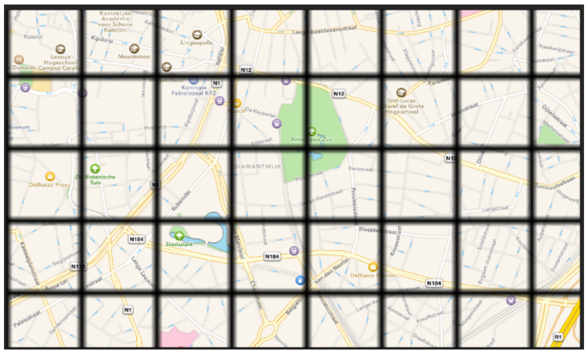
## Short Description

You will implement a simple program that uses a two-dimensional nested list to represent a rectangular grid of cells on a map. Each cell contains a value that represents the housing price for a typical house in that cell. The data is contained in a file that you read at the start. However, the data is incomplete - there are some missing values, represented by zeros in this grid. A realtor estimates such a missing price by taking the average of the prices in the neighboring cells. The realtor is also interested in the average and the maximum price of housing in the whole neighborhood (the whole grid).

Map with a rectangular overlay of cells



→

House prices in each cell

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 4 | 5 | 0 | 6 | 2 | 1 | 4 |
| 2 | 3 | 1 | 4 | 3 | 6 | 2 | 9 |
| 6 | 2 | 4 | 7 | 5 | 0 | 9 | 1 |
| 7 | 4 | 3 | 9 | 2 | 4 | 8 | 5 |
| 2 | 0 | 7 | 5 | 1 | 4 | 6 | 8 |

## Learning Outcomes

- Practice basic programming skills
- Discover new language features[1]
- Practice interacting with the terminal window, file manager, and WingIDE
- Learn how to create, display, access and update nested lists

## Detailed Explanation and Things To Do

- Neighbors are all the cells that are in the grid and are horizontally, vertically or diagonally adjacent. So the maximum number of neighbors is eight (for any cell away from the edges), and the minimum number is three (for a corner cell).
- Here are some sample runs of the grid program.

Sample run 1:

[1] To create this program, you may need to use a few programming language features that have not been used yet in class. Discovering new language features and how to use them is an integral part of problem-solving in computing science and an essential skill that you should learn. Think about what you need to do, search the web for python3 programming examples, and/or use the Python documentation to help you find the programming constructs that you need. If you get stuck, ask your TA for help/hints about the programming constructs you need to use.

```
This is our grid:
| 64000 | 81000 | 0 |
| 26000 | 52000 | 68000 |
| 84000 | 43000 | 0 |

This is our newly calculated grid:
| 64000 | 81000 | 67000 |
| 26000 | 52000 | 68000 |
| 84000 | 43000 | 54333 |

STATS
Average housing price in this area is:  59926
Maximum housing price in this area is:  84000
```

Sample run 2:

```
This is our grid:
| 94000 | 64000 | 30000 | 0 | 14000 | 92000 |
| 37000 | 49000 | 50000 | 29000 | 35000 | 0 |
| 0 | 88000 | 85000 | 96000 | 60000 | 22000 |
| 13000 | 44000 | 73000 | 0 | 45000 | 53000 |
| 20000 | 33000 | 67000 | 71000 | 82000 | 0 |
| 36000 | 0 | 62000 | 55000 | 44000 | 75000 |

This is our newly calculated grid:
| 94000 | 64000 | 30000 | 31600 | 14000 | 92000 |
| 37000 | 49000 | 50000 | 29000 | 35000 | 44600 |
| 46200 | 88000 | 85000 | 96000 | 60000 | 22000 |
| 13000 | 44000 | 73000 | 72375 | 45000 | 53000 |
| 20000 | 33000 | 67000 | 71000 | 82000 | 59800 |
| 36000 | 43600 | 62000 | 55000 | 44000 | 75000 |

STATS
Average housing price in this area is:  53227
Maximum housing price in this area is:  96000
```

- Simplifying assumption: you can assume that if a cell value is missing (the value is given as 0), then none of the neighboring cell values are also missing. So you will always take averages over non-zero values. You can think about how to solve the problem correctly if some neighbors are also missing (0), but you do not have to program that case in this lab.

- Your code must implement the major logical tasks, such as generating a data grid as a nested list, displaying the grid, filling the gaps, finding the average and maximum prices, and accessing elements of the grid as *user-defined functions*.

- You should do all computations and store all values with full precision as provided by Python. Only when you are printing, you round all printed values to

the nearest integers. You can use the built-in Python function round() for this purpose. See
https://docs.python.org/3/library/functions.html?#round

- Make sure you are following the code quality standards outlined in the Software Quality Tests.

- Implement the following functions:
    - **`create_grid(filename)`**
        - Description: Create a nested list based on the data given in a file. (More information is given in the Resources section)
        - Parameters:
            - `filename` - A string representing the name of a file
        - Returns: A two-dimensional nested list populated with data

    - **`display_grid(grid)`**
        - Description: Display the grid by printing it to the terminal as seen in the sample run. This function should round the values to integers using round().
        - Parameters:
            - `grid` - A two-dimensional nested list

        - Returns: None. This function works through the side effect of printing to the terminal.

    - **`find_neighbors(row_index, col_index, grid)`**
        - Description: Find the values of all the neighbors of a particular cell in the grid. (See the Examples section)
        - Parameters:
            - `row_index` - An int representing the row index
            - `col_index` - An int representing the column index
            - `grid` - A two-dimensional nested list
        - Returns: A list with the values of all the neighbors of a given cell

    - **`fill_gaps(grid)`**
        - Description: Create a **new** two-dimensional list that is identical to the original, but with all zero-cells replaced with the average of their neighbors. You should be calling `find_neighbors(row_index, col_index, grid)` on each zero-cell in order to help you calculate the average of all neighbors. (See the Examples section)
        - Parameters:
            - `grid` - A two-dimensional nested list
        - Returns: A new two-dimensional nested list with no zero cell

    - **`find_max(grid)`**
        - Description: find and return the maximum house value in all cells in the grid using nested loops
        - This function should not change the grid

- Parameters:
    - `grid` - A two-dimensional nested list
- Returns: the maximum as described above

- **`find_average(grid)`**
    - Description: find and return the average house value in all cells in the grid using nested loops.
    - This function should not change the grid
    - Parameters:
        - `grid` - A two-dimensional nested list
    - Returns: the average as described above

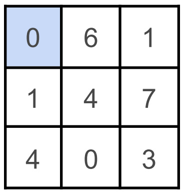
- The main function must handle:
    - Create a grid with the given data from a file
    - Display the grid
    - Calculate the average of all neighbors for each missing value in the grid. Call `fill_gaps()` to create a second, updated grid. This new grid will be identical to the original, but all zero-cells will be replaced
    - Display the updated grid
    - Display the average of all cells in the grid
    - Display the maximum number of all cells in the grid

## Examples - `find_neighbors()`

Below are some examples of how the `find_neighbors()` function should work. We cover three use-cases here (corner, wall, and middle).

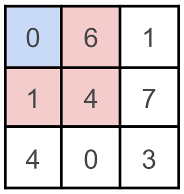
### `find_neighbors(0, 0, grid)`

Position (0, 0) is highlighted in blue.



| | | |
|---|---|---|
| 0 | 6 | 1 |
| 1 | 4 | 7 |
| 4 | 0 | 3 |

The neighbors are highlighted in red.



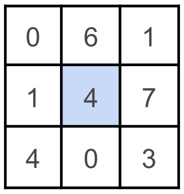
| | | |
|---|---|---|
| 0 | 6 | 1 |
| 1 | 4 | 7 |
| 4 | 0 | 3 |

The function returns a list of the neighbors' values:

```
[6, 1, 4]
```

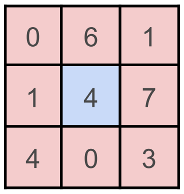
### `find_neighbors(1, 1, grid)`

Position (1, 1) is highlighted in blue.



| | | |
|---|---|---|
| 0 | 6 | 1 |
| 1 | 4 | 7 |
| 4 | 0 | 3 |

The neighbors are highlighted in red.



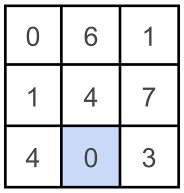
| | | |
|---|---|---|
| 0 | 6 | 1 |
| 1 | 4 | 7 |
| 4 | 0 | 3 |

The function will return a list of the neighbors' prices:

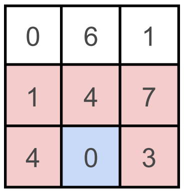
```
[0, 6, 1, 1, 7, 4, 0, 3]
```

**`find_neighbors(2, 1, grid)`**

Position (2, 1) is highlighted in blue.



The neighbors are highlighted in red.



The function returns a list of the neighbors:

```
[1, 4, 7, 4, 3]
```

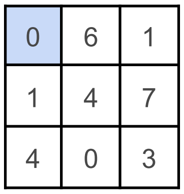
---

## Example - `fill_gaps(grid)`

Below is one example of how `fill_gaps()` should work. Just remember that the `fill_gaps()` function creates a new grid, and does NOT modify the original one.

**`fill_gaps(grid)`**

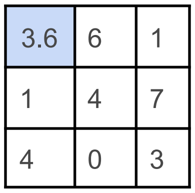
Position (0, 0) of our grid is highlighted in blue.



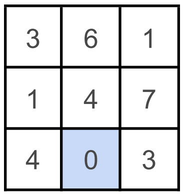
We can get the average value of the neighbor cell prices using the **`find_neighbors`** function.

```
[6, 1, 4]
```

Position (0, 0) in the new grid is set to that average (assume repeating value for 3.666):



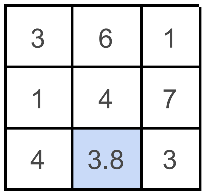
Position (2, 1) of our grid is highlighted in blue.



We can average the values from our **`find_neighbors`** function to fill the gap.

```
[1, 4, 7, 4, 3]
```

Position (2, 1) of our new grid becomes:



## Resources

The data in the file is in the following format:

- The first two lines contain the dimensions of the grid. The first line contains the number of rows and the second line the number of columns. Each remaining line contains the house value for one cell in your grid, in the following order: all data for the first row, followed by the second row, and so on. The data for each row is just the values, going from left to right in that row.
- Download the following files to test your program:
  - data_1.txt
  - data_2.txt

## Submission Information

- Please submit the following file by the submission due date for this lab.

  **`grid.py`**