An explanation of Project 3

w+XO++
w+XO++
w+XO++
w+XO++

```
DUMMY = ‘w’
# base case
If board [row][column] == ‘X” or board[row][column] = ‘O’ :
        Return board [row][column]
#recursive case
board[row][column] = DUMMY
# move down - if we can, and if we haven’t already tried that
If row < (len(board) - 1) and board[row][column] != DUMMY:
        result = eval_square(board, row+1, column)
# move up
If row > 0 and board[row][column] != DUMMY:
        result = eval_square(board, row-1, column)
# move right
If column < (len(board[0]) - 1) and board[row][column+1] != DUMMY:
        result = eval_square(board, row, column+1)
# move left
If column > 0 and board[row][column-1] != DUMMY:
        result = eval_square(board, row, column-1)
```

Final Exam Study Guide

Importing and Modularity

- Explain what a library is, what it is used for, and who is intended to use it.
  - Pre-written python code; packaged; for use by anybody who doesn't want to rewrite the code
- Explain the purpose of the import keyword
  - You will NOT be given examples of import in use and asked to explain the details of how they execute
  - Makes libraries available for use by your program
-

File I/O

- *Explain* the purpose and functionality of the open() function.
  - Opens a pre-existing file so you can access the data
  - Two arguments - file name, and mode
- *Explain* the meaning and functionality of the second open() argument and what values it can take (“w”, “r”, “a”) indicate about the opened file will be used.
  - R - opens file for read; starting at beginning of file
  - W - opens file for write; DELETES ANY EXISTING CONTENT; starts writing at beginning of file
  - A - opens file for write; DOES NOT DELETE existing content; writes new content at end
- *Define, utilize, compare* and contrast the methods for reading from a file:
  - `file.read() - whole file as one string`
  - `file.readline() - one line of the file as a string`
  - `file.readlines() - whole file as a list of strings`
- *Define and utilize* the method `file.write() - write output to file`
- *Define* `file.close()` and *remember to close any files that you open*. - you have to close a file when you’re done

Recursion

- Define recursion, recursive call, and base case - function solves a problem by repeatedly calling itself on a simpler version of the problem - base case - simplest case of problem; directly solved - recursive call - is the call to the function
- Given a problem, redefine it recursively -
    - Define base case; define recursive case; make sure recursive call has a simpler version of the problem
- Implement a recursive function
- Define the call stack
    - Python interpreter keeps track of what module is executing - main program; function;
    - Multiple copies of the same function; keep track of which one
- Explain what problems are best solved with recursion or iteration
    - Recursion is good when you can break the problem down to simpler versions of itself

Sorting and Searching

- *Explain* the problem of sorting a list - put the values in order, smallest to largest - can take a lot of operations - a long time
- *Explain* the problem of searching a list - find a specific value; may take a long time
- *Define* the following sorting algorithms and apply them to an input list:
    - Bubble sort - each time through: “bubble” largest value to end of the list by swapping values - go through list until no more swapping
    - Selection sort - pick out the smallest element in list; swap that with the first element in the list
    - Quick sort - pick a “pivot” value - put everything less than pivot is earlier in list; on left; everything greater than pivot is after the pivot - “on right” - THEN - apply quicksort to left; then to right.
- *Define* the following searching algorithms and apply them to a input list and target value:
    - Linear/Sequential search - check each item in list to find the value we’re looking for
    - Binary search - check the middle value in the list. If target is less, look to the left (before the middle); else look to right (after the middle)
- *Explain* why binary search only works for lists that are already sorted - if values were not sorted, you wouldn’t know after checking the middle whether target was on left or right
- **NOTE:** No dancing will be required before, during or after the exam

Asymptotic Analysis

- *Define* big o and big omega in terms of best and worst case performance
- ` Big O - worst case behavior - Big Omega - best case behavior
- *Define* big theta in terms of big o and big omega
- *Rank* the following functions from slowest growing to fastest growing (i.e. fastest to slowest) - 1; log2n; n; nlogn; n2
  - 1 (constant) -
  - $\log_2 n$ (logarithmic)
  - n (linear)
  - $n * \log_2 n$ (“n log n”)
  - n^2 (quadratic)
- *Analyze* the best and worst performance of the following algorithms as functions of n:
  - Bubble sort O(n2) Omega(n)
  - Selection sort O(n2) Omega(n)
  - Quick sort - same
  - Linear/Sequential search - O(n) Omega(1)
  - Binary search O(logn) Omega (1)
- *Provide inputs* (e.g. lists to be sorted) for the best and worst case scenarios for the above algorithms
  - Best behavior - list is already sorted; value to be found is found first time
  - Worst behavior: search - value is not found; sorting: exactly backwards
- *Define* log in terms of division

Data representation

- *Define* the decimal, binary and hexadecimal counting systems
  - Base 2, base 10, base 16
  - 0, 1; binary
  - Hex: A, B, C, D, E, F
- *Convert* a number from any one counting system to another counting system
  - Binary to hex: 4 bits to 1 hex digit
  - 10011010 = 9A
- NOTE: you will NOT be permitted to use a calculator, but arithmetic will be reasonable to solve by hand

```
Import random

From random import randint
```