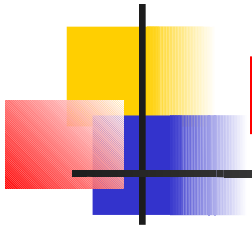
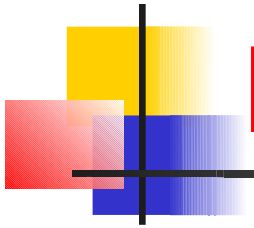
Review & Computing with First-class Functions

Corky Cartwright
Stephen Wong
Department of Computer Science
Rice University



Plan for today

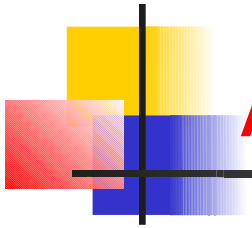
- Review of the design recipe including some in-class drill.
- More immersion in computing with functions as values.



Review: the Design Recipe

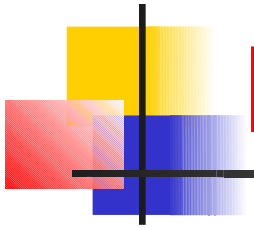
How should I go about writing programs?

- Analyze problem, which includes:
  - defining any requisite data types (and correspondig templates) that are not primitive;
  - determining what top-level (visible) functions must written.
- For each top-level function *f* to be written:
  - State contract (type signature) and purpose of *f*.
  - Give input-output examples for *f* written as tests
  - Select and instantiate a template for the function body. In most cases, the template is simple structural recursion. Other common examples include:
    - a degenerate template, e.g. trivial function, delegation to help function
    - minor variations on structural recursion
      - simultaneous structural recursion, e.g. adding two vectors represented as lists
      - extra base cases, but often better handled by a help function, e.g. max of list
    - a generative recursion template.
  - Code the function by filling in the template
  - Run the tests and confirm that they succeed.



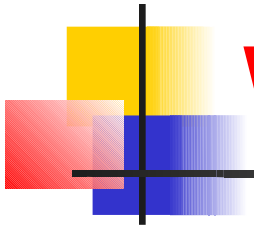
Addendum to the Design Recipe

- Writing a function may require help functions. Add these functions to the list of functions to be written.
- Use local? No. Use local only for
  - defining values (other than functions) that are used more than once,
  - to factor the definition of a complex value into a series of simpler values, or
  - to construct a non-trivial closure that closes over a free variable.
- Sometimes, we slightly deviate from standard templates
  - Extra base cases



Extra Base Cases?

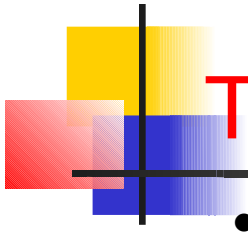
```
; fib: nat -> nat
   ; Purpose computes nth Fibonacci number (inefficiently)
   (define (fib n)
     (cond [(= n 0) 1)]
           [(= n 1) 1)]
           [else (+ (fib (- n 1)) (fib (- n 2)))]))
; max-list: list-of-numbers -> number
   ; Purpose: (max-list lon) finds the maximum element in
   ;    lon; throws an error on the empty list
(define (max-list lon)
     (cond [(empty? lon) (error 'max-list "applied to empty")]
           [else (ml-help (first lon) (rest lon))])))
(define (ml-help ans)
     (cond [(empty? lon) ans]
           [(< ans (first lon)) (ml-help (first lon) (rest lon))]
           [else (ml-help ans (rest lon))])))]
```



What goes in a template?

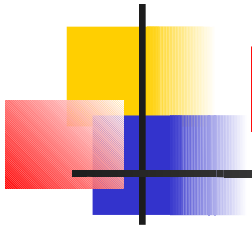
- Division into cases corresponding to an inductive definition of the data (which may be simply a union).
- Identification of recursive sub-problems (form of recursive calls)
- No calls on auxiliary functions or predicates other than those required for case analysis, such as:
  - testing that input has form assumed in contract
  - including logic from the "glue" code (what is inserted in the ellipsis of a properly written template)

and those that appear in explicit form of recursive calls in generative recursion.



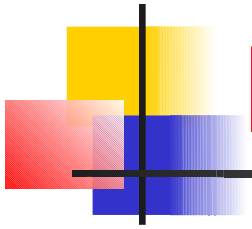
Template vs. Template Instantiation

- Template is part of a **data definition**
  - function name is generic
  - extra arguments to function are unspecified
- Template Instantiation is a prelude to writing a specific function. After you select the appropriate template, you tailor it to the function you are writing:
  - function name is specific
  - extra argument are specified in header and in recursive calls if possible
- Nothing else appears in a template instantiation.



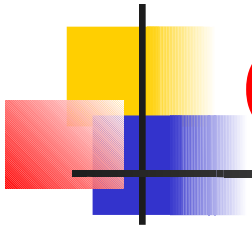
More review materials

- Homework problems
- Look at past first and second mid-terms from Comp 210, ignoring last 5 pages of 2nd exam which cover
  - Parsing
  - Graph traversal
  - Software engineering trade-off questions



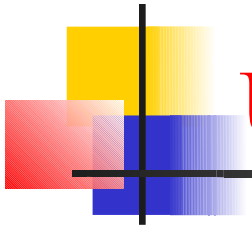
Exam Description

- Take home. Closed book. Closed computer.
- Don't worry about Scheme library functions. You will be given all of the operations you can use in coding.
- Three and 1/2 hours with optional 15 minute break in middle.



Class exercise

- Write insert-everywhere/in-all-words (problem 12.4.2 from HTDP)
- See link to 12.4.2.sol.ss on wiki



Using Functions to Represent Objects

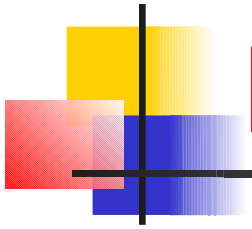
- How can we represent a pair so that the only operations that code can perform on pairs are:

```
(create-pair x y)
(pair-first p)
(pair-second p)
(pair-equal? p1 p2)
```

- What if we represent a pair as a list? As a struct? Structs are not as robust as you might think. In the advanced language level try:

```
(define-struct Pair (first second))
(define p (make-Pair 1 2))
(set-Pair-first! P 17)
p
```

-



For Next Class

- New Homework due Monday
- Reading: review for the exam.
- Exam distributed on Monday
- Due Wednesday after break.