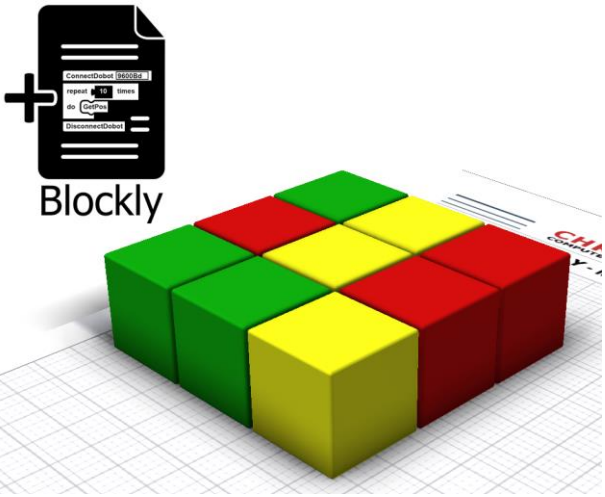
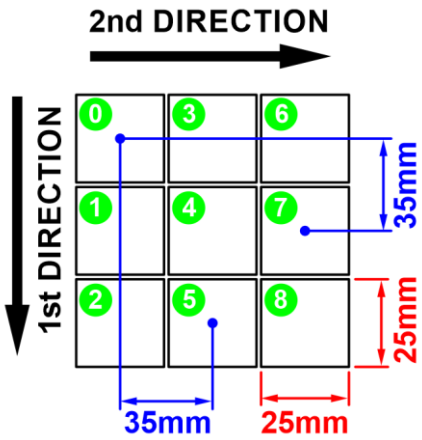
4 Blockly - Developing a Cube Matrix

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

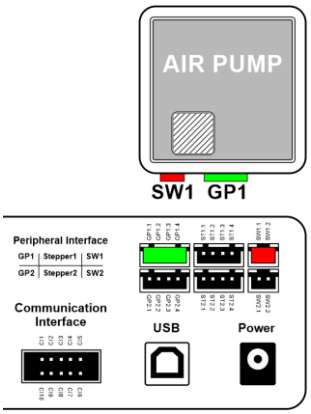
Essential questions answered in this activity include:

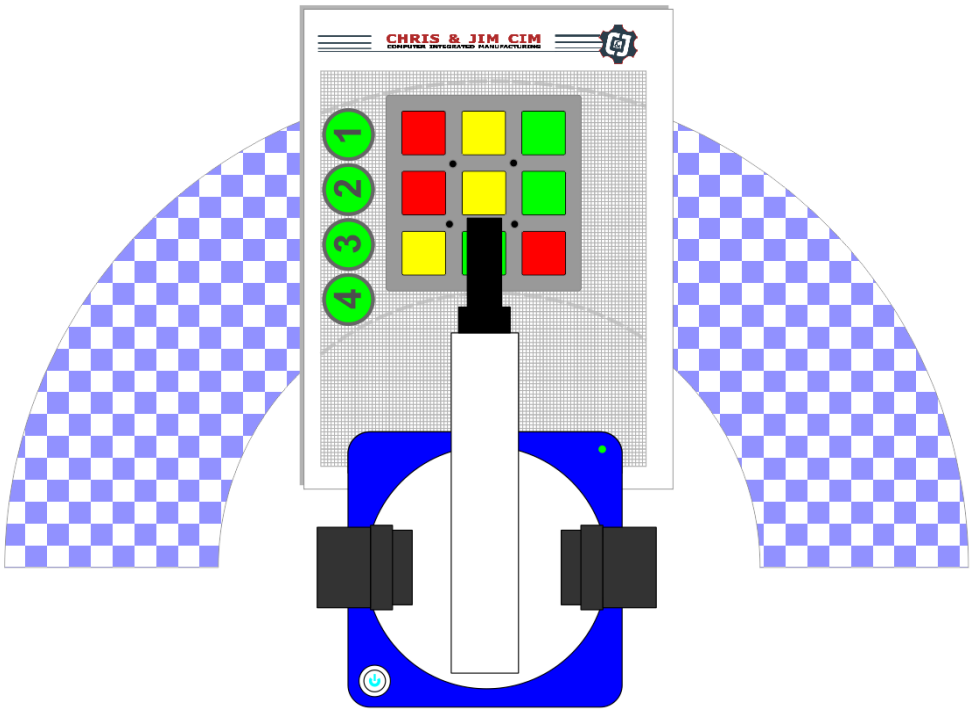
- Why are variables used?
- How do I use variables in blockly?
- How can I use math to palletize?
- What are some of the blockly commands needed to do this?
- Why is palletization and de-palletization important in industry?

PROCEDURE

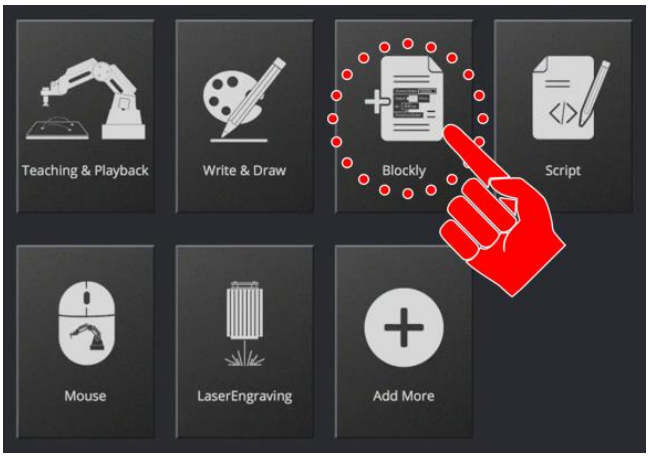
Caution: NEVER wire anything to the Dobot Magician while it has power on. ALWAYS turn it off before making connections or damage to the robot could occur. Be sure to ask your instructor if you have any questions.

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

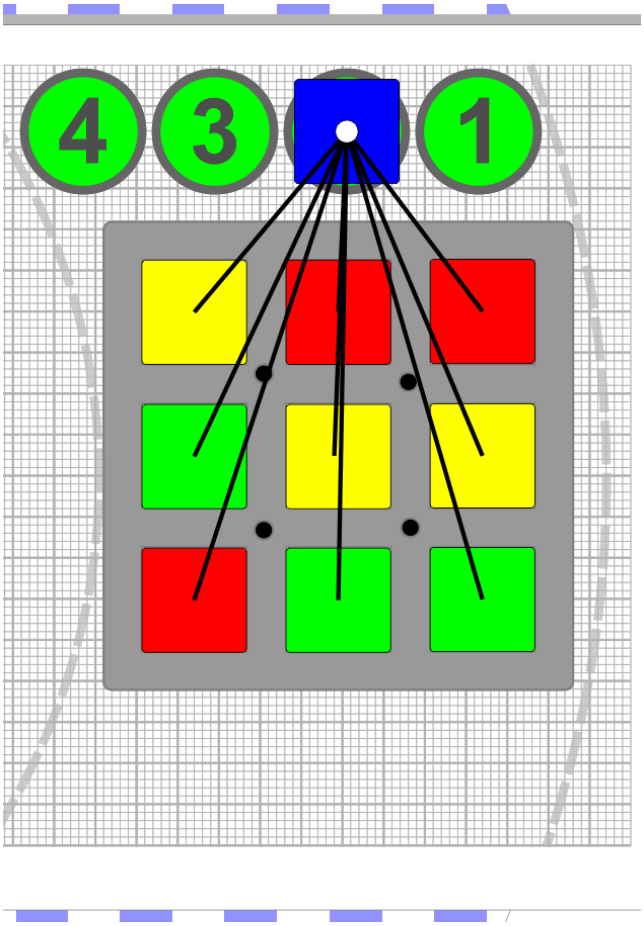
When you re-open this program check that the name of the file on top matches the code in the file, if it does not, you may end up overwriting another program

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.

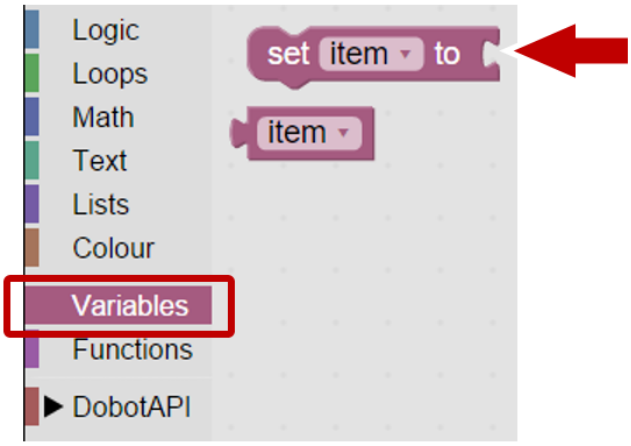
We can use these common factors to simplify our operations and our code. We will do this with user defined *variables* and *functions*.

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.

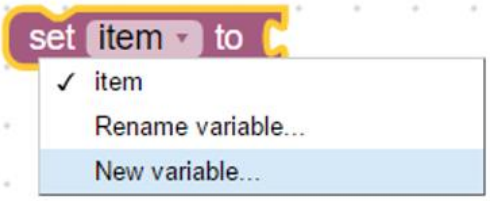
We will start our program by defining a set of variables. *Variables* are a changeable quantity in a program that can be represented by a word or a letter. Variables can be assigned, changed, or referenced throughout a program.



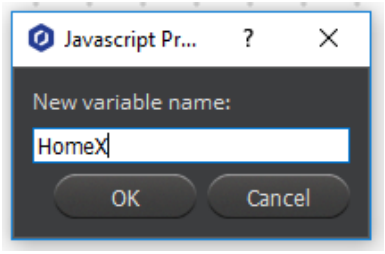
List of variables to define

- 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

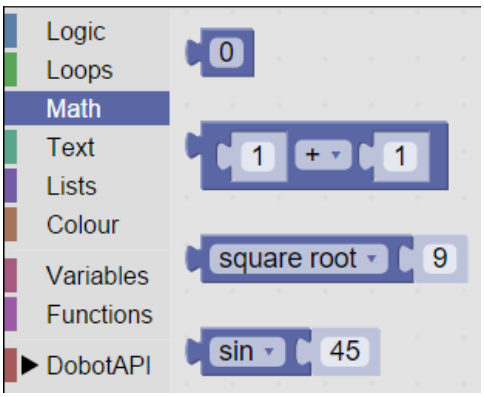
Drag out a ***set item to variable*** from the *Variables* section.

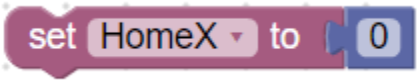


Click on the "item" portion of the variable tool and select *New variable*. This will be our HomeX value.



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.

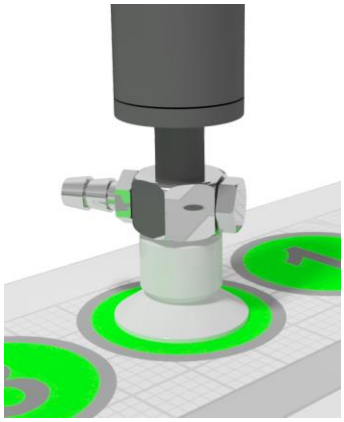
Expand the Operation Panel

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

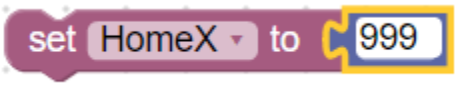
- ***Use Cube '0', from the diagram on page 1, as the X and Y location for the Pick Location.***
- ***Round all values to the nearest whole number***
- ***Align the center of the vacuum gripper to the center of the cube***
- ***Use the location of dip dank 2 for the common drop off point***
- ***Ensure 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.***



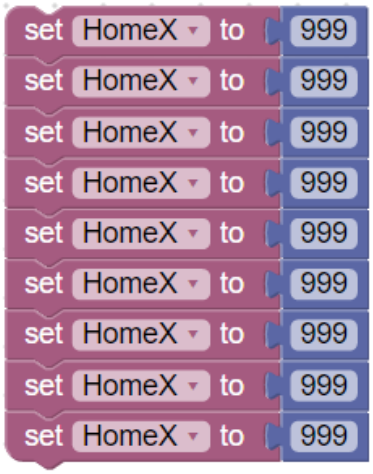
HomeX	
HomeY	
HomeZ	
PickX	
PickY	
PlaceX	
PlaceY	
CommonZAt	
CommonZAbove	

Part 1: Setting Variables

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



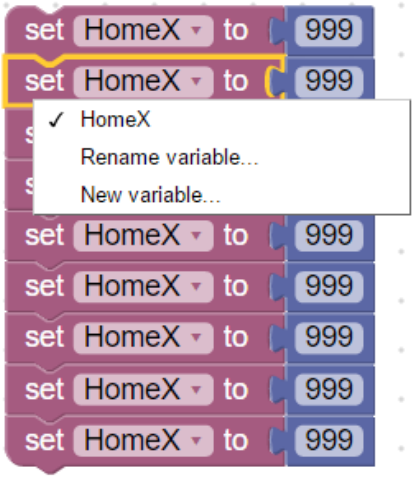
Duplicate the HomeX block until you have 9 variables total.



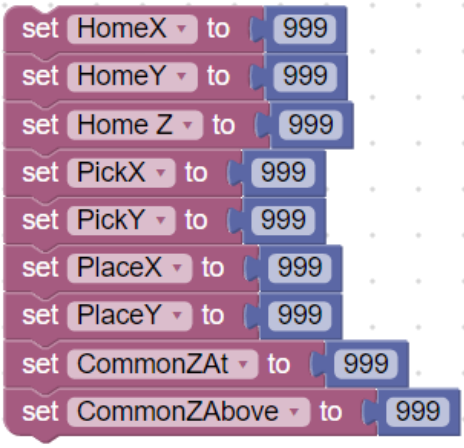
Left Click on the next variable name and select New Variable.

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

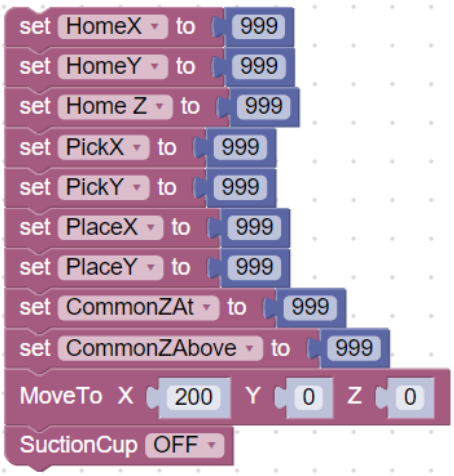
If you select Rename Variable, it will only Rename that value and every where it has been used in the program



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



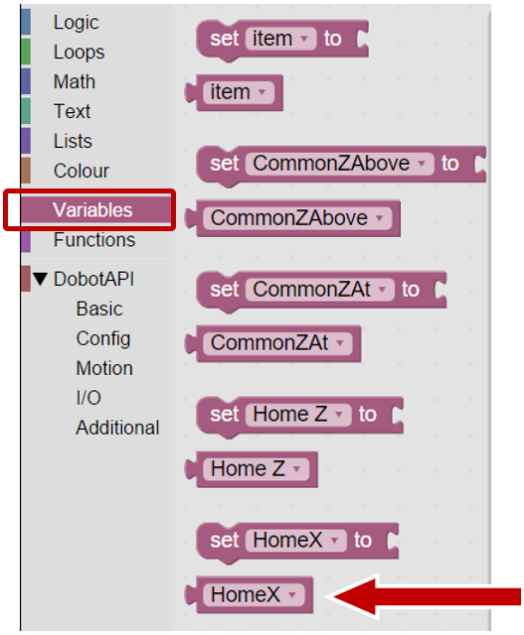
The first actual movement is for the robot to always go home and turn off the vacuum. Please drag and add these blocks to the program.



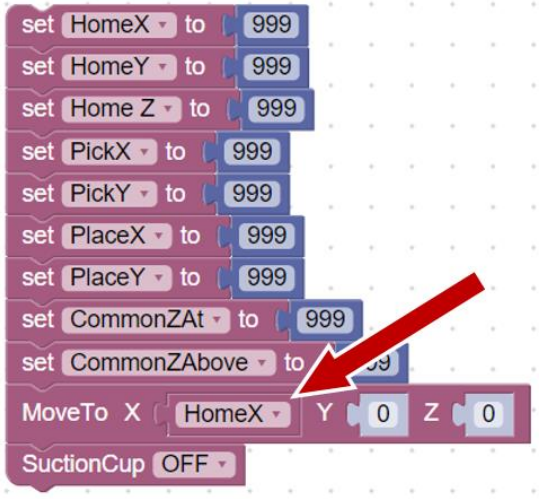
Remember that these tools come from the DobotAPI/Motion section.

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

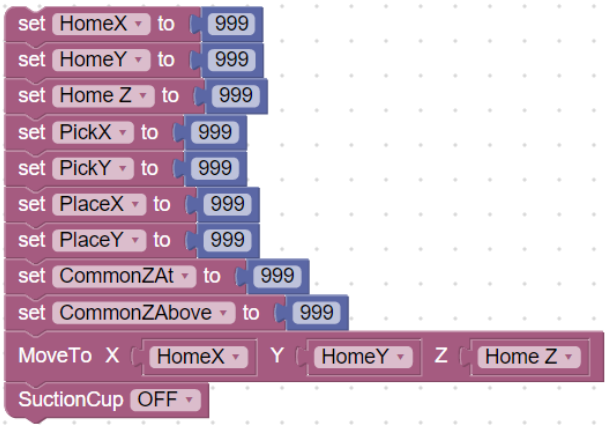
From the Variable section, you can now see that all of the variables that we have created are now options to use.



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.

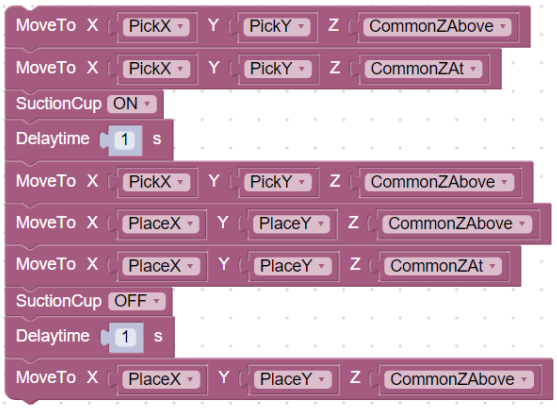


Assigning variables really helps when XYZ coordinates in a program are going to have to change multiple times or is going to be used as a template for multiple programs. The can be altered once at the beginning of the program and that will update everywhere they were used in the program. It also makes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

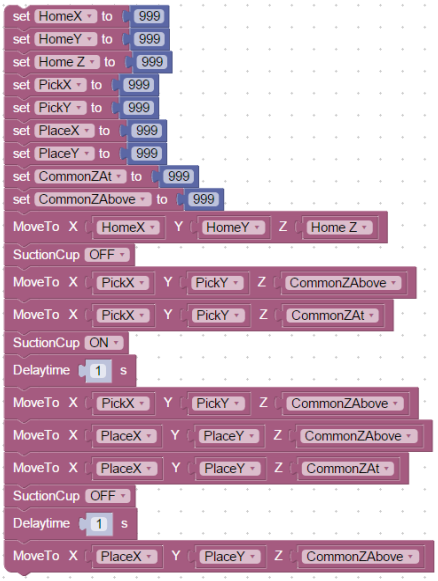
We will now add all the steps needed to go get the first block '0' from the assigned pick location and move it to the place location.

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?

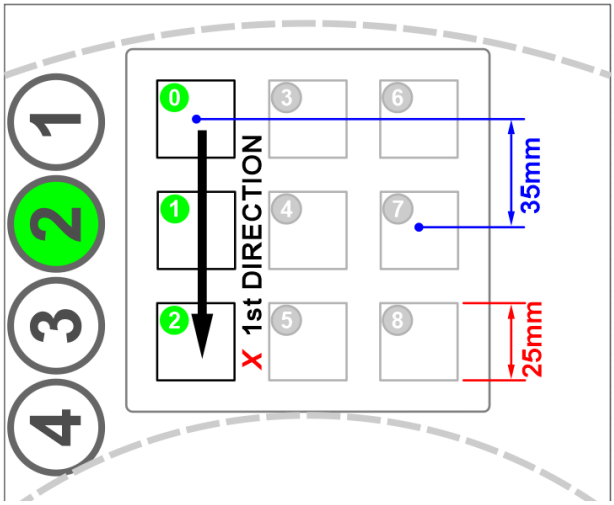
ALL NEW - THE TRICKY PART!!!

Part 2: Adjusting Variables

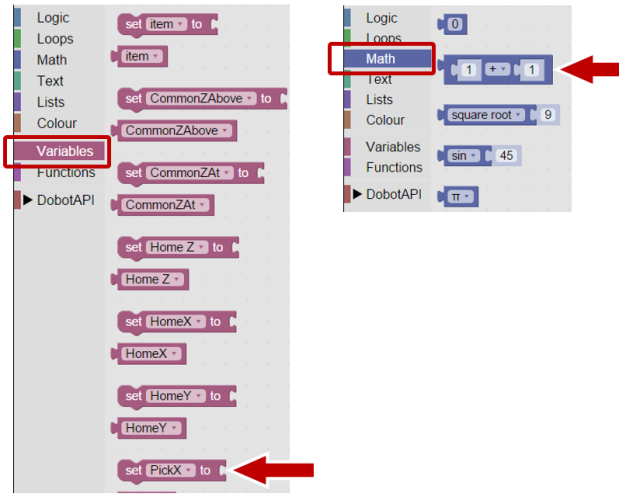
Now it's time for the second cube.

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

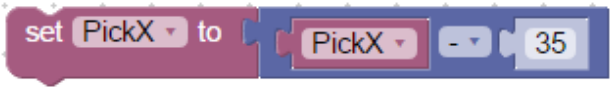
From the diagram shown, the blocks are 35mm apart and that the block is moving *toward* the robot. Moving toward the robot is shown mathematically by subtracting 35mm from the current X position value for block number 2 and again for block number 3.



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.

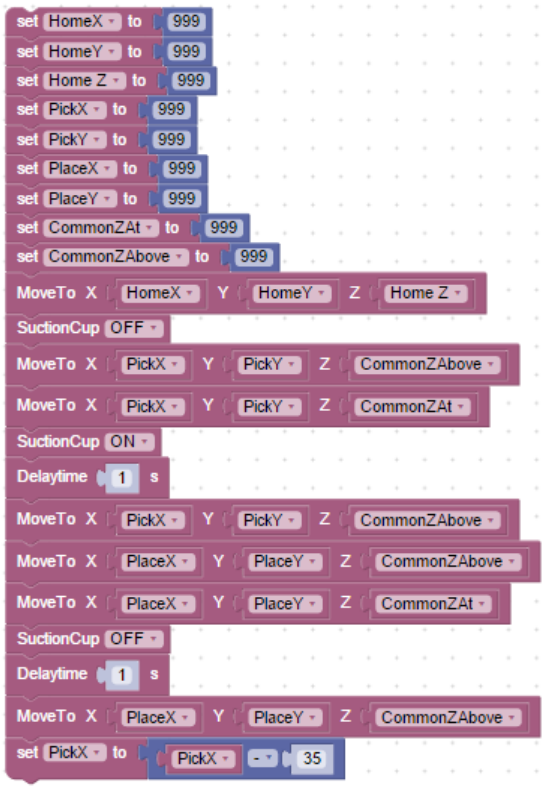


Current X value equals itself plus a constant of 35

Mathematically: PickX = PickX - 35

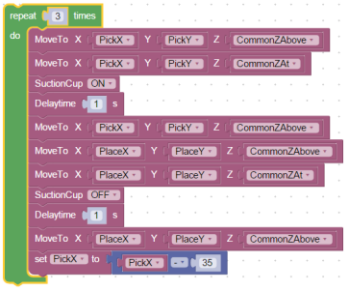
Grab the PickX Variable from the Variables section or duplicate it from the main program

Add this block line to the bottom of your program

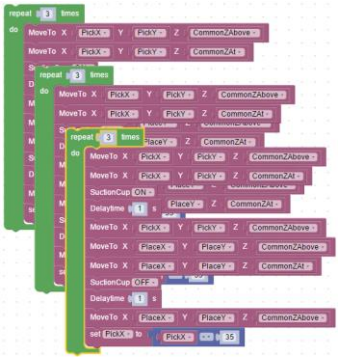


Now that the PickX value has been adjusted, repeat the entire process of picking up the second and third cubes and dropping them off at the place location.

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.

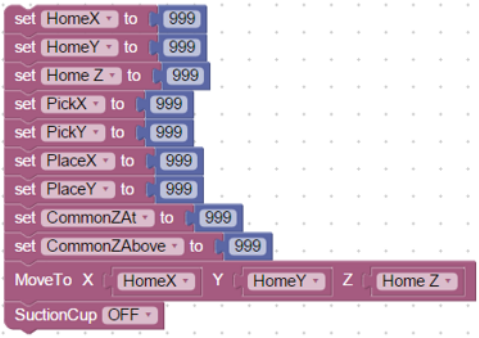


If a loop is deleted before pulling the group of code out of the loop, it will delete the loop and its content.

Section 1

Assign Variables

Home Robot

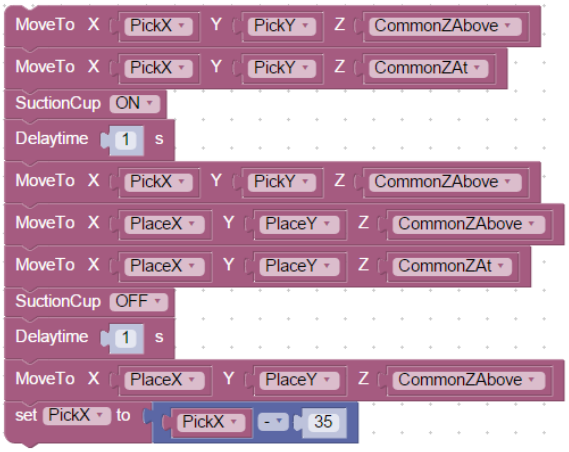


Section 2

Pick up Block

Drop off Block

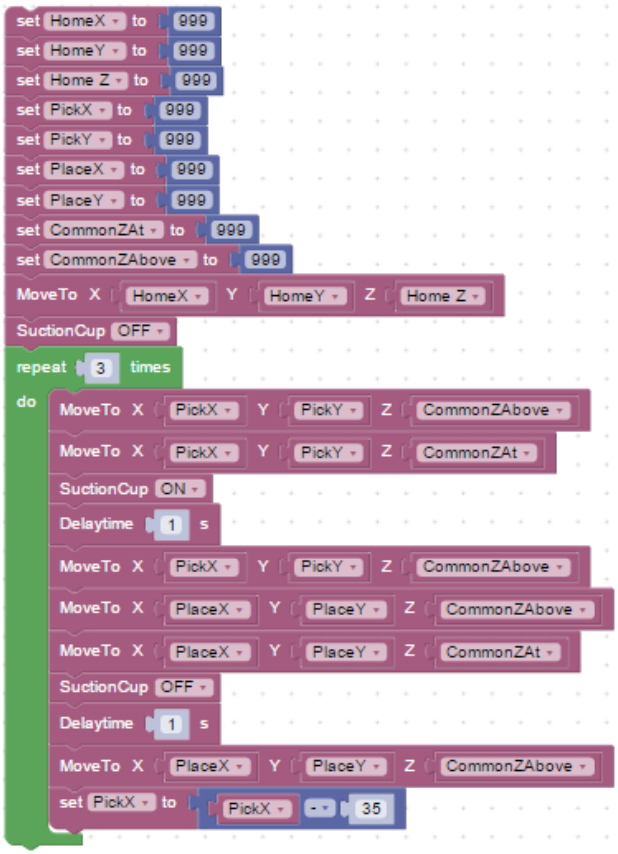
Reassign Variable



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

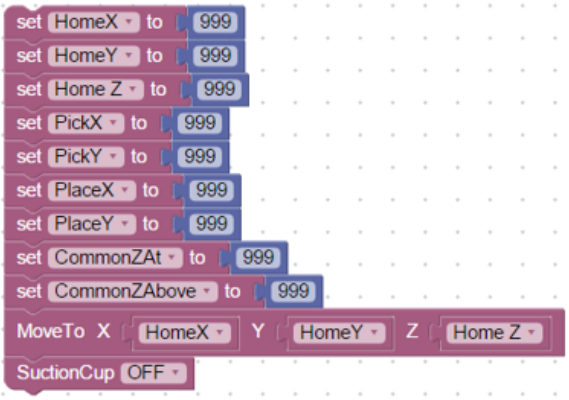
Break your code up into 3 separate parts:

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

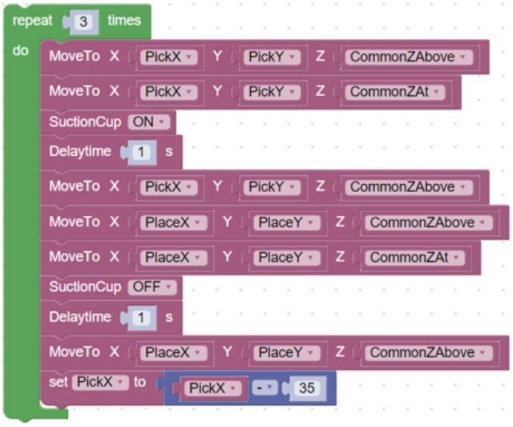


Section 2

Pick up Block

Drop off Block

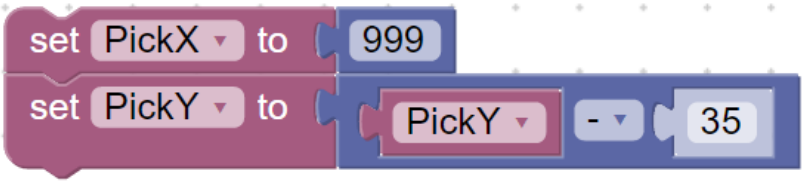
Reassign Variable



Section 3

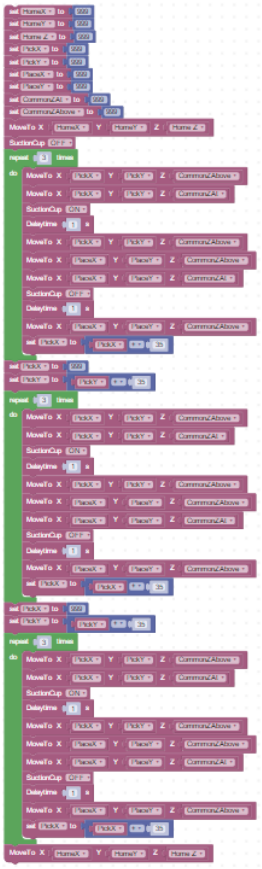
Reassign PickX to the Original Value

Assign PickY to PickY +/- 35



Rather than create a really long code that looks like:

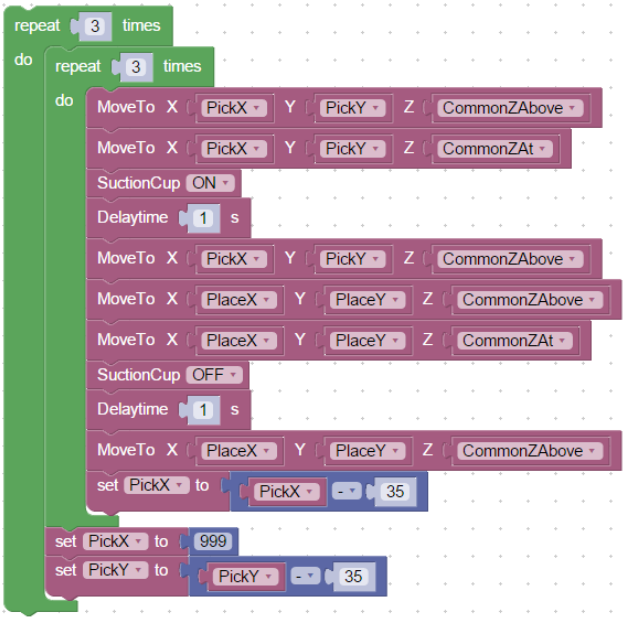
Section 1
Section 2
Section 3
Section 2
Section 3
Section 2
Section 3
Send Robot to Home



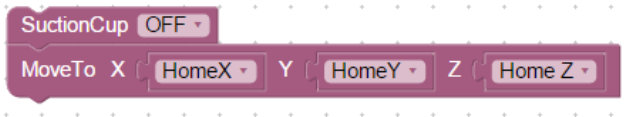
Look to the right to see how long it actually makes our program. This would make it really hard to work with in the programming window as a lot of scrolling would have to be done in order to make any edits.

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.

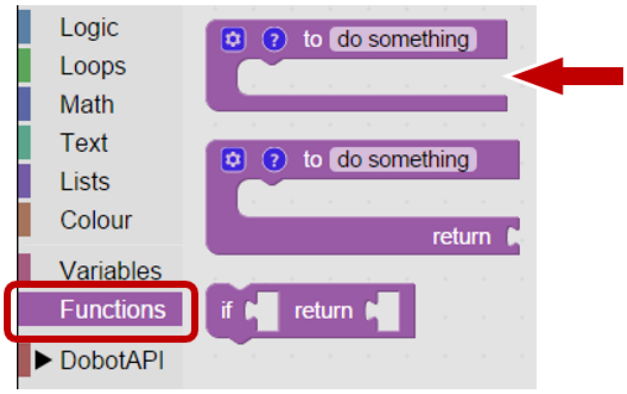
Part 4: Adding Functions

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

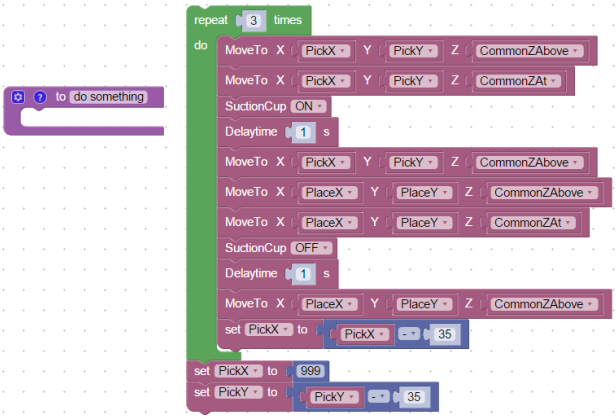
The next step is to simplify it even further with a block of code called a function. A *function* is a named section of a program that performs a task. It can also be considered a procedure or a routine and greatly simplifies otherwise complicated programs.

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

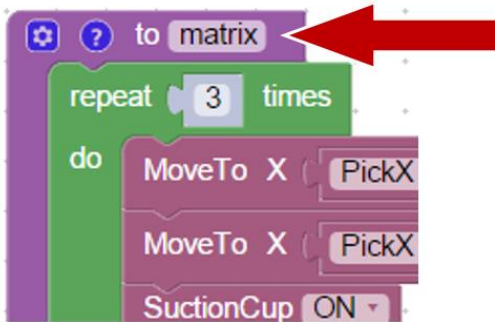
Drag over a ***Function*** block with no Output .



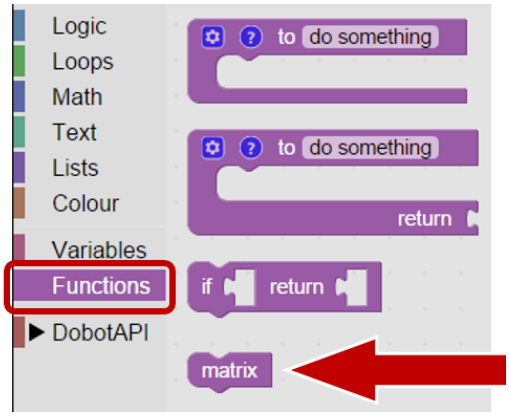
We can pull the ENTIRE matrix looping process into the Function



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



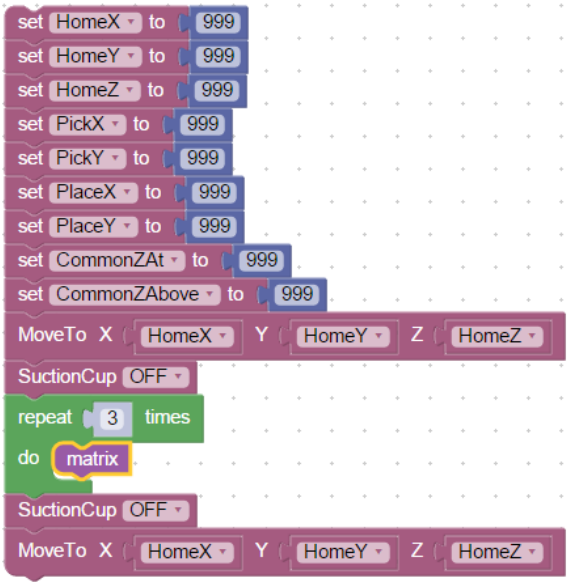
The Function that was just created is now available to be used in the program as many times and wherever needed.

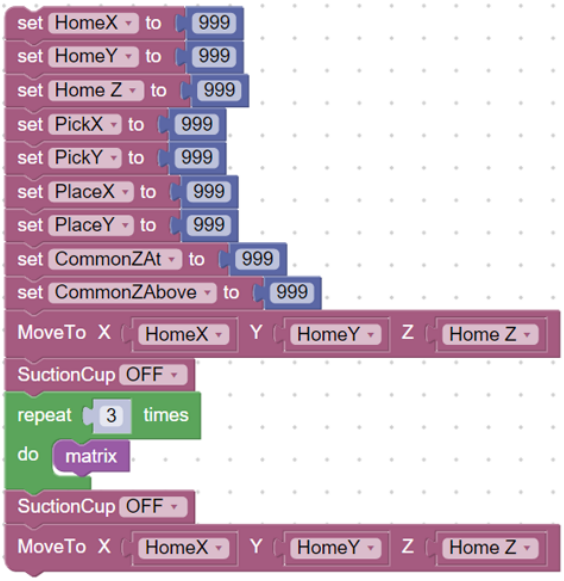


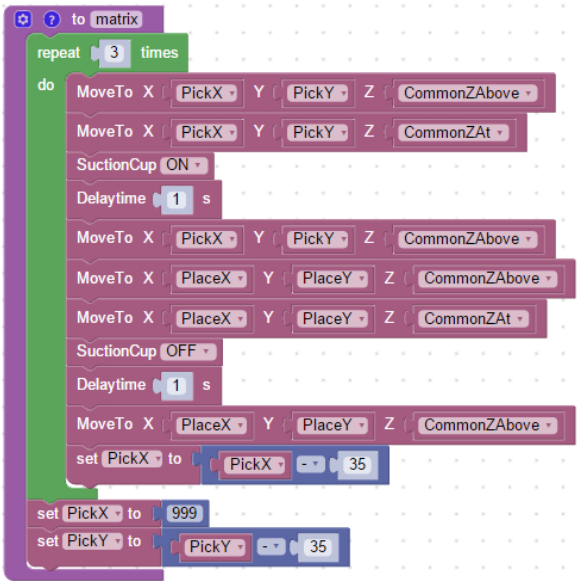
Drag the new *Function* into the Repeat Loop.

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.*

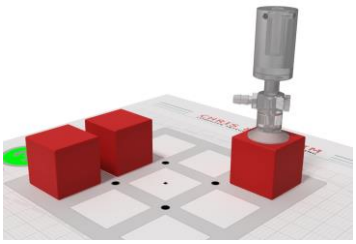
3. *Explain what would have to be done to palletize two layers using bullet points or a step by step list below.*

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.