Helper Functions and Reuse

- Conditionals
- Evaluation Rules for cond
- Design Recipe with cond
- Compound Data
Designing Programs

Design recipe

• As outlined last lecture

Helper functions and reuse

• Writing writing a function, consider whether existing functions help
  ○ Example: `wearing-glasses?` uses `add-glasses`

• Look for functions that you wish you had written
  ○ Example: `same-person-maybe-disguised?` needs `wearing-beard?`
Another Example

Write the function `bigger-image?` which checks whether one image has more pixels than a second image.
Another Example

Write the function `bigger-image?` which checks whether one image has more pixels than a second image

; bigger-image? : image image -> bool
Another Example

Write the function `bigger-image?` which checks whether one image has more pixels than a second image

```scheme
; bigger-image? : image image -> bool
; Returns true if a has more pixels than b
```
Another Example

Write the function `bigger-image?` which checks whether one image has more pixels than a second image

```scheme
; bigger-image? : image image -> bool
; Returns true if a has more pixels than b
(define (bigger-image? a b) ...)
```
Another Example

Write the function `bigger-image?` which checks whether one image has more pixels than a second image.

; `bigger-image?` : image image -> bool
; Returns true if a has more pixels than b
(define (bigger-image? a b) ...)
Another Example

Write the function `bigger-image?` which checks whether one image has more pixels than a second image

```scheme
; bigger-image? : image image -> bool
; Returns true if a has more pixels than b
(define (bigger-image? a b)
  (> (* (image-width a) (image-height a))
      (* (image-width b) (image-height b))))

(bigger-image? ■ ■) "should be" true
(bigger-image? ■ ■) "should be" false
```
Write the function \texttt{bigger-image?} which checks whether one image has more pixels than a second image

\begin{verbatim}
; bigger-image? : image image -> bool
; Returns true if a has more pixels than b
(define (bigger-image? a b)
  (> (image-size a) (image-size b)))
\end{verbatim}

\begin{quote}
\texttt{(bigger-image? ■ ■)} "should be" true
\texttt{(bigger-image? ■ ■■)} "should be" false
\end{quote}

\textit{Wish list: image-size}
Another Example

Write the function `bigger-image?` which checks whether one image has more pixels than a second image

; `bigger-image?: image image -> bool`
; Returns true if a has more pixels than b
(define (bigger-image? a b)
 (>', (image-size a) (image-size b)))

(bigger-image? □ □) "should be" true
(bigger-image? □ ■) "should be" false

Wish list: `image-size`

Fullfill wishes by applying the recipe again

(exercise for the reader)
Reuse

We should be able to use `bigger-image?` to write the `max-image` function
We should be able to use `bigger-image?` to write the `max-image` function

```
; max-image : image image -> image
; Returns a if a has more pixels than b,
; otherwise returns b
(define (max-image a b) ...)
```
Reuse

We should be able to use `bigger-image?` to write the `max-image` function

; max-image : image image -> image
; Returns a if a has more pixels than b, otherwise returns b
(define (max-image a b) ...)

(max-image □ □) "should be" □
(max-image □ □) "should be" □
We should be able to use `bigger-image?` to write the `max-image` function

\[
\text{; max-image : image image -> image}
\text{; Returns a if a has more pixels than b, otherwise returns b}
\text{(define (max-image a b)}
\text{ ... (bigger-image? a b) ...)}
\]

\[
\text{(max-image ■ ■) "should be" ■}
\text{(max-image ■ ■) "should be" ■}
\]
Reuse

We should be able to use `bigger-image?` to write the `max-image` function

```scheme
; max-image : image image -> image
; Returns a if a has more pixels than b,
; otherwise returns b
(define (max-image a b)
  ... (bigger-image? a b) ...)

(max-image    ■ ■) "should be" ■
(max-image ■ ■) "should be" ■
```

Instead of returning a `bool`, we need to do one of two things, so we need `cond`
Helper Functions and Reuse

Conditionals

Evaluation Rules for cond

Design Recipe with cond

Compound Data
Conditionals in Algebra

General format of conditionals in algebra:

\[
\{ \quad \text{answer} \quad \text{question} \\
\text{...} \\
\text{answer} \quad \text{question} \\
\}
\]

Example:

\[
\text{abs}(x) = \begin{cases} 
  x & \text{if } x > 0 \\
  -x & \text{otherwise}
\end{cases}
\]

\[
\text{abs}(10) = 10 \\
\text{abs}(-7) = 7
\]
Conditionals

General syntax of cond in our language:

```
(cond
    [question answer]
    ...
    [question answer])
```

- Any number of cond lines
- Each line has one question expression and one answer expression

```
(define (abs x)
  (cond
    [(> x 0) x]
    [else (- x)]))

(abs 10) "should be" 10
(abs -7) "should be" 7
```
Completing max-image

• Use cond to complete `max-image`

```
(define (max-image a b)
  (cond
    [(bigger-image? a b) a]
    [else b]))
```
Helper Functions and Reuse
Conditionals
Evaluation Rules for cond
Design Recipe with cond
Compound Data
Evaluation Rules for cond

First question is literally **true** or **else**

```
(cond
  [true answer] → answer
  ...
  [question answer])
```

• Keep only the first answer

Example:

```
(* 1 (cond ... → (* 1 0) → 0
  [true 0]))
```
Evaluation Rules for cond

First question is literally \textit{false}

\begin{verbatim}
(cond
  [false answer]
  [question answer]
  ...
  [question answer])
\end{verbatim}

• Throw away the first line

Example:

\begin{verbatim}
(+ 1 (cond
  [false 1]
  [true 17]))
\end{verbatim}

\begin{verbatim}
→ (+ 1 17) → 18
\end{verbatim}
Evaluation Rules for cond

First question isn’t a value, yet

\[
\text{(cond [question answer] … [question answer])} \rightarrow \text{(cond [nextques answer] … [question answer])}
\]

where \text{question} \rightarrow \text{nextques}

• Evaluate first question as sub-expression

Example:

\[
(+ 1 \text{(cond [ (< 1 2) 5] [else 8] ))) \rightarrow (+ 1 \text{(cond [true 5] [else 8] ))) \rightarrow (+ 1 5) \rightarrow 6
\]
Evaluation Rules for cond

Only question is false answers

\[
\begin{align*}
&\text{(cond} \\
&\quad [\text{false} \ 10]) \\
\end{align*}
\rightarrow \ \text{error: all questions false}
\]
Helper Functions and Reuse
Conditionals
Evaluation Rules for \texttt{cond}
Design Recipe with \texttt{cond}
Compound Data
When the problem statement divides the input into several categories, test each one

Example:

Write the function \texttt{line-part} that determines whether a number is on zero, to the left, or to the right on a number line

\begin{itemize}
\item \texttt{(line-part 0)} "should be" 'zero
\item \texttt{(line-part -3)} "should be" 'left
\item \texttt{(line-part 3)} "should be" 'right
\end{itemize}
When the problem statement divides the input into $N$ categories:

• Start the body with a `cond` expression and $N$ lines

• Formulate a question to recognize each category

Example:

Write the function `line-part` that determines whether a number is on zero, to the left, or to the right on a number line

Three cases, so three lines:

```scheme
(define (line-part n)
  (cond
    [(= n 0) ...]
    [(< n 0) ...]
    [(> n 0) ...]))
```
- Helper Functions and Reuse
- Conditionals
- Evaluation Rules for \( \text{cond} \)
- Design Recipe with \( \text{cond} \)
- Compound Data
Finding Images

(image-inside? true) → true

(image-inside? false) → false
Now we can combine such operators with \texttt{cond}:

\begin{verbatim}
; detect-person : image image image image → image
; Returns a or b, depending on which is in i
(define (detect-person i a b)
  (cond
    [(image-inside? i a) a]
    [(image-inside? i b) b]))
\end{verbatim}

\begin{verbatim}
(detect-person )
"should be"
\end{verbatim}
Finding and Adjusting Images

Suppose we want to write \texttt{frame-person}: 

\[
\texttt{(frame-person )}
\]

"should be"

Need an operator that reports where an image exists
Finding an Image Position

\texttt{find-image : image image \rightarrow num num}

Must return a single value

Correct contract:

\texttt{find-image : image image \rightarrow posn}

- A \texttt{posn} is a \textit{compound value}
Positions

• A posn is

\[(\text{make-posn } X \ Y)\]

where \(X\) is a num and \(Y\) is a num

Examples:

\[(\text{make-posn } 1 \ 2)\]

\[(\text{make-posn } 17 \ 0)\]

A posn is a value, just like a number, symbol, or image
posn-x and posn-y

The \texttt{posn-x} and \texttt{posn-y} operators extract numbers from a \texttt{posn}:

\[
\begin{align*}
\text{(posn-x (make-posn 1 2))} & \rightarrow 1 \\
\text{(posn-y (make-posn 1 2))} & \rightarrow 2
\end{align*}
\]

- General evaluation rules for any \texttt{X} and \texttt{Y}:

\[
\begin{align*}
\text{(posn-x (make-posn X Y))} & \rightarrow X \\
\text{(posn-y (make-posn X Y))} & \rightarrow Y
\end{align*}
\]
Positions and Values

Is \texttt{(make-posn 100 200)} a value?

Yes.

A \texttt{posn} is

\texttt{(make-posn X Y)}

where \texttt{X} is a \texttt{num} and \texttt{Y} is a \texttt{num}
Positions and Values

Is \texttt{(make-posn (+ 1 2) 200)} a value?

\textbf{No.} \( (+ 1 2) \) is not a \texttt{num}, yet.

- Two more evaluation rules:
  \[
  \begin{align*}
    & (\texttt{make-posn } X \ Y) \rightarrow (\texttt{make-posn } Z \ Y) \\
    & \quad \text{when } X \rightarrow Z \\
    & (\texttt{make-posn } X \ Y) \rightarrow (\texttt{make-posn } X \ Z) \\
    & \quad \text{when } Y \rightarrow Z
  \end{align*}
  \]

Example:

\[
(\texttt{make-posn } (+ 1 2) \ 200) \rightarrow (\texttt{make-posn } 3 \ 200)
\]
More Examples

Try these in DrScheme's stepper:

\[
\text{(make-posn (+ 1 2) (+ 3 4))}
\]

\[
\text{(posn-x (make-posn (+ 1 2) (+ 3 4)))}
\]

; pixels-from-corner : posn -> num
\[
\text{(define (pixels-from-corner p)}
\quad (+ \text{(posn-x p) (posn-y p)})
\text{)}
\]

\[
\text{(pixels-from-corner (make-posn 1 2))}
\]

; flip : posn -> posn
\[
\text{(define (flip p)}
\quad (\text{make-posn (posn-y p) (posn-x p)})
\text{)}
\]

\[
\text{(flip (make-posn 1 2))}
\]