**Python - Numbers**

[Previous](https://www.tutorialspoint.com/python/python_user_input.htm)

[Quiz](https://www.tutorialspoint.com/python/quiz_on_python_numbers.htm)

[Next](https://www.tutorialspoint.com/python/python_booleans.htm)

[Python](https://www.tutorialspoint.com/python/index.htm) has built-in support to store and process numeric data (**Python Numbers**). Most of the times you work with numbers in almost every [Python application](https://www.tutorialspoint.com/python/python_application_areas.htm). Obviously, any computer application deals with numbers. This tutorial will discuss about different types of Python Numbers and their properties.

Python - Number Types

There are three built-in number types available in Python:

* integers **(int)**
* floating point numbers **(float)**
* **complex** numbers

Python also has a bult-in Boolean [data type](https://www.tutorialspoint.com/python/python_data_types.htm) called **bool**. It can be treated as a sub-type of **int** type, since it's two possible values **True** and **False** represent the integers 1 and 0 respectively.

Advertisement

-[PS](https://playstream.media/)

PauseSkip backward 5 secondsSkip forward 5 seconds

Mute

Fullscreen

Python − Integer Numbers

In Python, any number without the provision to store a fractional part is an integer. (Note that if the fractional part in a number is 0, it doesn't mean that it is an integer. For example a number 10.0 is not an integer, it is a float with 0 fractional part whose numeric value is 10.) An integer can be zero, positive or a negative whole number. For example, 1234, 0, -55 all represent to integers in Python.

There are three ways to form an integer object. With (a) literal representation, (b) any expression evaluating to an integer, and (c) using **int()** function.

Literal is a notation used to represent a constant directly in the source code. For example −

>>> a =10

However, look at the following assignment of the integer variable *c*.

Open Compiler

a = 10

b = 20

c = a + b

print ("a:", a, "type:", type(a))

print ("c:", c, "type:", type(c))

It will produce the following **output** −

a: 10 type: <class 'int'>

c: 30 type: <class 'int'>

Here, *c* is indeed an integer variable, but the expression *a + b* is evaluated first, and its value is indirectly assigned to *c*.

The third method of forming an integer object is with the return value of int() function. It converts a floating point number or a [string](https://www.tutorialspoint.com/python/python_strings.htm) in an integer.

>>> a=int(10.5)

>>> b=int("100")

You can represent an integer as a binary, octal or Hexa-decimal number. However, internally the object is stored as an integer.

Binary Numbers in Python

A number consisting of only the binary digits (1 and 0) and prefixed with **"0b"** is a binary number. If you assign a binary number to a variable, it still is an int variable.

A represent an integer in binary form, store it directly as a literal, or use int() function, in which the base is set to 2

Open Compiler

a=0b101

print ("a:",a, "type:",type(a))

b=int("0b101011", 2)

print ("b:",b, "type:",type(b))

It will produce the following **output** −

a: 5 type: <class 'int'>

b: 43 type: <class 'int'>

There is also a **bin()** function in Python. It returns a binary string equivalent of an integer.

Open Compiler

a=43

b=bin(a)

print ("Integer:",a, "Binary equivalent:",b)

It will produce the following **output** −

Integer: 43 Binary equivalent: 0b101011

Octal Numbers in Python

An octal number is made up of digits 0 to 7 only. In order to specify that the integer uses octal notation, it needs to be prefixed by **"0o"** (lowercase O) or **"0O"** (uppercase O). A literal representation of octal number is as follows −

Open Compiler

a=0O107

print (a, type(a))

It will produce the following **output** −

71 <class 'int'>

Note that the object is internally stored as integer. Decimal equivalent of octal number 107 is 71.

Since octal number system has 8 symbols (0 to 7), its base is 7. Hence, while using int() function to covert an octal string to integer, you need to set the base argument to 8.

Open Compiler

a=int('20',8)

print (a, type(a))

It will produce the following **output** −

16 <class 'int'>

Decimal equivalent of octal 30 is 16.

In the following code, two int objects are obtained from octal notations and their addition is performed.

Open Compiler

a=0O56

print ("a:",a, "type:",type(a))

b=int("0O31",8)

print ("b:",b, "type:",type(b))

c=a+b

print ("addition:", c)

It will produce the following **output** −

a: 46 type: <class 'int'>

b: 25 type: <class 'int'>

addition: 71

To obtain the octal string for an integer, use **oct()** function.

Open Compiler

a=oct(71)

print (a, type(a))

Hexa-decimal Numbers in Python

As the name suggests, there are 16 symbols in the Hexadecimal number system. They are 0-9 and A to F. The first 10 digits are same as decimal digits. The alphabets A, B, C, D, E and F are equivalents of 11, 12, 13, 14, 15, and 16 respectively. Upper or lower cases may be used for these letter symbols.

For the literal representation of an integer in Hexadecimal notation, prefix it by **"0x"** or **"0X"**.

Open Compiler

a=0XA2

print (a, type(a))

It will produce the following **output** −

162 <class 'int'>

To convert a Hexadecimal string to integer, set the base to 16 in the **int()** function.

Open Compiler

a=int('0X1e', 16)

print (a, type(a))

Try out the following code snippet. It takes a Hexadecimal string, and returns the integer.

Open Compiler

num\_string = "A1"

number = int(num\_string, 16)

print ("Hexadecimal:", num\_string, "Integer:",number)

It will produce the following **output** −

Hexadecimal: A1 Integer: 161

However, if the string contains any symbol apart from the Hexadecimal symbol chart an error will be generated.

Open Compiler

num\_string = "A1X001"

print (int(num\_string, 16))

The above program generates the following error −

Traceback (most recent call last):

File "/home/main.py", line 2, in

print (int(num\_string, 16))

ValueError: invalid literal for int() with base 16: 'A1X001'

Python's standard library has **hex()** function, with which you can obtain a hexadecimal equivalent of an integer.

Open Compiler

a=hex(161)

print (a, type(a))

It will produce the following **output** −

0xa1 <class 'str'>

Though an integer can be represented as binary or octal or hexadecimal, internally it is still integer. So, when performing arithmetic operation, the representation doesn't matter.

Open Compiler

a=10 #decimal

b=0b10 #binary

c=0O10 #octal

d=0XA #Hexadecimal

e=a+b+c+d

print ("addition:", e)

It will produce the following **output** −

addition: 30

Python − Floating Point Numbers

A floating point number has an integer part and a fractional part, separated by a decimal point symbol (.). By default, the number is positive, prefix a dash (-) symbol for a negative number.

A floating point number is an object of Python's float class. To store a float object, you may use a literal notation, use the value of an arithmetic expression, or use the return value of float() function.

Using literal is the most direct way. Just assign a number with fractional part to a variable. Each of the following statements declares a float object.

>>> a=9.99

>>> b=0.999

>>> c=-9.99

>>> d=-0.999

In Python, there is no restriction on how many digits after the decimal point can a floating point number have. However, to shorten the representation, the **E** or **e** symbol is used. E stands for Ten raised to. For example, E4 is 10 raised to 4 (or 4th power of 10), e-3 is 10 raised to -3.

In scientific notation, number has a coefficient and exponent part. The coefficient should be a float greater than or equal to 1 but less than 10. Hence, 1.23E+3, 9.9E-5, and 1E10 are the examples of floats with scientific notation.

>>> a=1E10

>>> a

10000000000.0

>>> b=9.90E-5

>>> b

9.9e-05

>>> 1.23E3

1230.0

The second approach of forming a float object is indirect, using the result of an expression. Here, the quotient of two floats is assigned to a variable, which refers to a float object.

Open Compiler

a=10.33

b=2.66

c=a/b

print ("c:", c, "type", type(c))

It will produce the following **output** −

c: 3.8834586466165413 type <class 'float'>

Python's float() function returns a float object, parsing a number or a string if it has the appropriate contents. If no arguments are given in the parenthesis, it returns 0.0, and for an **int** argument, fractional part with 0 is added.

>>> a=float()

>>> a

0.0

>>> a=float(10)

>>> a

10.0

Even if the integer is expressed in binary, octal or hexadecimal, the float() function returns a float with fractional part as 0.

Open Compiler

a=float(0b10)

b=float(0O10)

c=float(0xA)

print (a,b,c, sep=",")

It will produce the following **output** −

2.0,8.0,10.0

The **float()** function retrieves a floating point number out of a string that encloses a float, either in standard decimal point format, or having scientific notation.

Open Compiler

a=float("-123.54")

b=float("1.23E04")

print ("a=",a,"b=",b)

It will produce the following **output** −

a= -123.54 b= 12300.0

In mathematics, infinity is an abstract concept. Physically, infinitely large number can never be stored in any amount of memory. For most of the computer hardware configurations, however, a very large number with 400th power of 10 is represented by Inf. If you use "Infinity" as argument for float() function, it returns Inf.

Open Compiler

a=1.00E400

print (a, type(a))

a=float("Infinity")

print (a, type(a))

It will produce the following **output** −

inf <class 'float'>

inf <class 'float'>

One more such entity is Nan (stands for Not a Number). It represents any value that is undefined or not representable.

>>> a=float('Nan')

>>> a

Nan

Python − Complex Numbers

In this section, we shall know in detail about Complex data type in Python. Complex numbers find their applications in mathematical equations and laws in electromagnetism, electronics, optics, and quantum theory. Fourier transforms use complex numbers. They are Used in calculations with wavefunctions, designing filters, signal integrity in digital electronics, radio astronomy, etc.

A complex number consists of a real part and an imaginary part, separated by either "+" or "−". The real part can be any floating point (or itself a complex number) number. The imaginary part is also a float/complex, but multiplied by an imaginary number.

In mathematics, an imaginary number "i" is defined as the square root of -1 (&bsol;sqrt{−1}&bsol;sqrt{−1}). Therefore, a complex number is represented as "x+yi", where x is the real part, and "y" is the coefficient of imaginary part.

Quite often, the symbol "j" is used instead of "I" for the imaginary number, to avoid confusion with its usage as current in theory of electricity. Python also uses "j" as the imaginary number. Hence, "x+yj" is the representation of complex number in Python.

Like int or float data type, a complex object can be formed with literal representation or using complex() function. All the following statements form a complex object.

>>> a=5+6j

>>> a

(5+6j)

>>> type(a)

<class 'complex'>

>>> a=2.25-1.2J

>>> a

(2.25-1.2j)

>>> type(a)

<class 'complex'>

>>> a=1.01E-2+2.2e3j

>>> a

(0.0101+2200j)

>>> type(a)

<class 'complex'>

Note that the real part as well as the coefficient of imaginary part have to be floats, and they may be expressed in standard decimal point notation or scientific notation.

Python's **complex()** function helps in forming an object of complex type. The function receives arguments for real and imaginary part, and returns the complex number.

There are two versions of complex() function, with two arguments and with one argument. Use of complex() with two arguments is straightforward. It uses first argument as real part and second as coefficient of imaginary part.

Open Compiler

a=complex(5.3,6)

b=complex(1.01E-2, 2.2E3)

print ("a:", a, "type:", type(a))

print ("b:", b, "type:", type(b))

It will produce the following **output** −

a: (5.3+6j) type: <class 'complex'>

b: (0.0101+2200j) type: <class 'complex'>

In the above example, we have used x and y as float parameters. They can even be of complex data type.

Open Compiler

a=complex(1+2j, 2-3j)

print (a, type(a))

It will produce the following **output** −

(4+4j) <class 'complex'>

Surprised by the above example? Put "x" as 1+2j and "y" as 2-3j. Try to perform manual computation of "x+yj" and you'll come to know.

complex(1+2j, 2-3j)

=(1+2j)+(2-3j)\*j

=1+2j +2j+3

=4+4j

If you use only one numeric argument for complex() function, it treats it as the value of real part; and imaginary part is set to 0.

Open Compiler

a=complex(5.3)

print ("a:", a, "type:", type(a))

It will produce the following **output** −

a: (5.3+0j) type: <class 'complex'>

The complex() function can also parse a string into a complex number if its only argument is a string having complex number representation.

In the following snippet, user is asked to input a complex number. It is used as argument. Since Python reads the input as a string, the function extracts the complex object from it.

Open Compiler

a= "5.5+2.3j"

b=complex(a)

print ("Complex number:", b)

It will produce the following **output** −

Complex number: (5.5+2.3j)

Python's built-in complex class has two attributes **real** and **imag** − they return the real and coefficient of imaginary part from the object.

Open Compiler

a=5+6j

print ("Real part:", a.real, "Coefficient of Imaginary part:", a.imag)

It will produce the following **output** −

Real part: 5.0 Coefficient of Imaginary part: 6.0

The complex class also defines a conjugate() method. It returns another complex number with the sign of imaginary component reversed. For example, conjugate of x+yj is x-yj.

>>> a=5-2.2j

>>> a.conjugate()

(5+2.2j)

Number Type Conversion

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you need to coerce a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.

* Type [int(x)](https://www.tutorialspoint.com/python/python-int-function.htm) to convert x to a plain integer.
* Type [long(x)](https://www.tutorialspoint.com/python/python-long-function.htm) to convert x to a long integer.
* Type [float(x)](https://www.tutorialspoint.com/python/python-float-function.htm) to convert x to a floating-point number.
* Type [complex(x)](https://www.tutorialspoint.com/python/python-complex-function.htm) to convert x to a complex number with real part x and imaginary part zero. In the same way type **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions