```
;;
;; ************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 05, Problem 2
;; ************************************************
;;



;; 2a)

;;(my-list-ref lst index) produces the value index steps from the front of lst
;;  or false if lst is not long enough
;; my-list-ref: (listof Num) Nat -> (anyof Num false)
;; Examples:
(check-expect (my-list-ref '(1 2 3 4) 0) 1)
(check-expect (my-list-ref empty 3) false)
(check-expect (my-list-ref '(2) 20) false)

(define (my-list-ref lst index)
  (cond [(empty? lst) false]
        [(zero? index) (first lst)]
        [else (my-list-ref (rest lst) (sub1 index))]))

;; Tests:
(check-expect (my-list-ref '(5 4 3) 1) 4)
(check-expect (my-list-ref empty 0) false)
(check-expect (my-list-ref '(4 5 6) 2) 6)



;; 2b)

;;(zip keys values) produces an association list with the given keys and values
;; zip: (listof Num) (listof Str) -> (listof (list Num Str))
;; requires: length of keys is equal to length of values
;;           all numbers in keys are unique
;; Examples:
(check-expect (zip '(1 2 3) '("A" "B" "C")) '((1 "A") (2 "B") (3 "C")))
(check-expect (zip empty empty) empty)

(define (zip keys values)
  (cond [(empty? keys) empty]
        [else (cons (list (first keys) (first values))
                    (zip (rest keys) (rest values)))]))

;; Test:
(check-expect (zip '(9) '("Z")) '((9 "Z")))



;; 2c)

;;(count-symbol los sym) produces the number of sym that occur in los
;; count-symbol: (listof Sym) Sym -> Nat
;; Examples:
(check-expect (count-symbol '(X _ X O) 'X) 2)
(check-expect (count-symbol empty 'X) 0)

(define (count-symbol los sym)
  (cond [(empty? los) 0]
        [(symbol=? sym (first los))
```

```
         (add1 (count-symbol (rest los) sym))]
        [else (count-symbol (rest los) sym)]))


;;(count-symbol/2D grid sym) counts all occurences of sym in any lists in grid
;; count-symbol/2D: (listof (listof Sym)) Sym -> Nat
;; Examples:
(check-expect (count-symbol/2D '((X O X) (_ _ _) (O O O)) 'O) 4)
(check-expect (count-symbol/2D empty 'O) 0)

(define (count-symbol/2D grid sym)
  (cond [(empty? grid) 0]
        [else (+ (count-symbol (first grid) sym)
                 (count-symbol/2D (rest grid) sym))]))

;; Tests:
(check-expect (count-symbol/2D '(()()) 'Z) 0)
(check-expect (count-symbol/2D '((a b c) (q r s)) 't) 0)
(check-expect (count-symbol/2D '((a b c) (q q q)) 'q) 3)
(check-expect (count-symbol/2D '((a q c) (a a a)) 'q) 1)
(check-expect (count-symbol/2D '((a c c) (a a a) (q q)) 'q) 2)



;;
;; ************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 05, Problem 3
;; ************************************************
;;


;; A Book is a (list Str Str)

;; useful constants for the examples
(define my-bookshelf '(("The Colour of Magic" "Pratchett, Terry")
                       ("Mostly Harmless" "Adams, Douglas")
                       ("Pyramids" "Pratchett, Terry")
                       ("A Brief History of Time" "Hawking, Stephen")))

(define discworld-books '(("The Colour of Magic" "Pratchett, Terry")
                          ("Pyramids" "Pratchett, Terry")))

;; An AuthorIndex is a (listof (cons Str (listof Str)))
;; requires: The first Str in each inner list is unique

(define my-index '(("Pratchett, Terry" "The Colour of Magic" "Pyramids")
                   ("Hawking, Stephen" "A Brief History of Time")
                   ("Adams, Douglas")))



;; 3a)

;; book-template: Book -> Any
(define (book-template b)
  (... (first b) ...
       (second b) ...))

;; listof-book-template: (listof Book) -> Any
(define (listof-book-template lob)
  (cond [(empty? lob) ...]
```

```
        [else (... (book-template (first lob))
                   (listof-book-template (rest lob)) ...)]))


;; also acceptable

(define (listof-book-template/alt lob)
  (cond [(empty? lob) ...]
        [else (... (first (first lob)) ...
                   (second (first lob)) ...
                   (listof-book-template/alt (rest lob)) ...)]))



;; 3b)

;;(book<? bk1 bk2) determines if book b1 would come before book b2 on a shelf
;; book<?: Book Book -> Bool
;; Examples:
(check-expect (book<? '("Foo" "Bar") '("X" "Z")) true)
(check-expect (book<? '("Z" "Foo") '("A" "Foo")) false)

(define (book<? bk1 bk2)
  (or (string<? (second bk1) (second bk2))
      (and
       (string=? (second bk1) (second bk2))
       (string<? (first bk1) (first bk2)))))


;;(book-insert bk lobk) inserts bk into the correct location in lobk to maintain
;;  sorted order
;; book-insert: Book (listof Book) -> (listof Book)
;; requires: lobk is in sorted order
;; Example:
(check-expect (book-insert '("The Colour of Magic" "Pratchett, Terry")
                           '(("Mostly Harmless" "Adams, Douglas")
                             ("The Hobbit" "Tolkien, J.R.R.")))
              '(("Mostly Harmless" "Adams, Douglas")
                ("The Colour of Magic" "Pratchett, Terry")
                ("The Hobbit" "Tolkien, J.R.R.")))
(check-expect (book-insert '("A" "B") empty) '(("A" "B")))

(define (book-insert bk lobk)
  (cond [(empty? lobk) (list bk)]
        [(book<? bk (first lobk)) (cons bk lobk)]
        [else (cons (first lobk)
                    (book-insert bk (rest lobk)))]))


;;(sort-books lobk) produces a sorted version of lobk
;; sort-books: (listof Book) -> (listof Book)
;; Examples:
(check-expect (sort-books empty) empty)
(check-expect (sort-books '(("Q" "X") ("A" "B") ("Z" "X")))
              '(("A" "B") ("Q" "X") ("Z" "X")))

(define (sort-books lobk)
  (cond [(empty? lobk) empty]
        [else (book-insert (first lobk)
                           (sort-books (rest lobk)))]))

;; Test:
(check-expect (sort-books '(("A" "B"))) '(("A" "B")))
```

```
;; 3c)

;;(books-by-author lobk author) produces a list of the books in lobk that
;;  are by author
;; books-by-author: (listof Book) Str -> (listof Book)
;; Examples:
(check-expect (books-by-author empty "X") empty)
(check-expect (books-by-author my-bookshelf "Pratchett, Terry") discworld-books)

(define (books-by-author lobk author)
  (cond [(empty? lobk) empty]
        [(string=? author (second (first lobk)))
         (cons (first lobk) (books-by-author (rest lobk) author))]
        [else (books-by-author (rest lobk) author)]))

;; Tests:
(check-expect (books-by-author my-bookshelf "X") empty)
(check-expect (books-by-author my-bookshelf "Hawking, Stephen")
              '(("A Brief History of Time" "Hawking, Stephen")))



;; 3d)

;;(book-by-author? aindex author title) determines if aindex contains a book
;;  by the given author and with the given title
;; book-by-author?: AuthorIndex Str Str -> Bool
;; Examples:
(check-expect (book-by-author? empty "A" "B") false)
(check-expect (book-by-author? my-index "Pratchett, Terry" "Pyramids") true)

(define (book-by-author? aindex author title)
  (cond [(empty? aindex) false]
        [(string=? author (first (first aindex)))
         (member? title (rest (first aindex)))]
        [else (book-by-author? (rest aindex) author title)]))

;; Tests:
(check-expect (book-by-author? my-index "King, Stephen" "It") false)
(check-expect (book-by-author? my-index "Adams, Douglas" "Mostly Harmless")
              false)
(check-expect (book-by-author? '(("X" "Y" "Z") ("Q" "T")) "Q" "T") true)



;; 3e)

;;(book-titles lobk) lobk produces a list of the titles of books in lob
;;  (in the same order)
;; book-titles: (listof Book) -> (listof Str)
;; Examples:
(check-expect (book-titles discworld-books) '("The Colour of Magic" "Pyramids"))
(check-expect (book-titles empty) empty)

(define (book-titles lobk)
  (cond [(empty? lobk) empty]
        [else (cons (first (first lobk))
                    (book-titles (rest lobk)))]))
```

```
;;(build-author-index lobk authors) creates an AuthorIndex for the given authors
;;  using books from lobk
;; build-author-index: (listof Book) (listof Str) -> AuthorIndex
;; Examples:
(check-expect (build-author-index my-bookshelf empty) empty)
(check-expect (build-author-index my-bookshelf '("Pratchett, Terry"))
              '(("Pratchett, Terry" "The Colour of Magic" "Pyramids")))

(define (build-author-index lobk authors)
  (cond [(empty? authors) empty]
        [else (cons (cons (first authors)
                          (book-titles (books-by-author lobk (first authors))))
                    (build-author-index lobk (rest authors)))]))

;; Tests:
(check-expect (build-author-index empty '("A" "B"))
              '(("A") ("B")))
(check-expect (build-author-index my-bookshelf '("A" "B"))
              '(("A") ("B")))
(check-expect (build-author-index my-bookshelf '("Pratchett, Terry" "A"))
              '(("Pratchett, Terry" "The Colour of Magic" "Pyramids") ("A")))



;;
;; ************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 05, Problem 4
;; ************************************************
;;


;; A Tic Tac Toe Grid (T3Grid) is a (listof (listof (anyof 'X 'O '_)))
;; requires: all lists have the same length, and that length is odd
;;           number of 'X is equal to either number of 'O, or (add1 number of 'O)

;; Helpful Constants

(define grid1 '((_ _ _)
                (_ _ _)
                (_ _ _)))

(define grid2 '((X O O O _)
                (X X O _ _)
                (_ _ _ _ _)
                (_ _ _ _ _)
                (_ _ _ _ X)))

(define grid3 '((X)))

;; End of Starter (Start of Solutions)


(define sample-3x3 '((X _ _)
                     (X O _)
                     (_ _ _)))


;; 4a)

;; Helpers (lifted from Q2)
```

```
;;(count-symbol los sym) produces the number of sym that occur in los
;; count-symbol: (listof Sym) Sym -> Nat
;; Examples:
(check-expect (count-symbol '(X _ X O) 'X) 2)
(check-expect (count-symbol empty 'X) 0)

(define (count-symbol los sym)
  (cond [(empty? los) 0]
        [(symbol=? sym (first los))
         (+ 1 (count-symbol (rest los) sym))]
        [else (count-symbol (rest los) sym)]))


;;(count-symbol/2D grid sym) counts all occurences of sym in any lists in grid
;; count-symbol/2D: (listof (listof Sym)) Sym -> Nat
;; Examples:
(check-expect (count-symbol/2D '((X O X) (_ _ _) (O O O)) 'O) 4)
(check-expect (count-symbol/2D empty 'O) 0)

(define (count-symbol/2D grid sym)
  (cond [(empty? grid) 0]
        [else (+ (count-symbol (first grid) sym)
                 (count-symbol/2D (rest grid) sym))]))


;;(whose-turn grid) produces 'X or 'O, depending on which player should move
;;  next in grid
;; whose-turn: T3Grid -> (anyof 'X 'O)
;; Example:
(check-expect (whose-turn sample-3x3) 'O)

(define (whose-turn grid)
  (cond [(= (count-symbol/2D grid 'X)
            (count-symbol/2D grid 'O)) 'X]
        [else 'O]))

;; Tests:
(check-expect (whose-turn '((_))) 'X)
(check-expect (whose-turn '((_ _ _)
                            (_ _ _)
                            (_ _ _))) 'X)
(check-expect (whose-turn '((X _ _)
                            (_ _ _)
                            (_ _ _))) 'O)
(check-expect (whose-turn '((X X X X X)
                            (O O O O O)
                            (X X X X X)
                            (_ O O O O)
                            (X X X X X))) 'O)


; 4b)

;;(gridref grid row col) produces the symbol located at the position
;;  (row,col) in grid
;; gridref: T3Grid Nat Nat -> (anyof 'X 'O '_)
;; requires: row,col < (length grid)
;; Examples:
(check-expect (grid-ref '((X)) 0 0) 'X)
(check-expect (grid-ref sample-3x3 2 2) '_)

(define (grid-ref grid row col)
  (list-ref (list-ref grid row) col))
```

```
;; Tests
(check-expect (grid-ref sample-3x3 0 0) 'X)
(check-expect (grid-ref sample-3x3 0 2) '_)
(check-expect (grid-ref sample-3x3 2 0) '_)
(check-expect (grid-ref sample-3x3 1 1) 'O)
(check-expect (grid-ref grid2 1 2) 'O)



;; 4c)

;;(get-column grid col) produces the markers from column # col in grid
;; get-column: T3Grid Nat -> (listof (anyof 'X 'O '_))
;; requires: col < (length grid)
;; Examples:
(check-expect (get-column grid3 0) '(X))
(check-expect (get-column grid2 1) '(O X _ _ _))

(define (get-column grid col)
  (cond [(empty? grid) empty]
        [else (cons (list-ref (first grid) col)
                    (get-column (rest grid) col))]))

;; Tests:
(check-expect (get-column grid2 0) '(X X _ _ _))
(check-expect (get-column grid2 4) '(_ _ _ _ X))
(check-expect (get-column grid1 1) '(_ _ _))



;; 4d)

(define win-grid1 '((X O _)
                    (X _ X)
                    (_ O X)))

(define win-grid2 '((X X X X O)
                    (X X _ X X)
                    (O _ O O O)
                    (O O O _ O)
                    (O O X O O)))


;;(will-win? grid row col player) produces true if player can win by
;;  legally placing a mark at (row,col) in grid
;; will-win?: T3Grid Nat Nat (anyof 'X O) -> Bool
;; requires: row, col < (length grid)
;;           no player has already won
;; Examples:
(check-expect (will-win? '((_)) 0 0 'X) true)
(check-expect (will-win? grid1 1 2 'O) false)

(define (will-win? grid row col player)
  (and (symbol=? (grid-ref grid row col) '_)
       (or (= (count-symbol (list-ref grid row) player)
              (sub1 (length grid)))
           (= (count-symbol (get-column grid col) player)
              (sub1 (length grid))))))

;; Tests:
(check-expect (will-win? win-grid1 2 0 'X) true)
(check-expect (will-win? win-grid1 2 0 'O) false)
```

```
(check-expect (will-win? win-grid1 1 1 'O) true)
(check-expect (will-win? win-grid1 1 1 'X) true)
(check-expect (will-win? win-grid2 0 4 'X) false)
(check-expect (will-win? win-grid2 0 4 'O) false)
(check-expect (will-win? win-grid2 1 2 'O) false)
(check-expect (will-win? win-grid2 1 2 'X) true)
(check-expect (will-win? win-grid2 2 1 'O) true)
(check-expect (will-win? win-grid2 2 1 'X) false)
(check-expect (will-win? win-grid2 3 4 'X) false)
(check-expect (will-win? win-grid2 3 3 'O) true)
(check-expect (will-win? win-grid2 4 2 'X) false)
(check-expect (will-win? win-grid2 4 2 'O) false)
```