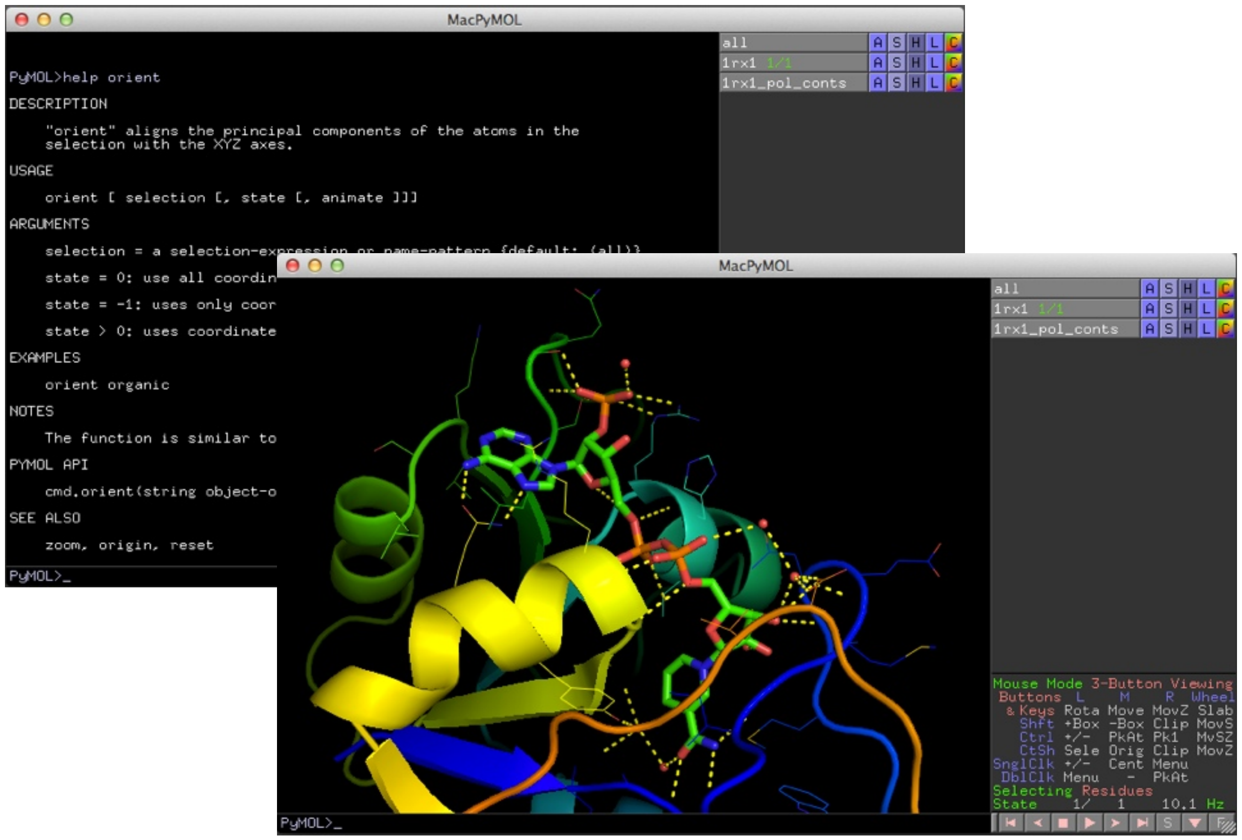
Intermediate PyMOL



```
PyMOL>help orient
DESCRIPTION

    "orient" aligns the principal components of the atoms in the
    selection with the XYZ axes.

USAGE

    orient [ selection [, state [, animate ]]]

ARGUMENTS

    selection = a selection-expression or name-pattern {default: (all)}

    state = 0: use all coordin

    state = -1: uses only coor

    state > 0: uses coordinate

EXAMPLES

    orient organic

NOTES

    The function is similar to

PYMOL API

    cmd.orient(string object-o

SEE ALSO

    zoom, origin, reset
PyMOL>_
```

Preface

Copyright Notice

About This Booklet

This is a follow-along guide for the *Intermediate PyMOL* classroom tutorial taught by Schrödinger. It covers advanced selection making, settings and commands, automation with scripts, and commonly requested activities. This tutorial exposes the foundation that differentiates novice users from power users of PyMOL. Once completed, the user should be able to operate flexibly with selections, use commands and settings, extend PyMOL to create new functionality and automate commonly encountered tasks.

Those new to PyMOL should first attend or read the *Introduction to PyMOL* tutorial before attempting this tutorial.

Requirements

Before starting this tutorial, please ensure that you have the following.

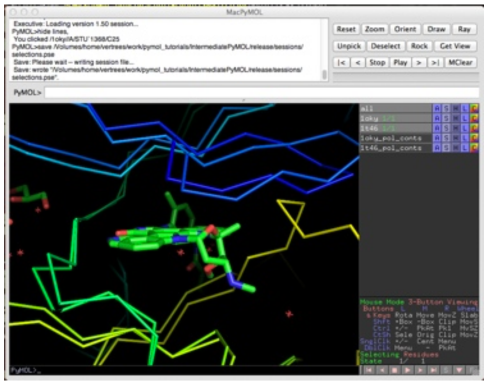
- PyMOL version 1.2 or greater
- A 3-button wheel mouse
- Tutorial-specific data files found in the **IntermediatePyMOL** subfolder of the **PyMOLTutorials** archive supplied by Schrödinger. This archive is typically provided as a compressed "ZIP" file, which should be extracted on to your Desktop
- The reader should be very familiar with PyMOL to the point of having read and understood, or attended the Introduction to PyMOL classroom tutorial taught by Schrödinger.

Please also make sure that your keyboard Caps Lock key is turned off when using PyMOL.

Typing Commands

Load the Tutorial Session File

Please load the file **PyMOLTutorials/IntermediatePyMOL/sessions/selections.pse**. Your screen should look similar to Figure BP1. This session consists of two kinase molecules shown as rainbow colored ribbons with their ligands, Staurosporine and Imatinib, shown as sticks. We will use this PyMOL session to help us learn about commands and selections.



```
Executive: Loading version 1.50 session...
PyMOL>hide lines,
 You clicked /1oky//A/STU`1368/C25
PyMOL>save /Volumes/home/vertrees/work/pymol_tutorials/IntermediatePyMOL/release/sessions/
selections.pse
 Save: Please wait -- writing session file...
 Save: wrote "/Volumes/home/vertrees/work/pymol_tutorials/IntermediatePyMOL/release/sessions/
selections.pse".
```

Figure BP1. PyMOL tutorial session for selection making.

The Command Lines

PyMOL has two command line interfaces, one is at the bottom of the Display Area and the other is at the bottom of the Upper Control Window. Figure CL1 shows the PyMOL user interface with the two command lines highlighted. Please take a moment to find these command lines. Click on each and verify that you can enter text into the command line. Into the Upper Control Window's command line, please type `orient` into and press `ENTER`. PyMOL orients the scene by the principal axes determined by the coordinates of all the atoms in the scene. Now, please type `zoom resn STU` into the command line at the bottom of the Display Area. Press `ENTER`. PyMOL zooms in on the molecule whose residue name field in the PDB is "STU." This happens to be the Staurosporine molecule.

Features Common to Both Command Lines

Both command lines in PyMOL support the same basic functionality and behave similarly to the Unix/Linux-style command line interfaces. You can enter text by typing into them, reposition the cursor, review your command history, and get help through auto-completion.

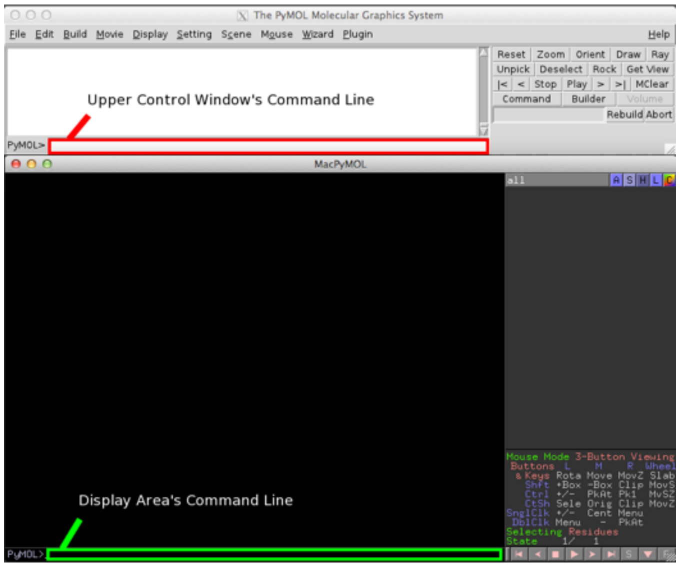


Figure CL1. PyMOL GUI with highlighted command lines.

Try It—Positioning the Cursor

You can position the cursor at the start and end of the command lines. Please type `orient`, but this time **do not** press `ENTER`. Your cursor should be at the end of the command line. Press `CTRL-A`. PyMOL positions the cursor at the beginning of the command line. Now press `CTRL-E`. PyMOL repositions the cursor at the end of the command line.

You can also use the arrow keys to navigate the command lines. Please press the left arrow key until the cursor is underneath or directly to the left of the 'e' in `orient`. Now, press `CTRL-K`. PyMOL deletes all the text to the right of the cursor. Please finish typing the command `origin` and press `ENTER`.

Try It—Examining the Command History

You can scroll back through your history by pressing the `Up Arrow`. Please try that now and review the commands you've entered. You can scroll forward through the history using the `Down Arrow`.

Try It—Auto-completion

You can get PyMOL to auto-complete what you're typing by pressing the `TAB` key. For example, please erase all the text on the command line and then type in `orig` and press `TAB`. PyMOL completes the command name, `origin`. If the remainder of the text is unique, PyMOL will auto-complete the word for you. If the remainder of the text is not unique, PyMOL will show you a list of possible completions. To see this, please type `ori` and press `TAB`. PyMOL displays:

```
 parser: matching commands:
  orient  origin
```

which means that it found two possible completions of the string `ori`; the commands `orient` and `ori-gin`.

Two last tips: to see a list of all commands, press the `TAB` key when there is no text on the command line; and, to get help for using the keyboard you can type `help keyboard`. Don't forget to press the `ESC` key to see the help text in the console if it doesn't show up.

Special Features of the Upper Control Window's Command Line

Now we focus on special features of the Upper Control Window's command line. Please click there to give it focus.

The command line in the Upper Control Window also supports copy and paste. You can copy text from an external application like Notepad or your web browser and paste it into this command line. Let's see this in action. Please type `select none` into the command line, **but do not press** `ENTER`. We will now copy the text. Please hold the `SHIFT` key and press the left arrow key until all the text on the command line has been selected. (You can also select text with the mouse.) Now press `CTRL-X` and PyMOL will cut the text from the command line. You can now paste this text into another application if you desire. For now, let's just paste it back into PyMOL by pressing `CTRL-V`. Only text cut with `CTRL-X` can be pasted.

Try It—Special Features of the Display Area's Command Line: Copying, Cutting, and Pasting Atoms

Now we focus on special features of the Display Area's command line. Please click there to give it focus. *Please note some of the features described in this section are new in PyMOL v1.5, so if you have an ear-lier version these commands may not work for you.*

The command line beneath the Display Area has two different modes depending on whether there is text on the command line or not. If there is text on the command line maps its special keys to operate on the text. So, as you saw, `CTRL-A` will go to the start of the command line. However, when the command line is free of text PyMOL will map those same keys to operate on selected atoms in the scene. For example, when the command line is empty, `CTRL-A` will select all atoms. These actions for special keys are similar to a text editor. PyMOL allows you to use `CTRL-C` to copy atoms, `CTRL-X` to cut atoms, and `CTRL-V` to paste the selected atoms into a new object. Please note that when copying or cutting atoms, PyMOL acts only on the *currently enabled* selection. So, if no selection is enabled no atoms will be copied.

Let's try copying a selection and pasting into a new object. Using the keyboard we will create a new selection consisting of the first complete organic small molecule. Then we will copy and paste it into a new and distinct PyMOL object. Please select the first organic small molecule in the scene by typing the following: `select bm. first organic` and press `ENTER`. To see what we've selected, please type `zoom sele`. Now please press `CTRL-C`. PyMOL copies the selected atoms into memory, even though nothing visually changes. Press `CTRL-V`. PyMOL pastes your selected atoms into a new PyMOL object, named "obj01".

This functionality only exists when focus is on the Display Area and when the command line is free of text. PyMOL supports other operations on atoms. Please see Table K1 to see a listing of all the keys and their respective functions in PyMOL.

Special Key	Text on the Command Line	Empty Command Line
`TAB`	Auto-complete text	Display list of all commands
`CTRL-A`	Position cursor at beginning of line	Select all atoms
`CTRL-C`		Copy currently selected atoms
`CTRL-E`	Position cursor at end of line	
`CTRL-I`		Invert selection
`CTRL-K`	Delete all text to the right of the cursor	
`CTRL-V`	Paste into the command line	Paste copied atoms into new object
`CTRL-X`		Cut atoms
`CTRL-Y`		Undo
`CTRL-Z`		Redo

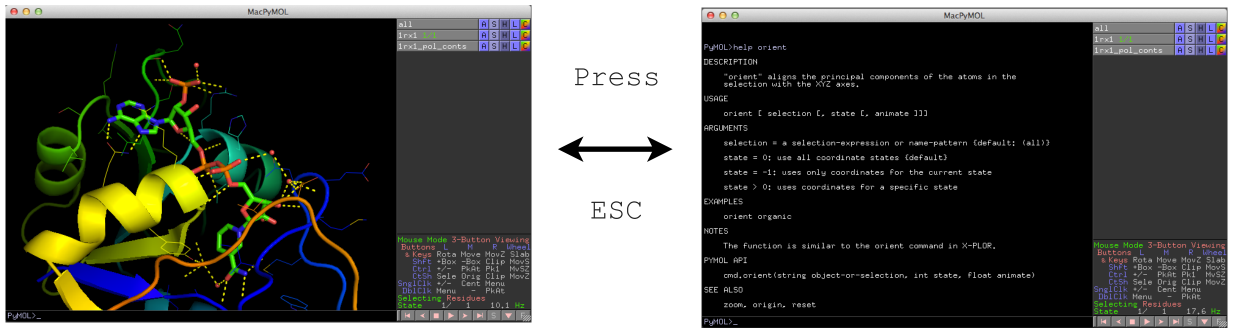
Table K1. List of known special keys and their functions.

Introduction to PyMOL Commands

PyMOL has more than 160 commands and each command performs a specific action. But, don't be intimidated, getting help is easy.

Try It—Getting Help With Commands

You can get help for each command by typing `help` followed by the command name. Please type `help orient`. The Upper Control Panel shows help text for the `orient` command. This help text is also displayed in the Display Area's console, but the text console is currently hidden by the graphics. You can switch from graphics mode to console mode by pressing `ESC` when focus is in the Display Area, as shown in Figure C1. So, please click somewhere inside the Display Area and press the `ESC` key. PyMOL switches from graphics mode to console mode where you should see the help text for the `orient` command.



```
PyMOL>help orient
DESCRIPTION

    "orient" aligns the principal components of the atoms in the
    selection with the XYZ axes.

USAGE

    orient [ selection [, state [, animate ]]]

ARGUMENTS

    selection = a selection-expression or name-pattern {default: (all)}

    state = 0: use all coordinate states {default}

    state = -1: uses only coordinates for the current state

    state > 0: uses coordinates for a specific state

EXAMPLES

    orient organic

NOTES

    The function is similar to the orient command in X-PLOR.

PYMOL API

    cmd.orient(string object-or-selection, int state, float animate)

SEE ALSO

    zoom, origin, reset
PyMOL>_
```

Figure C1. Pressing the **ESC** key when focus is in the Display Area switches PyMOL from graphics mode to console mode.

You can also get the usage for a specific command by simply typing the command name followed by a question mark. Please type `orient ?`. PyMOL displays the usage for the **orient** command:

```
Usage: orient [ selection [, state [, animate ]]]
```

This might look complicated, but it is simply telling you that PyMOL expects the command name, `orient`, followed by up to three optional arguments: selection, state and animate. Bracketed words, like `[animate]`, indicate that the argument is not required for the command to work properly. You will notice that all three arguments are optional; so, just typing `orient` alone is valid usage. Please type `orient`. Notice that PyMOL orients the scene with respect to all atoms. Now, please type `orient organic`. PyMOL now orients the scene, but only with respect to the coordinates of the small organic molecule(s) in the scene. Because the word `organic` immediately followed `orient`, `organic` is assigned to the `selection` argument per the usage. You can manually specify an argument by typing the argument name followed by an equal sign followed by your input. For example, `orient animate=1`. Please type that now.

Command Syntax

All PyMOL commands follow a similar syntactical structure. The first word is the name of the command, like `orient`, `ray`, `align`, or `intra_fit`, followed by a space. After the space is a comma-separated list of arguments. So typical usage looks like the following: `command arg1, arg2, arg3`.

Often a command has sensible default values that makes manual specification of the arguments unnecessary. For example, as we saw above, the `orient` command's first argument is the selection to orient, which defaults to all atoms. You can provide a specific set of atoms to orient if you want, but the default is all atoms. Command completion works for all commands and many of their arguments.

Before we move on to working with selections let's learn how to use a new command and determine its parameters. We'll save a custom sized and custom resolution image of the scene with the `png` command.

Try It—Create a Custom Sized and Resolution Image

In this example we will use the `png` command to save a 3 inch by 2 inch, ray traced image, at 300 dots per inch (DPI). However, a review of how resolution works in (raster) images is useful so we can understand if the `png` command has returned the desired result.

Every (raster) image has a property called **resolution**—usually measured in dots per inch or DPI. Images for presentation or display on a computer are best set between 72 DPI and 200 DPI because that's what the display is capable of reproducing. Images for printing or for publication are typically required to be of much higher resolution, like 300 to 1200 DPI. Given the resolution of an image as DPI and its size in inches, we can determine how many dots, or **pixels**, the image should be.

To determine the height and width in dots we multiply the resolution by the size in inches. So, to create our 3 inch by 2 inch image at 300 DPI we need an image that's 900 dots (3 inches x 300 DPI = 900 dots) wide by 600 dots (2 inches x 300 DPI) tall. Now that we know the width, height and DPI, let's see how to specify that information to the `png` command. See Figure P1 for an illustration of the calculation.

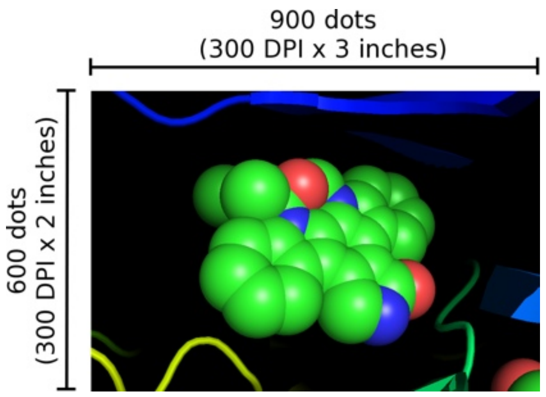


Figure P1. Example calculation of image size in dots, or pixels, given a resolution of 300 DPI, width of 3 inches and height of 2 inches.

Please type `png ?` to see the usage for the `png` command. PyMOL shows the usage as:

```
    Usage: png filename [, width [, height [, dpi [, ray [, quiet [,
        prior [, format ]]]]]]]
```

For this image, we need to provide a filename, and the optional parameters for width, height, and DPI. Luckily, PyMOL has exactly those arguments for the DPI command. Now please use the numbers we calculated with the png command by typing:

```
    png ~/pymol.png, width=900, height=600, dpi=300, ray=1
```

PyMOL ray traces the custom image and saves it as "pymol.png" in your home directory. By setting the ray argument to 1 we forced PyMOL to ray trace the image before saving it. Please note it is important to repeat this procedure for all your custom high resolution images.

Let’s move on to learning about atom selections in PyMOL. Most PyMOL commands are operations that deal with atom selections. This means that under the hood, creating and operating on atom selections are the most commonly performed actions in PyMOL. For those who frequently use PyMOL, mastering atoms selections may dramatically increase your productivity.

Mastering Selections

Objects vs Selections

An **object** in PyMOL is an entity represented in memory as a special Python object or a compiled graphics object (CGO). Objects are commonly molecules, molecular complexes, and CGOs. Molecular objects have a hierarchical structure that can include molecules, segments, chains, residues, atoms, and more. Objects appear by name in PyMOL’s Object Menu Panel. See Figure OS1.

An **atom selection** or **selection** is a user-defined set of atoms chosen from one or more objects. Selections themselves do not have any properties except for a name. Selections are a just way to register our interest in a special list of atoms of our choosing. Selections appear in the Object Menu Panel by their name surrounded by parentheses. See Figure OS1.

We will now use selections to perform a few common tasks. First, we will create selections for solvent and inorganic atoms and then remove these atoms. Next, we will create a selection named "pocket" that contains only those atoms within 5 Angstroms of the Staurosporine small molecule. We then highlight this pocket by showing the selected atoms as lines and surface and finding polar contacts.

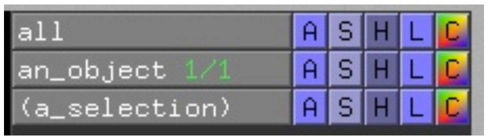


Figure OS1. Objects and selections in the Object Menu Panel. Selection names are always enclosed in parentheses.

Try It—Removing Unwanted Atoms

PDB files typically consist of four types of molecule: protein, solvent, small organic, and inorganic. However, often we are only interested in the protein and the ligand, usually a small organic compound. So, let’s select and remove everything but the protein and ligand. To create the named selection please type, `select unwanted, solvent or inorganic`. PyMOL creates the selection, displays "(unwanted)" in the Object Menu Panel, and indicates the atoms with pink dots. In the console window PyMOL also offers feedback on the selection by reporting the selection’s size, here 400 atoms. We can now control these 400 atoms all at once by referring to their selection name, "unwanted". They’re in the way now, so let’s remove them. For the selection "(unwanted)", please click **A→Remove Atoms.** PyMOL removes the atoms and the "(unwanted)" selection. No other atoms were effected, just those that we selected for removal.

Try It—Identifying the Binding Pocket

Another common task is identifying the region around a ligand. Once identified, we can prepare it for manual molecular editing, find polar contacts nearby, or even clean up the selection with a minimizer.

Let's begin by creating the named selection, "pocket", that consists of all atoms in 1oky within 5 Angstroms of the Staurosporine molecule. In the PDB file 1oky.pdb the Staurosporine molecule's name ("residue name" field) is "STU". So, in PyMOL we'll use `resn STU` to refer to Stuarosporine; `resn` just stands for "residue name". Please create this named selection by typing: `select pocket, 1oky within 5 of resn STU`. PyMOL creates the named selection, displays "(pocket)" in the Object Menu Panel, and indicates the selected atoms with pink dots. Your screen should look like Figure RS1. This one concise command filters through thousands of atoms to select just those we desire. Now, for the "(pocket)" selection, please click **S→Show→Lines**. PyMOL shows the selected atoms as lines. To view the binding pocket as a surface, please click **S→Show→Surface** for the selection.

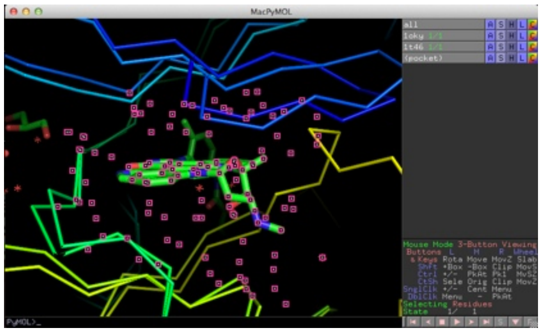


Figure RS1. Atoms within 5 Angstroms of the organic small molecule Staurosporine selected and indicated with pink dots.

Let's finish this example by finding all polar contacts within the selection. For "(pocket)", please click, **A→find→polar contacts→within selection.** PyMOL identifies and shows the polar contacts as yellow dashed lines.

Here we manually created selections for the binding pocket. PyMOL can also auto-generate these selections with a few mouse clicks. Let's look at that, too.

Try It—Identifying the Binding Pocket with Auto-generated Selections

Just like the **A→Action→Presets** menu has a set of convenient presets for visualization, **A→Action→Generate→Selection...** has a list of selections it can auto-generate. The current list consists of: all, polymer, organic, solvent, polar hydrogens, non-polar hydrogens, donors, acceptors, and surface atoms. Let's explore these selections by repeating the previous example of finding the binding pocket, but using only the mouse and the other protein in the session, 1t46. Please disable all objects and selections except for 1t46. Recall that clicking on the object name in the Object Menu Panel will toggle whether the object is enabled or not. Let PyMOL find the organic small molecule in 1t46 by for you by clicking **A→generate→selection→organic** for 1t46. PyMOL creates the named selection "(1t46_organic)", and indicates the selected atoms with pink dots. Please rename that selection "(pocket)" by clicking **A→rename selection**, for "(1t46_organic)". PyMOL creates the "(pocket)" selection overwriting the previous one. Now, let's expand the selection to all atoms within 5 Angstroms of 1t46 by clicking **A→modify→expand→by 5 A.** Because we have two molecules loaded, PyMOL selected *every* atom within 5 Angstroms of 1t46; that means, some atoms in 1oky were also selected even though 1oky is not enabled. Let's now restrict the selection to just 1t46 by clicking **A→modify→restrict→to object→1t46**. We can show lines for the selection by clicking **S→show→lines** for "(pocket)"**.** Please also show the surface for "pocket" by clicking **S→Show→Surface.** Last, we can find the polar contacts by clicking **A→find→polar contacts→within selection**, for "(pocket)".

PyMOL Selection Language

PyMOL ships with a robust selection language. Using this language requires us to type selection commands into the command line. Please click in the Viewer Window to focus on the command line. To completely understand selections in PyMOL, we need to understand two more concepts: selection-expressions and how to use the `select` command.

Selection-Expressions

In PyMOL, a "selection-expression" or "selector" is text representing a specific set of atoms to be selected. For example, above we typed, `zoom resn STU`. The command is `zoom` and the selection-expression is `resn STU`. In PyMOL the keyword `resn STU` means the residue named "STU" as read from the PDB file. There could be thousands atoms across multiple objects, and PyMOL gives us a way to focus on just what we want—this one residue. A selection-expression can be as simple as a single word, like `solvent`, or as complex as a multi-line, multi-parenthetic, multi-operator expression spanning multiple objects and properties.

Single Word Selectors

PyMOL makes it easy to select commonly used sets of atoms by providing built-in single-word selectors. For example, the `organic` selector finds all organic small molecules. Another example is `polymer` which selects only those atoms in a polymer—like proteins or nucleic acids.

Single Word	Abbreviated Form	Description
`all`	`all`	All atoms
`none`	`none`	No atoms (empty selection)
`solvent`	`sol`	All waters
`organic`	`org`	All atoms in non-polymer organic compounds
`inorganic`	`ino`	All non-polymer inorganic atoms and ions
`polymer`	`poly`	All atoms in a polymer

Table SE1: List of commonly used single-word selectors.

Table SE1 has a list of the most commonly used single-word selection-expressions. Please see Index A for the entire list of single-word selectors. Please review this table. Frequent users of PyMOL are suggested to memorize the entries in this table; this will save time in the future. Each single-word selection-expression has an abbreviated form. For example, `org` is the abbreviated form of `organic`.

Property Selector	Abbreviated Form	Description
`name`	`n.`	list of up to 4-letter codes for atoms in proteins or nucleic acids; example: `name CA`
`resn`	`r.`	three-letter residue name; example: `resn CYS`
`resi`	`i.`	residue identifier (number); example: `resi 4`
`chain`	`c.`	chain name; example: `chain A`
`ss`	`ss`	secondary structure ('s' or 'h' or ''); example: `ss 'h'`
`b`	`b`	b-factor value; example: `b < 30`
`alt`	`alt`	alternate coordinates; example: `alt 'B'`

Table PS1: Commonly used property selectors.

Try It—Using Single-Word Selection-Expressions

Please select all atoms by typing, `select all`. PyMOL shows the pink indicator dots over all atoms, regardless of whether or not the atoms are displayed or enabled. Now, create the empty selection by typing `select none`. PyMOL updates the selection such that no atoms are in it. Now, please type `select polymer`. Notice that PyMOL has located and indicated with pink dots all polymer atoms. This single-word selection-expression, `polymer`, works on proteins and nucleic acids. We already removed the solvent and inorganic atoms, so they're not present in this session. You can verify that by typing `select solvent` and noticing that no atoms were selected.

Now, let's look at selecting atoms based upon their properties as read in from structure files.

Property Selectors

PyMOL reads files in many formats. Some of the data fields in these formats allow PyMOL to assign properties to atoms. For example, the PDB file format defines a field for "residue name," which corresponds to PyMOL's `resn` single-word selector. PyMOL selects atoms according to these properties using *property selectors* and *identifiers*. The property selectors correspond to the fields in the data files, like residue name, and the identifiers, like `resn STU`, correspond to the target words or to match or the target numbers to compare. The most common property selectors are shown in Table PS1. Frequent user of PyMOL are suggested to memorize the contents of Table PS1.

Let's sample some common property selectors. Then, we will follow by combining these property selectors with single-word selectors to create more useful customized selections.

Try It—Using Property Selectors

A common task in PyMOL is coloring a protein by secondary structure. Let's try this by coloring the helices blue, beta sheets green, and loops red for 1t46. Let's start by showing the protein as lines only; so please **A→preset→default** for 1t46. We will use the `color` command; it takes a color name and a selection-expression. Please color the helices blue by typing, `color blue, ss 'h'`. PyMOL colors the helices blue. The last few characters of that command, `ss 'h'` specify all atoms of the secondary structure type 'h' or helix. Now, let's color the beta sheets green. Please type, `color green, ss 's'`. This colors all beta sheets green; if nothing changed then there are no beta sheets present. Now, let's finish by coloring all loops red. The secondary structure code for loops in PyMOL is either the blank string '' or 'L' depending on the PDB file. Please type, `color red, polymer and ss ''`. The selection expression in that command contains two clauses `polymer` and `ss ''`, connected by the boolean operator "and". As you know, the first clause selects all polymer atoms. The second clause selects all atoms whose secondary structure type matches is blank. The final result is a selection-expression that selects only those atoms that are polymer and have no secondary structure specified. Let's see the final result as cartoons, so please type "as cartoon".

Try It—More Complex Selection Expressions

Before moving on, please show your protein as lines, by typing `as lines`. Please take a few moments to type each of the following selections in Table Sel1. Be sure to ask if you don't know why a given selection acts as it does. For the last selection in Code MSC.1 show your protein as b-factor putty by clicking **A→Preset→B factor putty** for 1t46. The selected atoms should correspond to the size and color of the b factor putty.

Code MSC.1: Examples of advanced selections

```
# select all carbons and all nitrogens
select n. c or n. n

# select all lysines
select resn lys

# select residues numbered 646 through 655
select i. 646-655

# select all atoms with alternate coordinates
select not (alt '')

# select all helices in chain A
select ss 'h' and c. A

# select atoms in 1t46 with b-factor greater than 30
select 1t46 and b>30
```

Settings

PyMOL has over 700 settings. Unfortunately, getting help for settings is not easy—and in some cases there is no help for a given setting. In this chapter we will teach you how to use settings, review commonly used settings, and provide tips on figuring out what a given setting might do.

Controlling Settings

To get the value of a setting we use the `get` command. Please type `help get` and take a moment to read the documentation. You can get the value of a setting by typing `get` followed by the setting name; for example, `get cartoon_color`. You can change setting values to see how they affect your scene. If you don't like the new setting value, just restore that setting to its original value.

To set a value we use the `set` command. Typical usage is `set setting-name, value`. If you do not specify `value`, the default is 1. For example `set ray_opaque_background, 1` and `set ray_opaque_background` both set the setting to 1.

There are a few types of setting. Valid values for boolean settings are `0`, `1`, `on`, `off`, `true`, `false`. Numeric values can be integral, like `-2` or `7`, or floating point, like `0.25`. String values are usually just text, like: `volume`. Color values may be specified by their name, like `red`, `orange` or `lightblue`; by their PyMOL-specific index, like `30`; by their hexadecimal RGB value, like `0xff00ff`. There are two possible way to pass array-type settings: a space separated list, like `0.2 0.5 0.9`; or a bracketed, comma-separated list, like `[0.2, 0.5, 0.9]`. To see the entire list of settings, click **Setting→Edit All...**

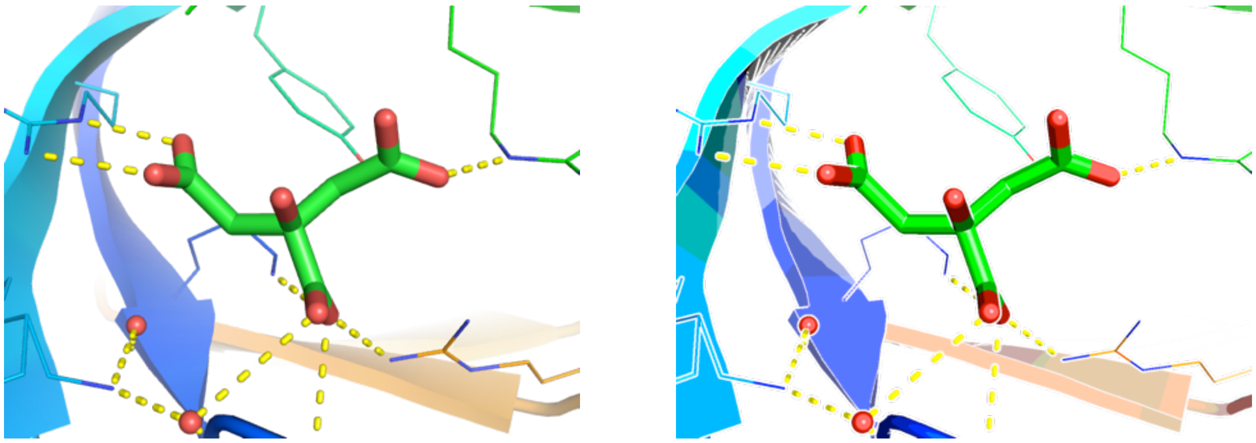


Figure R1. The left image shows a ray-traced scene when `ray_trace_mode` was 0. The right image shows the same image after `ray trace mode` was set to 3.

Try It—Adjusting the Ray Tracing Mode

Please click **A→Preset→Ligand Sites→Cartoon** for 1t46. To illustrate how settings effect PyMOL, let's consider the `ray_trace_mode` setting. This setting controls the type of post-rendering effects PyMOL applies to the image, as shown in Figure R1. Please first get the setting by typing, `get` `ray_trace_mode`. By default it is set to 0. Now, please type `ray` to ray trace the current scene. Take note of the image. Now, set `ray_trace_mode` to 3 by typing, `set ray_trace_mode, 3`. Please type `ray`. Notice the difference between the previous image and the current. When `ray_trace_mode` is set to 3, PyMOL applies a post-rendering color quantization step that gives the image a cartoon-like quality.

Setting Scope

Settings can be applied globally, to objects, or to specific atoms and bonds. You've already adjusted a global setting when you typed, `set ray_trace_mode, 3`. To adjust a setting for a specific object or selection simply add the object name after the setting value.

Try It—Creating a Ball-and-Stick Figure by Adjusting Selection-level Settings

Please load the file **PyMOLTutorials/IntermediatePyMOL/sessions/settings.pse.** To illustrate setting scope we will create Figure LB1, a figure of a ligand bound to a protein where the ligand will be shown as ball-and-stick and the protein as a grey-colored surface. First, we show the entire system as surface. Second, we select the ligand and show it as sticks and spheres. Next, we orient the ligand. After that, we adjust the `stick_radius` and `sphere_scale` settings to create the ball-and-stick representation for ligand. Next, we color the surface grey. Last, we ray trace the image. Let's begin.

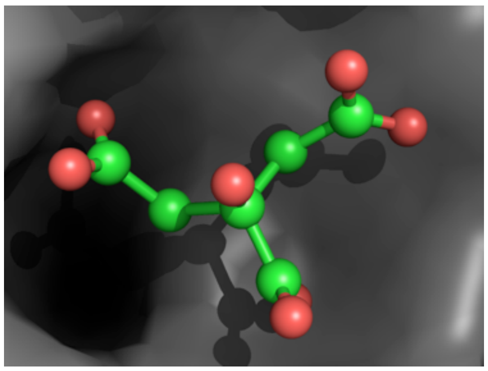


Figure LB1. Ball-and-stick figure made from customized settings.

Please show the system as a surface by typing, `as surface`. After a few moments, PyMOL shows the protein by its surface representation—don't worry if the small molecule disappeared, we'll get it back; by default PyMOL does not surface ligands. Next, please select the ligand by typing, `select ligand, org`. Now, orient on the ligand by typing, `orient ligand.` PyMOL zooms on in on the organic small molecule and aligns it by its principal axes. Now, please show the small molecule as sticks and spheres by typing the following two commands: `as sticks, ligand`, and then, *`show`* `spheres ligand`. Currently, your screen should look similar to Figure LB2. The sphere occlude the sticks. We need to adjust the sizes of the spheres and sticks to perfect the representation. So, please adjust the sphere scale by typing, `set sphere_scale, 0.25, ligand`. The spheres shrink to a smaller radius. The sticks in the ligand are still too thick. Let's make them thinner by typing, `set_bond stick_radius, 0.125, ligand`. Finally, set the surface color to grey by typing, `set surface_color, grey`. If you'd like to ray trace your image click the Ray button or type `ray`. Your final image should look like Figure LB1.

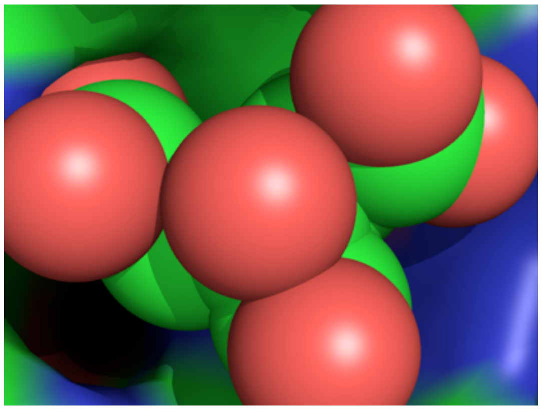


Figure LB2. Ligand shown as spheres and sticks. The spheres occlude the sticks. We fix this by adjusting the sphere_scale and stick_radius settings.

If you're ahead of the class, try adjusting a few more settings. Try using the `set_bond` command to set the `stick_transparency` of ligand to 0.5: `set_bond stick_transparency, 0.5, (ligand)`. Also, try setting the `stick_color` to `lightblue`. Last, try enabling `valence` on the ligand.

Once you have a slight grasp on the commands, you can begin to write scripts and automate tasks in PyMOL. Being able to use PyMOL from the command line is a powerful next step in using PyMOL.

Scripted Automation

Scripted automation allows us to use PyMOL's fairly rich command language to systematically load and display structures. As an example, some organizations use PyMOL scripting to automate the creation of PyMOL sessions for storage and distribution of molecular data.

PyMOL scripts are plain text scripts. You can use programs like Microsoft Notepad (Windows), TextEdit (Mac), or Emacs (Linux) to create and save your scripts. The scripts *must be stored as plain text* to be used with PyMOL.

There are two types of script that PyMOL can read and execute. The first is the **PyMOL Script**. These end in ".pml" and are run from the PyMOL command line by typing, `@my_script.pml`. The command syntax in a PyMOL script is the same as what you would type on the command line. For example, to show the ligand as sticks, we would enter, `as sticks, organic`.

The next type of script is a **Python Program** script. The filename for these scripts ends in ".py" and we run a Python program script by typing, `run my_script.py`. Python program scripts use the PyMOL API and looks slightly different. For example, to show the ligand as sticks, we would type, `cmd.show_as("sticks", "organic")`. All PyMOL commands have an API equivalent. Usually converting to the PyMOL cmd API is simple. For example, if the command you would normally type is `some_command` and it takes two arguments `arg1` and `arg2`, which are strings, then the conversion is, `cmd.some_command("arg1", "arg2")`.

Let’s take a look at some frequently encountered tasks in PyMOL that we can perform using scripting. Each script discussed here can be found in the **PyMOLTutorials/IntermediatePyMOL/scripts** directory.

Creating a Custom Preset View

Let’s start scripting by creating a small script that we can apply to a newly loaded protein. It will automatically setup the view to focus on the first ligand in a PDB. We will further customize the view by adding hydrogen bonds to neighbors, setting colors, and labels. In NotePad or equivalent, please load the script **PyMOLTutorials/IntermediatePyMOL/scripts/custom_view.py**. The source code for this script is located in Code CP1. Let's take a moment to discuss the commands that we haven't yet seen.

Code CP1. PyMOL Script to Auto-generate a Custom View (custom_view.py).

```
    # create a named selection for the ligand
 1  select ligand, org

    # show the ligand as sticks
 2  as sticks, ligand

    # orient the scene on the complete first small organic molecule
 3  orient bm. first ligand

    # create a named selection for all atoms with 5 Ang. of the ligand selection
 4  select pocket, poly within 5 of ligand

    # show the pocket selection only as surface
 4  as surface, pocket

    # find polar contacts from the pocket to the ligand
 5  distance polar_contacts, poly, ligand, mode=2

    # set all surfaces to draw as light grey
 6  set surface_color, grey70

    # make surfaces transparent for polymer atoms
 7  set transparency, 0.5, poly

    # turn on valences for the ligand
 8  set_bond valence, 1, ligand

    # position the labels 3.5 Ang. closer to the camera to un-obscure them
 9  set label_position, [0, 0, 3.5]

    # label the ligand with its name
10  label first ligand, "Molecule: %s" % (resn)
```

Line #3, `orient bm. first ligand`, is the `orient` command operating on the `bm. first ligand` selection. The text, `first ligand`, selects the first atom in the ligand. The text `bm.` is modifier that stands for "by molecule." It takes the selection that follows it, here, `first ligand`, and selects all atoms in the molecular object. We say it **completes the molecule** for the selection.

Line #5, `distance polar_contacts, poly, ligand mode=2`, creates polar contacts from the polymer atoms to the ligand. The code `mode=2` tells PyMOL that we're only hunting for polar contacts, not all distances.

Lines #8 and #9 use the `set_bond` function. `Set_bond` acts analogously to the `set` command, except that it only handles bond-specific settings. Please type help `get_bond` for more information on this command.

Line #10 just labels the first atom in the ligand with the text "Molecule: %s." PyMOL will place the %s with the residue name (`resn`) field from the PDB. Here, PyMOL takes a trick from Python. Any "%s" within the label will be substituted with the actual value in the list that follows the string.

To use this script, load any molecule that has a ligand and type: `@ custom_view.py`.

Creating Your Own Functions

Basic Python programming experience is required for this section.

We've seen how useful scripted automation can be. It can save us time and save us from having to retype many commands. PyMOL can go one step further. Instead of running the script each time we want to use it, we can create a Python function and directly call that function from within PyMOL. It's simple and only requires two steps:

1. Write your code as a function in a Python program script
    - make sure you use the `extend` command to expose your function to PyMOL
2. Run the Python script *once*, using the `run` command.

After that, you can use the function just as if it had been built into PyMOL.

As an example of this, we'll look at a script that calculates the center of geometry (COG) of user-defined selections. After typing, `run COG.py`, we can simply run the script on any selection by typing: `COG your-selection`. For example, `COG organic`, to get the center of geometry for the organic small molecule. The commented code that achieves this is in Code Listing F1.

Line #2 imports PyMOL's cmd module. PyMOL stores all of its commands here. Importing this module gives us access to those commands.

Line #4 imports a special math library that ships with PyMOL. We'll use this library to sum up the atomic coordinates and determine the center of geometry.

Line #6 creates the new function called COG. COG takes two parameters, selection and quiet. Selection defaults to "all" which is the selection of all atoms. Quiet defaults to 0, which means PyMOL will print out the coordinates of the COG each time the command is run.

Line #8 gets the selected atoms as a chempy model. Chempy is the Python-based chemistry library that ships with PyMOL.

Line #10 counts the number of atoms in the selection. Line #12 initializes the object that will hold our summed coordinates.

Line #14 loops over all atoms in the selection and (line #15) adds that atom's coordinates to the summed coordinates. In Line #17, we divide by the number of atoms, yielding the center of geometry.

Lines #19 and #20 print out the value of the COG if the user wanted to see it.

Line #24 returns the COG as a list to the user.

Code F1. PyMOL Script to Auto-generate a Custom View (custom_view.py).

```
 1  # import the cmd module, all PyMOL-related Python programs should start with this
 2  from pymol import cmd
 3  # import a math library from chempy
 4  from chempy import cpv
 5  #define the new function
 6  def COG(selection='all', quiet=0):
 7      # get the chempy model
 8      model = cmd.get_model(selection)
 9      # count the number of atoms
10      nAtom = len(model.atom)
11      # create the list to store the center of mass
12      COM = cpv.get_null()
13      # loop over each atom and sum up the X, Y, and Z coordinates
14      for a in model.atom:
15          COM = cpv.add(COM, a.coord)
16      # divide by the number of atoms
17      COM = cpv.scale(COM, 1./nAtom)
18      # print the results if the user wants to see them
19      if not int(quiet):
20          print ' COG: [%8.3f,%8.3f,%8.3f]' % tuple(COM)
21      # return the center of geometry to the user
22      return COM
23  # expose the "COG" function to PyMOL's namespace
24  cmd.extend("COG", COG)
```

Now, for the session we have loaded, if I type, `COG org`, which gets the center of geometry for the organic small molecule, PyMOL prints:

```
PyMOL>COM org
 COM: [   9.964,  18.459,  -5.100]
```

`COG` returns its value in the last line. We can use that value; we could put a marker atom right at that position identifying the center of geometry. To do that, we type: `cog = COG("org")`. This runs the COG command and returns the value into the variable, `cog`. Next, we type `cmd.pseudoatom("cog_org", pos=cog)`. PyMOL draws a non-bonded atom at the specified location.

Next Steps

Other Courses

Congratulations on finishing the Intermediate PyMOL tutorial. We hope you enjoyed the lesson and are motivated to use PyMOL for your molecular visualization, movie-making, scripting, and editing tasks. This isn't the end, though: Schrödinger has other courses available covering Moviemaking, and Molecular Editing and Cleanup. Please contact us for more information on those courses via help@schrodinger.com.

Joining the PyMOL–Users Mailing List

As of Summer 2011, well over 1,500 PyMOL users subscribe to the pymol–users mailing list, which is where the community exchanges tips on how to use the software effectively and how to solve any problems that come up. To join the mailing list, please click on the **Mailing List** link that appears at the top of the PyMOL Home Page, (`http://www.pymol.org`). Or, to quickly search the mailing list archives for posts on a specific topic, click the **Mailing List Archive** link.

Visiting the PyMOLWiki Community Web Site

The community also runs a "Wiki" dynamic content site that aggregates information from the mailing list and provides other kinds of PyMOL–related documentation (`http://www.pymolwiki.org`). PyMOLWiki users can create their own pymol–related pages, on this site or elaborate upon useful information that is already posted. The site boasts over 1,300 users, over 7,000,000 page views, is typically accessed between 4,000–6,000 times a day, and has many useful scripts, plugins and tutorials free for download.

Accessing the Official Documentation Site

PyMOL subscribers access Schrödinger's Official Documentation site (`http://pymol.org/dsc`) using the subscription credentials shown from their most recent PyMOL receipt. This site contains the latest chapters from an updated PyMOL Manual, an assortment of narrated ScreenCasts, and plenty of reference information to help users on their way becoming PyMOL experts.

Indices

Index A: Selection Operators

Single Word	Abbreviated Form	Description
`not s1`	`! s1`	Atoms not in s1
`s1 and s2`	`s1 & s2`	Atoms in both s1 and s2
`s1 or s2`	`s1 | s2`	Atoms in s1, s2 or both
`s1 in s2`		Atoms whose identifiers name, resi, resn, chain and segi **all** match
`s1 like s2`	`s1 l. s2`	Atoms in s1 whose identifiers name and resi match atoms in s2
`s1 gap X`		Atoms in s1 whose van der Waals radii are separated from the van der Waals radii of s2 by at least X Angstroms
`s1 around X`	`s1 a. X`	Atoms with centers within X Angstroms of any atom in s1
`s1 expand X`	`s1 x. X`	Expands s1 to all atom within X Angstroms of any atom in s1
`s1 within X of s2`	`s1 w. X of s1`	Atoms in s1 within X Angstroms of s1
`s1 near_to X of s2`	`s1 nto. X of s2`	Atoms in s1 within X Angstroms of s1, *excluding* s2
`s1 beyond X of s2`	`s1 be. X of s2`	Atoms in s1 at least X Angstroms from s2
`byres s1`	`br. s1`	Expands s1 to complete residues
`bymolecule s1`	`bm. s1`	Expands s1 to complete molecule
`byfragment s1`	`bf. s1`	Expands s1 to complete fragment
`bysegment s1`	`bs. s1`	Expands s1 to complete segment
`byobject s1`	`bo. s1`	Expands s1 to complete PyMOL object
`bycell s1`	`bc. s1`	Expands s1 to complete unit cell
`neighbor s1`	`nbr. s1`	Atoms directly bonded to s1
`s1 extend X`	`s1 xt. X`	Extends s1 by X bonds to s1

Single Word	Abbreviated Form	Description
`pepseq SEQ`	`ps. SEQ`	Selects the peptide sequence SEQ

Index B: Single Word Selectors

Single Word	Abbreviated Form	Description
`all`	`all`	All atoms
`none`	`none`	No atoms (empty selection)
`solvent`	`sol`	waters
`organic`	`org`	atoms in non-polymer organic compounds
`inorganic`	`ino`	non-polymer inorganic atoms and ions
`polymer`	`poly`	atoms in a polymer
`hydro`	`h.`	hydrogens
`hetatm`	`het`	heteroatoms
`visible`	`v.`	visible atoms
`present`	`pr.`	atoms with coordinates in the current state
`enabled`		atoms in enabled objects
`masked`	`msk.`	masked atoms
`protected`	`prt.`	protected atoms
`bonded`		bonded atoms
`donors`	`don.`	hydrogen bond donors
`acceptors`	`acc.`	hydrogen bond acceptors
`fixed`	`fxd.`	fixed (unmovable) atoms
`restrained`	`rst.`	harmonically restrained atoms

Index C: Property Selectors

Single Word	Abbreviated Form	Description
`symbol`	`e.`	Elemental symbol
`name`	`n.`	Elemental name
`resn`	`r.`	Residue name
`resi`	`i.`	Residue number (identifier)
`alt`	`alt`	Alternate coordinates
`chain`	`c.`	Chain
`segi`	`s.`	Segment identifier
`flag`	`f.`	Flag number
`numeric_type`	`nt.`	Numeric representation of atom type
`text_type`	`tt.`	Text representation of atom type
`id`	`id`	External index number
`index`	`idx.`	Internal index number
`ss`	`ss`	Secondary structure type
`b`	`b`	B-factor value
`q`	`q`	Occupancy value
`formal_charge`	`fc.`	Formal atomic charge
`partial_charge`	`pc.`	Partial atomic charge