Symbols

- We've seen symbols in three contexts so far:
  - `(setf sym 3)`
    3
  - `(let ((sym 3)) ...)`
    > Sym
    SYM

Symbols – Context

- The first of these refers to a special (or global) variable.
- The second refers to a lexical (or local) variable.
- The third refers to a global symbol.
- But these are all uses of symbols!

Special Variables

- Also called dynamic variables
- Created by `setf`, `defvar`, and others
- Scope is from when bound to a value until whenever
- Global...
- Should rarely be used.

Lexical, Perplexical

- Lexical or static variables are created by functions like `let`, `do`, etc.
- Can only be referenced within textual region defined
- However, bindings can be changed at any time...
  Remember closure.
A symbol to hold a lexical variable is never really created — the value is merely substituted as needed. Symbols, however, are big structures that include pointers to values and functions. Special variables are symbols whose value pointers point to the value of the variable. Named functions are symbol pointers as well.

Symbols are represented internally a little like this:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;RIGHT-TRIANGLE-P&quot;</td>
<td>#&lt;right-triangle-p&gt;</td>
<td>...</td>
</tr>
</tbody>
</table>

Remember Lisp converts symbols to all caps. Use `(symbol-name symbol)` to get a string holding the name of the symbol. Symbol names can have whitespace — use `| |` to enclose the symbol when declaring it: `| This is a really long symbol name |`

Symbols are represented internally a little like this:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;RIGHT-TRIANGLE-P&quot;</td>
<td>#&lt;right-triangle-p&gt;</td>
<td>...</td>
</tr>
</tbody>
</table>

You can create symbols with `(intern)`. Reverses process of `(symbol-name)`. Returns symbol and whether the symbol previously existed:

- `nil` — Didn’t exist
- `:internal` — Already present in this package
- `:external` — Imported from another package
- `:inherited` — Imported via use-package

Kenneth W. Flynn (7-12) 09/23/98
Intern Examples (Not what you're thinking!)

> (intern "KENN")
KENN
NIL

> (intern "KENN" 'common-lisp)
CL::KENN
NIL

> (intern "CAR" 'common-lisp)
CAR
:EXTERNAL

Packages II

Create packages with (defpackage)
(defpackage "PACKAGE-NAME"
  (:use "COMMON-LISP" ...)
  (:nicknames "PN")
  (:export "SYM1" "SYM2" ...)
)

(in-package 'PACKAGE-NAME)

Packages...

Packages allow for source code management
To use other packages, we can refer to exported symbols as PACKAGE-NAME:SYMBOL, or PN:SYMBOL
Using a package allows us to not have to have the qualifier
Most implementations auto use 'Common-Lisp

Symbols Again

Symbols know their package name and are contained within

<table>
<thead>
<tr>
<th>Name</th>
<th>&quot;CAR&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>#$&lt;car&gt;</td>
</tr>
<tr>
<td>Function</td>
<td>COMMON-LISP</td>
</tr>
<tr>
<td>Package</td>
<td></td>
</tr>
</tbody>
</table>

Keywords

Keyword arguments, or symbols beginning with a mere : – such as :input are in the KEYWORD package
They are put there silently. They are then accessible anywhere – :symbol means look in that package, which is auto "used."
Functions that take symbols as args should use keywords

Numbers

Lisp has many functions to handle numbers
But first, we need to talk about types:
Although we never see it, all Lisp variables have a data type.
These types are not mutually exclusive!
See Steele, pg. 50 for complete list
### Data Types
- Numbers
- Characters
- Symbols
- Lists
- Arrays
- Packages
- Streams
- Structures
- Functions
- Hash-Tables
- ReadTables
- Random-States

### Number Types
- t
- number
- complex
- real
- float
- rational
- ratio
- integer
- nil

### Typing
- (typep 123 'integer)
  T
- (typep 3.3 'float)
  T
- Can be used with any type, even a structure you define
- Also (numberp, integerp, etc...)

### More on Numbers
- Conversion
  - (float) (truncate) (floor) (ceiling) (round) ...
  - Take any number of args
- Comparison
  - = does numeric checking
  - #= requires same type and numeric equal

### Some Notes On Style
- Lisp is functional
- All functions return at least one value
- Functions do not modify their arguments, instead they return a new value
- This is what we mean by "no variables"
- No side-effects
- Use as few setf's as possible! This will avoid errors

### House of Lisp Style
- In source files, should only find package definitions, global constants, and functions
- Do not declare global variables unless you have a very, very, very good reason!
- Then use (load) to load your file

Kenneth W. Flynn (19-24)
09/23/98
That's It

- Covered Chapters #8, 9 in Graham
- For next week:
  - Project #2 spec due soon
  - Take a break
- Next Week:
  - Macros
  - Exception Handling