Learn Discover Contribute More Insiders

Tracks / Python / Syllabus / Numbers

Nu

Numbers in

5 exercises

✓ You've mastered Numbers in Python. Explore more concepts →

About Numbers

Python has three different types of built-in numbers: integers (`int`), floating-point (`float`), and complex (`complex`). Fractions (`fractions.Fraction`) and Decimals (`decimal.Decimal`) are also available via import from the standard library.

Whole numbers including hexadecimal (*`hex()`*), octal (*`oct()`*) and binary (*`bin()`*) numbers **without** decimal places are also identified as `ints` :

```
# Ints are whole numbers.
>>> 1234
```

```
1234
>>> type(1234)
<class 'int'>

>>> -12
-12
```

Numbers containing a decimal point (with or without fractional parts) are identified as `floats`:

```
>>> 3.45
3.45
>>> type(3.45)
<class 'float'>
```

Arithmetic

Python fully supports arithmetic between these different number types, and will convert narrower numbers to match their less narrow counterparts when used with the binary arithmetic operators (`+`, `-`, `*`, `/`, `//`, and `%`).

All numbers (except complex) support all **arithmetic operations**, evaluated according to **operator precedence**. Support for mathematical functions (beyond `+` and `-`) for complex numbers can be found in the **cmath** module.

Addition and subtraction

Addition and subtraction operators behave as they do in normal math. If one or more of the operands is a `float`, the remaining `int` s will be converted to `float` s as well:

```
>>> 5 - 3
2
# The int is widened to a float here, and a float is returned.
>>> 3 + 4.0
7.0
```

Multiplication

As with addition and subtraction, multiplication will convert narrower numbers to match their less narrow counterparts:

```
>>> 3 * 2
6

>>> 3 * 2.0
6.0
```

Division

Division always returns a `float`, even if the result is a whole number:

```
>>> 6/5
1.2
```

```
>>> 6/2
3.0
```

Floor division

If an `int` result is needed, you can use floor division to truncate the result. Floor division is performed using the `//` operator:

```
>>> 6//5
1

>>> 6//2
3
```

Modulo

The modulo operator (`%`) returns the remainder of the division of the two operands:

```
# The result of % is zero here, because dividing 8 by 2 leaves no remainder
>>> 8 % 2
0

# The result of % is 2 here, because 3 only goes into 5 once, with 2 left over
```

```
>>> 5 % 3
2
```

Another way to look at 5 % 3:

```
>>> whole_part = int(5/3)
1

>>> decimal_part = 5/3 - whole_part
0.6666666666666667

>>> whole_remainder = decimal_part * 3
2.0
```

Exponentiation

Exponentiation is performed using the `**` operator:

```
>>> 2 ** 3
8

>>> 4 ** 0.5
2
```

Conversions

Numbers can be converted from `int` to `floats` and `floats` to `int` using the built-in functions `int()` and `float()` :

```
>>> int(3.45)
3

>>> float(3)
3.0
```

Round

Python provides a built-in function **`round(number, <decimal_places>)`** to round off a floating point number to a given number of decimal places. If no number of decimal places is specified, the number is rounded off to the nearest integer and will return an `int` :

```
>>> round(3.1415926535, 2)
3.14

>>> round(3.1415926535)
3
```

Priority and parentheses

Python allows you to use parentheses to group expressions. This is useful when you want to override the default order of operations.

```
>>> 2 + 3 * 4
14

>>> (2 + 3) * 4
20
```

Python follows the **PEMDAS** rule for operator precedence. This means calculations within `()` have the highest priority, followed by `**`, then `*`, `/`, `//`, `%`, `+`, and `-`:

```
>>> 2 + 3 - 4 * 4
-11

>>> (2 + 3 - 4) * 4
4

# In the following example, the `**` operator has the highest priority, then `*
# Meaning we first do 4 ** 4, then 3 * 256, then 2 + 768
>>> 2 + 3 * 4 ** 4
770
```

Precision & Representation

Integers in Python have **arbitrary precision** -- the number of digits is limited only by the available memory of the host system.

Floating point numbers are usually implemented using a `double` in C (*15 decimal places of precision*), but will vary in representation based on the host system. Complex numbers have a `real` and an `imaginary` part, both of which are represented by floating point numbers.

For a more detailed discussions of the issues and limitations of floating point arithmetic across programming languages, take a look at **0.30000000000000004.com** and **The Python Tutorial**.

Learn More

integers

floats

complex numbers

fractions

Decimals

Arithmetic Operations

Operator Precedence

cmath: mathematical operations for complex numbers

Pythons Numerical and Mathematical Modules

5 authors
3 contributors

Edit via GitHub