Today

- 2 one-of templating
    - orthogonal axes of variation
    - reasoning about and simplifying code at the model level

- Have a good reading week, but
    - Don’t slow down now!
    - During next week
        - spend time practicing modules 4 through 7a
        - must be completely comfortable with all that material when we start back

- Read ALL answers, then check ALL THAT YOU FEEL APPLY before submit.

- To me, Piazza:
  A. Is a place to go when I have questions.
  B. Has too many questions to keep up with.
  C. Is something I don’t have time for.
  D. Is a commitment to read and learn and participate in the course community
  E. Is less helpful than Discord

- Being able to contribute to and get help from a discussion forum is an essential professional skill. This skill will affect your performance in your first co-op.
  - Be polite at all times. (Even when responding to a question that may violate other norms of behaviour.)
  - Don’t clutter the forum. Invest time looking for an answer before asking. Not doing so projects that you value other’s time as less important than your own.
  - Make contributions to the forum. Write up questions carefully. Include examples, include links, include all relevant details. Again, not doing so projects that you value other’s time as less important than your own.
  - Also do not ask or answer questions that violate the course academic integrity policy.

```
(define-struct terminal (label weight color))
(define-struct group (color subs))
;; Region is one of:
;;  - (make-terminal String Natural Color)
;;  - (make-group Color ListOfRegion)

;; ListOfRegion is one of:
;;  - empty
;;  - (cons Region ListOfRegion)
```

- And we know the type comments completely determine the template
  - as such they are a model of the template
  - less detailed, but they say enough to understand much about the template

```
(define (fn-for-region r)
  (cond [(single? r)
         (... (single-label r)
              (single-weight r)
              (single-color r))]
        [else
         (... (group-color r)
              (fn-for-lor (group-subs r)))]))

(define (fn-for-lor lor)
  (cond [(empty? lor) (...)]
        [else
         (... (fn-for-region (first lor))
              (fn-for-lor (rest lor)))]))
```

all-labels

```
(define-struct terminal (label weight color))
(define-struct group (color subs))
;; Region is one of:
;;  - (make-terminal   (list String) Natural Color)
;;  - (make-group Color  (NMR ListOfRegion)  )

;; ListOfRegion is one of:
;;  - empty
;;  - (cons (append (NMR Region)
;;                  (NR ListOfRegion)))
```

- Working with models
  - is one of the most important ideas in science and engineering
  - helps control complexity
    - by making it possible to reason in terms of simpler description
    - How many NR are there? Is there MR? How many?...

  - today:

    can manipulate design at model level

    to manipulate the actual resulting code

prefix=?

lst1 lst2	empty	(cons String LOS)
empty		
(cons String LOS)		

prefix=?

lst1 lst2	empty	(cons String LOS)
empty	true	true
(cons String LOS)	false	(and (string=? (first lst1) (first lst2)) (prefix=? (rest lst1) (rest lst2)))

prefix=?

lst1 lst2	empty	(cons String LOS)
empty	true [1]	true [1]
(cons String LOS)	false [2]	`(and (string=? (first lst1) (first lst2)) (prefix=? (rest lst1) (rest lst2)))` [3]

model

ListOfString is one of: <2 cases>
ListOfString is one of: <2 cases>

$$\downarrow$$

code

```
(define (fn los1 los2)
  (cond <4 cases>))
```

$$\begin{tikzcd}
\begin{array}{l}\text{model}\\ \text{ListOfString is one of: <2 cases>}\\ \text{ListOfString is one of: <2 cases>}\end{array} \arrow[r] \arrow[d, "\times" marking] & \begin{array}{l}\text{simplified model}\\ \text{cross product table}\end{array} \arrow[d] \\
\begin{array}{l}\text{code}\\ \texttt{(define (fn los1 los2)}\\ \texttt{\ \ (cond <4 cases>))}\end{array} \arrow[r, "\times" marking] & \begin{array}{l}\text{simplified code}\\ \texttt{(define (fn los1 los2)}\\ \texttt{\ \ (cond <3 cases>))}\end{array}
\end{tikzcd}$$

merge

l2 l1	empty	(cons N LON)
empty		
(cons N LON)		

merge

l2 l1	empty	(cons N LON)
empty	empty	l2

(cons N LON)	l1	`(if (< (first l1) (first l2))`
		`    (cons (first l1)`
		`          (merge (rest l1) l2))`
		`    (cons (first l2)`
		`          (merge l1 (rest l2))))`

merge

```
l2            | empty          (cons N LON)
l1            |
--------------+---------------------------------------------------------
empty         | l2    [1]      l2                                  [1]
              |
(cons N LON)  | l1    [2]      (if (< (first l1) (first l2))       [3]
              |                    (cons (first l1)
              |                          (merge (rest l1) l2))
              |                    (cons (first l2)
              |                          (merge l1 (rest l2))))
              |
```

has-path?

bt p	false	(make-node Nat Str BT BT)
empty		
(cons "L" Path)		
(cons "R" Path)		

has-path?

bt p	false	(make-node Nat Str BT BT)
empty	false [1]	true [2]
(cons "L" Path)	false [1]	(has-path? (node-l bt) (rest p)) [3]
(cons "R" Path)	false [1]	(has-path? (node-r bt) (rest p)) [4]