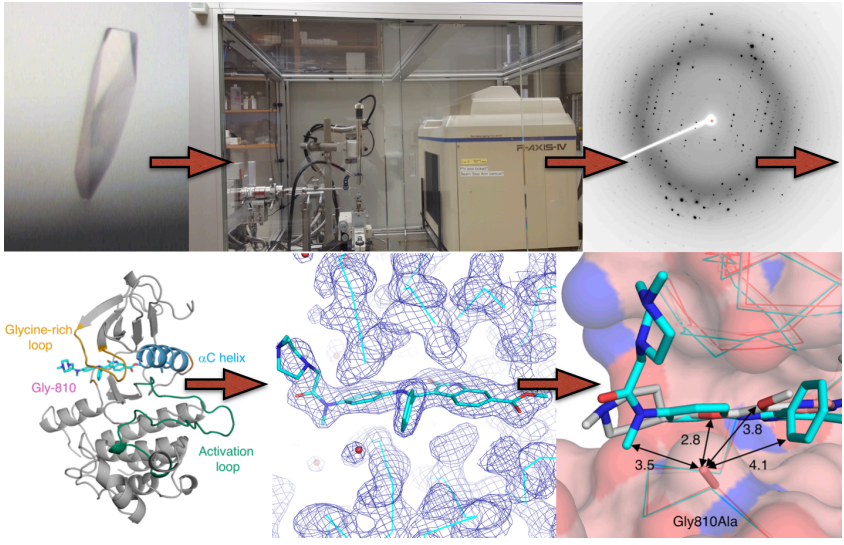
Functional elisp from the beginning

Blaine Mooers, PhD
blaine-mooers@ouhsc.edu
405-271-8300

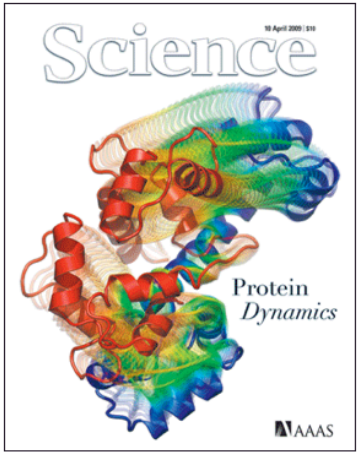
Department of Biochemistry & Molecular Biology
University of Oklahoma Health Sciences Center, Oklahoma City

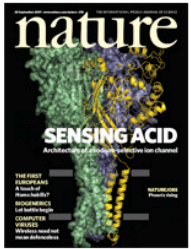
Austin Emacs Meetup
Austin, Texas
Virtual Meeting
4 May 2022

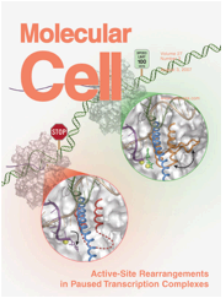
Workflow in the Mooers Lab



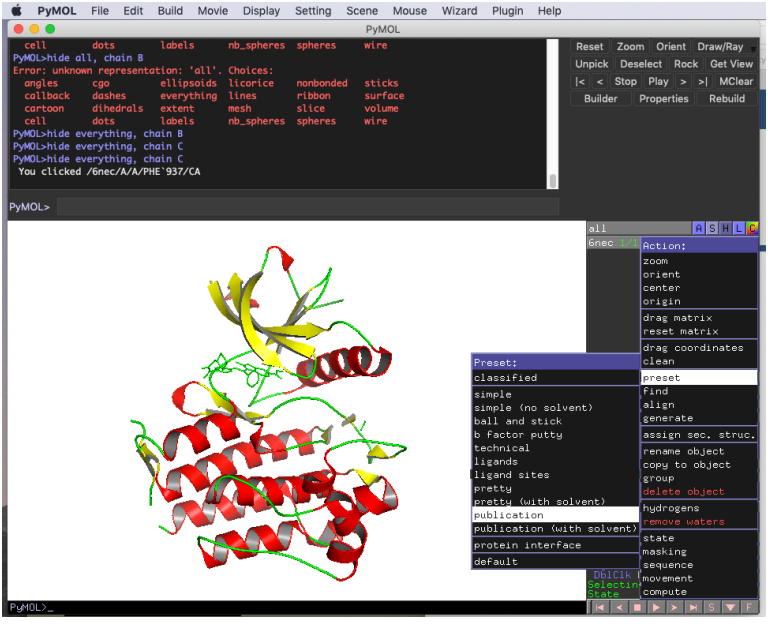
Cover images made with PyMOL



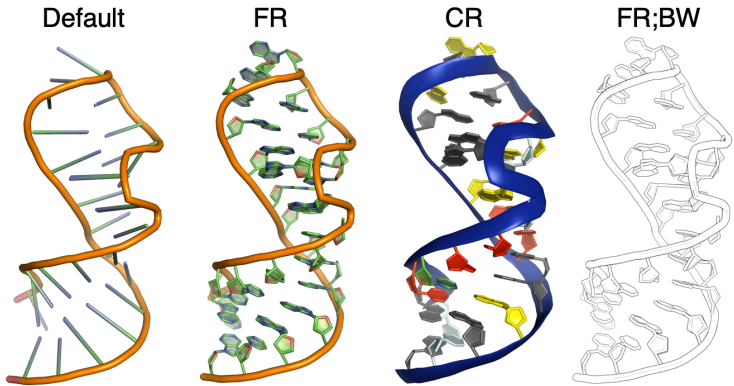




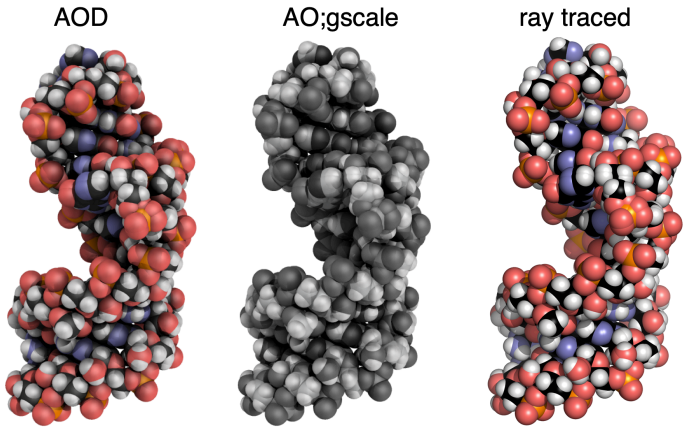
"Molecular visualization is a … tool to simplify, clarify, model, analyze, illustrate, and communicate molecular structure, properties, and function."


PyMOL File Edit Build Movie Display Setting Scene Mouse Wizard Plugin Help
PyMOL
cell dots labels nb_spheres spheres wire
PyMOL>hide all, chain B
Error: unknown representation: 'all'. Choices:
angles cgo ellipsoids licorice nonbonded sticks
callback dashes everything lines ribbon surface
cartoon dihedrals extent mesh slice volume
cell dots labels nb_spheres spheres wire
PyMOL>hide everything, chain B
PyMOL>hide everything, chain C
PyMOL>hide everything, chain C
You clicked /6nec/A/A/PHE`937/CA
PyMOL>
Reset Zoom Orient Draw/Ray
Unpick Deselect Rock Get View
|< < Stop Play > >| MClear
Builder Properties Rebuild
all A S H L C
6nec
Action:
zoom
orient
center
origin
drag matrix
reset matrix
drag coordinates
clean
preset
Find
align
generate
assign sec. struc.
rename object
copy to object
group
delete object
hydrogens
remove waters
state
masking
sequence
movement
compute
Preset:
classified
simple
simple (no solvent)
ball and stick
b factor putty
technical
ligands
ligand sites
pretty
pretty (with solvent)
publication
publication (with solvent)
protein interface
default
PyMOL>_

Customized molecular representations

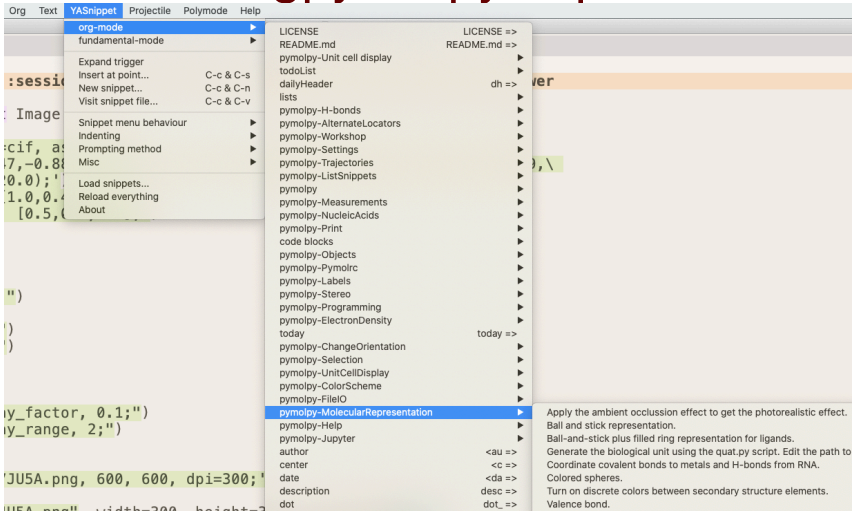


Missing molecular representations



https://github.com/MooersLab/pymolshortcuts
https://github.com/MooersLab/pymolsnips

orgpymolpysnips



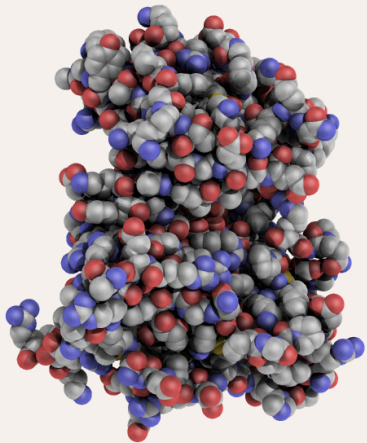
https://github.com/MooersLab/orgpymolpysnips

Code block to make one image

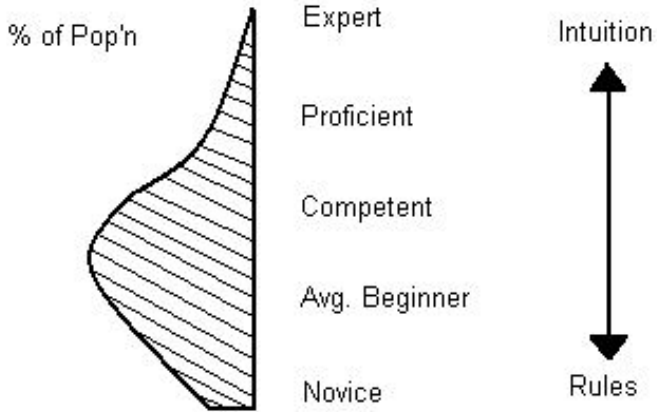
```
#+BEGIN_SRC jupyter-python :session pymol :kernel cp38 :exports both :results raw drawer
from pymol import cmd
from IPython.display import Image

cmd.do('fetch 7JU5:A, type=cif, async=0;')
cmd.do('set_view (-0.11,0.47,-0.88,-0.56,0.7,0.44,0.82,0.54,0.19,0.0,0.0,-203.71,20.89,\
6.7,-25.54,174.56,232.88,-20.0);')
cmd.do("set_color oxygen, [1.0,0.4,0.4];")
cmd.do("set_color nitrogen, [0.5,0.5,1.0];")
cmd.do("remove solvent;")
cmd.do("as spheres;")
cmd.do("util.cbaw;")
cmd.do("bg white;")
cmd.do("set light_count,10;")
cmd.do("set spec_count,1;")
cmd.do("set shininess, 10;")
cmd.do("set specular,0.25;")
cmd.do("set ambient,0;")
cmd.do("set direct,0;")
cmd.do("set reflect,1.5;")
cmd.do("set ray_shadow_decay_factor, 0.1;")
cmd.do("set ray_shadow_decay_range, 2;")
cmd.do("set depth_cue, 0;")
cmd.do("ray;")
cmd.do('png /Users/blaine/7JU5A.png, 600, 600, dpi=300;')
PATH = "/Users/blaine/"
Image(filename=_PATH + "7JU5A.png", width=300, height=300, unconfined=True)
#+END_SRC
```

https://emacsconf.org/2021/talks/molecular/

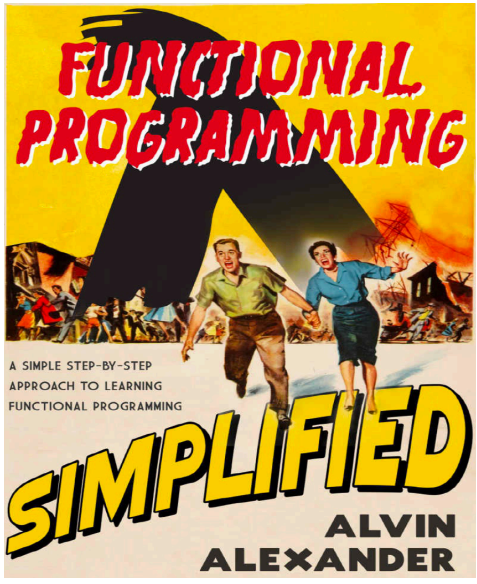
```
62 cmd.do("ray;")
63 cmd.do('png /Users/blaine/7JU5A.png, 600, 600, dpi=300;')
64 PATH = "/Users/blaine/"
65 Image(filename = PATH + "7JU5A.png", width=300, height=300, unconfined=True)
66 #+END_SRC
67
68 #+RESULTS:
69 :results:
70 # Out[1]:
71
```



Levels of expertise



Dreyfus, S. E. and Dreyfus, H. L. (1980) A five-stage model of the mental activities involved in directed skill acquisition. DTIC Document.


FUNCTIONAL PROGRAMMING
A SIMPLE STEP-BY-STEP APPROACH TO LEARNING FUNCTIONAL PROGRAMMING
SIMPLIFIED
ALVIN ALEXANDER

Functional Programmming

- A programming paradigm characterized by the use of **mathematical functions** and the avoidance of **side effects**.

- A programming style that uses only **pure functions** without **side effects**.

Features of functional programmming

- Variables are immutable.
- Recursion instead of loops.
- Pure functions without side effects.
- Higher order functions (functions that take functions as arguments)
- Division of programs into actions, calculations, and data (functional thinking).

Why learn functional programmming (at least programming with functional thinking)?

- Cleaner code that is easier to debug and maintain
- More elegant
- Sometimes faster code
- Makes you a better programmer

Two topics

- Ways to execute elisp code
- Functional features of elisp

Methods of running elisp

- elisp file (.el)
  (documentation 'main) ; C-x C-e at space to right of ')'.
  (documentation 'main) ; C-M-x inside of the parentheses.

- scratch buffer ; C-j

- M-x eval-buffer ; evaluate whole buffer
- M-x eval-region ; evaluate region

Elisp REPLs (3 options inside Emacs)

- M-: ; REPL in mini buffer
- M-x ielm ; eval one expression at a time
  (defalias 'erepl 'ielm)
  M-x erepl
- M-x eshell

elisp REPL ouside of Emacs

```
alias erepl="rlwarp emacs --batch --eval \"(progn (require 'cl)
↪  (loop (print (eval (read)))))\""
```

Source: https://www.reddit.com/r/emacs/comments/58ciar/question_can_elis_be_run_outside_of_emacs/

Elisp script

```
#!emacs --script
(defun main ()
  "Print the version of Emacs and print arguments as a list.
  Example: chmod +x test5.el && ./test5.el dog cat
  Modified test4.el script by Dr. John Kitchin found here:
  https://kitchingroup.cheme.cmu.edu/blog
  /2014/08/06/Writing-scripts-in-Emacs-lisp/"
(print (version))
(print (format "Called with %s" command-line-args))
(print (format "You did it! You passed in %s!"
↪  command-line-args-left))
(print (format "This is the documentation string for main():
  %s" (documentation 'main))) )
(when (member "-scriptload" command-line-args) (main))
```

Code block with output in the RESULTS drawer

Place cursor inside code block and enter C-c C-c to run the code.

```
#+BEGIN_SRC emacs-lisp :results value scalar
(*40 1000 1000 1000 1000 1000 1000 1000)
#+END_SRC
#+RESULTS:
: 40000000000000000000
```

Note that the ":results value scalar" was needed with emacs-lisp. In the org-babel configuration, emacs-lisp has to be in the list of languages.

Config for matching parentheses

```
(setq show-paren-delay 0)
(show-paren-mode t)
```

'

Syntax of lambda or anonymous functions

```
(lambda (arg-variables...)
  [documentation-string]
  [interactive-declaration]
  body-forms...)
```

lambda function with one arugment

```
#+BEGIN_SRC emacs-lisp :results value scalar
((lambda (a) (1+ a)) 19)
#+END_SRC
#+RESULTS:
: 20
```

One required and one optional argument

```
#+BEGIN_SRC emacs-lisp :results value scalar
((lambda
  (a &optional b)
  (if b (+ a b)
    (1+ a)))
10 12)
#+END_SRC
#+RESULTS:
: 22
```

One required and eight optional arguments

```
#+BEGIN_SRC emacs-lisp :results value scalar
((lambda
  (a &rest b)
    (+ a (apply '+ b)))
10 20 30 40 50 60 70 80 90)
;(+ 10 (+ 20 30 40 50 60 70 80 90))
#+END_SRC
#+RESULTS:
: 450
```

2 required, 2 optional, and 1 rest argument

```
#+BEGIN_SRC emacs-lisp :results value scalar
(
(lambda (a b &optional c d &rest e)
  "Add two or more variables."
  (+ a b c d (apply '+ e)))
1 2 3 4 10 20 30 40 50 60 70 80 90
)
#+END_SRC
#+RESULTS:
: 460
```

funcall applied to a lambda function

```
#+Begin_SRC emacs-lisp :results value scalar
(funcall (lambda (c d) (expt c d)) 10 4)
#+End_SRC
#+RESULTS:
: 10000
```

Mapping functions

A function that applies a give function (not a special form nor a macro) to a list or other collection.

- mapcar
- mapconcat
- mapcan

mapcar applies a function to a list

mapcar takes a function and a list as arguments. The function and list need to be single-quoted.

```
#+BEGIN_SRC emacs-lisp :results value scalar
(mapcar '1+ '(9 10 11))
#+END_SRC
#+RESULTS:
: (10 11 12)
```

apply operates on arguments

In contrast, apply operates on arguments that may or may not be lists and applies the first argument to the remaining arguments.
The function and any lists need to be single-quoted.

```
#+BEGIN_SRC emacs-lisp :results value scalar
(apply 'max 3 4 7 11 '(4 8 9))
#+END_SRC

#+RESULTS:
: 11
```

mapcar applying a lambda function

```
#+Begin_SRC emacs-lisp :results value scalar
((lambda (x)(capitalize x))
  "lisp")
#+END_SRC
#+Results:
: "Lisp"

#+BEGIN_SRC emacs-lisp :results value scalar
(mapcar '(lambda (x)(capitalize x))
  '("lisp" "is" "cool"))
#+END_SRC
#+RESULTS:
: ("Lisp" "Is" "Cool"
```

mapcar applying *car* and *cdr*

```
#+BEGIN_SRC emacs-lisp :results value scalar
(mapcar 'car '((1 2) (3 4) (5 6)))
#+End_SRC
#+Results:
: (1 3 5)

#+BEGIN_SRC emacs-lisp :results value scalar
(mapcar 'cdr '((1 2) (3 4) (5 6)))
#+END_SRC
#+RESULTS:
: ((2) (4) (6))
```

mapcar applying the *length* function

```
#+BEGIN_SRC emacs-lisp :results value scalar
(mapcar 'cdr '((1 2) (3 4) (5 6)))
#+END_SRC

#+RESULTS:
: ((2) (4) (6))
```

mapcan returns a single list

```
#+BEGIN_SRC emacs-lisp :results value scalar
(mapcan 'cdr '((1 2) (3 4) (5 6)))
#+END_SRC

#+RESULTS:
: (2 4 6)
```

mapconcat

Takes a function, a list, and a separator as arguments.

```
#+BEGIN_SRC emacs-lisp :results value scalar
(mapconcat 'symbol-name
       '(Green eggs and ham.)
       " ")
#+END_SRC
#+RESULTS:
: "Green eggs and ham."
```

mapconcat applying a lambda function

```
#+BEGIN_SRC emacs-lisp :results value scalar
(mapconcat (lambda (x) (format "%c" (1+ x)))
       "HAL-8000"
       "")
#+END_SRC
#+Results:
: "IBM.9111"
```

Really HAL 9000 from the 1968 film *2001: A Space Odyssey*. IBM-9111 was released in 2005.

Acknowledgements

Jens Jensen's "Functional programming: an (Emacs) Lisp view"
`https://www.youtube.com/watch?v=MMRAfd9_d6k`

Funding:

- Warren Delano Memorial Open-Source PyMOL Fellowship
- NIH: R01 CA242845, R01 AI088011
- NIH: P20 GM103640, P30 CA225520, P30 AG050911-07S1
- OCAST HR20-002