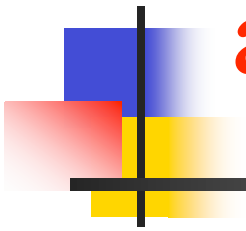


Auxiliary functions, booleans, and conditionals





Last Lecture

- Programs as
 - Numbers
 - Variables
 - Function definitions
 - Function applications
- Design recipe and homework format
 - Touching on how to deal with bigger problems
- Syntax and runtime errors



Today's Goals

- Decomposing bigger problems
 - Discovering and creating auxiliary function definitions
- More basic types: Booleans
 - The values: **true, false**
 - The operations: **and, or, not**
- Comparing numbers
- And even more types: Symbols
- Conditional statements
 - **cond**



Decomposing Bigger Problems

- Real-world problem descriptions are fuzzy
- Nevertheless, to actually solve real problems, we must look for clear, well-defined structure in these problems
- Problem descriptions often contain dependencies
 - “...distance is defined as...”
 - “...the weekly pay for the editor is determined by...”
- A dependency is often solved with a function
- Decompose problem dependencies into smaller ones
 - Often, producing many auxiliary functions
 - And one main function



Why Auxiliary Functions?

Consider the program to the right

- What does it do?

```
(define (profit price)
  (- (* (+ 120
         (* (/ 15 .10)
              (- 5.00
                 price)))
      price)
     (+ 180
        (* .04
           (+ 120
              (* (/ 15 .10)
                  (- 5.00
                     price))
              )))))
```



Why Auxiliary Functions?

```
(define (profit ticket-price)
  (- (revenue ticket-price)
     (cost ticket-price)))
(define (revenue ticket-price)
  (* (attendees ticket-price)
     ticket-price))
(define (cost ticket-price)
  (+ 180
     (* .04
        (attendees ticket-price))))
(define (attendees ticket-price)
  (+ 120 (* (/ 15 .10)
            (- 5.00 ticket-price))))
```

Now consider this program

- What does it do?



Why Auxiliary Functions?

```
;; Good program design
(define (profit ticket-price)
  (- (revenue ticket-price)
     (cost ticket-price)))
(define (revenue ticket-price)
  (* (attendees ticket-price)
     ticket-price))
(define (cost ticket-price)
  (+ 180
     (* .04
        (attendees ticket-price))))
(define (attendees ticket-price)
  (+ 120 (* (/ 15 .10)
            (- 5.00 ticket-price))))
```

```
;; Bad program design
(define (profit price)
  (- (* (+ 120
         (* (/ 15 .10)
            (- 5.00
              price))))
     (+ 180
        (* .04
           (+ 120
              (* (/ 15 .10)
                 (- 5.00
                   price)))))))
```



Aside 1

```
;; Good program design
(define (profit ticket-price)
  (- (revenue ticket-price)
     (cost ticket-price)))
(define (revenue ticket-price)
  (* (attendees ticket-price)
     ticket-price))
(define (cost ticket-price)
  (+ 180
     (* .04
        (attendees ticket-price))))
(define (attendees ticket-price)
  (+ 120 (* (/ 15 .10)
            (- 5.00 ticket-price))))
```

Clearly, good function names help.

Similarly, things like 180, .04, 120, 15, .10, and 5.00 make this program a bit harder to understand than it needs to be.

Such values, that occur in programs seemingly out of nowhere, are called magic numbers.

This program can be improved by using these values to define variables, and giving them helpful names.



Aside 2

```
;; Good program design
(define (profit ticket-price)
  (- (revenue ticket-price)
     (cost ticket-price)))
(define (revenue ticket-price)
  (* (attendees ticket-price)
     ticket-price))
(define (cost ticket-price)
  (+ 180
     (* .04
        (attendees ticket-price))))
(define (attendees ticket-price)
  (+ 120 (* (/ 15 .10)
            (- 5.00 ticket-price))))
```

We apply the design recipe to each auxiliary function.

This means we develop contracts, purpose statements, examples, and we do the testing for each of these smaller functions.

The smaller we can make the functions, the more effective the recipe is at producing correct programs.

Tip: Remember this for HWK!



Decomposing After the Fact

Compare the following two programs:

- ```
(define (area-of-ring outer inner)
 (- (* 3.14 (* outer outer)
 (* 3.14 (* inner inner))))
```
- ```
(define (area-of-ring outer inner)
  (- (area-of-disk outer)
     (area-of-disk inner)))
```
- Conciseness helps readability
- Choosing good names programs shorter, clearer, and easier to change.



Booleans

- Named after mathematician George Bool
- Like numbers, Booleans are a type
 - Two atomic values: **true**, **false**
 - Alternative view of “0s and 1s”
- A smaller number of basic operations
 - **and**, **or**, **not**
 - Important: Not to be confused with everyday language
- All other functions can be defined in terms of these ones
 - Other functions that people know?
- Basis for digital logic. (Logic in COMP, HW in ELEC)



Booleans

The operations seem intuitive, but are they?

- `(and true true) = true`
- `(and true false) = false`
- `(and false true) = false`
- `(and false false) = false`

True if and only if both arguments are true



Booleans

The operations seem intuitive, but are they?

- `(or true true)` = `true`
- `(or true false)` = `true`
- `(or false true)` = `true`
- `(or false false)` = `false`

False if and only if both arguments are false



Booleans

The operations seem intuitive, but are they?

- **(not true)** = false
- **(not false)** = true

False if and only if the argument is true

Follow these rules very carefully when evaluating
Boolean expressions



Comparing Numbers

- Booleans give us values to represent decisions
- Comparisons usually have a contract like this
 - number number \rightarrow boolean
- Examples:
 - `(= 1 1) = true`
 - `(= 1 (- 4 2)) = (= 1 2) = false`
 - `(< 1 5) = true`
 - `(<= 5 3) = false`
- Don't forget: The red `=` is reduction.



Symbols

- Like numbers and Booleans, symbols are a type
- Very useful for referring to external concepts
- Can be any sequence of characters
 - As long as they start with an apostrophe `'`
 - Don't confuse apostrophe with backtick ```
 - Examples `'rabbit`, `'Ringo`, `'submarine`
 - Symbols are a lot like constants
- There is one main thing we want to do with symbols
 - **`symbol=?`**
 - Example: `(symbol=? 'rabbit 'bunny) = false`
- Don't confuse symbols with variables:
 - Example: `(define rabbit 'rabbit)`



The Conditional Statement

- What do we do if a problem requires doing different things in different cases?
 - “...if the shorthand is 1, this means it is Monday...”

```
(define (day n)
  (cond [ (= n 1) 'Monday ]
        [ (= n 2) 'Tuesday ]
        ...
        [ (= n 7) 'Sunday ]
        [ else 'NotADay ] ) )
```



The Conditional Statement

- Reduction requires a lot of care:

```
(day 2)
= (cond [(= 2 1) 'Monday]
        [(= 2 2) 'Tuesday] ...)
= (cond [false 'Monday]
        [(= 2 2) 'Tuesday] ...)
= (cond [false 'Monday]
        [true 'Tuesday] ...)
= 'Tuesday
```



For Next Class

- Homework due Wednesday before class
- Reading:
 - Chapters 4, 5, 6 are the reading assignment
 - Pointers to companion have also been posted
- Quiz:
 - Due Tue at 8:00 am --- chapters 1, 2, 3
 - Due Wed at 8:00 am -- chapters 4, 5, 6