Version 1.02

Assignment 2
Diffusion

1. Submission Guidelines

Deadline:	11:59 PM on Wednesday 14 December 2022
Submission procedure:	Submit only one file labelled `diffusion.py` through blackboard (via TurnItIn)
Version requirement:	Your code must run using **Python 3.10.5 on a PC**
Allowable import modules:	None

2. Overview

In this assignment, you are tasked with writing code that simulates diffusion. The task involves creating an M by N array, `space`, which contains value states (a float value between 0.0 and 1.0). This state value represents the normalised density of a diffusing material. For example, a state of 1.0 represents a high concentration of the material and 0.0 represents no presence of the material.

The simulation will use a modified Diffusion equation to describe how the material diffuses through space and time. It will also require an initial condition (the initial state of the space) and how to handle the boundaries of the space (left, right, top, and bottom borders).

3. Numerical methods

Our numerical model is based on the Diffusion equation, which is also known as the Heat equation.

Given an initial state of the space and the boundary conditions, you will need to use the following calculations to predict subsequent space states:

The state of `space` should be updated using the following equations:

$$\Delta ux_{i,j} = (u_{i,j+1}-2u_{i,j}+u_{i,j-1})/h^2$$

$$\Delta uy_{i,j} = (u_{i+1,j}-2u_{i,j}+u_{i-1,j})/h^2$$

$$\Delta u_{i,j} = 0.0001*\Delta t*(\Delta ux_{i,j}+\Delta uy_{i,j})$$

where `u` is that current state value in one of the `space`'s cell located at index `i` (spanning the rows from top to bottom) and `j` (spanning the columns from left to right). $\Delta u_{i,j}$ is the change in state that should be added to the current state to get to the next state. For this assignment, use a Δt value of 1 and a `h` value of 1.

The boundary conditions can be either Direchlet boundary conditions or Neumann boundary conditions. These boundary conditions must be somehow defined, otherwise, you will not know how to handle the states at the border, where $u_{i,j+1}$ or $u_{i,j-1}$ may not exist.

In the Direchlet boundary conditions, these 'one index beyond the border' states ($u_{i,j+1}$ or $u_{i,j-1}$) are set to a constant value (eg, as defined by the value in `left_bc`).

In the Neumann boundary conditions, the derivative at the border state values is set to a particular value (eg, as defined by the value in `left_bc`). However, in the Neumann boundary conditions, the 'one index beyond the border' states ($u_{i,j+1}$ or $u_{i,j-1}$) are not defined. You will need to use the derivative, below, to determine the border state:

$$(u_{i,j+1}-u_{i,j-1})/(2*h) = bc_j$$

Where bc_j is element `j` of the boundary condition (`left_bc`, `right_bc`, `top_bc`, or `bottom_bc`) defined as the attribute values.

4. Class `Diffusion`

The class `Diffusion` should be used to simulate how particles diffuse in space.

4.1. Attribute values

Objects of type `Diffusion` should have the following variable attributes:

`space`	A 2-dimensional array of type `list`. Each element contains a value of type `float`. The value in each element represents the normalised density of a diffusing material. The value is a unitless value between 0 and 1.
`rows`	The number of rows in `space`. The attribute `rows` should be of type `int`.
`cols`	The number of columns in `space`. The attribute `cols` should be of type `int`.
`bc_settings`	A 1-dimensional array of type list containing 4 elements, with each element containing a value of type `int`. Each element (e.g., `bc_settings[0]`) represents the boundary condition setting for the left, right, top, and bottom boundaries of the space, in that order. The possible settings are `1` or `2`, where `1` represents a Direchlet boundary condition and `2` represents the Neumann boundary condition. So, to set the left and right boundaries to Direchlet boundary conditions and the top and bottom boundaries to Neumann boundary conditions, `bc_settings` should be set to `[1, 1, 2, 2]`.
`left_bc`	A 1-dimensional array of type `list` containing values of type `float`. The length of the array should be equal to `rows`. The meaning of the value will change depending on the value of `bc_settings[0]`.
`right_bc`	A 1-dimensional array of type `list` containing values of type `float`. The length of the array should be equal to `rows`. The meaning of the value will change depending on the value of `bc_settings[1]`.
`top_bc`	A 1-dimensional array of type `list` containing values of type `float`. The length of the array should be equal to `cols`. The meaning of the value will change depending on the value of `bc_settings[2]`.
`bottom_bc`	A 1-dimensional array of type `list` containing values of type `float`. The length of the array should be equal to `cols`. The meaning of the value will change depending on the value of `bc_settings[3]`.

4.2. Methods

`__init__()`

Return values:	None
Arguments:	Number of rows, number of columns, the boundary conditions settings, the left boundary conditions, the right boundary conditions, the top boundary conditions, and the bottom boundary conditions. The number of rows and columns should be of type `int` and have default values of `10`. The boundary conditions settings arguments should be a 1-dimensional array of type `list` containing values of type `int` and have a default value of `[1, 1, 1, 1]`. The left and right boundary conditions should be a 1-dimensional array of type `list` containing values of type `float` and have a default value of `rows*[0]`. The top and bottom boundary conditions should be a 1-dimensional array of type `list` containing values of type `float` and have a default value of `cols*[0]`.

The method can have any number between 0 and 7 arguments passed into it (this count does not include the `self` argument). All arguments should be positional arguments. Arguments that are not defined should be set to their default values.

Function: In this method, use the provided arguments or default values to setup all 8 variable attributes. This includes creating the `space` variable attribute. The default value in space should be `0`.

`set_cell()`

Return values: None

Arguments: The row range, the column range, and a state.

The row range argument is a 1-dimensional array of type `list` that contains 2 elements containing values of type `int`. Element 0 contains the first row index that you would like to set while element 1 contains the last row index that you would like to set.

The column range argument is a 1-dimensional array of type `list` that contains 2 elements containing values of type `int`. Element 0 contains the first column index that you would like to set while element 1 contains the last column index that you would like to set.

The state is the value that you would like to change the target cells to. State should be a value of type `float` between 0.0 and 1.0.

All 3 arguments (not counting `self`) are required for the method to be run successfully.

Function: In this method, change the state of the cells in `space` as defined by the arguments. Changes to `space` will be made as single element changes, lines of elements, or squares.

Sample code:

```
space1 = Diffusion(11,15,[1, 1, 1, 1], 11*[1], 11*[1], 15*[0], 15*[0])
space1.print_space()
print()
space1.set_cell([4,6],[5,9],1)
space1.print_space()
```

Sample console output:

```
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 1.00 1.00 1.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 1.00 1.00 1.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 1.00 1.00 1.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

`next_step()`

Return values: None

Arguments: The number of time steps

The number of time steps should be a single value of type int. It should represent the number of time steps to run the simulation with.

Function: This is the main part of the simulation code and will take the majority of your time to implement.

Simulate the how the normalised density of a diffusing material changes over time according to the equations described in section 3.

Sample code:

```
space1 = Diffusion(11,15,[1, 1, 1, 1], 11*[1], 11*[1], 15*[0], 15*[0])
space1.set_cell([4,6],[5,9],1)
space1.next_step(100)
space1.print_space()
```

Sample console output:

```
0.0099 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0001 0.0098 0.0099 0.0099 0.0099 0.0098 0.0001 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0098 0.9804 0.9901 0.9901 0.9901 0.9804 0.0098 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0099 0.9901 0.9999 0.9999 0.9999 0.9901 0.0099 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0098 0.9804 0.9901 0.9901 0.9901 0.9804 0.0098 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0001 0.0098 0.0099 0.0099 0.0099 0.0098 0.0001 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0099
0.0099 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0099
```

Sample code:

```
space1 = Diffusion(11,15,[1, 1, 1, 1], 11*[0], 11*[0], 15*[0], 15*[0])
space1.set_cell([4,6],[5,9],1)
space1.set_cell([0,2],[0,2],1)
space1.next_step(10000)
space1.print_space()
```

Sample console output:

```
0.2383 0.3427 0.2974 0.1649 0.0656 0.0234 0.0111 0.0083 0.0071 0.0053 0.0028 0.0011 0.0003 0.0001 0.0000
0.3427 0.4932 0.4286 0.2405 0.1031 0.0501 0.0377 0.0353 0.0320 0.0241 0.0128 0.0048 0.0014 0.0003 0.0001
0.2973 0.4284 0.3752 0.2211 0.1228 0.1066 0.1161 0.1203 0.1111 0.0841 0.0445 0.0168 0.0048 0.0011 0.0002
0.1644 0.2386 0.2169 0.1567 0.1597 0.2326 0.2933 0.3127 0.2905 0.2202 0.1165 0.0440 0.0126 0.0028 0.0005
0.0638 0.0955 0.1013 0.1237 0.2297 0.4075 0.5324 0.5713 0.5313 0.4027 0.2132 0.0805 0.0230 0.0052 0.0010
0.0192 0.0322 0.0500 0.1088 0.2601 0.4837 0.6367 0.6841 0.6364 0.4823 0.2553 0.0964 0.0275 0.0062 0.0011
0.0051 0.0111 0.0281 0.0833 0.2143 0.4030 0.5314 0.5711 0.5313 0.4027 0.2132 0.0805 0.0230 0.0052 0.0010
0.0013 0.0040 0.0135 0.0445 0.1167 0.2202 0.2905 0.3122 0.2904 0.2202 0.1165 0.0440 0.0126 0.0028 0.0005
0.0003 0.0013 0.0050 0.0169 0.0446 0.0841 0.1110 0.1193 0.1110 0.0841 0.0445 0.0168 0.0048 0.0011 0.0002
0.0001 0.0003 0.0014 0.0048 0.0128 0.0241 0.0318 0.0342 0.0318 0.0241 0.0128 0.0048 0.0014 0.0003 0.0001
0.0000 0.0001 0.0003 0.0011 0.0028 0.0053 0.0070 0.0076 0.0070 0.0053 0.0028 0.0011 0.0003 0.0001 0.0000
```

Sample code:

```
space1 = Diffusion(11,15,[2, 2, 1, 1], 11*[0], 11*[0], 15*[1], 15*[1])
space1.set_cell([4,6],[5,9],1)
space1.set_cell([0,2],[0,2],1)
space1.next_step(10000)
space1.print_space()
```

Sample console output:

```
0.9280 0.8966 0.7951 0.6459 0.5427 0.4997 0.4874 0.4846 0.4834 0.4816 0.4791 0.4773 0.4765 0.4763 0.4763
0.8176 0.7726 0.6273 0.4151 0.2721 0.2180 0.2055 0.2030 0.1997 0.1919 0.1805 0.1725 0.1691 0.1680 0.1678
0.6094 0.5708 0.4477 0.2728 0.1696 0.1525 0.1618 0.1659 0.1568 0.1298 0.0902 0.0625 0.0505 0.0468 0.0461
0.3218 0.3021 0.2417 0.1700 0.1704 0.2427 0.3033 0.3227 0.3006 0.2302 0.1266 0.0540 0.0226 0.0130 0.0111
0.1229 0.1180 0.1088 0.1268 0.2318 0.4094 0.5342 0.5731 0.5332 0.4046 0.2151 0.0824 0.0249 0.0072 0.0038
0.0371 0.0388 0.0522 0.1097 0.2607 0.4843 0.6372 0.6846 0.6369 0.4829 0.2559 0.0970 0.0281 0.0070 0.0029
0.0116 0.0145 0.0304 0.0853 0.2161 0.4049 0.5332 0.5729 0.5332 0.4046 0.2151 0.0824 0.0249 0.0072 0.0038
0.0126 0.0143 0.0236 0.0546 0.1268 0.2302 0.3005 0.3222 0.3005 0.2302 0.1266 0.0540 0.0226 0.0130 0.0111
0.0463 0.0470 0.0506 0.0626 0.0902 0.1298 0.1567 0.1650 0.1567 0.1298 0.0902 0.0625 0.0505 0.0468 0.0461
0.1679 0.1681 0.1691 0.1725 0.1805 0.1918 0.1995 0.2019 0.1995 0.1918 0.1805 0.1725 0.1691 0.1680 0.1678
0.4763 0.4763 0.4765 0.4773 0.4791 0.4816 0.4833 0.4838 0.4833 0.4816 0.4791 0.4773 0.4765 0.4763 0.4763
```

5. Coding rules

Do not declare any variables in the global space of `diffusion.py` other than the class described here. The global space definitions must have the 3 methods requested. A `main()` function could be used if you so desire, but is not required. The class space may have additional function definitions that can be called by the 3 methods requested in this assignment. No module is allowed for this assignment.

6. Marking criteria

We will mark your submitted `diffusion.py` code according to the following categories:

- **60 marks:** Implementation and evidence of coding knowledge
- **30 marks:** Coding efficiency.
  Efficiency marks can only be obtained if `next_step()` is implemented with either the Dirchlet or Neumann conditions.
- **10 marks:** Coding style and commenting. See Lecture 4 for expectations.

We will use `import diffusion` near the top of our script to test your code. We will run several test conditions against each of your classes and methods. We will investigate what was assigned in your attribute variables, so the type, length, etc. of your attribute variables must be accurately defined. We will test far more test cases than the examples we have included in this assignment sheet.

Appendix I.

The `print_space()` method of the class `Diffusion` is not marked in this assignment. For convenience, I am providing the `print_space()` that was used to produce the sample output. If you use this method, make sure to place it within the class `Diffusion` scope.

```
  def print_space(self):
    for i in range(self.rows):
      for j in range(self.cols):
        print(f'{self.space[i][j]:1.4f}', end=' ')
      print()
```