Outline

- Errata
- Homework #3
- Sequences
- Strings
- Output
  - format
- Input
  - reader
- Files
- Examples
  - pattern matching
  - information retrieval
Homework #3

- make-iterator
- mapiter
- mapiter-t
- pattern matching
- mapiteri
Basics

- Symbols
  - arbitrary tokens
  - symbol table

- Characters
  - single letter
  - \#

- Strings
  - sequence of characters
Sequence

- Not a list
- Includes
  - lists
  - strings
  - simple vectors
Many Sequence Methods

- length, subseq, position, find, concatenate, elt, map, etc. etc.
- elt
  - generalized nth
- map
  - generalized mapcar
- concatenate
  - generalized append
Invoking Sequence Methods

- "String append"
  
  (concatenate 'string "abc" "def")

- 2nd argument
  
  - what kind of sequence it is
String equality

- `string=`
  - case sensitive
- `string-equal`
  - case insensitive
- `equal`
  - not string-specific
  - case sensitive
- `equalp`
  - not string-specific
  - case insensitive

In general
- best to use the most specific applicable test
I/O

- Stream-based I/O
  - Output
    - create a stream
    - write stuff to it
  - Input
    - create a stream
    - read stuff from it
Output Streams

- *standard-output*
  - many functions take T
- a string
  - some functions take nil
  - also (make-string-output-stream)
- a file
  - need a file stream
Output Methods

- **Many**
  - `print`
  - `prin1`
  - `princ`
  - `pprint`
  - `write`

- **print**
  - use when you want Lisp to read in the result

- **pprint**
  - when displaying data structures to the user

- **format**
  - for almost everything else
Format

- Lisp's answer to printf
  \((\text{format stream directive args})\)

- Many directives
  - \(S = \text{symbol}\)
  - \(A = \text{ascii}\)
  - \(D = \text{integer}\)
  - \(F = \text{floating point}\)

- Many options
Example

- Print a hand of cards
Input Streams

- *standard-input*
- string
  - make-string-input-stream
- file
  - need a file stream
Read

- read
  - extremely powerful
  - Lisp interprets the input

- Examples
  - lists
  - chars
  - strings
  - floats, etc
Other Read Functions

- read-line
  - gets strings from text files and terminal
- read-char
  - gets a single character
- peek-char
  - examine the next char without reading
Example

- Eliza
  - format
  - strings
  - read
  - more pattern matching
  - simple rule-based system
Files

- To convert a file to stream
  - open
- Both input and output
- Error handling
Examples

(open "foo.dat" :direction :input)
(open "foo.dat" :direction :output)
(open "foo.dat" :direction :input :if-does-not-exist nil)
(open "foo.dat" :direction :output :if-exists :append)
With-Open-File
(with-open-file (in "foo.dat" :direction :input) ...)

- Advantages
  - file-specific let
  - stream automatically closed
  - even with thrown exception
Paths

- Paths are extremely flexible
  - not OS specific
- Parts
  - host
  - device
  - directory
  - name
  - type
  - version
  - defaults
Path functions

- directory
- probe-file
- also
  - copy, delete, etc.
- In most cases
  - a string can replace a path object
Structures

- pathname is actually a structure
  - named collection of data
- similar to struct in C
Example

- gin game
  - hand1
  - hand2
  - deck
  - discards
List?

- Could just create a list
- Synonyms for car, cadr, etc.
- defstruct does it all for you
defstruct

- defines
  - accessor
  - mutators
  - constructors
  - read macro
Example

- polynomial term
Symbolic Computing Example

- Information retrieval
Homework #4

- polynomial->string and back
- save / load IR index
- modifications to Eliza
Symbolic Programming Example

- Cards