Lecture 4:
"Difference Between Dates"
and
data abstraction
<table>
<thead>
<tr>
<th>Week</th>
<th>Dates</th>
<th>Lecture:</th>
<th>Lab:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Jan 29-Feb 3</td>
<td>Conditionals, Case Studies</td>
<td>Explore &quot;Difference between Dates&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reading:  &quot;DbD&quot; case study</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Feb 5-9</td>
<td>Data abstraction in DbD: extending to short dates</td>
<td>More Difference between dates</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Miniproject 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Feb 12-16</td>
<td>Introduction to Recursion</td>
<td>Recursion</td>
</tr>
<tr>
<td>6</td>
<td>Feb 19-23</td>
<td>&lt;holiday&gt;</td>
<td>Recursion II</td>
</tr>
<tr>
<td>7</td>
<td>Feb 26 - Mar 2</td>
<td>Midterm 1</td>
<td>Advanced recursion</td>
</tr>
</tbody>
</table>
How useful has the case study been?
Any questions about last week's materials?

- (SchemeHandler...)
This week

• A few exercises on Tue/Wed

• Mini-project #1: You are to write `century-day-span`
  - Extend the `day-span` program to correctly handle dates in (possibly) different years.
  - Consider the central lesson of the case study: there are easier and harder ways to solve problems. Choose easier.
This is your first large program

Use helper functions

- Break out self-contained tasks into helper procedures: they should be easy to name.
- If you can get your main procedure to read like English, you are doing well.

• Test, and test some more.
  - Remember to put test cases above each helper procedure.

• Reuse code that you have already written

• Add comments!
  - Above each procedure, at least.
  - Within some `cond` cases, additionally.
“the process of leaving out consideration of one or more properties of a complex object or process so as to attend to others”
• Abstracting with a new function

(square x) instead of (* x x)
(third sent) instead of (first (bf (bf sent)))

• Abstracting a new datatype

A datatype provides functionality necessary to store "something" important to the program

- Selectors: to look at parts of the "something".
- Constructor: to create a new "something".
- Tests (sometimes): to see whether you have a "something", or a "something else"
Data abstraction: words and sentences

**Constructors**: procedures to make a piece of data
- word, sentence

**Selectors**: procedures to return parts of that data piece
- first, butfirst, etc.

**Tests**: predicates that tell you which type of data you have
- word?, sentence?
Write `card-greater?` The procedure takes two cards and returns true if and only if the first card is bigger than second.

Cards are represented by a two-character word, where the first character represents the rank (`a`, `k`, `q`, `j`, `0`, `9`, `8`, `7`, `6`, `5`, `4`, `3`, and `2`), and the second character represents the suit (`s`, `h`, `d`, and `c`). For instance, `2h` is the two of hearts, `qc` is queen of clubs, `0s` is the 10 of spades, etc. For this problem, consider all spades to rank higher than hearts, which all rank higher than diamonds, which all rank higher than clubs.

```
(card-greater? 'ac '3d) → #f
(card-greater? 'kh 'qh) → #t
(card-greater? '4s '4s) → #f
```

Comment all your procedures. Assume you have a working version of `outranks?`, as you wrote in lab, to use. (Remember, `outranks?` takes two ranks and returns true if the first is higher than the second.)
Benefits

• Why is "leaving out consideration of", or "not knowing about", a portion of the program a good thing?

• Consider two ways one can "understand a program":
  - Knowing what each function does
  - Knowing what the inputs are (can be), and what the outputs are (will be).
Data abstraction in the DbD code

• How does the code separate out processing of the date-format from the logic that does the "real" work?

  - **Selectors**
    - month-name (takes a date)
    - date-in-month (takes a date)
    - ? month-number (takes a month name)

  - **Constructors? Tests?**