

# Tictactoe Game

Making Game with Python (1)

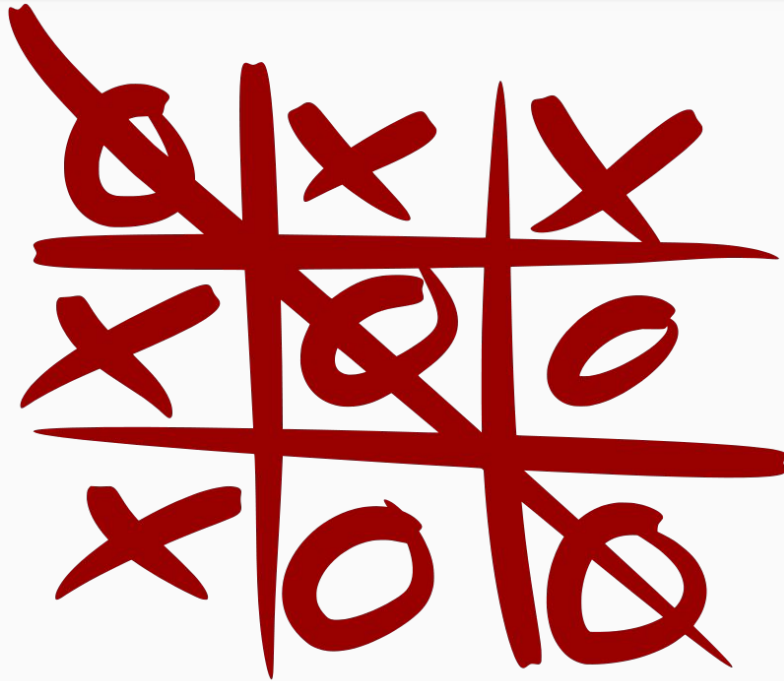
Zhihong (John) Zeng & Andrew Zeng



# Agenda

- Introduction
- Numpy (multi-dimensional arrays)
- Flowchart
- function definition and test
  - create\_board, input\_player\_selection, who\_go\_first, make\_move
  - game\_won, column\_won, row\_won, diag\_won
  - Get\_available\_move, get\_player\_move, get\_random\_move
- Main function

# Introduction



# Numpy

- NumPy is a Python library, adding support for large, multi-dimensional arrays, along with a large collection of high-level mathematical functions
- Installation:
  - Find the python executable path:
    - `import sys`
    - `print(sys.executable)`
  - Pip install
    - Go to the path
    - `python -m pip install numpy`
  - Test
    - `Import numpy`

# Numpy (cont): create 2D arrays

- import numpy as np
- a = np.full((height, width), value)
- Size of array: a.shape (height, width)
- Exercise:

- import numpy as np
- a = np.full((3,3), 'x ')
- print(a)
- print(a.shape)
- c = np.arange(6)
- print(c, c.shape)
- d = c.reshape((2,3))
- print(d)
- print(d.shape)
- d[1,1]

```
A= array([[ 0,  1,  2],  
         [ 3,  4,  5],  
         [ 6,  7,  8],  
         [ 9, 10, 11]])
```

```
print(A)
```

```
A.shape  
# (4,3): height=4, width=3
```

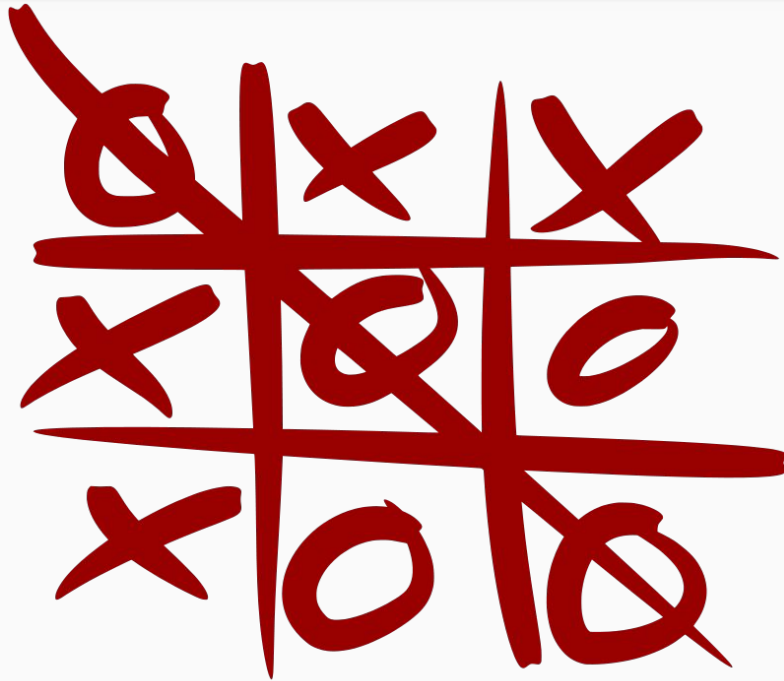
```
print(A[2,1])  
# 7
```

# Numpy (cont): access

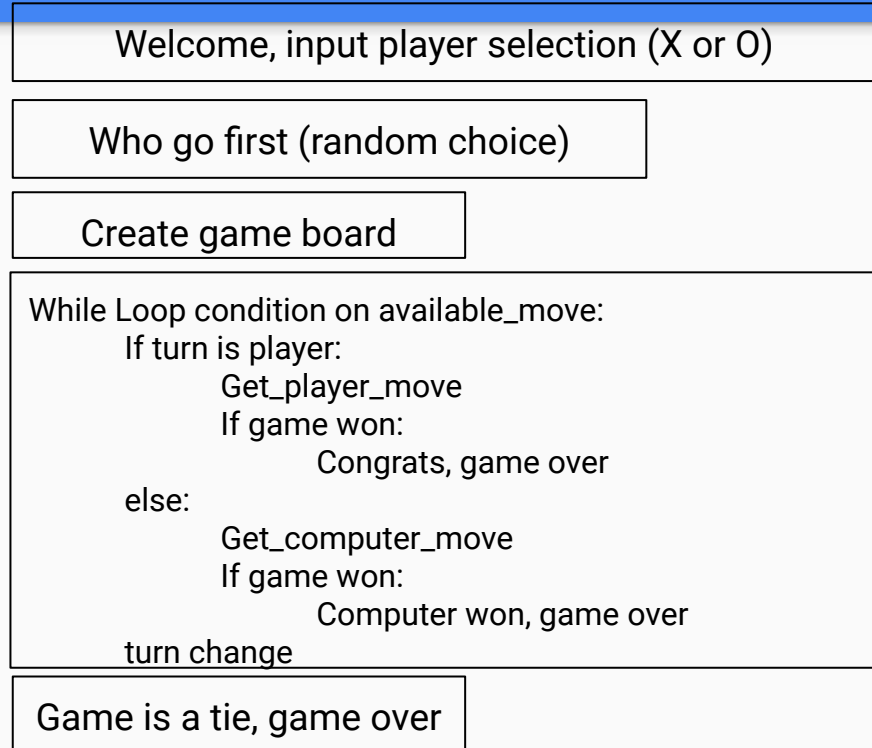
- `array[y_index, x_index]`
- Exercise:
  - `a = np.arange(6).reshape((2,3))`
  - `print(a)`
  - `print(a[0,1])`
  - `a[0,1] = 9`
  - `print(a)`
- For loop:
- Exercise:

```
for row in a:  
    print(row)  
    for x in row:  
        print(x)
```

# Tictactoe



# Flowchart





# Debug: function definition and test

tictactoe.py: Function definition

```
def game_won():  
    ....
```

tictactoe\_test.py

```
from tictactoe import *  
print(game_won())  
assert create_board==True, 'Fail'
```



If tictactoe\_test.py fails, go to check code

# Assertion

- Assertion: boolean expression to check if the condition is true.
  - The expression is True: do nothing
  - The expression is False: the program stop and throw an error
- Syntax:
  - `assert <condition>, <error message>`
  - Exercise:
    - `a = 1`
    - `assert a==1, 'First check: Wrong'`
    - `assert a==2, 'Second check: Wrong'`

# Create game board

```
def create_board(size, value=' '):  
    board = np.full((size,size), value)  
    return board
```

```
print('\n----Test create_board\n')  
print(create_board(3, ' '))  
print(create_board(2, 5))  
print(create_board(4))
```

# input\_player\_selection

```
def input_player_selection():  
    letter = "  
    while letter not in ['X', 'O']:  
        letter = input('Do you want to be X or O?')  
        letter = letter.upper()  
    if letter == 'X':  
        return ['X', 'O']  
    else:  
        return ['O', 'X']
```

```
print('\n----test input_player_letter\n')  
print(input_player_selection())
```

# Who go first

```
def who_go_first():  
    ans = random.randint(0,1)  
    if ans:  
        return 'computer'  
    else:  
        return 'player'
```

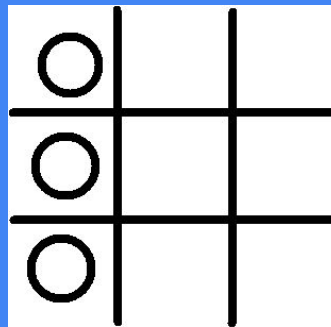
```
print('\n----test who_go_first\n')  
print(who_go_first())  
print(who_go_first())  
print(who_go_first())
```

# Make move

```
def make_move(board, letter, position):  
    board[position[0], position[1]] = letter
```

```
print('\n----test make_move\n')  
board = create_board(3)  
make_move(board, 'X', (0, 0))  
print(board)  
make_move(board, 'O', (2, 2))  
print(board)
```

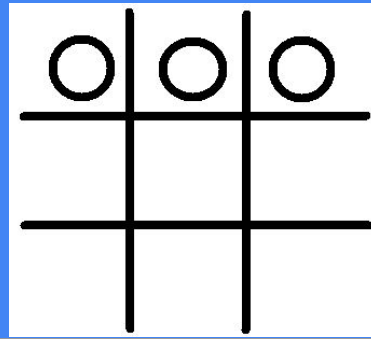
# Check column won



```
def column_won(board, letter):  
    height, width = board.shape  
    for x in range(width):  
        won = True  
        for y in range(height):  
            if board[y, x] != letter:  
                won = False  
                break  
        if won:  
            return True  
    return False
```

```
print('\n----test column_won\n')  
board = create_board(3)  
print(board)  
ans = column_won(board, 'X')  
print(ans)  
board[0, 0] = 'X'  
board[1, 0] = 'X'  
board[2, 0] = 'X'  
print(board)  
ans = column_won(board, 'X')  
print(ans)
```

# Check row won

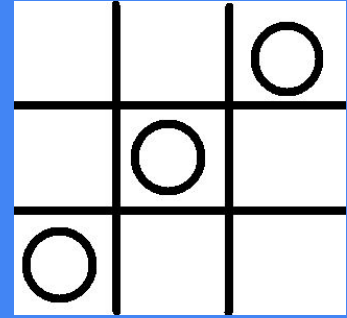
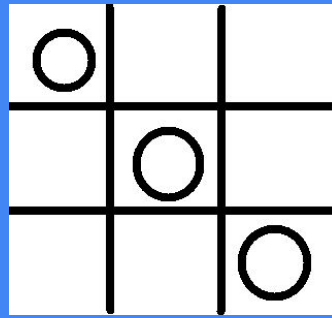


```
def row_won(board, letter):  
    height, width = board.shape  
    for y in range(height):  
        won = True  
        for x in range(width):  
            if board[y, x] != letter:  
                won = False  
                break  
        if won:  
            return True  
    return False
```

```
print('\n----test row_won\n')  
board = create_board(3)  
print(board)  
ans = row_won(board, 'X')  
assert not ans, 'Fail'  
board[1, 0] = 'X'  
board[1, 1] = 'X'  
board[1, 2] = 'X'  
print(board)  
ans = row_won(board, 'X')  
assert ans, 'Fail'
```



# Check diagonal won



```
def diag_won(board, letter):
    height, width = board.shape
    won = True
    for y in range(height):
        if board[y, y] != letter:
            won = False
            break
    if won:
        return True

    won = True
    for y in range(height):
        if board[y, height-1-y] != letter:
            won = False
            break

    return won
```

```
print('\n----test diag_won\n')
board = create_board(3)
print(board)
ans = diag_won(board, 'X')
assert not ans, 'Fail'
board[0, 0] = 'X'
board[1, 1] = 'X'
board[2, 2] = 'X'
print(board)
ans = diag_won(board, 'X')
assert ans, 'Fail'
board = create_board(3)
board[0, 2] = 'X'
board[1, 1] = 'X'
board[2, 0] = 'X'
print(board)
ans = diag_won(board, 'X')
assert ans, 'Fail'
```

# Check game won

```
def game_won(board, letter):  
    return column_won(board, letter) \  
        or row_won(board, letter) \  
        or diag_won(board, letter)
```

```
print('\n-----test game_won\n')  
board = create_board(3)  
ans = game_won(board, 'X')  
assert not ans, 'Fail'  
board[0, 2] = 'X'  
board[1, 2] = 'X'  
board[2, 2] = 'X'  
ans = game_won(board, 'X')  
assert ans, 'Fail'  
print('\n==== Success====\n')
```

# Get available move

```
def get_available_move(board):  
    height, width = board.shape  
    move = []  
    for y in range(height):  
        for x in range(width):  
            if board[y, x] == '':  
                move.append([y, x])  
    return move
```

```
print('\n----test get_available_move\n')  
board = create_board(3)  
board[0, 0] = 'X'  
board[1, 1] = 'O'  
board[2, 2] = 'O'  
ans = get_available_move(board)  
print(board)  
print('available_move: {}'.format(ans))
```

# Get random move

```
def get_random_move(board, letter):  
    print('computer move')  
    available_move = get_available_move(board)  
    move = random.choice(available_move)  
    make_move(board, letter, move)
```

```
print('\n-----test get_random_move\n')  
board = create_board(3)  
get_random_move(board, 'O')  
print(board)  
board = create_board(3)  
get_random_move(board, 'O')  
print(board)
```

# Get player move

```
def get_player_move(board, letter):  
    height, width = board.shape  
    available_move = get_available_move(board)  
    move = (-1, -1)  
    while move not in available_move:  
        ans = input(f'What is your next move (0-{height-1},  
0-{width-1})? ')  
        move = ans.split(',')  
        move = [int(move[0]), int(move[1])]  
    make_move(board, letter, move)
```

```
print('\n----test get_player_move\n')  
board = create_board(3)  
print(board)  
get_player_move(board, 'X')  
print(board)
```

A = '1,2'

A = A.split(',')      # ['1', '2']

# Main function

```
def tictactoe(size=3):  
    print('Welcome Tic-Tac-Toe!')  
    player, computer = input_player_selection()  
    print(f'You are {player}, and computer is {computer}')  
    first = who_go_first()  
    print(f'{first} go first')  
    board = create_board(size, ' ')  
    turn = first
```

# Main function (cont)

```
while get_available_move(board):
    if turn == 'player':
        get_player_move(board, player)
        print(board)
        turn = 'computer'
    if game_won(board, player):
        print('Congratulations. You won the game')
        return
    else:
        get_random_move(board, computer)
        print(board)
        turn = 'player'
    if game_won(board, computer):
        print('The computer won the game')
        return
print('The game is a tie')
```

# program entry

```
if __name__ == '__main__':  
    tictactoe()
```



# Shallow copy and deep copy

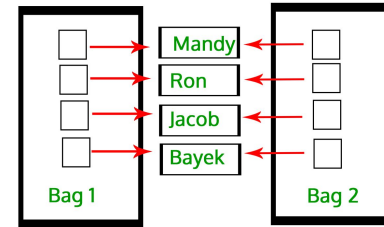
```
import numpy as np
A = np.full((3,3), ' ')
B = A
B[0,0] = 'X'
print(B)
print(A)
```

```
import copy
```

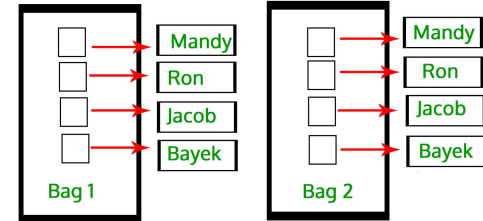
```
A = np.full((3,3), ' ')
B = copy.deepcopy(A)
B[0,0] = 'X'
print(B)
print(A)
```



Shallow Copy

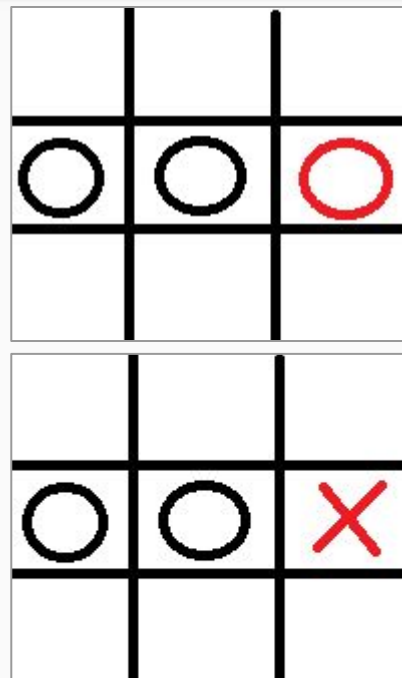


Deep Copy



# Get smart move

```
def get_win_defense_move(board, letter, available_move, win_defense_flag):  
    if win_defense_flag: # win check  
        target = letter  
    else: # defense check  
        if letter == 'O':  
            target = 'X'  
        else:  
            target = 'O'  
    for move in available_move:  
        board_clone = copy.deepcopy(board)  
        make_move(board_clone, target, move)  
        if game_won(board_clone, target):  
            make_move(board, letter, move)  
            return True  
    return False
```

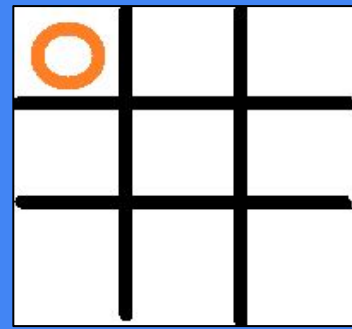
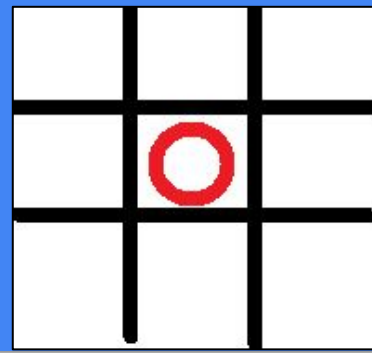


# Get smart move

```
def get_smart_move(board, letter):  
    available_move = get_available_move(board)  
    win_move = get_win_defense_move(board, letter, available_move, True)  
    if win_move:  
        return True  
    return get_win_defense_move(board, letter, available_move, False)
```

```
print('\n-----test get_smart_move\n')  
board = create_board(3)  
board[1, 0:2] = ['X', 'X']  
print(board)  
get_smart_move(board, 'X')  
print(board)  
assert game_won(board, 'X'), 'Fail'  
board = create_board(3)  
board[1, 0:2] = ['O', 'O']  
print(board)  
get_smart_move(board, 'X')  
print(board)  
assert not game_won(board, 'X'), 'Fail'  
print('\n==== Success====\n')
```

# Get preferred random move



```
def get_preferred_random_move(board, letter):
    available_move = get_available_move(board)
    height, width = board.shape
    preferred_move = [[int(height/2), int(width/2)], [0, 0], \
                      [0, width-1], [height-1, 0], [height-1, width-1]]
    move = None
    for p in preferred_move:
        if p in available_move:
            move = p
            break
    if move is None:
        move = random.choice(available_move)
    make_move(board, letter, move)
```

```
board = create_board(3)
get_preferred_random_move(board, 'X')
print(board)
get_preferred_random_move(board, 'X')
print(board)
print('\n==== Success====\n')
```

# Main function

```
def tictactoe_smart():  
    print('Welcom to Tic-Tac-Toe!')  
    player, computer = input_player_selection()  
    print('You are {}, and computer is {}'.format(player, computer))  
    first = who_go_first()  
    print('{} go first'.format(first))  
    board = create_board(3)  
    turn = first
```

# Main function (cont)

```
while get_available_move(board):
    if turn == 'player':
        get_player_move(board, player)
        print(board)
        turn = 'computer'
    if game_won(board, player):
        print('Congratulations. You won the game')
        return
    else:
        if not get_smart_move(board, computer):
            get_prefered_random_move(board, computer)
        print(board)
        turn = 'player'
    if game_won(board, computer):
        print('The computer won the game')
        return
print('The game is a tie')
```



# program entry

```
if __name__ == '__main__':  
    tictactoe_smart()
```