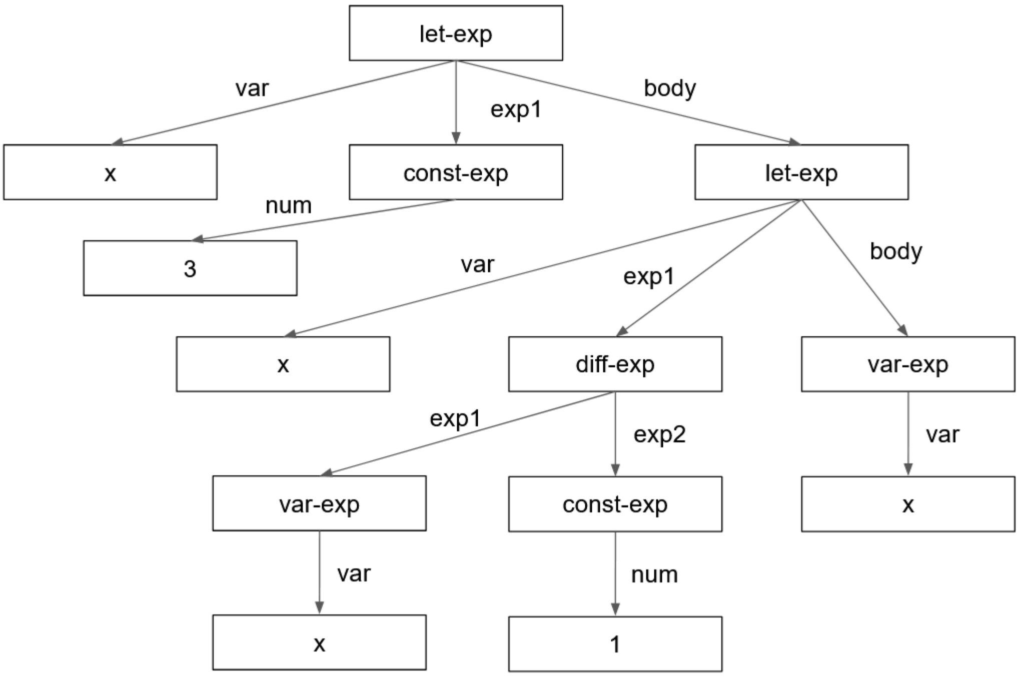
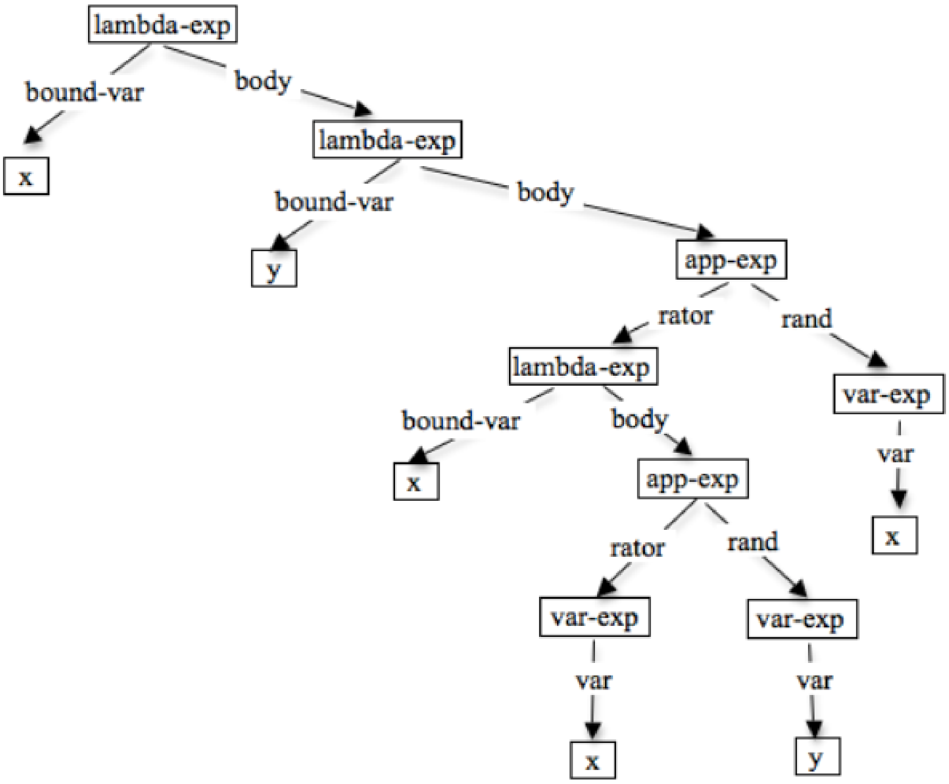
Answers

P1 - Part A.



P1 - Part B.



P2.

```
; Base function
(define path
 (lambda (n bin-tree)
  (cond
   null? bin-tree) '())
   ((eqv? n (car bin-tree)) '(root))
   (else (path-rec n bin-tree '(root))))))

; Recursive auxillary function
(define path-rec
  (lambda (n bin-tree path-so-far)
   (cond
    ((null? bin-tree) '())
    ((eqv? n (car bin-tree)) path-so-far)
    (else
     (append
      (path-rec n (cadr bin-tree) (append path-so-far '(left)))
      (path-rec n (caddr bin-tree) (append path-so-far '(right))))))))
```

P3.

- **a)** Evaluates to 3. The expression is syntactically correct, but `x` is not used anyways and it just returns 3.
- **b)** Evaluates to -7 from $-4 - 3 = -7$
- **c)** Evaluates to 0. Though out of scope of this class, when you consider that an else expression is optional in a grammar such as LET's, you can derive an expression in more than one way. Such a grammar would be called "ambiguous". Google "ambiguous grammar" if interested!