(AP) The two missing CS 5 “Black” penguins surfaced on a large ice floe near Antarctica. Authorities indicated that an enormous penguin dance party was underway there that would likely last well into Novembrrrr. The CS 5 instructors were relieved to hear that the penguins are safe. “I’m glad they’re OK, but I’m a bit disappointed that I wasn’t invited to this very 'cool' party,” said one of the profs.
# Tic-tac-toe by Ran Libeskind-Hadas

debug = True

def main():
    """Play tic-tac-toe with a human""
    welcome()
    while True:
        if debug: print("About to enter playGame")
        playGame()
        response = input("Would you like to play again? (y or n): ")
        if response not in ["y", "Y", "yes", "Yes", "Yup", "si", "oui", "youbetcha"]:  
            print("Bye")
            return

def welcome():
    """Prints the welcome message for the game.  
    We might also print the rules for the game and any other  
    information that the user might need to know.""
    print("Welcome to tic-tac-toe!")

def playGame():
    """Play one game of tic-tac-toe""
    if debug: print("Entering the playGame function")
    board = [[" ", " ", " "], [[" ", " ", " "], [" ", " ", " "]]]
    player = 1
    while not gameOver(board):
        print("The board looks like this:"
        printBoard(board)
        getMove(board, player)
        if player == 1:
            player = 2
        else:
            player = 1

        What's this?!  

        How `bout:  
        row = [" ", " ", " "]
        board = [row, row, row]

        Or  
        board = [[" ", " ", " "] * 3]
In [1]: board1 = [[' ',' ',' '], [' ',' ',' '], [' ',' ',' ']]

In [2]: board1
Out[2]: [[' ',' ',' '], [' ',' ',' '], [' ',' ',' ']]

In [3]: row = [' ',' ',' ']

In [4]: board2 = [row, row, row]

In [5]: board3 = [[' ',' ',' ']] * 3

In [6]: board1
Out[6]: [[' ',' ',' '], [' ',' ',' '], [' ',' ',' ']]

In [7]: board2
Out[7]: [[' ',' ',' '], [' ',' ',' '], [' ',' ',' ']]

In [8]: board3
Out[8]: [[' ',' ',' '], [' ',' ',' '], [' ',' ',' ']]

In [9]: board1[0][0] = "x"

In [10]: board1
Out[10]: [['x', ' ',' '], [' ',' ',' '], [' ',' ',' ']]

In [11]: board2[0][0] = "x"

In [12]: board2
Out[12]: [['x', ' ',' '], ['x', ' ',' '], ['x', ' ',' ']]

In [13]: board3[0][0] = "x"

In [14]: board3
Out[14]: [['x', ' ',' '], ['x', ' ',' '], ['x', ' ',' ']]

In [15]: row
Out[15]: ['x', ' ',' ']

In [16]: board4 = [[' ']] * 3

In [17]: board4
Out[17]: [[' ',' ',' '], [' ',' ',' '], [' ',' ',' ']]

In [18]: board4[0][0] = "x"

In [19]: board4
Out[19]: [['x', ' ',' '], ['x', ' ',' '], ['x', ' ',' ']]

In [20]: x = " "

In [21]: board4[0][0] = "x"

In [22]: board4
Out[22]: [['x', ' ',' '], ['x', ' ',' '], ['x', ' ',' ']]
In [21]: board5 = [[x]*3] * 3

In [22]: board5
Out[22]: [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [23]: board5[0][0] = "x"

In [24]: board5
Out[24]: [['x', ' ', ' '], ['x', ' ', ' '], ['x', ' ', ' ']]

In [25]: foo = [1, 2, 3]

In [26]: bar = foo

In [27]: bar = 5

In [28]: foo
Out[28]: [1, 2, 3]

In [30]: bar
Out[30]: 5

In [34]: bar = foo

In [35]: bar[0] = 5

In [36]: foo
Out[36]: [5, 2, 3]

In [37]: bar[0:3] = [2, 4, 6]

In [38]: foo
Out[38]: [2, 4, 6]

In [42]: import copy

In [43]: bar = copy.deepcopy(foo)

In [44]: foo
Out[44]: [2, 4, 6]

In [45]: bar
Out[45]: [2, 4, 6]

In [46]: bar[0] = 5

In [47]: foo
Out[47]: [2, 4, 6]

In [48]: bar
Out[48]: [5, 4, 6]
In [49]: row
Out[49]: ['x', '', ' ']

In [50]: row = [' ', ' ', ' ']

In [51]: from copy import deepcopy

In [52]: N = 3

In [53]: board5 = [deepcopy(row) for i in range(3)]

In [54]: board5
Out[54]: [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [55]: board5[0][0] = 'x'

In [56]: board5
Out[56]: [['x', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [57]: board5 = [deepcopy(row) for i in range(N)]

In [58]: board5
Out[58]: [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [59]: N = 5

In [60]: board5 = [deepcopy(row) for i in range(N)]

In [61]: board5
Out[61]: [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [62]: board5[0][0] = 'x'

In [63]: board5
Out[63]: [['x', '', ' '], [' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [64]: board6 = []

In [65]: for i in range(3):
   
   
   ....:     board6 += [deepcopy(row)]
...:

In [66]: board6
Out[66]: [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [68]: board6[0][0] = 'x'

In [69]: board6
Out[69]: [['x', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [70]: board7 = [' ' for i in range(3)]

In [71]: board7
Out[71]: [' ', ' ', ' ']

In [72]: board7 = [deepcopy([' ' for i in range(3)]) for j in range(3)]

In [73]: board7
Out[73]: [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [74]: board7[0][0]
Out[74]: '

In [75]: board7[0][0] = x

In [76]: board7[0][0] = 'x'

In [77]: board7
Out[77]: [['x', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [78]: board8 = [[' ' for i in range(3)] for j in range(3)]

In [79]: board8
Out[79]: [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [80]: board8[0][0] = 'x'

In [81]: board8
Out[81]: [['x', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [82]: board9 = [[' ']*3 for j in range(3)]

In [83]: board9
Out[83]: [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [84]: board9[0][0] = 'x'

In [85]: board9
Out[85]: [['x', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

In [87]: d = dict(a=1, b=2)
In [88]: d
Out[88]: {'a': 1, 'b': 2}

In [89]: e = d

In [90]: e['c'] = 3

In [91]: d
Out[91]: {'a': 1, 'b': 2, 'c': 3}

In [94]: d = dict(a=1, b=2)

In [95]: e = deepcopy(d)

In [96]: d
Out[96]: {'a': 1, 'b': 2}

In [97]: e
Out[97]: {'a': 1, 'b': 2}

In [98]: e['c'] = 3

In [99]: d
Out[99]: {'a': 1, 'b': 2}
Deep vs. Shallow Copy

```python
>>> foo = [1, 2, 3]
>>> bar = foo
>>> bar[0] = 42
>>> foo
[42, 2, 3]
>>> bar = ["spam", "is", "yummy"]
>>> foo
[42, 2, 3]
>>> x = 42
>>> y = x
>>> y = 57
>>> x
42
```

This is a deep concept!
Deep vs. Shallow Copy

```python
>>> from copy import *
>>> foo = [1, 2, 3]
>>> bar = deepcopy(foo)
>>> bar[0] = 42
>>> foo
[1, 2, 3]
```

This is a deep concept!
Isn't Mutability Always Preferable?

```python
def ben(List):
    if List != []: List[0] = -1

def jerry():
    myList = [1, 2, 3]
    ben(myList)
    print("My list is " + str(myList))

>>> jerry()
My list is [-1, 2, 3]
```

Watch out here!
```python
def gameOver(board):
    """Returns False if the game is NOT over. Otherwise, prints a message
    indicating which player has won and then returns True indicating that the
    game is over. THIS FUNCTION IS NOT IMPLEMENTED CORRECTLY!""
    return False

def getMove(board, player):
    """Takes the board and the current player (1 or 2) as input.
    Asks the player for her/his move. If it's a legitimate move,
    the change is made to the board. Otherwise, the player
    is queried again until a valid move is provided.""
    print("Player " + str(player) + "'s turn")
    while True:

def printBoard(board):
    # new line!
```

```python
>>> board = [["1", "2", " "], [" ", "1", "2"], [" ", " ", "1"]]
>>> printBoard(board)
1 | 2 |
---|---|
1 | 2 |
---|---|
1
```
def gameOver(board):
    """Returns False if the game is NOT over. Otherwise, prints a message indicating which player has won and then returns True indicating that the game is over. THIS FUNCTION IS NOT IMPLEMENTED CORRECTLY!""
    return False

def getMove(board, player):
    """Takes the board and the current player (1 or 2) as input. Asks the player for her/his move. If it's a legitimate move, the change is made to the board. Otherwise, the player is queried again until a valid move is provided.""
    print("Player " + str(player) + "'s turn")
    while True:
        row = input("Enter row number:")
        col = input("Enter column number:")
        # Check that they're integers
        try:
            row = int(row)
            col = int(col)
        except:
            print("Please input an int!")
            continue
        # Check that they're in the right range
        if row not in (1,2,3) or col not in (1,2,3):
            print("Enter 1, 2, or 3")
            continue
        # Check that the space is open
        if board[row-1][col-1] != " ":
            print("You can't cheat!")
            continue
        board[row-1][col-1] = str(player)
        break
def gameOver(board):
    """Returns False if the game is NOT over. Otherwise, prints a message indicating which player has won and then returns True indicating that the game is over. THIS FUNCTION IS NOT IMPLEMENTED CORRECTLY!""
    return False

def getMove(board, player):
    """Takes the board and the current player (1 or 2) as input. Asks the player for her/his move. If it's a legitimate move, the change is made to the board. Otherwise, the player is queried again until a valid move is provided.""
    print("Player " + str(player) + "'s turn")
    while True:
        try:
            row = int(input("Enter the row: "))
            column = int(input("Enter the column: "))
        except:
            print("You must enter numbers from 0 to 2 for the row and column")
            continue
        if not 0 <= row <= 2 or not 0 <= column <= 2:
            print("That's not a valid location on the board! Try again.")
        elif board[row][column] != " ":
            print("That cell is already taken! Try again.")
        else:
            board[row][column] = str(player)
            break
Another way to write this!
def printBoard(board):
    for row in range(3):
        for column in range(3):
            print(board[row][column], end=" ")
            if column < 2: print("|", end=" ")
    if row < 2:
        print()
        print("----------")
    print()  # CAUSES A LINEBREAK!

# CAUSES A LINEBREAK!

And one more thing...

if __name__ == "__main__": main()
About the Midterm

- Thursday, November 8th (next week)
  Comprehensive through 11/1
- Time in class Thursday for questions / review
- You should definitely know:
  - Recursion (including multiple base cases)
  - map, reduce, filter, lambda
  - List comprehension: \([x*2 \text{ for } x \text{ in } L]\)
  - Higher-order functions (functions that return functions)
  - Use-it-or-lose-it
More You Should Know

• Logic circuits
  - AND, OR, NOT
  - Writing truth tables
  - Minterm expansion principle
  - Using AND and OR to choose an output

• Hmmm programming
  - Recursion techniques
  - Conditional jumps
  - (We will supply a Hmmm reference sheet)
• Simple imperative programming
  - Assignment statements
  - If/else/elif
  - For loops (for i in iterable)
  - While loops
  - Break and continue
  - Return from inside loops
Your Cheat Sheet

You’re allowed ONE sheet of 8.5x11 paper, with contents up to you

Double-sided!
>>> int(1.3)
1
>>> str(1.3)
'1.3'
>>> str(2+3j)
'(2+3j)'
>>> str([1, 2, 3])
'[1, 2, 3]'
>>> str("hello")
'
'hello'
>>> int("hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: invalid literal for int(): hello
ValueError vs TypeError??!

```python
>>> class_probs = cross_val_predict(self.model, X, Y, cv=num_folds, method="predict_proba")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: Input X must be non-negative

>>> int("hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hello'

>>> a = Alien()
>>> int(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: int() argument must be a string, a bytes-like object or a number, not 'Alien'
```