# Introduction to Computer Science with MakeCode for Minecraft

**Lesson 8: Arrays**

| | |
|---|---|
| In computer science, an array is a series of places to store things, like a list of items, a row of mailboxes, or a train of container boxes. |  |

We learned that *variables* are used to store information. Specifically, variables help the computer find and retrieve information that is stored somewhere in its memory. We can give variables names so that when we refer to them by name, the computer can look up that variable by its name, and then find out where in memory that information is stored, how much space that information is expected to take, and what type of information it is.

When you have a lot of variables to deal with, or when you don't know ahead of time how many variables you are going to need, it's more convenient to use an array. An array is a way of organizing several variables in a series, so that you can go right down the line and access information in each of them.

| | |
|---|---|
| The location of a particular item in the array is known as its *index*. The first item in an array always has index 0. The next one has index 1, and so on.<br><br>The length of an array is the same as the number of items in the array, and that number is always one more than the index of the last element (because the indexes start at 0.)<br><br>There are 7 mailboxes with indices from 0 to 6. |  |

An array of pigs in Minecraft – notice that the index of the fourth pig is 3 because the indexes start at zero.

In MakeCode, you can store different types of things in an array:

- Numbers
- Text
- Chickens
- Positions
- Blocks
- Zombies

You can add objects to an array, remove them, and count the number of things in the array at any given time. You can even reverse their order in the array with a single command. Arrays are a very convenient and powerful way to work with lots of items at once.

Sorting Array Values

Once you start saving lots of different values in an array you will probably want to have some way to sort those values. In computer science, there are certain common strategies, or algorithms, for sorting a collection of values. And understanding how those different sorting algorithms work is an important part of computer science. As students go on to further study they will learn other algorithms, as well as their relative efficiency. Some of the different sorting algorithms include:

- Selection sort
- Insertion sort
- Quick sort
- Merge sort
- Bubble sort

- Shell sort

Here is a good video that visually displays a number of different types of array sorting algorithms: https://www.youtube.com/watch?v=kPRA0W1kECg

In this chapter, we will create an instant zoo, and fill it with animals from your array. We will also create a warp belt that allows you to save locations on your map and instantly teleport to them. Finally, we will challenge you to create an independent project of your own that uses arrays of colored wool, animals and monsters, decorated sandstone, or anything else you can think of!

**Unplugged activity: Discussion of Arrays in real life**

Questions to ask students:

- Ask your students if any of them collect anything. What is it? *Comic books, game/sports cards, coins, action figures, books, etc.*
- How big is the collection?
- How is it organized?
- Are the items sorted in any way?
- How would you go about finding a particular item in the collection?

In the discussion, see if you can explore the following array vocabulary words in the context of your students' personal collections:

- Array Length: the total number of items in the collection
- Sort: Items in the collection that are ordered by a particular attribute (for example, date, price, name, color, etc.)
- Index: A unique address or location in the collection (for example, page number in an album, shelf on a bookcase, etc.)
- Type: The type of item being stored in the collection (for example, DC Comics, $1 coins, Pokemon cards, etc.)

**Unplugged activity: Bubble Sort**

In this activity, you will demonstrate one type of sorting method on your own students. This is an unplugged activity, so your students will be standing at the front of the room. If you or your students are curious to see what these different sorts look like in code, we have included a MakeCode version of the bubble sort algorithm in this lesson for you to explore if you choose.

Materials needed:

- 10 sheets of paper numbered 1–10 in big letters

Have ten students volunteer to stand up at the front of the classroom. Ask another student to volunteer to be the Sorter.

Mix up the order of the papers and give each student a piece of paper with a number on it. They should hold the paper facing outward so their number is visible to the rest of the class. Each of these students represents a value in the array.

Initial Sort

Ask the Sorter to place the students in order by directing them to move, one at a time, to the proper place. Once the students are sorted, ask students the following questions:

- How did she sort you into the right order?
- Did you see a pattern?
- What did she do?

Try to get students to be as precise as possible in explaining their thinking. Sometimes it helps to put the steps on the board, in an algorithm for example:

- First, she went to the first student in the line, then put him in the right place.
- Then she went to each of the next students and put them in the right place.
- Or, she went to the #1 student first, then the #2 student, etc.

Ask for clarification when necessary: What does it mean when you say "put them in the right place"?

For example, to put someone in the right place means:

- Bring the person to the front of the line and then compare that person's number with the first person's number
- If it's larger, then move that person to the right
- Keep doing this as long as the person's number is larger than the person on the right

This is a good point to bring up the fact that humans are much smarter than computers. What comes naturally to us (pattern matching and sorting), is much more difficult for computers to understand – and that's why we have to be very specific and exact in our instructions to a computer for how to sort items in an array.

The Bubble Sort

One basic type of sorting algorithm is called a Bubble Sort, so named because the larger values tend to "bubble up" toward the end of the array.

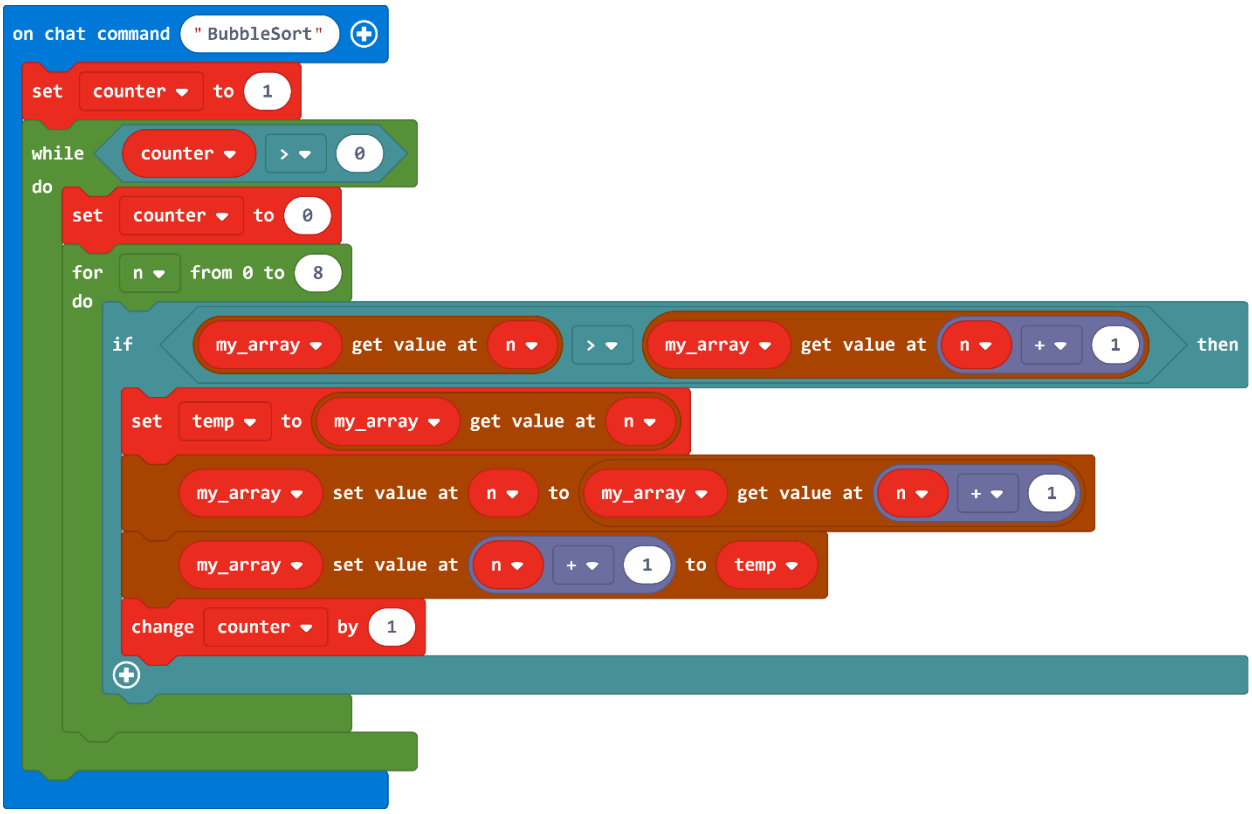
How to Bubble Sort:

- Compare the first two students.
- If the student on the right is smaller than the student on the left, they should swap places.
- Then compare the second and third students.
- If the student on the right is smaller than the student on the left, they should swap places.
- When you reach the end, start over at the beginning again.
- Continue in this way until you make it through the entire row of students without swapping anybody.

In Pseudocode:

1. Create a variable called counter
2. Set the counter to zero
3. Go through the entire array
4. If the value you are considering is greater than the value to its right:
    a. Swap them
    b. Add one to counter
5. Repeat steps 2 through 4 as long as counter is greater than zero

In Blocks:




In JavaScript:

```
let my_array: number[] = []
let temp = 0
let counter = 0
player.onChat("BubbleSort", function () {
    counter = 1
    while (counter > 0) {
        counter = 0
        for (let n = 0; n <= 8; n++) {
            if (my_array[n] > my_array[n + 1]) {
                temp = my_array[n]
                my_array[n] = my_array[n + 1]
                my_array[n + 1] = temp
                counter += 1
            }
        }
```

```
    }
})
```

**Activity: We Built a Zoo**

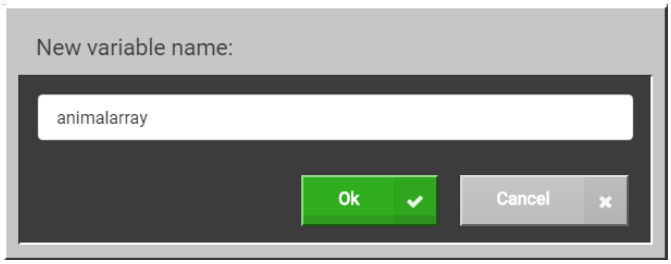
You can store animals in an array and spawn them wherever you like. We'll use this capability to build a fenced-in animal pen and create an instant zoo anytime we want.  When this project starts up, it will create an array and fill it with animals of your choice.

This project will feature two commands:

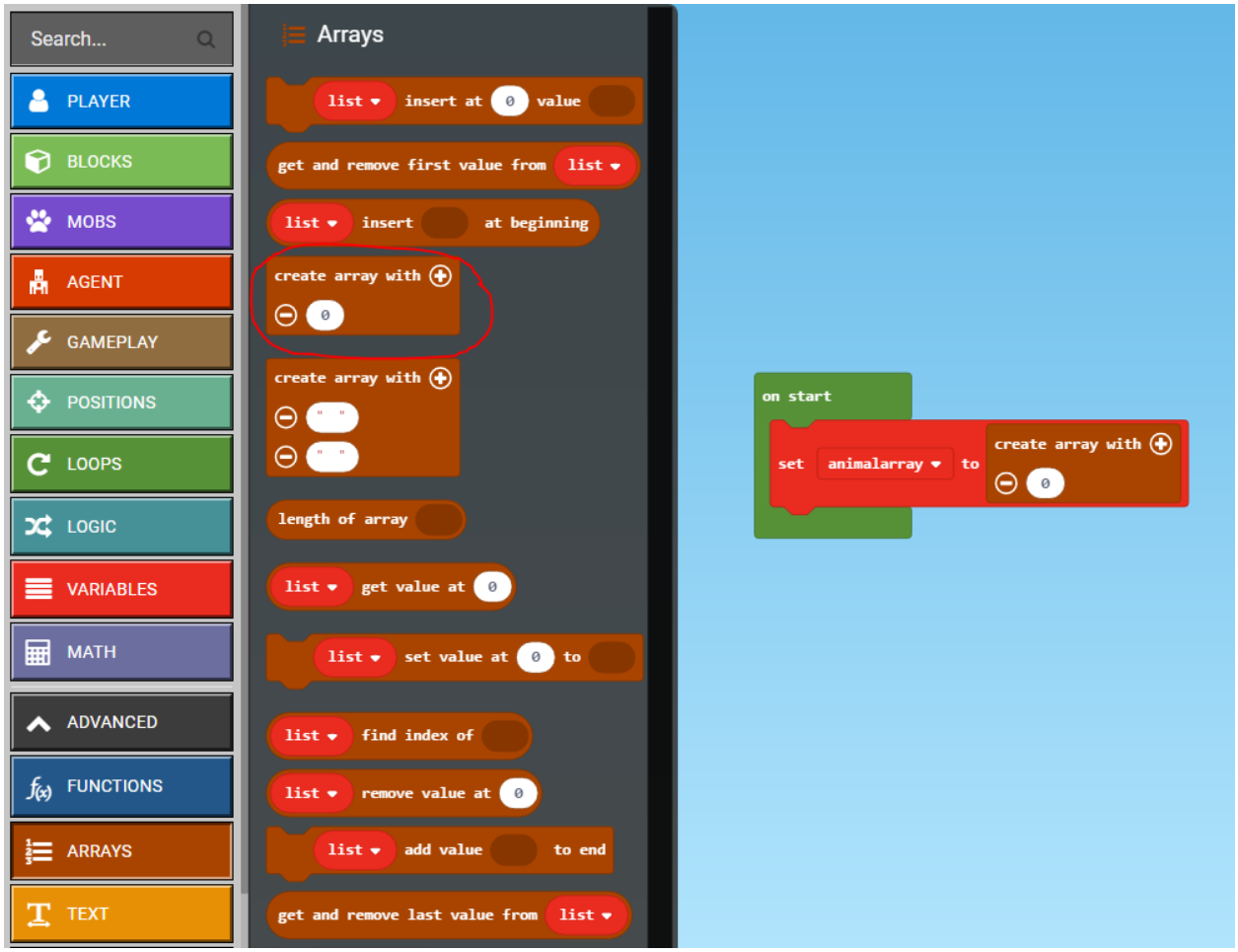
- "pen": Call this first to command the Builder to create a fenced-in pen around your position, so the animals don't escape.
- "zoo": At this command, MakeCode will go through your array and spawn two animals of each type, within the pen.

Steps:

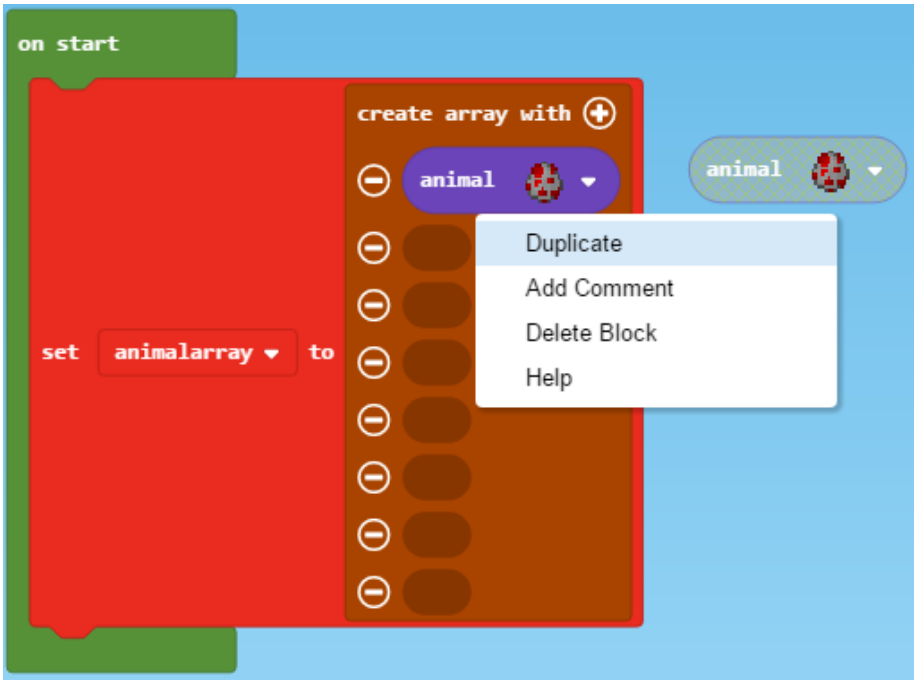
1. Create a new MakeCode project called Zoo.
2. In the Loops Drawer, there is an On start block that will run its commands once, as soon as the project starts up. Drag that block into the coding Workspace.
3. From the Variables Toolbox drawer, click the 'Make a Variable' button
4. Name this variable "animalarray", and click Ok




5. From the Variables Toolbox drawer, drag a Set block into the On start block
6. Using the drop-down menu in the Set block, select the animalarray variable
7. Click on the Advanced tab in the Toolbox to display the Arrays Toolbox drawer
8. From the Arrays Toolbox drawer, drag a Create array with block and drop it into the Set block

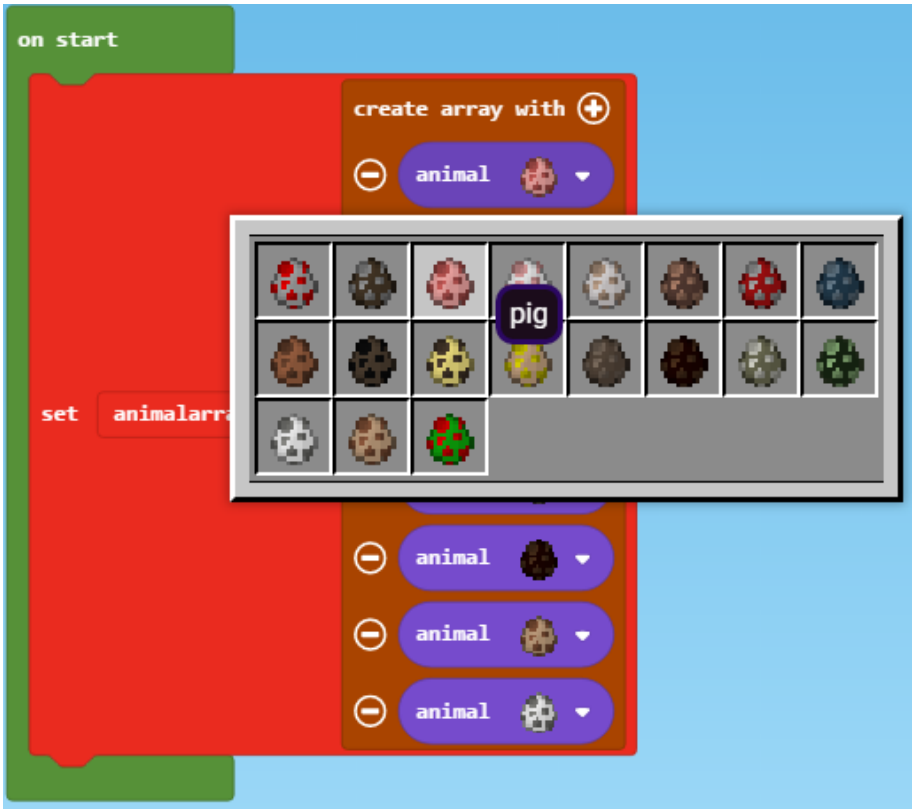



9. Click the Plus (+) sign on the Create array with block to add 7 more slots in your array so the total length of your array is 8
10. From the Mobs Toolbox drawer, drag an Animal block into the first slot of the Create array with block, replacing the number 0
11. Populate the rest of your array with Animal blocks. Note – you can right-click on the Animal block and select Duplicate to make copies of this block, instead of dragging more blocks from the Toolbox.




Create a zoo with 8 different types of animals. Be aware that certain animals will eat other animals! For example, ocelots and chickens don't get along very well. Think about what kind of zoo you want, and plan accordingly.

12. Using the drop-down menus in the Animal blocks, select different types of animals in your array

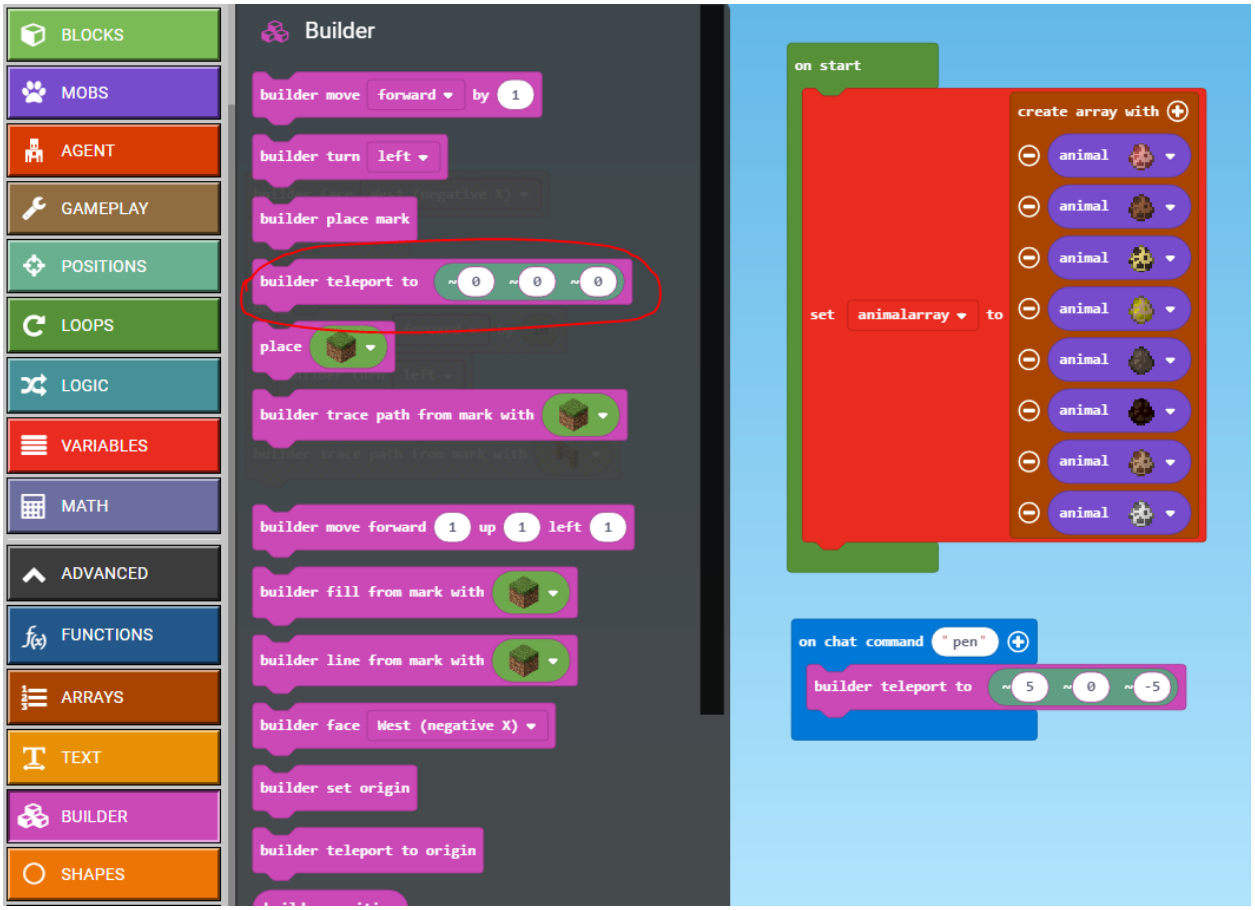



Now that we have our Animal Array set up, let's work on creating a fenced-in enclosure for our zoo. We will use the Builder blocks for this. The Builder is like an invisible cursor in the game that can place blocks along a path very quickly. We'll direct the Builder to go to a point in the southeast corner, and create a mark, which is an invisible point of reference. Then we'll give it a series of commands to move in a square, then tell it to trace its path from the mark with fences.

13. From the Player Toolbox drawer, drag an On chat command block to the Workspace
14. Rename the command to "pen"
15. Click on the Advanced tab in the Toolbox to display the Builder Toolbox drawer
16. From the Builder Toolbox drawer, drag the Builder teleport to block into the On chat command block

Recall that Minecraft coordinates are always specified in X, Y, Z coordinates where X is west to east and Z is north to south. We want the Builder to start in the northeast corner of the pen with relation to the player, so go ahead and change the coordinates to specify a location 5 blocks east and 5 blocks north of your position.

17. In the Builder teleport to block, change the position values to (~5, ~0, ~-5)




Let's make sure the Builder is facing the right way so that it draws the pen around you, and then place the build starting point mark.

18. From the Builder Toolbox drawer, drag a Builder face direction block under the Builder teleport to block
19. From the Builder Toolbox drawer, drag a Builder place mark block after the Builder face block

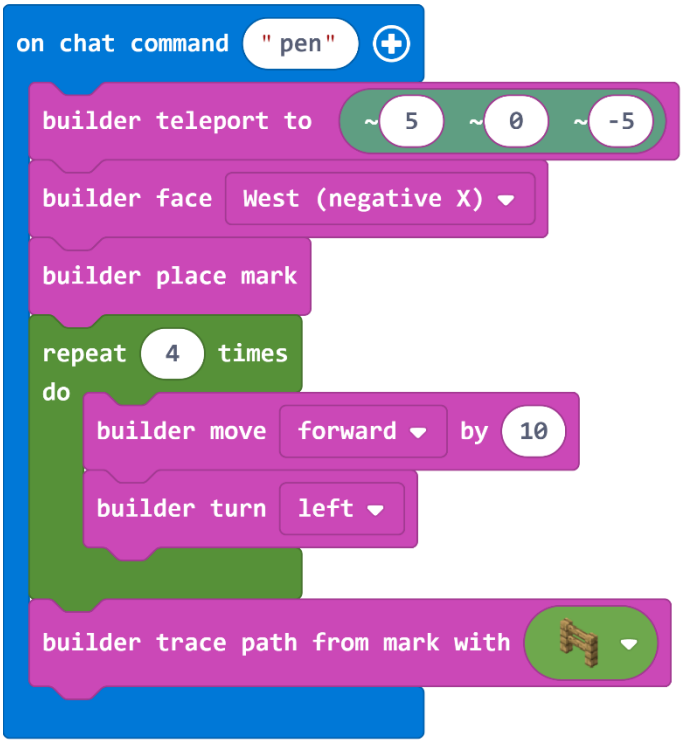
Now we'll simply have the Builder draw a square.

20. From the Loops Toolbox drawer, drag a Repeat loop block and place it after the Builder place mark block
21. From the Builder Toolbox drawer, drag a Builder move block into the Repeat loop
22. Type 10 into the Builder move block to specify that the length of the sides of our pen will be 10 blocks
23. From the Builder Toolbox drawer, drag a Builder turn block after the Builder move block in the Repeat loop

And the last step is to have the Builder place fences along the square.

24. From the Builder Toolbox drawer, drag a Builder trace path from mark block and place it after the Repeat loop
25. Using the drop-down menu in the Builder trace path from mark block, select a fence

Your code should look similar to this:



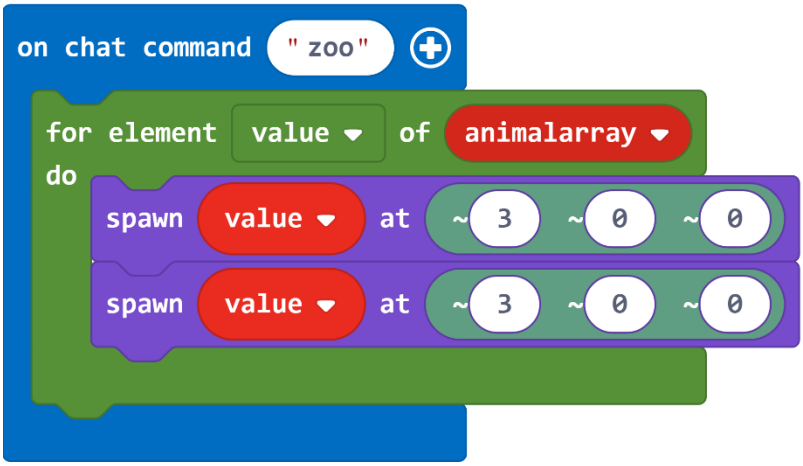

Now, open up a Flat World in the Minecraft game, and type “pen” in the chat window. You should see a pen being built all the way around you! For an extra challenge, you might try to get the Builder to add a fence gate.

Now comes the fun part. The array is loaded up with animals, the pen has been built, and now it’s time to let them loose! For this command, we will simply go through the entire array and for each animal in the array, we will spawn two of them a few blocks away from you but still within the pen.

26. From the Player Toolbox drawer, drag an On chat command block into the coding Workspace and rename it “zoo”
27. From the Loops Toolbox drawer, drag a For element block into the zoo On chat command block
28. In the For element block, use the drop-down menu in the second slot to select AnimalArray

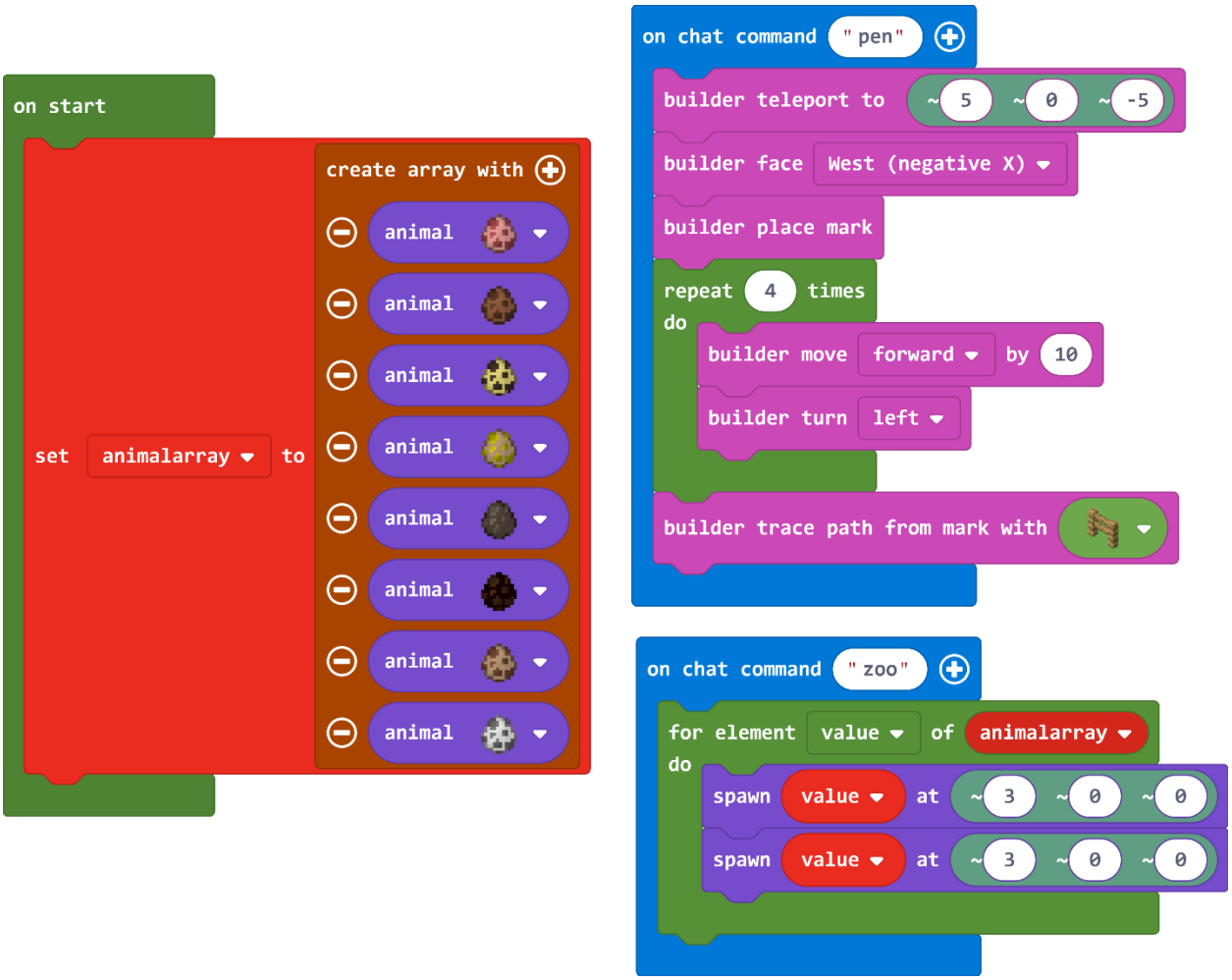

29. From the Mobs Toolbox drawer, drag a Spawn animal block and place it inside the For element loop
30. From the Variables Toolbox drawer, drag the value variable into the Spawn animal block, replacing the default chicken animal
31. In the Spawn animal block, change the spawn position to (~3, ~0, ~0), so the animals will spawn a few blocks away from the Player
32. To create pairs of animals, right-click on the Spawn animal block to Duplicate it

Your code should look similar to this:




Go back into your Minecraft world, and type the command “zoo” into the chat window, and watch the animals appear!

Full Zoo Program:




JavaScript:

```
let animalarray: number[] = []
player.onChat("pen", function () {
    builder.teleportTo(positions.create(5, 0, -5))
    builder.face(CompassDirection.West)
    builder.mark()
    for (let i = 0; i < 4; i++) {
        builder.move(SixDirection.Forward, 10)
        builder.turn(TurnDirection.Left)
    }
    builder.tracePath(blocks.block(Block.OakFence))
})
player.onChat("zoo", function () {
```

```
    for (let value of animalarray) {
        mobs.spawn(value, positions.create(3, 0, 0))
        mobs.spawn(value, positions.create(3, 0, 0))
    }
})
animalarray = [mobs.animal(AnimalMob.Pig),
mobs.animal(AnimalMob.Rabbit),
mobs.animal(AnimalMob.Ocelot),
mobs.animal(AnimalMob.Horse),
mobs.animal(AnimalMob.Donkey),
mobs.animal(AnimalMob.Mule),
mobs.animal(AnimalMob.Llama),
mobs.animal(AnimalMob.PolarBear)]
```

Shared Program: https://makecode.com/_2Jc55Xbjz1v8

**Activity: Warp Belt**

One of the most enjoyable activities when playing Minecraft is to build houses. Often, these houses are spread all over the world. Or, sometimes while exploring, you may make some awesome discoveries of hidden temples or villages in remote locations. It can be challenging to remember how to get back to all of those places. In this activity, we'll show you how to create a tool that you can use to teleport between all of those different locations.

We'll support four main commands in this program:

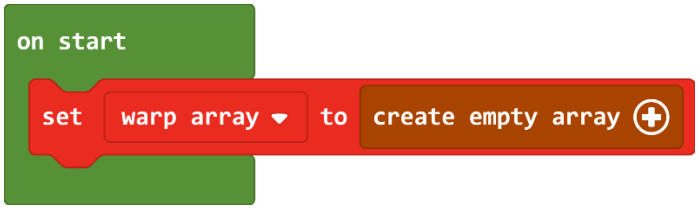
- **Delete**: This will create an empty array, effectively deleting your old one
- **Save**: This will save your current position to the next empty spot in the array
- **Warp**: This command, when entered with a number, will teleport the player to the position stored at that index in the array
- **List**: This command prints out all the positions in the array, with their index numbers

Steps:

1. Create a new MakeCode project called "Warp"
2. From the Loops Toolbox drawer, drag an On start block onto the coding Workspace
3. Open the Variables Toolbox drawer and click the 'Make a Variable' button

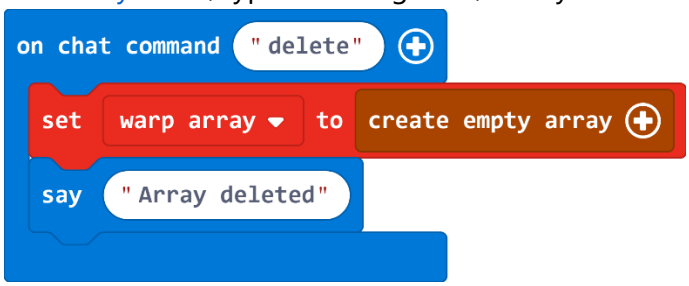
4. Name this variable “warp array”, and click Ok
5. From the Variables Toolbox drawer, drag a Set block into the On start block
6. Using the drop-down menu in the Set block, select the warp array variable


7. Click on the Advanced tab in the Toolbox to display the Arrays Toolbox drawer
8. From the Arrays Toolbox drawer, drag a Create array with block and drop it into the Set block
9. In the Create array with block, click the minus sign (–) next to the 0 slot to change the block to a Create Empty Array block




Now, let’s create a command to delete our array. When we ‘delete’ the array, we simply want to set it to an empty array.

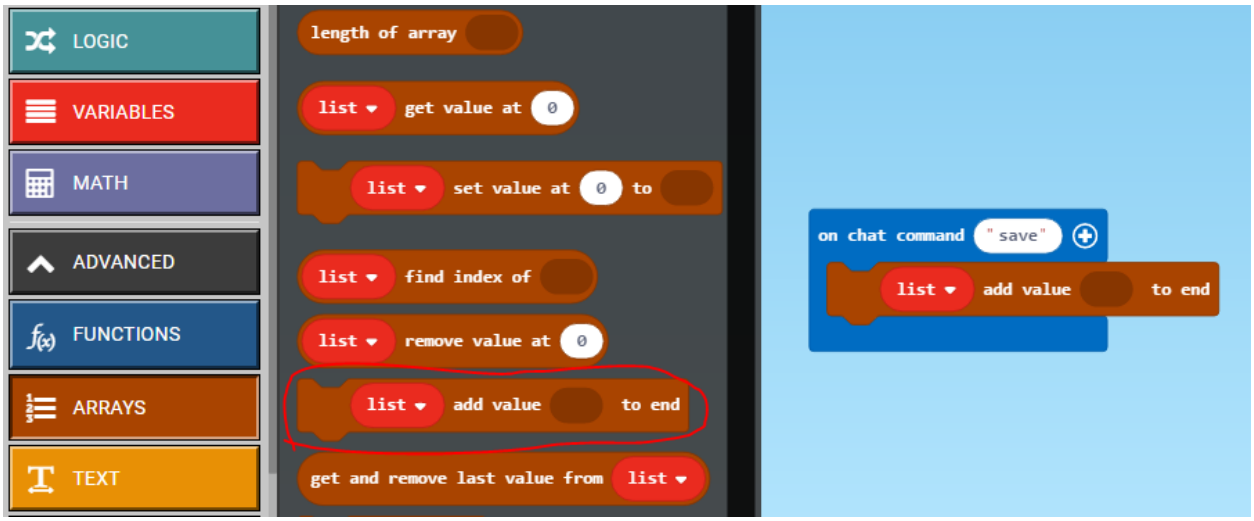
10. From the Player Toolbox drawer, drag an On chat command block onto the Workspace
11. Name this command “delete”
12. Right-click on the existing Set block in your program and select Duplicate
13. Drag this new Set block into your delete On chat command
14. From the Player Toolbox drawer, drag a Say block after the Set block
15. In the Say block, type a message like, “Array deleted”



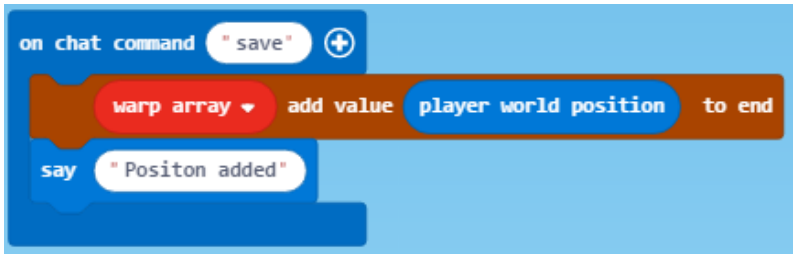

To save locations into our array, we will want the save command to just add the current position to the end of our array.

16. From the Player Toolbox drawer, drag an On chat command block to the Workspace
17. Name this command “save”

18. From the Arrays Toolbox drawer, drag the Add value to end block into the On chat command



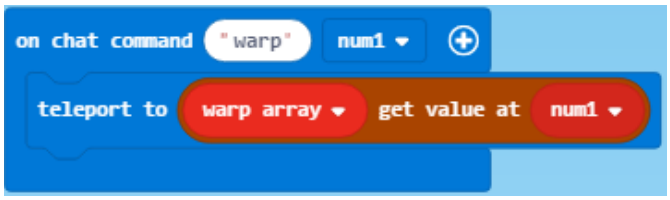

19. Using the drop-down menu in the Add value to end block, select warp array as the array we want to add values to
20. From the Player Toolbox drawer, drag a Player World Position block into the Add value to end block
21. Lastly, from the Player Toolbox drawer, drag a Say block after the Add value to end block
22. In the Say block, type a message like, "Position added"




The player will use the warp command by passing in a number, such as "warp 1" or "warp 2" to teleport themselves to a saved location in the Array.

23. From the Player Toolbox drawer, drag an On chat command block to the Workspace
24. Name this command "warp"
25. In the On chat command block, click the plus sign (+) to add a parameter, called num1. This will represent the array index of the position you are warping to.
26. From the Player Toolbox drawer, drag a Teleport to block into the On chat command block
27. From the Arrays Toolbox drawer, drag a Get value at block into the Teleport to block replacing the existing position block
28. Using the drop-down menu in the Get value at block, select the warp array variable

29. From the Variables Toolbox drawer, drag the num1 variable block into the second slot of the Get value at block




Now, when you call the warp command with a number, you will teleport to the position referenced by that array index.

Additional improvements:
There are a couple of ways we can improve this warp command.
First, recall that the first item in an array always has index 0. But it's a little strange to type "warp 0" to get to the first position in our array. Let's adjust this so that the player can enter the number of the warp starting with 1. To do this, all we need to do is subtract one from num1 to get the right index, and we'll do this in the MakeCode JavaScript editor.

30. Click the JavaScript tab at the top of the screen to see your program in the JavaScript editor


31. Find the player.onChat "warp" function
32. In the following line of code, subtract 1 from the index by changing to player.teleport (warp_array [num1 - 1])

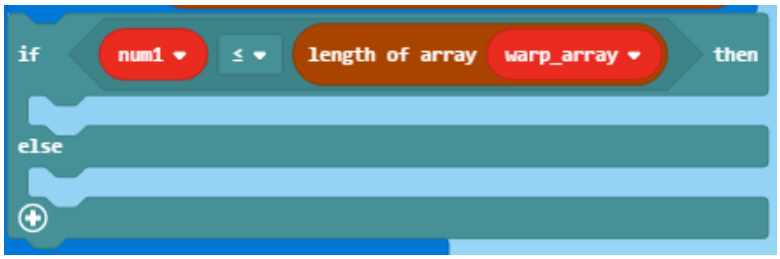
```
player.onChat("warp", function (num1) {
    player.teleport(warp_array[num1 - 1])
})
```

33. Then click the Blocks button at the top of the screen to go back to the Block editor

There is one more improvement we can do to make our program more user-friendly. If the player enters a number that is larger than the last index in the array, there should be some sort of a warning. Otherwise, it's odd to enter a command and have nothing happen. It is good programming practice to give feedback to the user for bad input or other errors.

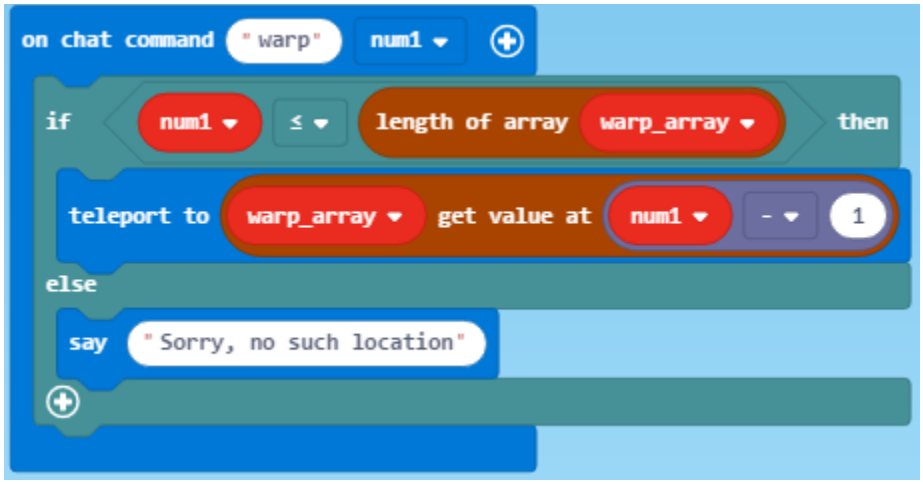
34. From the Logic Toolbox drawer, drag an If then else block into the warp On chat command
35. From the Logic Toolbox drawer, drag a Less than (<) comparison block into the first slot of the If then else block, replacing true

36. From the Variables Toolbox drawer, drag the num1 variable block into the first slot of the Less than (<) comparison block
37. From the Arrays Toolbox drawer, drag a Length of Array block into the second slot of the Less than (<) comparison block
38. From the Variables Toolbox drawer, drag the warp array variable block into the Length of Array block
39. Click on the drop-down menu in the Less than (<) comparison block and select Less than or equal to (≤)




If the index the user entered is less than or equal to the length of our array, that's okay and we'll teleport them to that location. Otherwise (else), we should give the user an error message.

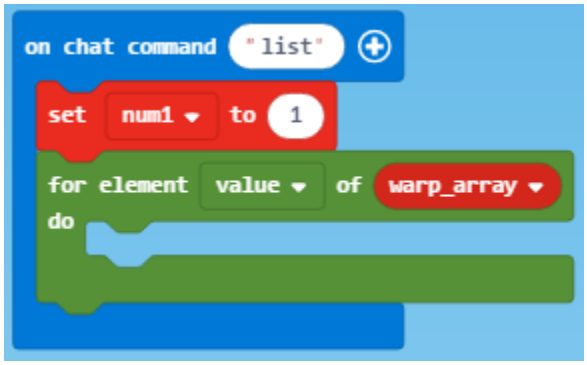
40. Drag the existing Teleport to block into the *if* clause of the If then else block
41. From the Player Toolbox drawer, drag a Say block into the *else* clause of the If then else block
42. In the Say block, type a message like, "Sorry, no such location"




One last command to complete! Let's create a way to print out all the values in the array, so we can record these later. We'll go through every index in the array and print it out along with the corresponding position.

43. From the Player Toolbox drawer, drag an On chat command block to the Workspace

44. Name this command “list”
45. From the Variables Toolbox drawer, drag a Set block into the On chat command
46. Using the drop-down menu in the Set block, select the num1 variable
47. In the Set block, type the number 1 to be the initial value
48. From the Loops Toolbox drawer, drag a For element loop after the Set block
49. In the For element block, use the variable drop-down menu to select the warp array variable




50. From the Player Toolbox drawer, drag a Say block into the For element loop
51. Under the Advanced tab in the Toolbox, you will find the Text Toolbox drawer. From the Text Toolbox drawer, drag a Join block into the Say block.
52. From the Variables Toolbox drawer, drag both the num1 variable and the value variable into the two slots in the Join block.

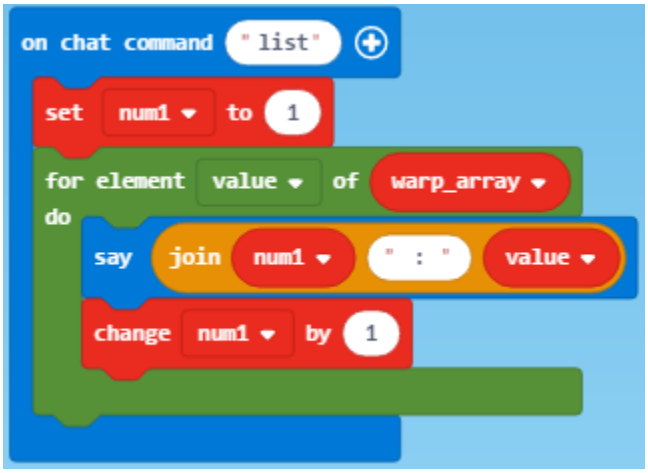
This will print out the index of the array as well as the value of the array, but these numbers will be mushed together. So let’s give them some room in between. The easiest way to do this is in JavaScript.

53. Click the JavaScript tab at the top of the screen to see your program in the JavaScript editor


54. Find the player.onChat “list” function, and within that function, find the line player.say (“” + num1 + value)
55. Change this line of code to player.say (num1 + “ : “ + value)

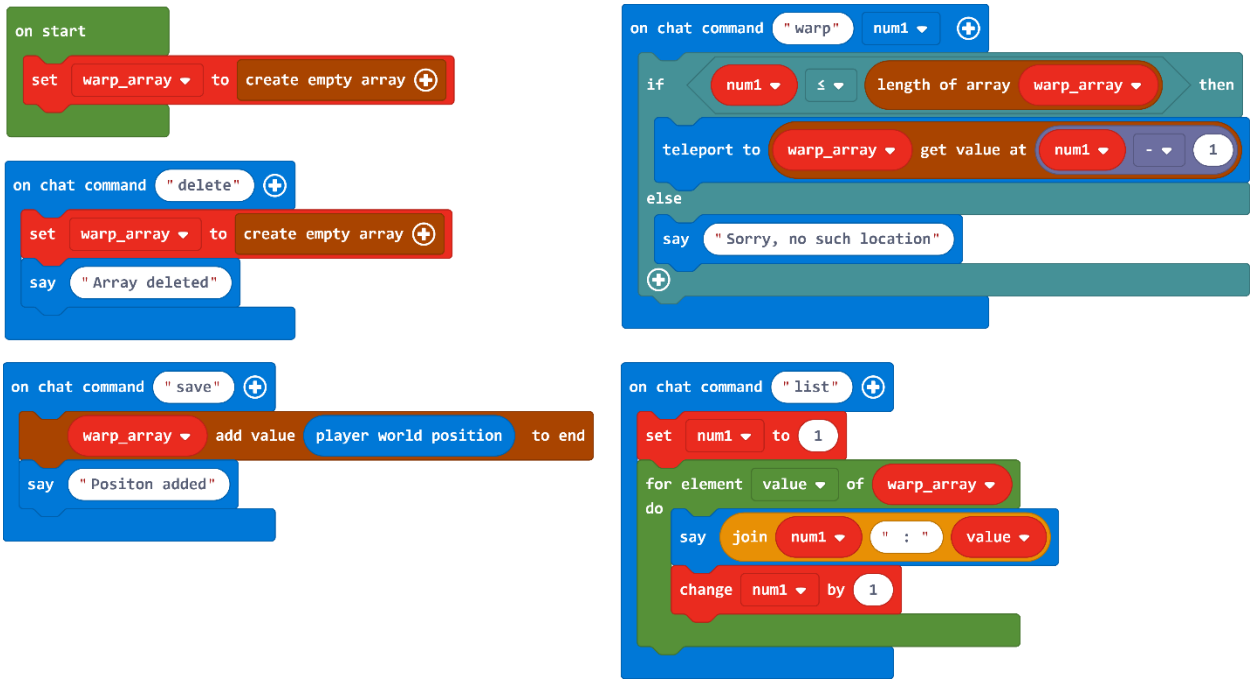
```
player.onChat("list", function () {
    num1 = 1
    for (let value of warp_array) {
        player.say(num1 + " : " + value)
    }
})
```

56. Then click the Blocks button at the top of the screen to go back to the Block editor
57. From the Variables Toolbox drawer, drag a Change block after the Say block
58. In the Change block, use the drop-down menu to select the *num1* variable




Now you have your very own way to keep track of important locations in your world during a Minecraft session!  Keep in mind that exiting your program will reset your array, so it might be a good idea to take a screenshot or write down the list of all your coordinates.

Final program:




JavaScript:

```
let warp_array: Position[] = []
let num1 = 0
```

```
player.onChat("warp", function (num1) {
    if (num1 <= warp_array.length) {
        player.teleport(warp_array[num1 - 1])
    } else {
        player.say("Sorry, no such location")
    }
})
player.onChat("delete", function () {
    warp_array = []
    player.say("Array deleted")
})
player.onChat("save", function () {
    warp_array.push(player.position())
    player.say("Positon added")
})
player.onChat("list", function () {
    num1 = 1
    for (let value of warp_array) {
        player.say(num1 + " : " + value)
        num1 += 1
    }
})
warp_array = []
```

Shared Program: https://makecode.com/_21x7XPK26Msx

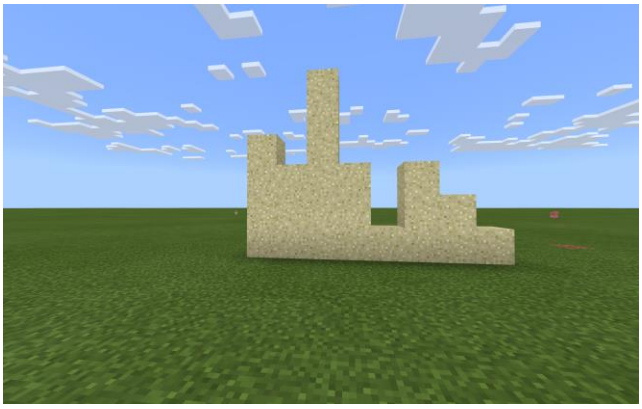
**Independent Project**

For this chapter's independent project, use an array to create a piece of artwork. For inspiration, take a look at the project called Rainbow Beacon in the Super Powers category in Code Connection. This project loads an array with colored wool, and then shoots it straight up toward the sky, as far as the eye can see.

How might you modify this project? Could you make a rainbow road stretching out to the horizon? Or create a flat square with a rainbow design? With arrays and loops, you have the power to create enormous public artwork that stretches out, and up, in different dimensions. See what kind of colorful, multi-dimensional artwork you can create!

Some ideas:

- Create designs and patterns by placing various types of decorated sandstone from an array
- Create a bar graph by dropping various amounts of sand from the sky based on the numbers in an array



- Create a pattern using an array of only black and white wool blocks

Your project should do the following:

- Use an array of items
- Access items in the array either in sequence or at random
- Only access indexes that exist in the array (no "out-of-bounds" access)

**Minecraft Diary**

Compose a diary entry addressing the following:

- What kind of art did you decide to make? What does your program do?
- Describe how your program creates its artwork
- How did you ensure that only valid indexes are accessed?
- Include at least one screenshot of your artwork
- Share your project to the web and include the URL here

NOTE: If you decided to improve one of this lesson's activities, please talk about the new code you wrote in addition to what was already provided in the lesson.

**Assessment**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Diary | Minecraft Diary entry is missing 4 or more of the required prompts. | Minecraft Diary entry is missing 2 or 3 of the required prompts. | Minecraft Diary entry is missing 1 of the required prompts. | Minecraft Diary addresses all prompts. |

| | | | | |
|---|---|---|---|---|
| Project | Project lacks all of the required elements. | Project lacks 2 of the required elements. | Project lacks 1 of the required elements. | Project creates a piece of artwork, efficiently and effectively. |
| Array | Array is improperly created or not accessed at all. | Array is properly created, some elements not reachable, and out-of-bounds access. | Array is properly created, possible to access all elements, or no out-of bounds access. | Array is properly created, possible to access all elements, no out-of-bounds access. |

**CSTA Standards**

- 3A-DA-09 Translate between different bit representations of real-world phenomena, such as characters, numbers, and images.
- 2-AP-10 Use flowcharts and/or pseudocode to address complex problems as algorithms.
- 2-AP-11 Create clearly named variables that represent different data types and perform operations on their values.
- 2-AP-12 Design and iteratively develop programs that combine control structures, inlcuding nested loops and compound conditionals.
- 2-AP-14 Create procedures with parameters to organize code and make it easier to reuse.

K-12 Computer Science Framework Core concept: Control Structures

- CT.L2-12 - Use abstraction to decompose a problem into sub problems
- CPP.L1:6-05 - Construct a program as a set of step-by-step instructions to be acted out CPP.L1:6-06 - Implement problem solutions using a block-based visual programming language
- NGSS 3-5-ETS1-2 - Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem