

Python 1.5.1

MODULE sys: argv builtin_module_names
copyright exc_info() -> (type, value, traceback)
exec_prefix executable exit(*i*) exitfunc
getrefcount(*obj*) last_type last_value
last_traceback modules path platform
prefix settrace(*func*) setcheckinterval(*i*)
setprofile(*func*) stdin stdout stderr
tracebacklimit version maxint psl ps2

MODULE types (all names actually end in Type,
alternate names in parentheses): None Type Int
Long Float Complex String Tuple List Dict
(Dictionary) Function (Lambda) Code Class
Instance Method (UnboundMethod) Ellipsis
BuiltinFunction (BuiltinMethod) Module
File Slice xrange Traceback Frame

MODULE struct:

pack(*fmt, items...*)
unpack(*fmt, str*)
calcsizes(*fmt*)

1st format char can be:

@: native byte order & align
=: native order, standard align
<: little-endian order, std align
>, !: big-endian (network) order

Format chars:

x: pad byte (no value)
c: char (as string of length 1)
b: signed char (as int)
B: unsigned char (as int)
h: short
H: unsigned short
i: int
I: unsigned int (as long int)
l: long
L: unsigned long (as long int)
f: float
d: double
s: string (preceding # is length)
p: Pascal string
Except for formats 's' and 'p', a
preceding # gives a repeat count

MODULES pickle/cPickle:

Pickler(*file[, binary]*).dump(*obj*)
Unpickler(*file*).load()
dump(*obj, file[, binary]*) load(*file*)
dumps(*obj[, binary]*) loads(*str*)

MODULE string: digits hexdigits letters
lowercase octdigits uppercase whitespace
atof(*str*) atoi(*str[, base]*) atol(*str[, base]*)
capitalize(*str*) capwords(*str*)
expandtabs(*str, tabsizes*)
find(*str, substr[, start[, end]]*)
rfind(*str, substr[, start[, end]]*)
index(*str, substr[, start[, end]]*)
rindex(*str, substr[, start[, end]]*)
count(*str, substr[, start[, end]]*)
split(*str, sep[, maxsplit]*) join(*seq[, sep]*)
rstrip(*str*) lstrip(*str*) strip(*str*) swapcase(*str*)
maketrans(*from, to*) translate(*str, table[, delete]*)
lower(*str*) ljust(*str, width*) rjust(*str, width*)
upper(*str*) center(*str, width*) zfill(*str, width*)
replace(*str, old, new[, maxcount]*)

On failure: [r]find
returns -1;
[r]index raises
ValueError

MODULE re:

escape(*str*) compile(*patt[, flags]*) -> RegexObject
match(*patt, str[, flags]*) -> MatchObject or None
search(*patt, str[, flags]*) -> MatchObject or None
split(*patt, str[, maxsplit]*) -> list
sub(*patt, repl, str[, maxcount]*) -> str
subn(*patt, repl, str[, maxcount]*) -> (str, count)

Flags: I IGNORECASE L LOCALE M MULTILINE S DOTALL
X VERBOSE

RegexObject: r.flags r.pattern r.groupindex
r.match(*str[, pos[, endpos]]*)
r.search(*str[, pos[, endpos]]*)
r.split(*str[, maxsplit]*)
r.sub(*repl, str[, maxcount]*)
r.subn(*repl, str[, maxcount]*)

match: checks
start of string
search: scans
entire string

MatchObject: m.pos m.endpos m.re m.string
m.group([*groups...*]) m.groups() m.start([*group*])
m.end([*group*]) m.span([*group*]) -> (start, end)

Some special re forms: \A, \Z start, end of string
 \b, \B (non-) word boundary \D, \d (non-) digit
 \s, \S (non-) whitespace \W, \w (non-) alphanumeric
 (?iLmsx) set flags (? :re) nongrouping parens
 (?P<name>re) named group (?P=name) backmatch
 (?=re) lookahead assertion (?!re) negative lookahead

MODULES StringIO/cStringIO:

StringIO([*initialContents*]) -> file-like object
obj.getvalue() obj.close() frees buffer

MODULE copy: copy(*obj*) deepcopy(*obj*)

Quick Ref 1.0, ©1998
JasonHarper@pobox.com

PRECEDENCE (low to high):

1: lambda *args: expr*
2: or (boolean)
3: and (boolean)
4: not (boolean)
5: in, not in, is, is not,
 <, <=, >, >=, !=, ==
6: | (bitwise OR)
7: ^ (bitwise XOR)
8: & (bitwise AND)
9: <<, >>
10: +, - (binary)
11: *, /, %
12: +, -, ~ (unary); ** (right to left)
13: x.a, x[i], x[i:j], f(*args*)
14: (...), [...], {...}, `...`

STATEMENTS:

pass break continue
expression
assert *expr[, message]*
[*targets... =*]... *targets... = exprs...*
del *targets...*
print *exprs...*
return [*expr*]
raise *exception[, detail[, traceback]]*
import *modules...*
from *module* import *identifiers...*
from *module* import *
global *identifiers...*
exec *str/file/code* [in *globals[, locals]*]
def *funcname(params...): suite*
class *classname([bases...]): suite*

if *expr: suite*
[elif *expr: suite*]...
[else: *suite*]

while *expr: suite*
[else: *suite*]

for *targets in expr: suite*
[else: *suite*]

try: *suite*
except [*exception[, target]]: suite...*
[else: *suite*]

try: *suite*
finally: *suite*

BUILT-IN ATTRIBUTES & METHODS:

Many objects: `__methods__` `__members__`

Sequences: `append(x)` `count(x)` `index(x)`

`insert(i,x)` `remove(x)` `reverse()`

`sort([comparisonFunc])`

Mappings: `clear()` `copy()` `get(k[,default])`

`has_key(k)` `items()` -> list of (key, value)

`keys()` `update(otherMapping)` `values()`

Files: `closed` `mode` `name` `softspace`

`close()` `flush()` `isatty()` `fileno()`

`read([size])` `readline([size])`

`readlines([sizehint])` `readinto(buffer)`

`seek(offset[,whence])` **whence:** 0 = start,

`tell()` `truncate([size])` 1 = current,

`write(str)` `writelines(list)` 2 = end

Complex numbers: `real` `imag` `conjugate()`

Built-in functions & methods: `__doc__`

`__name__` `__self__` (None for functions)

User-defined functions: `func_doc` `__doc__`

`func_name` `__name__` `func_defaults`

`func_code` `func_globals`

User-defined methods: `im_func` `im_class`

`__doc__` `__name__` `im_self` (None if unbound)

Modules: `__dict__` `__name__` `__doc__`

`__file__`

Classes: `__dict__` `__name__` `__bases__`

`__doc__` `__module__` (name, not the module itself)

Class instances: `__dict__` `__class__`

Code objects: `co_argcount` `co_code`

`co_consts` `co_filename` `co_firstlineno`

`co_flags` `co_lnotab` `co_name` `co_names`

`co_nlocals` `co_stacksize` `co_varnames`

Frame objects: `f_back` `f_builtins` `f_code`

`f_exc_type` `f_exc_value` `f_exc_traceback`

`f_globals` `f_lasti` `f_lineno` `f_locals`

`f_restricted` `f_trace`

Traceback objects: `tb_next` `tb_frame`

`tb_lineno` `tb_lasti`

Slice objects: `start` `stop` `step`

RESERVED WORDS:

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

BUILT-IN FUNCTIONS (conversions first):

Numeric: `float(x)` `int(x)` `long(x)`

`complex(real[,imag])` `coerce(x,y)` -> 2-tuple

`abs(x)` `cmp(x,y)` -> -/0/+

`divmod(x,y)` -> (x/y, x%y)

`max(seq)` `min(seq)` these can also take multiple params

`pow(x,y[,z])` -> (x**y) % z

`round(x[,decimalPlaces])`

String: `chr(i)` `ord(char)` `hex(x)` `oct(x)`

`intern(str)` `str(obj)` `repr(obj)` same as `obj`

Sequence: `list(seq)` `tuple(seq)`

`len(seq)`

`range([start,]stop[,step])`

`slice([start,]stop[,step])`

`xrange([start,]stop[,step])`

Function calling:

`apply(func,[argsTuple[,keywordsDict]])`

`callable(obj)`

`compile(str,filename,'exec'|'eval'|'single')`

`eval(str|code[,globals[,locals]])`

`execfile(filename[,globals[,locals]])`

`filter(func,list)` `func = None` removes false items

`map(func,lists...)` `func = None` transposes lists

`reduce(func,list[,initializer])`

Environment:

`__import__(name[,globals[,locals[,list]])]`

`dir([obj])` `globals()` `locals()` `vars([obj])`

`reload(module)`

Object:

`delattr(obj,name)` `getattr(obj,name)`

`hasattr(obj,name)` `setattr(obj,name,value)`

`hash(obj)` `id(obj)` `type(obj)`

`isinstance(obj,class|type)`

`issubclass(class1,class2)`

I/O:

`input([prompt])` `raw_input([prompt])`

`open(filename[,mode[,bufsize]])`

EXCEPTION HIERARCHY:

Exception <- StandardError <- all others

ArithmeticError <- OverflowError,
ZeroDivisionError, FloatingPointError

LookupError <- IndexError, KeyError

Others: AssertionError AttributeError EOFError

IOError ImportError KeyboardInterrupt MemoryError

NameError RuntimeError SyntaxError SystemError

SystemExit TypeError ValueError

UnexpectedSpanishInquisition

SPECIAL METHODS:

`__init__(self,[args...])`

`__del__(self)`

`__repr__(self)`

`__str__(self)`

`__cmp__(self,other)` -> -/0/+

`__hash__(self)`

`__nonzero__(self)` -> object's Boolean value

`__getattr__(self,name)`

`__setattr__(self,name,value)`

`__delattr__(self,name)`

`__getinitargs__(self)`

`__getstate__(self)`

`__setstate__(self,state)`

Callable objects:

`__call__(self[,args...])`

Sequences & mappings:

`__len__(self)`

`__getitem__(self,key)` **negative key:**

`__setitem__(self,key,value)` **unchanged**

`__delitem__(self,key)`

`__getslice__(self,i,j)` **missing i, j: 0, maxint**

`__setslice__(self,i,j,seq)` **negative i,j:**

`__delslice__(self,i,j)` **len() added**

Numeric:

`__add__(self,right)` **Implements:**

`__sub__(self,right)` +

`__mul__(self,right)` -

`__div__(self,right)` *

`__mod__(self,right)` /

`__divmod__(self,right)` % `divmod()`

`__pow__(self,right[,z])` `pow()`

`__lshift__(self,right)` <<

`__rshift__(self,right)` >>

`__and__(self,right)` &

`__xor__(self,right)` ^

`__or__(self,right)` |

The above also have a `__r*__(self,left)` form

`__neg__(self)` -

`__pos__(self)` +

`__abs__(self)` `abs()`

`__invert__(self)` ~

`__int__(self)` `int()`

`__long__(self)` `long()`

`__float__(self)` `float()`

`__complex__(self)` `complex()`

`__oct__(self)` `oct()`

`__hex__(self)` `hex()`

`__coerce__(self,other)` `coerce()`