


List-of-Posn

```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)
```


List-of-Posn

```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)
```




List-of-Posn

```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)  
  
; A posn is  
; (make-posn num num)
```

Two yellow arrows originate from the text 'list-of-posn' in the first line. One arrow points to the word 'posn' in the second line, and the other points to the word 'posn' in the third line.

List-of-Posn


```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)  
  
; A posn is  
; (make-posn num num)
```



```
; func-for-lop : list-of-posn -> ...  
(define (func-for-lop l)  
  (cond  
    [(empty? l) ...]  
    [(cons? l) ...])))
```

List-of-Posn


```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)  
  
; A posn is  
; (make-posn num num)
```



```
; func-for-lop : list-of-posn -> ...  
(define (func-for-lop l)  
  (cond  
    [(empty? l) ...]  
    [(cons? l)  
     ... (first l)  
     ... (rest l) ...])))
```

List-of-Posn


```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)  
  
; A posn is  
; (make-posn num num)
```



```
; func-for-lop : list-of-posn -> ...  
(define (func-for-lop l)  
  (cond  
    [(empty? l) ...]  
    [(cons? l)  
     ... (first l)  
     ... (func-for-lop (rest l)) ...])))
```

List-of-Posn

```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)  
  
; A posn is  
; (make-posn num num)
```



```
; func-for-lop : list-of-posn -> ...  
(define (func-for-lop l)  
  (cond  
    [(empty? l) ...]  
    [(cons? l)  
     ... (func-for-posn (first l))  
     ... (func-for-lop (rest l)) ...]))  
  
; func-for-posn : posn -> ...  
(define (func-for-posn p)  
  ... (posn-x p) ... (posn-y p) ...)
```

List-of-Posn

```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)  
  
; A posn is  
; (make-posn num num)
```

```
; func-for-lop : list-of-posn -> ...  
(define (func-for-lop l)  
  (cond  
    [(empty? l) ...]  
    [(cons? l)  
     ... (func-for-posn (first l))  
     ... (func-for-lop (rest l)) ...]))  
  
; func-for-posn : posn -> ...  
(define (func-for-posn p)  
  ... (posn-x p) ... (posn-y p) ...)
```