Debugging!

Calvin Wong, format template from Maxson Yang

Edited by Anthony Lu, Pranav Chhabra, Kyaw Swar Ye Myint, Martin Garcia-Angel, Yolanda Shen

Table of Contents (Clickable!):

Error Messages and Common Mistakes

Expecting [ ] But Got [ ]

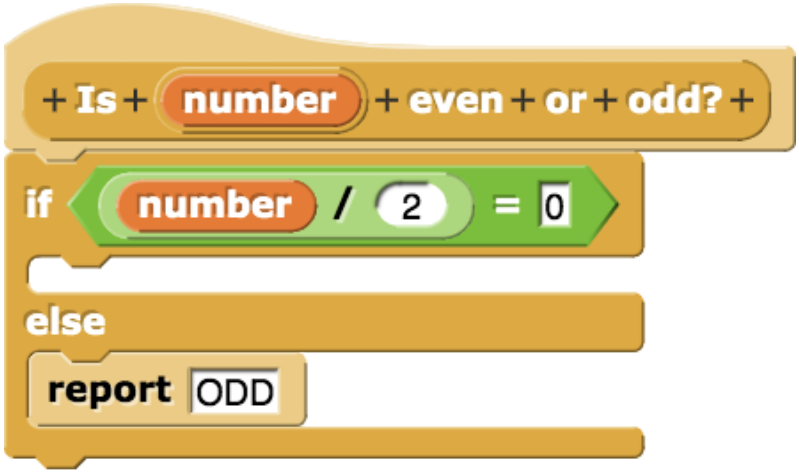
- This error means a block encountered a domain or range error. For example:

- This means the map block was receiving a numerical input despite requiring a list
- Use visible stepping (above) to see what line had the wrong input

- Check the input types on that line (see below for examples of input types)
- Ensure whatever you put into the blanks is the correct input type (and what you intended to put in)

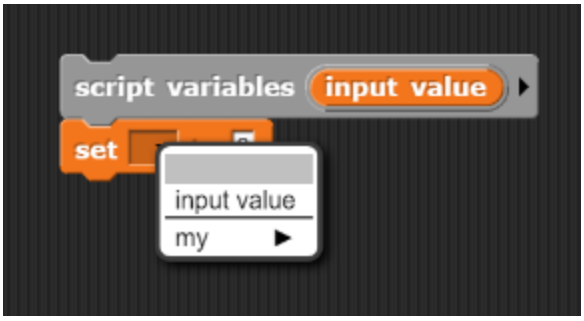
Reporter Didn't Report

- Function finished running but didn't report anything
- Notice in the left example that "Is number even or odd?" block runs but doesn't report anything, and because if the first condition runs, the "else" portion does not run, so nothing is reported.

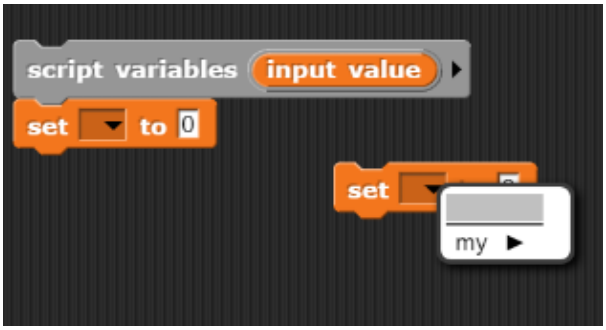


Scope of Variables Bugs/Variable Does Not Exist

- Is a global/local variable deleted?
- Are you referencing a variable that does not exist in the frame? (remember that script variables only exist in the current script)
    - *e.g. here we can access Input Value (it's within the same block):*



    - *In the right block we can't, because it isn't attached to the script variable:*



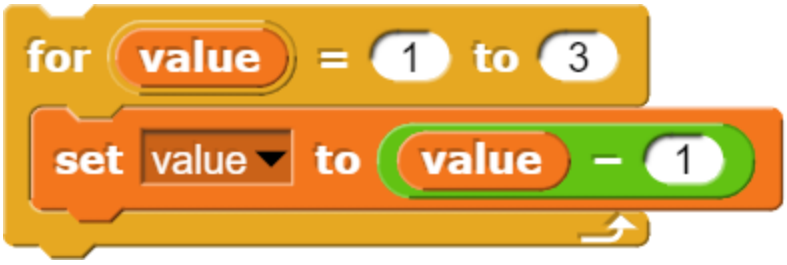
User-created blocks also have this same behavior

Infinite Loop

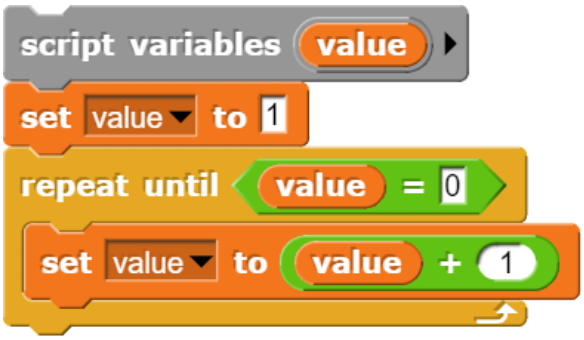
- Use visible stepping to see where the program gets stuck

Potential errors:

- Did a "repeat until condition X" not reach condition X?
- Did a for loop change the variable it's using to iterate (for example, in a "for i" loop decreasing "i" by 1 during each iteration)
    - *e.g. value will stay at 0 forever because as we iterate it becomes set to 0*



- Are you using a *forever* or *while* loop and the program never breaks out of the *forever* loop or never fulfills the *while* condition?
    - *e.g. the value never reaches zero so the loop never exits*

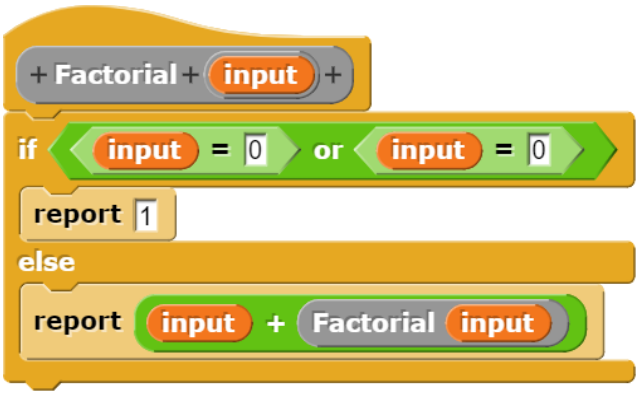


Infinite Recursion

- Use visible stepping to see where the program gets stuck

Potential errors:

- Did you forget a base case?
- Does your program not reach the base case?
    - *e.g. base case will never be reached as recursive call does not decrement input*



Repeat Variable Names

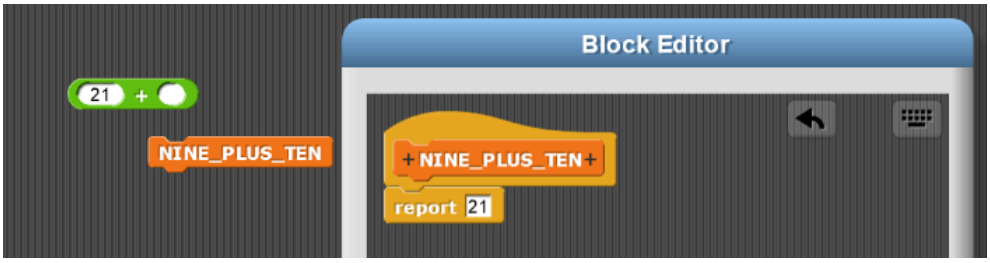
- Having 2 variables with the same name won't cause an error, but it might make it confusing when trying to keep track of values, or changing what the value of the variable is.
    - *e.g. Having a global and local variable, each named "list". In the image below, the top block is the global variable, and the script variable is the local one.*

- The top 'list' is the global variable, while the bottom 'list' refers to the script variable
- Here we can see that only one of the two variables called "list" has changed, so the same name and different values can make things confusing.

Having a Report Block in a Command Block Definition

Trying to debug your code by having your command block report the output is not necessarily wrong, but is definitely not the most viable option either. This is because command blocks are for side effects, and you won't be able to use what the command block reports.

For example, does it make sense to put NINE_PLUS_TEN into the blank circle below?



No, it doesn't make sense to put NINE_PLUS_TEN into the blank circle below! Recall that a command block corresponds to an action that Snap! already knows so essentially, by reporting the output 21 inside NINE_PLUS_TEN doesn't make sense because NINE_PLUS_TEN isn't a built in action in Snap!

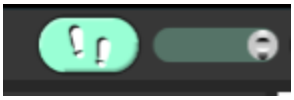
Debugging Tips

Help

- You can right click blocks, and then click “help…” to see a description of the block!

Visible Stepping

- Visible stepping allows you to watch Snap “step” through code and see what blocks run when.
- To enable, press the footsteps icon, located at the top right corner of your screen.
- There is a sliding bar, the further right it is, the faster it steps through code.
    - Fast setting:

      
    - Slow setting:

      
- Here is an example: Stepping Example *(should replace with a professionally shot video)*
- Useful for finding what line in Snap raises an error or where the program stops

Pause Button and Pause Block

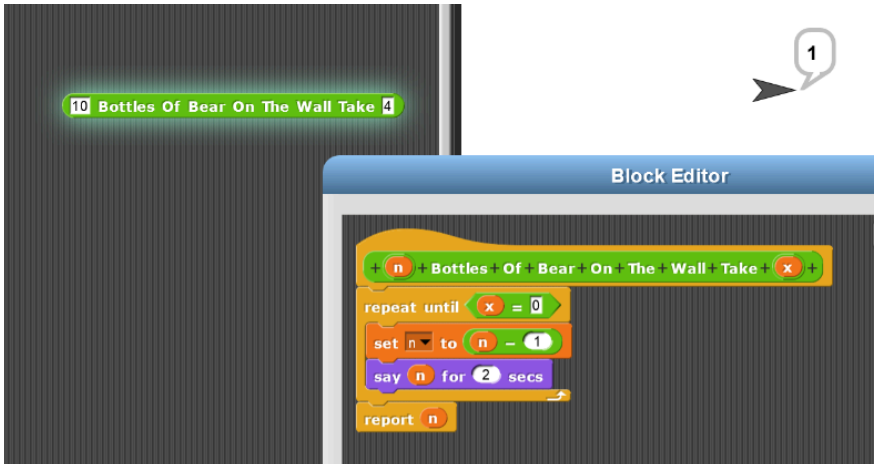
- Many forget it exists!

- This is the pause button and can be found in the top right corner of your Snap! Window.
- It can be helpful when used with visible stepping, as you can stop your script midway, and resume from that point at a different time. i.e you don't have to rerun the code to get to the same point.

- There is also a "pause all" block as shown below, which can be put in your code, and it will automatically pause the code when it reaches that block.

Using "Say" Blocks to Track Variable Changes

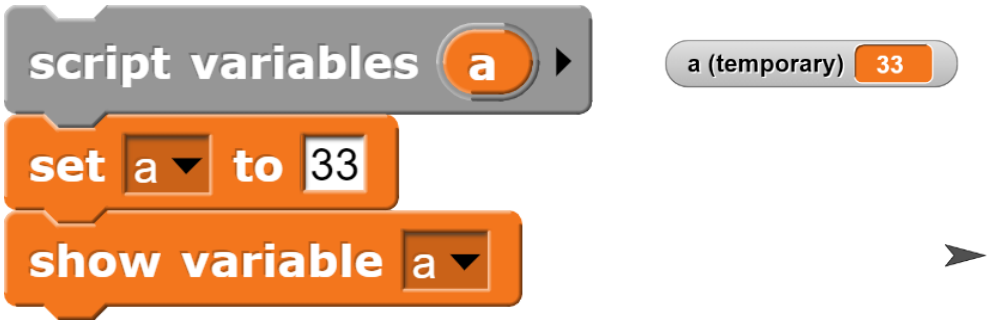
- Allows you to see how a variable changes as the code runs
- For example, in the block below, we forgot to decrease x in our loop! If we didn't have the say block, the Bottles of Bear block would simply run forever, and we wouldn't see what happens at each step.
    - However, since we have the say block, the sprite (the arrow) says what n is at each step, which gives us the clue that it is decreasing n, which we want, but something else is wrong.



Using "show variable" Block to Track Variable Changes

- This lets you keep track of both script and global variables without using the 'say' block.
- Include the following blocks in your code to show/hide variables:

- Pressing on the 'show variable' will then display on the stage like so:

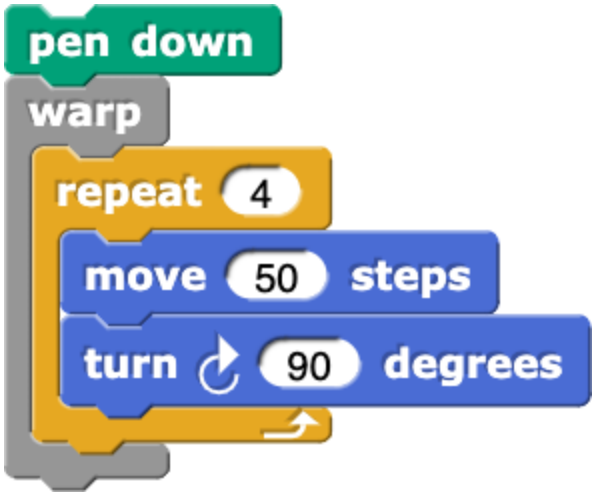


- This will show the script variable a on the stage. Including the 'hide variable' block will then remove the variable from the stage.

Making Code Run Faster: Turbo Mode/Warp Block

- You can always use the warp block or go into turbo mode to make your code run faster.

- This algorithm draws a square. Normally, you'd see the sprite draw every side.



- By using the warp block, Snap! skips the drawing, and jumps straight to the end product.

How to get "unstuck" (potentially move to another guide)

Everyone gets stuck working on problems but don't worry! Here are some tips on how you can get "unstuck" when working on a problem:

1. Think about what you're doing in regular English.
    a. For example, if you're summing the numbers between 1 and 10, you'd want to take numbers 1, 2, 3, …, 9, 10. Then you'd want to add 1 and 2, and then take that sum and add that to 3, and so forth.
2. Translate what you've thought of in regular English into code. This is all about converting inputs to outputs.
3. You should also break the problem down into small steps. What are the inputs? What do you start with?
    a. In 1, 2, 3, …, 9, 10, you know you have 1 as your start and 10 as your end. You still need the numbers in between, which you know that the next number after 1 is 2.
    b. If your next number after 1 is 2, how do you get 2? By adding 1 to 1!
    c. Thus, you need to add start to start + 1.
    d. Since you're going to be adding this sum to other numbers in the list, you need to keep track of the sum, so lets create a script variable X to store the value.
    e. But does this repeat? To get the next number, 3, we need to take our number, start + 1, and add 1 again, then take X and add that next number 3 to X. Until you reach 10. Sounds repetitive. Sounds like a job for something that repeats...maybe a repeat block? Or a loop?