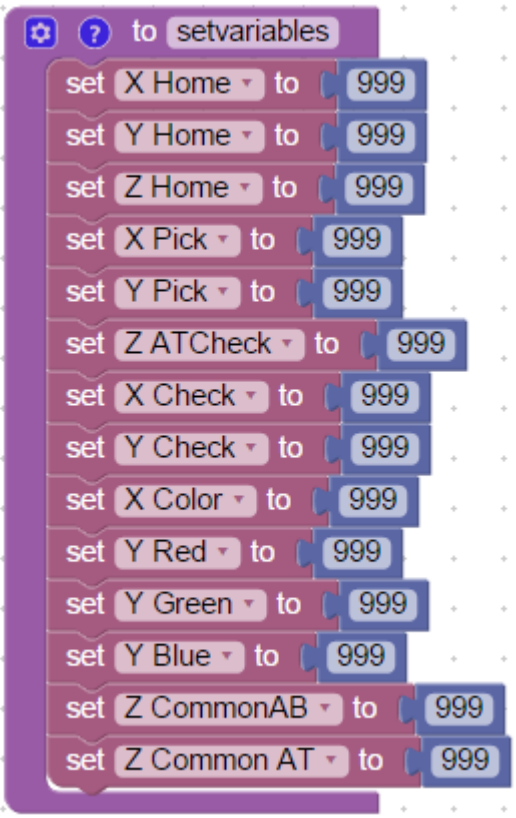
Remember that the color sensor is an analog sensor that reports a range of variable values as it detects colors to the program. Since we cannot see or adjust this number, you may need to play around with different colors and distances .

Now that we have the color sensor reading correctly, we can start developing the rest of the program.

In order to keep our program short and organized we will use several *functions/voids*.

Create the first *function* which will house all of our variables for the entire activity.

Call this function **setvariables.**



HELPFUL TIPS

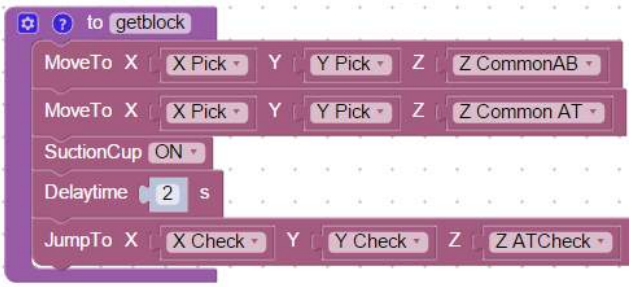
The Z values can be shared for both the pick and place positions.

The X Value can be shared for all of the place positions

The Lower Z Value for the Color Sensor needs to be about 10mm above the sensor face. This height allows the sensor to catch the reflection off the block to determine its color.

The second *Function* will go and get the block from a common location and JUMP to the Z ATCheck for the color sensor

Call this function **getblock**



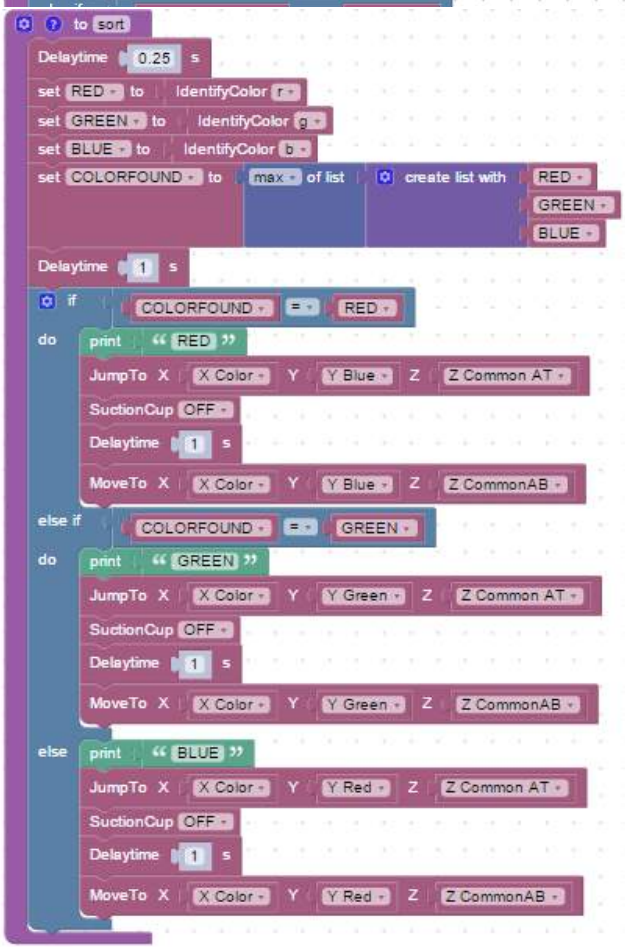
Add all the necessary JumpTo and MoveTo commands for each individual color.

The blocks are in a row. They should all share the same X value

Once the If Statements are complete, we will place it into its own function labeled **sort**.

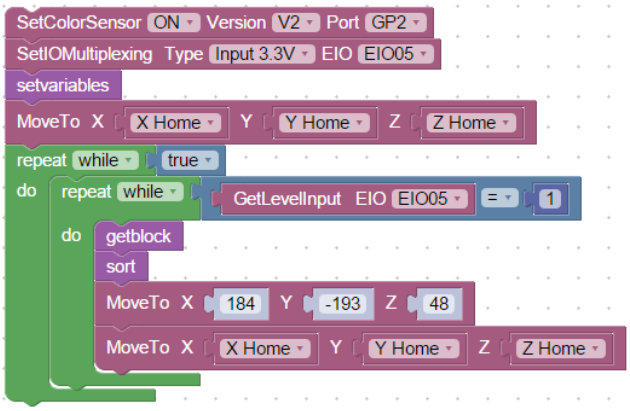
Each Statement will include:

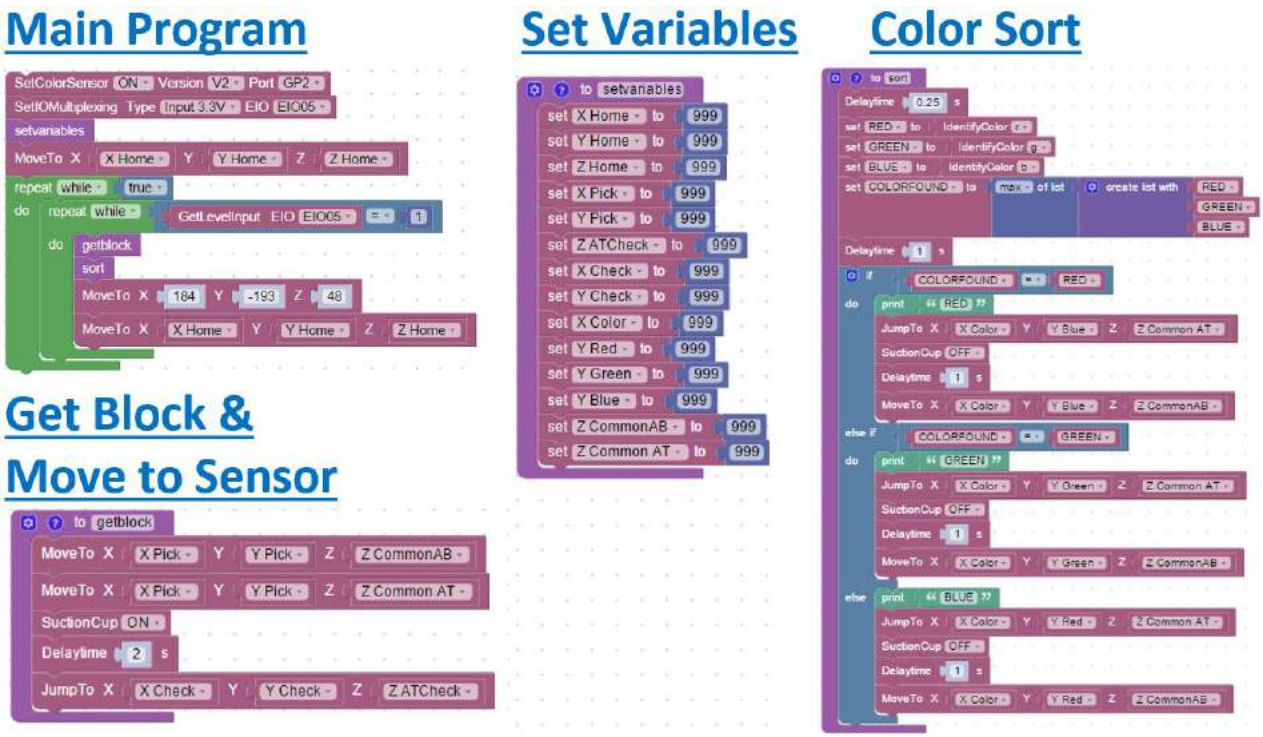
1. A short ***DelayTime*** before the IF Statement to ensure the cube is in place and the color has been evaluated correctly (Remember some of the error readings you may have received earlier during your testing)
2. *Jump* to correct color sort location
3. Turn the Vacuum off
4. Delay - 1 Second
5. Move up



Finish developing the main program by creating two loops

1. Forever Loop
2. Conditional Input Loop
    a. Get the block and move it to the sensor
    b. Identify the color and move it to the correct location
    c. Move to home





All of the separate sections of the program should look like this.

Be sure to consult the Dobot Input/Output Guide if you want to use other inputs and outputs, as damage to your robot or your other equipment may result.

Once the program is completed, run it and see if it works correctly. If it does not work, troubleshoot it until it does. Issues with the colors? See a simple color checking program below the "Going Beyond Section"

If your set up did not work correctly the first time, wait did you have to do to make it work?

CONCLUSION

1. *What happens if a block isn't there when the color sensor is told to get a color with the current program? Give a reason why.*

2. *How might you keep track of how many blocks there are of each color?*

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

_____ 1. Randomly arrange cubes in a 3x3 palletized matrix for the robot to pick from. After determining its color, drop the cube off at a specific color location. Manually remove the cube once it has been placed.

_____ 2. Have the robot stack the cubes in columns by color.

_____ 3. Color the 9 squares on the Dobot field diagram either red, blue, or green. You will then write a program that takes cubes from a cube matrix and puts the respective color cubes on the colored squares.

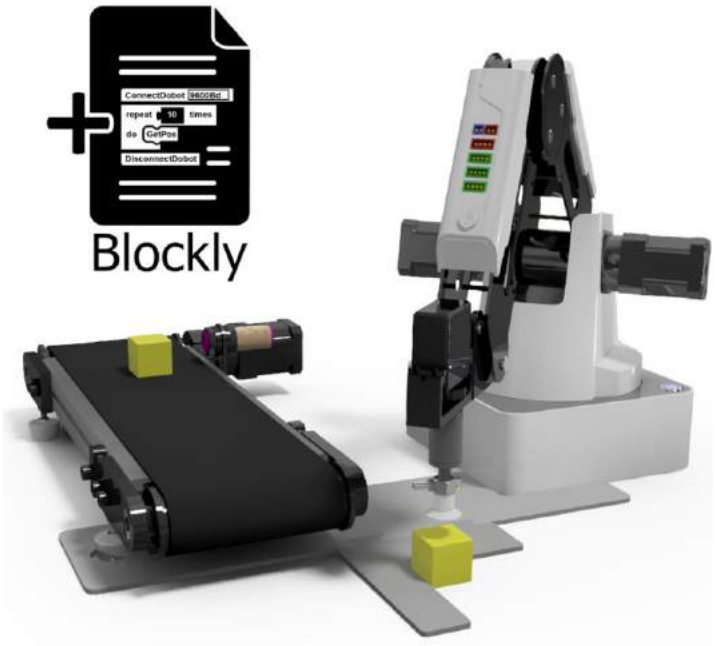
6 Blockly - Start & Stop Conveyor

NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Robotic arms need to communicate with each other as well as other peripherals such as conveyor belts or linear rails in order to move materials or products through stages of a work cell.

In this activity you will learn how to program a robot to control a conveyor belt. We will use an Infrared Sensor to create a closed loop system.



Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

KEY VOCABULARY

- Stepper Motor
- Conveyor Belt
- Output
- IR - Infrared Sensor

EQUIPMENT & SUPPLIES

- Robot Magician
- Dobot Field Diagram
- 1” cylinders or cubes
- Dobot Conveyor
- DobotStudio software
- Suction Cup Gripper
- *Dobot Input/Output Guide*
- IR Sensor

ESSENTIAL QUESTIONS

Essential questions answered in this activity include:

- Is the conveyor an input or an output to the robot?
- Is the Infrared sensor an input or an output?
- How do I wire the components together?
- What can I do with the infrared sensor?
- How do I code the conveyor in blockly?
- How do I code the infrared sensor in blockly?

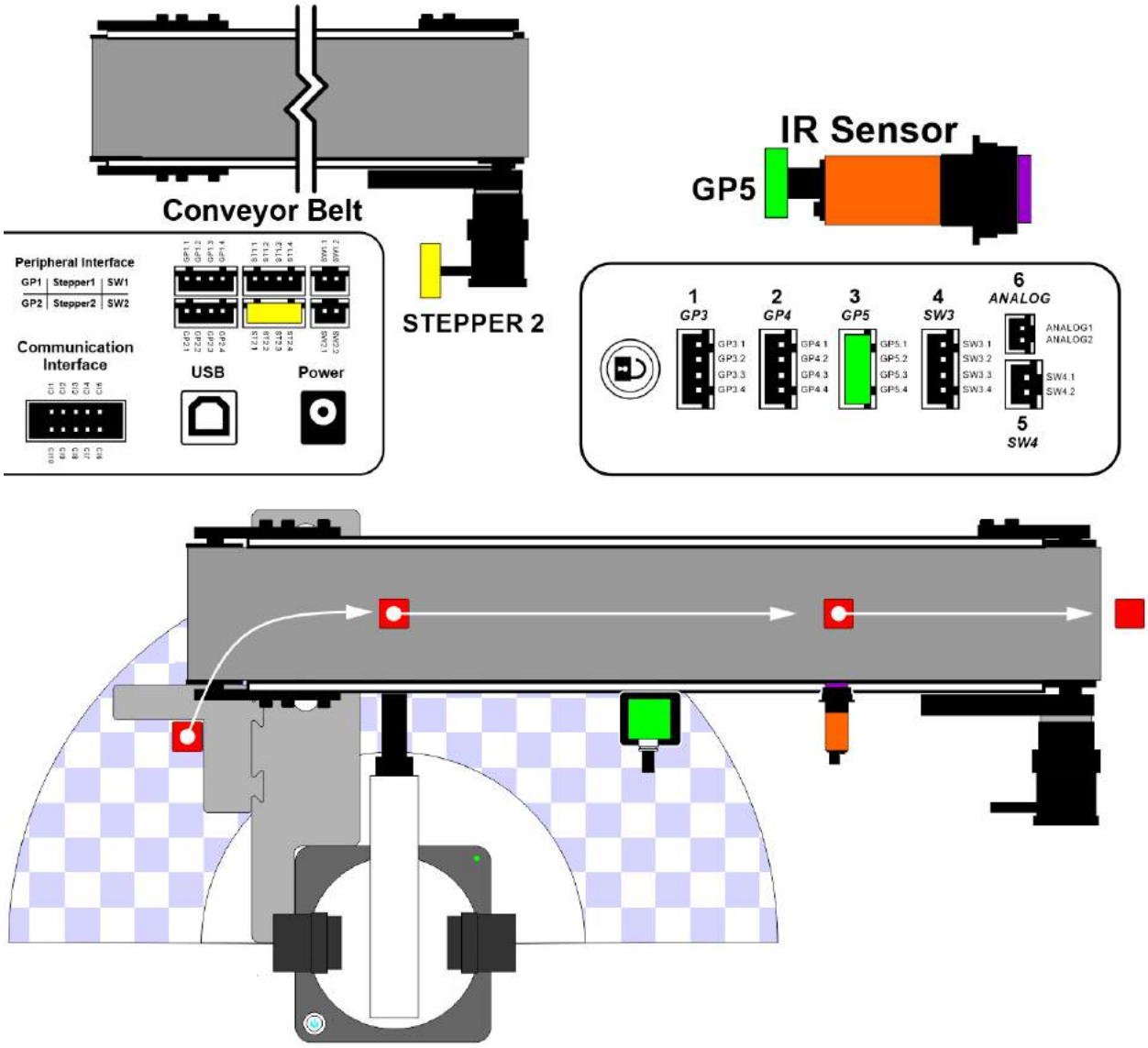
PROCEDURE

Order of operations

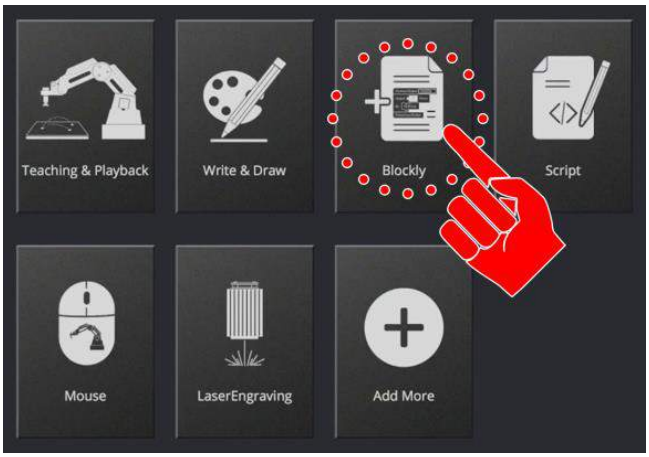
- The Robot will pick up a cube from a known location and place it on the conveyor belt.
- The Robot will return home and then start the conveyor belt.
- The conveyor belt will run until the block arrives at the IR Sensor for inspection.
- The block will be manually removed from the belt, inspected and then returned to the belt.
- Once the block is returned to the belt, the belt will run again until the parts runs off the belt and into storage.
- The process will loop forever.

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

1. Set up the robot with the suction cup gripper.
2. Plug the Conveyor Belt into **STEPPER2** of Robot and plug the IR sensor into **GP5** of the Robot


Conveyor Belt
IR Sensor
GP5
STEPPER 2
Peripheral Interface
GP1 | Stepper1 | SW1
GP2 | Stepper2 | SW2
Communication Interface
USB
Power
1 GP3
2 GP4
3 GP5
4 SW3
6 ANALOG
5 SW4
ANALOG1
ANALOG2

Open up Blockly in the software



HELPFUL TIPS

When you re-open this program check that the name of the file on top matches the code in the file, if it does not, you may end up overwriting another program

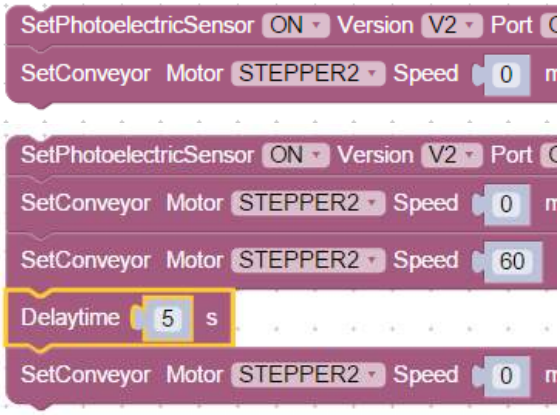
The first step will be to setup all of our INPUT and OUTPUT ports.

Drag over the ***SetPhotoelectricSensor*** block from the DobotAPI/Additional tool box

Set the sensor to ON / V2 / GP5

Drag over the ***SetConveyor*** from the DobotAPI/Additional tool box

Set the conveyor to Stepper 1 / 60 mm/s



Once the header code in established, we are going to write a quick code that will turn on the conveyor for 5 seconds

In this example, I have used 60 mm/s for the speed of the conveyor. Play with this value until you have a controllable speed for this activity.

Once the program is completed, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

If your set up did not work correctly the first time, what did you have to do to make it work?

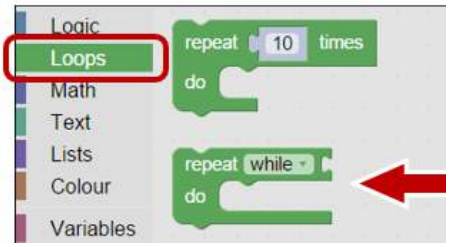
We will now edit our current code to get the conveyor belt to stop when an object is detected by the IR Sensor.

Remove the ***DelayTime*** from the program.

Create a loop that will keep checking the IR Sensor value until an object is detected

Bring over a ***RepeatWhile*** loop.

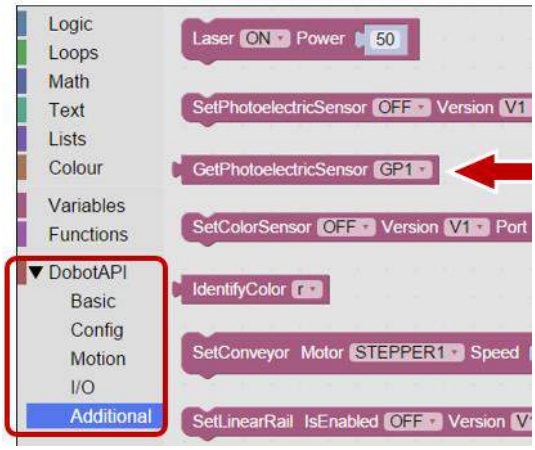
Change the loop into a ***RepeatUntil*** Loop



Create a *condition* that evaluates the IR sensors value. We are looking for the sensors value to change from 0 to 1.

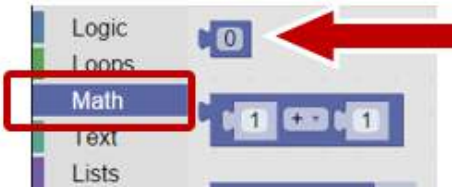
The IR Sensor reads true/high when an object is present. The small LED on the back of the IR Sensor will also light up.

Drag over the ***ReturnTrue*** condition from the Logic Toolbox.



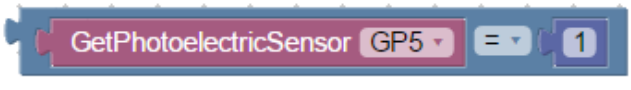
Drag over the ***GetPhotoelectricSensor*** from the DobotAPI/Additional Toolbox.

Drag over the ***Number Block*** from the Math Toolbox.



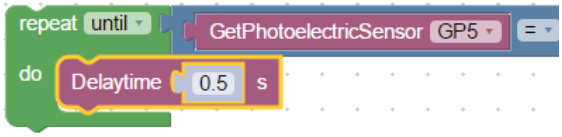
Add all three blocks together to complete the statement.

Make sure to change the ***GetPhotoelectricSensor*** value from GP1 to GP4 and the number value to 1



Add the *condition* statement to the ***RepeatUntil*** loop

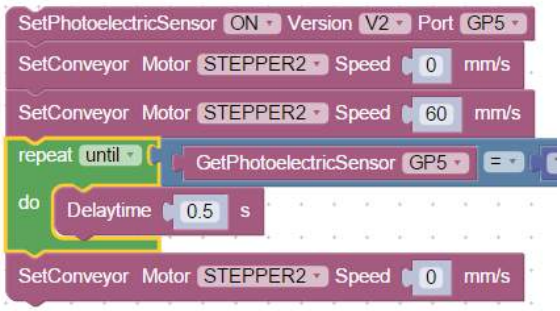
Add a small ***DelayTime*** into the ***RepeatUntil*** loop in order to give the operation something to process.



Put together the code that we have so far.

Run the program. Place a cube in front of the sensor. The conveyor should stop.

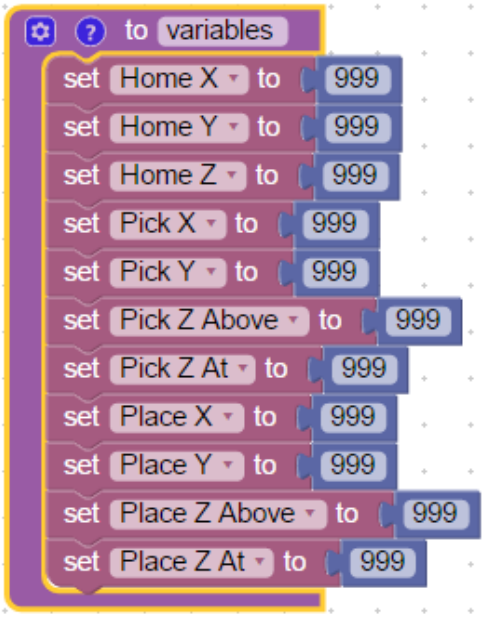
If it does not work, troubleshoot it until it does.



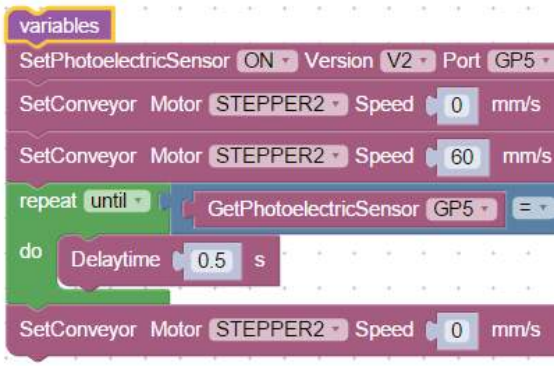
If your set up did not work correctly the first time, what did you have to do to make it work?

Now that we can start and stop the conveyor using the IR Sensor we can find our positions and set our variables for this activity.

Place the *variables* and positions in a function as has been done in previous activities.

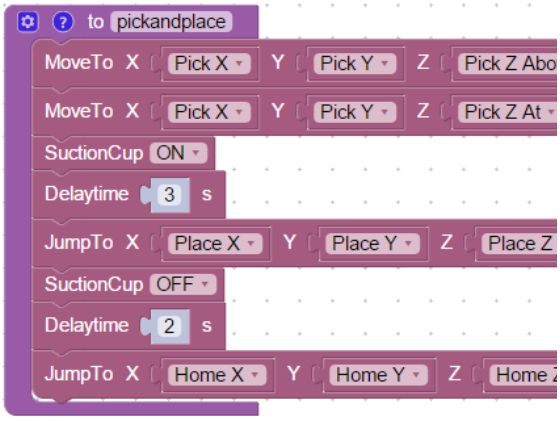


Add the *variables* ***Function*** to the beginning of your program

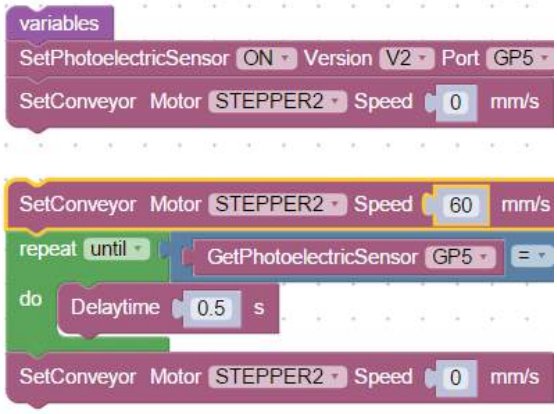


Create a ***Function*** for the Pick and Place operation.

- Pick up the cube
- Place it on the belt
- Return home

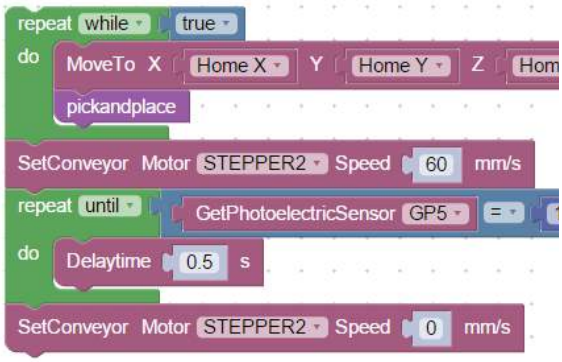


Pull the main program apart between the first two ***SetConveyor*** blocks.



Drag over and create a ***Forever*** Loop.

Start developing the loop

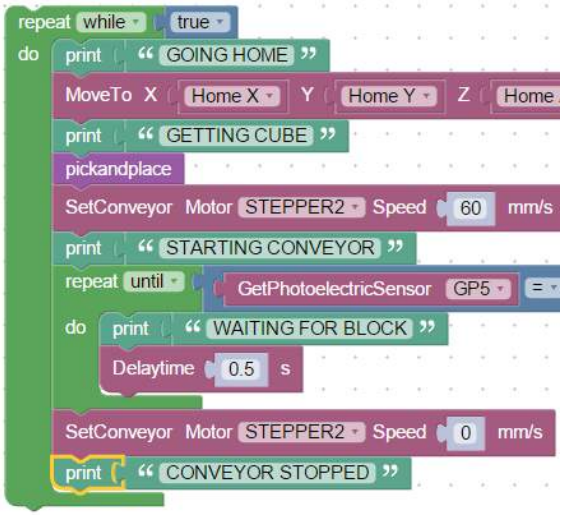


Consider the order of operations

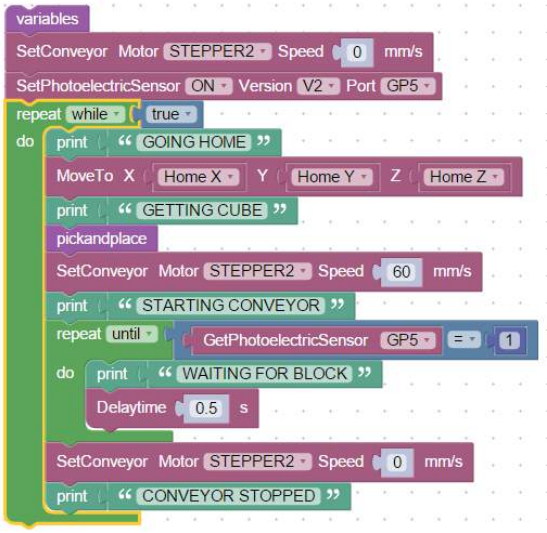
- Start with the **robot at a Home** position
- **Pick and Place** - Get the block and put it in the belt and return to home.
- **Start the belt** and run it **until the cube reaches the sensor**
- **Stop** the belt

As the program starts getting more complex, it may help to start adding print commands to the code to be able to troubleshoot the code

Add them anywhere in the program where an operation may be changing or a value may be read.



Assemble the code that we have so far.



Once the program is completed, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

If your set up did not work correctly the first time, what did you have to do to make it work?

Reminder:

This activity requires the following operations:

- The Robot will pick up a cube from a known location and place it on the conveyor belt.
- The Robot will return home and then start the conveyor belt.
- The conveyor belt will run until the block arrives at the IR Sensor for inspection.
- The block will be manually removed from the belt, inspected and then returned to the belt.
- Once the block is returned to the belt, the belt will run again until the parts runs off the belt and into storage.
- The process will loop forever.

The next step is to create a loop that will allow the block to be removed and then wait for it to be returned. The issue is that we have no idea when the block has been removed, how long it will take for the inspection, or when it will be returned.

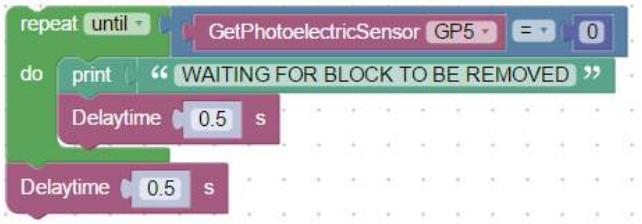
We need to create a closed loop system that will look for the following conditions without respect to time.

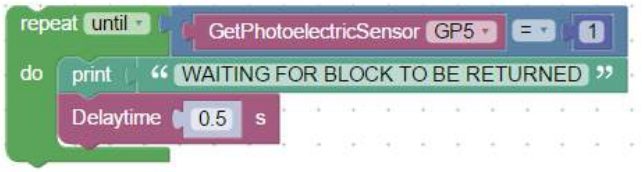
The Block has been REMOVED
The Block has been RETURNED

These two conditions can be developed using two separate ***RepeatUntil*** Loops just as we did for stopping the conveyor.

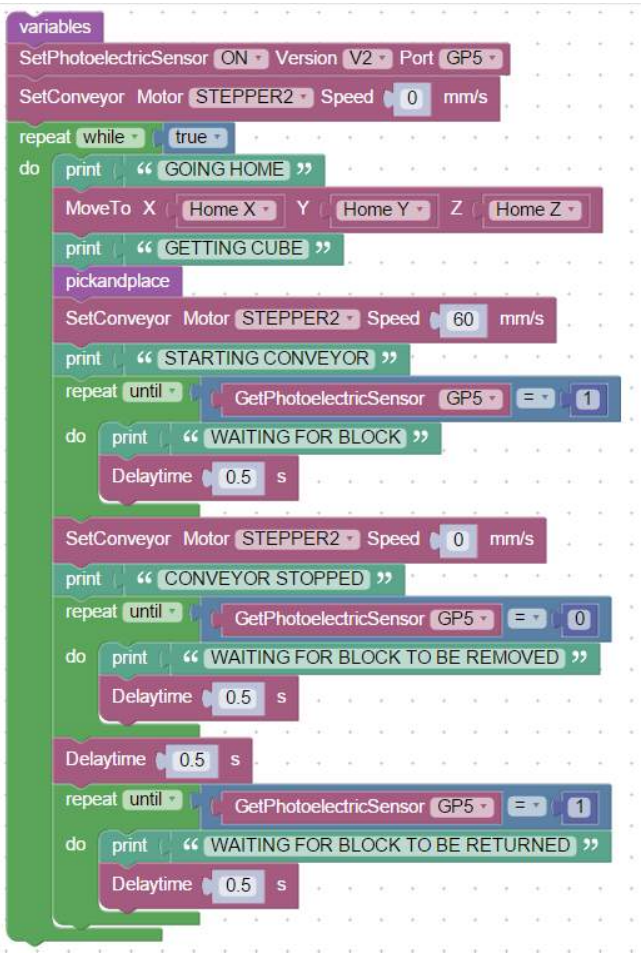
One that will wait for a '0' and one that will wait for a '1'

Place a small **DelayTime** in between the two loops to keep them separate and then add them into our main loop

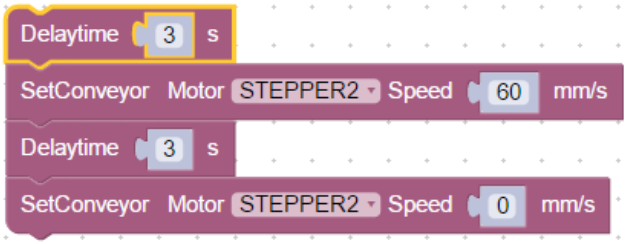




Some of this program can be condensed into separate functions in order to simplify the main program.

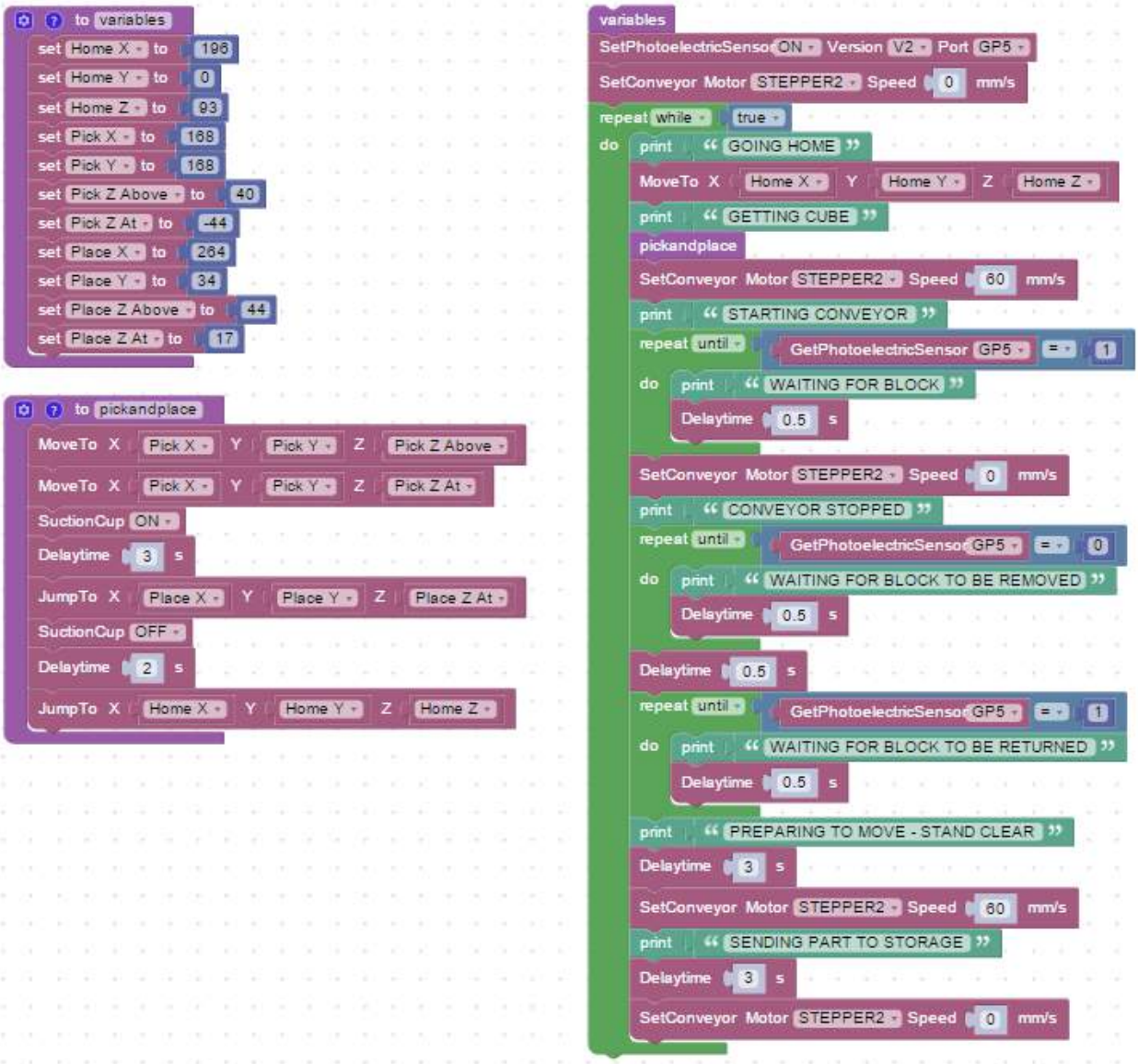


Our final task is to start the conveyor again and run the part off the end and into a container. Use the same code you started with to accomplish this task. The time needed to run the part off the conveyor will be different for each project, depending on where the sensor is mounted.



Place a small ***DelayTime*** in front of the code to allow time for the user to get their hand away from the belt before it starts running.

Add this last group inside the end of your Forever Loop.



Once the program is written, try it and make sure that it works as expected. If it does not work correctly, troubleshoot until it does.

If your set up did not work correctly the first time, what did you have to do to make it work?

CONCLUSION

1. *Why is it better to have the infrared sensor stopping the conveyor belt rather than just running it for time?*

2. *How would the program be different if the conveyor belt could not be run as a linear rail?*

3. *What's one way to determine where the robot is at any given time in the program?*

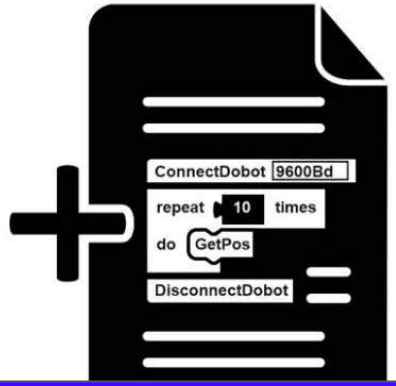
GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

_____ 1. Use functions to make your program as short as possible.

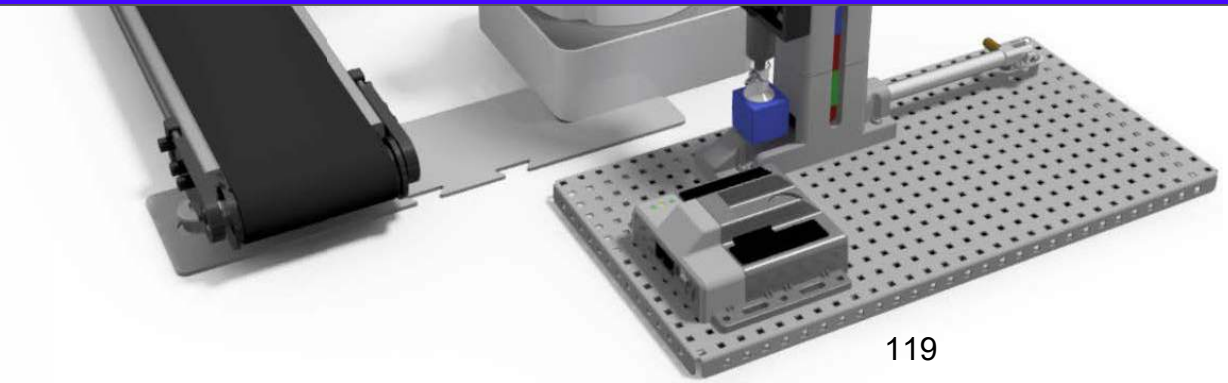
_____ 2. Use the color sensor and make the robot report what color block is being sent down the conveyor.

DOBOT



Blockly & Dobot

Hardware Connections



Connecting the Dobot Magician

There are many inputs and outputs that can be connected to the Dobot Magician

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

Types of Inputs

- Digital Switch
- Color sensor
- Infrared detector
- Another robot
- Microcontroller

Types of Outputs

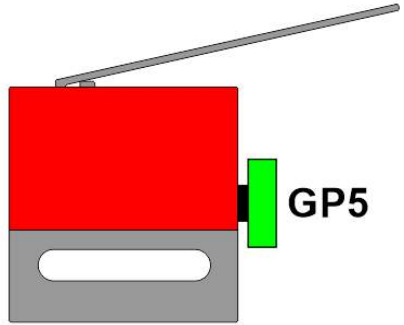
- Gripper
- Vacuum pump
- Conveyor belt motor
- Another robot
- Microcontroller

Inputs - Dobot Magician

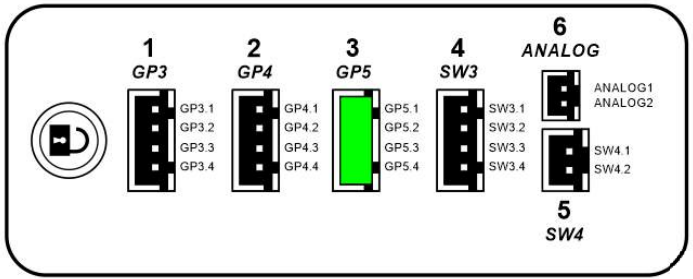
Digital Switch

A digital switch can be used to manually send an input to a robot.

***Example:** A worker inspects a part while a robot waits a safe distance away. When inspection is complete and the part is returned, the switch is hit by the worker telling the robot to come get the part.*



Limit Switch

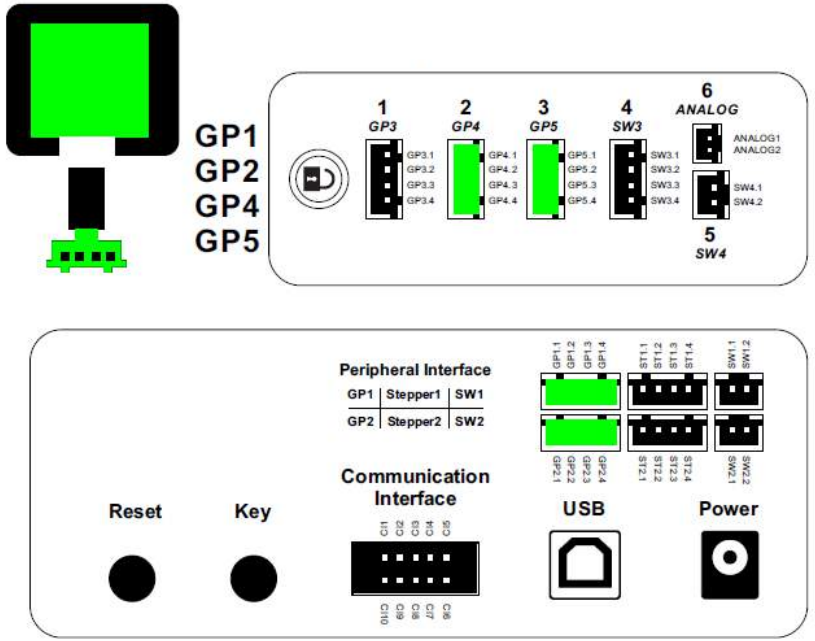


Inputs - Dobot Magician

Color Sensor

A color sensor can be used to distinguish the difference in color between red, blue, and green objects.

Example: *Make the robot sort products and place them in appropriate bins with other products of similar color.*

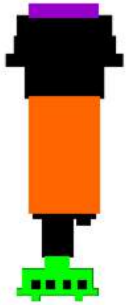


Inputs - Dobot Magician

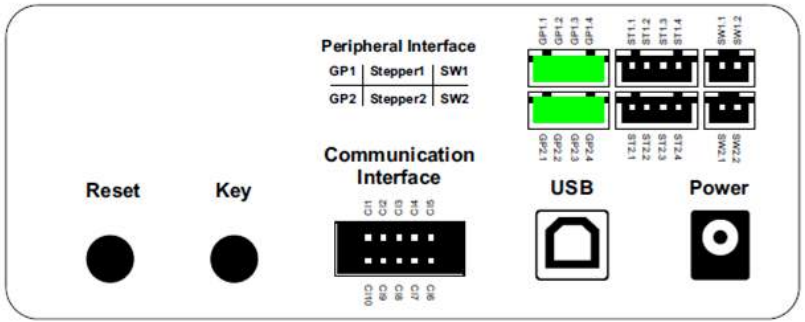
Infrared (IR) Sensor

An infrared sensor can be used to detect whether an object is in place in a work cell.

***Example:** Make the robot wait until a part is put in place before moving. Place this sensor on a conveyor, and when a part is seen by this sensor, the robot can start and stop the belt.*



GP1
GP2
GP4
GP5

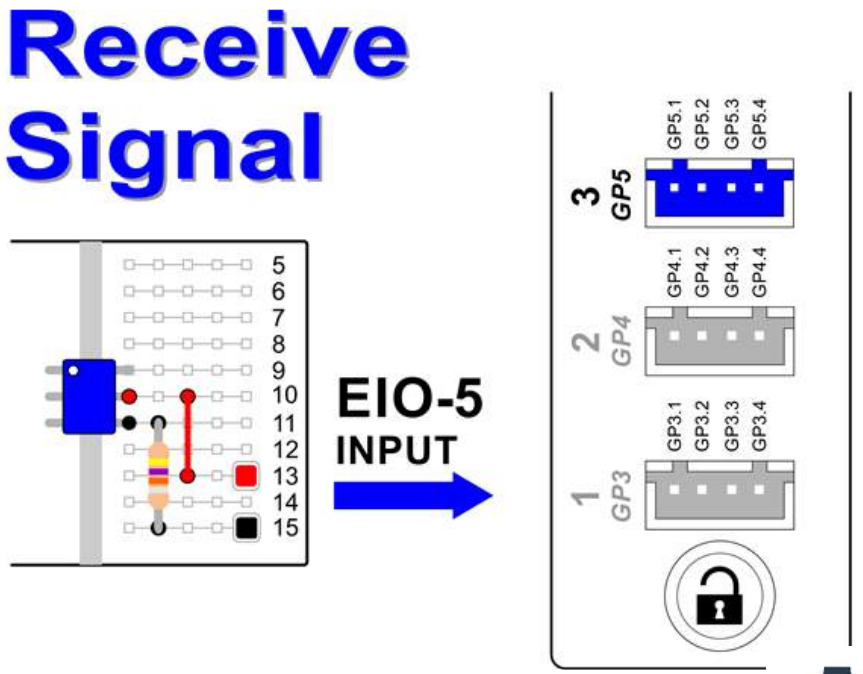


Inputs - Dobot Magician

Another Robot

Your robot can wait for a signal from another robot. A handshake module is used to isolate different voltages.

Example: *Wait until a robot has completed an operation before this one starts.*

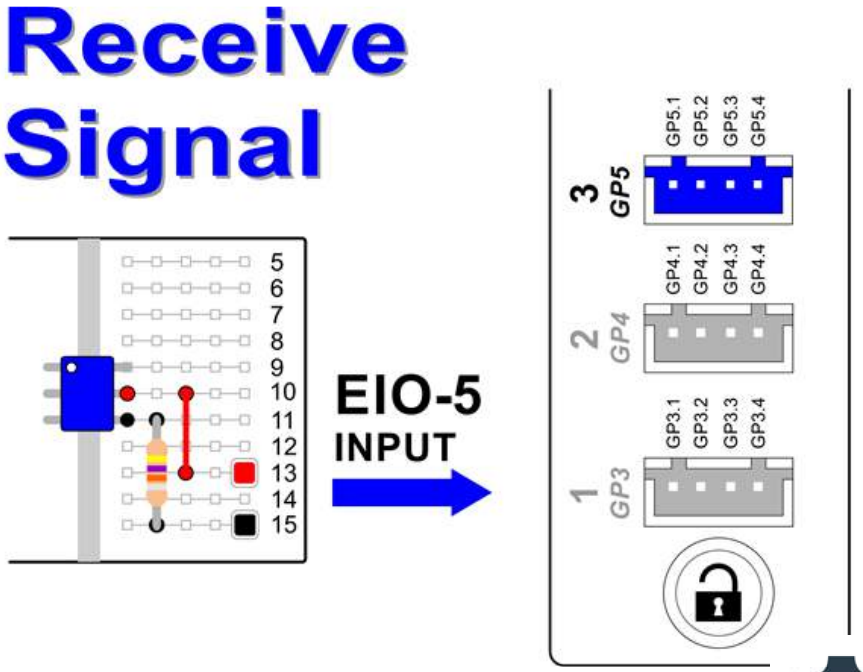


Inputs - Dobot Magician

Microcontroller

A robot can wait for a signal from another machine or a microcontroller. A handshake module is used to isolate the different voltages.

***Example:** Make the robot wait until a part is finished in a CNC machine.*

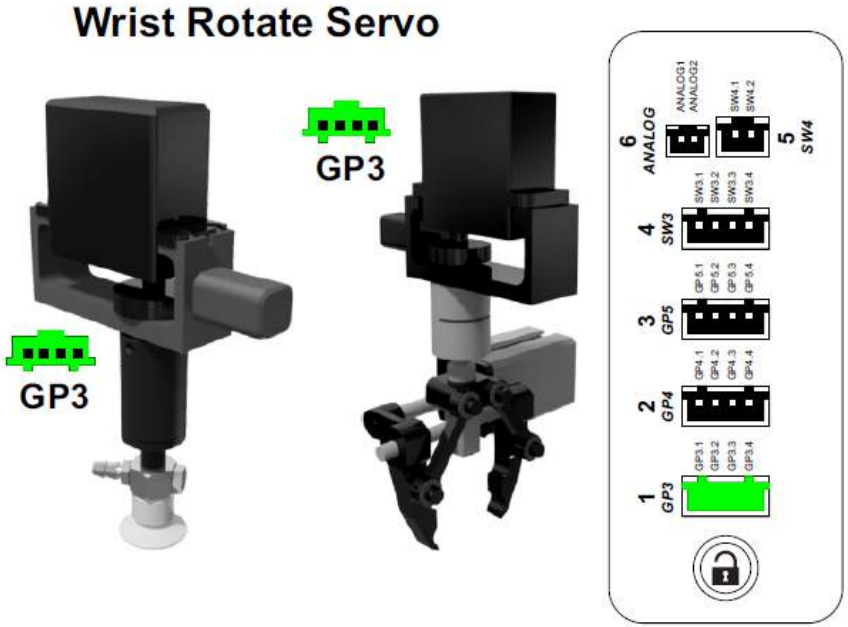


Outputs - Dobot Magician

Wrist Rotate Servo

A robot can send a signal to End of Arm Tooling (EoAT) to rotate a gripper, in turn rotating the part.

Example: A palletizing operation requires you to place nine rectangular boxes on a pallet. This servo allows you to orient the boxes correctly on the pallet.

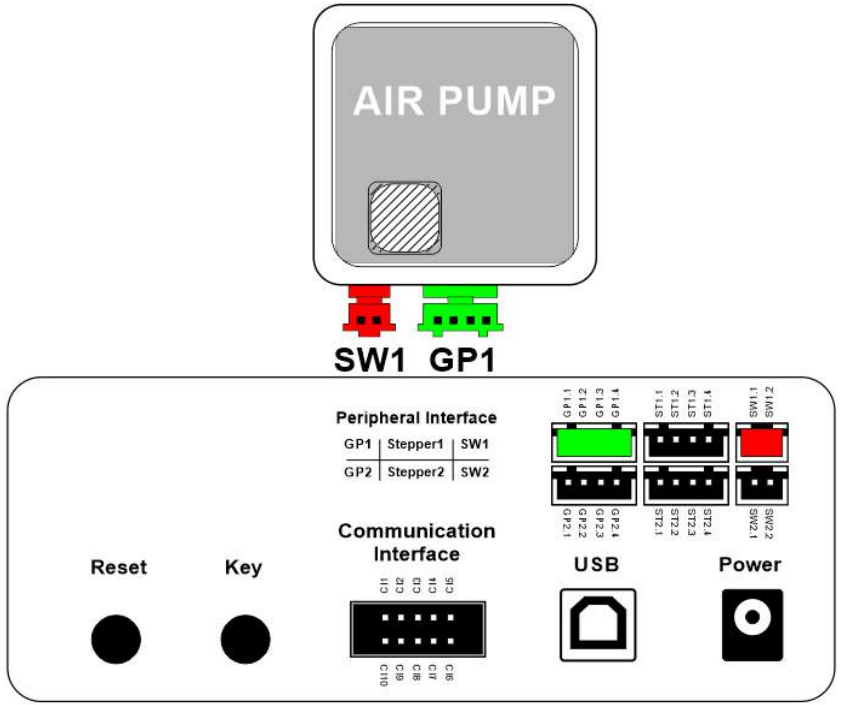


Outputs - Dobot Magician

Vacuum Pump

A robot can turn pneumatic devices on and off. Many grippers and EoAT use pneumatics for activation

***Example:** This can be used as part of pick and operation to activate the vacuum gripper on or off.*

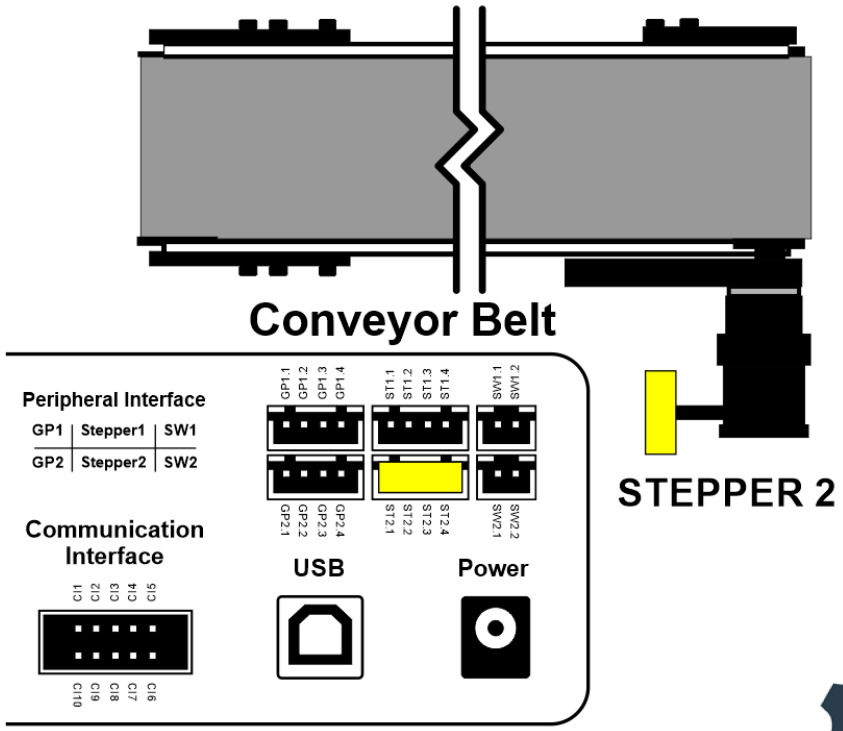


Outputs - Dobot Magician

Conveyor System

A robot can turn a conveyor system on and off.

***Example:** Have your robot place a part on the conveyor, then turn the belt on to move the part to another operation within a workcell.*

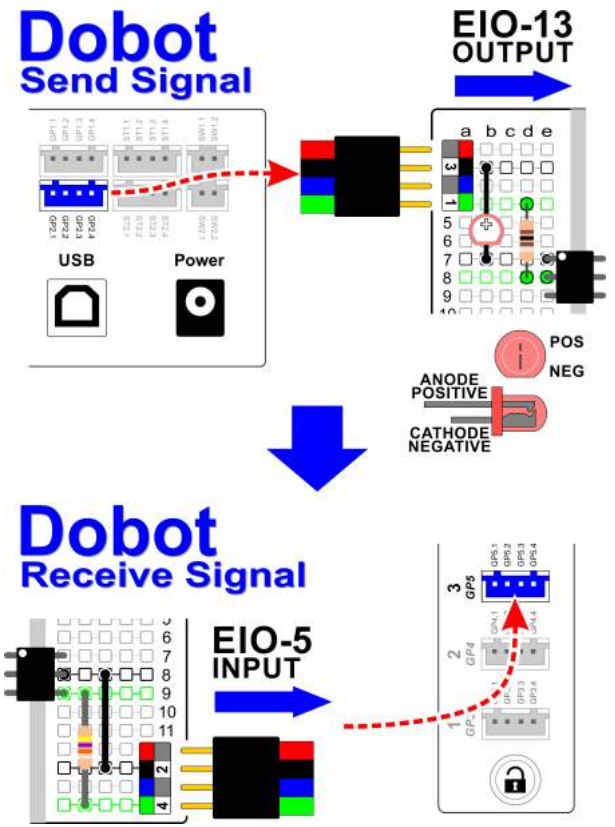


Outputs - Dobot Magician

Another Robot

A robot can send a signal to another robot. A handshake module is used to isolate different voltages.

***Example:** Complete an operation with your robot and send a signal to another one to come get it and place it on a pallet.*

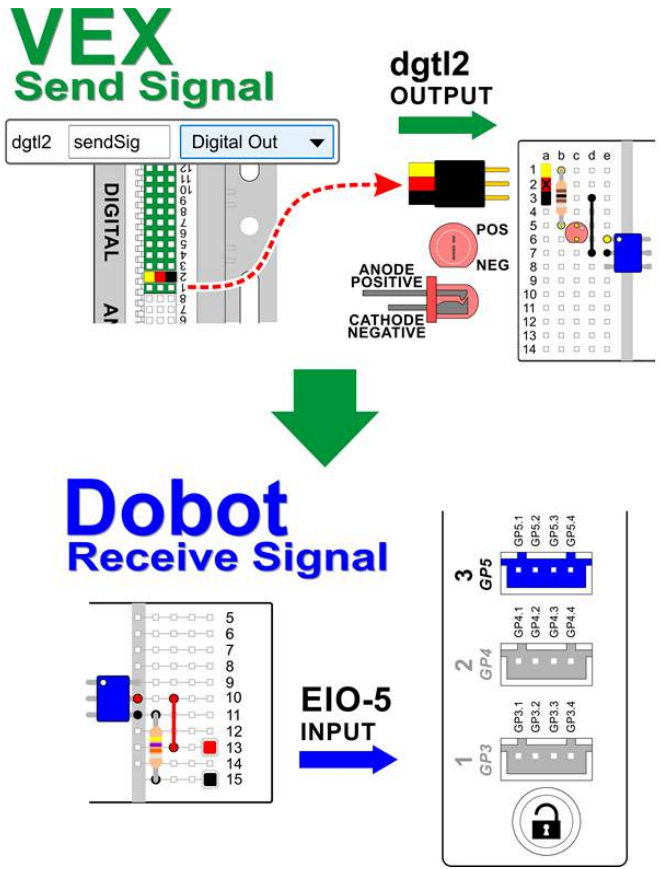


Outputs - Dobot Magician

Microcontroller

A robot can wait for a signal from another machine or a microcontroller.A handshake module is used to isolate different voltages.

***Example:** Make the robot wait until a part is finished in a CNC machine.*



Resources

All photos, graphics, images & icons included in this presentation are the intellectual property of ChrisandJimCIM.com.

7 Blockly - Dobot to Dobot Handshaking

NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Robotic arms need to communicate with other robots in a work cell, or factory. This is called *HANDSHAKING* and can be done between different machines, devices and robots. It is a very simple form of communication and is done with simple ones and zeros; or "ons" and "offs".

In this activity you will learn how to make a robot handshake with another robot.

Robot 1 will pass a part into another robot's work envelope, go to a safe position and then send a signal to other robot. The signal received by the other robot will initiate a sequence to get the part, and place it somewhere else.

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

KEY VOCABULARY

- Input
- Output
- Function / Voids
- Handshaking

EQUIPMENT & SUPPLIES

- 2 Robot Magicians
- 1" cylinders or cubes
- Suction Cup Gripper
- DobotStudio software
- *Dobot Input/Output Guide*
- Handshake Modules

ESSENTIAL QUESTIONS

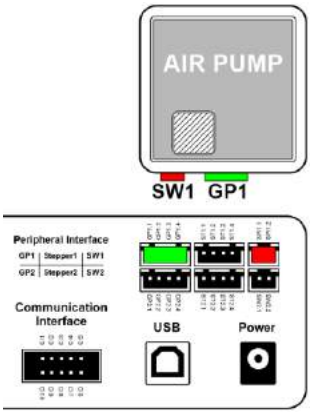
Essential questions answered in this activity include:

- How do I make a robot send a signal?
- How do I get a robot to receive a signal?
- How is this done in Dobot Studio Software?
- How do I make two robots talk to one another?

PROCEDURE

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

1. Set up both ***Robots*** with a suction cups - ***GP1 & SW1*** and place Dobot field diagrams, taped to the work surface, between the two robots.
2. Wire ***Robot1*** with an ***OUTPUT*** signal ***GP2 - EIO13***.
3. Wire ***Robot2*** with an ***INPUT*** signal ***GP5 - EIO5.***
4. Wire both Robots to the Handshaking module as shown in the following pages. Be sure that wires are not going to be pulled out by the motion of the robots.



Open Loop System Block Diagram: Robot #1 acts as the input for robot #2. Robot #2 does not communicate back to robot #1 so there is no feedback. The Handshake Module acts as the device that helps make the handshake happen safely.

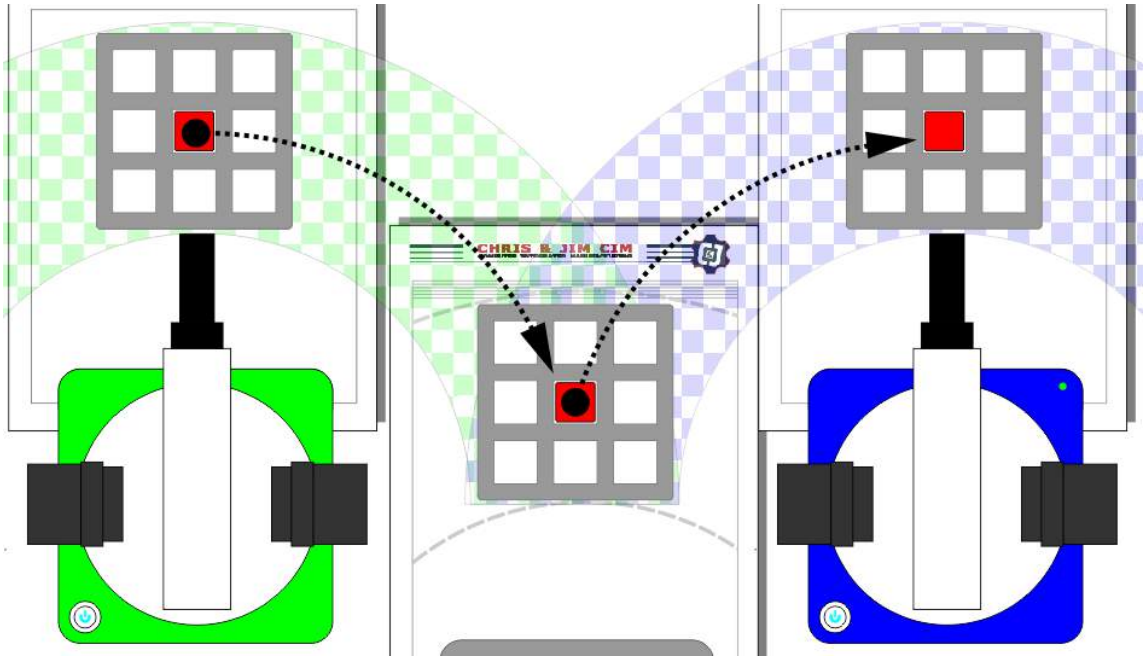
Order of operations

ROBOT 1 - OUTPUT SIGNAL

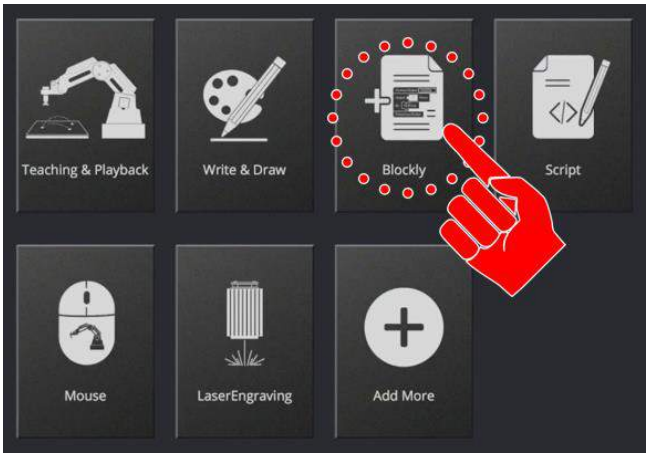
- Move - Home
- Move - ABPick
- Move - ATPick
- Jump - ATPlace
- Move - ABPlace
- Move - Home
- Send Output Signal

ROBOT 2 - INPUT SIGNAL

- Move - Home
- Wait for INPUT Signal
- Move - ABPick
- Move - ATPick
- Jump - ATPlace
- Move - ABPlace
- Move - Home



Open up Blockly in the software

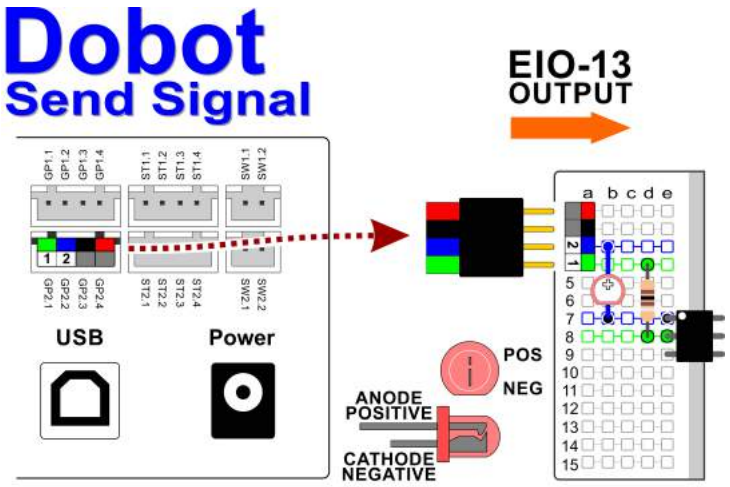


HELPFUL TIPS

When you re-open this program check that the name of the file on top matches the code in the file, if it does not, you may end up overwriting another program

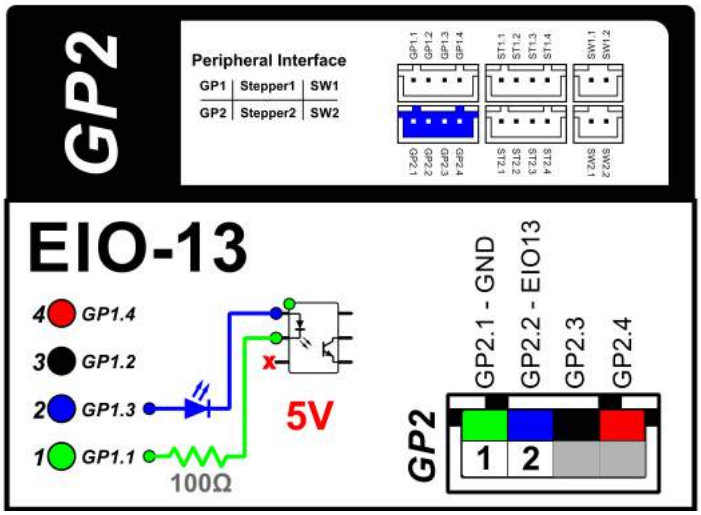
How to SEND an OUTPUT signal.

The first part of this activity will focus on how to send a signal.

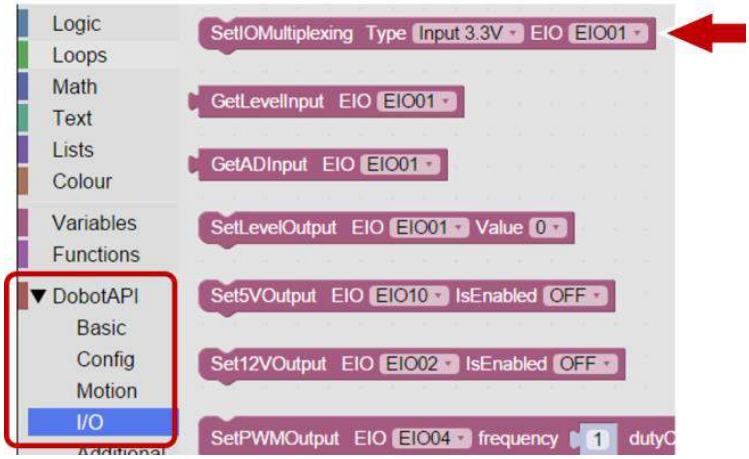


In order to make sure our setup, wiring, and program are all correct, we will only use **ONE ROBOT**.

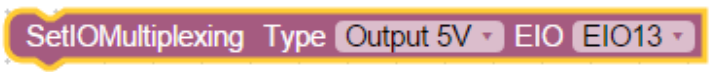
Nothing should be connected to the Output side of the handshake module.



Drag over the ***SetIOMultiplexing*** block from the DobotAPI/ I/O Toolbox

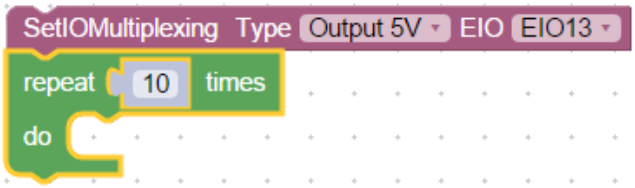


Change the settings to **OUTPUT 5V** and **EIO13** In order to test the OUTPUT signal, we will create a simple program that loops the process of turning on and off the output.



Drag over the ***RepeatTimes*** Loop

10 times should be enough to ensure our setup is working correctly so make it repeat this many times.

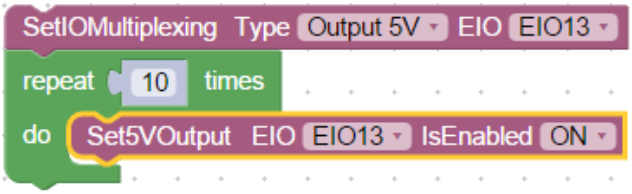


Drag over the ***Set5VOutput*** block from the DobotAPI/ I/O Toolbox.

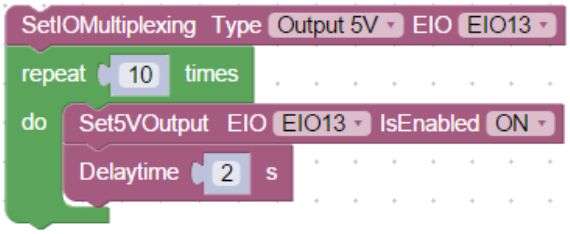
Add the block to our loop

Set the port to **EIO13**

Set Enabled to **ON**

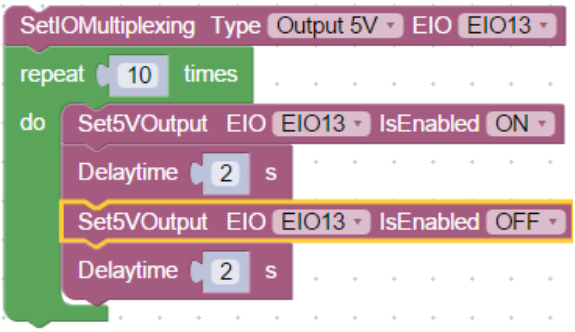


Insert a ***TimeDelay*** under the ***Set5VOutput*** block. Set it to send the signal for two seconds.

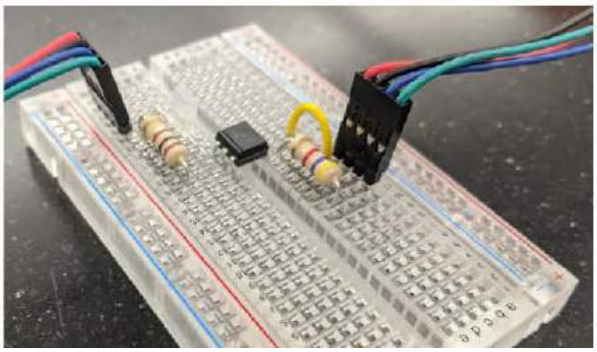


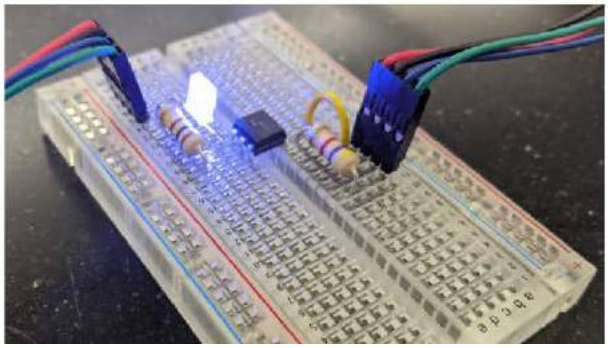
Duplicate both steps

Set Enabled to **OFF**



Once this portion of the program is completed, run it and see if it works correctly. The LED on the handshake module should light up whenever EIO13 is enabled ON. If it does not work, troubleshoot it until it does.





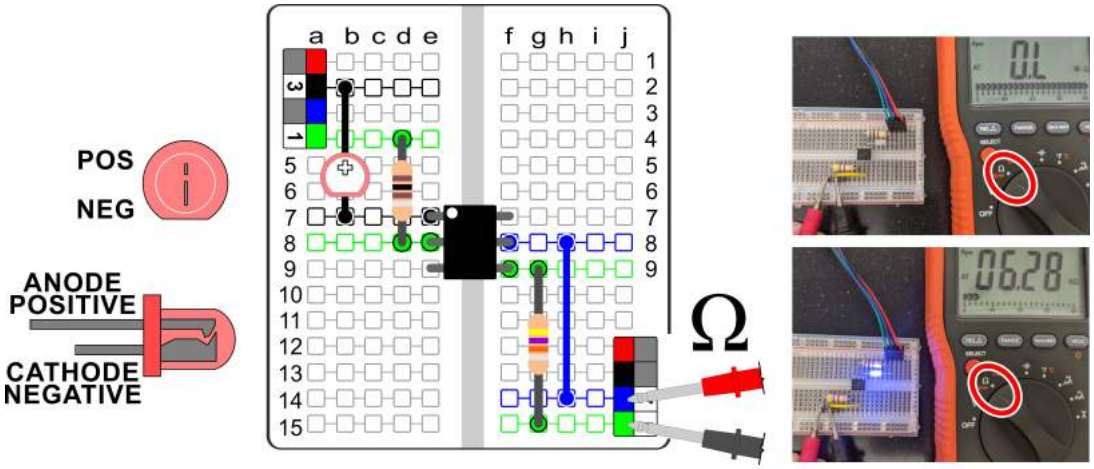
If your set up did not work correctly the first time, what did you have to do to make it work?

TROUBLESHOOTING THE OPTICAL ISOLATOR

HELPFUL TIPS

The LED helps us determine if the signal is actually being sent by the first robot. We can write a simple Blockly program to see if the second robot is getting the signal. The issue is if the second robot is not getting the signal….. WHY is it not getting the signal? It could be a damaged optical isolator.

An easy way to check is to use a voltage meter set to check RESISTANCE. When no signal is present, the voltmeter should read zero. When a signal is present it should read a little over the resistance value of the resistor used.



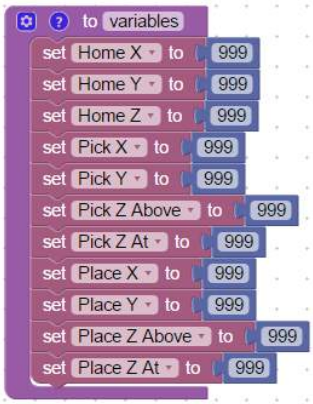
Optical Isolator OFF = 0.L or ---
Optical Isolator ON = 4.9+ kΩ

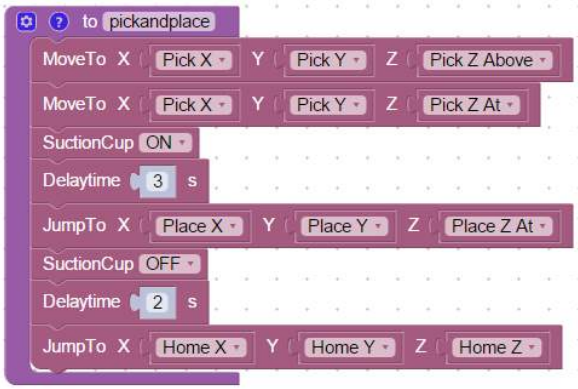
Programming for ROBOT 1

Throughout the remainder of this activity, we will use many of the skills that you have developed in previous activities. If you need clarification at any point, you can go back to the other activities for more information.

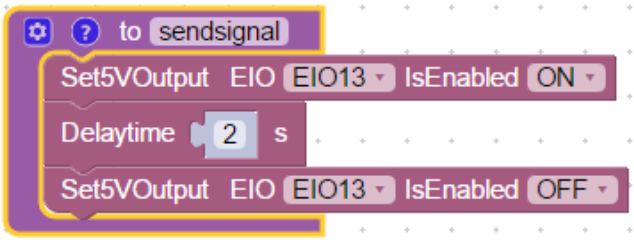
As we have done in the past,

- Create *Functions*
- Create Variables
- Find *Positions*

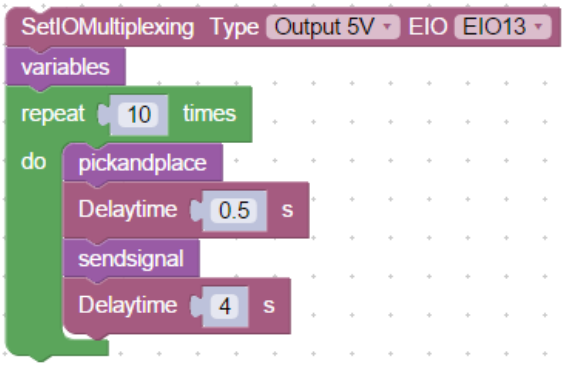




Create a new *Function* - SendSignal



Put the program together!



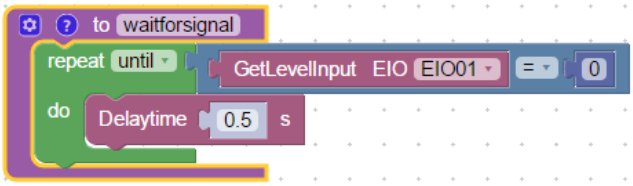
Programming for ROBOT 2

Throughout the remainder of this activity, we will make assumptions that you have already learned several concepts in previous activities

As we have done in the past,

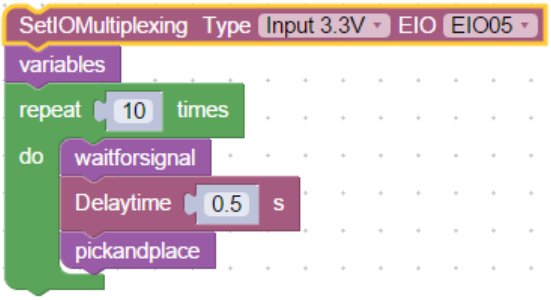
- Create *Functions*
- Create *Variables*
- Find *Positions*

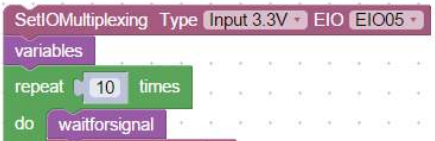
Create a ***RepeatUntil*** Loop to wait for the signal

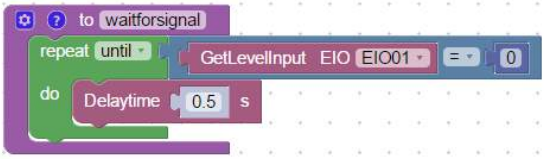


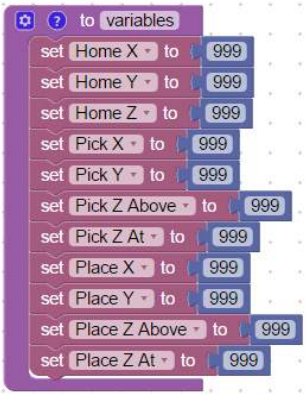
Reminder!! Your repeat until will need to wait for a Zero/Low signal for the Magician V2 and a One/High signal for the Magician V1

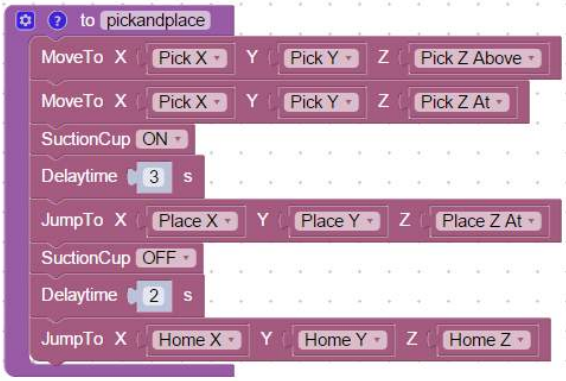
Put the different parts of the program together.



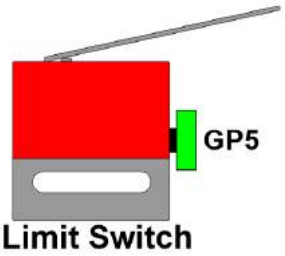




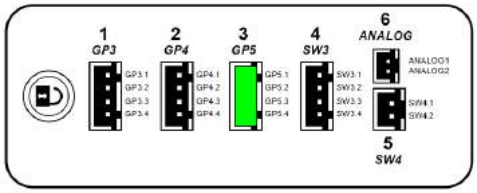




USE A LIMIT SWITCH HARDWIRED TO THE ARM FOR TESTING THIS PROGRAM



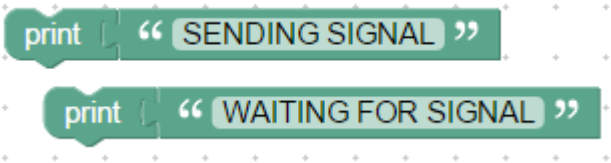
Once this portion of the program is completed, run it and see if it works correctly. If it does not work, troubleshoot it until it does.



When you press the switch, what happens?

If your set up did not work correctly the first time, what did you have to do to make it work?

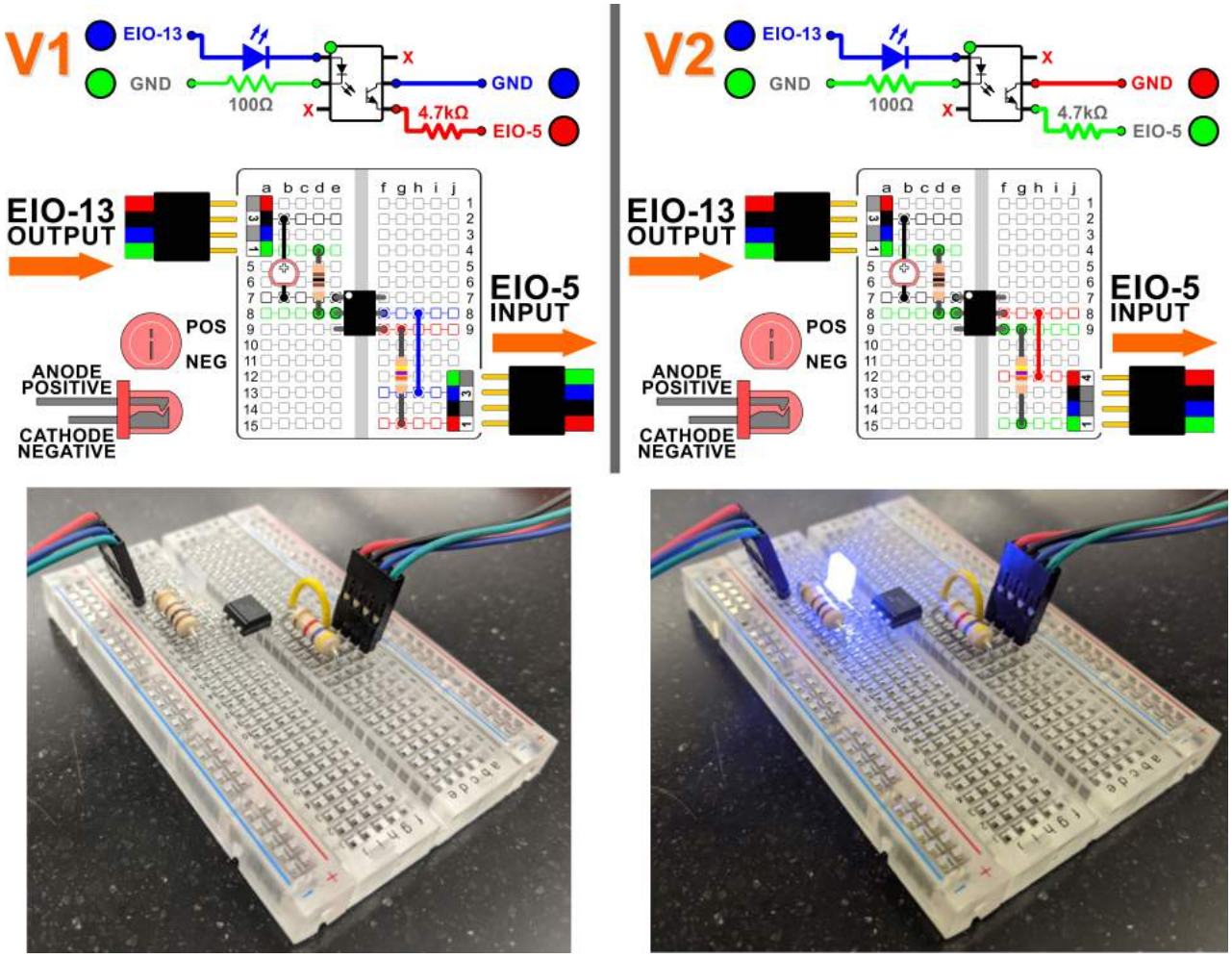
Consider adding Print to Running Log commands to both programs so that you can see what the robot is doing at any given time.



Now that both programs are created **AND** tested, it is time to put it all together.

Finish the Handshake Module and replace the manual limit switch on ***ROBOT 2*** with the breadboard connection.

Dobot Magician V1 (All White IO Ports)	*Dobot Magician V2 (Colored IO Ports)*



Be sure to consult the Dobot Input/Output Guide if you want to use other inputs and outputs, as damage to your robot or your other equipment may result.

Once the program is completed, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

If your set up did not work correctly the first time, what did you have to do to make it work?

CONCLUSION

1. *What would you have to do to make this program run five times without any human intervention? Explain fully below.*

2. *What other inputs could you use on your robot to start this process? Use the Dobot Input/Output Guide to answer this question, and do not attempt to try it without your instructor's permission.*

3. *What other outputs could you use on your robot to start this process? Use the Dobot Input/Output Guide to answer this question, and do not attempt to try it without your instructor's permission.*

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

_____ 1. Change the LEDs to Motors. Be sure to get your instructor's permission, and be sure to use the correct outputs for the motor chosen.

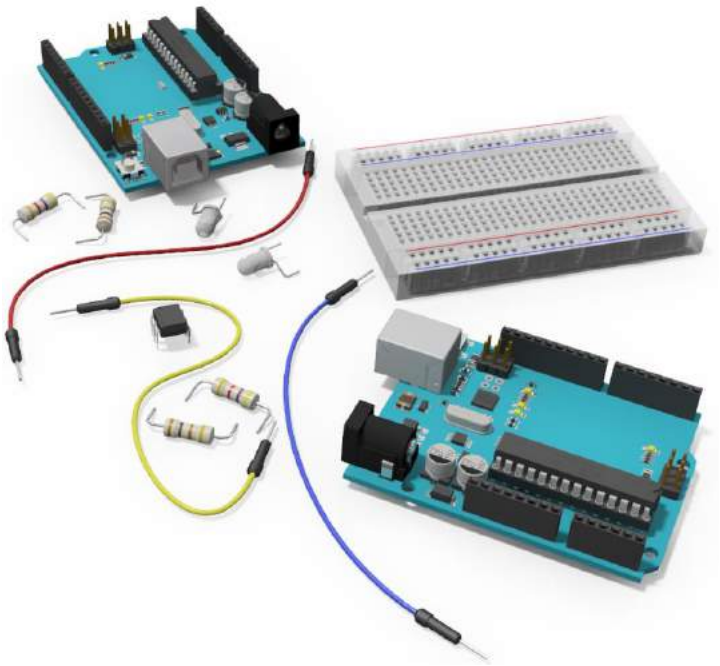
Be sure to consult the Dobot Input/Output Guide if you want to use other inputs and outputs, as damage to your robot or your other equipment may result.

8 Blockly - Handshaking - Arduino to Arduino

NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Often the process of handshaking goes beyond a robot's need to communicate with another robot. In industry the process of have Programmable Logic Controllers (PLCs) communicate with each other, or robotic arms can be just as simple as having two robots communicate. It is a very simple form of communication and is done with simple ones and zeros; or "ons" and "offs".



In this activity we will reuse the theory of communicating with simple ones and zeros; or "ons" and "offs" from the previous activity. We will replace the robots with two microcontrollers. For this activity, we will focus on the wiring and syntax programming for two Arduino microcontrollers.

The limit switch attached to each microcontroller will control the on and off function of the other controller's motor.

HELPFUL TIPS

This activity was written for arduino uno microcontrollers, but the principles taught in this activity could be applied to any other microcontroller as well. The principles of communication between devices are the same, but the hardware and software setup may be different. Even when using different "brands" of arduinos, the programming may be a bit different. If you try something as presented here, and it does not work with your microcontroller, you may have to do some troubleshooting to get it to work. It is also advisable to check the documentation for microcontrollers, as well as any user forums on line as well.

KEY VOCABULARY

- Limit Switch
- If/Else Statement
- Loop
- Void/Function
- Handshaking

EQUIPMENT & SUPPLIES

- 2 Arduino Uno Microcontrollers
- 2 Limit Switches
- 2 USB Cables
- Jumper Wires
- Handshake Modules
- 4 LEDs
- 4 100 Ohm Resistors
- 2 4.7K Ohm Resistors

ESSENTIAL QUESTIONS

Essential questions answered in this activity include:

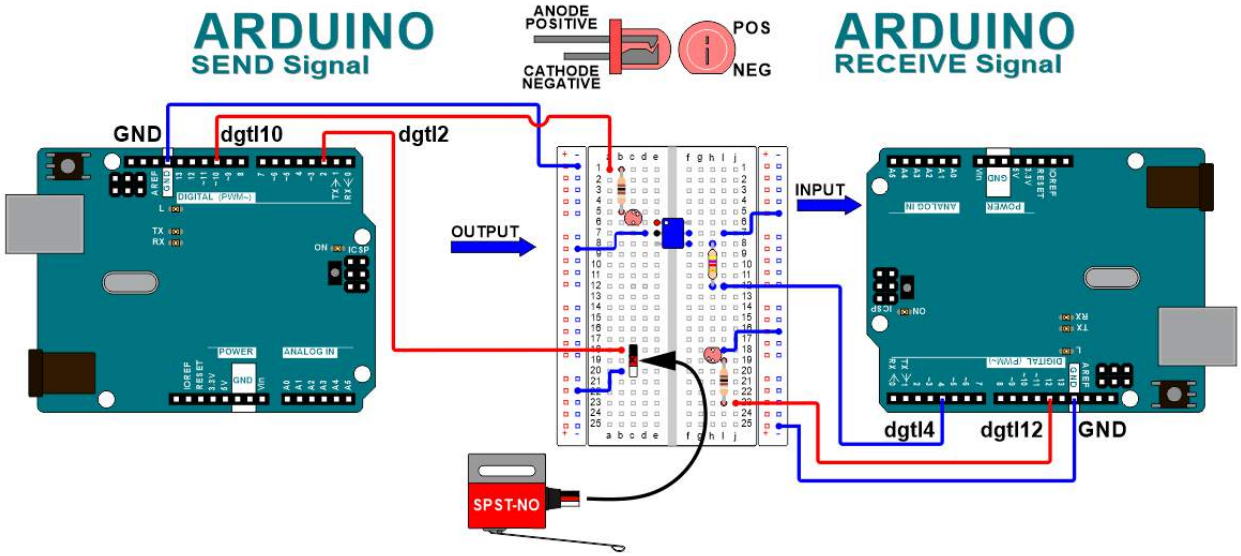
- How do I make my microcontroller send and receive signals?
- What kind of software is necessary to communicate?
- How do I wire the hardware to communicate?
- How do I troubleshoot a complex system?

PROCEDURE

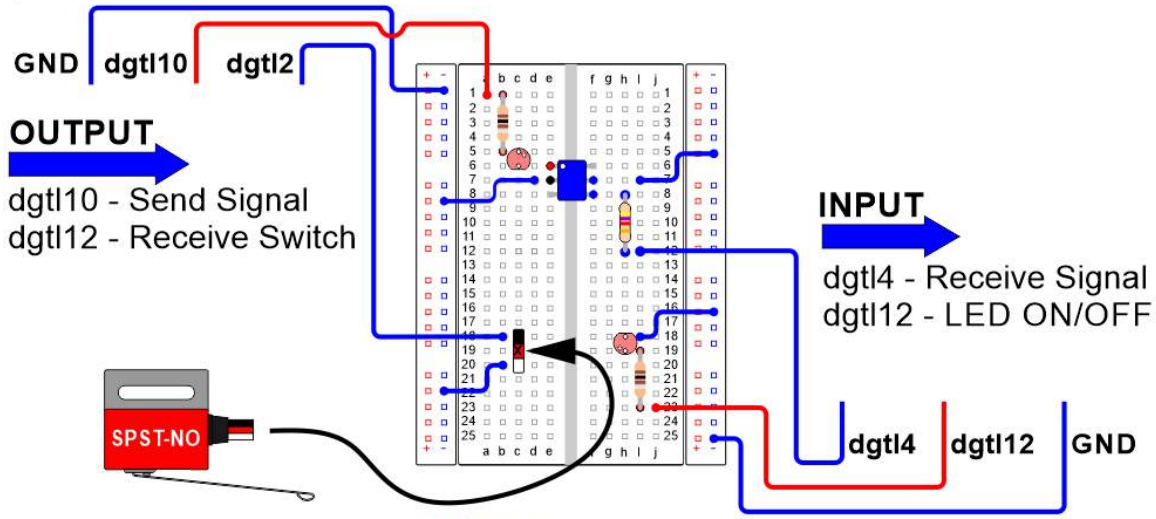
Each Arduino will be able to control the ON/OFF of the LED on the other Arduino.

1. The *digital output* of each Arduino will be attached to a *digital input* of the other Arduino
2. A SPST Normally Open (N.O.) *limit switch* will be connected to each Arduino
3. An LED will be used as a controllable output on each LED
4. Build and wire 2 handshake modules. One as seen below and one mirrored.

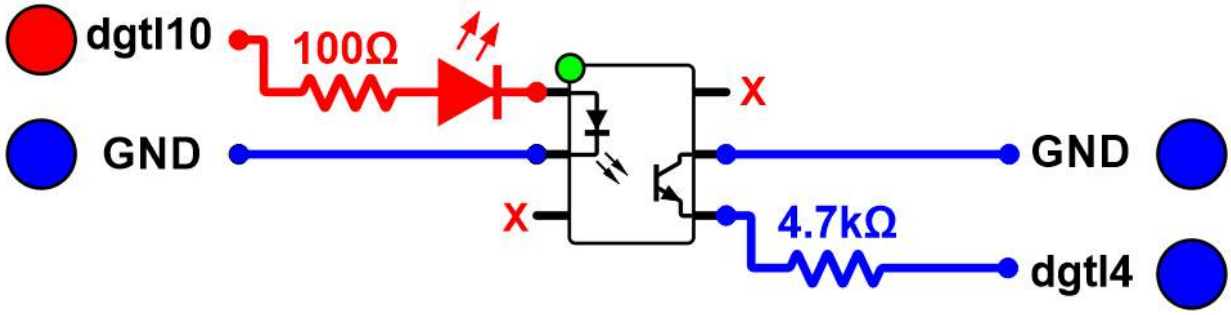
The wiring diagram to send a signal one way is shown below:



```
void setup() {
pinMode (2, INPUT_PULLUP); // Set Port 2 to Digital Pullup (Limit Switch)
pinMode (10, OUTPUT); // Set Port 10 to Output (Send Signal)
}
```

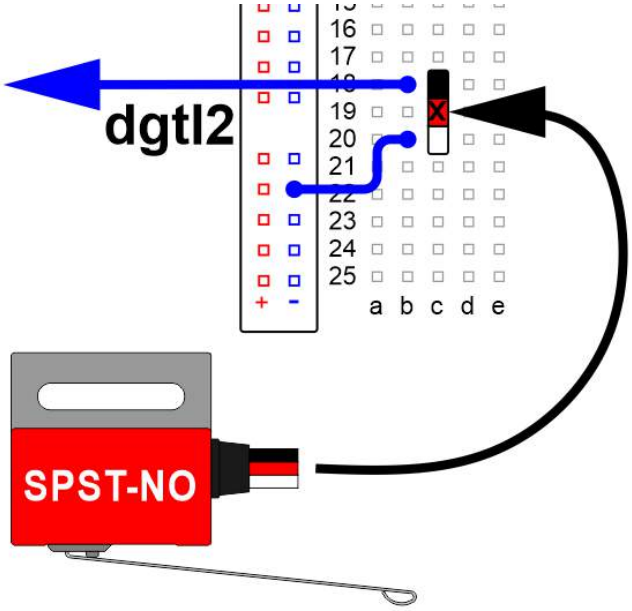



```
void setup() {
pinMode (4, INPUT_PULLUP); // Set Port 4 to Digital Pullup (Receive Signal)
pinMode (12, OUTPUT); // Set Port 12 to Output (Turn On LED)
}
```



Wire a *limit switch* to each Arduino on dgtl2.

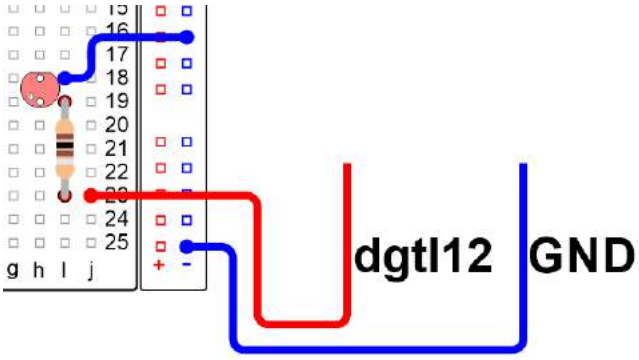
Repeat the process for the other Arduino




```
pinMode (2, INPUT_PULLUP);
```

Wire an LED to dgtl12.

Repeat the process for the other Arduino




```
pinMode (12, OUTPUT);
```

```
void setup() {
// ****************************************
// Send Setup
pinMode (2, INPUT_PULLUP); // Set Port 2 to Digital Pullup (Limit Switch)
pinMode (10, OUTPUT); // Set Port 10 to Output (Send Signal)
// ****************************************
// Receive Setup
pinMode (4, INPUT_PULLUP); // Set Port 4 to Digital Pullup (Receive Signal)
pinMode (12, OUTPUT); // Set Port 12 to Output (Turn On LED)
}
```

Once all of the wiring is complete start developing the syntax programming for this activity. Start by defining the individual steps that will control each individual microcontroller (the program for one controller should be identical to the other)

PRESS A LIMIT SWITCH	→	SEND SIGNAL
RECEIVE A SIGNAL	→	TURN ON LED

Now take care of the other conditions….

STOP PRESSING SWITCH	→	STOP SENDING SIGNAL
STOP RECEIVING A SIGNAL	→	TURN OFF LED

What has to be done when we are not sending or receiving signals?

Start by grouping individual steps into like groups

PRESS MY LIMIT SWITCH	→	SEND SIGNAL TO OTHER CONTROLLER
DO NOT PRESS MY LIMIT SWITCH	→	STOP SENDING SIGNAL TO OTHER CONTROLLER

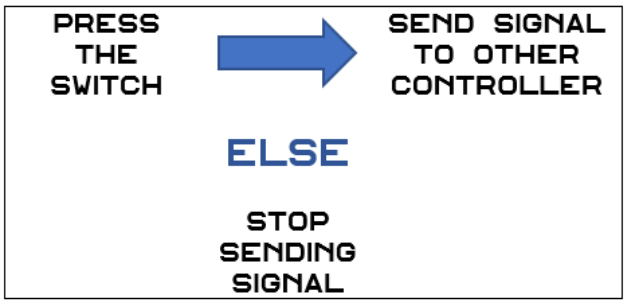
AND

RECEIVE A SIGNAL FROM OTHER CONTROLLER	→	TURN ON MY LED
DO NOT PRESS MY LIMIT SWITCH	→	STOP SENDING SIGNAL TO OTHER CONTROLLER

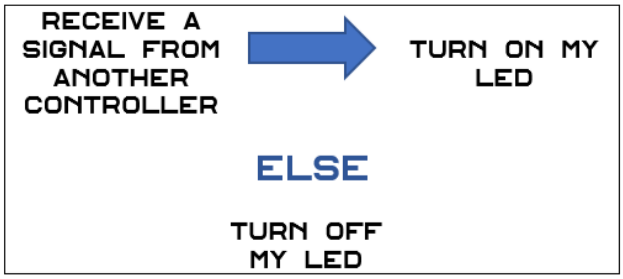
```
if(Condition) {
   // Body
}
if(Condition) {
   // Body
}
```

We should notice that there are only two possible situations for each grouping. It is, or is not. We can simplify our code to: "If a specific condition is met, do operation 1 else, do operation 2".

```
if(Condition) {
   // Body
}
else {
   // Body
}
```



OR



We will use a digitalRead() command to evaluate if an input signal is high or low.

```
digitalRead (2)==LOW
```

Reminder

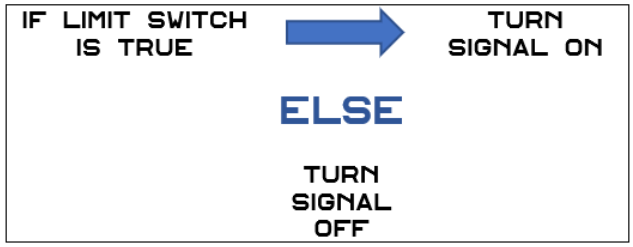
1 (=) sets a value

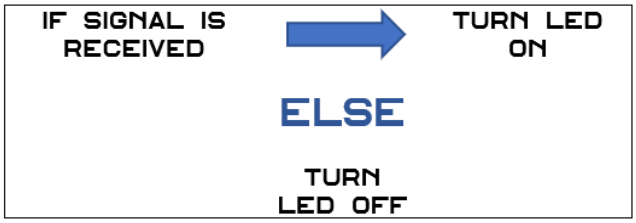
2 (==) evaluates a value

Because we are using dgtl2 as a pullup input, all of its values will read backwards. On is Off and Off is on… HIGH is OFF and LOW is ON

Now simplify the code down to only the essentials

```
if(digitalRead (2) ==LOW) {
   // Body
}
else{
   // Body
}
```





Instead of having to read the condition in our IF statement as ***digitalRead(2)***, we can set the port value to a variable. This will make reading and troubleshooting our programs easier.

An example is shown to the right.

```
int limitSwitch =  digitalRead (2);
```

We can now use limitSwitch in place if digitalRead(2).

```
if(limitSwitch ==LOW) {
   // Body
}
else{
   // Body
}
```

In order to get the program to constantly evaluate either or both situations (multiple If statements can be true at one time) put the program inside the function LOOP.

```
void loop() {
// Body
}
```

Use the digitalWrite command each time we need to turn on and off the digital output.

```
digitalWrite(10, HIGH); // Set Port10 to ON
```

The entire program is shown in the example here:

```
void loop() {
//************************************
//Send Signal Conditions
int Variable =  Port;
   if(Condition) {
     //Body
   }
   else{
     //Body
   }
//************************************
//Receive Signal Conditions
int Variable =  Port;
   if(Condition) {
     //Body
   }
   else{
     //Body
   }
}
```

```
void loop() {
//************************************
//Send Signal Conditions
int limitSwitch =  digitalRead (2);
   if(limitSwitch==LOW) {
     digitalWrite(10, HIGH);
   }
   else{
     digitalWrite(10, LOW);
   }
//************************************
//Receive Signal Conditions
int receiveSignal =  digitalRead (4);
   if(limitSwitch==LOW) {
     digitalWrite(12, HIGH);
   }
   else{
     digitalWrite(12, LOW);
   }
}
```

```
void setup() {
//************************************
//Send Signal Setup
pinMode (2, INPUT_PULLUP); // Set Port 2 to Digital Pullup
pinMode (10, OUTPUT); // Set Port 2 to Output
//************************************
//Receive Signal Setup
pinMode (4, INPUT_PULLUP); // Set Port 2 to Digital Pullup
pinMode (12, OUTPUT); // Set Port 2 to Output
}
```

```
void loop() {
//***********************************
//Send Signal Conditions
int limitSwitch =  digitalRead (2); // Set Variable limitSwitch to Port2 Value
   if(limitSwitch==LOW) { // When the Limit Switch is pressed, Send Signal
     digitalWrite(10, HIGH); // Set Port10 to ON
   }
   else{ // While the Limit Switch is not Pressed
     digitalWrite(10, LOW); // Set Port10 to OFF
   }
//***********************************
//Receive Signal Conditions
int receiveSignal =  digitalRead (4); // Set Variable receiveSignal to Port4 Value
   if(limitSwitch==LOW) { // When the Signal is Received, LED ON
     digitalWrite(12, HIGH); // Set Port12 to ON
   }
   else{ // While the Limit Switch is not Pressed
     digitalWrite(12, LOW); // Set Port12 to OFF (LED OFF)
   }
}
```

Once this portion of the program is completed, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

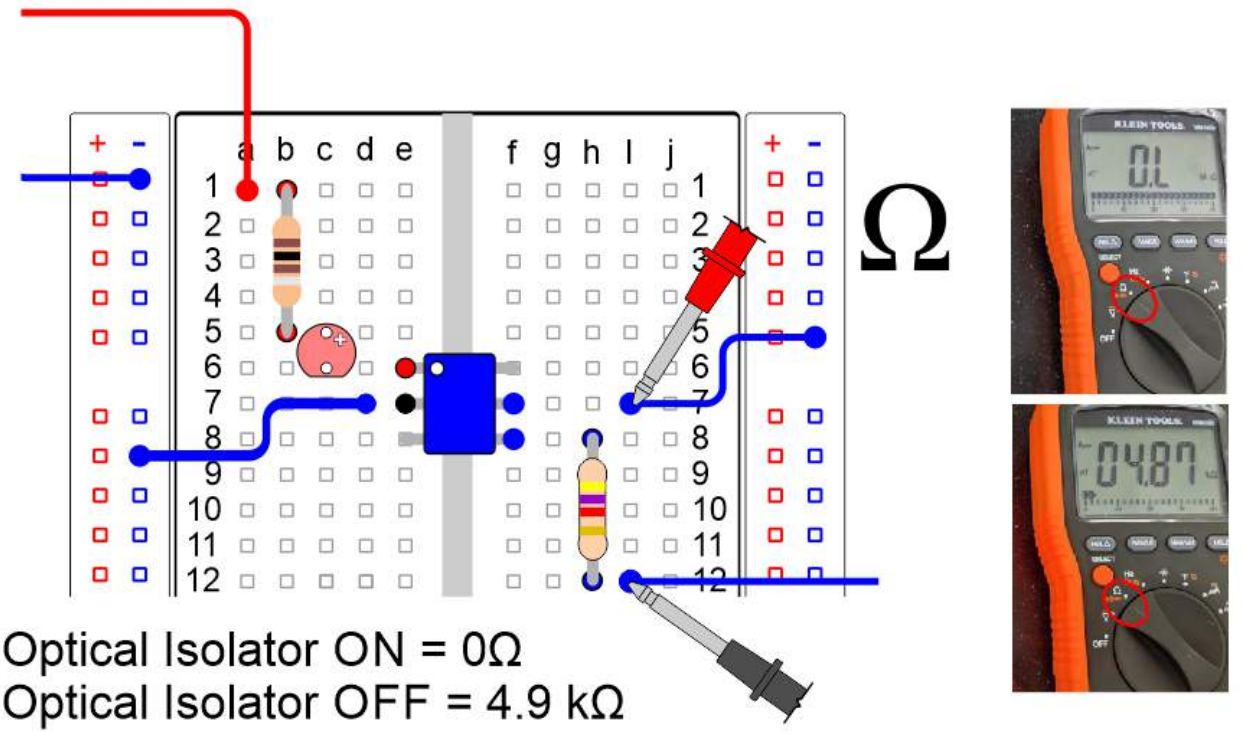
If your set up did not work correctly the first time, what did you have to do to make it work?

TROUBLESHOOTING THE OPTICAL ISOLATOR

HELPFUL TIPS

When the digital Output is turned on / high, the LED on the handshake module should turn on.

If the Debugger window reads correctly, the LED turns on, but no signal is being seen by the other Arduino, check your wiring. As a last resort, the Optical Isolator could be damaged. Ask your instructor to help you evaluate the signal side of the Isolator using a voltmeter and the images below. Voltmeter should be set to OHMS. The meter should read zero when no signal is present and approximately the value of the resistor used on the signal side when a signal is present.



CONCLUSION

1. *What are some similarities of microcontroller programming and blockly? List at least two and explain how they are similar.*

2. *What are some differences between microcontroller programming and blockly? List at least two and explain how they are different.*

3. *Why is it important to use the optical isolator when communicating? Explain in your own words.*

GOING BEYOND

Finished early? Try the action below. When finished, show your instructor and have them initial on the line.

_____ 1. Use a motor instead of an LED.

Be sure to consider what type of motor you are using, and be sure to check with your teacher before attempting this; especially if you switch outputs on the microcontroller.

9 Blockly - Handshaking -VEX to VEX

NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Often the process of handshaking goes beyond a robot's need to communicate with another robot. In industry the process of have Programmable Logic Controllers (PLCs) communicate with each other, or Robotic arms can be just as simple as having two robots communicate. It is a very simple form of communication and is done with simple ones and zeros; or "ons" and "offs".

In this activity we will reuse the theory of communicating with simple ones and zeros; or "ons" and "offs" from the previous activity. We will replace the robots with two microcontrollers. For this activity, we will focus on the wiring and syntax programming for two VEX microcontrollers.

The limit switch attached to each microcontroller will control the on and off function of the other controllers motor.

KEY VOCABULARY

- Limit Switch
- If/Else Statement
- While True Loop (Forever Loop)
- Handshaking

EQUIPMENT & SUPPLIES

- 2 VEX Cortex
- 2 Limit Switches
- 2 VEX Cables
- Servo Extension Cables
- Handshake Modules
- 2 Motors
- 2 VEX Batteries
-

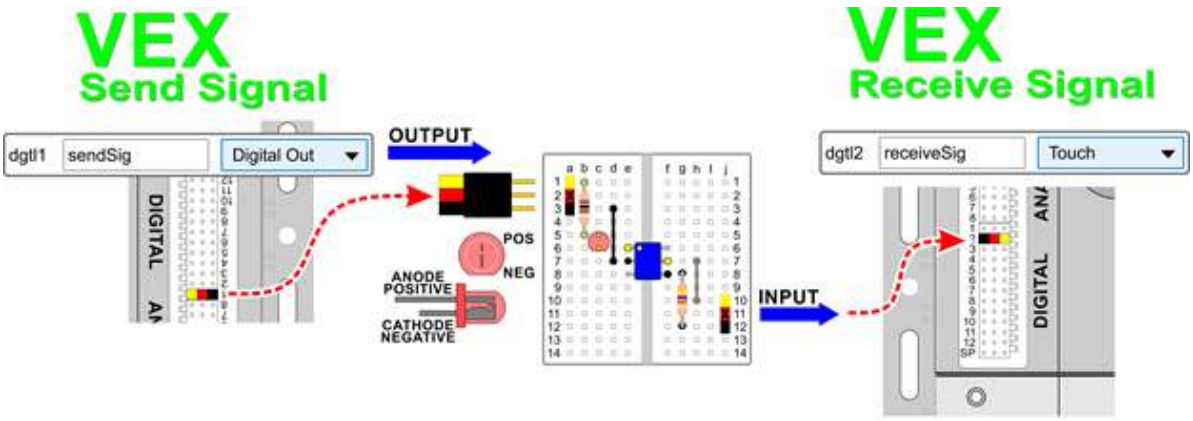
ESSENTIAL QUESTIONS

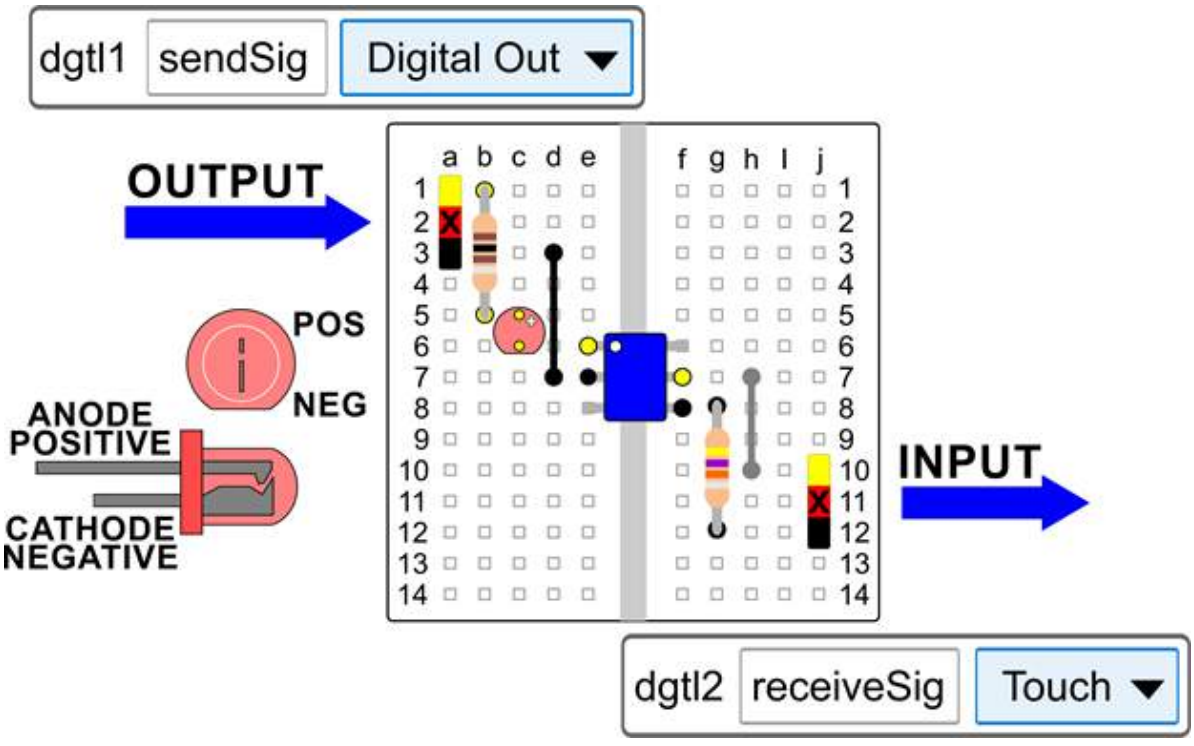
Essential questions answered in this activity include:

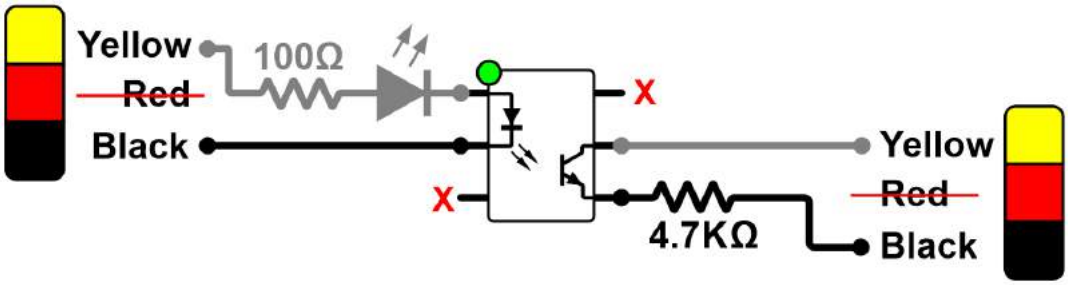
- How do I make my VEX Cortex send and receive signals?
- What kind of software is necessary to communicate?
- How do I wire the hardware to communicate?
- How is RobotC similar to Arduino software? How is it different?

PROCEDURE

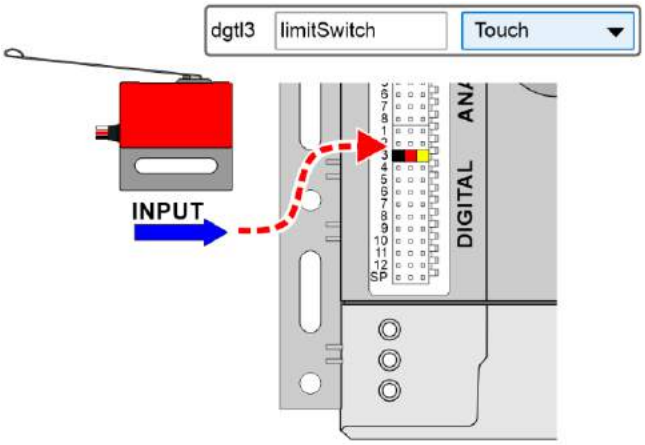
1. The *digital output* of each Cortex (dgtl1) will be attached to a touch input (dgtl2) of the other Cortex
2. Build and wire 2 handshake modules. One as seen below and one mirrored.





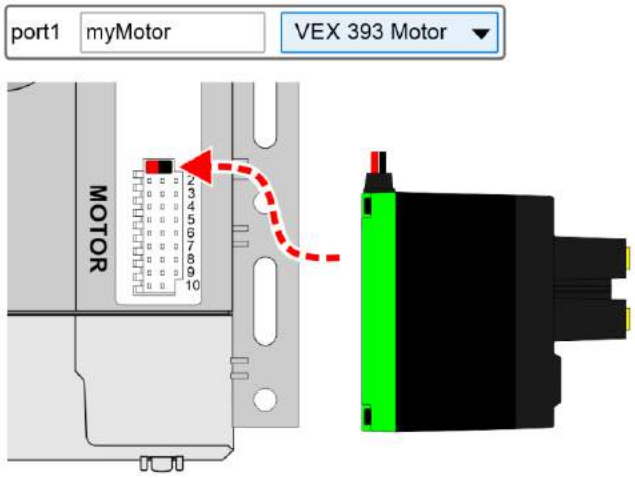


Wire a *limit switch* to each cortex as a Touch on dgtl3.

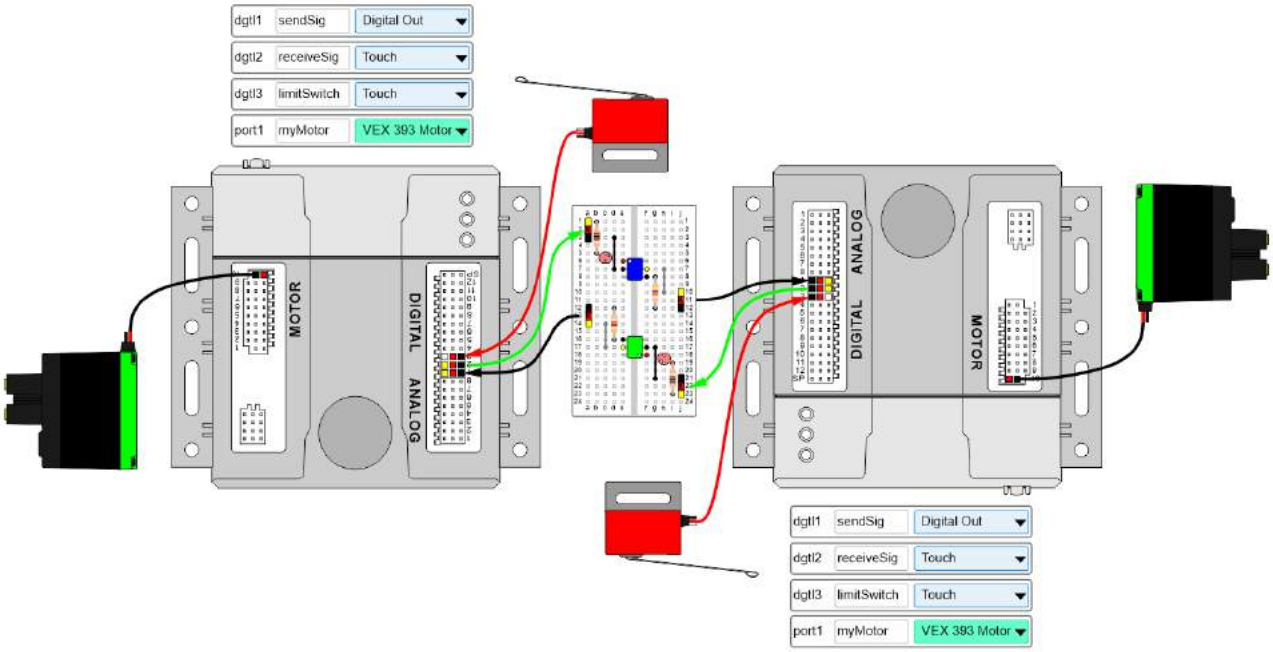


Repeat the process for the other Cortex

Wire a motor to either motor port 1 or port 2.



Repeat the process for the other Cortex



Once all of the wiring is complete develop the syntax programming for this activity. Start by defining the actions that will control each individual microcontroller (the program for one controller should be identical to the other)

PRESS A LIMIT SWITCH	→	SEND SIGNAL
RECEIVE A SIGNAL	→	TURN ON MOTOR

Define the other conditions…. What to do when no signal is sent or received.

STOP PRESSING SWITCH	→	STOP SENDING SIGNAL
STOP RECEIVING A SIGNAL	→	TURN OFF MOTOR

Now start grouping our individual actions into like groups

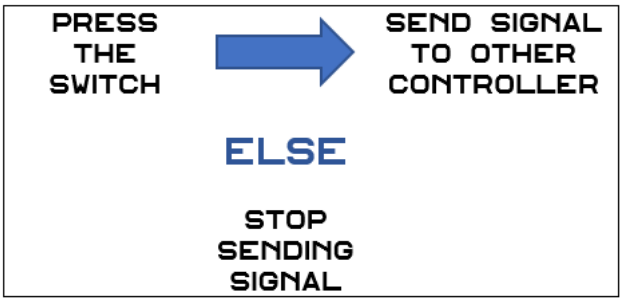
```
if (condition)
{
  body
}
if (condition)
{
  body
}
```

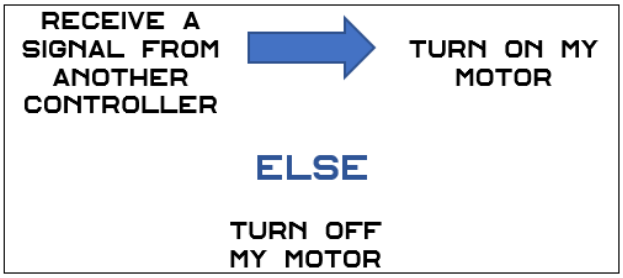
PRESS MY LIMIT SWITCH	→	SEND SIGNAL TO OTHER CONTROLLER
DO NOT PRESS MY LIMIT SWITCH	→	STOP SENDING SIGNAL TO OTHER CONTROLLER

RECEIVE A SIGNAL FROM OTHER CONTROLLER	→	TURN MY MOTOR ON
DO NOT RECEIVE A SIGNAL	→	TURN MY MOTOR OFF

We should notice that there are only two possible situations for each grouping. It is, or is not. We can simplify our code to If a specific condition is met, do operation 1 else, do operation 2.

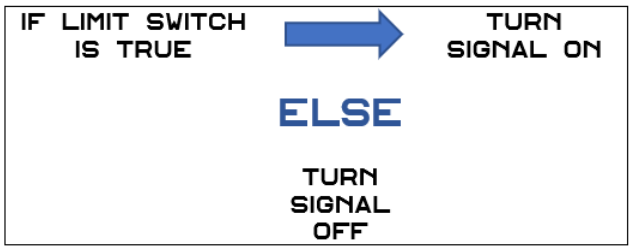
```
if (condition)
{
  body
}
else
{
  body
}
```

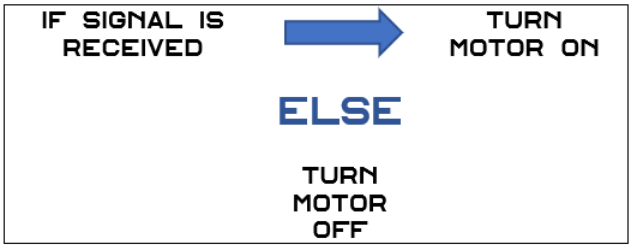




Now simplify the code down to only the essentials

```
if (SensorValue[limitSwitch]==1)
  {
    body
  }
```





In order to get the program to constantly evaluate either or both situations (multiple If statements can be true at one time) we will put our program in a While True Loop

```
while (true)
{
  body
}
```

We will use the SensorValue command each time we need to evaluate or a sensor value or turn on and off the *digital output*.

```
SensorValue[sendSig]=1; // Turn On dgtl1
SensorValue[sendSig]=0; // Turn Off dgtl1
if (SensorValue[limitSwitch]==1)//Is Touch ON?

if (SensorValue[receiveSig]==1)//Is Touch ON?
```

Reminder

1 (=) sets a value

2 (==) evaluates a value

An example of the entire program in RobotC is seen below.

Program Only

```
task main()
{
  while (true)
  {
    if (condition)
    {
      body
    }
    else
    {
      body
    }
    if (condition)
    {
      body
    }
    else
    {
      body
    }

  }
}
```

Program with Comments

```
task main()
{
  while (true) //Forever Loop
  {
    if (condition)//Limit Switch is ON
    {
      body // Turn ON dgtl1
    }
    else
    {
      body// Turn OFF dgtl1
    }
    if (condition)//Signal Received
    {
      body //Motor ON
    }
    else
    {
      body //Motor OFF
    }
  }
}
```

```
#pragma config(Sensor, dgtl1,  sendSig,        sensorDigitalOut)
#pragma config(Sensor, dgtl2,  receiveSig,     sensorTouch)
#pragma config(Sensor, dgtl3,  limitSwitch,    sensorTouch)
#pragma config(Motor,  port1,           myMotor,       tmotorVex393_HBridge, openLoop)
```

```
task main()
{
  while (true) //Forever Loop
  {
    if (SensorValue[limitSwitch]==1)//Limit Switch is ON
    {
      SensorValue[sendSig]=1; // Turn ON dgtl1
    }
    else // No Limit Switch
    {
      SensorValue[sendSig]=0;// Turn OFF dgtl1
    }
    if (SensorValue[receiveSig]==1)//Signal Received
    {
      startMotor(myMotor, 127); //Motor ON
    }
    else //No Signal
    {
      stopMotor(myMotor); //Motor OFF
    }
  }
}
```

Once this portion of the program is completed, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

If your set up did not work correctly the first time, what did you have to do to make it work?

TROUBLESHOOTING THE OPTICAL ISOLATOR

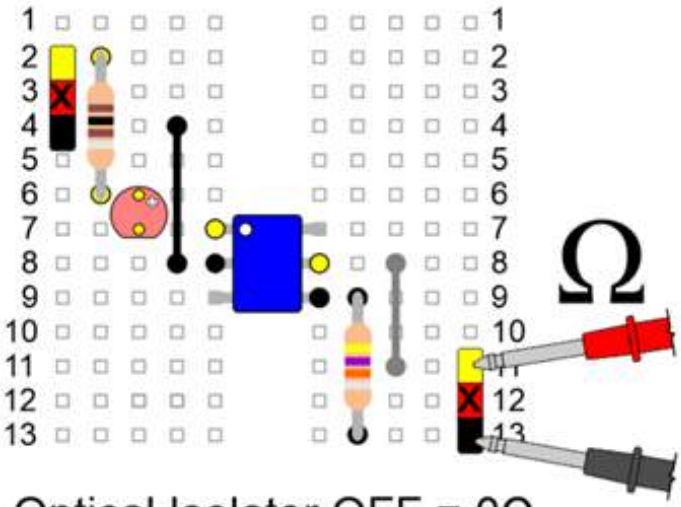
Use the Debugger window to troubleshoot your program. When the Limit Switch is pressed, both the dgtl1 (Send Signal) and dgtl3 (Limit Switch) should read true

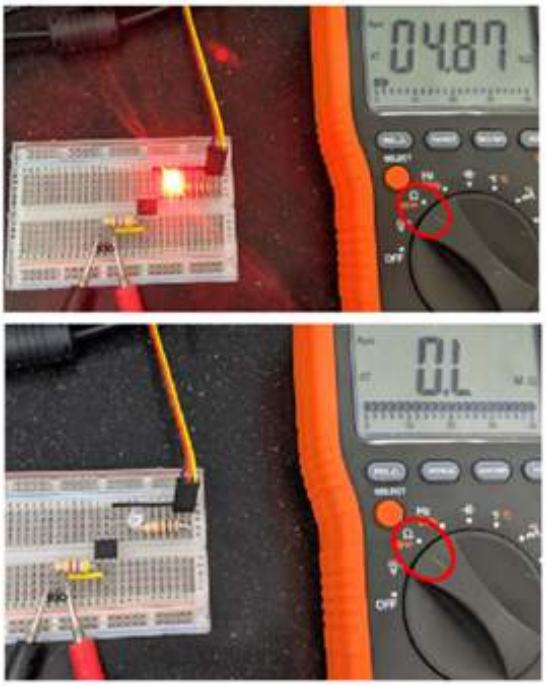
Sensors

Index	Sensor	Type	Value
dgtl1	sendSig	Digital Out	1
dgtl2	receiveSig	Touch	0
dgtl3	limitSwitch	Touch	1

When the digital Output is turned on / high, the LED on the handshake module should turn on.

If the Debugger window reads correctly, the LED turns on, but no signal is being seen by the other cortex, check your wiring. As a last resort, the Optical Isolator could be damaged. Ask your instructor to help you evaluate the signal side of the Isolator using a voltmeter and the images below. Voltmeter should be set to OHMS. The meter should read zero when no signal is present and approximately the value of the resistor used on the signal side when a signal is present.





CONCLUSION

1. *What would you have to do to make this program run five times without any human intervention? Explain fully below.*

2. *What other inputs could you use on your VEX Cortex to start this process? Use the Dobot Input/Output Guide to answer this question, and do not attempt to try it without your instructor's permission.*

3. *What other outputs could you use on your VEX Cortex? Be sure to check with your instructor first!*

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

_____ 1. Change the Motors to servos. Be sure to get your instructor's permission, and be sure to use the correct inputs and outputs in RobotC.

10 Blockly - Handshaking - Dobot to VEX

NAME: _____ *Date:* _____ *Section:* _____

INTRODUCTION

Often robotic arms need to communicate with other devices or controllers in a work cell, or factory. This is called *HANDSHAKING* and can be done between different machines, devices and robots. It is a very simple form of communication and is done with simple ones and zeros; or "ons" and "offs".

In this activity you will use all of the knowledge learned in previous activities including activity seven and eight to make a Dobot Magician Robot communicate with a VEX cortex.

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

KEY VOCABULARY

- Inputs and Outputs
- Variables
- Function / Voids
- Handshaking

EQUIPMENT & SUPPLIES

- Dobot Magician
- 1" cylinders or cubes
- Suction Cup Gripper
- Various VEX parts for grinding station
- VEX Cortex
- Simple VEX Grinding Station
- Handshake Modules

PROCEDURE

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

1. Set up ***Robot*** with a suction cup - ***GP1 & SW1***
2. Wire ***Robot*** with an ***OUTPUT*** signal ***GP2 - EIO13***.
3. Wire ***Robot*** with an ***INPUT*** signal ***GP5 - EIO5.***
4. Wire the ***Microcontroller*** Digital Output (sendSig) to *dgtl1* and the Touch (receiveSig) to *dgtl2*
5. Wire both the ***Robot*** and the ***Microcontroller*** to the handshake modules as shown on the next pages.

Order of operations

ROBOT

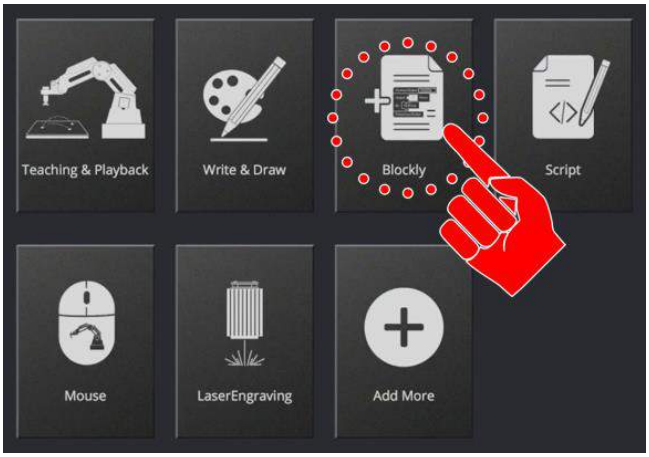
- Move - Home
- Move - ABPick
- Move - ATPick
- Pick Up Cube
- Jump - ATMachine
- Send ***OUTPUT*** signal to VEX
- Wait for ***INPUT*** Signal
- Jump- ATPick
- Release Cube
- Move - ABPick
- Move - Home

VEX - Microcontroller

- Wait for ***INPUT*** Signal
- Turn on Grinding Station
- Run the station for 3 seconds
- Send ***OUTPUT*** signal to Dobot

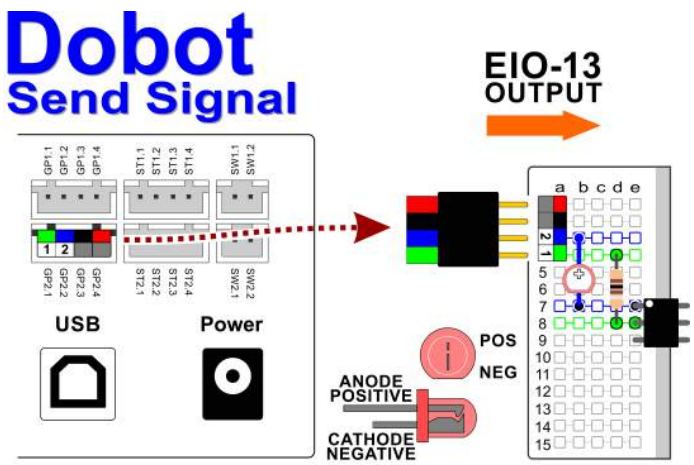
Open up Blockly in the software

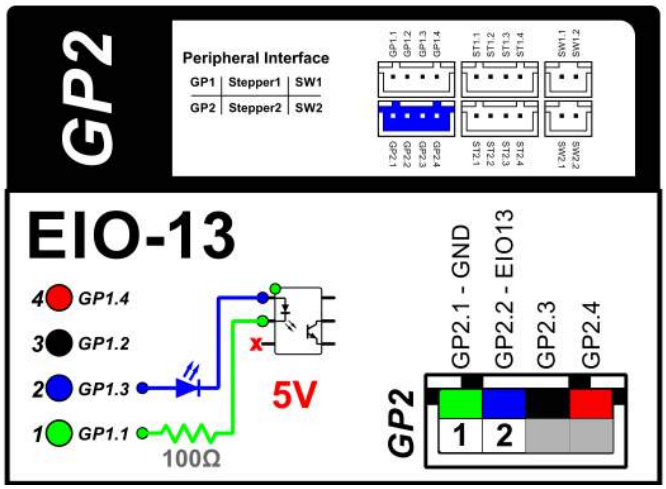
When you re-open this program check that the name of the file on top matches the code in the file, if it does not, you may end up overwriting another program

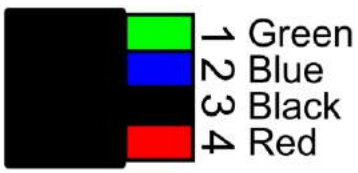


Dobot Setup

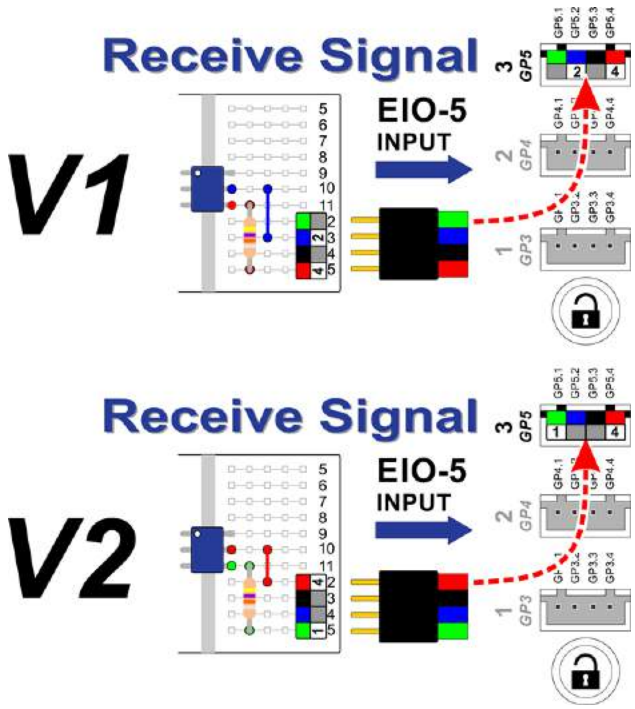
SEND Output

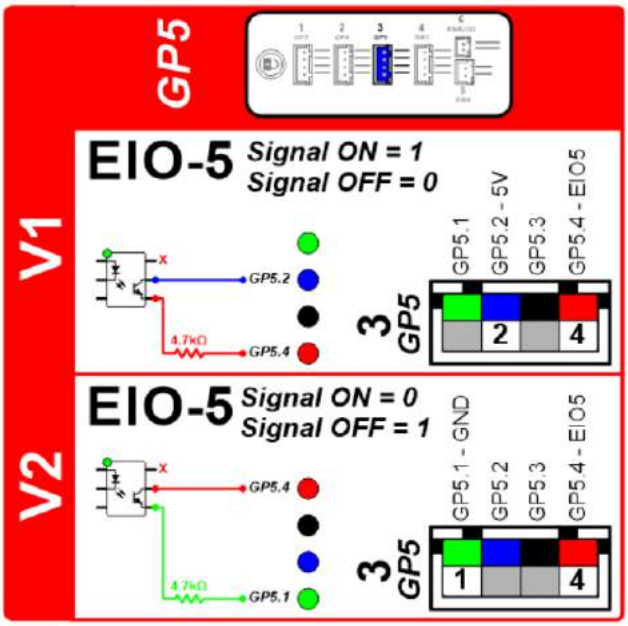


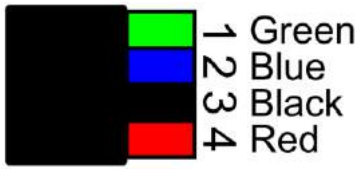




RECEIVE Input

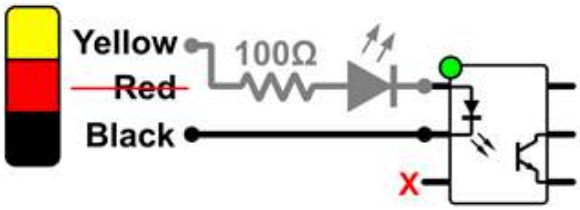




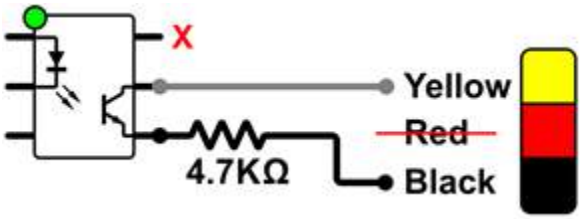


VEX Microcontroller Setup

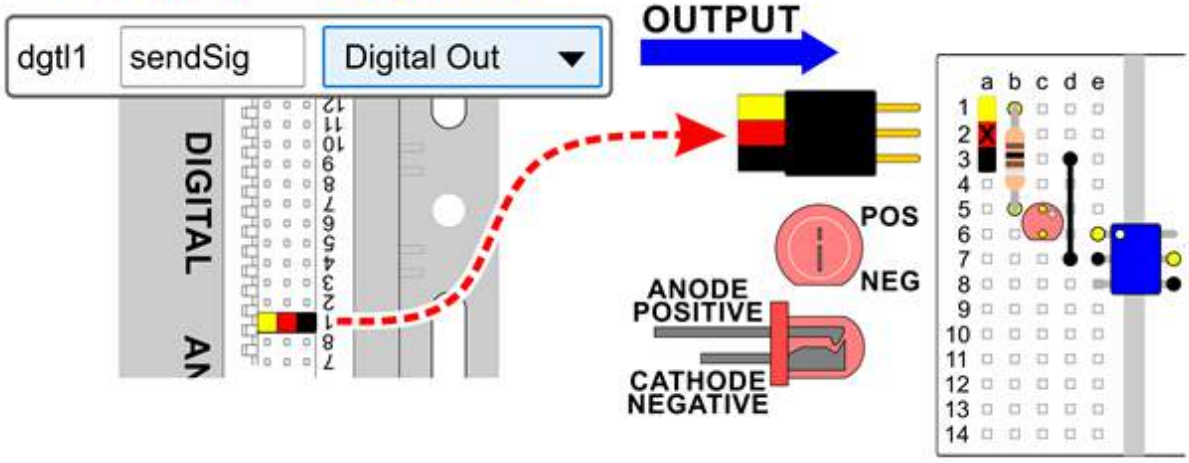
SEND Output



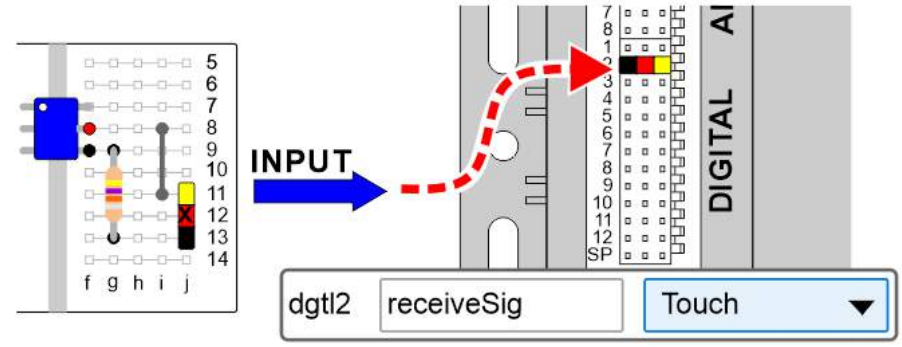
RECEIVE Input



VEX Send Signal

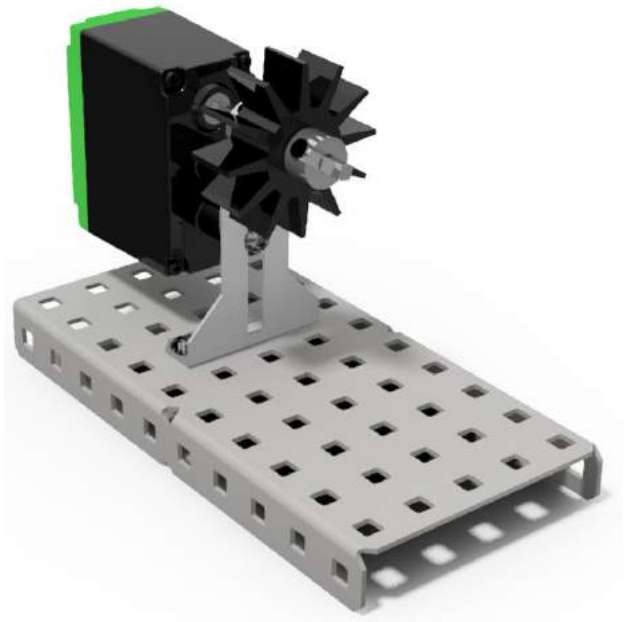


VEX Receive Signal



Be sure to consult the Input/Output Guide if you want to use other inputs and outputs, as damage to your robot or your other equipment may result.

Create a simple Grinding Station using VEX material. Use the rubber intake wheel as the grinding wheel.



…… Where's the rest of the activity? If you need additional assistance, please refer to the previous activities. All concepts were taught, and this one was left open ended so that you may apply what you've learned.

Once the program is completed, run it and see if it works correctly. If it does not work, troubleshoot it until it does.

If your set up did not work correctly the first time, what did you have to do to make it work?

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

_____ 1. Have the robot take blocks from a matrix, machine them, and drop them off at a finished location (*example:* dropped into a bin).

_____ 2. Make another machine out of VEX parts and make the robot perform two operations.

_____ 3. Have the robot pick them up when a switch is hit, and then palletize them after grinding.

11 Blockly - Workcell Design

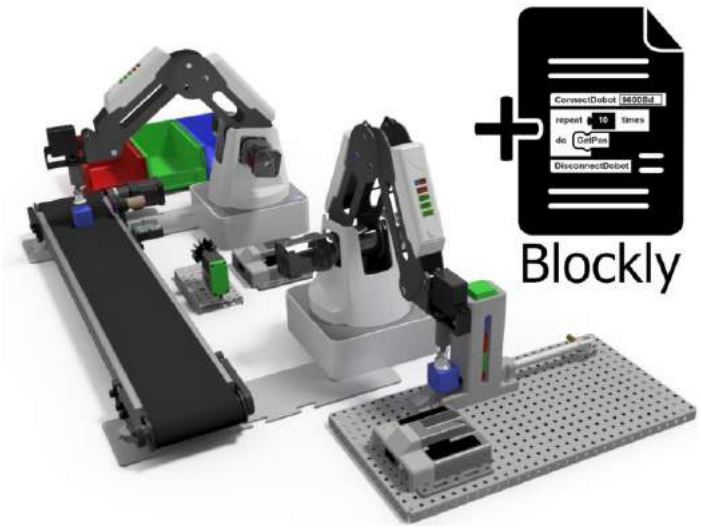
NAME: _____ *Date:*_____ *Section:* _____

INTRODUCTION

A robotic workcell is defined as the complete environment around a robot. This environment may include tools, machines and/or other robots.

In this activity you will use a robot and a microcontroller system to recreate a workcell. Your workcell will incorporate all of the devices that you have learned about in previous activities including:

- Inputs & outputs
- Sensors
- Conveyor belt
- Machines
- Robots



An example of a workcell using robots, prototyped parts and VEX components

KEY VOCABULARY

- Handshaking
- Workcell
- Output
- Nesting
- Optical Isolator
- Sensor
- Input
- Palletize

EQUIPMENT & SUPPLIES

- Robot Magician(s)
- Microcontroller System & Components
- 1” cylinders or cubes
- Servo extension cables
- *Dobot Input/Output Guide*
- Breadboard/wire/4N25 Optical Isolator and 100 ohm & 4.7K ohm resistors
- Conveyor System
- DobotStudio software
- RobotC or other VEX control software
- Pneumatic Gripper or Suction Cup Gripper
- Handshake device
- Input & Output devices
- VEX, PIC, Arduino may all be used in this activity, but wiring may vary.

ESSENTIAL QUESTIONS

Essential questions answered in this activity include:

- How do you integrate robots and other parts of a workcell to complete a given task?
- How do you safely communicate between a microcontroller and a robot?
- What are the different types and styles of inputs and outputs needed to complete your given tasks?
- Which end of arm tooling is most appropriate for your workcell?
- Where is it appropriate to use the jump command within my workcell?
- Where would it be appropriate in your programming to use either variables or functions while programming?
- What components of blockly did you need to complete this task?
- What other software & hardware will I need to complete this task?

PROCEDURE

Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

1. You and your team will design, organize, create, program and test a full work cell. Your instructor will have you pick from the list below the items that must be included in your work cell.

☐ ***Dobot Magician - # of _____***

☐ ***Dobot Conveyor System***

☐ ***Dobot IR Sensor***

☐ ***Dobot Color Sensor***

☐ ***Handshakes - # of _____***

☐ _____

☐ ***Microcontroller - # of _____***

☐ ***Student Designed Feeder System***

☐ ***Student Designed Machine - # of _____***

☐ ***Student Designed Storage System***

☐ _____

☐ _____

Machine Examples	
• Pneumatic Stamping Press	• Painting Station
• Rotary or Reciprocating Saw	• Drying Station
• Welder	• Drilling Station
• Quality Control Station	• ???

2. Be sure to note any additional parameters that are given to you by your instructor (due date, size or storage requirements, and additional items from home. Take notes in the space below.

3. In the time allotted for this project design a workcell that includes the following:
    a. Accurate Robot pick and place routines.
    b. Jumps where appropriate.
    c. Proper usage of Functions and Variables.
    d. A simulation of a manufacturing process.
    e. A palletize or stacking routine.

4. Create a video of your workcell.

Be sure to consult the Dobot Input/Output Guide if you want to use other inputs and outputs, as damage to your robot or your other equipment may result.

CONCLUSION

1. *Make a flowchart/Process flowchart. of your workcell as indicated by your instructor in the space below.*

2. *What's the pseudocode that you used for your microcontroller program? Copy and paste it here.*

3. *What are the inputs you used in your workcell?*

4. *What are the outputs you used in your workcell?*

GOING BEYOND

Finished early? Try some of the actions below. When finished, show your instructor and have them initial on the line.

_____ 1. Make your workcell communicate with someone else's workcell in your class. When your process ends, theirs begins.

_____ 2. Same as number one, but use the same part in both cells. In other words, perform 2 operations in two separate cells, on the same part.

Blockly Glossary

The vocabulary below is used throughout all the blockly activities, lessons, and presentations for the Dobot Magician. These key terms are all related to the Blockly programming language in DobotStudio. Please note that not all commands are defined here, just the ones most important to completing the activities.

Term	Type of Command *(if Applicable)*	Definition
Blockly		A programming language used to program a Dobot Magician. Lines of complex code are represented as simple "blocks" that fit together to form programs. A graphical programming method rather than text based.
ChooseEndTools	Config	A blockly config command that allows you to set the end effector to be used in the program
Comment		Can be used in blockly to name a position, or add a comment in english, that the program will not read. These are commonly used as notes for the programmer and the operator.
Condition		This is what needs to be true in order for a set of instructions or a program to continue. It is also very useful in looping programs.
Delaytime	Basic	A blockly basic command that allows you to delay, or pause in a program for a given amount of time in seconds.
Forever loop		A programming method used to make an instruction or a set of instructions continue forever.
Function		A named section of a program that performs a task. It can also be considered a procedure or a routine and greatly simplifies otherwise complicated programs.
GetADInput	I/O	A blockly I/O command that returns the value of a specified analog input. this can only work with analog inputs inputs 1, 5, 7, 9, 12, and 15.
GetCurrentCoordinate	Motion	A blockly motion command that returns the XYZ values of the robot's current position.
GetJointAngle	Motion	A blockly motion command that returns the value of a specified robot joint.
GetLevelInput	I/O	A blockly I/O command that returns the value of a specified input.

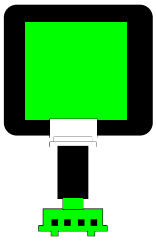
GetPhotoelectricSensor	Additional	A blockly additional command that returns the value of the sensor plugged into a given port.
Function		A named section of a program that performs a task. It can also be considered a procedure or a routine and greatly simplifies otherwise complicated programs. Also called a Void.
Gripper	Motion	A blockly motion command that allows you to turn the gripper on and off in order to grip or release a part.
Home	Basic	A blockly basic command
IdentifyColor	Additional	A blockly additional command that allows you to choose what color to identify with the color sensor: red, green, or blue.
If/Else If/Else	Logic	A blockly logic command that allows a branch in a program. It compares two or more sets of data and if it is true it does one thing, if false it will do another.
JumpTo	Motion	A blockly motion command that will move from one point to another, while increasing the Z height, causing the robot to "jump" to the next position. The default is set to 20 mm.
Laser	Additional	A blockly additional command that turns the laser on and off and allows you to set the power.
Matrix		A rectangular array of parts made of a finite number of rows and columns. a good example is an array of boxes on a pallet.
MoveDistance	Motion	A blockly motion command that allows you to move the robot's end effector relative to where it's position is presently in its work envelope.
MoveLinearRailTo	Additional	A blockly additional command that allows you to move the Linear slide rail to any given point.
MoveTo	Motion	A blockly motion command that allows you to move the robot's end effector to a given XYZ coordinate within the robot's work envelope.
NumberBlock	Math	A blockly math command that allows you to set a variable to a number.
Placeholder		A value used to fill in a blank in a program to be changed to a different value, once that value is determined. For example, the values of XYZ in a program do not need to be known before you write the program.
Repeat	Loop	A blockly loop command that allows you to make a set of instructions repeat a given number of times.
ReturnSum	Math	A blockly math command that can be used to easily change a variable.
SetColorSensor	Additional	A blockly additional command that allows to turn the color sensor

		on and off, what version it is, and what port it is plugged in to.
SetConveyor	Additional	A blockly additional command that allows you tell the program where it is plugged in, and what speed you want it to move at n mm per second.
SetCoordinateSpeed	Config	A blockly config command
SetEndEffectorParams	Config	A blockly config command that allows you to offset the XYZ position of an end effector.
SetJointAngle	Motion	A blockly motion command that allows you to move an individual joint on the robot.
SetJointSpeed	Config	A blockly config command
SetJumpHeight	Config	A blockly config command that allows you to set the jump height in a program.
SetLinearRailSpeed	Config	A blockly config command that allows you to set the velocity and acceleration of the attached linear rail.
SetLinearSlideRail	Additional	A blockly additional command that allows you to tell the program what version of the slide rail you are using, and whether you want it on or off.
SetIOMultiplexing	I/O	A blockly I/O command
SetLostStepParams	Config	A blockly config command
SetMotionRatio	Config	A blockly config command that allows you to set a default velocity and acceleration of a robot's moves between positions.
SetPhotoelectricSensor	Additional	A blockly additional command that allows you to turn the photoelectric sensor on or off, tell it what version it is that is plugged in, and what port it is plugged into.
SetItem	Variable	A blockly command that allows you to set a new variable or change a variable within a program.
SetR	Motion	A blockly motion command
SuctionCup{ON/OFF]	Motion	A blockly motion command that turns the suction cup on or off.
True	Logic	A blockly logic command that can be used to create a forever loop.
Variable		A changeable quantity in a program that can be represented by a word or a letter. Variables can be assigned, changed, or referenced throughout a program.
While loop	Loop	A programming method to make an instruction or set of instructions continue or repeat if a condition is true.

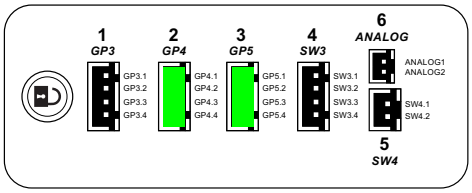
DOBOT Magician V1/V2

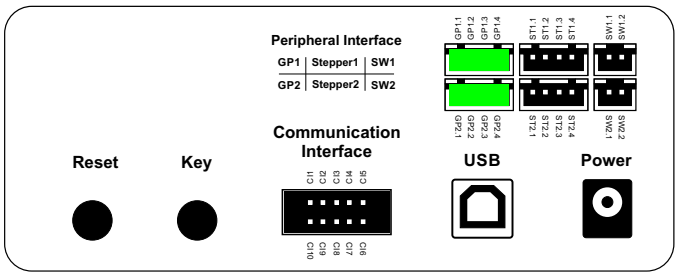
INPUT / OUTPUT GUIDE

Color Sensor
V1 GP2
V2 GP1, 2, 4, 5



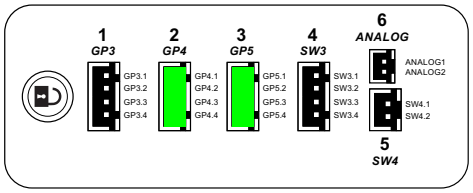
GP1
GP2
GP4
GP5

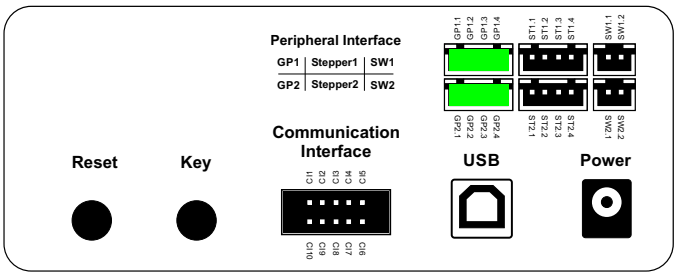


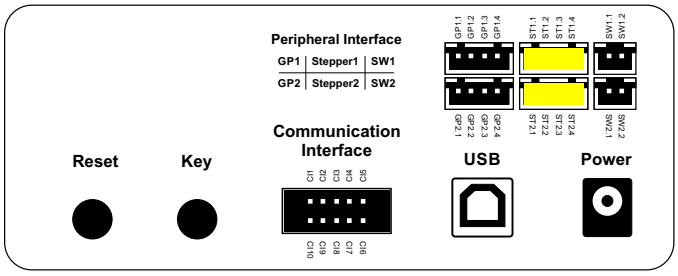


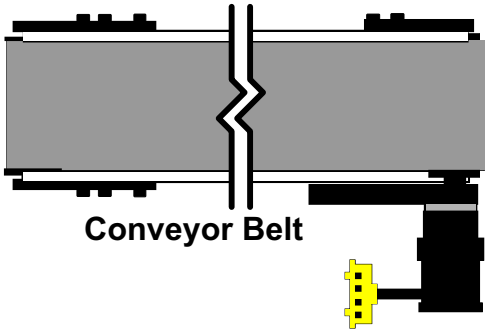
IR Sensor V1 & V2

GP1
GP2
GP4
GP5

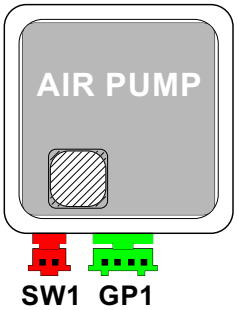




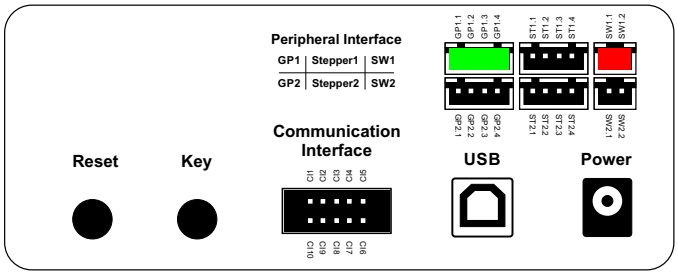




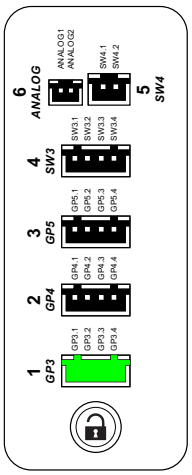
STEPPER 1
STEPPER 2



SW1 GP1



Wrist Rotate Servo



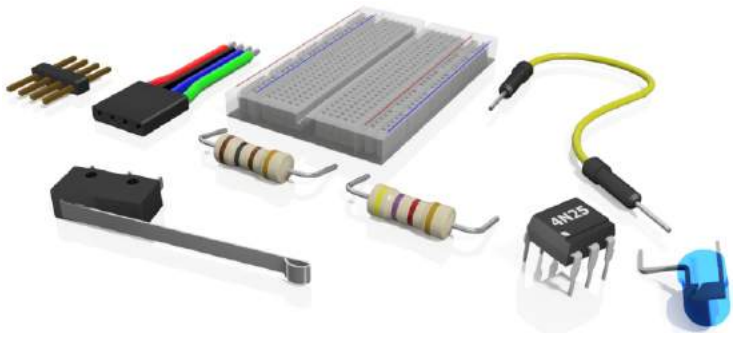
DOBOT Magician

Digital Signal Guide

NEVER wire anything to the Dobot Magician while it has power **ON**
ALWAYS shutdown the Dobot before making connections or damage to the robot could occur.

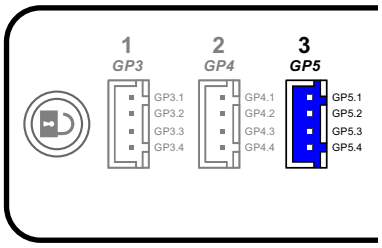
Parts Needed for Safe Communication

- Four Pin Male Headers
- Four Pin Jumper Wires
- 100Ω Resistors
- 4.7kΩ Resistors
- Breadboard
- Optical Isolators 4N25
- LEDs
- Jumper Wires
- Limit Switch



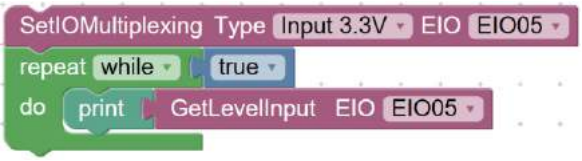
Test INPUT Communication to Dobot

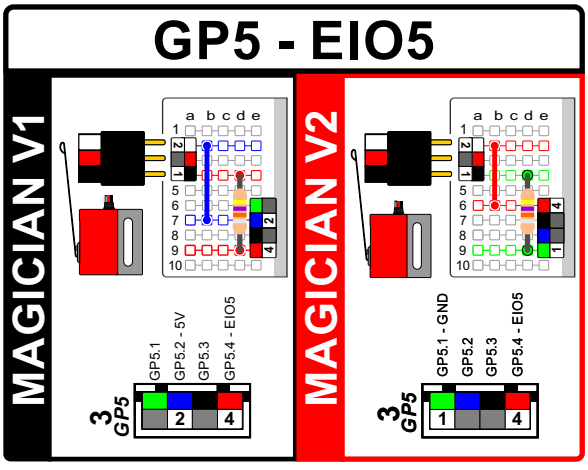
- Four Pin Male Headers
- Four Pin Jumper Wires
- 4.7kΩ Resistors
- Breadboard
- Limit Switch



Running Log:

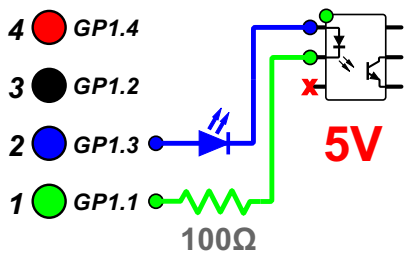
```
[17:07:08]0
[17:07:08]0
[17:07:08]0
[17:07:08]0
[17:07:08]1
[17:07:08]1
[17:07:08]1
```

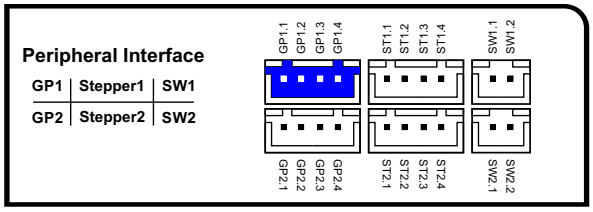


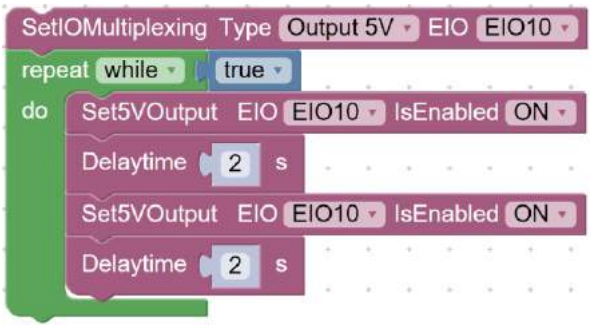


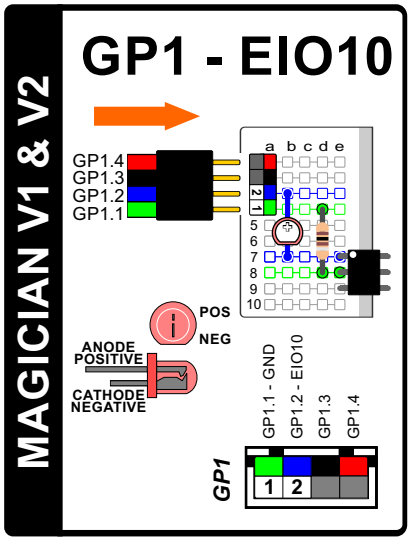
Test OUTPUT Communication from Dobot

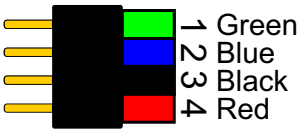
- Four Pin Male Headers
- Four Pin Jumper Wires
- 100Ω Resistors
- Breadboard
- Optical Isolators 4N25
- LEDs











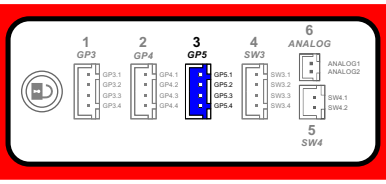
DOBOT Magician V1

Digital Signal Guide

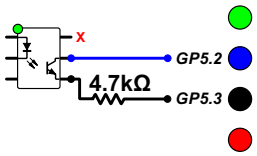
Caution: NEVER wire anything to the Dobot Magician while it has power on. **ALWAYS** shutdown the Dobot before making connections or damage to the robot could occur.

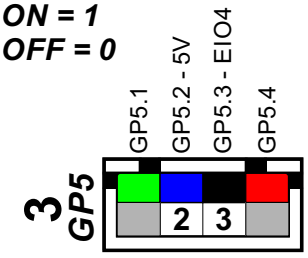
INPUTS

GP5

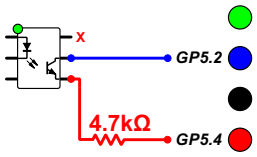


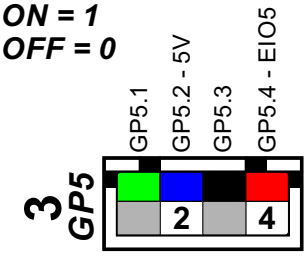
EIO-4 — Signal ON = 1, Signal OFF = 0



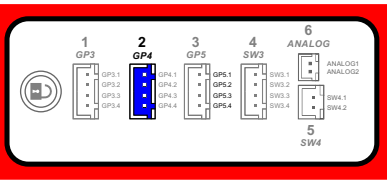


EIO-5 — Signal ON = 1, Signal OFF = 0

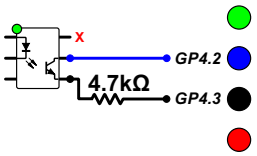


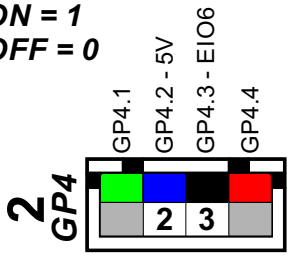


GP4

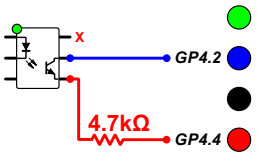


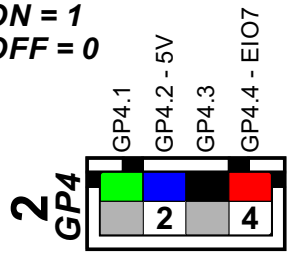
EIO-6 — Signal ON = 1, Signal OFF = 0





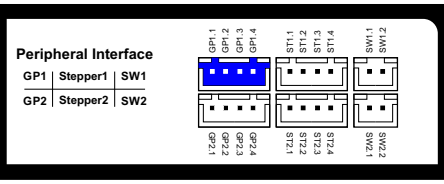
EIO-7 — Signal ON = 1, Signal OFF = 0



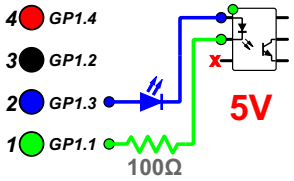


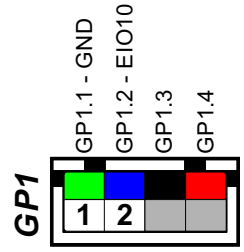
OUTPUTS

GP1

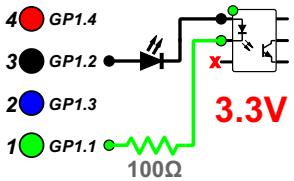


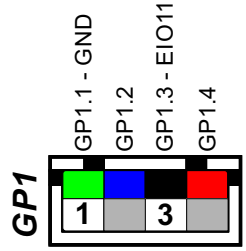
EIO-10



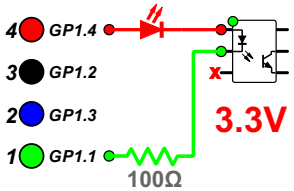


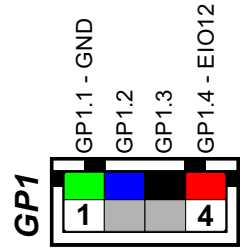
EIO-11



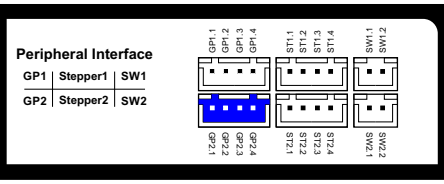


EIO-12

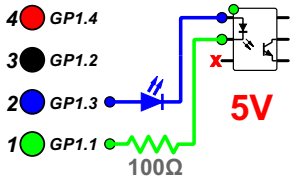


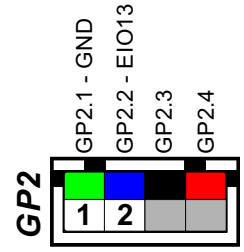


GP2

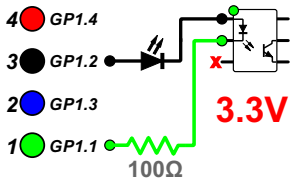


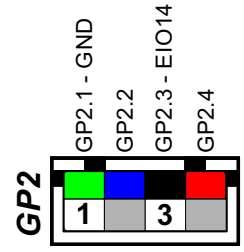
EIO-13



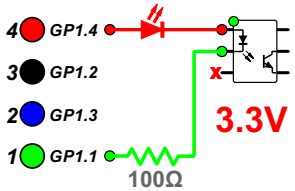


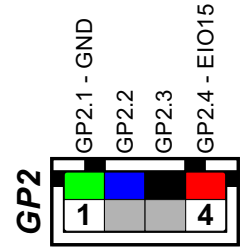
EIO-14

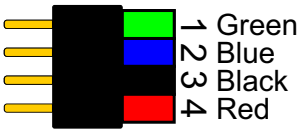




EIO-15







DOBOT Magician V2

Digital Signal Guide

Caution: NEVER wire anything to the Dobot Magician while it has power on. **ALWAYS** shutdown the Dobot before making connections or damage to the robot could occur.

WARNING

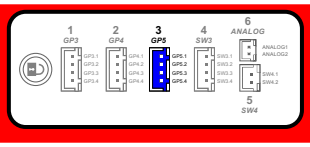
Power OFF

WARNING

Power OFF

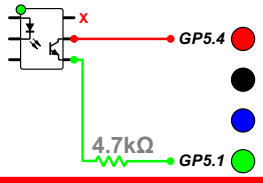
INPUTS

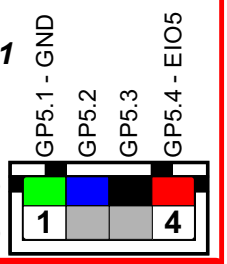
GP5



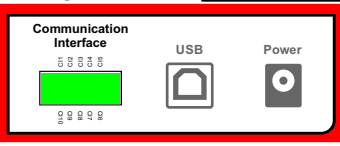
EIO-5

Signal ON = 0
Signal OFF = 1

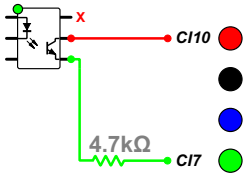


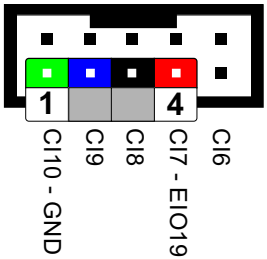


CI

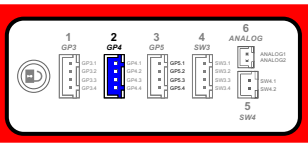


EIO-19



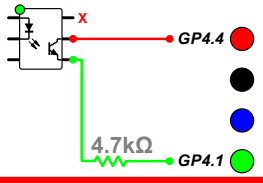


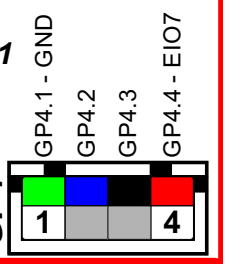
GP4



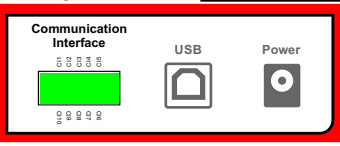
EIO-7

Signal ON = 0
Signal OFF = 1

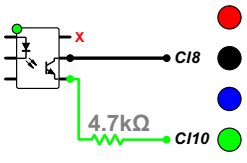


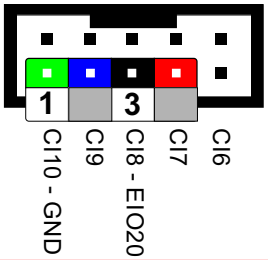


CI



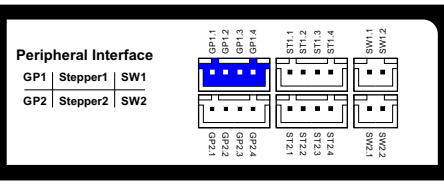
EIO-20



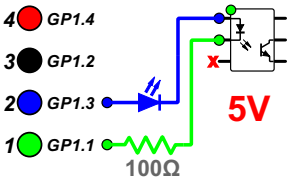


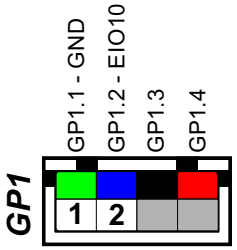
OUTPUTS

GP1

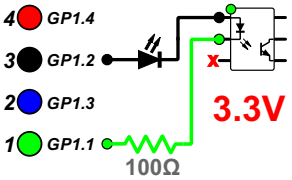


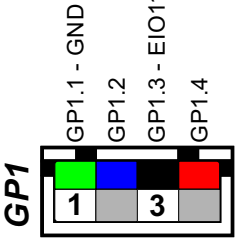
EIO-10



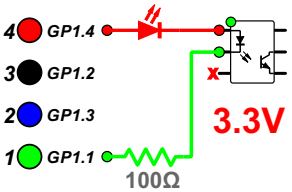


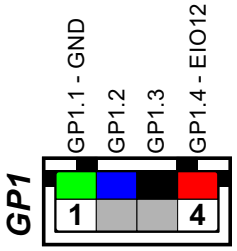
EIO-11



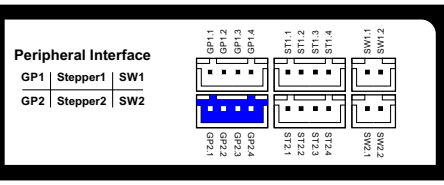


EIO-12

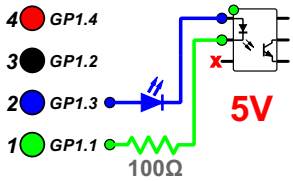


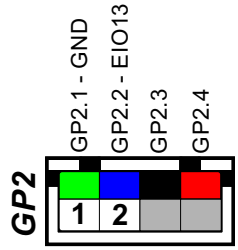


GP2

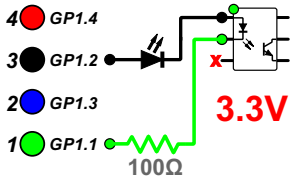


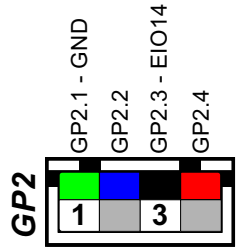
EIO-13



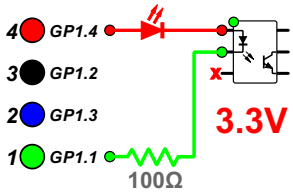


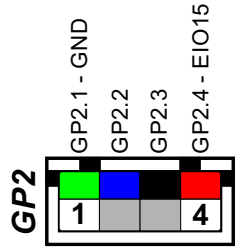
EIO-14

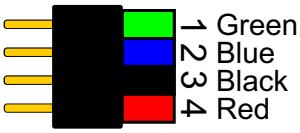




EIO-15







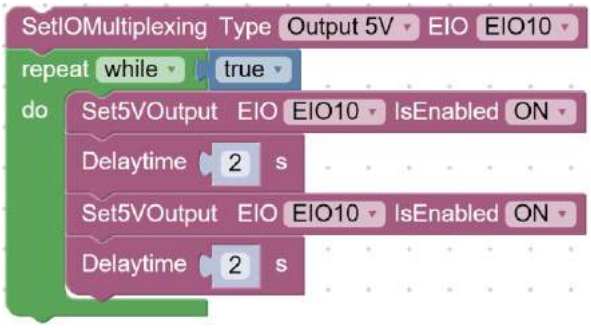
DOBOT Magician

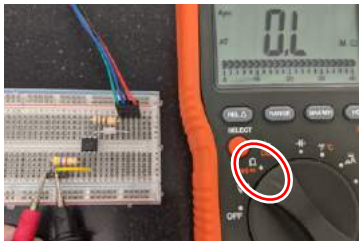
Signal Guide

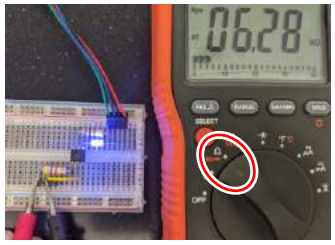
Caution: NEVER wire anything to the Dobot Magician while it has power on. **ALWAYS** shutdown the Dobot before making connections or damage to the robot could occur.

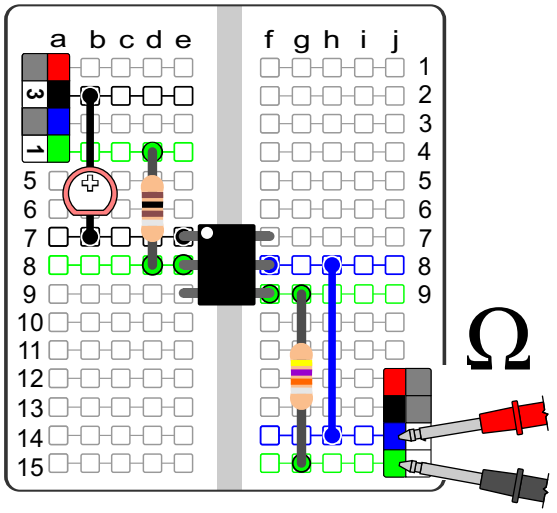
Check Optical Isolator with Volt Meter

WHY is the robot not receiving a signal from the handshake module? Wiring and programing are often the culprit. It could also be a damaged optical isolator. The LED on the INPUT side of the optical isolator helps us determine if the signal is actually being sent by the robot. Unfortunately, the OUTPUT side of the optical isolator will not allow us to use an LED as an indicator. We can however use a volt meter to measure the RESISTANCE across the optical Isolator to determine when a signal is ON or OFF. Use the Blockly program shown below to cycle the signal ON and OFF to see if the signal is getting across the optical isolator. When no signal is present, the voltmeter should read “O.L” or “---”. When a signal is present it should read near the resistance value of the resistor used.





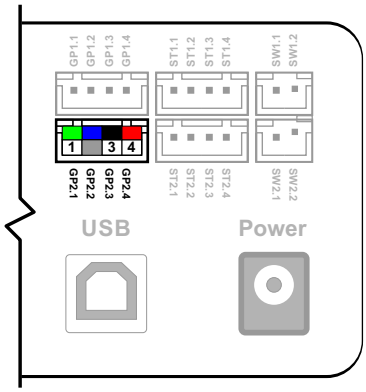


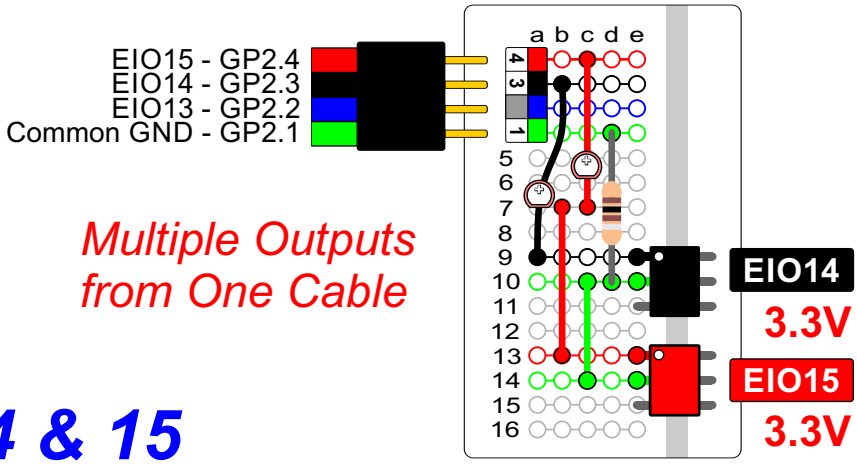


Optical Isolator OFF = 0.L or ---
Optical Isolator ON = 4.9+ kΩ

How to Use a Common GND

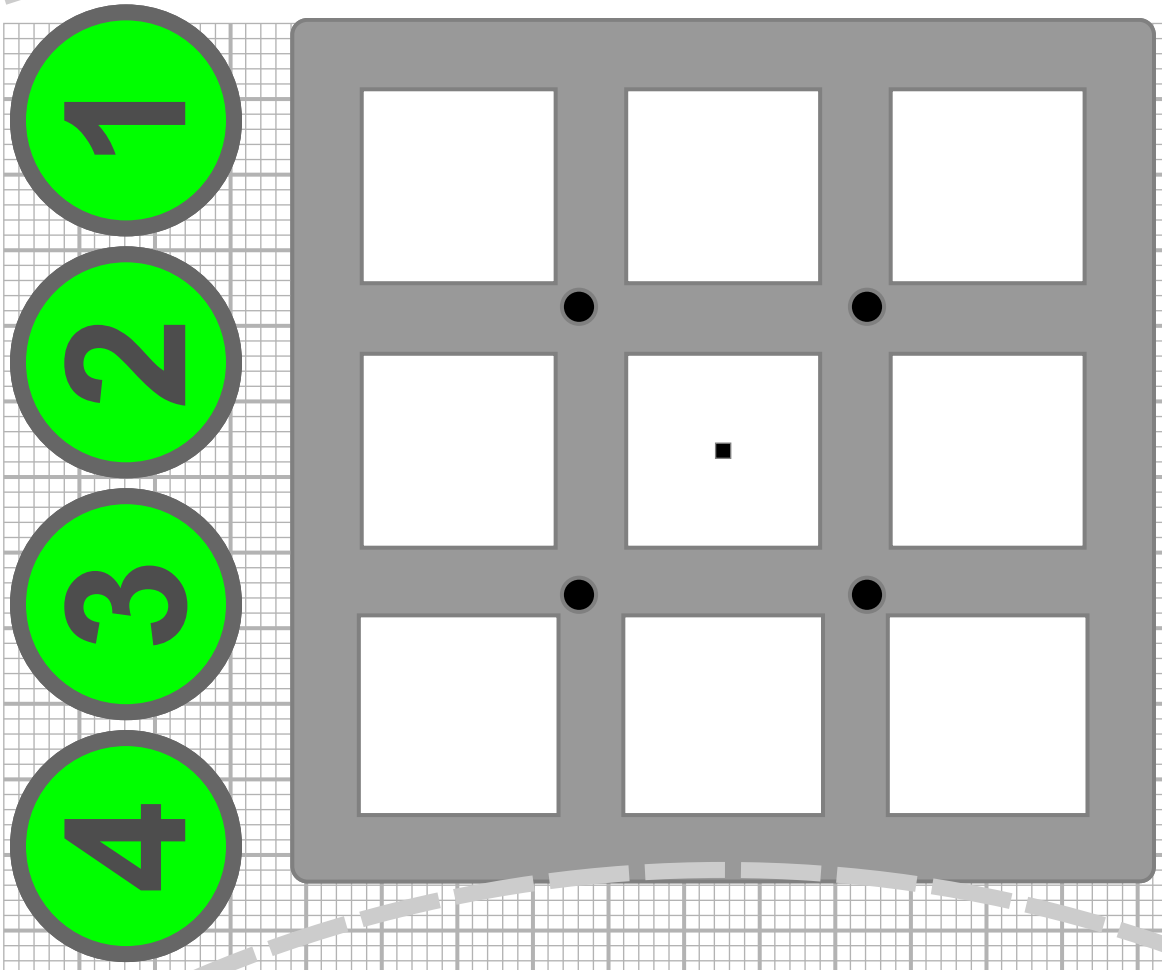
Can the ground (GND) be shared between multiple outputs? Yes, the ground can split among multiple optical isolators as a “shared” or common ground signal. The optical isolators then only need the signal wire to turn ON and OFF. See the illustration below.





Multiple Outputs from One Cable

GP2 - EIO14 & 15


1
2
3
4