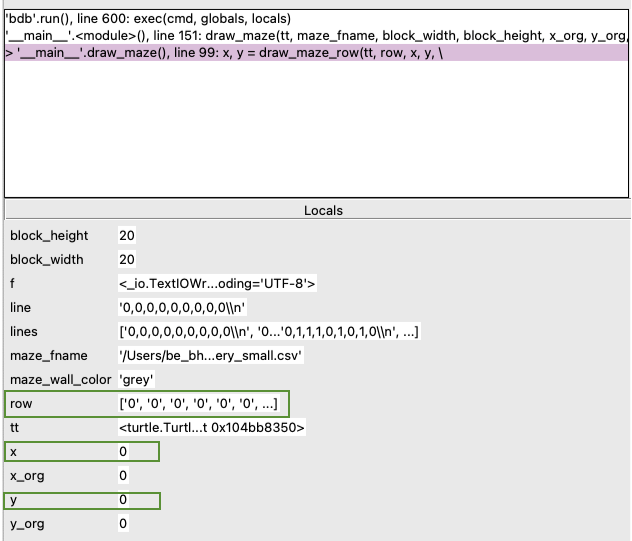
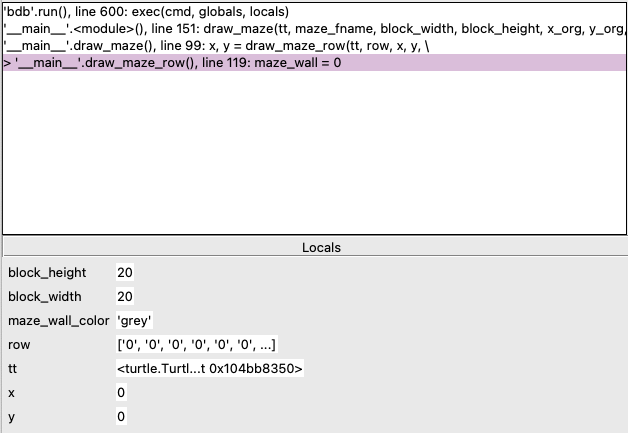
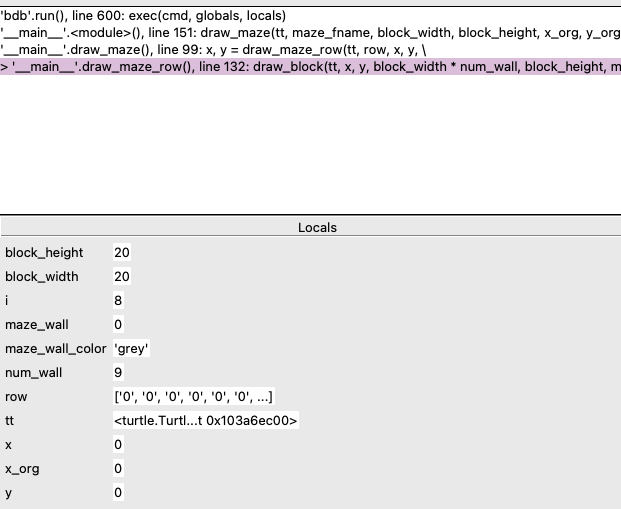
put a breakpoint at line 99. This is inside the main for-loop which iterates each row of the maze specification file and draw the corresponding wall blocks. This allows us to step through each line of codes and observe whether the values of the various variables are in fact what we expected them to be.  
**QUESTION (1)** what are the values of row, x and y shown by the debugger?  
  


Step into (or Step in IDLE) the function draw\_maze\_row() a couple of times.  
**QUESTION (2)**: once you are in the function draw\_maze\_row(), what are the values of x, y, block\_width and block\_height shown by the debugger? Please include a screenshot of your debugger window showing values of variables x, y, block\_width and block\_height.



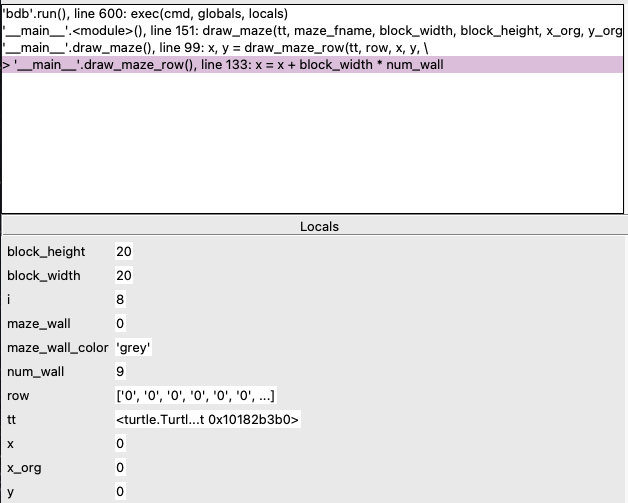
Continue to step over (or Over in IDLE) the lines in the function draw\_maze\_row(). When you step through the iteration of the for loop (line 122-135), you will notice that the value of i (the loop index) changes. Please step through until you reach line 132 (i.e. when the function draw\_block() is called).  
**QUESTION (3)**: list out all the values of i that you observed.

**i\_values =[0,1,2,3,4, 5, 6,7,8]**



**QUESTION (4)**: what is the value of num\_wall when turtle draws the bottom row (wall) i.e. when the function draw\_block() is called?

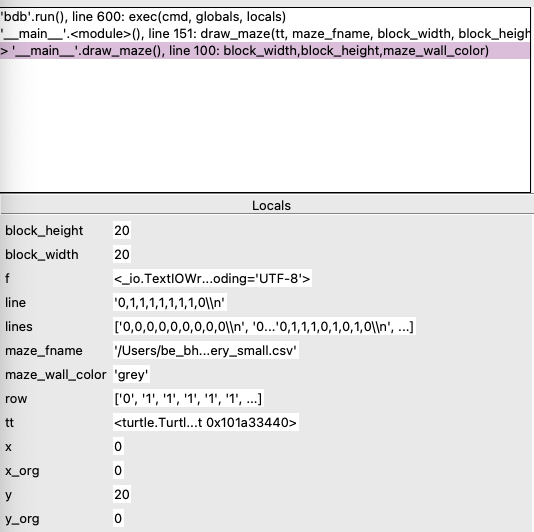
**When we step over the line 132 where the draw\_block() is called and turtle draws the bottom row (wall) the Num wall is 9**



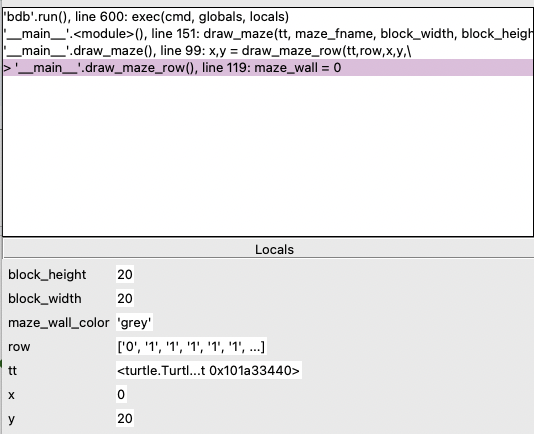
After the turtle draws the bottom row (wall) and the function draw\_maze\_row() returns, the program execution will continue with the second iteration of the for-loop (line 97-101). The program will now read the second row and attempt to draw it. Step into (or Step in IDLE) the function draw\_maze\_row().

**QUESTION (5)**: what is the values of x and y shown by the debugger? Please include a screenshot of your debugger window showing values of variables x, y.

**x = 0  
y = 20**

****

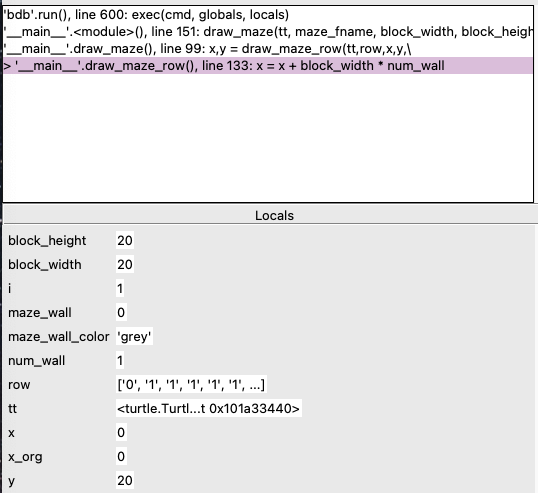
**Also another screenshot After multiple Step into (or Step in IDLE) the function draw\_maze\_row()**



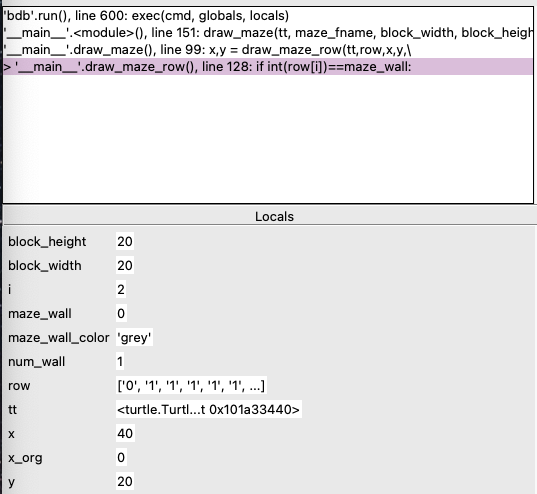
Continue to step over (or Over in IDLE) the lines in the function draw\_maze\_row() until the program draws the first block (wall) of the second row (from the bottom).

**QUESTION (6)**: what is the value of num\_wall when turtle draws the first block (wall) of the second row (from the bottom)?

**num\_wall = 1 also screenshot after the program draws the first block (wall) of the second row**

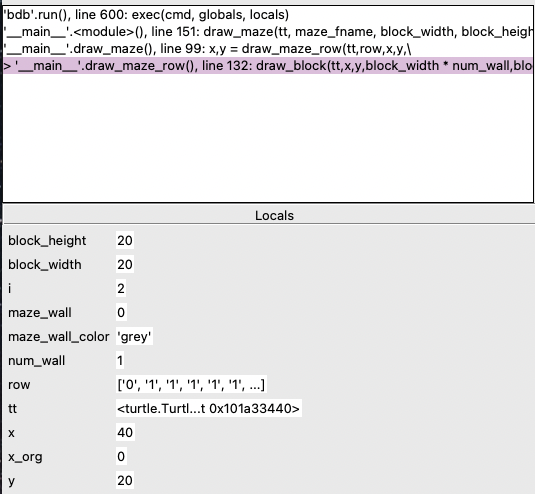
****

Continue to step over (or Over in IDLE) the lines in the for loop (line 122-135) – you should now be in the second iteration of this for-loop i.e. i should now have the value of 2. If we look at the maze specification CSV file, you will notice that this row (the second row from the bottom) consist of one wall block followed by 7 road blocks (i.e. 1’s) and ends with a wall block (0). We expect the program to draw one block at x=0 and then the second one at x = 160 (i.e. 8 x 20; note: width of block = 20).

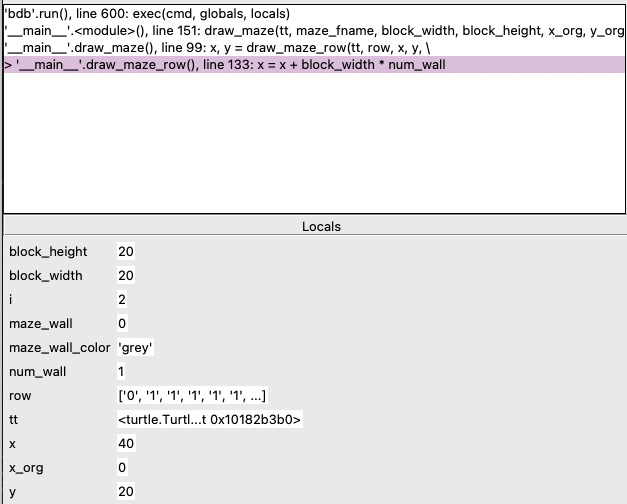
****

**QUESTION (7)**: what is the value of x when the program draws the second block (i.e. immediately before the program draws the second block)?

**x = 40**

**immediately before the program draws the second block**  


**After the program draws the second block**



Why did the program draw the second block earlier than expected?

**QUESTION (8)**: what conditions need to be true when the program draws a block or a row of blocks?

(int(row[i]) not equal to \_\_?\_\_\_ OR i == \_\_\_?\_\_\_ ) AND

num\_wall > \_\_?\_\_

**if (int(row[i]) != maze\_wall or i == len(row) - 1)**

**num\_wall > 0**

**int(row[i]) != maze\_wall ensures that the block is drawn only when a road block (1) is encountered, breaking the sequence of walls.**

**i == len(row) - 1 accounts for drawing wall blocks at the end of the row if there are consecutive wall blocks to draw.**

**num\_wall > 0 ensures there is at least one wall block to draw.**

**QUESTION (9):** what are the values of i, row[i] and num\_wall immediately before when the program draws the second block?

**i = 2**

**row[i] = 1**

**num\_wall = 1**

num\_wall is a variable used to track the number of wall blocks that have NOT been drawn but the program has “read” in memory. This variable is incremented at line 126 (when int(row[i])==maze\_wall and i<(len(row)-1) i.e. when the program is in the middle of a row of wall blocks) and line 130 (when the program is NOT in the middle of a row of blocks but the current block is a wall block). However, the program is currently NOT in the middle of a row of wall blocks and the current block is NOT a wall block. There should have been no updates to num\_wall and its values should be 0. So, it seems there is some error in the logic of this program.

Turns out, we forgot to reset the value of num\_wall to zero after the program draws the block(s). Recall num\_wall is used to keep track of the number of blocks NOT yet drawn.

Line 132 executes the codes to draw the block(s). Line 133 keeps track of the value of x (i.e. x-coordinates) after the blocks were drawn. We should add a line of code to reset the value of num\_wall to zero after line 133.  
**QUESTION (10)**: in the space provided below, please enter the codes (should be one line of codes) that should be added:

**num\_wall = 0**

**QUESTION (11):** please include a screenshot of your Git client showing the difference between your working copy and the copy on the source code repository.

