Goals of this tutorial

You should be able to...

- correctly apply **stepping rules**.
- understand and use **structures**.
Aside: Using Boolean Arguments

Compare:

;;; f: Bool Bool → Sym
(define (f x y)
  (cond
    [(boolean=? x true) 'red]
    [(boolean=? y false) 'blue]
    [else 'yellow])))

;;; f: Bool Bool → Sym
(define (f x y)
  (cond
    [x 'red]
    [(not y) 'blue]
    [else 'yellow])))
Review: Stepping Rules

Always evaluate the **topmost, leftmost** unsimplified expression first.

**Application of built-in functions:** \((f \ v_1 \ldots \ v_n) \Rightarrow v\) where \(f\) is a built-in function and \(v\) is the value of \(f(v_1, \ldots, v_n)\)

**Substitution of Constants:** \(id \Rightarrow val\), where \((\text{define id val})\) occurs to previously.
Review: Stepping Rules

Application of user-defined functions: The general substitution rule is:

\[(f \ v1 \ldots \ vn) \Rightarrow \text{exp'}\]

where \((\text{define} \ (f \ x1 \ldots \ xn) \ \text{exp})\) occurs previously, and \text{exp'} is obtained by substituting all occurrences of the formal parameter \(x_i\) replaced by the value \(v_i\) (for \(i\) from 1 to \(n\)) into the expression.
Group Problem - Stepping \texttt{middle}

The following definition has been processed in the Beginning Student language:

\begin{verbatim}
(define (middle x y z)
  (+ x y z (- (min x y z)) (- (max x y z))))
\end{verbatim}

Step through the following:

\texttt{(middle 4 16 7)}
Clicker Question

What are the next two steps for this code? (Do not skip any steps.)

(define x 3)
(define (foo a b) (+ a b x))
(foo 1 x)

A \Rightarrow (+ 1 3 3) \Rightarrow 7
B \Rightarrow (+ 1 3 x) \Rightarrow (+ 1 3 3)
C \Rightarrow (foo 1 3) \Rightarrow (+ 1 3 3)
D \Rightarrow (foo 1 3) \Rightarrow (+ 1 3 x)
Review: Stepping Rules
Substitution in cond expressions

There are three rules: when the first expression is false, when it is true, and when it is else.

\[
\text{(cond [false exp] \ldots ) } \Rightarrow \text{(cond \ldots )}
\]

\[
\text{(cond [true exp] \ldots ) } \Rightarrow \text{exp}
\]

\[
\text{(cond [else exp]) } \Rightarrow \text{exp}
\]

These suffice to simplify any cond expression.
Group Problem - Stepping **cond**

The following have been processed in the Beginning Student language:

```
(define x 1)
(define y 1)
```

Step through the following:

```
(cond [(= x 0) 'one]
      [else (< (/ y x) c)])
```
Review: Stepping Rules

Simplification Rules for and and or

The simplification rules we use for Boolean expressions involving and and or differ from the ones the Stepper in DrRacket uses.

- \( (\text{and} \ false \ldots) \Rightarrow \text{false} \)

- \( (\text{and} \ true \ldots) \Rightarrow (\text{and} \ldots) \)

- \( (\text{and}) \Rightarrow \text{true} \)

- \( (\text{or} \ true \ldots) \Rightarrow \text{true} \)

- \( (\text{or} \ false \ldots) \Rightarrow (\text{or} \ldots) \)

- \( (\text{or}) \Rightarrow \text{false} \)
Group Problem - Stepping and

The following have been processed in the Beginning Student language:

(define x 0)
(define y (+ x 1))

Step through the following:
(and (not (= x 0)) (<= (/ y x) c))
Review: Posn structures

- **constructor** function `make-posn`, with contract
  
  `;; make-posn: Num Num → Posn`

- **selector** functions `posn-x` and `posn-y`, with contracts
  
  `;; posn-x: Posn → Num`
  
  `;; posn-y: Posn → Num`

Example:

```scheme
(define mypoint (make-posn 8 1))

(posn-x mypoint) ⇒ 8

(posn-y mypoint) ⇒ 1
```
Review: Posn structures

- posn?, with contract

;; posn?: Any → Bool

Example:

(posn? (make-posn 5 4)) ⇒ true

(posn? (make-posn 'red "snake") ) ⇒ true
Review: Posn structures

Possible uses:

- coordinates of a point on a two-dimensional plane
- positions on a screen or in a window
- a geographical position

Note:

- An expression such as \( \text{(make-posn } 8 \ 1) \) is considered a value, which will not be simplified further by the Stepper or our semantic rules.
- The expression \( \text{(make-posn } (\text{+ } \ 4 \ 4) \ (\text{– } \ 3 \ 2)) \) would be simplified further to (eventually) \( \text{(make-posn } 8 \ 1) \).
Review - User-defined Structures

Consider the following structure used to represent a burger:

(define-struct burger (patty veg-count cheese?))

;; A Burger is a (make-burger Sym Nat Bool)
;; requires: patty is one of ('veggie, 'chicken, 'beef)
;; veg-count <= 10

Note that the veg-count field represents the number of vegetable toppings.
Review - User-defined Structures

(define veg-burger (make-burger 'veggie 6 false))
(define big-burger (make-burger 'beef 10 true))

(burger-patty veg-burger)
⇒ 'veggie

(burger-veg-count big-burger)
⇒ 10

(burger-cheese? big-burger)
⇒ true
Group Problem - Structures

Write a template function for Burger called my-burger-fn.
Group Problem - Structures

Burger prices are decided by the following rules:

- veggie patties cost $4
- chicken patties cost $5.5
- beef patties cost $7.5
- each vegetable topping costs $0.35
- cheese costs $1.75

Write a function `burger-price` that consumes a `Burger` and produces the price of that burger. Include a design recipe.