Sort Planets

A micro:bit makecode project (mentor guide)

REFORMAT ALL THESE INSTRUCTION AS APPROPRIATE FOR THE AGE RANGE OF YOUR GROUP

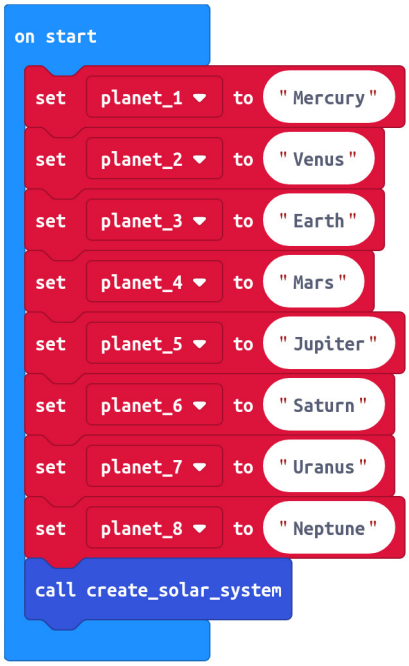
How to make this project more ‘kid friendly’...?

THINGS TO MAKE YOUR PROGRAM DO AS SOON AS IT STARTS RUNNING (BEFORE YOU PRESS ANY BUTTONS OR ANYTHING ELSE)

TASK 1

Declaring Variables

1. Make 8 variables named planet_1 to planet_8.
2. Assign a planet name (text string) to each of these 8 variables in the correct order, with planet_1 being the closest and planet_8 the furthest away from Sol (the name of our sun).
3. Add these blocks to an on start block.



To keep these planets in the correct order, we need to put them into some sort of structure, just like we'd put people in a queue or books|cards|... into a stack. When items are in a structure like this, we can deal with them in any order we want.

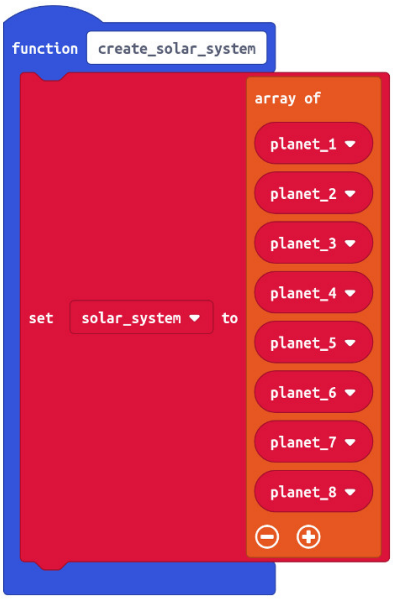
The rule for queues (like a queue of people joining a queue for a bus) is normally FIRST-IN-FIRST-OUT.

What do you think the rule for a stack of books|cards|... would be? We can put our planets into an array structure and use it however we want, either as a queue or a stack.

Make a function to create a solar system (called **create_solar_system**) that gets called when the program starts running.

Make an array variable called solar_system and add the 8 planet variables (text strings remember!) to it. Make sure they're in the correct order.

So far, you haven't told the program to display anything yet. That's ok. Everything is being stored in the micro:bits memory.



[DEMONSTRATE RUNNING PROGRAM]

nothing to see so far, unless you make an LED come on, or something.

[PROVE UNDERSTANDING]

explain how the program works so far

how will you know the program is running?

Array (queue) index numbers (0-7).

variable type used for planet variables.

THINGS TO MAKE YOUR PROGRAM DO WHEN YOU PRESS A BUTTON

TASK 2

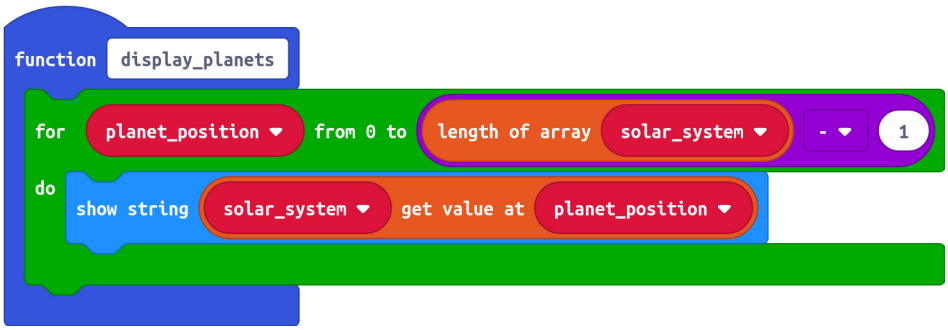
Create a function (called **display_planets)** to loop through the array, displaying the value of a planet at each loop.

1. Create a function called display_planets.

Make a for loop that will *do something* the same number of times as there are planets in the solar_system... BUT it has to start from zero!

The value of a variable called planet_position must change for each loop going FROM (zero) TO (the length of the solar_system array minus 1)

Make a block to show a string at every loop. Make this string by getting the value at planet_position in the solar_system array.



Create the code to call the display_planets function whenever button A is pressed

[DEMONSTRATE RUNNING PROGRAM]

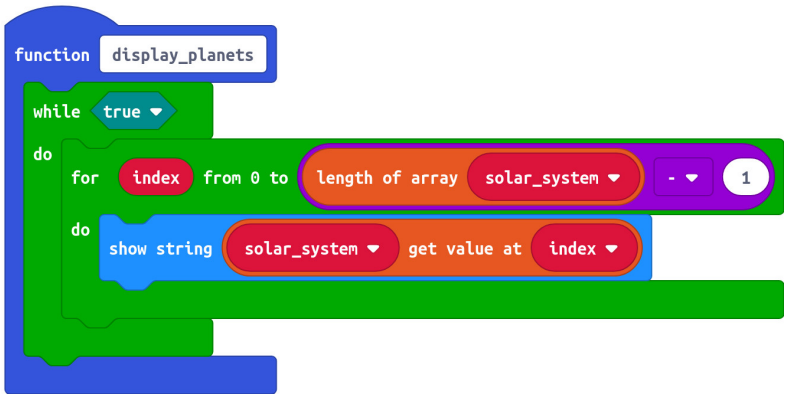
See if you can name the next planet before it scrolls onto the display!

[PROVE UNDERSTANDING]

pen and paper. Array (queue) index numbers (0-7).

TASK 3

update your display_planets function to use a while loop to keep the display going forever in an ‘infinite loop’.

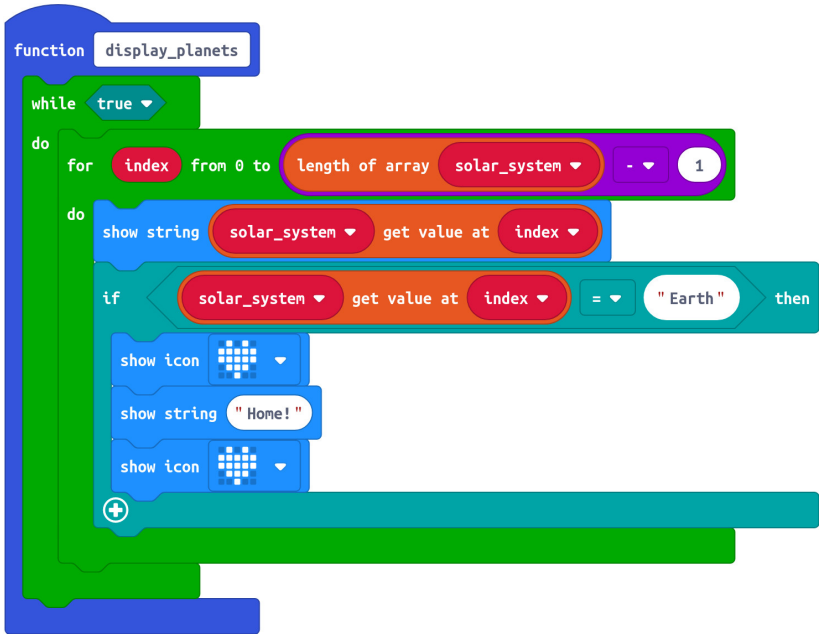


[DEMONSTRATE RUNNING PROGRAM]

[PROVE UNDERSTANDING]

TASK 4

update your display_planets function to show the text string “Home” whenever the text string used for planet3 (“Earth”) is displayed.



[DEMONSTRATE RUNNING PROGRAM]

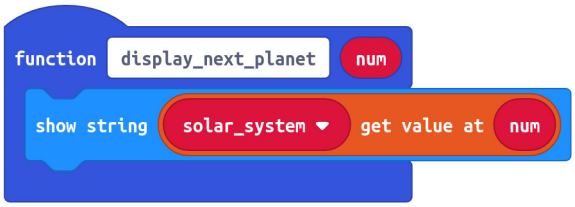
[PROVE UNDERSTANDING]

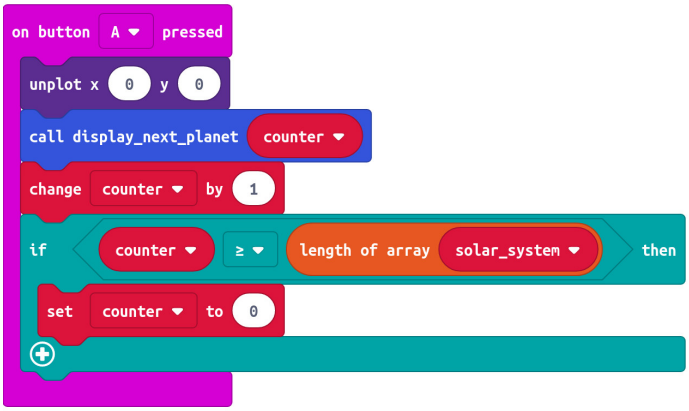
TASK 5

Update your makecode to display one planet at a time (still in order of distance from the sun), each time button A is pressed.

Write a new function called **display_next_planet**.

Whoever calls display_next_planet must also pass it the solar_system array index number of the planet it wants the new function to display.





[DEMONSTRATE RUNNING PROGRAM]

[PROVE UNDERSTANDING]

why do we have to reset the counter?

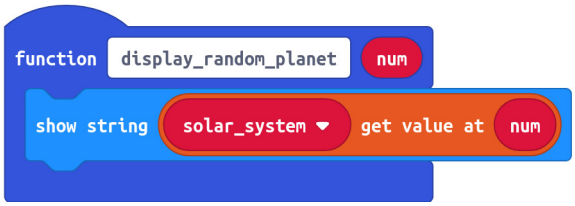
What would happen if the counter wasn't reset?

What's wrong with not just saying 'if counter >= 8...' (what if tomorrow, Pluto becomes the 9th planet again? How would the program need to be changed?)

TASK 6

To display a randomly chosen planet each time button A is pressed.

We need to be able to generate a randomly chosen number between 0 and 7.



[DEMONSTRATE RUNNING PROGRAM]

[PROVE UNDERSTANDING]

TASK 7

Now completely separate from the random planet displaying you've just done, you need to program button B to increment a counter with values between 1 and 8 (or 0 to 7) . Just as before, when the counter tries to go above 8 (or 7), it gets reset to the beginning. This time though, you need to display the counter value. A new function called display_counter to

In this order, put these blocks into an on button B pressed block:

1. A block to call the display_counter function and pass it the counter variable at the same time.
2. create a block to increment the value of counter by 1.
3. create an if block that does something if the value of counter becomes equal or greater than the number of planets in the solar_system.
4. Create a single block to do that 'something'.

create a new function called display_counter

only one block needs to go in it. make the correct block.

program button A+B to select the value currently set by the counter and assign it to a variable called planet_position_answer.

Create a function to check whether planet_position_answer is the same a the position of the planet displayed. If it is, display a "TRUE" message, otherwise display a "FALSE" message.

Program the device to keep the game going by displaying another randomly chosen planet after each response.

Program the device to be able to receive the planet_position_answer variable from another device by radio and then to respond with a message as before.

Program another device (device 2) with the counter and planet_position_answer selection functions.

Program device 2 as a radio transmitter that can send planet_position_answer variable to device 1.

… buzzer acknowledgement

… scores