```
1: Test Preview: Summary for yy4423 of q5
2: ---------------------------------------
3:
4:   Public Tests:
5:     Correctness tests:  2 / 2
6:
7: Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_autumn/python_cw1_yy4423.git
8: Commit ID: 6a8f9
```

```python
from bucket_fill import fill, load_image, show_image
import random
pattern = [
    (0, 0),
    (-1, 0),
    (-1000000000000000000000000000000000000000000000000, 1000000000000000000000000000000000000000000000000),
    (0.5, 0.5),
    ('x', 1),
    (1, 2),
    (5, 5),
    (25, 25),
    (10, 10),
    (100, 100),
    (4, 5),
    [5, 5],
    {1:5, 2:10}
]

def generate_test_image():
    n = random.randint(1, 25)

    image = [[random.randint(0, 1) for i in range(n)] for j in range(n)]
    print(image)
    return image

def test_pattern():
    image = generate_test_image() # load_image('test_image.txt')
    print(image)
    print("--------------------")
    print('Testing...')
    for i in pattern:
        try:
            filled_image = fill(image, seed_point=i)
            if filled_image == image:
                print('Test Passed and Keep Original Image')
            else:
                print('Test passed')
        except:
            print('Test failed')
    image = fill(image=image, seed_point=(-1, 7))

    print("-" * 25)
    print("After filling:")
    show_image(image)


if __name__ == '__main__':
    # This is just an example. Feel free to change this to whatever suits you.
    test_pattern()
```



```python
""" Coursework 1: Bucket Fill
"""

def load_image(filename):
    """ Load image from file made of 0 (unfilled pixels) and 1 (boundary pixels) and 2 (filled pixel)

    Example of content of filename:

0 0 0 0 1 1 0 0 0 0
0 0 1 1 0 0 1 1 0 0
0 1 1 0 0 1 0 1 1 0
1 1 0 0 1 0 1 0 1 1
1 0 0 1 0 0 1 0 0 1
1 0 0 1 0 0 1 0 0 1
1 1 0 1 0 0 1 0 1 1
0 1 1 0 1 1 0 1 1 0
0 0 1 1 0 0 1 1 0 0
0 0 0 0 1 1 0 0 0 0

    Args:
        filename (str) : path to file containing the image representation

    Returns:
        list : a 2D representation of the filled image, where
               0 represents an unfilled pixel,
               1 represents a boundary pixel
               2 represents a filled pixel
    """

    image = []
    with open(filename) as imagefile:
        for line in imagefile:
            if line.strip():
                row = list(map(int, line.strip().split()))
                image.append(row)
    return image


def stringify_image(image):
    """ Convert image representation into a human-friendly string representation

    Args:
        image (list) : list of lists of 0 (unfilled pixel), 1 (boundary pixel) and 2 (filled pixel)

    Returns:
        str : a human-friendly string representation of the image
    """

    if image is None:
        return ""

    # The variable "mapping" defines how to display each type of pixel.
    mapping = {
        0: " ",
        1: "*",
        2: "0"
    }

    image_str = ""
    if image:
        image_str += "+ " + "- " * len(image[0]) + "+\n"
    for row in image:
        image_str += "| "
        for pixel in row:
            image_str += mapping.get(pixel, "?") + " "
        image_str += "|"
        image_str += "\n"
    if image:
        image_str += "+ " + "- " * len(image[0]) + "+\n"

    return image_str


def show_image(image):
    """ Show image in terminal
```

```
    Args:
        image (list) : list of lists of 0 (unfilled pixel), 1 (boundary pixel) and 2 (filled pixel)
    """
    print(stringify_image(image))


def fill(image, seed_point):
    # Regulate the Input using if not + raise error
    if not isinstance(seed_point, tuple):
        raise TypeError("seed_point must be a tuple")
    if not len(seed_point) == 2:
        raise ValueError("seed_point must have two elements")
    if not isinstance(seed_point[0], int):
        raise TypeError("seed_point elements must be an integer")
    if not isinstance(seed_point[1], int):
        raise TypeError("seed_point elements must be an integer")
    if not seed_point[0] >= 0:
        raise ValueError("seed_point elements must be greater than or equal to 0")
    if not seed_point[1] >= 0:
        raise ValueError("seed_point elements must be greater than or equal to 0")

    # Make a copy of the image to avoid modifying the original
    filled_image = [row.copy() for row in image]

    # Extract the row and column from the seed point
    row, col = seed_point

    # Check if the seed point is valid, if not, return the original image
    if (not (0 <= row < len(filled_image)) or
        not (0 <= col < len(filled_image[0])) or
        filled_image[row][col] != 0):
        return filled_image

    def flood_fill(r, c):
        # Base conditions to stop the recursion
        if (r < 0 or r >= len(filled_image) or
            c < 0 or c >= len(filled_image[0]) or
            filled_image[r][c] != 0):
            return

        # Mark the current pixel as filled
        filled_image[r][c] = 2

        # Recursively fill neighboring pixels
        flood_fill(r + 1, c)  # Down
        flood_fill(r - 1, c)  # Up
        flood_fill(r, c + 1)  # Right
        flood_fill(r, c - 1)  # Left

    # Start the flood fill from the seed point
    flood_fill(row, col)

    return filled_image


def example_fill():
    image = [
    [1, 0, 0, 0, 1, 0, 0, 0, 1],
    [0, 1, 0, 1, 0, 1, 0, 1, 0],
    [0, 1, 1, 1, 0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0]]

    print("Before filling:")
    show_image(image)

    filled_image = fill(image = image, seed_point=(0, 1))

    print("-" * 25)
    print("After filling:")
    show_image(filled_image)


if __name__ == '__main__':
    example_fill()

```



```
0 0 0 0 1 1 1 1 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
0 0 1 0 1 0 0 0 1 0 1 0 0 0 0
0 0 1 0 1 0 0 0 1 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 0 0 0 0
0 0 1 1 0 0 1 0 0 0 1 1 0 0 0
0 1 0 0 0 1 0 0 0 0 1 0 1 0 0
1 0 0 1 0 0 0 0 0 1 0 0 0 1 0
1 0 0 0 1 1 1 1 1 0 0 0 1 0 1
1 0 0 0 0 0 0 0 0 0 0 0 1 0 1
0 1 0 0 0 0 0 0 0 0 0 1 1 0 1
0 0 1 1 1 1 1 1 1 1 1 0 0 1 0
```

```
 1: 0 1 1 1 1 1 1 1 1 1 1 0
 2: 1 0 0 0 0 0 0 0 0 0 0 1
 3: 1 0 0 1 0 0 0 1 0 0 1 1
 4: 1 0 1 0 1 0 1 0 1 0 1 1
 5: 1 0 0 1 0 0 0 1 0 0 1 1
 6: 1 0 0 0 0 0 0 0 0 0 1 1
 7: 1 0 1 1 1 1 1 1 1 0 1 1
 8: 1 0 1 0 0 0 0 0 1 0 1 1
 9: 1 0 0 1 0 0 0 1 0 0 1 1
10: 1 0 0 0 1 1 1 0 0 0 1 1
11: 0 1 1 1 1 1 1 1 1 1 1 0
```



```
1: 0 0 0 0 0 0 0 0
2: 0 0 0 0 0 0 0 0
3: 0 0 0 0 0 0 0 0
4: 1 1 1 1 1 1 1 1
5: 0 0 0 0 0 0 0 0
6: 0 0 0 0 0 0 0 0
7: 0 0 0 0 0 0 0 0
8: 0 0 0 0 0 0 0 0
```

```
1: -------- Test Output --------
2: Starting test.
3: Running public test script...
4: ==================================
5: BUCKET FILL TESTS
6: ==================================
7: ----------------------
8: <START> Correctness tests
9:
10: [PASS]: {#1} Standard test: wall in the middle (8x8), seed point in top half (Score: 1/1)
11:  |
12:  |  [PASS]: version 1/1 - {#1} Standard test: wall in the middle (8x8), seed point in
13:  |   top half
14:  |
15:  |---- (Score: 1/1)
16:
17: [PASS]: {#2} Standard test: wall in the middle (8x8), seed point in bottom half (Score: 1/1)
18:  |
19:  |  [PASS]: version 1/1 - {#2} Standard test: wall in the middle (8x8), seed point in
20:  |   bottom half
21:  |
22:  |---- (Score: 1/1)
23:
24: <END> Correctness tests (TOTAL: 2/2)
25: ----------------------
26:
27:
28: -------- Test Errors --------
29:
```

```
[
 {
  "name": "Correctness tests",
  "score": 2,
  "possible": 2
 }
]
```