```
;; CS 135
;; Winter 2014
;; Assignment 2, Question 1
;; SOLUTIONS

(define (q1a-original x)
  (cond [(p1? x)
         (cond
           [(not (p2? x)) (f3 x)]
           [(p3? x) (f2 x)]
           [(p2? x) (f1 x)])]
        [else
         (cond [(p2? x) (f2 x)]
               [else x])]))

(define (q1a x)
  (cond
    [(and (p1? x) (not (p2? x))) (f3 x)]
    [(and (p1? x) (p3? x)) (f2 x)]
    [(and (p1? x) (p2? x)) (f1 x)]
    [(and (not (p1? x)) (p2? x)) (f2 x)]
    [else x]))

(define (q1b-original x)
  (cond [(number? x)
         (cond [(cond [(p2? x) (p1? x)]
                      [else (p3? x)]) 'alfa]
               [else (cond [(p3? x) 'bravo]
                           [else 'charlie])])]
         [(symbol? x)
          (cond [(symbol=? x 'delta) 'echo]
                [else 'delta])]
         [else 'foxtrot]))

(define (q1b x)
  (cond
    [(and (number? x) (p2? x) (p1? x)) 'alfa]
    [(and (number? x) (not (p2? x)) (p3? x)) 'alfa]
    [(and (number? x) (not (p2? x)) (not (p3? x))) 'charlie]
    [(and (number? x) (p2? x) (not (p1? x)) (p3? x)) 'bravo]
    [(and (number? x) (p2? x) (not (p1? x)) (not (p3? x))) 'charlie]
    [(and (symbol? x) (symbol=? x 'delta)) 'echo]
    [(symbol? x) 'delta]
    [else 'foxtrot]))

;; Assignment 2, Question 2
;; SOLUTIONS

;; part (a):  using cond

;; arith-seq-cond?: Int Int Int -> Boolean
;; Purpose:
;; Consumes integers a,b,c and produces true iff a,b,c
;; can form an arithmetic sequence after rearrangement
;; Examples:
(check-expect (arith-seq-cond? 1 4 6) false)
(check-expect (arith-seq-cond? 6 2 4) true)
```

```
(define (arith-seq-cond? a b c)
  (cond
    [(= (+ a (* 2 (- b a))) c) true]
    [(= (+ a (* 2 (- c a))) b) true]
    [(= (+ b (* 2 (- c b))) a) true] ;; technically not needed
    [(= (+ b (* 2 (- a b))) c) true]
    [(= (+ c (* 2 (- a c))) b) true] ;; technically not needed
    [(= (+ c (* 2 (- b c))) a) true] ;; technically not needed
    [else false]))
;; Tests
(check-expect (arith-seq-cond? 10 20 30) true)
(check-expect (arith-seq-cond? 10 30 20) true)
(check-expect (arith-seq-cond? 20 30 10) true)
(check-expect (arith-seq-cond? 20 10 30) true)
(check-expect (arith-seq-cond? 30 10 20) true)
(check-expect (arith-seq-cond? 30 20 10) true)
(check-expect (arith-seq-cond? 30 20 30) false)
(check-expect (arith-seq-cond? 4 4 4) true)
(check-expect (arith-seq-cond? 4 5 7) false)
(check-expect (arith-seq-cond? 5 10 20) false)

;; part (b):  using booleans

;; arith-seq-bool?: Int Int Int -> Boolean
;; Purpose:
;; Consumes integers a,b,c and produces true iff a,b,c
;; can form an arithmetic sequence after rearrangement
;; Examples:
(check-expect (arith-seq-bool? 1 4 6) false)
(check-expect (arith-seq-bool? 6 2 4) true)
(define (arith-seq-bool? a b c)
  (or
    (= (+ a (* 2 (- b a))) c)
    (= (+ a (* 2 (- c a))) b)
    (= (+ b (* 2 (- c b))) a)  ;; technically not needed
    (= (+ b (* 2 (- a b))) c)
    (= (+ c (* 2 (- a c))) b)   ;; technically not needed
    (= (+ c (* 2 (- b c))) a)))   ;; technically not needed
;; Tests
(check-expect (arith-seq-bool? 10 20 30) true)
(check-expect (arith-seq-bool? 10 30 20) true)
(check-expect (arith-seq-bool? 20 30 10) true)
(check-expect (arith-seq-bool? 20 10 30) true)
(check-expect (arith-seq-bool? 30 10 20) true)
(check-expect (arith-seq-bool? 30 20 10) true)
(check-expect (arith-seq-bool? 30 20 30) false)
(check-expect (arith-seq-bool? 4 4 4) true)
(check-expect (arith-seq-bool? 4 5 7) false)
(check-expect (arith-seq-bool? 5 10 20) false)

;; Assignment 2, Question 3
;; SOLUTIONS

;; part (a):

;; constants

;; weights for different course elements
```

```
(define partic-weight 5/100)
(define assn-weight 20/100)
(define mt1-weight 10/100)
(define mt2-weight 20/100)
(define fe-weight 45/100)

;; grade benchmarks
(define pass-grade 50)
(define target-grade 60)
(define max-failing 46)

;; weighted exam benchmark
(define exam-pass
  (* pass-grade (+ mt1-weight mt2-weight fe-weight)))

;; clicker constants
(define click-max 75/100)
(define right-pts 2)
(define wrong-pts 1)

;; participation-mark: Nat Nat Nat -> Num
;; determines the cs135 class participation mark
;; for a student, based on the number of right and wrong
;; answers and the total number of questions asked
;; examples:
(check-expect (participation-mark 40 30 10) 100)
(check-expect (participation-mark 40 12 6) 50)
(check-expect (participation-mark 40 0 30) 50)

;; note... this was done without cond to show you it was not necessary
(define (participation-mark total right wrong)
  (* 100
     (/ (+ (* right-pts (min right (* total click-max)))
           (* wrong-pts (min wrong (max 0 (- (* total click-max) right)))))
        (* total click-max right-pts))))

;; tests:
(check-expect (participation-mark 400 0 0) 0)
(check-expect (participation-mark 400 0 400) 50)
(check-expect (participation-mark 400 0 300) 50)
(check-expect (participation-mark 400 0 200) 200/6)
(check-expect (participation-mark 400 0 100) 100/6)
(check-expect (participation-mark 400 400 0) 100)
(check-expect (participation-mark 400 300 0) 100)
(check-expect (participation-mark 400 200 0) 400/6)
(check-expect (participation-mark 400 100 0) 200/6)
(check-expect (participation-mark 400 300 100) 100)
(check-expect (participation-mark 400 200 200) 500/6)
(check-expect (participation-mark 400 100 300) 400/6)
(check-expect (participation-mark 400 100 50) 250/6)
(check-expect (participation-mark 400 50 100) 200/6)

;; part (b)
;; raw-cs135-grade: Nat Nat Nat Nat Nat -> Num
;; produces the unadjusted cs135 mark using the grades from the
;; final exam [fe], the two midterms [mt1 & mt2],
;; the assignments [assn] and the participation grade [partic].
;; examples:
```

```
(check-expect (raw-final-cs135-grade 40 40 40 40 40) 40)
(check-expect (raw-final-cs135-grade 40 60 60 70 80) 54)
(check-expect (raw-final-cs135-grade 50 50 50 50 50) 50)
(check-expect (raw-final-cs135-grade 60 60 60 40 60) 56)
(check-expect (raw-final-cs135-grade 50 100 100 100 100) 77.5)

(define (raw-final-cs135-grade fe mt1 mt2 assn partic)
  (+ (* partic-weight partic)
     (* fe-weight fe)
     (* mt1-weight mt1)
     (* mt2-weight mt2)
     (* assn-weight assn)))

;; tests:
(check-expect (raw-final-cs135-grade 0 0 0 0 0) 0)
(check-expect (raw-final-cs135-grade 100 100 100 100 100) 100)
(check-expect (raw-final-cs135-grade 50 60 70 80 90) 63)


;; final-cs135-grade: Nat Nat Nat Nat Nat -> Num
;; produces the adjusted cs135 mark using the grades from the
;; final exam [fe], the two midterms [mt1 & mt2],
;; the assignments [assn] and the participation grade [partic].
;; examples:
(check-expect (final-cs135-grade 40 40 40 40 40) 40)
(check-expect (final-cs135-grade 40 60 60 70 80) 46)
(check-expect (final-cs135-grade 50 50 50 50 50) 50)
(check-expect (final-cs135-grade 60 60 60 40 60) 46)
(check-expect (final-cs135-grade 50 100 100 100 100) 77.5)

(define (final-cs135-grade fe mt1 mt2 assn partic)
  (cond [(or (> 50 assn)
             (> exam-pass
                (+ (* fe-weight fe) (* mt1-weight mt1) (* mt2-weight mt2))))
         (min max-failing (raw-final-cs135-grade fe mt1 mt2 assn partic))]
        [else (raw-final-cs135-grade fe mt1 mt2 assn partic)]))

;; tests:
(check-expect (final-cs135-grade 0 0 0 0 0) 0)
(check-expect (final-cs135-grade 100 100 100 100 100) 100)
(check-expect (final-cs135-grade 0 100 100 100 100) 46)
(check-expect (final-cs135-grade 100 0 100 100 100) 90)
(check-expect (final-cs135-grade 100 100 0 100 100) 80)
(check-expect (final-cs135-grade 100 100 100 0 100) 46)
(check-expect (final-cs135-grade 100 100 100 100 0) 95)
(check-expect (final-cs135-grade 49 49 49 49 49) 46)
(check-expect (final-cs135-grade 50 50 50 0 50) 40)
(check-expect (final-cs135-grade 100 100 100 49 100) 46)
(check-expect (final-cs135-grade 50 60 70 80 90) 63)

;; Assignment 2, Question 4
;; SOLUTIONS

;; rpsls: Symbol Symbol -> Symbol
;; produces 'player1 if s1 defeats s2 in a game
;; of rock-paper-scissors-lizard-spock, 'player2
;; if s2 defeats s1, or 'tie otherwise
;; examples:
```

```
(check-expect (rpsls 'rock 'rock) 'tie)
(check-expect (rpsls 'rock 'paper) 'player2)
(check-expect (rpsls 'paper 'rock) 'player1)

(define (rpsls s1 s2)
  (cond
    [(symbol=? s1 s2) 'tie]
    [(or (and (symbol=? s1 'rock)
              (or (symbol=? s2 'lizard)
                  (symbol=? s2 'scissors)))
         (and (symbol=? s1 'paper)
              (or (symbol=? s2 'rock)
                  (symbol=? s2 'spock)))
         (and (symbol=? s1 'scissors)
              (or (symbol=? s2 'paper)
                  (symbol=? s2 'lizard)))
         (and (symbol=? s1 'lizard)
              (or (symbol=? s2 'spock)
                  (symbol=? s2 'paper)))
         (and (symbol=? s1 'spock)
              (or (symbol=? s2 'scissors)
                  (symbol=? s2 'rock)))) 'player1]
    [else 'player2]))

;; tests:
(check-expect (rpsls 'rock 'rock) 'tie)
(check-expect (rpsls 'rock 'paper) 'player2)
(check-expect (rpsls 'rock 'scissors) 'player1)
(check-expect (rpsls 'rock 'lizard) 'player1)
(check-expect (rpsls 'rock 'spock) 'player2)
(check-expect (rpsls 'paper 'rock) 'player1)
(check-expect (rpsls 'paper 'paper) 'tie)
(check-expect (rpsls 'paper 'scissors) 'player2)
(check-expect (rpsls 'paper 'lizard) 'player2)
(check-expect (rpsls 'paper 'spock) 'player1)
(check-expect (rpsls 'scissors 'rock) 'player2)
(check-expect (rpsls 'scissors 'paper) 'player1)
(check-expect (rpsls 'scissors 'scissors) 'tie)
(check-expect (rpsls 'scissors 'lizard) 'player1)
(check-expect (rpsls 'scissors 'spock) 'player2)
(check-expect (rpsls 'lizard 'rock) 'player2)
(check-expect (rpsls 'lizard 'paper) 'player1)
(check-expect (rpsls 'lizard 'scissors) 'player2)
(check-expect (rpsls 'lizard 'lizard) 'tie)
(check-expect (rpsls 'lizard 'spock) 'player1)
(check-expect (rpsls 'spock 'rock) 'player1)
(check-expect (rpsls 'spock 'paper) 'player2)
(check-expect (rpsls 'spock 'scissors) 'player1)
(check-expect (rpsls 'spock 'lizard) 'player2)
(check-expect (rpsls 'spock 'spock) 'tie)
```