Contents

- Invocation Options
- Environment Variables
- Lexical Entities: keywords, identifiers, strings, numbers, sequences, dictionaries, operators
- Basic Types And Their Operations
- Advanced Types
- Statements
- Built In Functions
- Built In Exceptions
- Standard methods & operators redefinition in user-created Classes
- Special informative state attributes for some types
- Important Modules: sys, os, posix, posixpath, shutil, time, string, re, math, getopt
- List of modules In base distribution
- Workspace Exploration And Idiom Hints
- Python Mode for Emacs
- The Python Debugger

Invocation Options

```
python [-diOStuvxX?] [-c command | script | - ] [args]
```

- 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 => suppresses asserts)
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 is 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 are 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

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
--to be verified--
   If non-empty, ignore case in file/module names (imports)
Notable lexical entities

Keywords

<table>
<thead>
<tr>
<th>and</th>
<th>del</th>
<th>for</th>
<th>is</th>
<th>raise</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert</td>
<td>elif</td>
<td>from</td>
<td>lambda</td>
<td>return</td>
</tr>
<tr>
<td>break</td>
<td>else</td>
<td>global</td>
<td>not</td>
<td>try</td>
</tr>
<tr>
<td>class</td>
<td>except</td>
<td>if</td>
<td>or</td>
<td>while</td>
</tr>
<tr>
<td>continue</td>
<td>exec</td>
<td>import</td>
<td>pass</td>
<td></td>
</tr>
<tr>
<td>def</td>
<td>finally</td>
<td>in</td>
<td>print</td>
<td></td>
</tr>
</tbody>
</table>

- (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)

Strings

"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.'''
"""
"""
"""
"""
"""

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 concatenated, e.g. 'Monty' ' Python' is the same as 'Monty Python'.
- u'hello' + ' world' --> u'hello world' (coerced to unicode)

String Literal Escapes

\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 or \xHHHH unicode char with hex value HHHH
\AnyOtherChar is left as-is

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

- **Decimal integer**: 1234, 1234567890546378940\text{L} (or \text{l})
- **Octal integer**: 0177, 017777777777777777L (begin with a 0)
- **Hex integer**: 0xFF, 0XFFFFFFFFFFFFFFFFFFFL (begin with 0x or 0X)
- **Long integer (unlimited precision)**: 1234567890123456\text{L} (ends with \text{L} or \text{l})
- **Float** (double precision): 3.14e-10, .001, 10., 1E3
- **Complex**: 1J, 2+3J, 4+5j (ends with J or j, + separates (float) real and imaginary parts)

Sequences

- **String** of length 0, 1, 2 (see above)
  "", 'I', "12", 'hello
'
- **Tuple** of length 0, 1, 2, etc:
  () (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'.

\begin{verbatim}
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.
\end{verbatim}

Dictionaries (Mappings)

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

Operators and their evaluation order

<table>
<thead>
<tr>
<th>Highest</th>
<th>Operator</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>(...) [...] [...] <code>...</code></td>
<td>Tuple, list &amp; dict. creation; string conv.</td>
<td></td>
</tr>
<tr>
<td>s[i] s[i:j] s.attr f(...)</td>
<td>indexing &amp; slicing; attributes, fct calls</td>
<td></td>
</tr>
<tr>
<td>+x, -x, ~x</td>
<td>Unary operators</td>
<td></td>
</tr>
<tr>
<td>x**y</td>
<td>Power</td>
<td></td>
</tr>
<tr>
<td>x*y x/y x%y</td>
<td>mult, division, modulo</td>
<td></td>
</tr>
<tr>
<td>x+y x-y</td>
<td>addition, substraction</td>
<td></td>
</tr>
<tr>
<td>x&lt;&lt;y x&gt;&gt;y</td>
<td>Bit shifting</td>
<td></td>
</tr>
<tr>
<td>x&amp;y</td>
<td>Bitwise and</td>
<td></td>
</tr>
<tr>
<td>x^y</td>
<td>Bitwise exclusive or</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>Bitwise or</td>
</tr>
</tbody>
</table>
Basic Types and Their Operations

Comparisons (defined between *any* types)

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Meaning</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>strictly less than</td>
<td>(1)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>strictly greater than</td>
<td></td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td></td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
<td></td>
</tr>
<tr>
<td>!= or &lt;&gt;</td>
<td>not equal to</td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>object identity</td>
<td>(2)</td>
</tr>
<tr>
<td>is not</td>
<td>negated object identity</td>
<td>(2)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value or Operator</th>
<th>Returns</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>None, numeric zeros, empty sequences and mappings</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>all other values</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>not x</td>
<td>True if x is False, else True</td>
<td></td>
</tr>
<tr>
<td>x or y</td>
<td>if x is False then y, else x</td>
<td>(1)</td>
</tr>
<tr>
<td>x and y</td>
<td>if x is False then x, else y</td>
<td>(1)</td>
</tr>
</tbody>
</table>

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**

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

**Bit operators on integers and long integers**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>~x</code></td>
<td>the bits of x inverted</td>
</tr>
<tr>
<td><code>x ^ y</code></td>
<td>bitwise exclusive or of x and y</td>
</tr>
<tr>
<td><code>x &amp; y</code></td>
<td>bitwise and of x and y</td>
</tr>
<tr>
<td>`x</td>
<td>y`</td>
</tr>
<tr>
<td><code>x &lt;&lt; n</code></td>
<td>x shifted left by n bits</td>
</tr>
<tr>
<td><code>x &gt;&gt; n</code></td>
<td>x shifted right by n bits</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x ) in ( s )</td>
<td>1 if an item of ( s ) is equal to ( x ), else 0</td>
<td></td>
</tr>
<tr>
<td>( x ) not in ( s )</td>
<td>0 if an item of ( s ) is equal to ( x ), else 1</td>
<td></td>
</tr>
<tr>
<td>( s + t )</td>
<td>The concatenation of ( s ) and ( t )</td>
<td></td>
</tr>
<tr>
<td>( s * n ) , ( n * s )</td>
<td>( n ) copies of ( s ) concatenated</td>
<td></td>
</tr>
<tr>
<td>( s[i] )</td>
<td>( i )th item of ( s ), origin 0</td>
<td>(1)</td>
</tr>
<tr>
<td>( s[i:j] )</td>
<td>Slice of ( s ) from ( i ) (included) to ( j ) (excluded)</td>
<td>(1), (2)</td>
</tr>
<tr>
<td>len(( s ))</td>
<td>Length of ( s )</td>
<td></td>
</tr>
<tr>
<td>min(( s ))</td>
<td>Smallest item of ( s )</td>
<td></td>
</tr>
<tr>
<td>max(( s ))</td>
<td>Largest item of ( s )</td>
<td></td>
</tr>
</tbody>
</table>

Notes:

1. If \( i \) or \( j \) is negative, the index is relative to the end of the string, i.e., \( \text{len}(s) + i \) or \( \text{len}(s) + j \) 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 \( i \leq k < j \).
   - If \( i \) or \( j \) is greater than \( \text{len}(s) \), use \( \text{len}(s) \). If \( i \) is omitted, use \( \text{len}(s) \). If \( i \) is greater than or equal to \( j \), the slice is empty.

Operations on mutable (=modifiable) sequences (lists)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s[i] = x )</td>
<td>Item ( i ) of ( s ) is replaced by ( x )</td>
<td></td>
</tr>
<tr>
<td>( s[i:j] = t )</td>
<td>Slice of ( s ) from ( i ) to ( j ) is replaced by ( t )</td>
<td></td>
</tr>
<tr>
<td>del ( s[i:j] )</td>
<td>Same as ( s[i:j] = [] )</td>
<td></td>
</tr>
<tr>
<td>s.append(( x ))</td>
<td>Same as ( s[\text{len}(s) : \text{len}(s)] = [x] )</td>
<td></td>
</tr>
<tr>
<td>s.extend(( x ))</td>
<td>Same as ( s[\text{len}(s) : \text{len}(s)] = x )</td>
<td>(5)</td>
</tr>
<tr>
<td>Operation</td>
<td>Result</td>
<td>Notes</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>\texttt{s.count}(x)</td>
<td>return number of i's for which ( s[i] == x )</td>
<td></td>
</tr>
<tr>
<td>\texttt{s.index}(x)</td>
<td>return smallest i such that ( s[i] == x )</td>
<td>(1)</td>
</tr>
<tr>
<td>\texttt{s.insert}(i, x)</td>
<td>same as ( s[i:i] = [x] ) if ( i &gt;= 0 )</td>
<td></td>
</tr>
<tr>
<td>\texttt{s.remove}(x)</td>
<td>same as \texttt{del s[s.index(x)]}</td>
<td>(1)</td>
</tr>
<tr>
<td>\texttt{s.pop}()</td>
<td>same as ( x = s[i]; \texttt{del s[i]}; \texttt{return x} )</td>
<td>(4)</td>
</tr>
<tr>
<td>\texttt{s.reverse}()</td>
<td>reverse the items of ( s ) in place</td>
<td>(3)</td>
</tr>
<tr>
<td>\texttt{s.sort}([\texttt{cmpFct}])</td>
<td>sort the items of ( s ) in place</td>
<td>(2), (3)</td>
</tr>
</tbody>
</table>

Notes:
(1) raise a \texttt{ValueError} exception when \( x \) is not found in \( s \) (i.e. out of range).
(2) The \texttt{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 \texttt{sort()} and \texttt{reverse()} methods \textbf{modify} the list \textbf{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) [\textit{New 1.5.2}] The \texttt{pop()} method is experimental and not supported by other mutable sequence types than lists. The optional argument \( i \) defaults to -1, so that by default the last item is removed and returned.
(5) [\textit{New 1.5.2}] Experimental ! Raises an exception when \( x \) is not a list object.

Operations on mappings (dictionaries)
**Operations on strings**

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.capitalize()</code></td>
<td>return a copy of <code>s</code> with only its first character capitalized.</td>
<td></td>
</tr>
<tr>
<td><code>s.center(width)</code></td>
<td>return a copy of <code>s</code> centered in a string of length <code>width</code>.</td>
<td>(1)</td>
</tr>
<tr>
<td><code>s.count(sub[,start[,end]])</code></td>
<td>return the number of occurrences of substring <code>sub</code> in string <code>s</code>.</td>
<td>(2)</td>
</tr>
<tr>
<td><code>s.encode([encoding[,errors]])</code></td>
<td>return an encoded version of <code>s</code>. Default encoding is the current default string encoding.</td>
<td>(3)</td>
</tr>
<tr>
<td><code>s.endswith(suffix[,start[,end]])</code></td>
<td>return true if <code>s</code> ends with the specified <code>suffix</code>, otherwise return false.</td>
<td>(2)</td>
</tr>
<tr>
<td><code>s.expandtabs([tabsize])</code></td>
<td>return a copy of <code>s</code> where all tab characters are expanded using spaces.</td>
<td>(4)</td>
</tr>
<tr>
<td><code>s.find(sub[,start[,end]])</code></td>
<td>return the lowest index in <code>s</code> where substring <code>sub</code> is found. Return -1 if <code>sub</code> is not found.</td>
<td>(2)</td>
</tr>
<tr>
<td><code>s.index(sub[,start[,end]])</code></td>
<td>like <code>find()</code>, but raise ValueError when the substring is not found.</td>
<td>(2)</td>
</tr>
<tr>
<td><code>s.isalnum()</code></td>
<td>return true if all characters in <code>s</code> are alphanumeric, false otherwise.</td>
<td>(5)</td>
</tr>
<tr>
<td><code>s.isalpha()</code></td>
<td>return true if all characters in <code>s</code> are alphabetic, false otherwise.</td>
<td>(5)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>s.isdigit()</code></td>
<td>return true if all characters in s are digit characters, false otherwise.</td>
<td>(5)</td>
</tr>
<tr>
<td><code>s.islower()</code></td>
<td>return true if all characters in s are lowercase, false otherwise.</td>
<td>(6)</td>
</tr>
<tr>
<td><code>s.isspace()</code></td>
<td>return true if all characters in s are whitespace characters, false otherwise.</td>
<td>(5)</td>
</tr>
<tr>
<td><code>s.istitle()</code></td>
<td>return true if string s is a titlecased string, false otherwise.</td>
<td>(7)</td>
</tr>
<tr>
<td><code>s.isupper()</code></td>
<td>return true if all characters in s are uppercase, false otherwise.</td>
<td>(6)</td>
</tr>
<tr>
<td><code>s.join(seq)</code></td>
<td>return a concatenation of the strings in the sequence seq, separated by 's'.</td>
<td></td>
</tr>
<tr>
<td><code>s.ljust(width)</code></td>
<td>return s left justified in a string of length width.</td>
<td>(1), (8)</td>
</tr>
<tr>
<td><code>s.lower()</code></td>
<td>return a copy of s converted to lowercase.</td>
<td></td>
</tr>
<tr>
<td><code>s.lstrip()</code></td>
<td>return a copy of s with leading whitespace removed.</td>
<td></td>
</tr>
<tr>
<td><code>s.replace(old, new[, maxsplit])</code></td>
<td>return a copy of s with all occurrences of substring old replaced by new.</td>
<td>(9)</td>
</tr>
<tr>
<td><code>s.rfind(sub[, start[, end]])</code></td>
<td>return the highest index in s where substring sub is found. Return -1 if sub is not found.</td>
<td>(2)</td>
</tr>
<tr>
<td><code>s.rindex(sub[, start[, end]])</code></td>
<td>like rfind(), but raise ValueError when the substring is not found.</td>
<td>(2)</td>
</tr>
<tr>
<td><code>s.rjust(width)</code></td>
<td>return s right justified in a string of length width.</td>
<td>(1), (8)</td>
</tr>
<tr>
<td><code>s.rstrip()</code></td>
<td>return a copy of s with trailing whitespace removed.</td>
<td></td>
</tr>
<tr>
<td><code>s.split([sep[, maxsplit]])</code></td>
<td>return a list of the words in s, using sep as the delimiter string.</td>
<td>(10)</td>
</tr>
<tr>
<td><code>s.splitlines([keepends])</code></td>
<td>return a list of the lines in s, breaking at line boundaries.</td>
<td>(11)</td>
</tr>
<tr>
<td><code>s.startsswith(prefix[, start[, end]])</code></td>
<td>return true if s starts with the specified prefix, otherwise return false.</td>
<td>(2)</td>
</tr>
<tr>
<td><code>s.strip()</code></td>
<td>return a copy of s with leading and trailing whitespace removed.</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Conversion</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Signed integer decimal.</td>
</tr>
</tbody>
</table>
### Conversion flag characters

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>The value conversion will use the &quot;alternate form&quot;.</td>
</tr>
<tr>
<td>0</td>
<td>The conversion will be zero padded.</td>
</tr>
<tr>
<td>-</td>
<td>The converted value is left adjusted (overrides &quot;:&quot;) .</td>
</tr>
<tr>
<td></td>
<td>(a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.</td>
</tr>
<tr>
<td>+</td>
<td>A sign character (&quot;+&quot; or &quot;-&quot;) will precede the conversion (overrides a &quot;space&quot; flag).</td>
</tr>
</tbody>
</table>

### File Objects

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

**Operators on file objects**

- `f.close()` Close file `f`.
- `f.fileno()` Get fileno (fd) for `f`.
- `f.flush()` Flush file's internal buffer.
- `f.isatty()` 1 if file is connected to a tty-like dev, else 0.
- `f.read([size])` Read at most `size` bytes from file and return as a string object. If `size` omitted, read to EOF.
- `f.readline()` Read one entire line from file.
- `f.readlines()` Read until EOF with `readline()` and return list of lines read.
f.\texttt{seek}(\texttt{offset}, \texttt{whence}=0) Set file's position, like "stdio's \texttt{fseek}()".

\begin{itemize}
  \item whence == 0 then use absolute indexing
  \item whence == 1 then offset relative to current pos
  \item whence == 2 then offset relative to file end
\end{itemize}

f.\texttt{tell}()          Return file's current position (byte offset)
f.\texttt{write}(\texttt{str})    Write string to file.
f.\texttt{writelines}(\texttt{list}) Write list of strings to file.

\textbf{File Exceptions}

\begin{itemize}
  \item \texttt{EOFError}
    
    End-of-file hit when reading (may be raised many times, e.g. if \texttt{f} is a tty).
  \item \texttt{IOError}
    
    Other I/O-related I/O operation failure
\end{itemize}

\textbf{Advanced Types}

-See manuals for more details -

\begin{itemize}
  \item \textit{Module} objects
  \item \textit{Class} objects
  \item \textit{Class instance} objects
  \item \textit{Type} objects (see module: types)
  \item \textit{File} objects (see above)
  \item \textit{Slice} objects
  \item \textit{XRange} objects
  \item Callable types:
    \begin{itemize}
      \item User-defined (written in Python):
        \begin{itemize}
          \item User-defined \textit{Function} objects
          \item User-defined \textit{Method} objects
        \end{itemize}
      \item Built-in (written in C):
        \begin{itemize}
          \item Built-in \textit{Function} objects
          \item Built-in \textit{Method} objects
        \end{itemize}
    \end{itemize}
  \item Internal Types:
    \begin{itemize}
      \item \textit{Code} objects (byte-compile executable Python code: \textit{bytecode})
      \item \textit{Frame} objects (execution frames)
      \item \textit{Traceback} objects (stack trace of an exception)
    \end{itemize}
\end{itemize}

\textbf{Statements}

\begin{itemize}
  \item \texttt{pass}          -- Null statement
  \item \texttt{del} \texttt{name[,name]*} -- Unbind \texttt{name(s)} from object. Object will be indirectly (and automatically) deleted only if no longer referenced.
  \item \texttt{print} [\texttt{>> fileobject,}] [s1 [, s2 ]* [,]]
    
    -- Writes to \texttt{sys.stdout}, or to \texttt{fileobject} if supplied.
    
    Puts spaces between arguments. Puts newline at end unless statement ends with comma.
    
    Print is not required when running interactively, simply typing an expression will print its value,
\end{itemize}
unless the value is None.

```python
exec x [in globals [,locals]]
```

-- Executes x in namespaces provided. Defaults to current namespaces. x can be a string, file object or a function object.

```python
callable(value,... [id=value], [*args], [**kw])
```

-- Call function callable with parameters. Parameters can be passed by name or be omitted if function defines 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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Result</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = b</td>
<td>Basic assignment - assign object b to label a</td>
<td>(1)</td>
</tr>
<tr>
<td>a += b</td>
<td>Roughly equivalent to a = a + b</td>
<td>(2)</td>
</tr>
<tr>
<td>a -= b</td>
<td>Roughly equivalent to a = a - b</td>
<td>(2)</td>
</tr>
<tr>
<td>a *= b</td>
<td>Roughly equivalent to a = a * b</td>
<td>(2)</td>
</tr>
<tr>
<td>a /= b</td>
<td>Roughly equivalent to a = a / b</td>
<td>(2)</td>
</tr>
<tr>
<td>a %= b</td>
<td>Roughly equivalent to a = a % b</td>
<td>(2)</td>
</tr>
<tr>
<td>a **= b</td>
<td>Roughly equivalent to a = a ** b</td>
<td>(2)</td>
</tr>
<tr>
<td>a &amp;= b</td>
<td>Roughly equivalent to a = a &amp; b</td>
<td>(2)</td>
</tr>
<tr>
<td>a</td>
<td>= b</td>
<td>Roughly equivalent to a = a</td>
</tr>
<tr>
<td>a ^= b</td>
<td>Roughly equivalent to a = a ^ b</td>
<td>(2)</td>
</tr>
<tr>
<td>a &gt;&gt;= b</td>
<td>Roughly equivalent to a = a &gt;&gt; b</td>
<td>(2)</td>
</tr>
<tr>
<td>a &lt;&lt;= b</td>
<td>Roughly equivalent to a = a &lt;&lt; b</td>
<td>(2)</td>
</tr>
</tbody>
</table>

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.

http://www.brinningonline.net/simon/python/quick-ref2_0.html (14 of 42) [5/18/2001 10:18:37 AM]
Control Flow

```python
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 executed after 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 exited with "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

```python
assert expr[, message]
  -- expr is evaluated. if false, raises exception AssertionError with message. Inhibited if __debug__ is 0.
try: suite1
[except [exception [, value]: suite2]+
[else: suite3]
  -- statements in suite1 are executed. If an exception occurs, look in "except" clauses for matching <exception>. If matches or bare "except" execute suite of that clause. If no exception happens suite 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 no exception, execute suite2 (even if suite1 is exited 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 to use 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 a class instance.
Can create a new one simply by creating a new string:

```python
my_exception = 'You did something wrong'
```

[Python 2.0 Quick Reference](http://www.brunningonline.net/simon/python/quick-ref2_0.html) (15 of 42) [5/18/2001 10:18:37 AM]
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.:

```python
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]'`.

```python
import module1 [as name1] [, module2]*
    -- imports modules. Members of module must be
    referred to by qualifying with [package.]module name:
    "import sys; print sys.argv;"
    "import package1subpackage.module;"
package1subpackage.module.foo()
```

`module1 renamed as name1, if supplied.`

```python
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 package1module import foo; foo()"
    `name1 renamed as othername1, if supplied.`
```

```python
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"
```

NB: "from package import *" only imports the symbols defined in 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 fct without "global" statements, assuming "a" is name that hasn't been used in fct or module so far:
  - Try to read from "a" -> NameError
  - Try to write to "a" -> creates "a" local to fcn
  - 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:**

```python
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:**

```python
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:**

```python
class c (c_parent):
```
```python
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:**
```python
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 callbacks..

**List comprehensions**

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

is equivalent to:

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

---

http://www.brunningonline.net/simon/python/quick-ref2_0.html (18 of 42) [5/18/2001 10:18:37 AM]
**Built-In Functions**

```python
__import__(name[, globals[, locals[, fromlist]]])
```
Imports module within the given context (see lib ref for more details)

```python
abs(x)
```
Return the absolute value of number \( x \).

```python
apply(f, args[, keywords])
```
Calls func/method \( f \) with arguments \( args \) and optional keywords.

```python
callable(x)
```
Returns 1 if \( x \) callable, else 0.

```python
chr(i)
```
Returns one-character string whose ASCII code is integer \( i \)

```python
cmp(x, y)
```
Returns negative, 0, positive if \( x <, ==, > \) to \( y \)

```python
coerce(x, y)
```
Returns a tuple of the two numeric arguments converted to a common type.

```python
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'.

```python
complex(real[, image])
```
Builds a complex object (can also be done using J or \( j \) suffix, e.g. \( 1+3J \))

```python
delattr(obj, name)
```
deletes attribute named \( name \) of object \( obj \) \(<=>\) del \( obj.name \)

```python
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.

```python
divmod(a, b)
```
Returns tuple of \( (a/b, a\%b) \)

```python
eval(s[, globals[, locals]])
```
Eval string \( s \) in (optional) \( globals, locals \) contexts.

\( s \) must have no NULL's or newlines. \( s \) can also be a code object.

**Example:** \( x = 1; \text{incr}_x = \text{eval}('x + 1') \)

```python
execfile(file[, globals[, locals]])
```
Executes a file without creating a new module, unlike import.

```python
filter(function, sequence)
```
Constructs a list from those elements of \( sequence \) for which \( function \) returns true. \( function \) takes one parameter.

```python
float(x)
```
Converts a number or a string to floating point.

```python
getattr(object, name[, default])
```
Gets attribute called \( name \) from \( object \), e.g. \( \text{getattr}(x, 'f') \) \(<=>\) \( x.f \). If not found, raises AttributeError or returns \( default \) if specified.

```python
globals()
```
Returns a dictionary containing current global variables.

```python
hasattr(object, name)
```
Returns true if \( object \) has attr called \( name \).

```python
hash(object)
```
Returns the hash value of the object (if it has one)

```python
hex(x)
```
Converts a number \( x \) to a hexadecimal string.

```python
id(object)
```
Returns a unique 'identity' integer for an object.
**Python 2.0 Quick Reference**

**print**([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 parameter 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) => instanceof(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 parameter 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 given to 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. First two args are same as those for C's "stdio open" function. 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])

Returns x to power y [modulo z]. See also ** operator.

**range**(start [,end [, step]])

Returns list of ints from >= start and < end.

With 1 arg, list from 0..arg-1

With 2 args, list from start..end-1

With 3 args, list from start up to end by step

**raw_input**([prompt])

Prints prompt if given, then reads string from std input (no trailing \n). See also input().

**reduce**(f, list [, init])

Applies the binary function f to the items of list so as to reduce the list to a single value.

If init given, it is "prepended" to list.

**reloads**(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 redefinissable (__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 recommanded 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's symbol 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 a big 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()' Base class for all built-in exceptions; derived from Exception root class.
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 user-defined classes), e.g.:

```python
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**

(s: self, o: other)

__init__(s, args) instance initialization (on construction)
__del__(s) called on object demise (refcount becomes 0)
__repr__(s) repr() and `...` conversions
__str__(s) str() and 'print' statement
__cmp__(s, o) Compares s to o and returns <0, 0, or >0.
             Implements >, <, == etc...
__hash__(s) Compute a 32 bit hash code; hash() and dictionary ops
__nonzero__(s) Returns 0 or 1 for truth value testing
__getattr__(s, name) called when attr lookup doesn't find <name>
__setattr__(s, name, val) called when setting an attr
                        (inside, don't use "self.name = value"
                        use "self.__dict__[name] = val")
__delattr__(s, 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 ading & trailing '__' (eg. __add__ or add).

**Numeric operations special methods**

(s: self, o: other)

```python
s+o = __add__(s,o)  s-o = __sub__(s,o)
s*o = __mul__(s,o)  s/o = __div__(s,o)
s%o = __mod__(s,o)  divmod(s,o) = __divmod__(s,o)
s**o = __pow__(s,o)
s&o = __and__(s,o)
s^o = __xor__(s,o)  s|o = __or__(s,o)
s<<o = __lshift__(s,o)  s>>o = __rshift__(s,o)
nonzero(s) = __nonzero__(s) (used in boolean testing)
-s = __neg__(s)  +s = __pos__(s)
abs(s) = __abs__(s)  ~s = __invert__(s) (bitwise)
s+=o = __iadd__(s,o)  s-=o = __isub__(s,o)
s*=o = __imul__(s,o)  s/=o = __idiv__(s,o)
```
Python 2.0 Quick Reference

```python
s %= o = __imod__(s, o)
s **= o = __ipow__(s, o)
s &= o = __iand__(s, o)
s ^= o = __ixor__(s, o)
s |= o = __ior__(s, o)
s <<= o = __ilshift__(s, o)
s >>= o = __irshift__(s, o)
```

**Conversions**

```python
int(s) = __int__(s)
long(s) = __long__(s)
float(s) = __float__(s)
complex(s) = __complex__(s)
oct(s) = __oct__(s)
hex(s) = __hex__(s)
coerce(s, o) = __coerce__(s, o)
```

**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)`

**All seqs and maps, general operations plus:**

(s: self, i: index or key)

- `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:**

```python
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)
sl + s2 = __concat__(sl, s2)
1 in s = __contains__(s, 1)
```

**Mappings, general methods, plus**

```python
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:**

- `__methods__` (list, R/O): list of method names of the object

**Modules:**

- `__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:**  

- `__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:**

- `__class__` (class, R/W): instance's class
User-defined functions:
__dict__ (dict, R/W): attributes
__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/W): ref to dictionary of func global variables

User-defined Methods:
__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:
__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:
co_name (string, R/O): function name
co_argcount (int, R/O): 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): litterals used by the bytecode, 1st one is fct 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_stacksize (int, R/O): total stack size (including local vars)
co_flags (int, R/O): flags for the interpreter
   bit 2 set if fct uses "*arg" syntax
   bit 3 set if fct uses "**keywords" syntax

Frames:
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:
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 occurred  
tb_lasti (int, R/O): precise instruction (index into bytecode)

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

Complex numbers:
real (float, R/O): real part  
imag (float, R/O): imaginary part

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

Important Modules

sys

Some variables:
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)

exc_type
exc_value
exc_traceback
-- Deprecated since release 1.5. Use exc_info() instead.

exitfunc
-- User can set to a parameterless fcn. 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,  
.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,
Functions:

exit(n) -- Exits with status n. Raises SystemExit exception. (Hence can be caught and ignored by program)

g Moffount(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 of code is exited.

setprofile(func) -- Sets a profile function for performance profiling.

exc_info() -- Info on exception currently being handled; this is a tuple (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 http://starship.skyporn.net/~lemburg/platform.py)

Variables

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 -- [1.5.2] line separator as used in binary files, ie newline on Unix, \

Functions

makedirs(path[, mode=0777]) [new in 1.5.2]

  -- Recursive directory creation (create required intermediary dirs); os.error if fails.

removedirs(path) [new in 1.5.2]

  -- Recursive directory delete (delete intermediary empty dirs); os.error if fails.

renames(old, new) [new in 1.5.2]

  -- Recursive directory or file renaming; os.error if fails.
posix

don't import this module directly, import os instead!

Variables:

- **environ** -- dictionary of environment variables, e.g. posix.enviror['HOME']. [Windows: before release 1.52 case is signifiant; from 1.52, os.environ is all uppercase but accesses are case insensitive]
- **error** -- exception raised on POSIX-related error. Corresponding value is tuple of errno code and perror() string.

Some Functions: (see also module: shutil for file copy & remove fcts)

- **chdir(path)** -- Changes current directory to path.
- **chmod(path, mode)** -- Changes the mode of path to the numeric mode
- **close(fd)** -- Closes file descriptor fd opened with posix.open.
- **_exit(n)** -- Immediate exit, with no cleanups, no SystemExit, etc. Should use this to exit a child process.
- **execv(p, args)** -- "Become" executable p with args args
- **getcwd()** -- Returns a string representing the current working directory
- **getpid()** -- Returns the current process id
- **fork()** -- Like C's fork(). Returns 0 to child, child pid to parent. [Not on Windows]
- **kill(pid, signal)** -- Like C's kill [Not on Windows]
- **listdir(path)** -- Lists (base)names of entries in directory path, excluding '.' and '..'
- **lseek(fd, pos, how)** -- Sets current position in file fd to position pos, expressed as an offset relative to beginning of file (how=0), to current position (how=1), or to end of file (how=2)
- **mkdir(path[, mode])** -- Creates a directory named path with numeric mode (default 0777)
- **open(file, flags, mode)** -- Like C's open(). Returns file descriptor. Use file object fcts rather than this low level ones.
- **pipe()** -- Creates a pipe. Returns pair of file descriptors (r, w) [Not on Windows].
- **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
values.
wait() -- Waits for child process completion. Returns tuple of
          pid, exit_status [Not on Windows]
waitpid(pid, options)
          -- Waits for process pid to complete. Returns tuple of
          pid, exit_status [Not on Windows]
write(fd, str) -- Writes str to file fd. Returns nb of bytes written.
```

---

**posixpath**

Do not import this module directly, import os instead and refer to this module as os.path. (e.g. os.path.exists(p)!)  

**Some Functions (see doc for more):**

```
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.
eexists(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) [new in 1.5.2]
               -- return the size in bytes of filename. raise os.error.
getmtime(filename) [new in 1.5.2]
                -- return last modification time of filename (integer nb of seconds
                since epoch).
getatime(filename) [new in 1.5.2]
                -- 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)
```

---

http://www.brunningonline.net/simon/python/quick-ref2_0.html (29 of 42) [5/18/2001 10:18:38 AM]
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.

[1.52, NT version: samefile, sameopenfile, samestat func. deprecated because not reliable]

---

**shutil**

High-level file operations (copying, deleting).

**Main functions:**

- `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 fct, path: concerned file.

(and also: copyfile, copypmode, copystat, copy2)

---

**time**

**Variables**

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

**Functions**

- `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 formatted string representing time.
- `mktime(tuple)` -- inverse of localtime(). Return a float.
- `strptime(string[, format])` [new in 1.5.2] -- parse a formatted 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 superceded by built-in string methods - see Operations on strings for details.

Some Variables:
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 Functions:
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
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:
.
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 occurrence 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 \ \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`\'' 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:
\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 (== [^0-9])
\s any whitespace char (== [ \t
\r\f\v])
\S any non-whitespace char (== [^ \t
\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:
error -- Exception when pattern string isn't a valid regexp.

Functions:
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 the
 current locale
 M or MULTILINE or (?m): matches every new line and not only
 start/end of the whole string
 S or DOTALL or (?s): '.' matches ALL chars, including newline
 X or VERBOSE or (?x)
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)  [new in 1.5.2]
    -- 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 fct 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)

Attributes:
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

Methods:
match(string[, pos][, endpos])
search(string[, pos][, endpos])

split(string[, maxsplit=0])
findall(string)  [new in 1.5.2]
sub(repl, string[, count=0])
subn(repl, string[, count=0]) -- see equivalent functions above.

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

Attributes:
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()

Methods:
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 tuple if 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 contribute 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 in base distribution

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

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aifc</td>
<td>Stuff to parse AIFF-C and AIFF files.</td>
</tr>
<tr>
<td>anydbm</td>
<td>Generic interface to all dbm clones. (dbhash, gdbm, dbm, dumbdbm)</td>
</tr>
<tr>
<td>asyncchat</td>
<td>Support for 'chat' style protocols</td>
</tr>
<tr>
<td>asyncore</td>
<td>Asynchronous File I/O (in select style)</td>
</tr>
<tr>
<td>atexit</td>
<td>Register functions to be called at exit of Python interpreter.</td>
</tr>
<tr>
<td>audiodev</td>
<td>Audio support for a few applications.</td>
</tr>
<tr>
<td>base64</td>
<td>Conversions to/from base64 RFC-MIME transport encoding.</td>
</tr>
<tr>
<td>BaseHTTPServer</td>
<td>Base class for http services.</td>
</tr>
<tr>
<td>Bastion</td>
<td>&quot;Bastionification&quot; utility (control access to instance vars)</td>
</tr>
<tr>
<td>bdb</td>
<td>A generic Python debugger base class.</td>
</tr>
<tr>
<td>binhex</td>
<td>Macintosh binhex compression/decompression.</td>
</tr>
<tr>
<td>bisect</td>
<td>List bisection algorithms.</td>
</tr>
<tr>
<td>calendar</td>
<td>Calendar printing functions.</td>
</tr>
<tr>
<td>cgi</td>
<td>Wraps the WWW Forms Common Gateway Interface (CGI).</td>
</tr>
<tr>
<td>CGIHTTPServer</td>
<td>CGI http services.</td>
</tr>
<tr>
<td>cmd</td>
<td>A generic class to build line-oriented command interpreters.</td>
</tr>
<tr>
<td>cmp</td>
<td>Efficiently compare files, boolean outcome only.</td>
</tr>
<tr>
<td>cmpcache</td>
<td>Same, but caches 'stat' results for speed.</td>
</tr>
<tr>
<td>code</td>
<td>Utilities needed to emulate Python's interactive interpreter</td>
</tr>
<tr>
<td>codecs</td>
<td>Lookup existing Unicode encodings and register new ones.</td>
</tr>
<tr>
<td>colorsys</td>
<td>Conversion functions between RGB and other color systems.</td>
</tr>
<tr>
<td>commands</td>
<td>Tools for executing UNIX commands.</td>
</tr>
<tr>
<td>compileall</td>
<td>Force &quot;compiled&quot; of all .py files in a directory.</td>
</tr>
<tr>
<td>ConfigParser</td>
<td>Configuration file parser (much like windows .ini files)</td>
</tr>
<tr>
<td>copy</td>
<td>Generic shallow and deep copying operations.</td>
</tr>
<tr>
<td>copy_reg</td>
<td>Helper to provide extensibility for pickle/cPickle.</td>
</tr>
<tr>
<td>dbhash</td>
<td>(g)dbm-compatible interface to bsdhash.hashopen.</td>
</tr>
<tr>
<td>dircache</td>
<td>Sorted list of files in a dir, using a cache.</td>
</tr>
<tr>
<td>dircmp</td>
<td>Defines a class to build directory diff tools on.</td>
</tr>
<tr>
<td>dis</td>
<td>Bytecode disassembler.</td>
</tr>
<tr>
<td>distutils</td>
<td>Package installation system.</td>
</tr>
<tr>
<td>dospath</td>
<td>Common operations on DOS pathnames.</td>
</tr>
<tr>
<td>dumbdbm</td>
<td>A dumb and slow but simple dbm clone.</td>
</tr>
<tr>
<td>dump</td>
<td>Print python code that reconstructs a variable.</td>
</tr>
<tr>
<td>exceptions</td>
<td>Class based built-in exception hierarchy.</td>
</tr>
<tr>
<td>filecmp</td>
<td>File comparison.</td>
</tr>
<tr>
<td>fileinput</td>
<td>Helper class to quickly write a loop over all standard input files.</td>
</tr>
<tr>
<td>find</td>
<td>Find files directory hierarchy matching a pattern.</td>
</tr>
<tr>
<td>fnmatch</td>
<td>Filename matching with shell patterns.</td>
</tr>
<tr>
<td>formatter</td>
<td>A test formatter.</td>
</tr>
<tr>
<td>fpformat</td>
<td>General floating point formatting functions.</td>
</tr>
<tr>
<td>ftplib</td>
<td>An FTP client class. Based on RFC 959.</td>
</tr>
<tr>
<td>gc</td>
<td>Perform garbage collection, obtain GC debug stats, and tune GC parameters.</td>
</tr>
<tr>
<td>getopt</td>
<td>Standard command line processing.</td>
</tr>
</tbody>
</table>

see also ftp://www.pauuhtun.org/pub/getargspy.zip

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.
http libr -- 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 -- Provides a way of writing customised import hooks.
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.
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 --
nntplib -- An NNTP client class. Based on RFC 977.
ntp path -- 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 C implementation 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.
pyexpat -- Interface to the Expay XML parser.
py_compile -- Routine to "compile" a .py file to a .pyc file.
pyclbr -- Parse a Python file and retrieve classes and methods.
Queue -- A multi-producer, multi-consumer queue.
quopri -- Conversions to/from quoted-printable transport encoding.
rnd -- Don't use unless you want compatibility with C's rand().
random -- Random variable generators (obsolete, use whrandom)
re -- Regular Expressions.

http://www.brunningonline.net/simon/python/quick-ref2_0.html (36 of 42) [5/18/2001 10:18:38 AM]
<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reconvert</td>
<td>Convert old &quot;regex&quot; regular expressions to new syntax &quot;re&quot;.</td>
</tr>
<tr>
<td>regex_syntax</td>
<td>Flags for regex.set_syntax().</td>
</tr>
<tr>
<td>regexp</td>
<td>Backward compatibility for module &quot;regexp&quot; using &quot;regex&quot;.</td>
</tr>
<tr>
<td>regrsub</td>
<td>Regular expression subroutines.</td>
</tr>
<tr>
<td>repr</td>
<td>Redo repr() but with limits on most sizes.</td>
</tr>
<tr>
<td>reexec</td>
<td>Restricted execution facilities &quot;safe&quot; exec, eval, etc.</td>
</tr>
<tr>
<td>rfc822</td>
<td>RFC-822 message manipulation class.</td>
</tr>
<tr>
<td>rlcompleter</td>
<td>Word completion for GNU readline 2.0.</td>
</tr>
<tr>
<td>robotparser</td>
<td>Parse robot.txt files, useful for web spiders.</td>
</tr>
<tr>
<td>sched</td>
<td>A generally useful event scheduler class.</td>
</tr>
<tr>
<td>sgmlib</td>
<td>A parser for SGML.</td>
</tr>
<tr>
<td>shelve</td>
<td>Manage shelves of pickled objects.</td>
</tr>
<tr>
<td>shlex</td>
<td>Lexical analyzer class for simple shell-like syntaxes.</td>
</tr>
<tr>
<td>shutil</td>
<td>Utility functions usable in a shell-like program.</td>
</tr>
<tr>
<td>SimpleHTTPServer</td>
<td>Simple extension to base http class</td>
</tr>
<tr>
<td>site</td>
<td>Append module search paths for third-party packages to sys.path.</td>
</tr>
<tr>
<td>smtplib</td>
<td>SMTP Client class (RFC 821)</td>
</tr>
<tr>
<td>sndhdr</td>
<td>Several routines that help recognizing sound.</td>
</tr>
<tr>
<td>SocketServer</td>
<td>Generic socket server classes.</td>
</tr>
<tr>
<td>stat</td>
<td>Constants and functions for interpreting stat/lstat struct.</td>
</tr>
<tr>
<td>statcache</td>
<td>Maintain a cache of file stats.</td>
</tr>
<tr>
<td>statvfs</td>
<td>Constants for interpreting statvfs struct as returned by os.statvfs()</td>
</tr>
<tr>
<td>and os.fstatvfs()</td>
<td>(if they exist).</td>
</tr>
<tr>
<td>string</td>
<td>A collection of string operations.</td>
</tr>
<tr>
<td>StringIO</td>
<td>File-like objects that read/write a string buffer (a faster C implementation exists in built-in module: cStringIO).</td>
</tr>
<tr>
<td>sunau</td>
<td>Stuff to parse Sun and NeXT audio files.</td>
</tr>
<tr>
<td>sunaudio</td>
<td>Interpret sun audio headers.</td>
</tr>
<tr>
<td>symbol</td>
<td>Non-terminal symbols of Python grammar (from &quot;graminit.h&quot;).</td>
</tr>
<tr>
<td>tabnanny</td>
<td>Check Python source for ambiguous indentation.</td>
</tr>
<tr>
<td>telnetlib</td>
<td>TELNET client class. Based on RFC 854.</td>
</tr>
<tr>
<td>tempfile</td>
<td>Temporary file name allocation.</td>
</tr>
<tr>
<td>threading</td>
<td>Proposed new higher-level threading interfaces</td>
</tr>
<tr>
<td>threading_api</td>
<td>(doc of the threading module)</td>
</tr>
<tr>
<td>toaiff</td>
<td>Convert &quot;arbitrary&quot; sound files to AIFF files.</td>
</tr>
<tr>
<td>token</td>
<td>Tokens (from &quot;token.h&quot;).</td>
</tr>
<tr>
<td>tokenize</td>
<td>Compiles a regular expression that recognizes Python tokens.</td>
</tr>
<tr>
<td>traceback</td>
<td>Format and print Python stack traces.</td>
</tr>
<tr>
<td>tty</td>
<td>Terminal utilities.</td>
</tr>
<tr>
<td>turtle</td>
<td>LogoMation-like turtle graphics.</td>
</tr>
<tr>
<td>types</td>
<td>Define names for all type symbols in the std interpreter.</td>
</tr>
<tr>
<td>tzparse</td>
<td>Parse a timezone specification.</td>
</tr>
<tr>
<td>unicodedata</td>
<td>Interface to unicode properties.</td>
</tr>
<tr>
<td>urlib</td>
<td>Open an arbitrary URL.</td>
</tr>
<tr>
<td>urlparse</td>
<td>Parse URLs according to latest draft of standard.</td>
</tr>
<tr>
<td>user</td>
<td>Hook to allow user-specified customization code to run.</td>
</tr>
<tr>
<td>UserDict</td>
<td>A wrapper to allow subclassing of built-in dict class.</td>
</tr>
<tr>
<td>UserList</td>
<td>A wrapper to allow subclassing of built-in list class.</td>
</tr>
<tr>
<td>UserString</td>
<td>A wrapper to allow subclassing of built-in string class.</td>
</tr>
<tr>
<td>util</td>
<td>Some useful functions that don't fit elsewhere !!</td>
</tr>
<tr>
<td>uu</td>
<td>UUencode/UUdecode.</td>
</tr>
<tr>
<td>wave</td>
<td>Stuff to parse WAVE files.</td>
</tr>
<tr>
<td>webbrowser</td>
<td>Platform independent URL launcher.</td>
</tr>
<tr>
<td>whatsound</td>
<td>Several routines that help recognizing sound files.</td>
</tr>
</tbody>
</table>
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)
xmlib -- 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.
zipfile -- Read & write PK zipped files.
zmod -- Demonstration of abstruse mathematical concepts.
_winreg -- Windows registry interface.

(following list not revised)

* Built-ins *

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
</table>
| sys | Interpreter state vars and functions
| __built-in__ | Access to all built-in python identifiers
| __main__ | Scope of the interpreters main program, script or |
| stdin | |
| 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 *

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getopt</td>
<td>Parse cmd line args in sys.argv. A la UNIX 'getopt'.</td>
</tr>
<tr>
<td>os</td>
<td>A more portable interface to OS dependent</td>
</tr>
<tr>
<td>re</td>
<td>Functions useful for working with regular expressions</td>
</tr>
<tr>
<td>string</td>
<td>Useful string and characters functions and exceptions</td>
</tr>
<tr>
<td>whrandom</td>
<td>Wichmann-Hill pseudo-random number generator</td>
</tr>
<tr>
<td>thread</td>
<td>Low-level primitives for working with process threads</td>
</tr>
<tr>
<td>threading</td>
<td>idem, new recommended interface.</td>
</tr>
</tbody>
</table>

* Unix/Posix *

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbm</td>
<td>Interface to Unix ndbm database library</td>
</tr>
<tr>
<td>grp</td>
<td>Interface to Unix group database</td>
</tr>
<tr>
<td>posix</td>
<td>OS functionality standardized by C and POSIX</td>
</tr>
<tr>
<td>standards</td>
<td></td>
</tr>
<tr>
<td>posixpath</td>
<td>POSIX pathname functions</td>
</tr>
<tr>
<td>pwd</td>
<td>Access to the Unix password database</td>
</tr>
<tr>
<td>select</td>
<td>Access to Unix select multiplex file synchronization</td>
</tr>
<tr>
<td>socket</td>
<td>Access to BSD socket interface</td>
</tr>
</tbody>
</table>

* Tk User-interface Toolkit *

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tkinter</td>
<td>Main interface to Tk</td>
</tr>
</tbody>
</table>
* 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 library
rotor       Implementation of a rotor-based encryption algorithm

* Stdwin * Standard Window System

stdin       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 <module>
dir()       get object keys, defaults to local name space
X.__methods__ list of methods supported by X (if any)
X.__members__ List of X's data attributes
if __name__ == '__main__': main() invoke main if running as script
map(None, lst1, lst2, ...) merge lists
b = a[:] create copy of seq structure
_ in interactive mode, is last value printed

---

http://www.brunningonline.net/simon/python/quick-ref2_0.html (39 of 42) [5/18/2001 10:18:38 AM]
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:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAB</td>
<td>Indent line appropriately</td>
</tr>
<tr>
<td>LFD</td>
<td>Insert newline, then indent</td>
</tr>
<tr>
<td>DEL</td>
<td>Reduce indentation, or delete single character</td>
</tr>
</tbody>
</table>

Primarily for reindenting existing code:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-c :</td>
<td>Guess py-indent-offset from file content; change locally</td>
</tr>
<tr>
<td>C-u C-c :</td>
<td>Ditto, but change globally</td>
</tr>
<tr>
<td>C-c TAB</td>
<td>Reindent region to match its context</td>
</tr>
<tr>
<td>C-c &lt;</td>
<td>Shift region left by py-indent-offset</td>
</tr>
<tr>
<td>C-c &gt;</td>
<td>Shift region right by py-indent-offset</td>
</tr>
</tbody>
</table>

MARKING & MANIPULATING REGIONS OF CODE

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-c</td>
<td>Mark block of lines</td>
</tr>
<tr>
<td>M-C-h</td>
<td>Mark smallest enclosing def</td>
</tr>
<tr>
<td>C-u M-C-h</td>
<td>Mark smallest enclosing class</td>
</tr>
<tr>
<td>C-c #</td>
<td>Comment out region of code</td>
</tr>
<tr>
<td>C-u C-c #</td>
<td>Uncomment region of code</td>
</tr>
</tbody>
</table>

MOVING POINT

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-c C-p</td>
<td>Move to statement preceding point</td>
</tr>
<tr>
<td>C-c C-n</td>
<td>Move to statement following point</td>
</tr>
<tr>
<td>C-c C-u</td>
<td>Move up to start of current block</td>
</tr>
<tr>
<td>M-C-a</td>
<td>Move to start of def</td>
</tr>
<tr>
<td>C-u M-C-a</td>
<td>Move to start of class</td>
</tr>
<tr>
<td>M-C-e</td>
<td>Move to end of def</td>
</tr>
<tr>
<td>C-u M-C-e</td>
<td>Move to end of class</td>
</tr>
</tbody>
</table>

EXECUTING PYTHON CODE

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-c</td>
<td>Sends the entire buffer to the Python interpreter</td>
</tr>
<tr>
<td>C-c</td>
<td>Sends the current region</td>
</tr>
<tr>
<td>C-c !</td>
<td>Starts a Python interpreter window; this will be used by</td>
</tr>
<tr>
<td></td>
<td>subsequent C-c C-c or C-c</td>
</tr>
</tbody>
</table>

VARIABLES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>py-indent-offset</td>
<td>Indentation increment</td>
</tr>
<tr>
<td>py-block-comment-prefix</td>
<td>Comment string used by py-comment-region</td>
</tr>
<tr>
<td>py-python-command</td>
<td>Shell command to invoke Python interpreter</td>
</tr>
<tr>
<td>py-scroll-process-buffer</td>
<td>T means always scroll Python process buffer</td>
</tr>
<tr>
<td>py-temp-directory</td>
<td>Directory used for temp files (if needed)</td>
</tr>
<tr>
<td>py-beep-if-tab-change</td>
<td>Ring the bell if tab-width is changed</td>
</tr>
</tbody>
</table>

The Python Debugger

(Not revised, possibly not up to date, see 1.5.2 Library Ref section 9.1; in 1.5.2, you may also use debugger integrated in IDLE)

Accessing

import pdb   # It's a module written in Python

    -- Defines functions:
    run(statement[,globals[, locals]])

    -- Execute statement string under debugger control, with
optional
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)
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 fcn <arg>
   if no arg, list breakpoints
c1, 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.

```python
whatis <arg>
    prints type of <arg>
!
executes rest of line as a Python statement in the current stack frame
```

```python
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)
```

## 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".