Design Strategies 3: Divide into cases

CS 5010 Program Design Paradigms
Lesson 2.2
Lesson 2.1

Data Representations
- Basics
- Mixed Data
  - Recursive Data
  - Functional Data
    - Objects & Classes
    - Stateful Objects

Design Strategies
- Combine simpler functions
- Use a template
- Divide into Cases
  - Call a more general function
  - Communicate via State

Generalization
- Over Constants
- Over Expressions
- Over Contexts
  - Over Data Representations
  - Over Method Implementations
Divide into cases on <condition>

• Sometimes you need to break up an argument in some way other than by its template.
• We already saw this in Lesson 0.4 in the definition of **abs**:

```
; abs : Real -> Real
; RETURNS: the absolute value of the given real number.
; STRATEGY: divide into cases on sign of x
(define (abs x)
  (if (< x 0)
    (- 0 x)
    x))
```
Example: income tax

• Imagine we are computing income tax in a system where there are three rates:
  – One on incomes less than $10,000
  – One on incomes between $10,000 and $20,000
  – One on incomes of $20,000 and over

• The natural thing to do is to partition the income into three cases, corresponding to these three income ranges.
Write a **cond** or **if** that divides the data into the desired cases

```scheme
;; STRATEGY: Cases on amt
;; f : NonNegReal -> ??
(define (f amt)
  (cond
   [(and (<= 0 amt) (<amt 10000)) ...
    [(and (<= 10000 amt) (<amt 20000)) ...
     [(<= 20000 amt) ...]])
```


Write a **cond** or **if** that divides the data into the desired cases

```scheme
;; tax-on : NonNegReal -> NonNegReal
;; GIVEN: A person’s income
;; RETURNS: the tax on the income
;; EXAMPLES: ....
;; STRATEGY: Cases on amt
(define (tax-on amt)
  (cond
    [(and (<= 0 amt) (< amt 10000)) ...]
    [(and (<= 10000 amt) (< amt 20000)) ...]
    [(<= 20000 amt) ...])))
```

The predicates must be exhaustive. Make them mutually exclusive when you can.

This is contract is sloppy. Currency amounts should never be **Real**. They should always be integers, and units should be specified. But we don't need to be so careful for this made-up example.
Now fill in the blanks

;; tax-on : NonNegReal -> NonNegReal
;; GIVEN: A person’s income
;; RETURNS: the tax on the income
;; EXAMPLES: ....
;; STRATEGY: Cases on amt

(define (tax-on amt)
  (cond
   [(and (<= 0 amt) (< amt 10000))
     0]
   [(and (<= 10000 amt) (< amt 20000))
     (* 0.10 (- amt 10000))]
   [(<= 20000 amt)
     (+ 1000 (* 0.20 (- amt 20000)))]))
Another example

;; ball-after-tick : Ball -> Ball
;; GIVEN: The state of a ball b
;; RETURNS: the state of given ball at the next tick
;; STRATEGY: cases on whether ball would hit the wall on
;; the next tick

(define (ball-after-tick b)
  (if (ball-would-hit-wall? b)
      (ball-after-bounce b)
      (ball-after-straight-travel b)))
Where does cases fit in our menu of design strategies?

- If you are inspecting a piece of enumeration or mixed data, you almost always want to use the template for that data type.
- Cases is just for when dividing up the data by the template doesn't work.
Next Steps

• If you have questions or comments about this lesson, post them on the discussion board.
• Go on to the next lesson