:mod:`pdb` --- The Python Debugger

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 3); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 6)

Unknown directive type "module".

```
.. module:: pdb
   :synopsis: The Python debugger for interactive interpreters.
```

Source code: :source:`Lib/pdb.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 9); *backlink*

Unknown interpreted text role "source".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 11)

Unknown directive type "index".

```
.. index:: single: debugging
```

The module :mod:`pdb` defines an interactive source code debugger for Python programs. It supports setting (conditional) breakpoints and single stepping at the source line level, inspection of stack frames, source code listing, and evaluation of arbitrary Python code in the context of any stack frame. It also supports post-mortem debugging and can be called under program control.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 15); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 21)

Unknown directive type "index".

```
.. index::
   single: Pdb (class in pdb)
   module: bdb
   module: cmd
```

The debugger is extensible -- it is actually defined as the class :class:`Pdb`. This is currently undocumented but easily understood by reading the source. The extension interface uses the modules :mod:`bdb` and :mod:`cmd`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 26); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 26); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 26); *backlink*

Unknown interpreted text role "mod".

The debugger's prompt is `(Pdb)`. Typical usage to run a program under control of the debugger is:

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
> <string>(0)?()
(Pdb) continue
> <string>(1)?()
(Pdb) continue
NameError: 'spam'
> <string>(1)?()
(Pdb)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 44)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   Tab-completion via the :mod:`readline` module is available for commands and
   command arguments, e.g. the current global and local names are offered as
   arguments of the ``p`` command.
```

:file:`pdb.py` can also be invoked as a script to debug other scripts. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 49); *backlink*

Unknown interpreted text role "file".

```
python3 -m pdb myscript.py
```

When invoked as a script, pdb will automatically enter post-mortem debugging if the program being debugged exits abnormally. After post-mortem debugging (or after normal exit of the program), pdb will restart the program. Automatic restarting preserves pdb's state (such as breakpoints) and in most cases is more useful than quitting the debugger upon program's exit.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 60)

Unknown directive type "versionadded".

```
.. versionadded:: 3.2
   :file:`pdb.py` now accepts a ``-c`` option that executes commands as if given
   in a :file:`.pdbrc` file, see :ref:`debugger-commands`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 64)

Unknown directive type "versionadded".

```
.. versionadded:: 3.7
   :file:`pdb.py` now accepts a ``-m`` option that execute modules similar to the way
   ``python3 -m`` does. As with a script, the debugger will pause execution just
   before the first line of the module.
```

The typical usage to break into the debugger is to insert:

```
import pdb; pdb.set_trace()
```

at the location you want to break into the debugger, and then run the program. You can then step through the code following this statement, and continue running without the debugger using the :pdbcmd:`continue` command.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 74); *backlink*

Unknown interpreted text role "pdbcmd".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 78)

Unknown directive type "versionadded".

```
.. versionadded:: 3.7
   The built-in :func:`breakpoint()`, when called with defaults, can be used
   instead of ``import pdb; pdb.set_trace()``.
```

The typical usage to inspect a crashed program is:

```
>>> import pdb
>>> import mymodule
>>> mymodule.test()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "./mymodule.py", line 4, in test
    test2()
  File "./mymodule.py", line 3, in test2
    print(spam)
NameError: spam
>>> pdb.pm()
> ./mymodule.py(3)test2()
-> print(spam)
(Pdb)
```

The module defines the following functions; each enters the debugger in a slightly different way:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 103)

Unknown directive type "function".

```
.. function:: run(statement, globals=None, locals=None)

   Execute the *statement* (given as a string or a code object) under debugger
   control.  The debugger prompt appears before any code is executed; you can
   set breakpoints and type :pdbcmd:`continue`, or you can step through the
   statement using :pdbcmd:`step` or :pdbcmd:`next` (all these commands are
   explained below).  The optional *globals* and *locals* arguments specify the
   environment in which the code is executed; by default the dictionary of the
   module :mod:`__main__` is used.  (See the explanation of the built-in
   :func:`exec` or :func:`eval` functions.)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 115)

Unknown directive type "function".

```
.. function:: runeval(expression, globals=None, locals=None)

   Evaluate the *expression* (given as a string or a code object) under debugger
   control.  When :func:`runeval` returns, it returns the value of the
   expression.  Otherwise this function is similar to :func:`run`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 122)

Unknown directive type "function".

```
.. function:: runcall(function, *args, **kwds)

   Call the *function* (a function or method object, not a string) with the
   given arguments.  When :func:`runcall` returns, it returns whatever the
   function call returned.  The debugger prompt appears as soon as the function
   is entered.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 130)

Unknown directive type "function".

```
.. function:: set_trace(*, header=None)

   Enter the debugger at the calling stack frame.  This is useful to hard-code
   a breakpoint at a given point in a program, even if the code is not
   otherwise being debugged (e.g. when an assertion fails).  If given,
   *header* is printed to the console just before debugging begins.

   .. versionchanged:: 3.7
      The keyword-only argument *header*.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 141)

Unknown directive type "function".

```
.. function:: post_mortem(traceback=None)

   Enter post-mortem debugging of the given *traceback* object.  If no
   *traceback* is given, it uses the one of the exception that is currently
   being handled (an exception must be being handled if the default is to be
   used).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 149)

Unknown directive type "function".

```
.. function:: pm()

   Enter post-mortem debugging of the traceback found in
   :data:`sys.last_traceback`.
```

The `run*` functions and :func:`set_trace` are aliases for instantiating the :class:`Pdb` class and calling the method of the same name. If you want to access further features, you have to do this yourself:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 155); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 155); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 159)

Invalid class attribute value for "class" directive: "Pdb(completekey='tab', stdin=None, stdout=None, skip=None, \
nosigint=False, readrc=True)".

```
.. class:: Pdb(completekey='tab', stdin=None, stdout=None, skip=None, \
               nosigint=False, readrc=True)

   :class:`Pdb` is the debugger class.

   The *completekey*, *stdin* and *stdout* arguments are passed to the
   underlying :class:`cmd.Cmd` class; see the description there.

   The *skip* argument, if given, must be an iterable of glob-style module name
   patterns.  The debugger will not step into frames that originate in a module
   that matches one of these patterns. [1]_

   By default, Pdb sets a handler for the SIGINT signal (which is sent when the
   user presses :kbd:`Ctrl-C` on the console) when you give a ``continue`` command.
   This allows you to break into the debugger again by pressing :kbd:`Ctrl-C`.  If you
   want Pdb not to touch the SIGINT handler, set *nosigint* to true.
```

```
The *readrc* argument defaults to true and controls whether Pdb will load
.pdbrc files from the filesystem.

Example call to enable tracing with *skip*::

   import pdb; pdb.Pdb(skip=['django.*']).set_trace()

.. audit-event:: pdb.Pdb "" pdb.Pdb

.. versionadded:: 3.1
   The *skip* argument.

.. versionadded:: 3.2
   The *nosigint* argument.  Previously, a SIGINT handler was never set by
   Pdb.

.. versionchanged:: 3.6
   The *readrc* argument.

.. method:: run(statement, globals=None, locals=None)
            runeval(expression, globals=None, locals=None)
            runcall(function, *args, **kwds)
            set_trace()

   See the documentation for the functions explained above.
```

Debugger Commands

The commands recognized by the debugger are listed below. Most commands can be abbreviated to one or two letters as indicated; e.g. `h(elp)` means that either `h` or `help` can be used to enter the help command (but not `he` or `hel`, nor `H` or `Help` or `HELP`). Arguments to commands must be separated by whitespace (spaces or tabs). Optional arguments are enclosed in square brackets (`[]`) in the command syntax; the square brackets must not be typed. Alternatives in the command syntax are separated by a vertical bar (`|`).

Entering a blank line repeats the last command entered. Exception: if the last command was a :pdbcmd:`list` command, the next 11 lines are listed.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 217); *backlink*

Unknown interpreted text role "pdbcmd".

Commands that the debugger doesn't recognize are assumed to be Python statements and are executed in the context of the program being debugged. Python statements can also be prefixed with an exclamation point (`!`). This is a powerful way to inspect the program being debugged; it is even possible to change a variable or call a function. When an exception occurs in such a statement, the exception name is printed but the debugger's state is not changed.

The debugger supports :ref:`aliases <debugger-aliases>`. Aliases can have parameters which allows one a certain level of adaptability to the context under examination.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 228); *backlink*

Unknown interpreted text role "ref".

Multiple commands may be entered on a single line, separated by `;;`. (A single `;` is not used as it is the separator for multiple commands in a line that is passed to the Python parser.) No intelligence is applied to separating the commands; the input is split at the first `;;` pair, even if it is in the middle of a quoted string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 238)

Unknown directive type "index".

```
.. index::
   pair: .pdbrc; file
   triple: debugger; configuration; file
```

If a file :file:`.pdbrc` exists in the user's home directory or in the current directory, it is read with `'utf-8'` encoding and executed as if

it had been typed at the debugger prompt. This is particularly useful for aliases. If both files exist, the one in the home directory is read first and aliases defined there can be overridden by the local file.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 242); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 248)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.11
   :file:`.pdbrc` is now read with ``'utf-8'`` encoding. Previously, it was read
   with the system locale encoding.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 252)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.2
   :file:`.pdbrc` can now contain commands that continue debugging, such as
   :pdbcmd:`continue` or :pdbcmd:`next`.  Previously, these commands had no
   effect.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 258)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: h(elp) [command]

   Without argument, print the list of available commands.  With a *command* as
   argument, print help about that command.  ``help pdb`` displays the full
   documentation (the docstring of the :mod:`pdb` module).  Since the *command*
   argument must be an identifier, ``help exec`` must be entered to get help on
   the ``!`` command.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 266)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: w(here)

   Print a stack trace, with the most recent frame at the bottom.  An arrow
   indicates the current frame, which determines the context of most commands.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 271)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: d(own) [count]

   Move the current frame *count* (default one) levels down in the stack trace
   (to a newer frame).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 276)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: u(p) [count]

   Move the current frame *count* (default one) levels up in the stack trace (to
```

```
   an older frame).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 281)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: b(reak) [([filename:]lineno | function) [, condition]]

   With a *lineno* argument, set a break there in the current file.  With a
   *function* argument, set a break at the first executable statement within
   that function.  The line number may be prefixed with a filename and a colon,
   to specify a breakpoint in another file (probably one that hasn't been loaded
   yet).  The file is searched on :data:`sys.path`.  Note that each breakpoint
   is assigned a number to which all the other breakpoint commands refer.

   If a second argument is present, it is an expression which must evaluate to
   true before the breakpoint is honored.

   Without argument, list all breaks, including for each breakpoint, the number
   of times that breakpoint has been hit, the current ignore count, and the
   associated condition if any.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 297)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: tbreak [([filename:]lineno | function) [, condition]]

   Temporary breakpoint, which is removed automatically when it is first hit.
   The arguments are the same as for :pdbcmd:`break`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 302)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: cl(ear) [filename:lineno | bpnumber ...]

   With a *filename:lineno* argument, clear all the breakpoints at this line.
   With a space separated list of breakpoint numbers, clear those breakpoints.
   Without argument, clear all breaks (but first ask confirmation).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 308)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: disable [bpnumber ...]

   Disable the breakpoints given as a space separated list of breakpoint
   numbers.  Disabling a breakpoint means it cannot cause the program to stop
   execution, but unlike clearing a breakpoint, it remains in the list of
   breakpoints and can be (re-)enabled.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 315)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: enable [bpnumber ...]

   Enable the breakpoints specified.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 319)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: ignore bpnumber [count]

   Set the ignore count for the given breakpoint number.  If count is omitted,
   the ignore count is set to 0.  A breakpoint becomes active when the ignore
   count is zero.  When non-zero, the count is decremented each time the
   breakpoint is reached and the breakpoint is not disabled and any associated
   condition evaluates to true.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 327)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: condition bpnumber [condition]

   Set a new *condition* for the breakpoint, an expression which must evaluate
   to true before the breakpoint is honored.  If *condition* is absent, any
   existing condition is removed; i.e., the breakpoint is made unconditional.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 333)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: commands [bpnumber]

   Specify a list of commands for breakpoint number *bpnumber*.  The commands
   themselves appear on the following lines.  Type a line containing just
   ``end`` to terminate the commands. An example::

      (Pdb) commands 1
      (com) p some_variable
      (com) end
      (Pdb)

   To remove all commands from a breakpoint, type ``commands`` and follow it
   immediately with ``end``; that is, give no commands.

   With no *bpnumber* argument, ``commands`` refers to the last breakpoint set.

   You can use breakpoint commands to start your program up again.  Simply use
   the :pdbcmd:`continue` command, or :pdbcmd:`step`,
   or any other command that resumes execution.

   Specifying any command resuming execution
   (currently :pdbcmd:`continue`, :pdbcmd:`step`, :pdbcmd:`next`,
   :pdbcmd:`return`, :pdbcmd:`jump`, :pdbcmd:`quit` and their abbreviations)
   terminates the command list (as if
   that command was immediately followed by end). This is because any time you
   resume execution (even with a simple next or step), you may encounter another
   breakpointâ€”which could have its own command list, leading to ambiguities about
   which list to execute.

   If you use the 'silent' command in the command list, the usual message about
   stopping at a breakpoint is not printed.  This may be desirable for breakpoints
   that are to print a specific message and then continue.  If none of the other
   commands print anything, you see no sign that the breakpoint was reached.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 367)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: s(tep)

   Execute the current line, stop at the first possible occasion (either in a
   function that is called or on the next line in the current function).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 372)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: n(ext)

   Continue execution until the next line in the current function is reached or
   it returns.  (The difference between :pdbcmd:`next` and :pdbcmd:`step` is
   that :pdbcmd:`step` stops inside a called function, while :pdbcmd:`next`
   executes called functions at (nearly) full speed, only stopping at the next
   line in the current function.)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 380)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: unt(il) [lineno]

   Without argument, continue execution until the line with a number greater
   than the current one is reached.

   With a line number, continue execution until a line with a number greater or
   equal to that is reached.  In both cases, also stop when the current frame
   returns.

   .. versionchanged:: 3.2
      Allow giving an explicit line number.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 392)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: r(eturn)

   Continue execution until the current function returns.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 396)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: c(ont(inue))

   Continue execution, only stop when a breakpoint is encountered.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 400)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: j(ump) lineno

   Set the next line that will be executed.  Only available in the bottom-most
   frame.  This lets you jump back and execute code again, or jump forward to
   skip code that you don't want to run.

   It should be noted that not all jumps are allowed -- for instance it is not
   possible to jump into the middle of a :keyword:`for` loop or out of a
   :keyword:`finally` clause.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 410)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: l(ist) [first[, last]]

   List source code for the current file.  Without arguments, list 11 lines
   around the current line or continue the previous listing.  With ``.`` as
   argument, list 11 lines around the current line.  With one argument,
   list 11 lines around at that line.  With two arguments, list the given range;
   if the second argument is less than the first, it is interpreted as a count.
```

```
   The current line in the current frame is indicated by ``->``.  If an
   exception is being debugged, the line where the exception was originally
   raised or propagated is indicated by ``>>``, if it differs from the current
   line.

   .. versionadded:: 3.2
      The ``>>`` marker.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 426)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: ll | longlist

   List all source code for the current function or frame.  Interesting lines
   are marked as for :pdbcmd:`list`.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 433)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: a(rgs)

   Print the argument list of the current function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 437)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: p expression

   Evaluate the *expression* in the current context and print its value.

   .. note::

      ``print()`` can also be used, but is not a debugger command --- this executes the
      Python :func:`print` function.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 447)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: pp expression

   Like the :pdbcmd:`p` command, except the value of the expression is
   pretty-printed using the :mod:`pprint` module.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 452)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: whatis expression

   Print the type of the *expression*.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 456)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: source expression
```

```
   Try to get source code for the given object and display it.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 462)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: display [expression]

   Display the value of the expression if it changed, each time execution stops
   in the current frame.

   Without expression, list all display expressions for the current frame.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 471)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: undisplay [expression]

   Do not display the expression any more in the current frame.  Without
   expression, clear all display expressions for the current frame.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 478)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: interact

   Start an interactive interpreter (using the :mod:`code` module) whose global
   namespace contains all the (global and local) names found in the current
   scope.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 488)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: alias [name [command]]

   Create an alias called *name* that executes *command*.  The command must
   *not* be enclosed in quotes.  Replaceable parameters can be indicated by
   ``%1``, ``%2``, and so on, while ``%*`` is replaced by all the parameters.
   If no command is given, the current alias for *name* is shown. If no
   arguments are given, all aliases are listed.

   Aliases may be nested and can contain anything that can be legally typed at
   the pdb prompt.  Note that internal pdb commands *can* be overridden by
   aliases.  Such a command is then hidden until the alias is removed.  Aliasing
   is recursively applied to the first word of the command line; all other words
   in the line are left alone.

   As an example, here are two useful aliases (especially when placed in the
   :file:`.pdbrc` file)::

      # Print instance variables (usage "pi classInst")
      alias pi for k in %1.__dict__.keys(): print("%1.",k,"=",%1.__dict__[k])
      # Print instance variables in self
      alias ps pi self
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 510)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: unalias name

   Delete the specified alias.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 514)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: ! statement

   Execute the (one-line) *statement* in the context of the current stack frame.
   The exclamation point can be omitted unless the first word of the statement
   resembles a debugger command.  To set a global variable, you can prefix the
   assignment command with a :keyword:`global` statement on the same line,
   e.g.::

      (Pdb) global list_options; list_options = ['-l']
      (Pdb)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 525)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: run [args ...]
                restart [args ...]

   Restart the debugged Python program.  If an argument is supplied, it is split
   with :mod:`shlex` and the result is used as the new :data:`sys.argv`.
   History, breakpoints, actions and debugger options are preserved.
   :pdbcmd:`restart` is an alias for :pdbcmd:`run`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 533)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: q(uit)

   Quit from the debugger.  The program being executed is aborted.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 537)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: debug code

   Enter a recursive debugger that steps through the code
   argument (which is an arbitrary expression or statement to be
   executed in the current environment).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pdb.rst, line 543)

Unknown directive type "pdbcommand".

```
.. pdbcommand:: retval

   Print the return value for the last return of a function.
```

Footnotes

[1] Whether a frame is considered to originate in a certain module is determined by the `__name__` in the frame globals.