2a.

This project was written in Python and uses a number of Python third-party libraries, such as Matplotlib, NumPy, and SymPy.

This program implements a level surface visualization for functions of three variables by graphing the function's level surfaces. The visualization is an analog to level curves for functions of two variables. Functions of one and two variables are trivial to represent in our three dimensional world, but functions of three dimensions are not. On a level surface graph, the surfaces represent where the function has the same value. An application of level surfaces would be to visualize how a variable changes over three dimensions, such as temperature, air pressure, or brightness from a light source.

The video demonstrates the visualization of the function $f(x, y, z) = \frac{z}{\sqrt{x^2+y^2}}$, as well as one possible interface to the program, the command line parameters.

2b.

Before starting this project, I was already familiar with Matplotlib 2D plotting, as well as how I can use NumPy matrices to plot data. In order to become familiar with the Matplotlib 3D interface, I initially plotted two variable functions by hard coding transformations of the x and y domain matrices. To make the program interactive, I replaced the hard coded expressions with evaluated interactive input. Then, I familiarized myself with enough relevant functions of the SymPy library in order to implement my mathematical algorithm. Using the interactive Python console to tinker with the libraries readily sped up my learning process. For the program as a whole, I initially implemented the program all in one to two functions, then I would separate different chunks into more readable, individual functions.

All development and library research was independent.

2c.

```
60 def solve_z_prompt():
61     """Solve C=f(x,y,z) for z for some expression f(x,y,z) that is prompted
```

```
    for to the user."""
    return solve_z_str(prompt_expression())
```

The `solve_z_prompt()` function integrates a mathematical algorithm with a logical algorithm in order to collect a string and manipulate it mathematically. There are two main branches of the algorithm in order to achieve this, the functions `solve_z()` and `prompt_expression()`.

The function `prompt_expression()` gathers a string of text from the user, deciding whether to read the command line artguments, from an interactive prompt, or from a shell pipe. The algorithm implemented by the function also decides whether the shell pipe flag should or should not have been supplied. The algorithm does not decide whether the string is a valid expression.

The function `solve_z()` implements the following algorithm: the expression $f(x, y, z)$ is put into an equation, $C = f(x, y, z)$, whereas C is an arbitrary constant; the equation is solved for z; then the solutions to the equation are returned. The library call dictates that a list of expressions that satisfy the solution is returned.

Putting these functions together, `solve_z_prompt()` prompts the user for an expression, then returns the list of solutions of that expression set equal to C and solved for z.

2d.

```
def map_f_xy(f, mesh):
    """Map a numpy (x,y) meshgrid to a function of two variables."""
    domain = np.array(mesh).transpose(1, 2, 0)
    z_range = np.empty(mesh[0].shape)

    for i, domain_i in enumerate(domain):
        for j, (x_ij, y_ij) in enumerate(domain_i):
            z_ij = f(x_ij, y_ij)
            z_range[i][j] = z_ij if z_ij.is_real else np.nan
```

```
92
93     return z_range
```

The function `map_f_xy()` makes it simple to iterate over a 2D table of coordinate pairs (which is a 3D table of numbers). The function is analogous to a Numpy vector function, like the functions `np.sin()` or `np.sqrt()`, which apply the same function to each individual number in the matrix, but `map_f_xy()` applies the same function to each coordinate pair in the input meshgrid. The structure of the meshgrid is that x and y values are stored in separate tables but in the same index, so working with individual coordinate pairs is difficult. Applying a transposition to the meshgrid will pair each corresponding x and y and arrange them in a 2D table. Then, it is simple to iterate through each row. On each iteration, the input function is applied to the coordinate pair, and the answer is stored in the output matrix.

This abstraction is required, because the main purpose of the program is to plot 3D functions over a 2D domain.