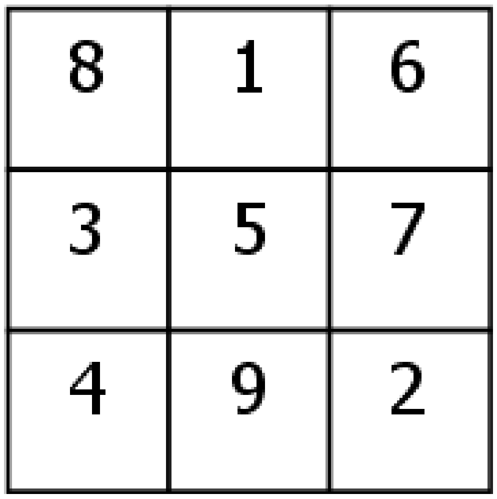
CIS 1051 Introduction to Programming in Python **Summer II 2014**

Lecture 4: July 17

Lecturer: Anwar Mamat

4.1 Magic Square

A magic square is an arrangement of distinct numbers (i.e. each number is used once), usually integers, in a square grid, where the numbers in each row, and in each column, and the numbers in the forward and backward main diagonals, all add up to the same number. For example: the numbers in each column, row, and diagonal of a 3*3 magic square add up to 15.



8	1	6
3	5	7
4	9	2

Figure 4.1: 3*3 Magic Square

The method to fill a magic square:

- Fill the central column of the first row with the number 1.
- After that, the fundamental movement for filling the squares is diagonally up and right, one step at a time.
- If a filled square is encountered, one moves vertically down one square instead, then continues as before.
- When an "up and to the right" move would leave the square, it is wrapped around to the last row or first column, respectively.

Listing 1: Magic Square Example

```python
def print_square(square,n):
    for i in range(n):
        for j in range(n):
            print(square[i][j],end="\t")
        print("")
def magic_square(n):
    square = [[0 for i in range(n)] for i in range(n)]
    #print(square)
    row = 0
    col = n // 2
    num = 1
    while( num <= n*n):
        if(square[row][col] == 0):
            square[row][col] = num
            row -= 1
            col +=1
            if(row < 0): row = n-1
            if(col >= n): col = 0
            num += 1
        else:
            row +=1
            col -=1
            if(row > n-1): row = 0
            if(col < 0): col = n-1
            row += 1
            if(row > n-1): row = 0
    print_square(square,n)
def main():
    magic_square(5)
main()
```

Listing 2: 5*5 Magic Square Output

```
17      24      1       8       15
23      5       7       14      16
4       6       13      20      22
10      12      19      21      3
11      18      25      2       9
```

4.2 Functions

Listing 3: Function Examples

```python
def test_function(a,b=20, c=30):
    print(a)
    print(b,c)

def test_function2(*args):
    l = len(args)
    print(l)
    print(args)
    a = args[0]
    print(a)
    for i in args:
        print(i)
def maxItem(*items):
    return max(items)

def test_function3(**kwargs):
    for i in kwargs:
        print(i)
        print(kwargs[i])


def inclusive_range(start,end,step):
    i = start
    while( i <= end):
        yield i
        i += step

def inclusive_range2(*args):
    numArgs = len(args)
    if(numArgs< 1):
        raise TypeError("Requires␣at␣least␣one␣argument")
    elif(numArgs == 1):
        start = 0
        end = args[0]
        step =1
    elif(numArgs ==2):
        start = args[0]
        end = args[1]
        step =1
    elif(numArgs == 3):
        start = args[0]
        end = args[1]
        step =args[2]
    elif(numArgs > 3):
        raise TypeError("Requires␣no␣more␣than␣3␣arguments")

    i = start
    while(i <= end):
        yield i
        i += step


def main():
    test_function(10,100)

    test_function2(10,20,30,40)

    print(maxItem(1,2,3))
    print(maxItem(20,30))
    test_function3(one=100,two=200,four=400)

    for i in inclusive_range2(10,20,2):
        print(i,end=",")
main()
```

4.3 Recursion

Listing 4: Function Examples

```
def factorial(n):
    print(n,end=",")
    if(n==1):
        return 1
    else:
        return n * factorial(n-1)
def main():
    n = 5
    print(factorial(n))
main()
```

4.4 Backtracking

4.4.1 Rat in a Maze

In a maze matrix, 0 represents path, and 1 represents wall. This program finds a path from [left, top] to [right,bottom] in a raze matrix.

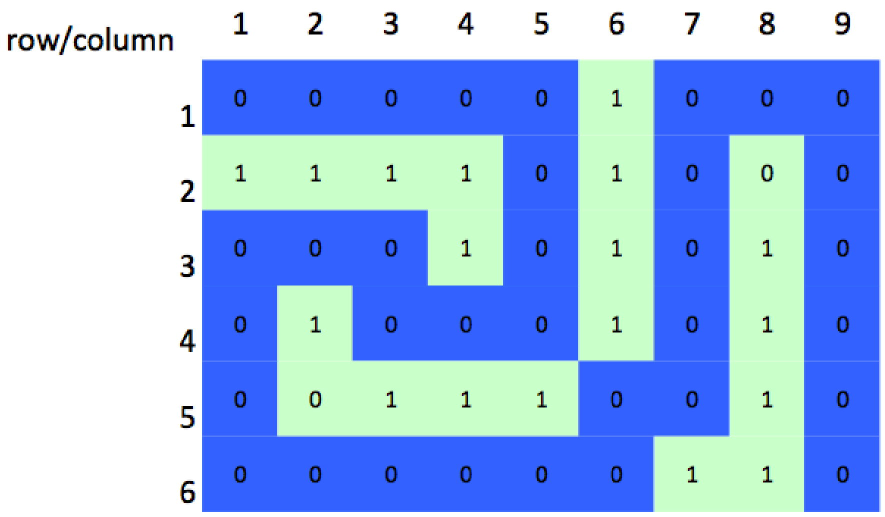


Figure 4.2: Maze

Listing 5: Maze Examples

```python
WIDTH = 9
HEIGHT = 6
maze = HEIGHT*[WIDTH*[0]]
maze[0] = [0,0,0,0,0,1,0,0,0]
maze[1] = [1,1,1,1,0,1,0,0,0]
maze[2] = [0,0,0,1,0,1,0,1,0]
maze[3] = [0,1,0,0,0,1,0,1,0]
maze[4] = [0,0,1,1,1,0,0,1,0]
maze[5] = [0,0,0,0,0,0,1,1,0]

visited = [ [0 for i in range (WIDTH)] for j in range(HEIGHT)]
solution = [ [0 for i in range (WIDTH)] for j in range(HEIGHT)]

def printMaze():
    print("  ",end="")
    for j in range(WIDTH):
        print(j+1,end="")
    print("\n-----------------------------");
    for i in range(HEIGHT):
        print(i+1,end='|');
        for j in range(WIDTH):
            print(maze[i][j],end="")
        print();
    print("\n-----------------------------\n");

def printSolution():

    print(" ",end="")
    for col in range( WIDTH):
        print(col + 1,end=" ");
    print("\n--------------------------------\n");
    for row in range(HEIGHT):
        print(row+1,end="");
        for col in range(WIDTH):
            if(solution[row][col] == 1):
                print( "* ",end="");#sol[i][j];
            else:
                print("  ",end="");
        print()

def isSafe(row, col):
    # if (row,col outside maze) return false
    if(row >= 0 and row < HEIGHT and col >= 0 and col < WIDTH and
      maze[row][col] == 0 and visited[row][col] == 0):
        return True
    else:
        return False
def solve(row, col):
    if(row == HEIGHT-1 and col == WIDTH-1):
        solution[row][col] = 1
        return True
    if(isSafe(row,col)):
        solution[row][col] = 1
        visited[row][col] = 1

        if(solve(row, col+1) == True):
            return True

        if(solve(row+1, col) == True):
            return True

        if(solve(row-1, col) == True):
            return True;

        if(solve(row, col-1) == True):
            return True

       #backtrack to the previous cell, start again.
        visited[row][col] = 1
        solution[row][col] = 0
    else:
        return False

def main():
    printMaze()
    if(solve(0,0)):
        printSolution()
    else:
        print("No solution")
main()
```

Listing 6: Maze Output

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

-------------------------------
Solution Path:
 1  2  3  4  5  6  7  8  9
-------------------------------
1*  *  *  *  *
2            *     *  *  *
3*  *  *     *     *     *
4*     *  *  *     *     *
5*  *           *  *     *
6   *  *  *  *  *        *
```