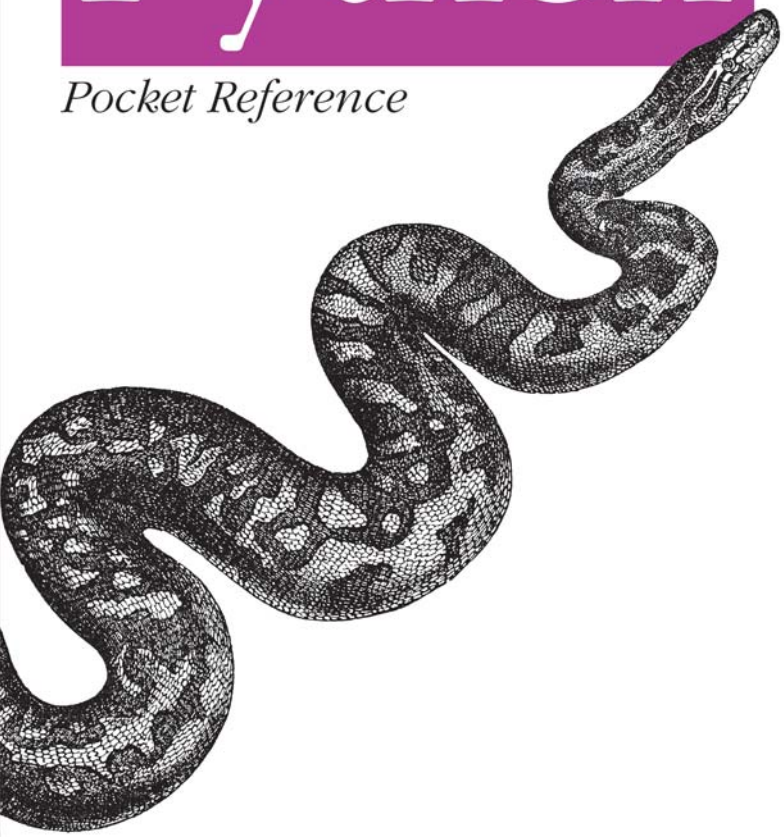
Python In Your Pocket

3rd Edition
Covers Python 2.4

Python

Pocket Reference



O'REILLY®

Mark Lutz

Python/Scripting Languages

O'REILLY®

Python Pocket Reference

A python is a muscular snake that kills its prey by squeezing it to the point of suffocation. But the *Python Pocket Reference* is not a book about snakes. This is a book about Python, the language, which is one of the most elegant, muscular, and widely used scripting languages available these days. With this book, you'll be the one doing the squeezing, as you apply the power of Python to boost the productivity of your programming work.

And speaking of squeezing, that's exactly what expert Python programmer Mark Lutz has done by condensing a ton of important reference information into the *Python Pocket Reference*. In this small but essential book, you'll find syntax for all Python statements, descriptions of built-in types and operators, call specifications for built-in functions and methods, and much, much more. Updated to cover the latest developments in Python 2.4, this is a handy and concise reference you won't want to be without.

Mark Lutz is a software developer, and a Python writer and trainer. He is also the author of *Programming Python* and coauthor of *Learning Python*, both from O'Reilly. Mark has programmed a variety of Python systems, teaches courses about the language, and has been involved with Python since 1992.

"The best little volume on Python that money can buy."

—Guido van Rossum, creator of Python

www.oreilly.com

ISBN 0-596-00940-2 **US $9.95** **CAN $13.95**

9 780596 009403 6 36920 00940 5

THIRD EDITION

Python
Pocket Reference

Mark Lutz

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Python Pocket Reference
by Mark Lutz

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor:	Jonathan Gennick
Production Editor:	Claire Cloutier
Cover Designer:	Edie Freedman
Interior Designer:	David Futato

Printing History:

October 1998:	First Edition.
January 2002:	Second Edition.
February 2005:	Third Edition.

0-596-00940-2
[C] [6/06]

Contents

Python Pocket Reference

Introduction

Python is a general-purpose, object-oriented, and open source computer programming language. It is commonly used for both standalone programs and scripting applications in a wide variety of domains, by hundreds of thousands of developers.

Python is designed to optimize developer productivity, software quality, program portability, and component integration. Python programs run on most platforms in common use, including mainframes and supercomputers, Unix and Linux, Windows and Macintosh, Palm OS and Pocket PC, Java and .NET, and more.

This pocket reference summarizes Python statements and types, built-in functions, commonly used library modules, and other prominent Python tools. It is intended to serve as a concise reference tool for developers and is designed to be a companion to other books that provide tutorials, code examples, and other learning materials.

This third edition covers Python Version 2.4 and later. It has been thoroughly updated for recent language and library changes and expanded for new topics. Most of it applies to earlier releases as well, with the exception of recent language extensions.

Conventions

The following conventions are used in this book:

`[]`
: Items in brackets are usually optional. The exceptions are those cases where brackets are part of Python's syntax.

`*`
: Something followed by an asterisk can be repeated zero or more times.

a | *b*
: Items separated by a bar are often alternatives.

Italic
: Used for filenames and URLs and to highlight new terms.

`Constant width`
: Used for code, commands, and command-line options, and to indicate the names of modules, functions, attributes, variables, and methods.

`Constant width italic`
: Used for replaceable parameter names in command syntax.

Command-Line Options

```
python [option*]
  [ scriptfilename | -c command | -m module | - ] [arg*]
```

Python Options

`-d`
: Turns on parser debugging output (for developers of the Python core).

`-E`
: Ignores environment variables (such as `PYTHONPATH`).

`-h`
: Prints help message and exit.

`-i`
: Enters interactive mode after executing a script, without reading the `PYTHONSTARTUP` file. Useful for postmortem debugging.

`-O`
: Optimizes generated byte-code (create and use *.pyo* byte-code files). Currently yields a minor performance improvement.

`-OO`
: Operates like `-O`, the previous option, but also removes docstrings from byte-code.

`-Q arg`
: Division options: `-Qold` (default), `-Qwarn`, `-Qwarnall`, `-Qnew`.

`-S`
: Doesn't imply "import site" on initialization.

`-t`
: Issues warnings about inconsistent tab usage (`-tt` issues error instead).

`-u`
: Forces *stdout* and *stderr* to be unbuffered and binary.

`-v`
: Prints a message each time a module is initialized, showing the place from which it is loaded; repeats this flag for more verbose output.

`-V`
: Prints Python version number and exit.

`-W` *`arg`*
: Functions as warning control; `arg` is *`action`*`:`*`message`*`:`*`category`*`:`*`module`*`:`*`lineno`*. See `warnings` module documentation in the Python Library Reference (*http://www.python.org/doc/*).

`-x`
: Skips first line of source, allowing use of non-Unix forms of `#!cmd`.

Program Specification

`scriptfilename`
: Denotes the name of a Python scriptfile to execute; the main, topmost file of a program, made available in `sys.argv[0]`.

`-c` *`command`*
: Specifies a Python command (as a string) to execute; `sys.argv[0]` is set to `-c`.

`-m` *`module`*
: Runs library module as a script: searches for module on `sys.path`, and runs it as a top-level file (e.g., `python -m profile` runs the Python profiler).

`-`
: Reads Python commands from *stdin* (the default); enters interactive mode if *stdin* is a tty (interactive device).

*`arg`**
: Indicates that anything else on the command line is passed to the scriptfile or command (and appears in the built-in list of strings `sys.argv[1:]`).

If no *`scriptfilename`*, *`command`*, or *`module`* is given, Python enters interactive mode, reading commands from *stdin* (and using GNU readline, if installed, for input).

Besides using traditional command lines, you can also generally start Python programs by clicking their filenames in a file

explorer GUI, by calling functions in the Python/C API, by using program launch menu options in IDEs such as IDLE and Komodo, and so on.

Environment Variables

`PYTHONPATH`
: Augments the default search path for imported module files. The format is the same as the shell's `PATH` setting: directory pathnames separated by colons (semicolons on DOS). On module imports, Python searches for the corresponding file or directory in each listed directory, from left to right. Merged into `sys.path`.

`PYTHONSTARTUP`
: If set to the name of a readable file, the Python commands in that file are executed before the first prompt is displayed in interactive mode.

`PYTHONHOME`
: If set, the value is used as an alternate prefix directory for library modules (or `sys.prefix`, `sys.exec_prefix`). The default module search path uses `sys.prefix/lib`.

`PYTHONCASEOK`
: If set, ignores case in import statements (on Windows).

`PYTHONDEBUG`
: If nonempty, same as `-d` option.

`PYTHONINSPECT`
: If nonempty, same as `-i` option.

`PYTHONOPTIMIZE`
: If nonempty, same as `-O` option.

`PYTHONUNBUFFERED`
: If nonempty, same as `-u` option.

`PYTHONVERBOSE`
: If nonempty, same as `-v` option.

Built-in Types and Operators

Operators and Precedence

Table 1 lists Python's expression operators. Operators in the lower cells of this table have higher precedence (i.e., bind tighter) when used in mixed-operator expressions without parentheses.

Table 1. Expression operators and precedence

Operator	Description
`lambda args: expr`	Anonymous function maker.
`X or Y`	Logical OR: Y is evaluated only if X is false.
`X and Y`	Logical AND: Y is evaluated only if X is true.
`not X`	Logical negation.
`X < Y, X <= Y, X > Y, X >= Y`	Comparison operators[a].
`X == Y, X <> Y, X != Y`	Equality operators.
`X is Y, X is not Y`	Object identity tests.
`X in S, X not in S`	Sequence membership.
`X | Y`	Bitwise OR.
`X ^ Y`	Bitwise exclusive OR.
`X & Y`	Bitwise AND.
`X << Y, X >> Y`	Shift X left, right by Y bits.
`X + Y, X - Y`	Addition/concatenation, subtraction.
`X * Y, X % Y, X / Y, X // Y`	Multiply/repetition, remainder/format, division, floor division[b].
`-X, +X, ~X, X ** Y`	Unary negation, identity, bitwise complement, power.
`X[i], X[i:j], X.attr, X(...)`	Indexing, slicing, attribute references, function calls.
`(...), [...], {...}, `...``	Tuple[c], list[d], dictionary, conversion to string[e].