Step 2.1

```
sited: 0
       0
eue:
```

```
sited: 0
       1
eue: 0
 Add 0 to the queue
 Mark 0 as visited
```

```
sited: 0
       1
eue:
 Remove 0 from the queue
 Print 0
```

```
sited: 0 C K
       1 1 1
eue: C K
 Add C and K to the queue
 Mark C and K as visited
```

```
sited: 0 C K
       1 1 1
eue: K
 Remove C from the queue
 Print 0 C
```

```
sited: 0 C K G
       1 1 1 1
eue: K G
 Add G to the queue
 Mark G as visited
```

```
Visited: 0 C K G
         1 1 1 1
Queue: G
1. Remove K from the queue
2. Print: 0 C K
```

```
Visited: 0 C K G
         1 1 1 1
Queue:
1. Remove G from the queue
2. Print 0 C K G
```

```
Visited: 0 C K G D
         1 1 1 1 1
Queue: D
1. Add D to the queue
2. Mark D as visited
```

```
Visited: 0 C K G D
         1 1 1 1 1
Queue:
1. Remove D from the queue
2. Print: 0 C K G D
```

```
Visited: 0 C K G D A I
         1 1 1 1 1 1 1
Queue: A I
1. Add A, I to the queue
2. Mark A, I as visited
```

```
Visited: 0 C K G D A I
         1 1 1 1 1 1 1
Queue: I
1. Remove A from the queue
2. Print: 0 C K G D A
```

```
Visited: 0 C K G D A I B
         1 1 1 1 1 1 1 1
Queue: I B
1. Add B to the queue
2. Mark B as visited
```

```
Visited: 0 C K G D A I B
         1 1 1 1 1 1 1 1
Queue: B
1. Remove I from the queue
2. Print: 0 C K G D A I
```

```
Visited: 0 C K G D A I B R
         1 1 1 1 1 1 1 1 1
Queue: B R
1. Add R to the queue
2. Mark R as visited
```

```
Visited: 0 C K G D A I B R
         1 1 1 1 1 1 1 1 1
Queue: R
1. Remove B from the queue
2. Print 0 C K G D A I B
```

```
Visited: 0 C K G D A I B R
         1 1 1 1 1 1 1 1 1
Queue:
1. Remove R from the queue
2. Print 0 C K G D A I B R
```

Step 2.2

```
class Solution:
    def hasPath(self, maze, start, destination):
        return self.bfs(maze, start, destination)

    def bfs(self, maze, start, destination):
        Row, Col = len(maze), len(maze[0])
        visited = [[False for _ in range(Col)] for _ in range(Row)]
        Q = [(start[0], start[1])]
        visited[start[0]][start[1]] = True
        while Q:
            r, c = Q.pop(0)
            if [r,c] == destination:
                return True
            for dr,dc in ((0,1), (1,0), (0,-1), (-1,0)):
                nr = r + dr
```

Step 2.1

```
            nc = c + dc
            while 0<= nr <Row and 0<= nc <Col and maze[nr][nc]==0:
                nr += dr
                nc += dc
            nr -= dr
            nc -= dc
            if visited[nr][nc] == False:
                visited[nr][nc] = True
                Q.append((nr, nc))
        return False


def main():
    maze = [[0, 0, 1, 0, 0],
            [0, 0, 0, 0, 0],
            [0, 0, 0, 1, 0],
            [1, 1, 0, 1, 1],
            [0, 0, 0, 0, 0]]

    start = [0, 4]
    destination = [4, 4]
    print(Solution().hasPath(maze, start, destination))


if __name__ == "__main__":
    main()
```

Output: