Do not turn this page until you have received the signal to start.
(Please fill out the identification section above,
and read the instructions below.) Good Luck!

# 1: _____/10
# 2: _____/ 5
# 3: _____/10
# 4: _____/10

TOTAL: _____/35
Question 1.  [10 marks]
For this question: use higher-order-functions 'map', 'apply', and/or 'filter' wherever appropriate.

Part (a)  [6 marks]
Write 'pull-back' that takes a unary function 'f', a domain list 'D', and a value 'y', and produces a list of each value in 'D' such that 'f' at the value is equal to 'y'.

(check-expect (pull-back sqr '(-3 2 1 -2) 4)  
  '(2 -2))
(check-expect (pull-back sqr '(-3 2 1 -2) 1)  
  '(1))

Part (b)  [4 marks]
Write 'pull-backs' that is like 'pull-back' but it takes a list of y values 'ys' and produces a list of each value in 'D' such that 'f' at the value is equal to one of the values in 'ys'. You may use 'pull-back'.

(check-expect (pull-backs sqr '(-3 2 1 -2) '(1 4))  
  '(1 2 -2))
Question 2. [5 marks]
Assume that 'and' is not in the racket library, but 'or' and 'not' are.
Write a macro to define 'and', that works with any number of boolean expressions.
Hint: recall DeMorgan's Law: \( \neg(P \lor Q) \equiv \neg P \land \neg Q \).

Question 3. [10 marks]
Show the tree of environments and closures for the following code, and the result value of the final expression:

\[
\text{(define (f x)} \\
\text{\hspace{1cm} (set! x (+ x 1))} \\
\text{\hspace{1cm} (lambda () x))} \\
\text{(define x (f 1))} \\
\text{(define y (f 20))} \\
\text{(y)}
\]
Question 4. [10 marks]

Consider the following runtime data language of propositional formulas.

A symbol is an atomic propositional formula. If P and Q are propositional formulas, then so are the lists:

- \((P \land Q)\)
- \((P \lor Q)\)
- \((\neg P)\)

Write a function ‘move’ that takes a propositional formula and moves the negations inward, producing a logically equivalent formula that has negation only in the form \((\neg A)\) where A is an atomic propositional formula.

Recall:

- \((\neg(\neg P))\equiv P\)
- \((\neg(P \land Q))\equiv ((\neg P) \lor (\neg Q))\)
- \((\neg(P \lor Q))\equiv ((\neg P) \land (\neg Q))\)

(check-expect (move '(a \land (\neg (b \lor (\neg c)))))) '((a \land ((\neg b) \land c)))

Use ‘match’.
Use quasi-quotation in the patterns and the result expressions where worthwhile.

Hint: include a clause or clauses for non-negated conjunction and disjunction.