<table>
<thead>
<tr>
<th>Lecture</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Lecture 05</td>
<td>12</td>
</tr>
<tr>
<td>5.1</td>
<td>Administrivia</td>
<td>12</td>
</tr>
<tr>
<td>5.2</td>
<td>String Functions</td>
<td>12</td>
</tr>
<tr>
<td>5.3</td>
<td>Helper Functions</td>
<td>12</td>
</tr>
<tr>
<td>5.4</td>
<td>Using Constants</td>
<td>13</td>
</tr>
<tr>
<td>5.5</td>
<td>Summary of Module 2</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Lecture 06</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>Lecture 07</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>Lecture 08</td>
<td>13</td>
</tr>
<tr>
<td>8.1</td>
<td>Administrivia</td>
<td>14</td>
</tr>
<tr>
<td>8.2</td>
<td>Posn Structures</td>
<td>14</td>
</tr>
<tr>
<td>8.3</td>
<td>Dynamic Typing</td>
<td>15</td>
</tr>
<tr>
<td>8.4</td>
<td>Data Definitions / Structure Definitions</td>
<td>15</td>
</tr>
<tr>
<td>8.5</td>
<td>Design Recipe Modifications</td>
<td>16</td>
</tr>
<tr>
<td>8.6</td>
<td>Templates</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>Lecture 09</td>
<td>16</td>
</tr>
<tr>
<td>9.1</td>
<td>Administrivia</td>
<td>17</td>
</tr>
<tr>
<td>9.2</td>
<td>Additions to Syntax/Semantics for Structures</td>
<td>17</td>
</tr>
<tr>
<td>9.3</td>
<td>More Examples</td>
<td>17</td>
</tr>
<tr>
<td>9.4</td>
<td>Using the Design Recipe for Compound Data</td>
<td>20</td>
</tr>
<tr>
<td>9.5</td>
<td>Mixed Data and Structures</td>
<td>20</td>
</tr>
<tr>
<td>9.6</td>
<td>Summary of M4</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>Lecture 10</td>
<td>20</td>
</tr>
<tr>
<td>10.1</td>
<td>Administrivia</td>
<td>21</td>
</tr>
<tr>
<td>10.2</td>
<td>Data Definitions for Lists</td>
<td>21</td>
</tr>
<tr>
<td>10.3</td>
<td>Template for a List of Symbols</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>Lecture 11</td>
<td>24</td>
</tr>
<tr>
<td>11.1</td>
<td>Administrivia</td>
<td>24</td>
</tr>
<tr>
<td>11.2</td>
<td>Template for a List of Symbols</td>
<td>24</td>
</tr>
<tr>
<td>11.3</td>
<td>Structural Recursion</td>
<td>25</td>
</tr>
<tr>
<td>11.4</td>
<td>Examples</td>
<td>25</td>
</tr>
<tr>
<td>11.5</td>
<td>Wrapping a Function</td>
<td>25</td>
</tr>
<tr>
<td>11.6</td>
<td>String Functions</td>
<td>25</td>
</tr>
</tbody>
</table>
1 Lecture 01

Outline

1. Administrivia
2. Introduction to CS 115 - Course Website and Slides 1-9
3. Introduction to DrRacket - Slides 10-20

1.1 Administrivia

1. Labs start this week.
2. I will lecture on the blackboard using the course slides as a resource. I will announce pre-reading for each future lecture.
3. Register your iClicker exactly once. The course staff will contact you at the end of January if your iClicker is not yet correctly registered.
1.2 Introduction to CS 115 - Course Website and Slides 1-9

Refer to the slides.

1.3 Introduction to DrRacket - Slides 10-20

Programming language design

- Imperative:
  - frequent changes to data
  - examples: machine language, Java, C++
- Functional:
  - examples: Excel formulas, LISP, ML, Haskell, Mathematica, XSLT, R (used in STAT 231)
  - more closely connected to mathematics
  - easier to design and reason about programs

We use DrRacket, a functional program, in CS 115. DrRacket is great for teaching, although I have yet to see a real computer system developed in it. DrRacket is Turing Complete, so in theory any computer system could be developed using it. DrRacket provides an easy entry point into coding.

You will work in the imperative language, Python, in CS 116. Real computer systems are written in Python.

Themes of the course

- design (the art of creation)
- abstraction (finding commonality, not worrying about details)
- refinement (revisiting and improving initial ideas)
- syntax (how to say it), expressiveness (how easy it is to say and understand), and semantics (the meaning of what’s being said)
- communication (in general)

Functions: A mathematical function definition consists of

- the name of the function,
- its parameters (aka argument) (what the function consumes), and
- a mathematical expression using the parameters, to define what the function produces. The mathematical expression is evaluated by substitution.
Functions in DrRacket:

- As in the slides, we start with \( f(x) = x^2 \) and \( g(x, y) = x - y \).
- Include exactly one set of parentheses for each function call.
- Write the function first inside the open parenthesis, followed by the arguments.
- Observe that following this syntax removes any ambiguity about the order of operations.
- Using this setup, try some examples of your own in DrRacket.

2 Lecture 02

Outline

1. Administrivia
2. Mathematical Functions in DrRacket
3. The DrRacket Environment
4. Defining New Functions in DrRacket

2.1 Administrivia

1. Email Barb Daly (contact information in Personnel page) with your issues with iClicker registration or Markus setup.

2.2 Mathematical Functions in DrRacket

<table>
<thead>
<tr>
<th>Familiar Math Notation</th>
<th>DrRacket</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 - 2</td>
<td>(- 3 2)</td>
</tr>
<tr>
<td>3 - 2 + 4/5</td>
<td>(+ (- 3 2) (/ 4 5))</td>
</tr>
<tr>
<td>(6 - 4)(3 + 2)</td>
<td>(* (- 6 4) (+ 3 2))</td>
</tr>
</tbody>
</table>

- CQ 2
- CQ 3
2.3 The DrRacket Environment

Built-in Functions - Examples

Integer division (in the positive integers to keep things simple for a start) produces a quotient and remainder, for example

- Dividing 17 by 5 produces the quotient 3 and the remainder 2 because $17 = 3 \cdot 5 + 2$.
- Dividing 15 by 5 produces the quotient 3 and the remainder 0 because $15 = 3 \cdot 5 + 0$.
- If dividing $a$ by $b$ produces the quotient $q$ and the remainder $r$, so that $a = q \cdot b + r$, then requiring $0 \leq r < b$ makes the choice of $q$ and $r$ unique, so that quotient and remainder are actually functions.

- $(\text{quotient } 75 7)$
- $(\text{remainder } 75 7)$

Bad Syntax / Semantics - Examples

- $(5)$
- $(+ (* 2 4))$
- $(5 * 14)$
- $(* + 3 5 2)$
- $(/ 25 0)$

- CQ 4

2.4 Defining New Functions in DrRacket

- DrRacket enforces correct syntax at all times.
- If you enter an expression which is not syntactically correct, then you will get an error message.
- Every syntactically correct DrRacket function call has the form $(\text{functionname exp1 . . . expk})$, for some function functionname and expressions exp1 ... expk.

<table>
<thead>
<tr>
<th>Familiar Math Notation</th>
<th>DrRacket</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x) = x^2$</td>
<td>$(\text{define } f x (\times x x))$</td>
</tr>
<tr>
<td>$g(x, y) = x - y$</td>
<td>$(\text{define } g x y (- x y))$</td>
</tr>
</tbody>
</table>
• **define** is a special form; it looks like a DrRacket function, but not all of its arguments are evaluated.
• It binds a name to an expression (which uses the parameters that follow the name).

**Identifiers**

• To give names to the function and parameters, we use identifiers.
• Syntax rule: an **identifier** starts with a letter, and can include letters, numbers, hyphens, underscores, and a few other punctuation marks.
• It cannot contain spaces or any of ( ), { } [ ] ‘ “ ”.
• Syntax rule: **function definition** is of the form
  \[
  \text{(define (id1 . . . idk) exp)}, \text{ where exp is an expression and each id is an identifier.}
  \]

• **CQ 5**

**DrRacket Definitions Window**

• can accumulate definitions and expressions
• Run button loads contents into Interactions window
• can save and restore Definitions window
• provides a Stepper to let one evaluate expressions step-by-step
• features include: error highlighting, subexpression highlighting, syntax checking, bracket matching

**DrRacket Constants**

<table>
<thead>
<tr>
<th>Familiar Math Notation</th>
<th>DrRacket</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k = 3 )</td>
<td>(define k 3)</td>
</tr>
<tr>
<td>( p = k^2 )</td>
<td>(define p (* k k))</td>
</tr>
</tbody>
</table>

**DrRacket Programs** A DrRacket program is a sequence of definitions and expressions.

• The definitions are of functions and constants.
• The expressions typically involve both user-defined and built-in functions and constants.

Programs may also make use of special forms (which look like functions, but don’t necessarily evaluate all their arguments).

• **CQ 6**
3 Lecture 03

Outline

1. Administrivia
2. Identifiers and Binding
3. Semantics
4. Values and Substitution

3.1 Administrivia

Anything?

3.2 Identifiers and Binding

1. Two functions can use the same parameter name (e.g. our definitions of \( f \) and \( g \) from last lecture both used \( x \), with no conflicts).
2. A constant and a parameter can share the same name, for example

\[
\begin{align*}
&\text{(define } x \text{ 3)} \\
&(\text{define (f } x \text{) (* } x \text{ x))}
\end{align*}
\]

but this is not very readable and hence is not recommended.

3. Two constants cannot share the same name, for example

\[
\begin{align*}
&(\text{define } x \text{ 4)} \\
&(\text{define } x \text{ 5)}
\end{align*}
\]

4. A constant and a function cannot share the same name, for example

\[
\begin{align*}
&(\text{define } x \text{ 4)} \\
&(\text{define (x } y \text{) (* 5 } y\text{))}
\end{align*}
\]

5. The name of a function can be the same as a parameter name of another function, for example

\[
\begin{align*}
&(\text{define (x } y \text{) (* 5 } y\text{))} \\
&(\text{define (z } x \text{) (* 3 } x\text{))}
\end{align*}
\]

but this is not very readable and hence is not recommended.
6. The name of a function can be the same as the name of one of its parameters, for example

\[
(define (x x) (+ 1 x))
\]

but this is not very readable and hence is not recommended.

Remarks:

1. \texttt{define} binds an identifier to
   
   (a) an expression in the case of a constant.
   (b) a function declaration (arguments plus expression) in the case of a function.

### 3.3 Semantics

- \textbf{Syntax} is concerned with deciding which expressions are written correctly for DrRacket to process them.
- \textbf{Semantics} is concerned with assigning a value to each syntactically correct DrRacket expression.
- We must make sure that our DrRacket programs are unambiguous (i.e. have exactly one interpretation).
- A DrRacket program is a sequence of definitions and expressions.
- Our model involves the simplification of the program using a sequence of steps (i.e. substitution rules).
- Each step yields a valid DrRacket program.
- A fully-simplified program is a sequence of definitions which ends in a \texttt{value} (see Definition 3.4.1).

### 3.4 Values and Substitution

\textbf{Definition 3.4.1.} A \textit{value} is an expression which cannot be further simplified.

- For example, 3 is a value, but (+ 3 5) and (f 3 2) are not.
- To ensure that we all agree on how to simplify any expression, we adopt the convention that we always simplify the \texttt{leftmost} expression requiring simplification first.
- As in the slides, we use \( \Rightarrow \) to separate steps in a simplification.
Examples:

1. (define (f x) (* x x))
   (define (g x y) (- x y))
   (g (f 2) (g 3 1))
   ⇒ (g (* 2 2) (g 3 1))
   ⇒ (g 4 (g 3 1))
   ⇒ (g 4 (- 3 1))
   ⇒ (g 4 2)
   ⇒ (- 4 2)
   ⇒ 2

2. (define (term x y) (* x (sqr y)))
   (term (- 5 3) (+ 1 2))
   ⇒ (term 2 (+ 1 2))
   ⇒ (term 2 3)
   ⇒ (* 2 (sqr 3))
   ⇒ (* 2 9)
   ⇒ 18

- Exercise: Run the Stepper on these two examples yourself.
- Be prepared to demonstrate tracing on exams.
- The Stepper may not use the same rules as our convention. Write your own trace to be sure you understand your code.

- CQ 7

3.5 Goals of Module 1

Refer to Slides 52-53.

4 Lecture 04

Outline

1. Administrivia
2. The Design Recipe - Examples (M2:1-20)
3. Tests/Examples - Check-Expect/Check-Within (M2:21-26)
4.1 Administrivia

1. Your submission for a01 need not follow the design recipe.
2. In DrRacket, \#i indicates an inexact value.

4.2 The Design Recipe - Examples

Background: Recall that the roots (i.e. the values for x which make the polynomial equal 0) of the quadratic polynomial \( ax^2 + bx + c \) are given by the quadratic formula

\[
\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.
\]

1. Solve, using the design recipe:
   
   Problem: Write a DrRacket function `quadratic-root-right` which consumes real numbers \( a > 0 \), \( b \) and \( c \) and produces the the right-hand root \( \frac{-b + \sqrt{b^2 - 4ac}}{2a} \) of the quadratic polynomial \( ax^2 + bx + c \). For now you may assume that \( b^2 - 4ac \geq 0 \).

   Solution: Solve it from scratch with examples / tests developed by the class. Refer to the associated DrRacket file for the finished version.

2. Exercise: Re-do the example for `quadratic-root-left`, and the root \( \frac{-b - \sqrt{b^2 - 4ac}}{2a} \).

3. Exercise: Re-do the example with \( b^2 - 4ac < 0 \).

Examples of Functions that can return inexact values:

1. /, division
2. sqrt, square root
3. exp, exponentiation, like on a01.
4. later, sin, cos, etc, trig functions.

   - CQ 1
   - CQ 2

4.3 Tests/Examples - Check-Expect/Check-Within

1. Use check-expect if DrRacket will produce the expected value exactly.
2. Use check-within if DrRacket will produce an inexact value.

We saw both of these in the preceding example!
5 Lecture 05

Outline

1. Administrivia
2. String Functions (M2:28-31)
3. Helper Functions (M2:32-36)
4. Using Constants (M2:37-41)
5. Summary of Module 2 (M2:42-43)

5.1 Administrivia

1. In the example from last time, we can enforce $a > 0$, instead of $a \neq 0$. Then there is no ambiguity about which root is the right hand root.
2. I will post all my .rkt example files on my Instructor Specific page soon.

5.2 String Functions

Examples:

1. **Problem:** Develop a function `start-end-middle`, which will consume a string `s` and construct a new string from it, composed of

   (a) the first character,
   (b) the last character and
   (c) the middle, i.e. the substring lying strictly between the first and the last characters.

   For example, `(start-end-middle "angle") ⇒ "aengl".

   **Solution:** Solve with the class; post the DrRacket file on your Instructor Specific Page afterward.

   ● CQ 4

5.3 Helper Functions

Remarks:

1. We already used helper functions in the preceding example.
5.4 Using Constants

Remarks:

1. In CS 115, the first natural number is 0, not 1. You may have used the convention that the first natural number is 1 in previous courses.

Examples:

1. **Problem:** Develop a function `electricity-charge`, which will consume natural numbers `total-kwh` and `high-rate-kwh` and produce the total charge, assuming that high rate kWh are charged at $0.15 and low rate kWh are charged at $0.06. Assume that `total-kwh` ≥ `high-rate-kwh`. For example, `(electricity-charge 10 5)` ⇒ $1.05.

   **Solution:** Solve with the class; post the DrRacket file on your Instructor Specific Page afterward.

- CQ 5

5.5 Summary of Module 2

Refer to slides 42-43.

6 Lecture 06

Collin to type ASAP.

7 Lecture 07

Collin to type ASAP.

8 Lecture 08

Outline

1. Administrivia
2. Posn Structures (M4:1-12)
3. Dynamic Typing (M4:13-15)
4. Data Definitions / Structure Definitions (M4:16-18)
5. Design Recipe Modifications (M4:19-20)
6. Templates (M4:21-27)
7. Additions to Syntax/Semantics for Structures (M4:28-29)

8.1 Administrivia

1. Office Hours today in DC 3108, 1:00-1:50 PM.
2. I still need to type my notes for Lecture 06 and Lecture 07. I will get this done ASAP.

8.2 Posn Structures

- The building block of a structure is a field.
- Posn is built-in in DrRacket. You can use Posn with no additional setup.
- Recall that Posn is meant to capture the x and y co-ordinates of a point in the Cartesian plane (draw a diagram of an example).
- Historically, many students have thought that all structures are Posns. Do not make this mistake!
- Example: Simplify the given expression.
  
  ```scheme
  ;; requires: (posn-x myPosn) is non-zero
  (define (slope myPosn) (/ (posn-y myPosn) (posn-x myPosn)))
  (slope (make-posn (posn-y (make-posn 8 2)) (posn-x (make-posn 5 7))))
  
  Solution:
  (slope (make-posn (posn-y (make-posn 8 2)) (posn-x (make-posn 5 7))))
  ⇒ (slope (make-posn 2 (posn-x (make-posn 5 7))))
  ⇒ (slope (make-posn 2 5))
  ⇒ (/ 5 2)
  ⇒ 2.5
  
  - CQ 1
  
  - Exercise: Use slope as a helper function to write a new function to compute the slope of the line-segment from any (make-posn x0 y0) to any (make-posn x1 y1).```
Recall that Posn is a structure that stores an x and a y together. DrRacket allows any values to be stored in x and y, numeric or otherwise. If you store something non-numeric and then try to process it as if it is numeric, you will get a run-time error. Try an example for yourself in DrRacket.

- CQ 2

### 8.3 Dynamic Typing

- As in the previous clicker question, DrRacket uses **dynamic typing**. This means that DrRacket determines the type of a value **at run time**.
- DrRacket is **interpreted**, not **compiled**, so there really is no other choice.
- Contracts are comments; they cannot be enforced by DrRacket.
- However we will assume for our purposes in the course that our contracts are always observed by the caller.
- So after adopting the convention for Posn structures on slide 15, we know that the x and y values of a Posn will always be numeric.
- For Pros and Cons of dynamic typing: Refer to slide 14.

### 8.4 Data Definitions / Structure Definitions

- A **Data Definition** is a comment specifying a data type.
- For a structure it specifies the numbers and types of our fields.
- Defining a structure automatically creates:
  - Constructor
  - Selectors
  - Predicate

**Example:**

```scheme
(define-struct course (subject number title))
;; A Course is a (make-course Str Nat Str)
;; requires
;; subject is the subject area code, e.g. "CS"
;; number is the catalog number, e.g. 115
;; title is the course title,
;; e.g. "Introduction to Computer Science 1"
```
Then a template for a function consuming a Course is

;;; course-template: Course -> Any
(define (course-template mycourse)
  (... (course-subject mycourse)
       (course-number mycourse)
       (course-title mycourse)...))

8.5 Design Recipe Modifications

- Refer to the slides for full details.

8.6 Templates

- Example: Develop a function update-title which consumes oldcourse (a Course) and newtitle (a string), and produces a new Course with the same subject and number, and title set to newtitle.

  Solution:

  (define (update-title oldcourse newtitle)
    (make-course (course-subject oldcourse)
                 (course-number oldcourse)
                 newtitle))

- CQ 3

9 Lecture 09

Outline

1. Administrivia
2. Additions to Syntax/Semantics for Structures (M4:28-29)
3. More Examples (M4:30-34)
4. Using the Design Recipe for Compound Data (M4:35-41)
5. Mixed Data and Structures (M4:42-50)
6. Summary of M4 (M4:51-52)
9.1 Administrivia

1. Office Hours (for my other course) today in DC 3108, 1:00-1:50 PM.
2. I still need to type my notes for Lecture 06 and Lecture 07. I will get this done ASAP.

9.2 Additions to Syntax/Semantics for Structures

- The list of legal values now include structures!
- The simplification rules for the selectors and the type predicate are the obvious things (See slides 28-29).

9.3 More Examples

Example: Use these definitions.

```
(define-struct instructor (last-name first-name office))
;; An Instructor is a (make-instructor Str Str Str)
;; requires
;; office is the building and room number,
;; e.g. "DC 3108"
```

```
(define-struct TA (last-name first-name loginid))
;; A TA is a (make-TA Str Str Str)
;; requires
;; loginid is the email prefix,
;; e.g. "cbrown" has email address "cbrown@uwaterloo.ca"
```

```
;; A Staff is one of:
;; * an Instructor or
;; * a TA

(define-struct course (subject number title staff1 staff2))
;; A Course is a (make-course Str Nat Str Staff Staff)
;; requires
```
1. Develop a template for a consumer of an Instructor.

   Solution:

   ;; instructor-template: Instructor -> Any
   (define (instructor-template myinstructor)
     (...(instructor-last-name myinstructor)
         (instructor-first-name myinstructor)
         (instructor-office myinstructor)
2. Develop a template for a consumer of a TA.

**Solution:**

```scheme
;; TA-template: TA -> Any
(define (TA-template myTA)
    (... (TA-last-name myTA)
        (TA-first-name myTA)
        (TA-office myTA)
    ...
)
```

3. Develop a template for a consumer of a Course.

**Solution:**

```scheme
;; course-template: Course -> Any
(define (course-template mycourse)
    (... (course-subject mycourse)
        (course-number mycourse)
        (course-title mycourse)
        (cond
            [(instructor? (course-staff1 mycourse)
                (... (instructor-last-name (course-staff1 mycourse))
                    (instructor-first-name (course-staff1 mycourse))
                    (instructor-office (course-staff1 mycourse))
                ...
            ]
            [(TA? (course-staff1 mycourse)
                (... (TA-last-name (course-staff1 mycourse))
                    (TA-first-name (course-staff1 mycourse))
                    (TA-loginid (course-staff1 mycourse))
                ...
            ]
            )
        (cond
            [(instructor? (course-staff2 mycourse)
                (... (instructor-last-name (course-staff2 mycourse))
                    (instructor-first-name (course-staff2 mycourse))
                    (instructor-office (course-staff2 mycourse))
                ...
            ]
```
4. Use the template to develop a function \texttt{first-staff-contact} to return the contact information (office for an instructor; email for a TA) for the first staff member of a consumed Course, \texttt{mycourse}.

\textbf{Solution:} Do this in DrRacket. Use your prepared version if you run out of time.

\textbf{Remarks:}

(a) We could make a helper function to produce the email address from the loginid, but we won’t. This could help readability but it will not shorten the code in this example.
(b) In M5, lists will provide us a more elegant way to store several course staff members (without needing to know how many there will be up front).

\section*{9.4 Using the Design Recipe for Compound Data}

\section*{9.5 Mixed Data and Structures}

- CQ4
- CQ5
- CQ6

\section*{9.6 Summary of M4}

\section*{10 Lecture 10}

Outline

1. Administrivia
10.1 Administrivia

1. Office Hours today in DC 3108: 1:00-1:50 PM.
2. I am still behind on typing L06/07 and uploading my clicker files. I will get caught up soon.

10.2 Data Definitions for Lists

A list is either

- empty or
- (cons f r), where
  - f is a value and
  - r is a list.

Remarks:

1. This definition is recursive, i.e. it refers back to itself.
2. It might appear that this definition can continue referring back to itself forever, in the case of an infinite list. However in CS 115 all of our lists will be finite. So in any actual example, when we start peeling off the first element of some list and retaining the rest, the rest will eventually equal empty.
3. The empty list empty is a built-in constant.
4. The constructor function cons is a built-in function.
5. f is the first item in the list. r is the rest of the list.
6. first and rest are also built-in functions in DrRacket. They can be called on any non-empty list. Calling either one on an empty list results in a run-time error.

Example (from slide 11): Suppose we have
(define mylist (cons 1 (cons 'blue (cons true empty))))
Then what expression evaluates to:

1. 1?
   Solution:
   (first mylist)
2. 'blue?
   Solution:
   (first (rest mylist))
3. (cons true empty)?
   Solution:
   (rest (rest mylist))
4. true?
   Solution:
   (first (rest (rest mylist)))
5. empty?
   Solution:
   (rest (rest (rest mylist)))

- CQ 1
- CQ 2

;; A (listof Sym) is one of:
;; * empty
;; * (cons Sym (listof Sym))

Remarks:

1. Informally: a list of symbols is either empty, or it consists of a first
   symbol followed by a list of symbols (the rest of the list).
2. This is a recursive definition, with a base case, and a recursive (self-
   referential) case.
3. Lists are the main data structure in standard Racket.
4. Last week our Course structure had Staff1 and Staff2 substructures.
   This worked under the assumption that every course has exactly two
   course staff. A better approach is to use (listof Staff), so that we
   can capture any number of staff members for a given course.

10.3 Template for a List of Symbols

See M5:16
;;; los-template: (listof Sym) -> Any
(define (los-template alos)
  (cond
   [(empty? alos) ... ]
   [else (... (first alos) ... ... (los-template (rest alos)) ... )]))

Examples:

1. **Problem:** Develop a predicate list-has-symbol? which consumes a list (mylist) and returns true if mylist contains at least one symbol and false otherwise.
   
   **Solution:** See the solution in DrRacket.

2. **Problem:** Give a condensed trace to simplify (list-has-symbol? (cons 1 (cons 'blue (cons true empty))))
   
   **Solution:**

   (list-has-symbol? (cons 1 (cons 'blue (cons true empty))))
   ⇒ (or (symbol? 1) (list-has-symbol? (cons 'blue (cons true empty))))
   ⇒ (or false (list-has-symbol? (cons 'blue (cons true empty))))
   ⇒ (or (list-has-symbol? (cons 'blue (cons true empty))))
   ⇒ (or (or (symbol? 'blue) (list-has-symbol? (cons true empty))))
   ⇒ (or (or true (list-has-symbol? (cons true empty))))
   ⇒ (or true)
   ⇒ true

3. **Problem:** Give a condensed trace to simplify (list-has-symbol? (cons 1 (cons 3.5 (cons "dog" empty)))).
   
   **Solution:** Exercise. (You should get false.)

4. Develop a function last-list-element which consumes a list (mylist) and returns
   
   (a) "empty list" if mylist is empty, or
   (b) the last element of mylist if mylist is non-empty.
   
   **Solution:** Exercise. We will solve this one in Lecture 12 on Feb 13 if not sooner.
11 Lecture 11

Outline

1. Administrivia
2. Template for a List of Symbols (M5:14-22)
4. Examples (M5:33-42)
5. Wrapping a Function (M5:43)
6. String Functions (M5:44-47)
7. Example: Portions of a Total (M5:48-49)

11.1 Administrivia

1. I am caught up on uploading my iClicker files now.
2. I am not yet caught up on typing my notes from L06/07. I will get this finished over the weekend at the latest.

11.2 Template for a List of Symbols

Example from slides: my-length

;; (my-length alos) produces the number of symbols in alos.
;; my-length: (listof Sym) -> Nat
;; Examples:
(check-expect (my-length empty) 0)
(check-expect (my-length (cons 'a (cons 'b empty))) 2)

(define (my-length alos)
  (cond
   [(empty? alos) 0]
   [else (+ 1 (my-length (rest alos)))]))

Condensed trace of my-length

(my-length (cons 'a (cons 'b empty)))
⇒ (+ 1 (my-length (cons 'b empty)))
⇒ (+ 1 (+ 1 (my-length empty)))
⇒ (+ 1 (+ 1 0))
⇒ 2

• CQ 3
11.3 Structural Recursion

- **Idea:** We need recursive programs in order to handle recursively-defined data.
- **Strategy:** Create templates from our data definitions. Then use these templates as starting points to write our functions.
- **Contracts:** Use the `(listof __________)` notation to indicate the type(s) that the list elements can have. E.g.
  1. `(listof Sym)` for symbols
  2. `(listof Num)` for numerics
  3. `(listof (anyof Str Sym))` for strings or symbols

- **CQ 4**

11.4 Examples

11.5 Wrapping a Function

11.6 String Functions

**Example:** Develop a function `count-after-cat` which consumes a list of strings `(alos)` and returns the number of elements of `alos` which are alphabetically after "cat".

**Solution:**

1. Note that this example is similar to `count-apples` in the slides.
2. The string "cat" should be a named constant.
3. Thinking about this, we also realize that we should write a helper function to determine whether we should add 0 or 1 for a given list element, after comparing it to the given string ("cat" for us), which should be a parameter. This will make our code more readable, and will facilitate re-use for other comparisons. At the end of the example, we created `count-after-fish`, by simply changing "cat" to "fish" in the constants section.
4. Complete this in DrRacket. Use your prepared version if you run out of time.
11.7 Example: Portions of a Total

- Show and explain the example from slides 48-49 here.
- CQ 5

12 Lecture 12

Outline

1. Administrivia
2. Nonempty Lists (M5: 50-51)
3. Functions with Multiple Base Cases (M5: 52)
4. Structures Containing Lists (M5: 53-55)
5. Lists of Structures (M5: 56-72)

12.1 Administrivia

1. The iClicker grades on LEARN are up to date now. Please let me know about any outstanding iClicker problems.
2. Office Hours today in DC 3108, 1:00-1:50 PM.
3. I still need to type my notes for Lecture 06 and Lecture 07. I will get this done ASAP.
4. The Mid-Term Exam will cover up to the end of Module 5 (i.e. to the end of Lecture 12).
5. A reminder about how cons works: Recall that (cons f r) makes sense only if f is an element, and r is a list. For example, processing the expression (cons 1 2) will generate a run-time error, since 2 is not a list.
6. Do not use the (list a1 ... an) notation for a05.

12.2 Nonempty Lists

;; A nonempty list of numbers (Neln) is either:
;; * (cons Num empty)
;; * (cons Num Neln)

- CQ 6

Create the implementation of max-lon in DrRacket here.
12.3 Functions with Multiple Base Cases
Develop all-same? in DrRacket here.

12.4 Structures Containing Lists
Solve an example like server-tips in DrRacket here, only if there is time.

12.5 Lists of Structures
Example: Recall this structure definition from Module 4.

(define-struct line (endpt1 endpt2))
;; A Line is a (make-line Posn Posn).

Develop the function midpoints which consumes a list of Lines (alol) and returns the corresponding list of their midpoints.
Solution: Develop the solution in DrRacket.

• CQ 7

12.6 Summary of M5
Refer to slides 73-74.

13 Lecture 13

Outline
1. Administrivia
2. Intro to Module 6 (M6:1)
3. Natural Numbers (M6:2-6)
4. Countdown (M6:7-10)
5. Subintervals of the Natural Numbers (M6:11-16)
6. Countdown-to (M6:17-20)
7. Countup (M6:21-24)
8. Countup-to (M6:25-29)
13.1 Administrivia

1. Please do not pass through the instructor’s corner while the cabinet door is open. Please go around instead.

13.2 Intro to Module 6

13.3 Natural Numbers

13.4 Countdown

Problem: Develop the function countdown-by which consumes a natural number (n) and another natural number (b) and produces the multiples of n from b · n down to 0, in descending order. For example

(countdown-by 4 5) ⇒ (cons 20 (cons 16 (cons 12 (cons 8 (cons 4 (cons 0 empty))))))

Solution: Develop in DrRacket. Use your prepared version if you run out of time.

Problem: Give a condensed trace to simplify (countdown-by 4 5)

Solution:

(countdown-by 4 5)
⇒ (cons (* 4 5) (countdown-by 4 4))
⇒ (cons 20 (countdown-by 4 4))
⇒ (cons 20 (cons (* 4 4) (countdown-by 4 3)))
⇒ (cons 20 (cons 16 (countdown-by 4 3)))
⇒ (cons 20 (cons 16 (cons (* 4 3) (countdown-by 4 2))))
⇒ (cons 20 (cons 16 (cons 12 (countdown-by 4 2))))
⇒ (cons 20 (cons 16 (cons 12 (cons (* 4 2) (countdown-by 4 1)))))
⇒ (cons 20 (cons 16 (cons 12 (cons 8 (countdown-by 4 1)))))
⇒ (cons 20 (cons 16 (cons 12 (cons 8 (cons (* 4 1) (countdown-by 4 0))))))
⇒ (cons 20 (cons 16 (cons 12 (cons 8 (cons 4 (countdown-by 4 0))))))
⇒ (cons 20 (cons 16 (cons 12 (cons 6 (cons 4 (countdown-by 4 0))))))
⇒ (cons 20 (cons 16 (cons 12 (cons 6 (cons 4 (cons 0 empty))))))

• CQ 1
13.5 Subintervals of the Natural Numbers

13.6 Countdown-to

13.7 Countup

13.8 Countup-to

Problem: Develop the function \texttt{countup-by} which consumes a natural number \((n)\) and another natural number \((b)\) and produces the multiples of \(n\) from 0 up to \(c \cdot n\), in ascending order. For example
\[(\text{countdown-by} \ 4 \ 5) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ 8 \ (\text{cons} \ 12 \ (\text{cons} \ 16 \ (\text{cons} \ 20 \ \text{empty}))))))\]

Solution: Develop in DrRacket. Use your prepared version if you run out of time.

Problem: Give a condensed trace to simplify
\[(\text{countup-by} \ 4 \ 5)\]

Solution:
\[(\text{countup-by} \ 4 \ 5) \Rightarrow (\text{countup-from-by} \ 4 \ 0 \ 5) \Rightarrow (\text{cons} \ (* \ 0 \ 4) \ (\text{countup-from-by} \ 4 \ 1 \ 5)) \Rightarrow (\text{cons} \ 0 \ (\text{countup-from-by} \ 4 \ 1 \ 5)) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ (* \ 1 \ 4) \ (\text{countup-from-by} \ 4 \ 2 \ 5))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{countup-from-by} \ 4 \ 2 \ 5))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ (* \ 2 \ 4) \ (\text{countup-from-by} \ 4 \ 3 \ 5)))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ 8 \ (\text{countup-from-by} \ 4 \ 3 \ 5)))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ 8 \ (\text{cons} \ (* \ 3 \ 4) \ (\text{countup-from-by} \ 4 \ 4 \ 5)))))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ 8 \ (\text{cons} \ 12 \ (\text{countup-from-by} \ 4 \ 4 \ 5)))))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ 8 \ (\text{cons} \ 12 \ (\text{cons} \ (* \ 4 \ 4) \ (\text{countup-from-by} \ 4 \ 5 \ 5))))))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ 8 \ (\text{cons} \ 12 \ (\text{cons} \ 16 \ (\text{countup-from-by} \ 4 \ 5 \ 5))))))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ 8 \ (\text{cons} \ 12 \ (\text{cons} \ 16 \ (\text{cons} \ (* \ 5 \ 4) \ \text{empty})))))))) \Rightarrow (\text{cons} \ 0 \ (\text{cons} \ 4 \ (\text{cons} \ 8 \ (\text{cons} \ 12 \ (\text{cons} \ 16 \ (\text{cons} \ 20 \ \text{empty})))))})

\[\bullet \ CQ \ 2\]