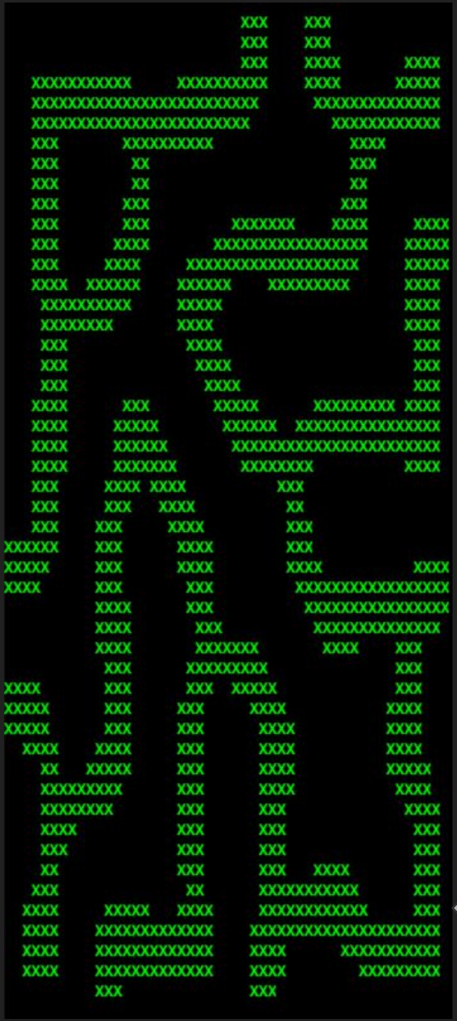
Program Structure by Sasha Bakker

1. Import libraries, initialize constant variables, initialize functions
2. Main() function
   a. Check command arguments for periodic boundary “p”, updating `isPeroidic` from 0 (False) to 1 (True).
   b. Initialize `u`, `v` arrays via the function `InitializeArrays()`.
   c. Execute time steps in a while loop by solving for the updated `u` and `v` via the function `IterativeSolution()`
   d. Print the final solution for `v` via the function `PrintSolution()`
2. Function Definitions
   a. `InitializeArrays()` initializes the 2D images for `u` and `v` as 1D arrays
   b. `IterativeSolution()` iterates through all elements of `u` and `v` and updates them using the functions `Update_u()` and `Update_v()`
   c. `Update_u()` and `Update_v()` update arrays `u` and `v` using the functions `ComputeDifferentialReflective()` or `ComputeDifferentialPeriodic()`
   d. `ComputeDifferentialReflective()` and `ComputeDifferentialPeriodic()` evaluate the discretized differential for input `u`, or `v` with reflective or periodic boundaries, depending on `isPeriodic`
   e. `PrintSolution()` prints the 2D image solution for input `v`



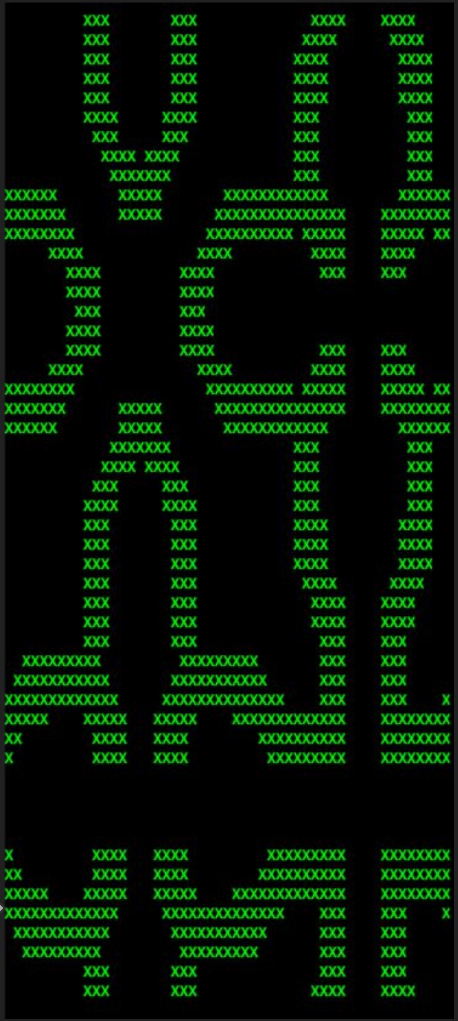
I believe the reflective boundary solution is correct because:

- The program uses the exact given equations and constants.
- The new values of `u` and `v` are computed before they are assigned such that the updated `v` does not used updated `u` in its computation.
- Each newly calculated element of `u` or `v` only depends on the old values by storing the new values separately. For example, $u^{i, j+1}$ will use the old $u^{i, j}$, not the new $u^{i, j}$ computed in the previous iteration.

I believe the periodic boundary solution is correct because it was modeled after the reflective boundary solution and the 2D image result of `v` is clearly periodic (ex. The lines on the right border continue to the left border, same for top and bottom).

Reflective Boundaries

Periodic Boundaries



Peer Review Process

I found it useful for these reasons:

1. Seeing that my output was slightly different than other students' made me realize I had to correct for point #3 regarding the reflective boundary on the previous slide. A student also commented on this which confirmed my reasoning for why my output looked different.
2. The advice from another student about what was confusing/redundant with my implementation (an unneeded function) allowed me to make my program more concise, which was helpful to edit before writing code for the periodic boundary condition
3. Seeing other code helped me better understand how to implement command line arguments.
4. The comments reminded me to get rid of extra output before submission.
5. There was a useful suggestion about implementing the `pow()` function instead of writing a function to compute the square.