Python 3.1 快速導覽 - 內建數字型態

內建的數字型態 (numeric types) 共有三種，分別是

|  |  |
| --- | --- |
| **型態** | **描述** |
| int | 整數 |
| float | 浮點數 |
| complex | 複數 |

可進行以下的計算

|  |  |
| --- | --- |
| **計算** | **描述** |
| x + y | x 加 y 之和 |
| x - y | x 剪 y 之差 |
| x \* y | x 乘 y 之積 |
| x / y | x 除 y 之商 |
| x // y | x 除 y 之整數商 |
| x % y | x 除 y 之餘數 |
| -x | x 取負數 |
| +x | x 取正數 |
| abs(x) | 回傳 x 的絕對值 |
| int(x) | 轉換 x 為整數 |
| float(x) | 轉換 x 為浮點數 |
| complex(re, im) | 轉換 re 為複數的實部， im 為虛部 |
| c.conjugate() | 回傳 c 的共軛複數 |
| divmod(x, y) | 回傳 (x // y, x % y) |
| pow(x, y) | x 的 y 次方 |
| x \*\* y | x 的 y 次方 |

整數型態 (integer type) 可以進行位元字串 (bit-string) 的運算

|  |  |
| --- | --- |
| **計算** | **描述** |
| x | y | x 逐位元對 y 做或運算 |
| x ^ y | x 逐位元對 y 做互斥或運算 |
| x & y | x 逐位元對 y 做且運算 |
| x << n | x 向左位移 n 位元 |
| x >> n | x 向右位移 n 位元 |
| ~x | 取 x 的位元補數 |

int 型態有以下的方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [int.bit\_length()](http://pydoing.blogspot.com/2011/02/python-bitlength.html) | 回傳 o 的二進位形式共佔多少長度 |

float 型態有以下的方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| [float.as\_integer\_ratio()](http://pydoing.blogspot.com/2011/02/python-asintegerratio.html) | 回傳兩個整數，第一個整數為浮點數的分子，第二個整數則為分母 |
| [float.is\_integer()](http://pydoing.blogspot.com/2011/02/python-isinteger.html) | 判斷 float 是否為整數 |
| [float.hex()](http://pydoing.blogspot.com/2011/02/python-floathex.html) | 回傳 float 的十六進位字串 |
| [classmethod float.fromhex(s)](http://pydoing.blogspot.com/2011/02/python-fromhex.html) | 將表示十六進位數字字串的 s 轉換成浮點數 |

complex 型態有以下的屬性

|  |  |
| --- | --- |
| **屬性** | **描述** |
| real | 複數的實部 |
| imag | 複數的虛部 |

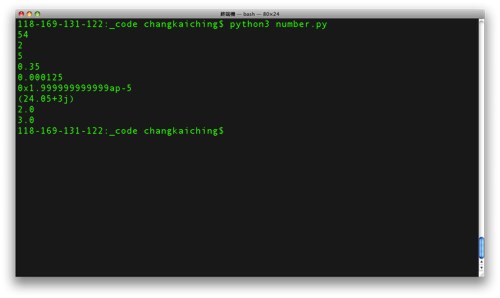
complex 型態有以下的方法

|  |  |
| --- | --- |
| **方法** | **描述** |
| complex.conjugate() | 回傳共軛複數 |

舉例示範如下

[?](http://pydoing.blogspot.tw/2011/02/python-numerictypes.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | a = 22  b = 0.05  c = 2 + 3j    print(a + 32)  print(a >> 3)  print(a.bit\_length())  print(b + 0.3)  print(b \*\* 3)  print(b.hex())  print(c + a + b)  print(c.real)  print(c.imag)    # 《程式語言教學誌》的範例程式  # <http://pydoing.blogspot.com/>  # 檔名：number.py  # 功能：示範 Python 程式  # 作者：張凱慶  # 時間：西元 2010 年 12 月 |

執行結果如下  


|  |  |
| --- | --- |
| **中英文術語對照** | |
| 數字型態 | numeric types |
| 整數型態 | integer type |
| 位元字串 | bit-string |

[內建型態](http://pydoing.blogspot.com/2011/03/python-builtinitype.html)

* [數字型態 int float complex](http://pydoing.blogspot.com/2011/02/python-numerictypes.html) 按一下展開目錄
* [迭代器型態](http://pydoing.blogspot.com/2011/03/python-iteratortype.html)
* [序列型態](http://pydoing.blogspot.com/2011/03/python-sequencetype.html) 按一下展開目錄
* [集合型態 set frozenset](http://pydoing.blogspot.com/2011/03/python-settype.html) 按一下展開目錄
* [配對型態 dict](http://pydoing.blogspot.com/2011/03/python-dicttype.html) 按一下展開目錄

4.4. Numeric Types — int, float, complex

There are three distinct numeric types: integers, floating point numbers, and complex numbers. In addition, Booleans are a subtype of integers. Integers have unlimited precision. Floating point numbers are usually implemented using double in C; information about the precision and internal representation of floating point numbers for the machine on which your program is running is available in sys.float\_info. Complex numbers have a real and imaginary part, which are each a floating point number. To extract these parts from a complex number z, use z.real and z.imag. (The standard library includes additional numeric types, fractions that hold rationals, and decimal that hold floating-point numbers with user-definable precision.)

Numbers are created by numeric literals or as the result of built-in functions and operators. Unadorned integer literals (including hex, octal and binary numbers) yield integers. Numeric literals containing a decimal point or an exponent sign yield floating point numbers. Appending 'j' or 'J' to a numeric literal yields an imaginary number (a complex number with a zero real part) which you can add to an integer or float to get a complex number with real and imaginary parts.

Python fully supports mixed arithmetic: when a binary arithmetic operator has operands of different numeric types, the operand with the “narrower” type is widened to that of the other, where integer is narrower than floating point, which is narrower than complex. Comparisons between numbers of mixed type use the same rule. [2] The constructors int(), float(), and complex() can be used to produce numbers of a specific type.

All numeric types (except complex) support the following operations, sorted by ascending priority (operations in the same box have the same priority; all numeric operations have a higher priority than comparison operations):

Operation

Result

Notes

Full documentation

x + y sum of x and y

x - y difference of x and y

x \* y product of x and y

x / y quotient of x and y

x // y floored quotient of x and y (1)

x % y remainder of x / y (2)

-x x negated

+x x unchanged

abs(x) absolute value or magnitude of x abs()

int(x) x converted to integer (3)(6) int()

float(x) x converted to floating point (4)(6) float()

complex(re, im) a complex number with real part re, imaginary part im. im defaults to zero. (6) complex()

c.conjugate() conjugate of the complex number c

divmod(x, y) the pair (x // y, x % y) (2) divmod()

pow(x, y) x to the power y (5) pow()

x \*\* y x to the power y (5)

Notes:

1.Also referred to as integer division. The resultant value is a whole integer, though the result’s type is not necessarily int. The result is always rounded towards minus infinity: 1//2 is 0, (-1)//2 is -1, 1//(-2) is -1, and (-1)//(-2) is 0.

2.Not for complex numbers. Instead convert to floats using abs() if appropriate.

3.Conversion from floating point to integer may round or truncate as in C; see functions math.floor() and math.ceil() for well-defined conversions.

4.float also accepts the strings “nan” and “inf” with an optional prefix “+” or “-” for Not a Number (NaN) and positive or negative infinity.

5.Python defines pow(0, 0) and 0 \*\* 0 to be 1, as is common for programming languages.

6.The numeric literals accepted include the digits 0 to 9 or any Unicode equivalent (code points with the Nd property).

See http://www.unicode.org/Public/6.0.0/ucd/extracted/DerivedNumericType.txt for a complete list of code points with the Nd property.

All numbers.Real types (int and float) also include the following operations:

Operation

Result

Notes

math.trunc(x) x truncated to Integral

round(x[, n]) x rounded to n digits, rounding half to even. If n is omitted, it defaults to 0.

math.floor(x) the greatest integral float <= x

math.ceil(x) the least integral float >= x

For additional numeric operations see the math and cmath modules.

4.4.1. Bitwise Operations on Integer Types

Bitwise operations only make sense for integers. Negative numbers are treated as their 2’s complement value (this assumes a sufficiently large number of bits that no overflow occurs during the operation).

The priorities of the binary bitwise operations are all lower than the numeric operations and higher than the comparisons; the unary operation ~ has the same priority as the other unary numeric operations (+ and -).

This table lists the bitwise operations sorted in ascending priority (operations in the same box have the same priority):

Operation

Result

Notes

x | y bitwise or of x and y

x ^ y bitwise exclusive or of x and y

x & y bitwise and of x and y

x << n x shifted left by n bits (1)(2)

x >> n x shifted right by n bits (1)(3)

~x the bits of x inverted

Notes:

1.Negative shift counts are illegal and cause a ValueError to be raised.

2.A left shift by n bits is equivalent to multiplication by pow(2, n) without overflow check.

3.A right shift by n bits is equivalent to division by pow(2, n) without overflow check.

4.4.2. Additional Methods on Integer Types

The int type implements the numbers.Integral abstract base class. In addition, it provides one more method:

int.bit\_length()

Return the number of bits necessary to represent an integer in binary, excluding the sign and leading zeros:

>>>>>> n = -37

>>> bin(n)

'-0b100101'

>>> n.bit\_length()

6

More precisely, if x is nonzero, then x.bit\_length() is the unique positive integer k such that 2\*\*(k-1) <= abs(x) < 2\*\*k. Equivalently, when abs(x) is small enough to have a correctly rounded logarithm, then k = 1 + int(log(abs(x), 2)). If x is zero, then x.bit\_length() returns 0.

Equivalent to:

def bit\_length(self):

s = bin(self) # binary representation: bin(-37) --> '-0b100101'

s = s.lstrip('-0b') # remove leading zeros and minus sign

return len(s) # len('100101') --> 6

New in version 3.1.

int.to\_bytes(length, byteorder, \*, signed=False)

Return an array of bytes representing an integer.

>>>>>> (1024).to\_bytes(2, byteorder='big')

b'\x04\x00'

>>> (1024).to\_bytes(10, byteorder='big')

b'\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00'

>>> (-1024).to\_bytes(10, byteorder='big', signed=True)

b'\xff\xff\xff\xff\xff\xff\xff\xff\xfc\x00'

>>> x = 1000

>>> x.to\_bytes((x.bit\_length() // 8) + 1, byteorder='little')

b'\xe8\x03'

The integer is represented using length bytes. An OverflowError is raised if the integer is not representable with the given number of bytes.

The byteorder argument determines the byte order used to represent the integer. If byteorder is "big", the most significant byte is at the beginning of the byte array. If byteorder is "little", the most significant byte is at the end of the byte array. To request the native byte order of the host system, use sys.byteorder as the byte order value.

The signed argument determines whether two’s complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised. The default value for signed is False.

New in version 3.2.

classmethod int.from\_bytes(bytes, byteorder, \*, signed=False)

Return the integer represented by the given array of bytes.

>>>>>> int.from\_bytes(b'\x00\x10', byteorder='big')

16

>>> int.from\_bytes(b'\x00\x10', byteorder='little')

4096

>>> int.from\_bytes(b'\xfc\x00', byteorder='big', signed=True)

-1024

>>> int.from\_bytes(b'\xfc\x00', byteorder='big', signed=False)

64512

>>> int.from\_bytes([255, 0, 0], byteorder='big')

16711680

The argument bytes must either be a bytes-like object or an iterable producing bytes.

The byteorder argument determines the byte order used to represent the integer. If byteorder is "big", the most significant byte is at the beginning of the byte array. If byteorder is "little", the most significant byte is at the end of the byte array. To request the native byte order of the host system, use sys.byteorder as the byte order value.

The signed argument indicates whether two’s complement is used to represent the integer.

New in version 3.2.

4.4.3. Additional Methods on Float

The float type implements the numbers.Real abstract base class. float also has the following additional methods.

float.as\_integer\_ratio()

Return a pair of integers whose ratio is exactly equal to the original float and with a positive denominator. Raises OverflowError on infinities and a ValueError on NaNs.

float.is\_integer()

Return True if the float instance is finite with integral value, and False otherwise:

>>>>>> (-2.0).is\_integer()

True

>>> (3.2).is\_integer()

False

Two methods support conversion to and from hexadecimal strings. Since Python’s floats are stored internally as binary numbers, converting a float to or from a decimal string usually involves a small rounding error. In contrast, hexadecimal strings allow exact representation and specification of floating-point numbers. This can be useful when debugging, and in numerical work.

float.hex()

Return a representation of a floating-point number as a hexadecimal string. For finite floating-point numbers, this representation will always include a leading 0x and a trailing p and exponent.

classmethod float.fromhex(s)

Class method to return the float represented by a hexadecimal string s. The string s may have leading and trailing whitespace.

Note that float.hex() is an instance method, while float.fromhex() is a class method.

A hexadecimal string takes the form:

[sign] ['0x'] integer ['.' fraction] ['p' exponent]

where the optional sign may by either + or -, integer and fraction are strings of hexadecimal digits, and exponent is a decimal integer with an optional leading sign. Case is not significant, and there must be at least one hexadecimal digit in either the integer or the fraction. This syntax is similar to the syntax specified in section 6.4.4.2 of the C99 standard, and also to the syntax used in Java 1.5 onwards. In particular, the output of float.hex() is usable as a hexadecimal floating-point literal in C or Java code, and hexadecimal strings produced by C’s %a format character or Java’s Double.toHexString are accepted by float.fromhex().

Note that the exponent is written in decimal rather than hexadecimal, and that it gives the power of 2 by which to multiply the coefficient. For example, the hexadecimal string 0x3.a7p10 represents the floating-point number (3 + 10./16 + 7./16\*\*2) \* 2.0\*\*10, or 3740.0:

>>>>>> float.fromhex('0x3.a7p10')

3740.0

Applying the reverse conversion to 3740.0 gives a different hexadecimal string representing the same number:

>>>>>> float.hex(3740.0)

'0x1.d380000000000p+11'

4.4.4. Hashing of numeric types

For numbers x and y, possibly of different types, it’s a requirement that hash(x) == hash(y) whenever x == y (see the \_\_hash\_\_() method documentation for more details). For ease of implementation and efficiency across a variety of numeric types (including int, float, decimal.Decimal and fractions.Fraction) Python’s hash for numeric types is based on a single mathematical function that’s defined for any rational number, and hence applies to all instances of int and fractions.Fraction, and all finite instances of float and decimal.Decimal. Essentially, this function is given by reduction modulo P for a fixed prime P. The value of P is made available to Python as the modulus attribute of sys.hash\_info.

CPython implementation detail: Currently, the prime used is P = 2\*\*31 - 1 on machines with 32-bit C longs and P = 2\*\*61 - 1 on machines with 64-bit C longs.

Here are the rules in detail:

•If x = m / n is a nonnegative rational number and n is not divisible by P, define hash(x) as m \* invmod(n, P) % P, where invmod(n, P) gives the inverse of n modulo P.

•If x = m / n is a nonnegative rational number and n is divisible by P (but m is not) then n has no inverse modulo P and the rule above doesn’t apply; in this case define hash(x) to be the constant value sys.hash\_info.inf.

•If x = m / n is a negative rational number define hash(x) as -hash(-x). If the resulting hash is -1, replace it with -2.

•The particular values sys.hash\_info.inf, -sys.hash\_info.inf and sys.hash\_info.nan are used as hash values for positive infinity, negative infinity, or nans (respectively). (All hashable nans have the same hash value.)

•For a complex number z, the hash values of the real and imaginary parts are combined by computing hash(z.real) + sys.hash\_info.imag \* hash(z.imag), reduced modulo 2\*\*sys.hash\_info.width so that it lies in range(-2\*\*(sys.hash\_info.width - 1), 2\*\*(sys.hash\_info.width - 1)). Again, if the result is -1, it’s replaced with -2.

To clarify the above rules, here’s some example Python code, equivalent to the built-in hash, for computing the hash of a rational number, float, or complex:

import sys, math

def hash\_fraction(m, n):

"""Compute the hash of a rational number m / n.

Assumes m and n are integers, with n positive.

Equivalent to hash(fractions.Fraction(m, n)).

"""

P = sys.hash\_info.modulus

# Remove common factors of P. (Unnecessary if m and n already coprime.)

while m % P == n % P == 0:

m, n = m // P, n // P

if n % P == 0:

hash\_ = sys.hash\_info.inf

else:

# Fermat's Little Theorem: pow(n, P-1, P) is 1, so

# pow(n, P-2, P) gives the inverse of n modulo P.

hash\_ = (abs(m) % P) \* pow(n, P - 2, P) % P

if m < 0:

hash\_ = -hash\_

if hash\_ == -1:

hash\_ = -2

return hash\_

def hash\_float(x):

"""Compute the hash of a float x."""

if math.isnan(x):

return sys.hash\_info.nan

elif math.isinf(x):

return sys.hash\_info.inf if x > 0 else -sys.hash\_info.inf

else:

return hash\_fraction(\*x.as\_integer\_ratio())

def hash\_complex(z):

"""Compute the hash of a complex number z."""

hash\_ = hash\_float(z.real) + sys.hash\_info.imag \* hash\_float(z.imag)

# do a signed reduction modulo 2\*\*sys.hash\_info.width

M = 2\*\*(sys.hash\_info.width - 1)

hash\_ = (hash\_ & (M - 1)) - (hash & M)

if hash\_ == -1:

hash\_ == -2

return hash\_

int(x)，將物件x轉換成一整數，或者是將x轉換整數。

int(s, base)，將str轉換成整數，if base引述有指定，則為一範圍2~36的整數。

float(x) 將物件x轉換成一浮點數，或者是將x轉換浮點數。