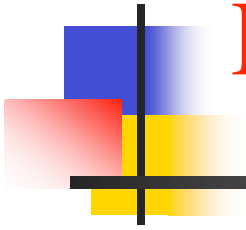


Design Abstractions with First-Class Functions





Plan for today

- Review of functions as values
- Designing with functions
- “The Trick”

- User interfaces
- Model-View-Controller recipe
- Tips for GUI programming in Dr. Scheme



Functions as Return Values

```
(define (f x) first)
(define (g x) f)
(define (h x)
  (cond
    ((empty? x) f)
    ((cons? x) g)))
```



Reduction

```
(define (add x)
  (local ((define (x-adder y) (+ x y)))
    x-adder))
```

```
(define f (add 5))
= (define f (local ((define (x-adder y) (+ 5 y)))
  x-adder))
= (define f (local ((define (x-adder5 y) (+ 5 y)))
  x-adder5))
= (define (x-adder5 y) (+ 5 y))
  (define f x-adder5)
```



(In class discussion)

- Renaming happens when we apply add
 - So it can happen many times
- **Not** when we define it
 - Which makes sense, because we only define it once, and one renaming is not enough



Reduction

```
(f 10)
= (x-adder5 10)
= (+ 5 10)
= 15
```

- Which parts of this reduction sequence are “business as usual”?
- Which are completely new?



Designing with Function Values

- Functions-as-values work really well with local definitions
- Basic idea of designing for functions: You can replace essentially any function of type
 - $A \rightarrow B \rightarrow C$with a function of (often easier-to-use) type
 - $A \rightarrow (B \rightarrow C)$



In Class Examples

$\text{map} : A \rightarrow B \rightarrow C$

$\text{map} : (X \rightarrow Y) [X] \rightarrow [Y]$

$\text{map}' : A \rightarrow (B \rightarrow C)$

$\text{map}' : (X \rightarrow Y) \rightarrow ([X] \rightarrow [Y])$



Standard Map

```
(define (map f l)
  (cond
    [(empty? l) empty]
    [else
     (cons (f (first l))
           (map f (rest l)))]))
```



Modified Map

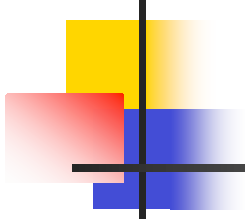
```
(define (map' f)
  (local
    ((define (map-f l)
      (cond
        [(empty? l) empty]
        [else
         (cons (f (first l))
                (map-f (rest l)))])))
    map-f))
```



“The Trick” (in class exercise)

- This is not covered in the book. But we can do this “automatically”
- Any ideas for how to write a function

magic: $(A \ B \ \rightarrow \ C) \ \rightarrow \ (A \ \rightarrow \ (B \ \rightarrow \ C))$



```
(define (magic f)
  (local ((define (g x)
            (local ((define (h y)
                      (f x y)))
              h))
          g))
```



Challenge: See if you can write

`unmagic : (A -> (B -> C)) -> (A B -> C)`

Note the equational properties:

`magic (unmagic f) = f`

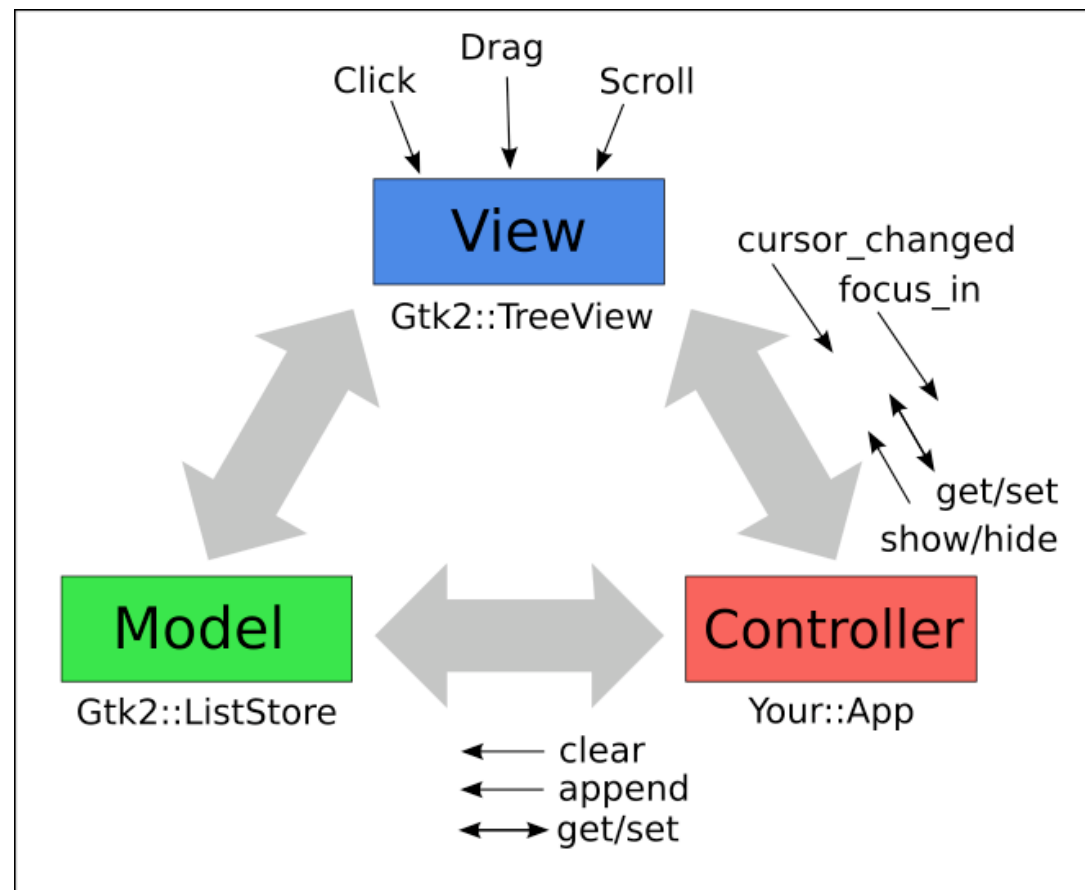
`unmagic (magic f) = f`



User Interfaces

- The framework for machine/user interaction
- Important research area
 - The ACM HCI conference
 - Larger scope: Human factors
- Highly interdisciplinary
 - Cognitive science
 - Psychology

Model-View-Controller (GUIs)





Tip for Managing Windows

As documented in the online help, the contract for `hide-window` is:

```
window -> true
```

So, to close the window, you need to capture it ahead of time as follows:

```
(define window
  (local ((define (handler ignored)
              (hide-window window)))
    (create-window
      (list (list (make-button "Close" handler))))))
```



For Next Class

- Homework due Monday
- Reading:
 - Ch 23: Mathematical examples using functions as values
- Quiz on reading