Memory Management and Limits

sys includes several functions for understanding and controlling memory usage.

Reference Counts

The primary implementation of Python (CPython) uses *reference counting* and *garbage collection* for automatic memory management. An object is automatically marked to be collected when its reference count drops to zero. To examine the reference count of an existing object, use `getrefcount()`.

```
# sys_getrefcount.py

import sys

one = []
print('At start         :', sys.getrefcount(one))

two = one

print('Second reference :', sys.getrefcount(one))

del two

print('After del        :', sys.getrefcount(one))
```

The value reported is actually one higher than expected because there is a temporary reference to the object held by `getrefcount()` itself.

```
$ python3 sys_getrefcount.py

At start         : 2
Second reference : 3
After del        : 2
```

See also

- gc – Control the garbage collector via the functions exposed in gc.

Object Size

Knowing how many references an object has may help find cycles or a memory leak, but it is not enough to determine what objects are consuming the *most* memory. That requires knowledge about how big objects are.

```
# sys_getsizeof.py

import sys


class MyClass:
    pass


objects = [
    [], (), {}, 'c', 'string', b'bytes', 1, 2.3,
    MyClass, MyClass(),
]

for obj in objects:
    print('{:>10} : {}'.format(type(obj).__name
```

```
    print('{:>10} : {}'.format(type(obj).__name__,
                               sys.getsizeof(obj)))
```

`getsizeof()` reports the size of an object in bytes.

```
$ python3 sys_getsizeof.py

      list : 64
     tuple : 48
      dict : 240
       str : 50
       str : 55
     bytes : 38
       int : 28
     float : 24
      type : 1056
   MyClass : 56
```

The reported size for a custom class does not include the size of the attribute values.

```
# sys_getsizeof_object.py

import sys


class WithoutAttributes:
    pass


class WithAttributes:
    def __init__(self):
        self.a = 'a'
        self.b = 'b'
        return


without_attrs = WithoutAttributes()
print('WithoutAttributes:', sys.getsizeof(without_attrs))

with_attrs = WithAttributes()
print('WithAttributes:', sys.getsizeof(with_attrs))
```

This can give a false impression of the amount of memory being consumed.

```
$ python3 sys_getsizeof_object.py

WithoutAttributes: 56
WithAttributes: 56
```

For a more complete estimate of the space used by a class, provide a `__sizeof__()` method to compute the value by aggregating the sizes of attributes of an object.

```
# sys_getsizeof_custom.py

import sys


class WithAttributes:
    def __init__(self):
        self.a = 'a'
        self.b = 'b'
        return

    def __sizeof__(self):
        return object.__sizeof__(self) + \
            sum(sys.getsizeof(v) for v in self.__dict__.values())


my_inst = WithAttributes()
print(sys.getsizeof(my_inst))
```

```
print(sys.getsizeof(my_inst))
```

This version adds the base size of the object to the sizes of all of the attributes stored in the internal `__dict__`.

```
$ python3 sys_getsizeof_custom.py

156
```

Recursion

Allowing infinite recursion in a Python application may introduce a stack overflow in the interpreter itself, leading to a crash. To eliminate this situation, the interpreter provides a way to control the maximum recursion depth using `setrecursionlimit()` and `getrecursionlimit()`.

```
# sys_recursionlimit.py

import sys

print('Initial limit:', sys.getrecursionlimit())

sys.setrecursionlimit(10)

print('Modified limit:', sys.getrecursionlimit())


def generate_recursion_error(i):
    print('generate_recursion_error({})'.format(i))
    generate_recursion_error(i + 1)


try:
    generate_recursion_error(1)
except RuntimeError as err:
    print('Caught exception:', err)
```

Once the stack size reaches the recursion limit, the interpreter raises a `RuntimeError` exception so the program has an opportunity to handle the situation.

```
$ python3 sys_recursionlimit.py

Initial limit: 1000
Modified limit: 10
generate_recursion_error(1)
generate_recursion_error(2)
generate_recursion_error(3)
generate_recursion_error(4)
generate_recursion_error(5)
generate_recursion_error(6)
generate_recursion_error(7)
generate_recursion_error(8)
Caught exception: maximum recursion depth exceeded while calling
a Python object
```

Maximum Values

Along with the runtime configurable values, `sys` includes variables defining the maximum values for types that vary from system to system.

```
# sys_maximums.py

import sys

print('maxsize   :', sys.maxsize)
print('maxunicode:', sys.maxunicode)
```

`maxsize` is the maximum size of a list, dictionary, string, or other data structure dictated by the C interpreter's size type. `maxunicode` is the largest integer Unicode point supported by the interpreter as currently configured.

```
$ python3 sys_maximums.py

maxsize   : 9223372036854775807
maxunicode: 1114111
```

Floating Point Values

The structure `float_info` contains information about the floating-point type representation used by the interpreter, based on the underlying system's `float` implementation.

```
# sys_float_info.py

import sys

print('Smallest difference (epsilon):', sys.float_info.epsilon)
print()
print('Digits (dig)              :', sys.float_info.dig)
print('Mantissa digits (mant_dig):', sys.float_info.mant_dig)
print()
print('Maximum (max):', sys.float_info.max)
print('Minimum (min):', sys.float_info.min)
print()
print('Radix of exponents (radix):', sys.float_info.radix)
print()
print('Maximum exponent for radix (max_exp):',
      sys.float_info.max_exp)
print('Minimum exponent for radix (min_exp):',
      sys.float_info.min_exp)
print()
print('Max. exponent power of 10 (max_10_exp):',
      sys.float_info.max_10_exp)
print('Min. exponent power of 10 (min_10_exp):',
      sys.float_info.min_10_exp)
print()
print('Rounding for addition (rounds):', sys.float_info.rounds)
```

These values depend on the compiler and underlying system. These examples were produced on OS X 10.9.5 on an Intel Core i7.

```
$ python3 sys_float_info.py

Smallest difference (epsilon): 2.220446049250313e-16

Digits (dig)              : 15
Mantissa digits (mant_dig): 53

Maximum (max): 1.7976931348623157e+308
Minimum (min): 2.2250738585072014e-308

Radix of exponents (radix): 2

Maximum exponent for radix (max_exp): 1024
Minimum exponent for radix (min_exp): -1021

Max. exponent power of 10 (max_10_exp): 308
Min. exponent power of 10 (min_10_exp): -307

Rounding for addition (rounds): 1
```

See also

- The `float.h` C header file for the local compiler contains more details about these settings.

Integer Values

The structure `int_info` holds information about the internal representation of integers used by the interpreter.

```
# sys_int_info.py

import sys

print('Number of bits used to hold each digit:',
      sys.int_info.bits_per_digit)
print('Size in bytes of C type used to hold each digit:',
      sys.int_info.sizeof_digit)
```

These examples were produced on OS X 10.9.5 on an Intel Core i7.

```
$ python3 sys_int_info.py

Number of bits used to hold each digit: 30
Size in bytes of C type used to hold each digit: 4
```

The C type used to store integers internally is determined when the interpreter is built. 64-bit architectures automatically use 30-bit integers by default, and they can be enabled for 32-bit architectures with the configuration flag `--enable-big-digits`.

See also

- Build and C API Changes from *What's New in Python 3.1*

Byte Ordering

`byteorder` is set to the native byte order.

```
# sys_byteorder.py

import sys

print(sys.byteorder)
```

The value is either `big` for big-endian or `little` for little-endian.

```
$ python3 sys_byteorder.py

little
```

See also

- Wikipedia: Endianness – Description of big and little endian memory systems.
- `array` and `struct` – Other modules that depend on the byte order of data.
- **`float.h` – The C header file for the local compiler contains** more details about these settings.

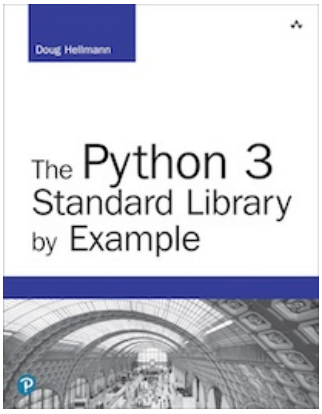
Runtime Environment

Exception Handling

Quick Links
Reference Counts
Object Size
Recursion
Maximum Values
Floating Point Values
Integer Values
Byte Ordering

This page was last updated 2018-03-18.

Navigation
Runtime Environment
Exception Handling



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example