Learning Goals

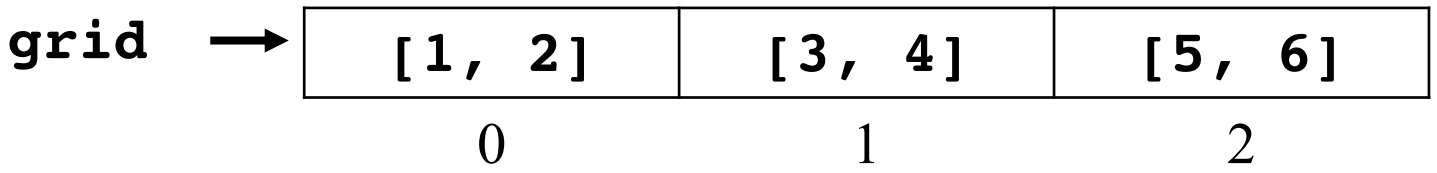
1. Understand 2-dimensional lists (list of lists)
2. Get experience programming with 2-D lists

Review:
2-Dimensional Lists
(List of lists)

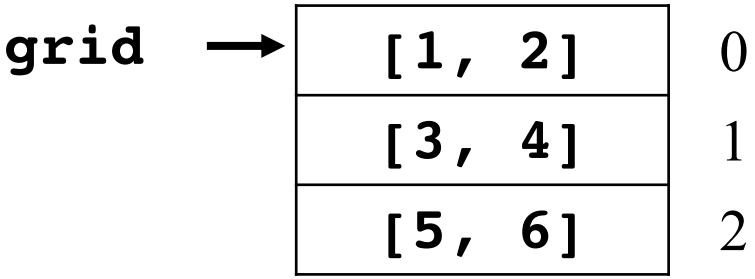
Recall, the 2-Dimensional List

- A 2-dimensional list is a "list of lists"
  - Each element of "outer" list is just another list
  - Can think of this like a grid
- Example:

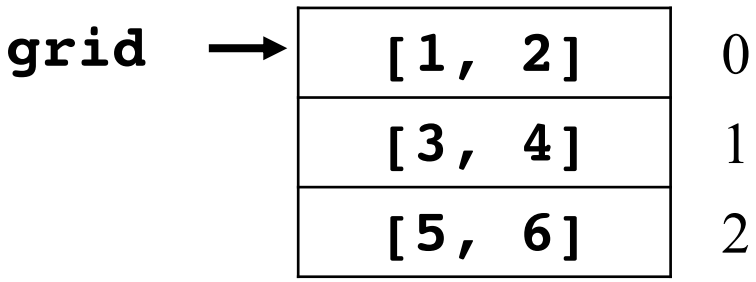
```
grid = [[1, 2], [3, 4], [5, 6]]
```



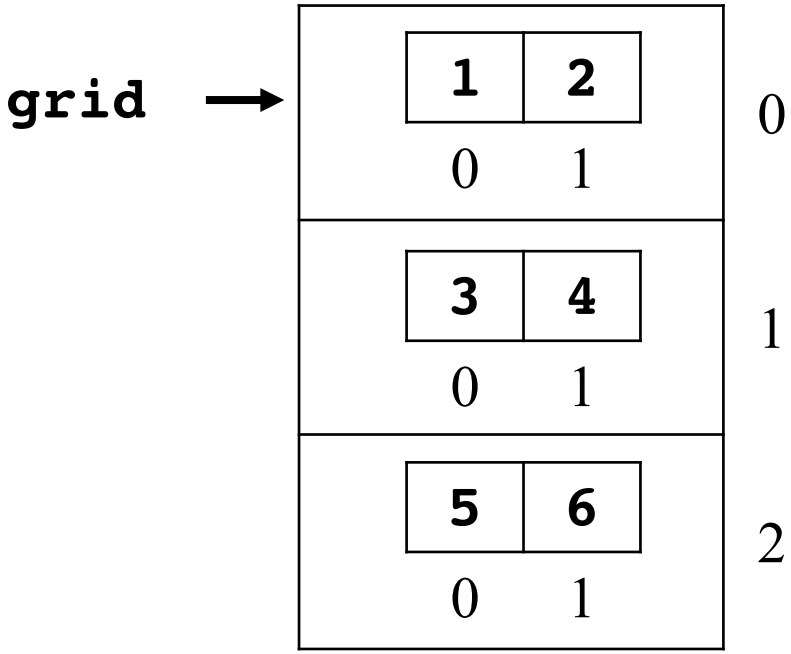
- Can be easier to think of like this:



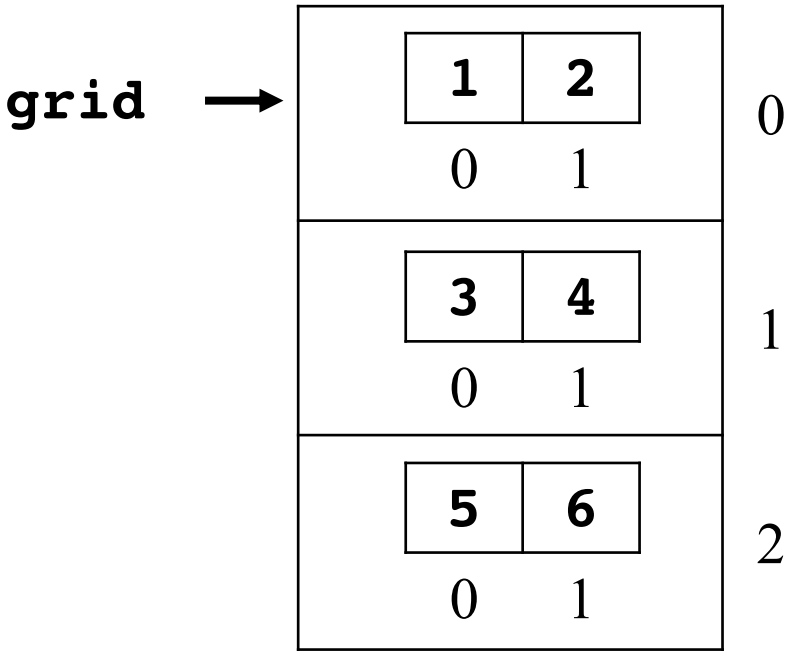
2-Dimensional List



- Um, can you zoom in on that…



2-Dimensional List

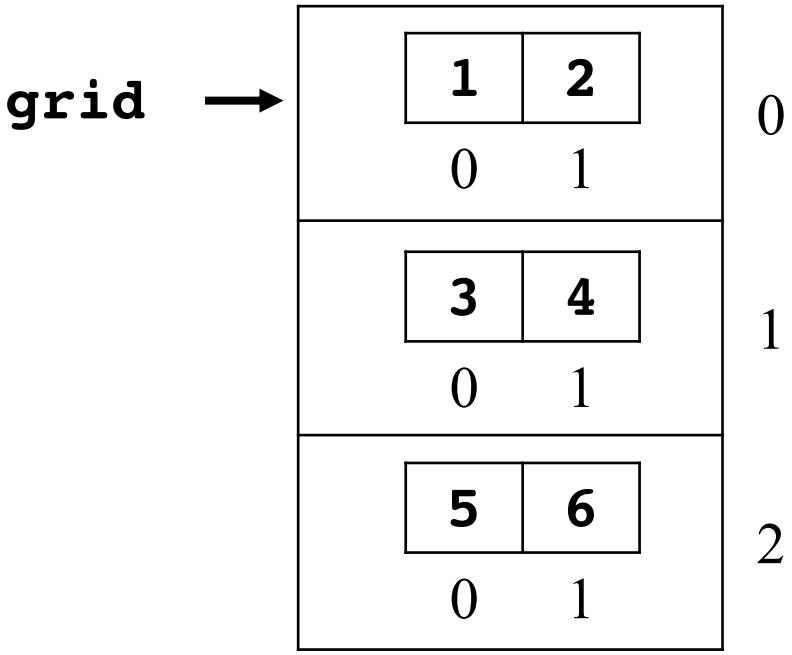


grid[0][0] 1	grid[0][1] 2
grid[1][0] 3	grid[1][1] 4
grid[2][0] 5	grid[2][1] 6

- To access elements, specify index in "outer" list, then index in "inner" list

```
grid[0][0]  →  1
grid[1][0]  →  3
grid[2][1]  →  6
```

2-Dimensional List



- So what if I only specify one index?

```
grid[0]    →  [1, 2]
grid[1]    →  [3, 4]
grid[2]    →  [5, 6]
```

- Remember, **`grid`** is just a list of lists
  - Elements of "outer" list are just lists

2-D lists as parameters

Swapping Elements in a Grid

```
def swap(grid, row1, col1, row2, col2):
    temp = grid[row1][col1]
    grid[row1][col1] = grid[row2][col2]
    grid[row2][col2] = temp


def main():
    my_grid = [[10, 20, 30], [40, 50, 60]]
    swap(my_grid, 0, 1, 1, 2)
    print(my_grid)
```

Output:

```
[[10, 60, 30], [40, 50, 20]]
```

Time to get funky!

Getting Funky With Lists

- Do the inner lists all have to be the same size?
  - No! Just be careful if they are not.

```
jagged = [[1, 2, 3], [4], [5, 6]]
jagged[0]      →  [1, 2, 3]
jagged[1]      →  [4]
jagged[2]      →  [5, 6]
```

- Can I have more than two dimensions?
  - Sure! You can have as many as you like (within reason).

```
cube = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
cube[0]        →  [[1, 2], [3, 4]]
cube[0][1]     →  [3, 4]
cube[0][1][0]  →  3
```

Looping Through a List of Lists

```
def main():
    grid = [[10, 20], [40], [70, 80, 100]]
    rows = len(grid)
    for i in range(rows):
        cols = len(grid[i])
        for j in range(cols):
            print("grid[" + str(i) + "][" + str(j)
                  + "] = " + str(grid[i][j]))
```

Output:

```
grid[0][0] = 10
grid[0][1] = 20
grid[1][0] = 40
grid[2][0] = 70
grid[2][1] = 80
grid[2][2] = 100
```

Simplified With a True Grid

```
def main():
    grid = [[1, 2], [10, 11], [20, 21]]
    rows = len(grid)
    cols = len(grid[0])
    for i in range(rows):
        for j in range(cols):
            print("grid[" + str(i) + "][" + str(j)
                  + "] = " + str(grid[i][j]))
```

Output:

```
grid[0][0] = 1
grid[0][1] = 2
grid[1][0] = 10
grid[1][1] = 11
grid[2][0] = 20
grid[2][1] = 21
```

Using For-Each With 2-D List

```
def main():
    grid = [[10, 20], [40], [70, 80, 100]]
    for row in grid:
        for elem in row:
            print(elem)
```

Output:

```
10
20
40
70
80
100
```

Creating a 2-D List

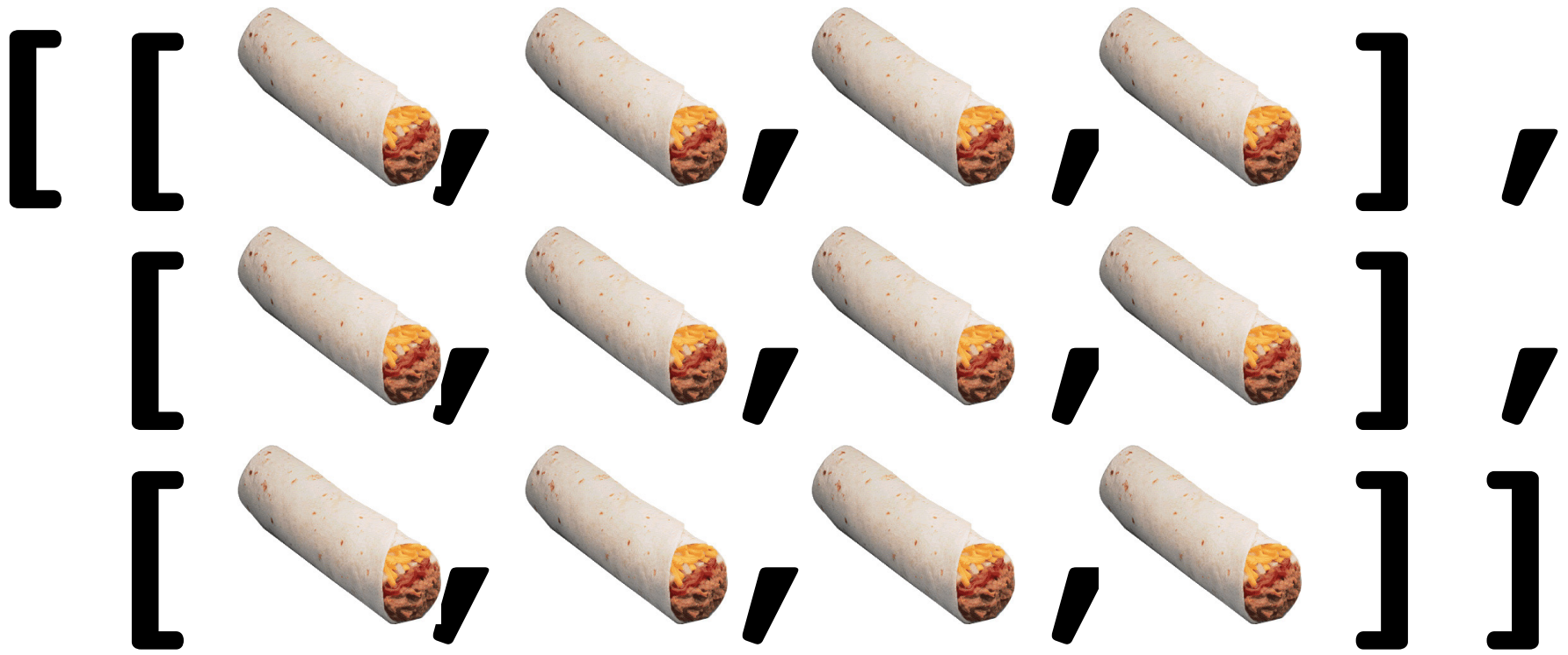
```
def create_grid(rows, cols, value):
    grid = []                       # Create empty grid
    for y in range(rows):           # Make rows one by one
        row = []
        for x in range(cols):       # Build up each row
            row.append(value)       # by appending to list

        grid.append(row)            # Append row (list)
                                    # onto grid
    return grid
```

Console:

```
>>> create_grid(2, 4, 1)
[[1, 1, 1, 1], [1, 1, 1, 1]]
>>> create_grid(3, 2, 5)
[[5, 5], [5, 5], [5, 5]]
```

Learning Goals

1. Understand 2-dimensional lists (list of lists)
2. Get experience programming with 2-D lists



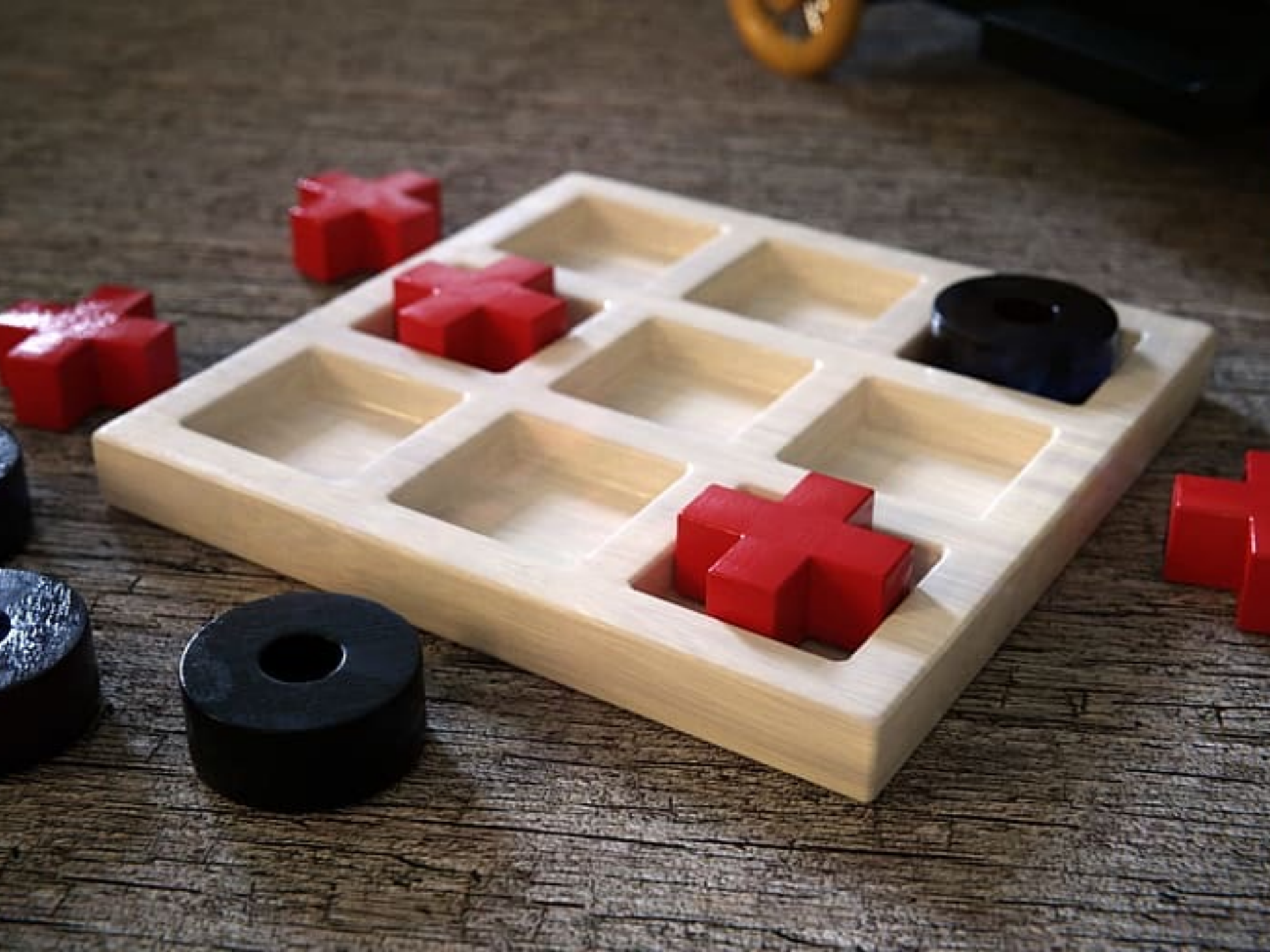
Learning Goals

1. Understand 2-dimensional lists (list of lists)
2. Get experience programming with 2-D lists

Putting it all together:
tictactoe.py

(This program give you practice with a lot of concepts!)

Added bonus: helpful for Assignment #4



The Main Program

```
SIZE = 3    # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

Creating the Board

```
def create_empty_board(n):
    grid = []                   # Create empty grid
    for y in range(n):          # Create rows one at a time
        row = []
        for x in range(n):      # Build up each row by
            row.append(None)    # appending to a list

        grid.append(row)        # Append the row (list)
                                # onto grid
     return grid
```

Console:

```
>>> create_empty_board(3)
[[None, None, None], [None, None, None], [None, None, None]]
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

Printing the Board

```
def print_board(board):
    rows = len(board)      # Could use SIZE, but wanted to
    cols = len(board[0])   # show general way to do this
     for y in range(rows):
         for x in range(cols):
             symbol = board[y][x]
             if not symbol:              # Print space if
                 symbol = " "            # symbol is None
             print(symbol, end="")
             if x < SIZE - 1:            # Print | marker if
                print(" | ", end="") # not at end of line
             else:
                 print("")               # Print end of line
         if y < SIZE - 1:                # Print row marker
            print_row_separator(cols)
```

Printing the Board

```
def print_board(board):
    rows = len(board)      # Could use SIZE, but wanted to
    cols = len(board[0])   # show general way to do this
     for y in range(rows):
         for x in range(cols):
             symbol = board[y][x]
             if not symbol:              # Print space if
                 symbol = " "            # symbol is None
             print(symbol, end="")
             if x < SIZE - 1:            # Print | marker if
                print(" | ", end="") # not at end of line
             else:
                 print("")               # Print end of line
         if y < SIZE - 1:                # Print row marker
            print_row_separator(cols)
```

```
print_board([['X',None,'O'], [None,'O',None], ['X',None,'X']])
```

Printing the Row Separator

```
def print_row_separator(columns):
    print("--+", end="")
    for i in range(1, columns - 1):
        print("---+", end="")
    print("--")
```

Console:

```
>>> print_row_separator(3)
--+---+--
```

Printing the Board

```
def print_board(board):
    rows = len(board)      # Could use SIZE, but wanted to
    cols = len(board[0])   # show general way to do this
     for y in range(rows):
         for x in range(cols):
             symbol = board[y][x]
             if not symbol:              # Print space if
                 symbol = " "            # symbol is None
             print(symbol, end="")
             if x < SIZE - 1:            # Print | marker if
                print(" | ", end="") # not at end of line
             else:
                 print("")               # Print end of line
         if y < SIZE - 1:                # Print row marker
            print_row_separator(cols)
```

```
X |   | O
--+---+--
  | O |
--+---+--
X |   | X
```

```
print_board([['X',None,'O'], [None,'O',None], ['X',None,'X']])
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

Getting a Player's Move

```
def player_turn(board, symbol):
    valid_move = False
    while not valid_move:
        print()   # Blank line
        print(symbol + "'s move")
        row = int(input("Row: "))
        col = int(input("Col: "))
        # Make sure move is on board and in empty space
        if row < 0 or row >= SIZE or col < 0 or col >= SIZE \
                or board[row][col]:
            print("Invalid move.  Try again.")
        else:
            board[row][col] = symbol   # Record valid move
            valid_move = True
```

```
>>> grid = [['X',None,'O'],[None,'O',None],['X',None,'X']]
>>> player_turn(grid, 'O')

O's move
Row: 0
Col: 1
>>> grid
[['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

Checking For a Winner

```
def check_winner(board):
    for row in range(SIZE):                 # Check rows
        winner = check_row(board, row)
        if winner:
            return winner
    for col in range(SIZE):                 # Check columns
        winner = check_column(board, col)
        if winner:
            return winner
    # Check diagonals
    winner = check_down_diagonal(board)
    if winner:
        return winner
    winner = check_up_diagonal(board)
    return winner   # Could be None if no winner
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> check_winner(grid)
```

Checking For a Winner

```
def check_winner(board):
    for row in range(SIZE):                 # Check rows
        winner = check_row(board, row)
        if winner:
            return winner
    for col in range(SIZE):                 # Check columns
        winner = check_column(board, col)
        if winner:
            return winner
    # Check diagonals
    winner = check_down_diagonal(board)
    if winner:
        return winner
    winner = check_up_diagonal(board)
    return winner   # Could be None if no winner
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> check_winner(grid)
```

Checking a Row for a Winner

```
def check_row(board, row):
    symbol = board[row][0]
    for col in range(1, SIZE):
        # If we find non-matching symbol then no winner
        if board[row][col] != symbol:
            return None
    return symbol   # Only get here if all symbols match
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> print(check_row(grid, 0))
None
```

Checking For a Winner

```
def check_winner(board):
    for row in range(SIZE):                 # Check rows
        winner = check_row(board, row)
        if winner:
            return winner
    for col in range(SIZE):                 # Check columns
        winner = check_column(board, col)
        if winner:
            return winner
    # Check diagonals
    winner = check_down_diagonal(board)
    if winner:
        return winner
    winner = check_up_diagonal(board)
    return winner   # Could be None if no winner
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> check_winner(grid)
```

Checking a Column for a Winner

```
def check_column(board, col):
    symbol = board[0][col]
    for row in range(1, SIZE):
        # If we find non-matching symbol then no winner
        if board[row][col] != symbol:
            return None
    return symbol   # Only get here if all symbols match
```

```
>>> grid = [['X',None,'O'],[None,'O',None],['X',None,'X']]
>>> print(check_column(grid, 0))
None
```

Checking For a Winner

```
def check_winner(board):
    for row in range(SIZE):                 # Check rows
        winner = check_row(board, row)
        if winner:
            return winner
    for col in range(SIZE):                 # Check columns
        winner = check_column(board, col)
        if winner:
            return winner
    # Check diagonals
    winner = check_down_diagonal(board)
    if winner:
        return winner
    winner = check_up_diagonal(board)
    return winner   # Could be None if no winner
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> check_winner(grid)
```

Checking Diagonal Down for a Winner

```
def check_down_diagonal(board):
    symbol = board[0][0]
    for row in range(1, SIZE):
        if board[row][row] != symbol:
            return None
    return symbol   # Only get here if all symbols match
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> print(check_down_diagonal(grid))
None
```

Checking For a Winner

```
def check_winner(board):
    for row in range(SIZE):                 # Check rows
        winner = check_row(board, row)
        if winner:
            return winner
    for col in range(SIZE):                 # Check columns
        winner = check_column(board, col)
        if winner:
            return winner
    # Check diagonals
    winner = check_down_diagonal(board)
    if winner:
        return winner
    winner = check_up_diagonal(board)
    return winner    # Could be None if no winner
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> check_winner(grid)
```

Checking Diagonal Up for a Winner

```
def check_up_diagonal(board):
    symbol = board[0][SIZE - 1]
    for row in range(1, SIZE):
        if board[row][SIZE - 1 - row] != symbol:
            return None
    return symbol   # Only get here if all symbols match
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> print(check_up_diagonal(grid))
None
```

Checking For a Winner

```
def check_winner(board):
    for row in range(SIZE):                 # Check rows
        winner = check_row(board, row)
        if winner:
            return winner
    for col in range(SIZE):                 # Check columns
        winner = check_column(board, col)
        if winner:
            return winner
    # Check diagonals
    winner = check_down_diagonal(board)
    if winner:
        return winner
    winner = check_up_diagonal(board)
    return winner   # Could be None if no winner
```

```
>>> grid = [['X', 'O', 'O'], [None, 'O', None], ['X', None, 'X']]
>>> check_winner(grid)
None
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```

Flipping the Player Turn

```
def flip_turn(symbol):
    if symbol == 'X':
        return 'O'
    else:
        return 'X'
```

The Main Program

```
SIZE = 3     # The board used will be SIZE x SIZE

def main():
    winner = None
    board = create_empty_board(SIZE)
    player = 'X'                    # Player X goes first
    num_moves = 0

    while winner == None:           # Take turns until a winner
        print_board(board)
        player_turn(board, player)
        num_moves += 1              # Keep track of total moves
        winner = check_winner(board)
        if not winner:
            if num_moves == SIZE ** 2: # If all spaces full
                winner = "No one"      # then no winner.
        player = flip_turn(player)

    print_board(board)
    print(winner + " won!")
```