Writing Functions in Scheme

• Suppose we want a function \texttt{ct} which takes a list of symbols and returns the number of symbols in the list

\[
(\texttt{ct }'(a\ b\ c)) \rightarrow 3
\]

\[
(\texttt{ct }'(\)) \rightarrow 0
\]

\[
(\texttt{ct }'(x\ y\ z\ w\ t)) \rightarrow 5
\]

How can we write this function?
Answer #1: Have the instructor write it it

;; ct : <list-of-sym>  →  <num>
;;  (ct '()) → 0
;;  (ct '(a b c)) → 3
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))
Checking My Answer: Empty List

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l))))]))
(ct '())

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l))))]))
(cons (cond
        [(null? '()) 0]
        [else (+ 1 (ct (cdr '())))]))
Checking My Answer: Empty List

\[
\text{(define (ct l)} \quad \rightarrow \quad \text{(define (ct l)} \\
\text{ (cond} \quad \quad \text{(cond} \\
\text{ [(null? l) 0]} \quad \quad [(null? l) 0] \\
\text{ [else (+ 1 (ct (cdr l))))])]} \quad \quad [\text{else (+ 1 (ct (cdr l))))]))]
\]

\[
\text{(cond} \quad \quad \text{(cond} \\
\text{ [(null? '()) 0]} \quad \quad [(#t 0] \\
\text{ [else (+ 1 (ct (cdr '())))]}] \quad \quad [\text{else (+ 1 (ct (cdr '())))]})]
\]
(define (ct l)   →    (define (ct l)
 (cond   (cond
  [(null? l) 0]  [(null? l) 0]
  [else (+ 1 (ct (cdr l)))]))    [else (+ 1 (ct (cdr l)))]))

(cond   0
  [#t 0]
  [else (+ 1 (ct (cdr '())))])
Checking My Answer: List of 3 Symbols

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(ct '(a b c))
Checking My Answer: List of 3 Symbols

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))
Checking My Answer: List of 3 Symbols

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1 (ct (cdr '(a b c))))
```
Checking My Answer: List of 3 Symbols

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))])

(+ 1 (ct (cdr '(a b c))))
```

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))])

(+ 1
  (ct '(b c)))
```
Checking My Answer: List of 3 Symbols

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (ct '(b c)))

→

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (cond
    [(null? '(b c)) 0]
    [else (+ 1 (ct (cdr '(b c))))])))
```
Checking My Answer: List of 3 Symbols

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (cond
    [(null? '(b c)) 0]
    [else (+ 1 (ct (cdr '(b c))))])))
Checking My Answer: List of 3 Symbols

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))])))

(+ 1
  (cond
    [#f 0]
    [else (+ 1 (ct (cdr 'b c)))])))
Checking My Answer: List of 3 Symbols

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (ct (cdr '(b c))))))
Checking My Answer: List of 3 Symbols

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (ct '(c))))
```

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (cond
      [(null? '(c)) 0]
      [else (+ 1 (ct (cdr '(c))))])))
```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (cond
      [(null? (c)) 0]
      [else (+ 1 (ct (cdr (c)))]))))

(→
 (define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (cond
      [#f 0]
      [else (+ 1 (ct (cdr (c)))]))))

Checking My Answer: List of 3 Symbols
Checking My Answer: List of 3 Symbols

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (cond
      [#f 0]
      [else (+ 1 (ct (cdr 'c)))])))
```
Checking My Answer: List of 3 Symbols

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (+ 1
      (ct (cdr 'c)))))))
```

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (+ 1
      (ct 'l))))))
```
Checking My Answer: List of 3 Symbols

```scheme
(define (ct l) 
  (cond 
    [(null? l) 0] 
    [else (+ 1 (ct (cdr l)))]))

(+ 1 
  (+ 1 
    (+ 1 
      (ct '()))))
```

```scheme
(define (ct l) 
  (cond 
    [(null? l) 0] 
    [else (+ 1 (ct (cdr l)))]))

(+ 1 
  (+ 1 
    (+ 1 
      (cond 
        [(null? '()) 0] 
        [else (+ 1 (ct (cdr '())))])))
```
Checking My Answer: List of 3 Symbols

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (+ 1
      (cond
        [(null? '()) 0]
        [else (+ 1 (ct (cdr '())))])))))
```

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (+ 1
      (cond
        [#t 0]
        [else (+ 1 (ct (cdr '())))])))))
```
Checking My Answer: List of 3 Symbols

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (+ 1
      (cond
        [#t 0]
        [else (+ 1 (ct (cdr '())))])))))
Checking My Answer: List of 3 Symbols

(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))

(+ 1
  (+ 1
    (+ 1
      0)))
Checking My Answer: List of 3 Symbols

\[
\text{(define (ct l)} \\
\text{(cond)} \\
\text{[ (null? l) 0]} \\
\text{[ else (+ 1 (ct (cdr l))) ]}) \\
\text{)} \\
\text{)
}\]

\[
\text{(define (ct l)} \\
\text{(cond)} \\
\text{[ (null? l) 0]} \\
\text{[ else (+ 1 (ct (cdr l))) ]}) \\
\text{)} \\
\text{)
}\]

\[
(+ 1) \\
(+ 1) \\
(1)) \\
(2)
\]
Checking My Answer: List of 3 Symbols

```
(define (ct l)
  (cond
    [(null? l) 0]
    [else (+ 1 (ct (cdr l)))]))
  (+ 1
    2))

  →  (define (ct l)
        (cond
          [(null? l) 0]
          [else (+ 1 (ct (cdr l)))]))

  3
```
Answer #2: Use the general design recipe

- Locate or write a data definition
- Write a contract
- Write examples
- Create a template that follows the shape of the data definition
- Convert the template to the final function
- Run examples as tests
Writing Functions in Scheme: Answer #2

Answer #2: Use the general design recipe

- Locate or write a data definition
- Write a contract
- Write examples
- Create a template that follows the shape of the data definition
- Convert the template to the final function
- Run examples as tests

works 90% of the time
Data Definitions

What is a "list of symbols"?

\[
\text{<list-of-sym>} ::= '() \\
\quad ::= (\text{cons} \text{<symbol>} \text{<list-of-sym>})
\]

- Sometimes the \textit{data definition} is given, sometimes you have to create it
- Usually include it in your code as a comment
Contracts

A contract is a comment that identifies set of input values and output values

;;; ct: <list-of-sym> -> <num>

• All mentioned data sets should have a data definition somewhere
Examples

Examples (usually in comments at first) help clarify the purpose of the function

```plaintext
;; (ct '()) →→ 0
;; (ct '(a b c)) →→ 3
```

- Make sure that every case in the data definition is covered at least once
A template reflects the structure of the input according to the data definition

\[
\begin{align*}
\langle\text{list-of-sym}\rangle & \ ::= \ '() \ \\
& \ ::= \ (\text{cons} \ <\text{symbol}> \ <\text{list-of-sym}>)
\end{align*}
\]

\text{(define (ct l)}
\text{(cond)}
\text{[(null? l) ...]}\n\text{[(pair? l) ... (car l) ... (ct (cdr l)) ...]})
A template reflects the structure of the input according to the data definition

\[
\text{<list-of-sym>} ::= '\()\n\text{:= (cons <symbol> <list-of-sym>)}
\]

\[
\text{(define (ct l)}
\text{(cond}
\text{[(null? l) ...]}
\text{[(pair? l) ...(car l)...(ct (cdr l))...]})
\]

- Two cases in data definition implies cond with two cond-lines
A template reflects the structure of the input according to the data definition

\[
\text{<list-of-sym>} :: \equiv \ '() \\
\equiv (\text{cons} <\text{symbol}> <\text{list-of-sym}>)
\]

\[
\text{(define (ct l)} \text{ (cond} \\
[\text{(null? l)} \ldots] \\
[\text{(pair? l)} \ldots(\text{car l})\ldots(\text{ct (cdr l)})\ldots])
\]

- Corresponding predicate for each data case
Template

A template reflects the structure of the input according to the data definition

\[
\text{<list-of-sym>} ::= \ '() \\
::= (\text{cons <symbol> <list-of-sym>})
\]

\[
\begin{align*}
\text{(define (ct l)} \\
& \ \text{(cond} \\
& \quad [(\text{null? l}) \ldots] \\
& \quad [(\text{pair? l}) \ldots (\text{car l})\ldots (\text{ct (cdr l)})\ldots] \))
\end{align*}
\]

- Extract parts in cases with meta-variables
Template

A template reflects the structure of the input according to the data definition

\[
\begin{align*}
\text{<list-of-sym>} & : = \ '() \\
& : = (\text{cons} \ <\text{symbol}> \ <\text{list-of-sym}>)
\end{align*}
\]

\[
\text{(define (ct l)} \\
\text{(cond} \\
\text{[ (null? l) ...]} \\
\text{[ (pair? l) ...(car l)...(ct (cdr l))... ]}))
\]

- Recursive call for self-references in data definition
Template

A template reflects the structure of the input according to the data definition

\[
\langle \text{list-of-sym} \rangle ::= \text{'(} \\
\quad ::= (\text{cons} \: \langle \text{symbol} \rangle \: \langle \text{list-of-sym} \rangle)
\]

\[
\text{(define (ct l)} \\
\quad (\text{cond} \\
\quad \quad [(\text{null?} \: l) \: \ldots] \\
\quad \quad [(\text{pair?} \: l) \: \ldots(\text{car} \: l)\ldots(\text{ct} \: (\text{cdr} \: l))\ldots])
\)
\]

- A template depends only on the input data; it ignores the function's purpose

(Nevertheless, generating a template, which is fairly automatic, usually provides most of the function)
Template to Function

Transform template to function line-by-line

(define (ct l)
  (cond
    [(null? l) ...]
    [(pair? l) ... (car l) ... (ct (cdr l)) ...]))
Template to Function

Transform template to function line-by-line

(define (ct l)
  (cond
    [(null? l) 0]
    [(pair? l) ...(car l)...(ct (cdr l))...])))
Template to Function

Transform template to function line-by-line

\[
\text{(define } (\text{ct } l) \\
\text{ (cond} \\
\quad [(\text{null? } l) 0] \\
\quad [(\text{pair? } l) (+ 1 (\text{ct } (\text{cdr } l)) )])\]
\]

- Sometimes, a part of the template isn't needed
Reminder: Recipe

- Locate or write a data definition
- Write a contract
- Write examples
  
  - Create a template that follows the shape of the data definition
  - Convert the template to the final function
  - Run examples as tests
Reminder: Template Steps

- Create a **cond** expression with one line for each case in the data definition
- Write down a predicate for each case
- For the answer, extract parts in cases with meta-variables
- For each self-reference in the data definition, add a recursive call

Shape of template shape == Shape of data definition
More Examples

(more examples in class)
Generalized Recipe

- Locate or write data definitions
- Write contracts
- Write examples
- Create a template that follows the shape of the data definition, one for each data definition
- Convert the templates to the final functions
- Run examples as tests