

## More List-of-Num Examples

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

Implement the function **feed-fish**, which takes an aquarium and feeds each fish 1 lb of food


Implement the function **large-fish**, which removes every fish that is less than 5 lbs from an aquarium

# List-of-Posn

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

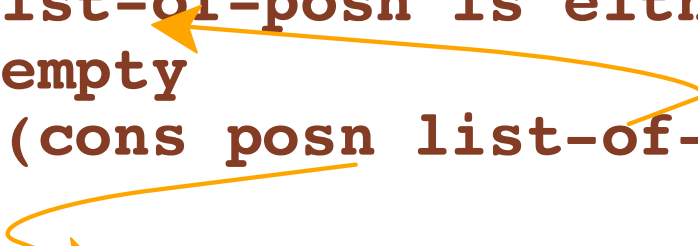
# List-of-Posn

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



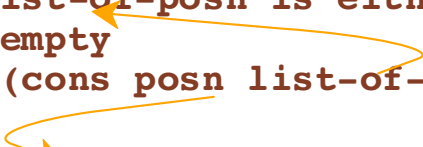
# List-of-Posn

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



# List-of-Posn

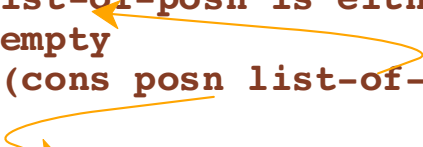
```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)  
  
; A posn is  
; (make-posn num num)
```



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

# List-of-Posn

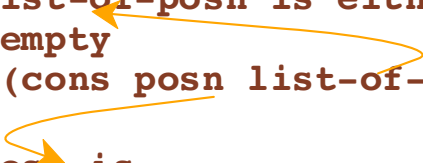
```
; A list-of-posn is either  
; - empty  
; - (cons posn list-of-posn)  
  
; A posn is  
; (make-posn num num)
```



```
; 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) ...)
```

# List-of-Posn Examples

Implement the function **flip-posns**, which flips the X and Y parts of every posn in a list of posns

# List-of-Grade Example

```
; A grade is either  
; - number  
; - empty
```

Implement the function **all-passed?**, which takes a list of grades and determines whether all are passes

# List-of-List-of-Num Example

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

Implement the function **sums**, which takes a list of list-of-numbers and produces a list of sums

# Writing Down Large Lists

What does the list containing 0 to 10 look like?

```
(cons 0 (cons 1 (cons 2 (cons 3 (cons 4 (cons 5 (cons 6 (cons 7 (cons 8 (cons 9 (cons 10 empty))))))))))
```

Here's a shortcut:

```
(list 0 1 2 3 4 5 6 7 8 9 10)
```

The **list** operator takes any number of arguments and constructs a list

Still, DrRacket prints **ll conses**

# Printing Large Lists

If you change DrRacket's language level to

## **Beginning Student with List Abbreviations**

then DrRacket prints using the shortcut

```
> (list 0 1 2 3 4 5 6 7 8 9 10)
(list 0 1 2 3 4 5 6 7 8 9 10)
```

```
> (cons 1 (cons 2 (cons 3 empty)))
(list 1 2 3)
```

# When to Change Language Levels

1. You're not tempted to write examples like this:

```
(check-expect (feed-fish (cons 1 (cons 2 empty)))  
              2 3)
```

2. Your eyes hurt when you see

```
(cons 1 (cons 2))
```

because it isn't a **list-of-num**

3. When you see

```
(list 1 2 3)  
(cons 1 (cons 2 (cons 3 empty)))
```

you recognize instantly that they're the same

Don't switch until you understand how **list-of-...**  
functions match the shape of the data definition

# Even Shorter

For the brave, there's an even shorter shortcut!

`' (1 2 3)`  
is the same as  
`(list 1 2 3)`

The apostrophe above doesn't make a symbol—it makes a list because it precedes a parenthesis

Furthermore, the apostrophe gets distributed to everything inside:

`' (apple banana)`  
is the same as  
`(list 'apple 'banana)`

For consistency, `' 1` is the same as `1`



# Even Shorter

Here's a **list-of-lon** using the shortcut:

```
'((1 2 3) (2 4 6 8) (3 9 27))
```

which is the same as

```
(list (list 1 2 3) (list 2 4 6 8) (list 3 9 27))
```

which is the same as

```
(cons (cons 1 (cons 2 (cons 3 empty))))  
      (cons (cons 2 (cons 4 (cons 6 (cons 8 empty))))  
            (cons (cons 3 (cons 9 (cons 27 empty)))  
                  empty)))
```