ASSIGNMENT ON LISTS

E1.30 (**) (sort/predicate pred loi) returns a list of elements sorted by the predicate

```
> (sort/predicate < '(8 2 5 2 3))
(2 2 3 5 8)

> (sort/predicate > '(8 2 5 2 3))
(8 5 3 2 2)

(define sort/predicate
  (lambda (pred loi)
    (
```

E1.29 (sort loi) returns a list of the elements of loi in ascending order

```
> (sort '(8 2 5 2 3))
(2 2 3 5 8)
```

E1.28 (merge loi1 loi2) where loi1 and loi2 are lists of integers that are sorted in ascending order, returns a sorted list of all the integers in loi1 and loi2.

```
>(merge '(1 4) '(1 2 8))
(1 1 2 4 8)
>(merge '(35 62 81 90 91) '(3 83 85 90))
(3 35 62 81 83 85 90 90 91)
```

E1.27 (flatten slist) returns a list of symbols contained in slist in the order in which they occur when slist is printed. intuitively, flatten removes all the parentheses from its argument.

```
> (flatten '(a b c))
(a b c)
> (flatten '((a) () (b c)) () (c)))
(a b c)
> (flatten '((a b) c (((d)) e)))
(a b c d e)
> (flatten '(a b (() (c))))
(a b c)
```

E1.26 (up lst) removes a pair of parentheses from each top-level element of lst. if a top-level element is not a list, it is included in the result, as is. the value of (up (down lst)) is equivalent to lst, but (down (up lst)) is not nessarily lst.

```
> (up '((1 2) (3 4)))
(1 2 3 4)
> (up '((x (y)) z))
(x (y) z)
```

E1.25 (exists? pred lst) returns #t if any element of lst satisfies pred, and returns #f otherwise.

> (exists? number? '(a b c 3 e))
#t
> (exists? number? '(a b c d e))
#f.

E1.24 (every? pred lst) returns #f if any element of lst fails to satisfy pred, returns #t otherwise.

> (every? number? '(a b c 3 e))
#f
> (every? number? '(1 2 3 4 5))
#t

E1.23 (list-index pred lst) returns the 0-based position of the first element of lst that satisfies the predicate pred. if no element of the lst satisfies the predicate, then list-index returns #f.

> (list-index number? '(a 2 (1 3) b 7))
1
> (list-index symbol? '(a (b c) 17 foo))
0
> (list-index symbol? '(1 2 (a b) . 3))
#f

E1.22 (filter-in pred lst) returns the list of those elements in lst that satisfy the predicate pred.

```
> (filter-in number? '(a 2 (1 3) b 7))
(2 7)
> (filter-in symbol? '(a (b c) 17 foo))
(a foo)
```

E1.21 (product sos1 sos2), where sos1 and sos2 are each a list of symbols without repetitions, returns a list of 2-lists that represents the Cartesian product of sos1 & sos2. the 2-lists ~~may~~ may appear in any order.

```
> (product '(a b c) '(x y))
((a x) (a y) (b x) (b y) (c x) (c y))
```

E1.20 (count-occurrences s slist) returns the number of occurrences of s in slist.

```
> (count-occurrences 'x '((f x) y (((x z) x))))
3
> (count-occurrences 'x '((f x) y (((x z) () x))))
3
> (count-occurrences 'w '((f x) y (((x z) () x))))
0
```

E1.19 (list-set lst n x) returns a list like lst, except that the n-th element, using zero-based indexing, is x.

> (list-set '(a b c d) 2 '(1 2))
(a b (1 2) d)
> (list-ref (list-set '(a b c d) 3 '(1 5 10)) 3)
(1 5 10)

E 1.18 (swapper s1 s2 slist) returns a list, the same as slist, but with all occurrences of s1 replaced by s2 and all occurrences of s2 replaced by s1.

> (swapper 'a 'd '(a b c d))
(d b c a)
> (swapper 'a 'd '(a d () c d))
(d a () c a)
> (swapper 'x 'y '((x) y (z (x))))
((y) x (z (y)))

E 1.17 (down lst) wraps parentheses around each top-level element of lst.

> (down '(1 2 3))
((1) (2) (3))
> (down '(a (fine) (idea)))
((a) (fine) (idea))
> (down '(a (more (complicated)) object))
((a) ((more (complicated))) (object))

E1.16 (invert lst), where lst is a list of 2-lists (lists of length 2), returns a list with each 2-list reversed.

```
> (invert '((a 1) (a 2) (1 b) (2 b)))
((1 a) (2 b) (b 1) (b 2))
```

E1.15 (duple n x) returns a list containing n copies of x.

```
> (duple 2 3)
(3 3)
> (duple 4 '(ha ha))
((ha ha) (ha ha) (ha ha) (ha ha))
> (duple 0 '(blah))
()
```