Goals of this tutorial:
By the end of this tutorial you should be able to...

- **Step** through a given piece of code with mathematical and boolean functions, and conditional statements.
- Navigate flat lists and lists with one level of nesting.
- Utilize list functions.

Clicker Question: Debugging Conditional Statements
Will the below function always produce the correct result?

```scheme
;; remainder-by-4: Nat → Str
(define (remainder-by-4 n)
  (cond
    [(= (remainder n 4) 1) "n-1 is divisible by 4"]
    [(= (remainder n 4) 2) "n-2 is divisible by 4"]
    [else "n-3 is divisible by 4"])
```

A  Yes
B  No
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 previously.

---

Review: Stepping Rules

**Application of user-defined functions:** The general substitution rule is:
\[(f \ v_1 \ldots \ v_n) \Rightarrow \text{exp'}\]
where \((\text{define } (f \ x_1 \ldots \ x_n) \ \text{exp})\) 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.

---

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

\(\text{(define } x \ 5)\)
\(\text{(define } (\text{foo } a \ b) (\ + \ a \ b \ x \ (\text{max } a \ (\text{sqr } b))))\)
\(\text{(foo } 1 \ x)\)

\(A \Rightarrow (\ + \ 1 \ 5 \ 5 \ (\text{max } 1 \ (\text{sqr } 5))) \Rightarrow (\ + \ 1 \ 5 \ 5 \ (\text{max } 1 \ 25))\)
\(B \Rightarrow (\ + \ 1 \ 5 \ x \ (\text{max } 1 \ (\text{sqr } 5))) \Rightarrow (\ + \ 1 \ 5 \ 5 \ (\text{max } 1 \ (\text{sqr } 5)))\)
\(C \Rightarrow (\text{foo } 1 \ 5) \Rightarrow (\ + \ 1 \ 5 \ 5 \ (\text{max } 1 \ (\text{sqr } 5)))\)
\(D \Rightarrow (\text{foo } 1 \ 5) \Rightarrow (\ + \ 1 \ 5 \ x \ (\text{max } 1 \ (\text{sqr } 5)))\)
Clicker Question
The following definitions have been processed:

\[
\text{(define x 10)}
\]
\[
\text{(define y (+ x x))}
\]

what are the next two steps for this code?

\[
(+ y y)
\]

\[
\begin{align*}
A \Rightarrow (+ (+ x x) y) & \Rightarrow (+ (+ 10 x) y) \\
B \Rightarrow (+ (+ 10 x) y) & \Rightarrow (+ (+ 10 10) y) \\
C \Rightarrow (+ 20 20) & \Rightarrow 40 \\
D \Rightarrow (+ 20 y) & \Rightarrow (+ 20 20)
\end{align*}
\]

Review: Stepping Rules
Substitution in cond expressions

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

\[
\begin{align*}
\text{(cond [false exp] \ldots) } & \Rightarrow \text{(cond \ldots)} \\
\text{(cond [true exp] \ldots) } & \Rightarrow \text{exp} \\
\text{(cond [else exp]) } & \Rightarrow \text{exp}
\end{align*}
\]

These suffice to simplify any \text{cond} expression, note the error case too:

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

Group Problem - Stepping \text{cond}

The following have been processed in the Beginning Student language:

\[
\begin{align*}
\text{(define x 1)} \\
\text{(define y 1)}
\end{align*}
\]

Step through the following:

\[
\text{(cond [(= x 0) 'one]} \\
\text{[else (\lt (/ y x) c)])}
\]
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.

\[(\text{and} \ false \ . \ . \ .) \Rightarrow \text{false}\]
\[(\text{and} \ true \ . \ . \ .) \Rightarrow (\text{and} \ . \ . \ .)\]
\[(\text{and}) \Rightarrow \text{true}\]
\[(\text{or} \ true \ . \ . \ .) \Rightarrow \text{true}\]
\[(\text{or} \ false \ . \ . \ .) \Rightarrow (\text{or} \ . \ . \ .)\]
\[(\text{or}) \Rightarrow \text{false}\]

Group Problem - Stepping and

The following have been processed in the Beginning Student language:

\[(\text{define} \ x \ 0)\]
\[(\text{define} \ y \ (+ \ x \ 1))\]

Step through the following:
\[(\text{and} \ (\text{not} \ (= \ x \ 0)) \ (\leq \ (/ \ y \ x) \ c))\]

Review: Lists

A list is a recursive structure, meaning it is made by combining a value with another list.

Think of Russian dolls:

- A list of 3 dolls, is a single doll with 2 dolls inside of it.
- A list of 2 dolls, is a single doll with 1 doll inside of it.
- A list of 1 doll, is a single doll with nothing inside of it.
- A list of 0 dolls is special. It is an empty list however, it is still a list.
Review: Making Lists

In beginner language you can make lists using the `(cons value list)` function. Lists can take in any data type as shown in the example below.

```scheme
;; A (listof Any) is one of:
;; * empty
;; * (cons Any (listof Any))
(define doll-set (cons "blue" (cons 'green (cons true empty))))
```

However, lists can also be restricted to take in very specific data.

```scheme
;; A (listof Int) is one of:
;; * empty
;; * (cons Int (listof Int))
(define no-scope (cons 3 (cons 6 (cons 0 empty))))
```

Problem 0: second, third, and fourth

Using only `first` and `rest`, define functions `my-second`, `my-third` and `my-fourth` that do the same thing as the built-in functions `second`, `third`, and `fourth`. You don’t have to write the entire recipe, but include contracts.

Clicker: Accessing an element in a list

Which of the following options will produce ‘kittens’ from `lst`?

```scheme
(define lst (cons 'raindrops (cons 'roses (cons (cons 'whiskers (cons 'kittens empty))

  (cons 'kettles (cons 'mittens (cons 'paper-packages (cons 'strings empty))

  empty)))))
```

A `(first (rest (rest lst)))`
B `(first (rest (first (rest (rest lst))))))`
C `(first (rest (rest (rest lst))))`
D `(first (fourth lst))`
E None of the above
Problem 1: 3D → 2D
You have been provided a set of 3-dimensional vectors in the form (cons x (cons z (cons y empty))).
Define a function that consumes a vector and produces a plane of the X and Y Values.

Problem 1: 3D → 2D - Design Recipe
;; (change-dimension vector) Consumes a vector and produces a plane.
;; change-dimension: (listof Num) → (listof Num)
;; requires: vector must be a 3 element list
;; Example:
(check-expect (change-dimension (cons 1 (cons 2 (cons -3 empty)))
  (cons 1 (cons -3 empty))))
(define (change-dimension vector) . . .)
;; Tests:
(check-expect (change-dimension (cons 0 (cons 2 (cons .5 empty)))
  (cons 0 (cons .5 empty))))
(check-expect (change-dimension (cons 1 (cons 2 (cons -3 empty)))
  (cons 1 (cons -3 empty))))

Extra Practice: Stepping
We will not cover these questions in the tutorial, unless we have some time left over in the end. Solutions will not be posted to these questions - you may discuss these problems with classmates and work together to come up with a solution.

Complete the practice questions on the Stepping webpage of the course practice. Try Modules 2a, 2b, 2c, 4a, and 4b.
Extra Practice: Triangle Touch

We will not cover these questions in the tutorial, unless we have some time left over in the end. Solutions will not be posted to these questions - you may discuss these problems with classmates and work together to come up with a solution.

For this question you will be working with a Cartesian graph ranging from 0 to 10 in both the X and Y direction. A triangle will be formed of the three points (cons 0 (cons 0 empty)), (cons 10 (cons 0 empty)), and a third coordinate entered by the use as the first parameter of the function. Write a function called “in-triangle” that consumes one point to complete a triangle with the points (0, 0) and (10, 0). Then consumes a second target point. If the the second point is on or within the area of the triangle produce true, otherwise produce false.

Triangle Touch: Hint 1

The first challenge we face when approaching this problem is how do we make a triangle? It might help up to think of this triangle as two slopes that meet at the X coordinate of peak, since regardless of where we put this point it will always result in two right-angle triangles. With this in mind, we can tackle a smaller portion of the large question, how do we form a slope from (make-posn 0 0) to (make-posn peak-x peak-y)?

Triangle Touch: Hint 2

Now the second problem we face is how to determine if a point falls within the area of the slope? Since we know exactly what Y value will be produced given your new found slope, we can just compare the point-y value to the y-value produced from running the slope with the point-x value. If the point-y value is greater than the produced value, then you know it is not in or on the slope’s area, otherwise you know it is within the slope’s area.