NAME

python − an interpreted, interactive, object-oriented programming language

SYNOPSIS

python [**−B**] [**−b**] [**−d**] [**−E**] [**−h**] [**−i**] [**−I**]
[**−m** *module-name*] [**−q**] [**−O**] [**−OO**] [**−s**] [**−S**] [**−u**]
[**−v**] [**−V**] [**−W** *argument*] [**−x**] [[**−X** *option*] **−?**]
[**−-check-hash-based-pycs** *default* | *always* | *never*]
[**−c** *command* | *script* | −] [*arguments*]

DESCRIPTION

Python is an interpreted, interactive, object-oriented programming language that combines remarkable power with very clear syntax. For an introduction to programming in Python, see the Python Tutorial. The Python Library Reference documents built-in and standard types, constants, functions and modules. Finally, the Python Reference Manual describes the syntax and semantics of the core language in (perhaps too) much detail. (These documents may be located via the **INTERNET RESOURCES** below; they may be installed on your system as well.)

Python's basic power can be extended with your own modules written in C or C++. On most systems such modules may be dynamically loaded. Python is also adaptable as an extension language for existing applications. See the internal documentation for hints.

Documentation for installed Python modules and packages can be viewed by running the **pydoc** program.

COMMAND LINE OPTIONS

−B Don't write *.pyc* files on import. See also PYTHONDONTWRITEBYTECODE.

−b Issue warnings about str(bytes_instance), str(bytearray_instance) and comparing bytes/bytearray with str. (-bb: issue errors)

−c *command*
Specify the command to execute (see next section). This terminates the option list (following options are passed as arguments to the command).

−−check-hash-based-pycs *mode*
Configure how Python evaluates the up-to-dateness of hash-based .pyc files.

−d Turn on parser debugging output (for expert only, depending on compilation options).

−E Ignore environment variables like PYTHONPATH and PYTHONHOME that modify the behavior of the interpreter.

−h , −? , −−help
Prints the usage for the interpreter executable and exits.

−i When a script is passed as first argument or the **−c** option is used, enter interactive mode after executing the script or the command. It does not read the $PYTHONSTARTUP file. This can be useful to inspect global variables or a stack trace when a script raises an exception.

−I Run Python in isolated mode. This also implies **−E** and **−s**. In isolated mode sys.path contains neither the script's directory nor the user's site-packages directory. All PYTHON* environment variables are ignored, too. Further restrictions may be imposed to prevent the user from injecting malicious code.

−m *module-name*
Searches *sys.path* for the named module and runs the corresponding *.py* file as a script.

−O Remove assert statements and any code conditional on the value of __debug__; augment the filename for compiled (bytecode) files by adding .opt-1 before the .pyc extension.

−OO Do **-O** and also discard docstrings; change the filename for compiled (bytecode) files by adding .opt-2 before the .pyc extension.

−q Do not print the version and copyright messages. These messages are also suppressed in non-interactive mode.

−s Don't add user site directory to sys.path.

−S Disable the import of the module *site* and the site-dependent manipulations of *sys.path* that it entails. Also disable these manipulations if *site* is explicitly imported later.

−u Force the stdout and stderr streams to be unbuffered. This option has no effect on the stdin stream.

−v Print a message each time a module is initialized, showing the place (filename or built-in module) from which it is loaded. When given twice, print a message for each file that is checked for when searching for a module. Also provides information on module cleanup at exit.

−V , −−version
Prints the Python version number of the executable and exits. When given twice, print more information about the build.

−W *argument*
Warning control. Python sometimes prints warning message to *sys.stderr*. A typical warning message has the following form: *file***:***line***:** *category***:** *message.* By default, each warning is printed once for each source line where it occurs. This option controls how often warnings are printed. Multiple **−W** options may be given; when a warning matches more than one option, the action for the last matching option is performed. Invalid **−W** options are ignored (a warning message is printed about invalid options when the first warning is issued). Warnings can also be controlled from within a Python program using the *warnings* module.

The simplest form of *argument* is one of the following *action* strings (or a unique abbreviation): **ignore** to ignore all warnings; **default** to explicitly request the default behavior (printing each warning once per source line); **all** to print a warning each time it occurs (this may generate many messages if a warning is triggered repeatedly for the same source line, such as inside a loop); **module** to print each warning only the first time it occurs in each module; **once** to print each warning only the first time it occurs in the program; or **error** to raise an exception instead of printing a warning message.

The full form of *argument* is *action***:***message***:***category***:***module***:***line.* Here, *action* is as explained above but only applies to messages that match the remaining fields. Empty fields match all values; trailing empty fields may be omitted. The *message* field matches the start of the warning message printed; this match is case-insensitive. The *category* field matches the warning category. This must be a class name; the match test whether the actual warning category of the message is a subclass of the specified warning category. The full class name must be given. The *module* field matches the (fully-qualified) module name; this match is case-sensitive. The *line* field matches the line number, where zero matches all line numbers and is thus equivalent to an omitted line number.

−X *option*
Set implementation specific option. The following options are available:

-X faulthandler: enable faulthandler

-X showrefcount: output the total reference count and number of used
memory blocks when the program finishes or after each statement in the
interactive interpreter. This only works on debug builds

-X tracemalloc: start tracing Python memory allocations using the
tracemalloc module. By default, only the most recent frame is stored in a
traceback of a trace. Use -X tracemalloc=NFRAME to start tracing with a
traceback limit of NFRAME frames

-X showalloccount: output the total count of allocated objects for each
type when the program finishes. This only works when Python was built with
COUNT_ALLOCS defined

-X importtime: show how long each import takes. It shows module name, cumulative time (including nested imports) and self time (excluding nested imports). Note that its output may be broken in multi-threaded application. Typical usage is python3 -X importtime -c 'import asyncio'

-X dev: enable CPython's "development mode", introducing additional runtime checks which are too expensive to be enabled by default. It will not be more verbose than the default if the code is correct: new warnings are only emitted when an issue is detected. Effect of the developer mode:
* Add default warning filter, as -W default
* Install debug hooks on memory allocators: see the PyMem_SetupDebugHooks() C function
* Enable the faulthandler module to dump the Python traceback on a crash
* Enable asyncio debug mode
* Set the dev_mode attribute of sys.flags to True
* io.IOBase destructor logs close() exceptions

-X utf8: enable UTF-8 mode for operating system interfaces, overriding the default locale-aware mode. -X utf8=0 explicitly disables UTF-8 mode (even when it would otherwise activate automatically). See PYTHONUTF8 for more details

-X pycache_prefix=PATH: enable writing .pyc files to a parallel tree rooted at the given directory instead of to the code tree.

−x Skip the first line of the source. This is intended for a DOS specific hack only. Warning: the line numbers in error messages will be off by one!

INTERPRETER INTERFACE

The interpreter interface resembles that of the UNIX shell: when called with standard input connected to a tty device, it prompts for commands and executes them until an EOF is read; when called with a file name argument or with a file as standard input, it reads and executes a *script* from that file; when called with **−c** *command*, it executes the Python statement(s) given as *command*. Here *command* may contain multiple statements separated by newlines. Leading whitespace is significant in Python statements! In non-interactive mode, the entire input is parsed before it is executed.

If available, the script name and additional arguments thereafter are passed to the script in the Python variable *sys.argv*, which is a list of strings (you must first *import sys* to be able to access it). If no script name is given, *sys.argv[0]* is an empty string; if **−c** is used, *sys.argv[0]* contains the string *'-c'*. Note that options interpreted by the Python interpreter itself are not placed in *sys.argv*.

In interactive mode, the primary prompt is '>>>'; the second prompt (which appears when a command is not complete) is '...'. The prompts can be changed by assignment to *sys.ps1* or *sys.ps2*. The interpreter quits when it reads an EOF at a prompt. When an unhandled exception occurs, a stack trace is printed and control returns to the primary prompt; in non-interactive mode, the interpreter exits after printing the stack trace. The interrupt signal raises the *KeyboardInterrupt* exception; other UNIX signals are not caught (except that SIGPIPE is sometimes ignored, in favor of the *IOError* exception). Error messages are written to stderr.

FILES AND DIRECTORIES

These are subject to difference depending on local installation conventions; ${prefix} and ${exec_prefix} are installation-dependent and should be interpreted as for GNU software; they may be the same. On Debian GNU/{Hurd,Linux} the default for both is */usr*.

${exec_prefix}/bin/python
Recommended location of the interpreter.

${prefix}/lib/python<version>
${exec_prefix}/lib/python<version>
Recommended locations of the directories containing the standard modules.

${prefix}/include/python<version>
${exec_prefix}/include/python<version>
Recommended locations of the directories containing the include files needed for developing Python extensions and embedding the interpreter.

ENVIRONMENT VARIABLES

PYTHONHOME
Change the location of the standard Python libraries. By default, the libraries are searched in ${prefix}/lib/python<version> and ${exec_prefix}/lib/python<version>, where ${prefix} and ${exec_prefix} are installation-dependent directories, both defaulting to */usr/local*. When $PYTHONHOME is set to a single directory, its value replaces both ${prefix} and ${exec_prefix}. To specify different values for these, set $PYTHONHOME to ${prefix}:${exec_prefix}.

PYTHONPATH
Augments the default search path for module files. The format is the same as the shell's $PATH: one or more directory pathnames separated by colons. Non-existent directories are silently ignored. The default search path is installation dependent, but generally begins with ${prefix}/lib/python<version> (see PYTHONHOME above). The default search path is always appended to $PYTHONPATH. If a script argument is given, the directory containing the script is inserted in the path in front of $PYTHONPATH. The search path can be manipulated from within a Python program as the variable *sys.path*.

PYTHONSTARTUP
If this is the name of a readable file, the Python commands in that file are executed before the first prompt is displayed in interactive mode. The file is executed in the same name space where interactive commands are executed so that objects defined or imported in it can be used without qualification in the interactive session. You can also change the prompts *sys.ps1* and *sys.ps2* in this file.

PYTHONOPTIMIZE
If this is set to a non-empty string it is equivalent to specifying the **−O** option. If set to an integer, it is equivalent to specifying **−O** multiple times.

PYTHONDEBUG
If this is set to a non-empty string it is equivalent to specifying the **−d** option. If set to an integer, it is equivalent to specifying **−d** multiple times.

PYTHONDONTWRITEBYTECODE
If this is set to a non-empty string it is equivalent to specifying the **−B** option (don't try to write *.pyc* files).

PYTHONINSPECT
If this is set to a non-empty string it is equivalent to specifying the **−i** option.

PYTHONIOENCODING
If this is set before running the interpreter, it overrides the encoding used for stdin/stdout/stderr, in the syntax *encodingname***:***errorhandler* The *errorhandler* part is optional and has the same meaning as in str.encode. For stderr, the *errorhandler*
part is ignored; the handler will always be ´backslashreplace´.

PYTHONNOUSERSITE
If this is set to a non-empty string it is equivalent to specifying the **−s** option (Don't add the user site directory to sys.path).

PYTHONUNBUFFERED
If this is set to a non-empty string it is equivalent to specifying the **−u** option.

PYTHONVERBOSE
If this is set to a non-empty string it is equivalent to specifying the **−v** option. If set to an integer, it is equivalent to specifying **−v** multiple times.

PYTHONWARNINGS
: If this is set to a comma-separated string it is equivalent to specifying the **−W** option for each separate value.

PYTHONHASHSEED
: If this variable is set to "random", a random value is used to seed the hashes of str and bytes objects.

  If PYTHONHASHSEED is set to an integer value, it is used as a fixed seed for generating the hash() of the types covered by the hash randomization. Its purpose is to allow repeatable hashing, such as for selftests for the interpreter itself, or to allow a cluster of python processes to share hash values.

  The integer must be a decimal number in the range [0,4294967295]. Specifying the value 0 will disable hash randomization.

PYTHONMALLOC
: Set the Python memory allocators and/or install debug hooks. The available memory allocators are *malloc* and *pymalloc*. The available debug hooks are *debug*, *malloc_debug*, and *pymalloc_debug*.

  When Python is compiled in debug mode, the default is *pymalloc_debug* and the debug hooks are automatically used. Otherwise, the default is *pymalloc*.

PYTHONMALLOCSTATS
: If set to a non-empty string, Python will print statistics of the pymalloc memory allocator every time a new pymalloc object arena is created, and on shutdown.

  This variable is ignored if the $**PYTHONMALLOC** environment variable is used to force the **malloc**(3) allocator of the C library, or if Python is configured without pymalloc support.

PYTHONASYNCIODEBUG
: If this environment variable is set to a non-empty string, enable the debug mode of the asyncio module.

PYTHONTRACEMALLOC
: If this environment variable is set to a non-empty string, start tracing Python memory allocations using the tracemalloc module.

  The value of the variable is the maximum number of frames stored in a traceback of a trace. For example, *PYTHONTRACEMALLOC=1* stores only the most recent frame.

PYTHONFAULTHANDLER
: If this environment variable is set to a non-empty string, *faulthandler.enable()* is called at startup: install a handler for SIGSEGV, SIGFPE, SIGABRT, SIGBUS and SIGILL signals to dump the Python traceback.

  This is equivalent to the **-X faulthandler** option.

PYTHONEXECUTABLE
: If this environment variable is set, *sys.argv[0]* will be set to its value instead of the value got through the C runtime. Only works on Mac OS X.

PYTHONUSERBASE
: Defines the user base directory, which is used to compute the path of the user *site-packages* directory and Distutils installation paths for *python setup.py install −−user*.

PYTHONPROFILEIMPORTTIME
: If this environment variable is set to a non-empty string, Python will show how long each import takes. This is exactly equivalent to setting **−X importtime** on the command line.

PYTHONBREAKPOINT
: If this environment variable is set to 0, it disables the default debugger. It can be set to the callable of your debugger of choice.

Debug-mode variables

Setting these variables only has an effect in a debug build of Python, that is, if Python was configured with the **--with-pydebug** build option.

PYTHONTHREADDEBUG
: If this environment variable is set, Python will print threading debug info.

PYTHONDUMPREFS
: If this environment variable is set, Python will dump objects and reference counts still alive after shutting down the interpreter.

AUTHOR

The Python Software Foundation: https://www.python.org/psf/

INTERNET RESOURCES

Main website: https://www.python.org/
Documentation: https://docs.python.org/
Developer resources: https://devguide.python.org/
Downloads: https://www.python.org/downloads/
Module repository: https://pypi.org/
Newsgroups: comp.lang.python, comp.lang.python.announce

LICENSING