Introduction to Emacs Lisp

What will we cover?

This is meant to be a practical series!

- Emacs Lisp fundamentals
- Everything about functions and variables
- Manipulating Emacs
- Working with the system
- Creating and using extensibility points
- Writing macros
- Publishing to MELPA

We will work on at least one new package as an example!

Who is this series for?

- Not only for programmers!
- Everyone should get to enjoy hacking Emacs

If you know Lisp well, don't worry if I skip concepts!

What is Lisp?

- That weird language with Lots of Irritating Superfluous Parentheses
- A language and environment based on the idea of interactivity
- The syntax enables new language constructs to be defined
- The ultimate hacker language!

Emacs Lisp

- A dialect of Lisp for configuring and extending Emacs
- Much of Emacs' built-in functionality is written with it!
- Has core features geared toward discoverability and extensibility
- APIs and data types for many types special to Emacs (buffers, windows, etc)

We will focus on core concepts in this video!

Lisp Syntax

The beauty of Lisp comes from the simplicity of its syntax!

Lisp syntax is primarily composed of lists, symbols, and values.

```
(defun the-meaning-of-life (answer)
  (message "The answer is %s" answer))

;; Newlines and whitespace can be added anywhere in lists
(list 1 2 3
      4 5 6
      7 8 9)
```

The code can also be treated as data!

You shouldn't be afraid of editing code with parentheses, there are packages for that :)

Values (or "Objects")

Any value, or object, has a type. It also has a textual representation that may or may not be "readable"!

Lisp Types

- Strings
- Numbers (both integer and float)
- Symbols
- Cons cells
- Arrays and Vectors
- ... and more

https://www.gnu.org/software/emacs/manual/html_node/elisp/Programming-Types.html#Programming-Types

Emacs Types

There are many types specific to Emacs as well, most don't have a code representation:

- Buffers
- Windows
- Frames
- Threads
- Keymaps
- ...

Things we do with these types can affect the Emacs interface:

```
;; Get the previous buffer and switch to it
(switch-to-buffer (other-buffer))
```

https://www.gnu.org/software/emacs/manual/html_node/elisp/Editing-Types.html#Editing-Types

Forms and Evaluation

A "form" is any lisp object that can be evaluated.

How evaluation works

Evaluation works differently for:

- Lists
- Symbols
- All other object types

Some are self-evaluating, meaning that they return their own value:

```
;; Primitives are usually self-evaluating
42

"Hello!"

[1 2 (+ 1 2)]

;; Not self-evaluating!
buffer-file-name

;; Evaluates a function!
(+ 300 11)

(300 100)

;; Some representations can't be evaluated!
#<buffer Emacs-Lisp-01.org>
```

The “Environment”

Everything is evaluated in terms of Emacs Lisp’s global environment!

Pros: you can change anything in the environment as you go Cons: your environment might get “dirty” over time in your Emacs session

```
;; Set the initial value
(setq efs/our-nice-variable "Hello System Crafters!")

;; Change it to something else (even a different type!)
(setq efs/our-nice-variable 1337)
```

Expressions

Lisp is an expression-based language, almost all forms return a value.

```
;; A very useful function...
(defun add-42 (num)
  (+ num 42))

;; It returns the result
(add-42 58)

;; Using the result in another call
(* (add-42 58) 100)
```

Symbols

A symbol is also a type of object, but it's not self-evaluating!

Symbols can contain alphanumeric characters plus many others:

```
# Possible symbol characters
- + = * / _ ~ ! @ $ % ^ & : < > { } ?
```

This gives you the ability to ascribe meanings to symbols based on the characters they contain. Some examples:

- `bui-keyword->symbol` - Convert from one type to another
- `efs/some-name` - Define a "namespace" for the symbol
- `*pcache-repositories*` - Indicates a global variable (not common in Emacs Lisp)
- `string=` - Check if something is equal to something else

When a symbol is evaluated, it returns the variable value associated with that binding:

```
;; The example we saw before
buffer-file-name
```

Function names can't be evaluated like this though:

```
get-file-buffer
```

We will discuss this point more in a future episode.

https://www.gnu.org/software/emacs/manual/html_node/elisp/Symbol-Type.html#Symbol-Type

Infix vs Prefix

Lisp expressions use “prefix” notation:

```
(+ 300 (- 12 1))
```

Why is this useful? Because it puts all functions and operators at the same level of importance, even the ones you define!

Exercise

Open up the `*scratch*` buffer and experiment with writing simple expressions. Use `C-x C-e` (`eval-last-sexp`) at the end of each expression to evaluate them.

Here are some you can try:

```
42

(* 42 10)

(concat "Hello " "Emacs!")

;; Simple list
'(1 2 3)

;; Another way to create a list
(list 1 2 3)

;; Get the second list item
(car (cdr '(1 2 3)))

;; A vector
[1 2 3]
```

Also, go take a look at your Emacs configuration and see what things you can recognize about it now!