i.-

```
((curry append) '(a b c)) '(6 7))

((lambda(x y) (append x y)) '(a b c)) '(6 7))

((lambda(y) (append '(a b c) y)) '(6 7))

(append '(a b c) '(6 7))

(a b c 6 7)
```

---

ii.-

```
(equal? '(a . (b c)) (append '(a) '() '(b . (c))))

(equal? '(a b c) (append '(a) '() '(b . (c))))

(equal? '(a b c) (append '(a) '() '(b c)))

(equal? '(a b c) '(a b c))

#t
```

---

iii.-

```
(filter (compose positive? car) (map list '(1 -3 5)))

(filter (lambda(x) (positive? (car x))) (map list '(1 -3 5)))

(filter (lambda(x) (positive? (car x))) '((1) (-3) (5)))

((1) (5))
```

---

iv.-

```
(let ((a 1) (b 2) (c 3))
     (let* ((a 'uno) (b (list a c)) (c 'tres))
          (list a b c)))

;a=1 b=2 c=3
     (let* ((a 'uno) (b (list a c)) (c 'tres))
          (list a b c)))

;a= uno b=(uno 3) c=tres
          (list a b c)

(uno (uno 3) tres)
```

---

v.-

```
((lambda ls (filter (compose negative? car) ls))
     '(1 -2) '(-4 5) '(6 -3))

(filter (compose negative? car) '((1 -2) (-4 5) (6 -3)))

(filter (compose negative? car) '((1 -2) (-4 5) (6 -3)))

(filter (lambda(x) (negative? (car x))) '((1 -2) (-4 5) (6 -3)))

((-4 5))
```

---

vi.-

```
(apply map + '((1 2) (3 4)))

(map + '(1 2) '(3 4))

( (+ 1 3) (+ 2 4))

(4 6)
```