Python 2.1 Quick Reference

Contents

Version 2.1.2
The latest version is to be found here.

7 Aug 2001 *upgraded by Simon Brunning for Python 2.1*
16 May 2001 *upgraded by Richard Gruet and Simon Brunning for Python 2.0*
2000/07/18 *upgraded by Richard Gruet, rgruet@intraware.com for Python 1.5.2 from V1.3 ref 1995/10/30, by Chris Hoffmann, choffman@vicorp.com*

NB: features added in 2.1 since 2.0 are coloured dark green.

NB: features added in 2.0 since 1.5.2 are coloured dark magenta.

Based on:
Python Bestiary, Author: Ken Manheimer, ken.manheimer@nist.gov
Python manuals, Authors: Guido van Rossum and Fred Drake
What's new in Python 2.0, Authors: A.M. Kuchling and Moshe Zadka
python-mode.el, Author: Tim Peters, tim_one@email.msn.com
and the readers of comp.lang.python
Python's nest: http://www.python.org
Development: http://python.sourceforge.net/

ActivePython : http://www.ActiveState.com/ASPN/Python/
newsgroup: comp.lang.python
Help desk: help@python.org
Resources: http://starship.python.net/ and http://www.vex.net/parnassus/
Full documentation: http://www.python.org/doc/
An excellent Python reference book: Python Essential Reference by David Beazley (New Riders)

Invocation Options

python [-diOStuUvxX?] [-c *command* | *script* | -] [*args*]

Invocation Options

Option	Effect
-d	Outputs parser debugging information (also PYTHONDEBUG=x)
-i	Inspect interactively after running script (also PYTHONINSPECT=x) and force prompts, even if stdin appears not to be a terminal
-O	Optimize generated bytecode (set __debug__ = 0 =>s suppresses *asserts*)
-S	Don't perform 'import site' on initialization
-t	Issue warnings about inconsistent tab usage (-tt: issue errors)
-u	Unbuffered binary stdout and stderr (also PYTHONUNBUFFERED=x).
-U	Force Python to interpret all string literals as Unicode literals.

-v	Verbose (trace import statements) (also PYTHONVERBOSE=x)
-x	Skip first line of source, allowing use of non-unix Forms of #!cmd
-X	Disable class based built-in exceptions (for backward compatibility management of exceptions)
-?	Help!
-c *command*	Specify the command to execute (see next section). This terminates the option list (following options are passed as arguments to the command).
script	the name of a python file (.py) to execute read from stdin. Anything afterward is passed as options to python script or command, not interpreted as an option to interpreter itself.
args	passed to script or command (in sys.argv[1:])
	If no script or command, Python enters interactive mode.

- Available IDEs in std distrib: **IDLE** (tkinter based, portable), Pythonwin (Windows).

Environment variables

Environment variables

Variable	Effect
PYTHONHOME	Alternate *prefix* directory (or *prefix*;*exec_prefix*). The default module search path uses *prefix*/lib
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 ':' or ';' without spaces around (semi-)colons! On Windows first search for Registry key *HKEY_LOCAL_MACHINE\Software\Python\PythonCore\x.y\PythonPath* (default value). You may also define a key named after your application with a default string value giving the root directory path of your app.
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 (no default).
PYTHONDEBUG	If non-empty, same as -d option
PYTHONINSPECT	If non-empty, same as -i option
PYTHONSUPPRESS	If non-empty, same as -s option
PYTHONUNBUFFERED	If non-empty, same as -u option
PYTHONVERBOSE	If non-empty, same as -v option
PYTHONCASEOK	If non-empty, ignore case in file/module names (imports)

Notable lexical entities

Keywords

```
and       del       for       is        raise
assert    elif      from      lambda    return
break     else      global    not       try
class     except    if        or        while
continue  exec      import    pass
def       finally   in        print
```

- (list of keywords in std module: **keyword**)
- Illegitimate Tokens (only valid in strings): @ $?
- A statement must all be on a single line. To break a statement over multiple lines use "\", as with the C preprocessor.
- Exception: can always break when inside any (), [], or {} pair, or in triple-quoted strings.
- More than one statement can appear on a line if they are separated with semicolons (";").
- Comments start with "#" and continue to end of line.

Identifiers

(*letter* | "_") (*letter* | *digit* | "_")*

- Python identifiers keywords, attributes, etc. are **case-sensitive**.
- Special forms: **_***ident* (not imported by 'from module import *'); **__***ident***__** (system defined name);
- **__***ident* (class-private name mangling)

String literals

Literal
"a string enclosed by double quotes"
'another string delimited by single quotes and with a " inside**'**
'''a string containing embedded newlines and quote (') marks, can be delimited with triple quotes.**'''**
""" may also use 3- double quotes as delimiters """
u'a unicode string**'**
U"Another unicode string"
r'a raw string where \ are kept (literalized): handy for regular expressions and windows paths!**'**
R"another raw string" -- raw strings cannot end with a \
ur'a unicode raw string**'**
UR"another raw unicode"

- Use \ at end of line to continue a string on next line.
- adjacent strings are concatened, e.g. 'Monty' ' Python' is the same as 'Monty Python'.
- u'hello' + ' world' --> u'hello world' (coerced to unicode)

String Literal Escapes

Escape	Meaning
newline	Ignored (escape newline)
\\	Backslash (\)
\e	Escape (ESC)
\v	Vertical Tab (VT)
\'	Single quote (')
\f	Formfeed (FF)
\OOO	char with octal value OOO
\"	Double quote (")
\n	Linefeed (LF)
\a	Bell (BEL)
\r	Carriage Return (CR)
\xHH	char with hex value HH
\b	Backspace (BS)
\t	Horizontal Tab (TAB)
\uHHHH	unicode char with hex value HHHH, can only be used in unicode string
\UHHHHHHHH	unicode char with hex value HHHHHHHH, can only be used in unicode string
AnyOtherChar	left as-is

- NUL byte (\000) is NOT an end-of-string marker; NULs may be embedded in strings.
- Strings (and tuples) are immutable: they cannot be modified.

Numbers

- **Decimal integer**: 1234, 1234567890546378940**L** (or **l)**
- **Octal** integer: **0**177, **0**17777777777777777L (begin with a **0**)
- **Hex** integer: **0x**FF, **0X**FFFFffffFFFFFFFFFFL (begin with **0x** or **0X**)
- **Long** integer (unlimited precision): 1234567890123456**L** (ends with **L** or **l**)
- **Float** (double precision): 3**.**14**e-10**, .001, 10., 1E3
- **Complex**: 1**J**, 2+3**J**, 4+5**j** (ends with **J** or **j, +** separates (float) real and imaginary parts)

Sequences

- **String** of length 0, 1, 2 (see above)
- '', '1', "12", 'hello\n'
- **Tuple** of length 0, 1, 2, etc:
- **()** (1,) (1,2) # parentheses are optional if len > 0
- **List** of length 0, 1, 2, etc:
- **[]** [1] [1,2]

Indexing is **0**-based. Negative indices (usually) mean count backwards from end of sequence.

Sequence **slicing [***starting-at-index* **:** *but-less-than-index***].** Start defaults to '0'; End defaults to 'sequence-length'.

a = (0,1,2,3,4,5,6,7)
a[3] ==> 3
a[-1] ==> 7
a[2:4] ==> (2, 3)
a[1:] ==> (1, 2, 3, 4, 5, 6, 7)
a[:3] ==> (0, 1, 2)
a[:] ==> (0,1,2,3,4,5,6,7) # makes a **copy** of the sequence.

Dictionaries (Mappings)

Dictionary of length 0, 1, 2, etc:
{} {1 : 'first'} {1 : 'first', 'next': 'second'}

Operators and their evaluation order

Operators and their evaluation order

Highest	Operator	Comment
	, [...] {...} '...'	Tuple, list & dict. creation; string conv.
	s**[i]** s**[i:j]** s.*attr* f**(...)**	indexing & slicing; attributes, fct calls
	+x, -x, ~x	Unary operators
	x******y	Power
	x*****y x/y x**%**y	mult, division, modulo
	x+y x-y	addition, substraction
	x<<y x>>y	Bit shifting
	x**&**y	Bitwise and
	x^y	Bitwise exclusive or
	x|y	Bitwise or
	x<y x<=y x>y x>=y x==y x**!**=y x<>y x **is** y x **is not** y x **in** s x **not in** s	Comparison, identity, membership
	not x	boolean negation
	x **and** y	boolean and
	x **or** y	boolean or
Lowest	**lambda** *args*: *expr*	anonymous function

- Alternate names are defined in module operator (e.g. __add__ and add for +)
- Most operators are overridable

-

Basic Types and Their Operations

Comparisons (defined between *any* types)

Comparisons

Comparison	Meaning	Notes
<	strictly less than	(1)
<=	less than or equal to	
>	strictly greater than	
>=	greater than or equal to	
==	equal to	
!= or <>	not equal to	
is	object identity	(2)
is not	negated object identity	(2)

Notes :

Comparison behavior can be overridden for a given class by defining special method __cmp__.

(1) X < Y < Z < W has expected meaning, unlike C

(2) Compare object identities (i.e. **id**(object)), not object values.

Boolean values and operators

Boolean values and operators

Value or Operator	Returns	Notes
None, numeric zeros, empty sequences and mappings	**False**	
all other values	**True**	
not *x*	**True** if *x* is **False**, else **True**	
x **or** *y*	if *x* is **False** then *y*, else *x*	(1)
x **and** *y*	if *x* is **False** then *x*, else *y*	(1)

Notes :

Truth testing behavior can be overridden for a given class by defining special method __nonzero__.

(1) Evaluate second arg only if necessary to determine outcome.

None

None is used as default return value on functions. Built-in single object with type NoneType.

Input that evaluates to None does not print when running Python interactively.

Numeric types

Floats, integers and long integers.

Floats are implemented with C doubles.
Integers are implemented with C longs.
Long integers have unlimited size (only limit is system resources)

Operators on all numeric types

Operators on all numeric types

Operation	Result
abs(*x*)	the absolute value of *x*
int(*x*)	*x* converted to integer
long(*x*)	*x* converted to long integer
float(*x*)	*x* converted to floating point
-*x*	*x* negated
+*x*	*x* unchanged
x + *y*	the sum of *x* and *y*
x - *y*	difference of *x* and *y*
x * *y*	product of *x* and *y*
x / *y*	quotient of *x* and *y*
x % *y*	remainder of *x* / *y*
divmod(*x*, *y*)	the tuple (*x*/*y*, *x*%*y*)
x ** *y*	*x* to the power *y* (the same as **pow**(*x*, *y*))

Bit operators on integers and long integers

Bit operators

Operation	>Result
~*x*	the bits of *x* inverted
x ^ *y*	bitwise exclusive or of *x* and *y*
x & *y*	bitwise and of *x* and *y*
x | *y*	bitwise or of *x* and *y*
x << *n*	*x* shifted left by *n* bits
x >> *n*	*x* shifted right by *n* bits

Complex Numbers

- represented as a pair of machine-level double precision floating point numbers.

- The real and imaginary value of a complex number z can be retrieved through
- the attributes z.real and z.imag.

Numeric exceptions

TypeError
raised on application of arithmetic operation to non-number
OverflowError
numeric bounds exceeded
ZeroDivisionError
raised when zero second argument of div or modulo op

Operations on all sequence types (lists, tuples, strings)

Operations on all sequence types

Operation	Result	Notes
x **in** *s*	1 if an item of *s* is equal to *x*, else 0	
x **not in** *s*	0 if an item of *s* is equal to *x*, else 1	
s + *t*	the concatenation of *s* and *t*	
s ***** *n*, *n******s*	*n* copies of *s* concatenated	
*s***[***i***]**	*i*'th item of *s*, origin 0	(1)
*s***[***i***:***j***]**	slice of *s* from *i* (included) to *j* (excluded)	(1), (2)
len(*s*)	length of *s*	
min(*s*)	smallest item of *s*	
max(*s*)	largest item of *(s*)	

Notes :

(1) if *i* or *j* is negative, the index is relative to the end of the string, ie len(*s*)+ *i* or len(*s*)+*j* is substituted. But note that -0 is still 0.
(2) The slice of *s* from *i* to *j* is defined as the sequence of items with index *k* such that *i* <= *k* < *j*. If *i* or *j* is greater thanlen(*s*), use len(*s*). If *i* is omitted, use len(*s*). If *i* is greater than or equal to *j*, the slice is empty.

Operations on mutable (=modifiable) sequences (lists)

Operations on mutable sequences

Operation	Result	Notes
*s***[***i***]** =*x*	item *i* of *s* is replaced by *x*	
s[*i*:*j*] = *t*	slice of *s* from *i* to *j* is replaced by *t*	
del *s*[*i*:*j*]	same as *s*[*i*:*j*] = []	
s.**append**(*x*)	same as *s*[len(*s*) : len(*s*)] = [*x*]	
s.**extend**(*x*)	same as *s*[len(*s*):len(*s*)]= *x*	(5)
s.**count**(*x*)	return number of *i*'s for which *s*[*i*] == *x*	
s.**index**(*x*)	return smallest *i* such that *s*[*i*] == *x*	(1)
s.**insert**(*i*, *x*)	same as *s*[*i*:*i*] = [*x*] if *i* >= 0	
s.**remove**(*x*)	same as del *s*[*s*.index(*x*)]	(1)
s.**pop([***i***])**	same as x = s[i]; del s[i]; return x	(4)
s.**reverse**()	reverse the items of *s* in place	(3)
s.**sort([***cmpFct***])**	sort the items of *s* in place	(2), (3)

Notes :

(1) raise a ValueError exception when *x* is not found in *s* (i.e. out of range).

(2) The sort() method takes an optional argument specifying a comparison fct of 2 arguments (list items) which should return -1, 0, or 1 depending on whether the 1st argument is considered smaller than, equal to, or larger than the 2nd argument. Note that this slows the sorting process down considerably.

(3) The sort() and reverse() methods **modify** the list **in place** for economy of space when sorting or reversing a large list. They don't return the sorted or reversed list to remind you of this side effect.

(4) The pop() method is not supported by mutable sequence types other than lists. The optional argument i defaults to -1, so that by default the last item is removed and returned.

(5) Raises an exception when x is not a list object.

Operations on mappings (dictionaries)

Operations on mappings

Operation	Result	Notes
len(*d*)	the number of items in *d*	
*d***[***k***]**	the item of *d* with key *k*	(1)
d[*k*] = x	set *d*[*k*] to *x*	
del *d*[*k*]	remove *d*[*k*] from *d*	(1)
d.**clear**()	remove all items from *d*	
d.**copy**()	a shallow copy of *d*	
d.**has_key**(*k*)	1 if *d* has key *k*, else 0	
d.**items**()	a copy of *d*'s list of (key, item) pairs	(2)
d.**keys**()	a copy of *d*'s list of keys	(2)
d1.**update**(*d2*)	for k, v in *d2*.items(): *d1*[k] = v	(3)
d.**values**()	a copy of *d*'s list of values	(2)
d.**get**(*k*,*defaultval*)	the item of *d* with key *k*	(4)
d.**setdefault**(*k*,*defaultval*)	the item of *d* with key *k*	(5)
d.**popitem**()	an arbitrary item of *d*, and removes item.	

Notes :
TypeError is raised if key is not acceptable
(1) KeyError is raised if key k is not in the map
(2) Keys and values are listed in random order
(3) *d2* must be of the same type as *d1*
(4) Never raises an exception if *k* is not in the map, instead it returns *defaultVal*. *defaultVal* is optional, when not provided and *k* is not in the map, None is returned.
(5) Never raises an exception if *k* is not in the map, instead it returns *defaultVal*, and adds *k* to map with value *defaultVal*. *defaultVal* is optional. When not provided and *k* is not in the map, None is returned and added to map.

Operations on strings

Note that these string methods largely (but not completely) supersede the functions available in the string module.

Operations on strings

Operation	Result	Notes
s.**capitalize***()*	return a copy of *s* with only its first character capitalized.	
s.**center***(width)*	return a copy of *s* centered in a string of length *width*.	(1)
s.**count***(sub***[***,start***[***,end***]]***)*	return the number of occurrences of substring *sub* in string *s*.	(2)
s.**encode***(***[***encoding***[***,errors***]]***)*	return an encoded version of *s*. Default encoding is the current default string encoding.	(3)

s.**endswith***(suffix*[,*start*[,*end*]]*)*	return true if *s* ends with the specified *suffix*, otherwise return false.	(2)
s.**expandtabs***(*[*tabsize*]*)*	return a copy of *s* where all tab characters are expanded using spaces.	(4)
s.**find***(sub*[,*start*[,*end*]]*)*	return the lowest index in *s* where substring *sub* is found. Return **-1** if *sub* is not found.	(2)
s.**index***(sub*[,*start*[,*end*]]*)*	like **find()**, but raise **ValueError** when the substring is not found.	(2)
s.**isalnum***()*	return true if all characters in *s* are alphanumeric, false otherwise.	(5)
s.**isalpha***()*	return true if all characters in *s* are alphabetic, false otherwise.	(5)
s.**isdigit***()*	return true if all characters in *s* are digit characters, false otherwise.	(5)
s.**islower***()*	return true if all characters in *s* are lowercase, false otherwise.	(6)
s.**isspace***()*	return true if all characters in *s* are whitespace characters, false otherwise.	(5)
s.**istitle***()*	return true if string *s* is a titlecased string, false otherwise.	(7)
s.**isupper***()*	return true if all characters in *s* are uppercase, false otherwise.	(6)
s.**join***(seq)*	return a concatenation of the strings in the sequence *seq*, seperated by '*s*'s.	
s.**ljust***(width)*	return *s* left justified in a string of length *width*.	(1), (8)
s.**lower***()*	return a copy of *s* converted to lowercase.	
s.**lstrip***()*	return a copy of *s* with leading whitespace removed.	
s.**replace***(old, new*[, *maxsplit*]*)*	return a copy of *s* with all occurrences of substring *old* replaced by *new*.	(9)
s.**rfind***(sub*[,*start*[,*end*]]*)*	return the highest index in *s* where substring *sub* is found. Return **-1** if *sub* is not found.	(2)
s.**rindex***(sub*[,*start*[,*end*]]*)*	like **rfind()**, but raise **ValueError** when the substring is not found.	(2)
s.**rjust***(width)*	return *s* right justified in a string of length *width*.	(1), (8)
s.**rstrip***()*	return a copy of *s* with trailing whitespace removed.	
s.**split***(*[*sep*[,*maxsplit*]]*)*	return a list of the words in *s*, using *sep* as the delimiter string.	(10)
s.**splitlines***(*[*keepends*]*)*	return a list of the lines in *s*, breaking at line boundaries.	(11)
s.**startswith***(prefix*[,*start*[,*end*]]*)*	return true if *s* starts with the specified *prefix*, otherwise return false.	(2)

s.**strip***()*	return a copy of *s* with leading and trailing whitespace removed.	
s.**swapcase***()*	return a copy of *s* with uppercase characters converted to lowercase and vice versa.	
s.**title***()*	return a titlecased copy of *s*, i.e. words start with uppercase characters, all remaining cased characters are lowercase.	
s.**translate***(table***[***,deletechars***]***)*	return a copy of *s* mapped through translation table *table*.	(12)
s.**upper***()*	return a copy of *s* converted to uppercase.	

Notes :

(1) Padding is done using spaces.

(2) If optional argument *start* is supplied, substring *s***[***start***:]** is processed. If optional arguments *start* and *end* are supplied, substring *s***[***start***:***end***]** is processed.

(3) Optional argument *errors* may be given to set a different error handling scheme. The default for *errors* is **'strict'**, meaning that encoding errors raise a **ValueError**. Other possible values are **'ignore'** and **'replace'**.

(4) If optional argument *tabsize* is not given, a tab size of 8 characters is assumed.

(5) Returns false if string *s* does not contain at least one character.

(6) Returns false if string *s* does not contain at least one cased character.

(7) A titlecased string is a string in which uppercase characters may only follow uncased characters and lowercase characters only cased ones.

(8) *s* is returned if *width* is less than **len(***s***)**.

(9) If the optional argument *maxsplit* is given, only the first *maxsplit* occurrences are replaced.

(10) If *sep* is not specified or **None**, any whitespace string is a separator. If *maxsplit* is given, at most *maxsplit* splits are done.

(11) Line breaks are not included in the resulting list unless *keepends* is given and true.

(12) *table* must be a string of length 256. All characters occurring in the optional argument *deletechars* are removed prior to translation.

String formatting with the % operator

formatString **%** *args*--> evaluates to a string

- *formatString* uses C printf format codes : %, c, s, i, d, u, o, x, X, e, E, f, g, G, r (details below).
- Width and precision may be a ***** to specify that an integer argument gives the actual width or precision.
- The flag characters -, +, blank, # and 0 are understood. (details below)
- %s will convert any type argument to string (uses *str()* function)
- *args* may be a single arg or a tuple of args

```
'%s has %03d quote types.' % ('Python', 2)  # => 'Python has 002 quote types
```

- Right-hand-side can also be a *mapping*:

```
a = '%(lang)s has %(c)03d quote types.' % {'c':2, 'lang':'Python}
```

(*vars()* function very handy to use on right-hand-side.)

Format codes

Conversion	Meaning
d	Signed integer decimal.
i	Signed integer decimal.
o	Unsigned octal.
u	Unsigned decimal.
x	Unsigned hexidecimal (lowercase).
X	Unsigned hexidecimal (uppercase).
e	Floating point exponential format (lowercase).
E	Floating point exponential format (uppercase).
f	Floating point decimal format.
F	Floating point decimal format.
g	Same as "e" if exponent is greater than -4 or less than precision, "f" otherwise.
G	Same as "E" if exponent is greater than -4 or less than precision, "F" otherwise.
c	Single character (accepts integer or single character string).
r	String (converts any python object using repr()).
s	String (converts any python object using str()).
%	No argument is converted, results in a "%" character in the result. (The complete specification is %%.)

Conversion flag characters

Flag	Meaning
#	The value conversion will use the ‘‘alternate form’’.
0	The conversion will be zero padded.
-	The converted value is left adjusted (overrides "-").
	(a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.
+	A sign character ("+" or "-") will precede the conversion (overrides a "space" flag).

File Objects

Created with built-in function **open**; may be created by other modules’ functions as well.

Operators on file objects

File operations

Operation	Result
f.**close**()	Close file *f*.
f.**fileno**()	Get fileno (fd) for file *f*.
f.**flush**()	Flush file *f*'s internal buffer.
f.**isatty**()	1 if file *f* is connected to a tty-like dev, else 0.
f.**read**([*size*])	Read at most *size* bytes from file *f* and return as a string object. If *size* omitted, read to EOF.
f.**readline**()	Read one entire line from file *f*.
f.**readlines**()	Read until EOF with **readline**() and return list of lines read.
f.**xreadlines**()	Return a sequence-like object for reading a file line-by-line without reading the entire file into memory.
f.**seek**(*offset***[**, *whence=0***]**)	Set file *f*'s position, like "stdio's fseek()". *whence* == 0 then use absolute indexing. *whence* == 1 then offset relative to current pos. *whence* == 2 then offset relative to file end.
f.**tell**()	Return file *f*'s current position (byte offset).
f.**write**(*str*)	Write string to file *f*.
f.**writelines**(*list*)	Write list of strings to file *f*.

File Exceptions

EOFError
: End-of-file hit when reading (may be raised many times, e.g. if *f* is a tty).

IOError
: Other I/O-related I/O operation failure

Advanced Types

-See manuals for more details -

- *Module* objects
- *Class* objects
- *Class instance* objects
- *Type* objects (see module: types)
- *File* objects (see above)
- *Slice* objects
- *XRange* objects
- Callable types:
-
    - User-defined (written in Python):
    -

        - User-defined *Function* objects
        - User-defined *Method* objects
    - Built-in (written in C):
    -
        - Built-in *Function* objects
        - Built-in *Method* objects
- Internal Types:
-
    - *Code* objects (byte-compile executable Python code: *bytecode*)
    - *Frame* objects (execution frames)
    - *Traceback* objects (stack trace of an exception)

Statements

Statement	Result
pass	Null statement
del *name*[,*name*]*	Unbind *name*(s) from object. Object will be indirectly(and automatically) deleted only if no longer referenced.
print[>> *fileobject*,] [*s1* [, *s2*]* [,]	Writes to sys.stdout, or to *fileobject* if supplied. Puts spaces between arguments. Puts newline at endunless statement ends with comma. Print is not required when running interactively, simply typing an expression will print its value, unless the value is None.
exec *x* [in *globals* [,*locals*]]	Executes *x* in namespaces provided. Defaultsto current namespaces. *x* can be a string, fileobject or a function object.
callable(*value*,... [*id=value*], [**args*], [***kw*])	Call function *callable* with parameters. Parameters can be passed by name or be omitted if functiondefines default values. E.g. if *callable* is defined as "def *callable*(p1=1, p2=2)" "*callable*()" <=> "*callable*(1, 2)" "*callable*(10)" <=> "*callable*(10, 2)" "*callable*(p2=99)" <=> "*callable*(1, 99)" **args* is a tuple of positional arguments. ***kw* is a dictionary of keyword arguments.

Assignment operators

Assignment operators

Operator	Result	Notes
a = *b*	Basic assignment - assign object *b* to label *a*	(1)
a += *b*	Roughly equivalent to *a* = *a* + *b*	(2)
a -= *b*	Roughly equivalent to *a* = *a* - *b*	(2)
a ***= *b*	Roughly equivalent to *a* = *a* * *b*	(2)
a /= *b*	Roughly equivalent to *a* = *a* / *b*	(2)
a **%**= *b*	Roughly equivalent to *a* = *a* % *b*	(2)
a ****= *b*	Roughly equivalent to *a* = *a* ** *b*	(2)
a **&**= *b*	Roughly equivalent to *a* = *a* & *b*	(2)
a |= *b*	Roughly equivalent to *a* = *a* | *b*	(2)
a ^= *b*	Roughly equivalent to *a* = *a* ^ *b*	(2)
a >>= *b*	Roughly equivalent to *a* = *a* >> *b*	(2)
a <<= *b*	Roughly equivalent to *a* = *a* << *b*	(2)

Notes :

(1) Can unpack tuples, lists, and strings.
first, second = a[0:2]; [f, s] = range(2); c1,c2,c3='abc'
Tip: x,y = y,x swaps x and y.

(2) Not exactly equivalent - *a* is evaluated only once. Also, where possible, operation performed in-place - *a* is modified rather than replaced.

Control flow statements

Statement	Result
if *condition***:** *suite* [**elif** *condition***:** *suite*]* [**else:** *suite*]	usual if/else_if/else statement
while *condition***:** *suite* [**else:** *suite*]	usual while statement. "else" suite is executedafter loop exits, unless the loop is exited with"break"
for *element* **in** *sequence***:** suite [**else:** *suite*]	iterates over *sequence*, assigning each element to *element*.Use built-in *range* function to iterate a number of times."else" *suite* executed at end unless loop exitedwith "break"
break	immediately exits "for" or "while" loop
continue	immediately does next iteration of "for" or "while" loop
return *[result]*	Exits from function (or method) and returns *result* (use a tuple to return more than one value). If no result given, then returns *None*.

Exception statements

Statement	Result
assert *expr*[, *message*]	*expr* is evaluated. if false, raises exception AssertionErrorwith message. Inhibited if __debug__ is 0.
try: *suite1* [**except** [*exception* [, *value*]**:** *suite2*]+ [**else:** *suite3*]	Statements in *suite1* are executed. If an exception occurs, lookin "except" clauses for matching <exception>. If matches or bare"except" execute suite of that clause. If no exception happenssuite in "else" clause is executed after *suite1*.If *exception* has a value, it is put in *value*.*exception* can also be tuple of exceptions, e.g."except (KeyError, NameError), val: print val"
try: *suite1* **finally:** *suite2*	Statements in *suite1* are executed. If noexception, execute *suite2* (even if *suite1* isexited with a "return", "break" or "continue"statement). If exception did occur, executes*suite2* and then immediately reraises exception.
raise *exception* [,*value* [, *traceback*]]	Raises *exception* with optional value*value*. Arg *traceback* specifies a traceback object touse when printing the exception's backtrace.
raise	A raise statement without arguments re-raises the last exception raised in the current function

- An exception is either a *string* (object) or (preferably) a *class instance*.
- Can create a new one simply by creating a new string:

```
my_exception = 'You did something wrong'
try:
    if bad:
        raise my_exception, bad
except my_exception, value:
    print 'Oops', value
```

- Exception classes must be derived from the predefined class: Exception, e.g.:

```
class text_exception(Exception): pass
try:
    if bad:
        raise text_exception()
        # This is a shorthand for the form
        # "raise <class>, <instance>"
 except Exception:
     print 'Oops'
     # This will be printed because
     # text_exception is a subclass of Exception
```

When an error message is printed for an unhandled exception which is a class, the class name is printed, then a colon and a space, and finally the instance converted to a string using the built-in function str().

All built-in exception classes derives from StandardError, itself derived from Exception.

Name Space Statements

[1.51: On Mac & Windows, the case of module file names must now match the case as used in the *import* statement]

```
Packages (>1.5): a package is a name space which maps to a directory including
                 module(s) and the special initialization module '__init__.py'
                 (possibly empty). Packages/dirs can be nested. You address a
                 module's symbol via '[package.[package...]module.symbol's.
```

Name space statements

Statement	Result
import *module1* [as *name1*] [, *module2*]*	Imports modules. Members of module must bereferred to by qualifying with [package.]module name:"import sys; print sys.argv:""import package1.subpackage.module; package1.subpackage.module.foo()" *module1* renamed as *name1*, if supplied.
from *module* **import** *name1* [as *othername1*] [, *name2*]*	Imports names from module *module* in current namespace. "from sys import argv; print argv" "from package1 import module; module.foo()" "from package1.module import foo; foo()" *name1* renamed as *othername1*, if supplied.
from *module* **import ***	Imports **all** names in *module*, except those starting with "_" *to be used sparsely, beware of name clashes* "from sys import *; print argv" "from package.module import *; print x' Only legal at the top level of a module. If *module* defines an *__all__* attribute, only names listed in *__all__* will be imported. NB: "from *package* import *" only imports the symbols definedin the package's __init__.py file, not those in the template modules!
global *name1 [, name2]*	Names are from global scope (usually meaning from module) rather than local (usually meaning only in function). E.g. in function without "global" statements, assuming "a" is name that hasn't been used in function or module so far: - Try to read from "a" -> NameError - Try to write to "a" -> creates "a" local to function If "a" not defined in fct, but is in module, then: - Try to read from "a", gets value from module - Try to write to "a", creates "a" local to fct But note "a[0]=3" starts with search for "a", will use to global "a" if no local "a".

Function Definition

```
def func_id ([param_list]): suite
                -- Creates a function object & binds it to name func_id.
```

```
    param_list ::= [id [, id]*]
    id ::= value | id = value | *id | **id

    [Args are passed by value.Thus only args representing a mutable object
    can be modified (are inout parameters). Use a tuple to return more than
    one value]

Example:
    def test (p1, p2 = 1+1, *rest, **keywords):
            -- Parameters with "=" have default value (v is
               evaluated when function defined).
               If list has "*id" then id is assigned a tuple of
               all remaining args passed to function (like C vararg)
               If list has "**id" then id is assigned a dictionary of
               all extra arguments passed as keywords.
```

Class Definition

```
class <class_id> [(<super_class1> [,<super_class2>]*)]: <suite>
        -- Creates a class object and assigns it name <class_id>
           <suite> may contain local "defs" of class methods and
           assignments to class attributes.
Example:
    class my_class (class1, class_list[3]): ...
                  Creates a class object inheriting from both "class1" and whatever
                  class object "class_list[3]" evaluates to. Assigns new
                  class object to name "my_class".
        - First arg to class methods is always instance object, called 'self'
          by convention.
        - Special method __init__() is called when instance is created.
        - Special method __del__() called when no more reference to object.
        - Create instance by "calling" class object, possibly with arg
          (thus instance=apply(aClassObject, args...) creates an instance!)
        - In current implementation, can't subclass off built-in
          classes. But can "wrap" them, see UserDict & UserList modules,
          and see __getattr__() below.
Example:
     class c (c_parent):
           def __init__(self, name): self.name = name
           def print_name(self): print "I'm", self.name
           def call_parent(self): c_parent.print_name(self)
           instance = c('tom')
           print instance.name
           'tom'
           instance.print_name()
           "I'm tom"
        Call parent's super class by accessing parent's method
        directly and passing "self" explicitly (see "call_parent"
        in example above).
        Many other special methods available for implementing
        arithmetic operators, sequence, mapping indexing, etc.
```

Documentation Strings

Modules, classes and functions may be documented by placing a string literal by itself as the first statement in the suite. The documentation can be retrieved by getting the '**__doc__**' attribute from the module, class or function.

```
Example:
```

```
     class C:
            "A description of C"
            def __init__(self):
                "A description of the constructor"
                # etc.
Then c.__doc__ == "A description of C".
Then c.__init__.__doc__ == "A description of the constructor".
```

Others

```
lambda [param_list]: returnedExpr
                -- Creates an anonymous function. returnedExpr must be
                   an expression, not a statement (e.g., not "if xx:...",
                   "print xxx", etc.) and thus can't contain newlines.
                   Used mostly for filter(), map(), reduce() functions, and GUI call
```

List comprehensions

```
result = [expression for item1 in sequence1  [if condition1]
                        [for item2 in sequence2 ... for itemN in sequenceN]
                      ]
```

is equivalent to:

```
result = []
for item1 in sequence1:
    for item2 in sequence2:
    ...
        for itemN in sequenceN:
             if (condition1) and further conditions:
                 result.append(expression)
```

Built-In Functions

Built-In Functions

Function	Result
__import__(*name*[, *globals*[, *locals*[, *from list*]]])	Imports module within the given context (see lib ref for more details)
abs(*x*)	Return the absolute value of number *x*.
apply(*f*, *args*[, *keywords*])	Calls func/method *f* with arguments *args* and optional keywords.
callable(*x*)	Returns 1 if *x* callable, else 0.
chr(*i*)	Returns one-character string whose ASCII code isinteger *i*
cmp(*x*,*y*)	Returns negative, 0, positive if *x* <, ==, > to *y*
coerce(*x*,*y*)	Returns a tuple of the two *numeric* arguments converted to a common type.

compile(*string, filename, kind*)	Compiles *string* into a code object. *filename* is used in error message, can be any string. It is usually the file from which the code was read, or eg. '<string>'if not read from file.*kind* can be **'eval'** if *string* is a single stmt, or **'single'** which prints the output of expression statements that evaluate to something else than None, or be **'exec'**.
complex(*real*[, *image*])	Builds a complex object (can also be done using J or **j** suffix,e.g. 1+3J)
delattr(*obj, name*)	deletes attribute named *name* of object *obj* <=> del obj.name
dir([*object*])	If no args, returns the list of names in current local symbol table. With a module, class or class instance object as arg, returns list of names in its attr. dict.
divmod(*a,b*)	Returns tuple of (*a*/*b*, *a*%*b*)
eval(*s*[, *globals*[, *locals*]])	Eval string *s* in (optional) *globals*, *locals* contexts. *s* must have no NUL's or newlines. *s* can also be a code object.Example: x = 1; incr_x = eval('x + 1')
execfile(*file*[, *globals*[, *locals*]])	Executes a file without creating a new module, unlike import.
filter(*function, sequence*)	Constructs a list from those elements of *sequence* for which *function* returns true. *function* takes one parameter.
float(*x*)	Converts a number or a string to floating point.
getattr(*object, name*[, *default*]))	Gets attribute called *name* from *object*,e.g. getattr(x, 'f') <=> x.f). If not found, raises AttributeError or returns *default* if specified.
globals()	Returns a dictionary containing current global variables.
hasattr(*object, name*)	Returns true if *object* has attr called *name*.
hash(*object*)	Returns the hash value of the object (if it has one)
hex(*x*)	Converts a number *x* to a hexadecimal string.
id(*object*)	Returns a unique 'identity' integer for an object.
input([*prompt*])	Prints *prompt* if given. Reads input and **evaluates** it.
int(*x*[, *base*])	Converts a number or a string to a plain integer. Optional *base* paramenter specifies base from which to convert string values.
intern(*aString*)	Enters *aString* in the table of "interned strings" and returns the string. Interned strings are 'immortals'.
isinstance(*obj, class*)	returns true if *obj* is an instance of *class*. If issubclass(A,B) then isinstance(x,A) => isinstance(x,B)

issubclass(*class1*, *class2*)	returns true if *class1* is derived from *class2*
len(obj)	Returns the length (the number of items) of an object (sequence, dictionary, or instance of class implementing __len__).
list(*sequence*)	Converts *sequence* into a list. If already a list, returns a **copy** of it.
locals()	Returns a dictionary containing current local variables.
long(*x*[, *base*])	Converts a number or a string to a long integer. Optional *base* paramenter specifies base from which to convert string values.
map(*function, list, ...*)	Applies *function* to every item of *list* and returns a list of the results. If additional arguments are passed, *function* must take that many arguments and it is givent o *function* on each call.
max(*seq*)	Returns the largest item of the non-empty sequence *seq*.
min(*seq*)	Returns the smallest item of a non-empty sequence *seq*.
oct(*x*)	Converts a number to an octal string.
open(*filename* [, *mode*='r', [*bufsize=implementation dependent*]])	Returns a new file object. *filename* is the file name to be opened. *mode* indicates how the file is to be opened: 'r' for reading 'w' for writing (truncating an existing file) 'a' opens it for appending '+' (appended to any of the previous modes) open the file for updating (note that 'w+' truncates the file) 'b' (appended to any of the previous modes) open the file in binary mode *bufsize* is 0 for unbuffered, 1 for line-buffered, negative for sys-default, all else, of (about) given size.
ord(*c*)	Returns integer ASCII value of *c* (a string of len 1). Works with Unicode char.
pow(*x*, *y* [, *z*	

reduce(*f*, *list* [, *init*])	Applies the binary function *f* to the items of *list* so as to reduce the list to a single value.I f *init* given, it is "prepended" to *list*.
reload(*module*)	Re-parses and re-initializes an already imported module. Useful in interactive mode, if you want to reload a module after fixing it. If module was syntactically correct but had an error in initialization, must import it one more time before calling reload().
repr(*object*)	Returns a string containing a printable and if possible **evaluable** representation of an object. <=> **‘**object**‘** (using backquotes). Class redefineable (__repr__). See also str()
round(*x*, *n*=0)	Returns the floating point value *x* rounded to *n* digits after the decimal point.
setattr(*object*, *name*, *value*)	This is the counterpart of getattr().setattr(o, ’foobar’, 3) <=> o.foobar = 3 **Creates** attribute if it doesn’t exist!
slice([*start*,] *stop*[, *step*])	Returns a *slice object* representing a range, with R/O attributes: start, stop, step.
str(*object*)	Returns a string containing a nicely printable representation of an object. Class overridable (__str__).See also repr().
tuple(*sequence*)	Creates a tuple with same elements as *sequence*. If already a tuple, return itself (not a copy).
type(*obj*)	Returns a *type object* [see module *types*] representing the type of *obj*. Example: import types if type(x) == types.StringType: print ’It is a string’NB: it is recommended to use the following form:if isinstance(x, types.StringType): etc...
unichr(*code*)	Returns a unicode string 1 char long with given *code*.
unicode(*string*[, *encoding*[, *error*]]])	Creates a Unicode string from a 8-bit string, using the given encoding name and error treatment (’strict’, ’ignore’,or ’replace’}.
vars([*object*])	Without arguments, returns a dictionary corresponding to the current local symbol table. With a module,class or class instance object as argument returns a dictionary corresponding to the object’ss ymbol table. Useful with "%" formatting operator.
xrange(*start* [, *end* [, *step*]])	Like range(), but doesn’t actually store entire list all at once. Good to use in "for" loops when there is abig range and little memory.
zip(*seq1*[, *seq2*, ...])	Returns a list of tuples where each tuple contains the *n*th element of each of the argument sequences.

Built-In Exceptions

Exception
Root class for all exceptions
SystemExit
On 'sys.exit()'
StandardError
Base class for all built-in exceptions; derived from Exception root class.
ArithmeticError
Base class for OverflowError, ZeroDivisionError, FloatingPointError
FloatingPointError
When a floating point operation fails.
OverflowError

On excessively large arithmetic operation
ZeroDivisionError
On division or modulo operation with 0 as 2nd arg

AssertionError
When an *assert* statement fails.
AttributeError

On attribute reference or assignment failure
EnvironmentError [**new** in 1.5.2]
On error outside Python; error arg tuple is (errno, errMsg...)
IOError [**changed** in 1.5.2]
I/O-related operation failure
OSError [**new** in 1.5.2]
used by the *os* module's *os.error* exception.
EOFError

Immediate end-of-file hit by input() or raw_input()
ImportError
On failure of 'import' to find module or name
KeyboardInterrupt
On user entry of the interrupt key (often 'Control-C')
LookupError
base class for IndexError, KeyError
IndexError
On out-of-range sequence subscript
KeyError
On reference to a non-existent mapping (dict) key
MemoryError
On recoverable memory exhaustion
NameError

On failure to find a local or global (unqualified) name

RuntimeError

Obsolete catch-all; define a suitable error instead

NotImplementedError [**new** in 1.5.2]

On method not implemented

SyntaxError

On parser encountering a syntax error

IndentationError

On parser encountering an indentation syntax error

TabError

On parser encountering an indentation syntax error

SystemError

On non-fatal interpreter error - bug - report it

TypeError

On passing inappropriate type to built-in op or func

ValueError

On arg error not covered by TypeError or more precise

Standard methods & operators redefinition in classes

```
Standard methods & operators map to special '__methods__' and thus may be
 redefined (mostly in in user-defined classes), e.g.:
    class x:
         def __init__(self, v): self.value = v
         def __add__(self, r): return self.value + r
    a = x(3) # sort of like calling x.__init__(a, 3)
    a + 4    # is equivalent to a.__add__(4)
```

Special methods for any class

Method	Description
__init__(*self*, *args*)	Instance initialization (on construction)
__del__(*self*)	Called on object demise (refcount becomes 0)
__repr__(*self*)	repr() and '...' conversions
__str__(*self*)	str() and 'print' statement
__cmp__(*self*, *other*)	Compares *self* to *other* and returns <0, 0, or >0. Implements >, <, == etc...
__lt__(*self*, *other*)	Called for *self* < *other* comparisons. Can return anything, or can raise an exception.
__le__(*self*, *other*)	Called for *self* <= *other* comparisons. Can return anything, or can raise an exception.
__gt__(*self*, *other*)	Called for *self* > *other* comparisons. Can return anything, or can raise an exception.
__ge__(*self*, *other*)	Called for *self* >= *other* comparisons. Can return anything, or can raise an exception.
__eq__(*self*, *other*)	Called for *self* == *other* comparisons. Can return anything, or can raise an exception.
__ne__(*self*, *other*)	Called for *self* != *other* (and *self* <> *other*) comparisons. Can return anything, or can raise an exception.
__hash__(*self*)	Compute a 32 bit hash code; hash() and dictionary ops
__nonzero__(*self*)	Returns 0 or 1 for truth value testing
__getattr__(*self*, *name*)	Called when attr lookup doesn't find <*name*>
__setattr__(*self*, *name*, *value*)	Called when setting an attr (inside, don't use "*self*.*name* = *value*", use "*self*.__dict__[*name*] = *value*")
__delattr__(*self*, *name*)	Called to delete attr <*name*
__call__(*self*, **args*)	called when an instance is called as function.

Operators

See list in the *operator* module. Operator function names are provided with 2 variants, with or without leading & trailing '__' (eg. __add__ or add).

Numeric operations special methods

Operation	Special method
self+*other*	**__add__**(*self*,*other*)
self-*other*	**__sub__**(*self*,*other*)
*self***other*	**__mul__**(*self*,*other*)
self/*other*	**__div__**(*self*,*other*)
self%*other*	**__mod__**(*self*,*other*)
divmod(*self*,*other*)	**__divmod__**(*self*,*other*)
*self*****other*	**__pow__**(*self*,*other*)
self&*other*	**__and__**(*self*,*other*)
self^*other*	**__xor__**(*self*,*other*)
self|*other*	**__or__**(*self*,*other*)
self<<*other*	**__lshift__**(*self*,*other*)
self>>*other*	**__rshift__**(*self*,*other*)
nonzero(*self*)	**__nonzero__**(*self*) (used in boolean testing)
-*self*	**__neg__**(*self*)
+*self*	**__pos__**(*self*)
abs(*self*)	**__abs__**(*self*)
~*self*	**__invert__**(*self*) (bitwise)
self+=*other*	**__iadd__**(*self*,*other*)
self-=*other*	**__isub__**(*self*,*other*)
*self**=*other*	**__imul__**(*self*,*other*)
self/=*other*	**__idiv__**(*self*,*other*)
self%=*other*	**__imod__**(*self*,*other*)
*self*****=*other*	**__ipow__**(*self*,*other*)
self&=*other*	**__iand__**(*self*,*other*)
self^=*other*	**__ixor__**(*self*,*other*)
self|=*other*	**__ior__**(*self*,*other*)
self<<=*other*	**__ilshift__**(*self*,*other*)
self>>=*other*	**__irshift__**(*self*,*other*)

Conversions

Method	Descripion
int(self)	__int__(self)
long(self)	__long__(self)
float(self)	__float__(self)
complex(self)	__complex__(self)
oct(self)	__oct__(self)
hex(self)	__hex__(self)
coerce(self,other)	__coerce__(self,other)

Right-hand-side equivalents for all binary operators exist; are called when class instance is on r-h-s of operator:
a + 3 calls __add__(a, 3)
3 + a calls __radd__(a, 3)

Special operations for some types

	Operation	Special method	Notes
All sequences and maps:			
	len(s)	__len__(s)	length of object, >= 0. Length 0 == false
	s[i]	__getitem__(s,i)	Element at index/key i, origin 0
Sequences, general methods, plus:			
	s[i]=v	__setitem__(s,i,v)	
	del s[i]	__delitem__(s,i)	
	s[i:j]	__getslice__(s,i,j)	
	s[i:j]=seq	__setslice__(s,i,j,seq)	
	del s[i:j]	__delslice__(s,i,j)	s[i:j] = []
	seq * n	__repeat__(seq, n)	
	s1 + s2	= __concat__(s1, s2)	
	i in s	__contains__(s, i)	
Mappings, general methods, plus			
	hash(s)	= __hash__(s)	hash value for dictionary references
	s[k]=v	= __setitem__(s,k,v)	
	del s[k]	= __delitem__(s,k)	

Special informative state attributes for some types:

Lists & Dictionaries

Attribute	Meaning
__methods__	(list, R/O): list of method names of the object

Modules

Attribute	Meaning
__doc__	(string/None, R/O): doc string (<=> __dict__['__doc__'])
__name__	(string, R/O): module name (also in __dict__['__name__'])
__dict__	(dict, R/O): module's name space
__file__	(string/undefined, R/O): pathname of .pyc, .pyo or .pyd (undef for modules statically linked to the interpreter)
__path__	(string/undefined, R/O): fully qualified package name when applies.

Classes

Attribute	Meaning
__doc__	(string/None, R/W): doc string (<=> __dict__['__doc__'])
__name__	(string, R/**W**): class name (also in __dict__['__name__'])
__bases__	(tuple, R/**W)**: parent classes
__dict__	(dict, R/**W**): attributes (class name space)

Instances

Attribute	Meaning
__class__	(class, R/W): instance's class
__dict__	(dict, R/W): attributes

User defined functions

Attribute	Meaning
__doc__	(string/None, R/**W**): doc string
__name__	(string, R/O): function name
func_doc	(R/**W**): same as __doc__
func_name	(R/O): same as __name__
func_defaults	(tuple/None, R/**W**): default args values if any
func_code	(code, R/**W**): code object representing the compiled function body
func_globals	(dict, R/O): ref to dictionary of func global variables

User-defined Methods

Attribute	Meaning
__doc__	(string/None, R/O): doc string
__name__	(string, R/O): method name (same as im_func.__name__)
im_class	(class, R/O): class defining the method (may be a base class)
im_self	(instance/None, R/O): target instance object (None if unbound)
im_func	(function, R/O): function object

Built-in Functions & methods

Attribute	Meaning
__doc__	(string/None, R/O): doc string
__name__	(string, R/O): function name
__self__	[methods only] target object
__members__	list of attr names: [’__doc__’,’__name__’,’__self__’])

Codes

Attribute	Meaning
co_name	(string, R/O): function name
co_argcount	(int, R/0): number of positional args
co_nlocals	(int, R/O): number of local vars (including args)
co_varnames	(tuple, R/O): names of local vars (starting with args)
co_code	(string, R/O): sequence of bytecode instructions
co_consts	(tuple, R/O): literals used by the bytecode, 1st one is function doc (or None)
co_names	(tuple, R/O): names used by the bytecode
co_filename	(string, R/O): filename from which the code was compiled
co_firstlineno	(int, R/O): first line number of the function
co_lnotab	(string, R/O): string encoding bytecode offsets to line numbers.
co_stacksize	(int, R/O): required stack size (including local vars)
co_firstlineno	(int, R/O): first line number of the function
co_flags	(int, R/O): flags for the interpreter bit 2 set if fct uses "*arg" syntaxbit 3 set if fct uses ’**keywords’ syntax

Frames

Attribute	Meaning
f_back	(frame/None, R/O): previous stack frame (toward the caller)
f_code	(code, R/O): code object being executed in this frame
f_locals	(dict, R/O): local vars
f_globals	(dict, R/O): global vars
f_builtins	(dict, R/O): built-in (intrinsic) names
f_restricted	(int, R/O): flag indicating whether fct is executed in restricted mode
f_lineno	(int, R/O): current line number
f_lasti	(int, R/O): precise instruction (index into bytecode)
f_trace	(function/None, R/W): debug hook called at start of each source line
f_exc_type	(Type/None, R/W): Most recent exception type
f_exc_value	(any, R/W): Most recent exception value
f_exc_traceback	(traceback/None, R/W): Most recent exception traceback

Tracebacks

Attribute	Meaning
tb_next	(frame/None, R/O): next level in stack trace (toward the frame where the exception occurred)
tb_frame	(frame, R/O): execution frame of the current level
tb_lineno	(int, R/O): line number where the exception occured
tb_lasti	(int, R/O): precise instruction (index into bytecode)

Slices

Attribute	Meaning
start	(any/None, R/O): lowerbound
stop	(any/None, R/O): upperbound
step	(any/None, R/O): step value

Complex numbers

Attribute	Meaning
real	(float, R/O): real part
imag	(float, R/O): imaginary part

xranges

Attribute	Meaning
tolist	(Built-in method, R/O): ?

Important Modules

sys

Some *sys* variables

Variable	Content
argv	The list of command line arguments passed to a Python script. sys.argv[0] is the script name.
builtin_module_names	A list of strings giving the names of all modules written in C that are linked into this interpreter.
check_interval	How often to check for thread switches or signals (measured in number of virtual machine instructions)
exitfunc	User can set to a parameterless function. It will get called before interpreter exits.
last_type, last_value, last_traceback	Set only when an exception not handled and interpreter prints an error. Used by debuggers.
maxint	maximum positive value for integers
modules	Dictionary of modules that have already been loaded.
path	Search path for external modules. Can be modified by program. sys.path[0] == dir of script executing
platform	The current platform, e.g. "sunos5", "win32"
ps1, ps2	prompts to use in interactive mode.
stdin, stdout, stderr	File objects used for I/O. One can redirect by assigning a new file object to them (or **any** object: with a method write(string) for stdout/stderr, or with a method readline() for stdin)
version	string containing version info about Python interpreter. (and also: copyright, dllhandle, exec_prefix, prefix)
version_info	tuple containing Python version info - *(major, minor, micro, level, serial)*.

Some *sys* functions

Function	Result
displayhook	The function used to display the output of commands issued in interactive mode - defaults to the builtin repr().
excepthook	Can be set to a user defined function, to which any uncaught exceptions are passed.
exit(*n*)	Exits with status *n*. Raises SystemExit exception.(Hence can be caught and ignored by program)
getrefcount(*object*)	Returns the reference count of the object. Generally 1 higher than you might expect, because of *object* arg temp reference.
setcheckinterval(*interval*)	Sets the interpreter's thread switching interval (in number of virtualcode instructions, default:10).
settrace(*func*)	Sets a trace function: called before each line ofcode is exited.
setprofile(*func*)	Sets a profile function for performance profiling.
exc_info()	Info on exception currently being handled; this is atuple (exc_type, exc_value, exc_traceback).**Warning**: assigning the traceback return value to a local variable in a function handling an exception will cause a circular reference.
setdefaultencoding(*encoding*)	Change default Unicode encoding - defaults to 7-bit ASCII.
getrecursionlimit()	Retrieve maximum recursion depth.
setrecursionlimit()	Set maximum recursion depth. (Defaults to 1000.)

os

"synonym" for whatever O/S-specific module is proper for current environment. this module uses posix whenever possible.
(see also M.A. Lemburg's utility platform.py)

Some *os* variables

Variable	Meaning
name	name of O/S-specific module (e.g. "posix", "mac", "nt")
path	O/S-specific module for path manipulations. On Unix, os.path.split() <=> posixpath.split()
curdir	string used to represent current directory ('.')
pardir	string used to represent parent directory ('..')
sep	string used to separate directories ('/' or '\'). **Tip**: use os.path.join() to build portable paths.
altsep	Alternate sep if applicable (None otherwise)
pathsep	character used to separate search path components (as in $PATH), eg. ';' for windows.
linesep	line separator as used in **binary** files, ie '\n' on Unix, '\r\n' on Dos/Win, '\r'

Some *os* functions

Function	Result
makedirs(*path*[, *mode*=0777])	Recursive directory creation (create required intermediary dirs); os.error if fails.
removedirs(*path*)	Recursive directory delete (delete intermediary **empty** dirs); if fails.
renames(*old*, *new*)	Recursive directory or file renaming; os.error if fails.

posix
don't import this module directly, import *os* instead !
(see also module: shutil for file copy & remove fcts)

***posix* Variables**

Variable	Meaning
environ	dictionary of environment variables, e.g.posix.environ['HOME'].
error	exception raised on POSIX-related error. Corresponding value is tuple of errno code and *perror()*

popen(*command*, *mode*='r', *bufSize*=0)	Opens a pipe to or from *command*. Result is a file object to read to or write from, as indicated by *mode* being 'r' or 'w'. Use it to catch a command output ('r' mode) or to feed it ('w' mode).
remove(*path*)	See unlink.
rename(*src*, *dst*)	Renames/moves the file or directory *src* to *dst*. [error if target name already exists]
rmdir(*path*)	Removes the empty directory *path*
read(*fd*, *n*)	Reads *n* bytes from file descriptor *fd* and return as string.
stat(*path*)	Returns st_mode, st_ino, st_dev, st_nlink, st_uid,st_gid, st_size, st_atime, st_mtime, st_ctime. [st_ino, st_uid, st_gid are dummy on Windows]
system(*command*)	Executes string *command* in a subshell. Returns exit status of subshell (usually 0 means OK).
times()	Returns accumulated CPU times in sec (user, system, children's user,children's sys, elapsed real time). [3 last not on Windows]
unlink(*path*)	Unlinks ("deletes") the file (not dir!) *path*. same as: remove
utime(*path*, (*aTime*, *mTime*))	Sets the access & modified time of the file to the given tuple of values.
wait()	Waits for child pro3 0 ile cdionll. Returnn tuple p_uiof

Function	Result
abspath(*p*)	Returns absolute path for path *p*, taking current working dir in account.
dirname/basename(*p*)	directory and name parts of the path *p*. See also split.
exists(*p*)	True if string *p* is an existing path (file or directory)
expanduser(*p*)	Returns string that is (a copy of) *p* with "~" expansion done.
expandvars(*p*)	Returns string that is (a copy of) *p* with environment vars expanded. [Windows: case significant; must use Unix: **$**var notation, not %var%]
getsize(*filename*)	return the size in bytes of *filename*. raise os.error.
getmtime(*filename*)	return last modification time of *filename* (integer nb of seconds since epoch).
getatime(*filename*)	return last access time of *filename* (integer nb of seconds since epoch).
isabs(*p*)	True if string *p* is an absolute path.
isdir(*p*)	True if string *p* is a directory.
islink(*p*)	True if string *p* is a symbolic link.
ismount(*p*)	True if string *p* is a mount point [true for all dirs on Windows].
join(*p*[,*q*[,...]])	Joins one or more path components intelligently.
split(*p*)	Splits *p* into (head, tail) where *tail* is last pathname component and <head> is everything leading up to that. <=> (dirname(p), basename(p))
splitdrive(*p*)	Splits path *p* in a pair ('drive:', tail) [Windows]
splitext(*p*)	Splits into (root, ext) where last comp of *root* contains no periods and *ext* is empty or starts with a period.
walk(*p, visit, arg*)	Calls the function *visit* with arguments(*arg*, *dirname*, *names*) for each directory recursively in the directory tree rooted at *p* (including *p* itself if it's a dir.) The argument *dirname* specifies the visited directory, the argument *names* lists the files in the directory. The *visit* function may modify *names* to influence the set of directories visited below*dirname*, e.g., to avoid visiting certain parts of the tree.

shutil

high-level file operations (copying, deleting).

Main *shutil* functions

Function	Result
copy(*src*, *dst*)	Copies the contents of file *src* to file *dst*, retaining file permissions.
copytree(*src*, *dst*[, symlinks])	Recursively copies an entire directory tree rooted at *src*into *dst* (which should not already exist). If *symlinks* is true, links in*src* are kept as such in *dst*.
rmtree(*path*[, *ignore_errors*[, *onerror*]])	Deletes an entire directory tree, ignoring errors if *ignore_errors* true,or calling *onerror*(func, path, sys.exc_info()) if supplied with *func*: faulty function, *path*: concerned file.

(and also: *copyfile, copymode, copystat, copy2*)

time

Variables

Variable	Meaning
altzone	signed offset of local DST timezone in sec west of the 0th meridian.
daylight	nonzero if a DST timezone is specified

Functions

Function	Result
time()	return a float representing UTC time in seconds since the epoch.
gmtime(*secs*), localtime(*secs*)	return a tuple representing time : (year aaaa, month(1-12),day(1-31), hour(0-23), minute(0-59), second(0-59), weekday(0-6, 0 is monday), Julian day(1-366), daylight flag(-1,0 or 1))
asctime(*timeTuple*),	
strftime(*format*, *timeTuple*)	return a formated string representing time.
mktime(*tuple*)	inverse of localtime(). Return a float.
strptime(*string*[, *format*])	parse a formated string representing time, return tuple as in gmtime().
sleep(*secs*)	Suspend execution for <secs> seconds. <secs> can be a float.

and also: *clock*, *ctime*.

string

As of Python 2.0, much (though not all) of the functionality provided by the string module have been superseded by built-in string methods - see Operations on strings for details.

Some *string* variables

Variable	Meaning
digits	The string ’0123456789’
hexdigits, octdigits	legal hexadecimal & octal digits
letters, uppercase, lowercase, whitespace	Strings containing the appropriate characters
index_error	Exception raised by index() if substr not found.

Some *string* functions

Function	Result
expandtabs(*s*, *tabSize*)	returns a copy of string <s> with tabs expanded.
find/rfind(*s*, *sub*[, *start*=0[, *end*=0])	Return the lowest/highest index in <s> where the substring <sub> is found such that <sub> is wholly contained in *s*[start:end]. Return -1 if <sub> not found.
ljust/rjust/center(*s*, *width*)	Return a copy of string <s> left/right justified/centerd in a field of given width, padded with spaces. <s> is never truncated.
lower/upper(*s*)	Return a string that is (a copy of) <s> in lowercase/uppercase
split(*s*[, *sep*=whitespace[, *maxsplit*=0]])	Return a list containing the words of the string <s>,using the string <sep> as a separator.
join(*words*[, *sep*=' '])	Concatenate a list or tuple of words with intervening separators; inverse of split.
replace(*s*, *old*, *new*[, *maxsplit*=0]	Returns a copy of string <s> with all occurences of substring<old> replaced by <new>. Limits to <maxsplit> first substitutions if specified.
strip(*s*)	Return a string that is (a copy of) <s> without leading and trailing whitespace. see also lstrip, rstrip.

re (sre)

Handles Unicode strings. Implemented in new module **sre**, **re** now a mere front-end for compatibility. Patterns are specified as strings. Tip: Use raw strings (e.g. **r**'\w*') to litteralize backslashes.

Regular expression syntax

Form	Description
.	matches any character (including newline if DOTALL flag specified)
^	matches start of the string (of every line in MULTILINE mode)
$	matches end of the string (of every line in MULTILINE mode)
*	0 or more of preceding regular expression (as **many** as possible)
+	1 or more of preceding regular expression (as **many** as possible)
?	0 or 1 occurence of preceding regular expression
*?, +?, ??	Same as *, + and ? but matches as **few** characters as possible
{m,n}	matches from m to n repetitions of preceding RE
{m,n}?	idem, attempting to match as **few** repetitions as possible
[ ]	defines character set: e.g. '[a-zA-Z]' to match all letters (see also \w \S)
[^]	defines complemented character set: matches if char is NOT in set
\	escapes special chars '*?+&$|()' and introduces special sequences (see below). Due to Python string rules, write as '\\' or r'\' in the pattern string.
\\	matches a litteral '\'; due to Python string rules, write as '\\\\' in pattern string, or better using raw string: r'\\'.
|	specifies alternative: 'foo|bar' matches 'foo' or 'bar'
(...)	matches any RE inside (), and delimits a *group*.
(?:...)	idem but doesn't delimit a *group*.
(?=...)	matches if ... matches next, but doesn't consume any of the string e.g. 'Isaac (?=Asimov)' matches 'Isaac' only if followed by 'Asimov'.
(?!...)	matches if ... **doesn't** match next. Negative of (?=...)
(?P<*name*>...)	matches any RE inside (), and delimits a **named** *group*. (e.g. r'(?P<id>[a-zA-Z_]\w*)' defines a group named *id*)
(?P=*name*)	matches whatever text was matched by the earlier group named *name*.
(?#...)	A comment; ignored.
(?*letter*)	*letter* is one of 'i','L', 'm', 's', 'x'. Set the corresponding flags (re.I, re.L, re.M, re.S, re.X) for the entire RE.

Special sequences

Sequence	Description
number	matches content of the *group* of the same number; groups are numbered starting from 1
\A	matches only at the start of the string
\b	empty str at beg or end of *word*: ’\bis\b’ matches ’is’, but not ’his’
\B	empty str NOT at beginning or end of word
\d	any decimal digit (<=> [0-9])
\D	any non-decimal digit char (<=> [^O-9])
\s	any whitespace char (<=> [\t\n\r\f\v])
\S	any non-whitespace char (<=> [^ \t\n\r\f\v])
\w	any alphaNumeric char (depends on LOCALE flag)
\W	any non-alphaNumeric char (depends on LOCALE flag)
\Z	matches only at the end of the string

Variables

Variable	Meaning
error	Exception when pattern string isn’t a valid regexp.

Functions

Function	Result
compile(*pattern*[, *flags*=0])	Compile a RE pattern string into a *regular expression object*. Flags (combinable by |): I or IGNORECASE or (?i) case insensitive matching L or LOCALE or (?L) make \w, \W, \b, \B dependent on thecurrent locale M or MULTILINE or (?m) matches every new line and not onlystart/end of the whole string S or DOTALL or (?s) '.' matches ALL chars, including newline X or VERBOSE or (?x) Ignores whitespace outside character sets
escape(*string*)	return (a copy of) string with all non-alphanumerics backslashed.
match(*pattern, string*[, *flags*])	if 0 or more chars at **beginning** of <string> match the RE pattern string,return a corresponding *MatchObject* instance, or None if no match.
search(*pattern, string*[, *flags*])	scan thru <string> for a location matching <pattern>, return a corresponding *MatchObject* instance, or None if no match.
split(*pattern, string*[, *maxsplit*=0])	split <string> by occurrences of <pattern>. If capturing () are used in pattern, then occurrences of patterns or subpatterns are also returned.
findall(*pattern, string*)	return a list of non-overlapping matches in <pattern>, either a list of groups or a list of tuples if the pattern has more than 1 group.
sub(*pattern, repl, string*[, *count*=0])	return string obtained by replacing the (<count> first) leftmost non-overlapping occurrences of <pattern> (a string or a RE object) in <string> by <repl>; <repl> can be a string or a function called with a single *MatchObj* arg, which must return the replacement string.
subn(*pattern, repl, string*[, *count*=0])	same as sub(), but returns a tuple (newString, numberOfSubsMade)

Regular Expression Objects

(RE objects are returned by the compile fct)

***re* object attributes**

Attribute	Descrition
flags	flags arg used when RE obj was compiled, or 0 if none provided
groupindex	dictionary of {group name: group number} in pattern
pattern	pattern string from which RE obj was compiled

***re* object methods**

Method	Result
match(*string*[, *pos*][, *endpos*])	If zero or more characters at the beginning of string match this regular expression, return a corresponding MatchObject instance. Return None if the string does not match the pattern; note that this is different from a zero-length match. The optional second parameter pos gives an index in the string where the search is to start; it defaults to 0. This is not completely equivalent to slicing the string; the ’’ pattern character matches at the real beginning of the string and at positions just after a newline, but not necessarily at the index where the search is to start. The optional parameter endpos limits how far the string will be searched; it will be as if the string is endpos characters long, so only the characters from pos to endpos will be searched for a match.
search(*string*[, *pos*][, *endpos*])	Scan through string looking for a location where this regular expression produces a match, and return a corresponding MatchObject instance. Return None if no position in the string matches the pattern; note that this is different from finding a zero-length match at some point in the string. The optional pos and endpos parameters have the same meaning as for the match() method.
split(*string*[, *maxsplit*=0])	Identical to the split() function, using the compiled pattern.
findall(*string*)	Identical to the findall() function, using the compiled pattern.
sub(*repl*, *string*[, *count*=0])	Identical to the sub() function, using the compiled pattern.
subn(*repl*, *string*[, *count*=0])	Identical to the subn() function, using the compiled pattern.

Match Objects

(Match objects are returned by the match & search functions)

***Match* object attributes**

Attribute	Description
pos	value of pos passed to search or match functions; index into string at which RE engine started search.
endpos	value of endpos passed to search or match functions; index into string beyond which RE engine won’t go.
re	RE object whose match or search fct produced this MatchObj instance
string	string passed to match() or search()

***Match* object functions**

Function	Result
group([*g1*, *g2*, ...])	returns one or more groups of the match. If **one** arg, result is a string; if multiple args, result is a tuple with one item per arg. If *gi* is 0, return value is entire matching string; if 1 <= *gi* <= 99, return string matching group #*gi* (or None if no such group); *gi* may also be a group *name*.
groups()	returns a tuple of all groups of the match; groups not participating to the match have a value of None. Returns a string instead of tupleif len(tuple)=1
start(*group*), end(*group*)	returns indices of start & end of substring matched by group (or None if group exists but doesn't contribute to the match)
span(*group*)	returns the 2-tuple (start(group), end(group)); can be (None, None) if group didn't contibute to the match.

math

Variables:

```
pi
e
```

Functions (see ordinary C man pages for info):

```
acos(x)
asin(x)
atan(x)
atan2(x, y)
ceil(x)
cos(x)
cosh(x)
exp(x)
fabs(x)
floor(x)
fmod(x, y)
frexp(x)        -- Unlike C: (float, int) = frexp(float)
ldexp(x, y)
log(x)
log10(x)
modf(x)         -- Unlike C: (float, float) = modf(float)
pow(x, y)
sin(x)
sinh(x)
sqrt(x)
tan(x)
tanh(x)
```

getopt

Functions:

```
getopt(list, optstr)    -- Similar to C. <optstr> is option
```

```
                        letters to look for. Put ':' after letter
                        if option takes arg. E.g.
# invocation was "python test.py -c hi -a arg1 arg2"
   opts, args =  getopt.getopt(sys.argv[1:], 'ab:c:')
# opts would be
   [('-c', 'hi'), ('-a', '')]
# args would be
   ['arg1', 'arg2']
```

List of modules and packages in base distribution

(built-ins and content of python Lib directory)
(Python NT distribution, may be slightly different in other distributions)

Standard library modules

Operation	Result
aifc	Stuff to parse AIFF-C and AIFF files.
anydbm	Generic interface to all dbm clones. (dbhash, gdbm, dbm,dumbdbm)
asynchat	Support for 'chat' style protocols
asyncore	Asynchronous File I/O (in *select* style)
atexit	Register functions to be called at exit of Python interpreter.
audiodev	Audio support for a few platforms.
base64	Conversions to/from base64 RFC-MIME transport encoding .
BaseHTTPServer	Base class forhttp services.
Bastion	"Bastionification" utility (control access to instance vars)
bdb	A generic Python debugger base class.
binhex	Macintosh binhex compression/decompression.
bisect	List bisection algorithms.
calendar	Calendar printing functions.
cgi	Wraps the WWW Forms Common Gateway Interface (CGI).
CGIHTTPServer	CGI http services.
cmd	A generic class to build line-oriented command interpreters.
cmp	Efficiently compare files, boolean outcome only.
cmpcache	Same, but caches 'stat' results for speed.
code	Utilities needed to emulate Python's interactive interpreter
codecs	Lookup existing Unicode encodings and register new ones.
colorsys	Conversion functions between RGB and other color systems.
commands	Tools for executing UNIX commands .
compileall	Force "compilation" of all .py files in a directory.

ConfigParser	Configuration file parser (much like windows .ini files)
copy	Generic shallow and deep copying operations.
copy_reg	Helper to provide extensibility for pickle/cPickle.
dbhash	(g)dbm-compatible interface to bsdhash.hashopen.
difflib	Tool for comparing sequences, and computing the changes required to convert one into another.
dircache	Sorted list of files in a dir, using a cache.
dircmp	Defines a class to build directory diff tools on.
dis	Bytecode disassembler.
distutils	Package installation system.
doctest	Unit testing framework based on running examples embedded in docstrings.
dospath	Common operations on DOS pathnames.
dumbdbm	A dumb and slow but simple dbm clone.
dump	Print python code that reconstructs a variable.
exceptions	Class based built-in exception hierarchy.
filecmp	File comparison.
fileinput	Helper class to quickly write a loop over all standard input files.
find	Find files directory hierarchy matching a pattern.
fnmatch	Filename matching with shell patterns.
formatter	A test formatter.
fpformat	General floating point formatting functions.
ftplib	An FTP client class. Based on RFC 959.
gc	Perform garbacge collection, obtain GC debug stats, and tune GC parameters.
getopt	Standard command line processing. See also ftp://www.pauahtun.org/pub/getargspy.zip
getpass	Utilities to get a password and/or the current user name.
glob	filename globbing.
gopherlib	Gopher protocol client interface.
grep	’grep’ utilities.
gzip	Read & write gzipped files.
htmlentitydefs	Proposed entity definitions for HTML.
htmllib	HTML parsing utilities.
httplib	HTTP client class.
ihooks	Hooks into the "import" mechanism.
imaplib	IMAP4 client.Based on RFC 2060.
imghdr	Recognizing image files based on their first few bytes.
imputil	Privides a way of writing customised import hooks.

inspect	Get information about live Python objects.
keyword	List of Python keywords.
knee	A Python re-implementation of hierarchical module import.
linecache	Cache lines from files.
linuxaudiodev	Linux /dev/audio support.
locale	Support for number formatting using the current locale settings.
macpath	Pathname (or related) operations for the Macintosh.
macurl2path	Mac specific module for conversion between pathnames and URLs.
mailbox	A class to handle a unix-style or mmdf-style mailbox.
mailcap	Mailcap file handling (RFC 1524).
mhlib	MH (mailbox) interface.
mimetools	Various tools used by MIME-reading or MIME-writing programs.
mimetypes	Guess the MIME type of a file.
MimeWriter	Generic MIME writer.
mimify	Mimification and unmimification of mail messages.
mmap	Interface to memory-mapped files - they behave like mutable strings.
multifile	Class to make multi-file messages easier to handle.
mutex	Mutual exclusion -- for use with module sched.
netrc	parses and encapsulates the netrc file format
nntplib	An NNTP client class. Based on RFC 977.
ntpath	Common operations on DOS pathnames.
nturl2path	Mac specific module for conversion between pathnames and URLs.
os	Either mac, dos or posix depending system.
packmail	Create a self-unpacking shell archive.
pdb	A Python debugger.
pickle	Pickling (save and restore) of Python objects (a faster Cimplementation exists in built-in module: cPickle).
pipes	Conversion pipeline templates.
poly	Polynomials.
popen2	variations on pipe open.
poplib	A POP3 client class. Based on the J. Myers POP3 draft.
posixfile	Extended (posix) file operations.
posixpath	Common operations on POSIX pathnames.
pprint	Support to pretty-print lists, tuples, & dictionaries recursively.
profile	Class for profiling python code.
pstats	Class for printing reports on profiled python code.
pty	Pseudo terminal utilities.

py_compile	Routine to "compile" a .py file to a .pyc file.
pyclbr	Parse a Python file and retrieve classes and methods.
pydoc	Interactively convert docstrings to HTML or text.
pyexpat	Interface to the Expay XML parser.
PyUnit	Unit test framework inspired by JUnit.
Queue	A multi-producer, multi-consumer queue.
quopri	Conversions to/from quoted-printable transport encoding.
rand	Don’t use unless you want compatibility with C’s rand().
random	Random variable generators (obsolete, use whrandom)
re	Regular Expressions.
reconvert	Convert old ("regex") regular expressions to new syntax ("re").
regex_syntax	Flags for regex.set_syntax().
regexp	Backward compatibility for module "regexp" using "regex".
regsub	Regular expression subroutines.
repr	Redo repr() but with limits on most sizes.
rexec	Restricted execution facilities ("safe" exec, eval, etc).
rfc822	RFC-822 message manipulation class.
rlcompleter	Word completion for GNU readline 2.0.
robotparser	Parse robot.txt files, useful for web spiders.
sched	A generally useful event scheduler class.
sgmllib	A parser for SGML.
shelve	Manage shelves of pickled objects.
shlex	Lexical analyzer class for simple shell-like syntaxes.
shutil	Utility functions usable in a shell-like program.
SimpleHTTPServer	Simple extension to base http class
site	Append module search paths for third-party packages to sys.path.
smtplib	SMTP Client class (RFC 821)
sndhdr	Several routines that help recognizing sound.
SocketServer	Generic socket server classes.
stat	Constants and functions for interpreting stat/lstat struct.
statcache	Maintain a cache of file stats.
statvfs	Constants for interpreting statvfs struct as returned by os.statvfs() and os.fstatvfs() (if they exist).
string	A collection of string operations.
StringIO	File-like objects that read/write a string buffer (a faster C implementation exists in built-in module: cStringIO).
sunau	Stuff to parse Sun and NeXT audio files.

sunaudio	Interpret sun audio headers.
symbol	Non-terminal symbols of Python grammar (from "graminit.h").
tabnanny	Check Python source for ambiguous indentation.
telnetlib	TELNET client class. Based on RFC 854.
tempfile	Temporary file name allocation.
threading	Proposed new higher-level threading interfaces
threading_api	(doc of the threading module)
toaiff	Convert "arbitrary" sound files to AIFF files .
token	Tokens (from "token.h").
tokenize	Compiles a regular expression that recognizes Python tokens.
traceback	Format and print Python stack traces.
tty	Terminal utilities.
turtle	LogoMation-like turtle graphics
types	Define names for all type symbols in the std interpreter.
tzparse	Parse a timezone specification.
unicodedata	Interface to unicode properties.
urllib	Open an arbitrary URL.
urlparse	Parse URLs according to latest draft of standard.
user	Hook to allow user-specified customization code to run.
UserDict	A wrapper to allow subclassing of built-in dict class.
UserList	A wrapper to allow subclassing of built-in list class.
UserString	A wrapper to allow subclassing of built-in string class.
util	some useful functions that don’t fit elsewhere !!
uu	UUencode/UUdecode.
warnings	Issue warnings, and filter unwanted warnings.
wave	Stuff to parse WAVE files.
weakref	Allows the creation of object references which do not force the object to remain extant. Also allows the creation of proxy objects.
webbrowser	Platform independent URL launcher.
whatsound	Several routines that help recognizing sound files.
whichdb	Guess which db package to use to open a db file.
whrandom	Wichmann-Hill random number generator.
xdrlib	Implements (a subset of) Sun XDR (eXternal Data Representation)
xmllib	A parser for XML, using the derived class as static DTD.
xml.dom	Classes for processing XML using the Document Object Model.
xml.sax	Classes for processing XML using the SAX API.

xreadlines	Provides a sequence-like object for reading a file line-by-line without reading the entire file into memory.
zipfile	Read & write PK zipped files.
zmod	Demonstration of abstruse mathematical concepts.

(following list not revised)

* Built-ins *

```
sys                 Interpreter state vars and functions
__built-in__        Access to all built-in python identifiers
__main__            Scope of the interpreters main program, script or st
array               Obj efficiently representing arrays of basic values
math                Math functions of C standard
time                Time-related functions
regex               Regular expression matching operations
marshal             Read and write some python values in binary format
struct              Convert between python values and C structs
```

* Standard *

```
getopt              Parse cmd line args in sys.argv.  A la UNIX 'getopt'
os                  A more portable interface to OS dependent functional
re                  Functions useful for working with regular expression
string              Useful string and characters functions and exception
whrandom            Wichmann-Hill pseudo-random number generator
thread              Low-level primitives for working with process thread
threading           idem, new recommanded interface.
```

* Unix/Posix *

```
dbm                 Interface to Unix ndbm database library
grp                 Interface to Unix group database
posix               OS functionality standardized by C and POSIX standar
posixpath           POSIX pathname functions
pwd                 Access to the Unix password database
select              Access to Unix select multiplex file synchronization
socket              Access to BSD socket interface
```

* Tk User-interface Toolkit *

```
tkinter             Main interface to Tk
```

* Multimedia *

```
audioop             Useful operations on sound fragments
imageop             Useful operations on images
jpeg                Access to jpeg image compressor and decompressor
rgbimg              Access SGI imglib image files
```

* Cryptographic Extensions *

```
md5                 Interface to RSA's MD5 message digest algorithm
mpz                 Interface to int part of GNU multiple precision libr
rotor               Implementation of a rotor-based encryption algorithm
```

* Stdwin * Standard Window System

```
stdwin              Standard Window System interface
stdwinevents        Stdwin event, command, and selection constants
rect                Rectangle manipulation operations
```

* SGI IRIX * (4 & 5)

```
al          SGI audio facilities
AL          al constants
fl          Interface to FORMS library
FL          fl constants
flp Functions for form designer
fm          Access to font manager library
gl          Access to graphics library
GL          Constants for gl
DEVICE      More constants for gl
imgfile     Imglib image file interface
```

* Suns *

```
sunaudiodev Access to sun audio interface
```

Workspace exploration and idiom hints

```
dir(<module>)                           list functions, variables in <m
dir();                                  get object keys, defaults to lo
X.__methods__;                          list of methods supported by X
X.__members__                           List of X's data attributes
if __name__ == '__main__': main()       invoke main if running as scrip
map(None, lst1, lst2, ...)              merge lists
b = a[:]                                create copy of seq structure
_                                       in interactive mode, is last va
```

Python Mode for Emacs

(Not revised, possibly not up to date)

```
Type C-c ? when in python-mode for extensive help.
INDENTATION
Primarily for entering new code:
        TAB      indent line appropriately
```

```
        LFD      insert newline, then indent
        DEL      reduce indentation, or delete single character
Primarily for reindenting existing code:
        C-c :    guess py-indent-offset from file content; change locally
        C-u C-c :        ditto, but change globally
        C-c TAB  reindent region to match its context
        C-c <    shift region left by py-indent-offset
        C-c >    shift region right by py-indent-offset
MARKING & MANIPULATING REGIONS OF CODE
C-c C-b         mark block of lines
M-C-h           mark smallest enclosing def
C-u M-C-h       mark smallest enclosing class
C-c #           comment out region of code
C-u C-c #       uncomment region of code
MOVING POINT
C-c C-p         move to statement preceding point
C-c C-n         move to statement following point
C-c C-u         move up to start of current block
M-C-a           move to start of def
C-u M-C-a       move to start of class
M-C-e           move to end of def
C-u M-C-e       move to end of class
EXECUTING PYTHON CODE
C-c C-c sends the entire buffer to the Python interpreter
C-c |   sends the current region
C-c !   starts a Python interpreter window; this will be used by
        subsequent C-c C-c or C-c | commands
VARIABLES
py-indent-offset        indentation increment
py-block-comment-prefix comment string used by py-comment-region
py-python-command       shell command to invoke Python interpreter
py-scroll-process-buffer        t means always scroll Python process buffer
py-temp-directory       directory used for temp files (if needed)
py-beep-if-tab-change   ring the bell if tab-width is changed
```

The Python Debugger

(Not revised, possibly not up to date, see 1.5.2 Library Ref section 9.1; in 1.5.2,

Accessing

```
import pdb      (it's a module written in Python)
        -- defines functions :
           run(statement[,globals[, locals]])
                        -- execute statement string under debugger control, with opt
                           global & local environment.
           runeval(expression[,globals[, locals]])
                        -- same as run, but evaluate expression and return value.
           runcall(function[, argument, ...])
                        -- run function object with given arg(s)
           pm()         -- run postmortem on last exception (like debugging a core f
           post_mortem(t)
                        -- run postmortem on traceback object <t>

        -- defines class Pdb :
           use Pdb to create reusable debugger objects. Object
           preserves state (i.e. break points) between calls.
```

```
        runs until a breakpoint hit, exception, or end of program
        If exception, variable '__exception__' holds (exception,value).
```

Commands

```
h, help
        brief reminder of commands
b, break [<arg>]
        if <arg> numeric, break at line <arg> in current file
        if <arg> is function object, break on entry to function <arg>
        if no arg, list breakpoints
cl, clear [<arg>]
        if <arg> numeric, clear breakpoint at <arg> in current file
        if no arg, clear all breakpoints after confirmation
w, where
        print current call stack
u, up
        move up one stack frame (to top-level caller)
d, down
        move down one stack frame
s, step
        advance one line in the program, stepping into calls
n, next
        advance one line, stepping over calls
r, return
        continue execution until current function returns
        (return value is saved in variable "__return__", which
        can be printed or manipulated from debugger)
c, continue
        continue until next breakpoint
a, args
        print args to current function
rv, retval
        prints return value from last function that returned
p, print <arg>
        prints value of <arg> in current stack frame
l, list [<first> [, <last>]]
               List source code for the current file.
               Without arguments, list 11 lines around the current line
               or continue the previous listing.
               With one argument, list 11 lines starting at that line.
               With two arguments, list the given range;
               if the second argument is less than the first, it is a count.
whatis <arg>
        prints type of <arg>
!
        executes rest of line as a Python statement in the current stack frame
q quit
        immediately stop execution and leave debugger
<return>
        executes last command again
Any input debugger doesn't recognize as a command is assumed to be a
Python statement to execute in the current stack frame, the same way
the exclamation mark ("!") command does.
```

Example

```
(1394) python
Python 1.0.3 (Sep 26 1994)
```

```
Copyright 1991-1994 Stichting Mathematisch Centrum, Amsterdam
>>> import rm
>>> rm.run()
Traceback (innermost last):
         File "<stdin>", line 1
         File "./rm.py", line 7
           x = div(3)
         File "./rm.py", line 2
           return a / r
ZeroDivisionError: integer division or modulo
>>> import pdb
>>> pdb.pm()
> ./rm.py(2)div: return a / r
(Pdb) list
         1     def div(a):
         2  ->     return a / r
         3
         4     def run():
         5         global r
         6         r = 0
         7         x = div(3)
         8         print x
[EOF]
(Pdb) print r
0
(Pdb) q
>>> pdb.runcall(rm.run)
etc.
```

Quirks

Breakpoints are stored as filename, line number tuples. If a module is reloaded after editing, any remembered breakpoints are likely to be wrong.

Always single-steps through top-most stack frame. That is, "c" acts like "n".