# 2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

| **Built-in Functions** | | | |
| --- | --- | --- | --- |
| **函数名称** | **传入值** | **作用** | |
| [abs()](https://docs.python.org/3/library/functions.html#abs) | 1数字 | 绝对值 |  |
| [all()](https://docs.python.org/3/library/functions.html#all) | \*可迭代对象 | 可迭代对象中所有元素都是真才返回真，否则返回假。 |  |
| [any()](https://docs.python.org/3/library/functions.html#any) | \*可迭代对象 | 可迭代对象有一个真的就是真的。 |  |
| [ascii()](https://docs.python.org/3/library/functions.html#ascii) | 无要求 | 返回对象的\_\_repr\_\_() |  |
| [bin()](https://docs.python.org/3/library/functions.html#bin) | 1整形 | 返回二进制字节 |  |
| [bool()](https://docs.python.org/3/library/functions.html#bool) | 无要求 | 返回布尔值 |  |
| [bytearray()](https://docs.python.org/3/library/functions.html#bytearray) |  | 转化成二进制字节，汉字要加上encoding=’utf-8’或其他 |  |
| [bytes()](https://docs.python.org/3/library/functions.html#bytes) |  | 转化成二进制字符 |  |
| [callable()](https://docs.python.org/3/library/functions.html#callable) | 无要求 | 是否可执行，类中有\_\_call\_\_ |  |
| [chr()](https://docs.python.org/3/library/functions.html#chr) | 1数字 | 把ascii码转化成字符 |  |
| [classmethod()](https://docs.python.org/3/library/functions.html#classmethod) |  | 调用类的类方法 |  |
| [compile()](https://docs.python.org/3/library/functions.html#compile) | 字符串 | 把字符串变成Python代码 |  |
| [complex()](https://docs.python.org/3/library/functions.html#complex) |  |  |  |
| [delattr()](https://docs.python.org/3/library/functions.html#delattr) |  |  |  |
| [dict()](https://docs.python.org/3/library/functions.html#func-dict) |  | 字典 |  |
| [dir()](https://docs.python.org/3/library/functions.html#dir) |  | 当前类的全部变量（属性方法） |  |
| [divmod()](https://docs.python.org/3/library/functions.html#divmod) | 两个数字 | 商，余 |  |
| [enumerate()](https://docs.python.org/3/library/functions.html#enumerate) |  | 给可迭代对象加上序号 | >>> a = ['alex', 'eric', 'lily']  >>> for e, b in enumerate(a, 1): print(e, b)  1 alex  2 eric  3 lily |
| [eval()](https://docs.python.org/3/library/functions.html#eval) |  | 把字符串表达式转换成值 | >>> eval("5\*7")  35 |
| [exec()](https://docs.python.org/3/library/functions.html#exec) |  |  |  |
| [filter()](https://docs.python.org/3/library/functions.html#filter) |  | 详见python学习笔记 |  |
| [float()](https://docs.python.org/3/library/functions.html#float) |  | 转化成浮点 |  |
| [format()](https://docs.python.org/3/library/functions.html#format) |  | 返回类的\_\_format\_\_ |  |
| [frozenset()](https://docs.python.org/3/library/functions.html#func-frozenset) |  | 不能改变元素的集合 |  |
| [getattr()](https://docs.python.org/3/library/functions.html#getattr) |  |  |  |
| [globals()](https://docs.python.org/3/library/functions.html#globals) |  | 当前的所有全局变量 |  |
| [hasattr()](https://docs.python.org/3/library/functions.html#hasattr) |  |  |  |
| [hash()](https://docs.python.org/3/library/functions.html#hash) |  | 不可哈希类型对象的哈希值（相当于给你编个号） |  |
| [help()](https://docs.python.org/3/library/functions.html#help) |  |  |  |
| [hex()](https://docs.python.org/3/library/functions.html#hex) |  | 十六进制 |  |
| [id()](https://docs.python.org/3/library/functions.html#id) |  |  |  |
| [input()](https://docs.python.org/3/library/functions.html#input) |  | 输入框 |  |
| [int()](https://docs.python.org/3/library/functions.html#int) |  |  |  |
| [isinstance()](https://docs.python.org/3/library/functions.html#isinstance) |  |  |  |
| [issubclass()](https://docs.python.org/3/library/functions.html#issubclass) |  |  |  |
| [iter()](https://docs.python.org/3/library/functions.html#iter) |  |  |  |
| [len()](https://docs.python.org/3/library/functions.html#len) |  |  |  |
| [list()](https://docs.python.org/3/library/functions.html#func-list) |  |  |  |
| [locals()](https://docs.python.org/3/library/functions.html#locals) |  | 显示当前对象的局部变量 |  |
| [map()](https://docs.python.org/3/library/functions.html#map) |  |  |  |
| [max()](https://docs.python.org/3/library/functions.html#max) |  | 最大值 |  |
| [memoryview()](https://docs.python.org/3/library/functions.html#func-memoryview) |  |  |  |
| [min()](https://docs.python.org/3/library/functions.html#min) |  |  |  |
| [next()](https://docs.python.org/3/library/functions.html#next) |  |  |  |
| [object()](https://docs.python.org/3/library/functions.html#object) |  |  |  |
| [oct()](https://docs.python.org/3/library/functions.html#oct) |  | 八进制 |  |
| [open()](https://docs.python.org/3/library/functions.html#open) |  |  |  |
| [ord()](https://docs.python.org/3/library/functions.html#ord) | 1数字 | 把字符转化成ascii码 |  |
| [pow()](https://docs.python.org/3/library/functions.html#pow) |  |  |  |
| [print()](https://docs.python.org/3/library/functions.html#print) |  |  |  |
| [property()](https://docs.python.org/3/library/functions.html#property) |  |  |  |
| [range()](https://docs.python.org/3/library/functions.html#func-range) |  |  |  |
| [repr()](https://docs.python.org/3/library/functions.html#repr) |  |  |  |
| [reversed()](https://docs.python.org/3/library/functions.html#reversed) |  | 反转 |  |
| [round()](https://docs.python.org/3/library/functions.html#round) |  | 四舍五入 |  |
| [set()](https://docs.python.org/3/library/functions.html#func-set) |  |  |  |
| [setattr()](https://docs.python.org/3/library/functions.html#setattr) |  |  |  |
| [slice()](https://docs.python.org/3/library/functions.html#slice) |  | 切片 |  |
| [sorted()](https://docs.python.org/3/library/functions.html#sorted) |  | 排序 |  |
| [staticmethod()](https://docs.python.org/3/library/functions.html#staticmethod) |  | 静态方法 |  |
| [str()](https://docs.python.org/3/library/functions.html#func-str) |  | 输出字符串的方法 |  |
| [sum()](https://docs.python.org/3/library/functions.html#sum) |  | 求和 |  |
| [super()](https://docs.python.org/3/library/functions.html#super) |  | 父类 |  |
| [tuple()](https://docs.python.org/3/library/functions.html#func-tuple) |  | 元组 |  |
| [type()](https://docs.python.org/3/library/functions.html#type) |  |  |  |
| [vars()](https://docs.python.org/3/library/functions.html#vars) |  | Dir()的字典形式 |  |
| [zip()](https://docs.python.org/3/library/functions.html#zip) |  |  |  |
| [\_\_import\_\_()](https://docs.python.org/3/library/functions.html#__import__) |  |  |  |

**abs**(x)

Return the absolute value of a number. The argument may be an integer or a floating point number. If the argument is a complex number, its magnitude is returned.

**all**(iterable)

Return True if all elements of the iterable are true (or if the iterable is empty).

Equivalent to:

**def** all(iterable):

**for** element **in** iterable:

**if** **not** element:

**return** **False**

**return** **True**

**any**(iterable)

Return True if any element of the iterable is true. If the iterable is empty, return False. Equivalent to:

**def** any(iterable):

**for** element **in** iterable:

**if** element:

**return** **True**

**return** **False**

**ascii**(object)

As [repr()](https://docs.python.org/3/library/functions.html#repr), return a string containing a printable representation of an object, but escape the non-ASCII characters in the string returned by [repr()](https://docs.python.org/3/library/functions.html#repr)using \x, \u or \U escapes. This generates a string similar to that returned by [repr()](https://docs.python.org/3/library/functions.html#repr) in Python 2.

**bin**(x)

Convert an integer number to a binary string. The result is a valid Python expression. If x is not a Python [int](https://docs.python.org/3/library/functions.html#int) object, it has to define an[\_\_index\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__index__) method that returns an integer.

把整形转换成二进制字符串

class**bool**([x])

Return a Boolean value, i.e. one of True or False. x is converted using the standard [truth testing procedure](https://docs.python.org/3/library/stdtypes.html#truth). If x is false or omitted, this returnsFalse; otherwise it returns True. The [bool](https://docs.python.org/3/library/functions.html#bool) class is a subclass of [int](https://docs.python.org/3/library/functions.html#int) (see [Numeric Types — int, float, complex](https://docs.python.org/3/library/stdtypes.html#typesnumeric)). It cannot be subclassed further. Its only instances are False and True (see [Boolean Values](https://docs.python.org/3/library/stdtypes.html#bltin-boolean-values)).

返回布尔值

class**bytearray**([source[, encoding[, errors]]])

Return a new array of bytes. The [bytearray](https://docs.python.org/3/library/functions.html#bytearray) class is a mutable sequence of integers in the range 0 <= x < 256. It has most of the usual methods of mutable sequences, described in [Mutable Sequence Types](https://docs.python.org/3/library/stdtypes.html#typesseq-mutable), as well as most methods that the [bytes](https://docs.python.org/3/library/functions.html#bytes) type has, see [Bytes and Bytearray Operations](https://docs.python.org/3/library/stdtypes.html#bytes-methods).

The optional source parameter can be used to initialize the array in a few different ways:

* If it is a string, you must also give the encoding (and optionally, errors) parameters; [bytearray()](https://docs.python.org/3/library/functions.html#bytearray) then converts the string to bytes using[str.encode()](https://docs.python.org/3/library/stdtypes.html#str.encode).
* If it is an integer, the array will have that size and will be initialized with null bytes.
* If it is an object conforming to the buffer interface, a read-only buffer of the object will be used to initialize the bytes array.
* If it is an iterable, it must be an iterable of integers in the range 0 <= x < 256, which are used as the initial contents of the array.

Without an argument, an array of size 0 is created.

See also [Binary Sequence Types — bytes, bytearray, memoryview](https://docs.python.org/3/library/stdtypes.html#binaryseq) and [Bytearray Objects](https://docs.python.org/3/library/stdtypes.html#typebytearray).

class**bytes**([source[, encoding[, errors]]])

Return a new “bytes” object, which is an immutable sequence of integers in the range 0 <= x < 256. [bytes](https://docs.python.org/3/library/functions.html#bytes) is an immutable version of [bytearray](https://docs.python.org/3/library/functions.html#bytearray)– it has the same non-mutating methods and the same indexing and slicing behavior.

Accordingly, constructor arguments are interpreted as for [bytearray()](https://docs.python.org/3/library/functions.html#bytearray).

Bytes objects can also be created with literals, see [String and Bytes literals](https://docs.python.org/3/reference/lexical_analysis.html#strings).

See also [Binary Sequence Types — bytes, bytearray, memoryview](https://docs.python.org/3/library/stdtypes.html#binaryseq), [Bytes](https://docs.python.org/3/library/stdtypes.html#typebytes), and [Bytes and Bytearray Operations](https://docs.python.org/3/library/stdtypes.html#bytes-methods).

**callable**(object)

Return [True](https://docs.python.org/3/library/constants.html#True) if the object argument appears callable, [False](https://docs.python.org/3/library/constants.html#False) if not. If this returns true, it is still possible that a call fails, but if it is false, calling objectwill never succeed. Note that classes are callable (calling a class returns a new instance); instances are callable if their class has a [\_\_call\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__call__)method.

*New in version 3.2:*This function was first removed in Python 3.0 and then brought back in Python 3.2.

**chr**(i)

Return the string representing a character whose Unicode code point is the integer i. For example, chr(97) returns the string 'a', while chr(8364)returns the string '€'. This is the inverse of [ord()](https://docs.python.org/3/library/functions.html#ord).

The valid range for the argument is from 0 through 1,114,111 (0x10FFFF in base 16). [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) will be raised if i is outside that range.

**classmethod**(function)

Return a class method for function.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

**class** **C**:

**@classmethod**

**def** f(cls, arg1, arg2, ...): ...

The @classmethod form is a function [decorator](https://docs.python.org/3/glossary.html#term-decorator) – see the description of function definitions in [Function definitions](https://docs.python.org/3/reference/compound_stmts.html#function) for details.

It can be called either on the class (such as C.f()) or on an instance (such as C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see [staticmethod()](https://docs.python.org/3/library/functions.html#staticmethod) in this section.

For more information on class methods, consult the documentation on the standard type hierarchy in [The standard type hierarchy](https://docs.python.org/3/reference/datamodel.html#types).

**compile**(source, filename, mode, flags=0, dont\_inherit=False, optimize=-1)

Compile the source into a code or AST object. Code objects can be executed by [exec()](https://docs.python.org/3/library/functions.html#exec) or [eval()](https://docs.python.org/3/library/functions.html#eval). source can either be a normal string, a byte string, or an AST object. Refer to the [ast](https://docs.python.org/3/library/ast.html#module-ast) module documentation for information on how to work with AST objects.

The filename argument should give the file from which the code was read; pass some recognizable value if it wasn’t read from a file ('<string>'is commonly used).

The mode argument specifies what kind of code must be compiled; it can be 'exec' if source consists of a sequence of statements, 'eval' if it consists of a single expression, or 'single' if it consists of a single interactive statement (in the latter case, expression statements that evaluate to something other than None will be printed).

The optional arguments flags and dont\_inherit control which future statements (see [**PEP 236**](https://www.python.org/dev/peps/pep-0236)) affect the compilation of source. If neither is present (or both are zero) the code is compiled with those future statements that are in effect in the code that is calling [compile()](https://docs.python.org/3/library/functions.html#compile). If the flags argument is given and dont\_inherit is not (or is zero) then the future statements specified by the flags argument are used in addition to those that would be used anyway. If dont\_inherit is a non-zero integer then the flags argument is it – the future statements in effect around the call to compile are ignored.

Future statements are specified by bits which can be bitwise ORed together to specify multiple statements. The bitfield required to specify a given feature can be found as the compiler\_flag attribute on the \_Feature instance in the [\_\_future\_\_](https://docs.python.org/3/library/__future__.html#module-__future__) module.

The argument optimize specifies the optimization level of the compiler; the default value of -1 selects the optimization level of the interpreter as given by [-O](https://docs.python.org/3/using/cmdline.html#cmdoption-O) options. Explicit levels are 0 (no optimization; \_\_debug\_\_ is true), 1 (asserts are removed, \_\_debug\_\_ is false) or 2 (docstrings are removed too).

This function raises [SyntaxError](https://docs.python.org/3/library/exceptions.html#SyntaxError) if the compiled source is invalid, and [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) if the source contains null bytes.

If you want to parse Python code into its AST representation, see [ast.parse()](https://docs.python.org/3/library/ast.html#ast.parse).

**Note**

When compiling a string with multi-line code in 'single' or 'eval' mode, input must be terminated by at least one newline character. This is to facilitate detection of incomplete and complete statements in the [code](https://docs.python.org/3/library/code.html#module-code) module.

*Changed in version 3.2:*Allowed use of Windows and Mac newlines. Also input in 'exec' mode does not have to end in a newline anymore. Added the optimize parameter.

*Changed in version 3.5:*Previously, [TypeError](https://docs.python.org/3/library/exceptions.html#TypeError) was raised when null bytes were encountered in source.

class**complex**([real[, imag]])

Return a complex number with the value real + imag\*1j or convert a string or number to a complex number. If the first parameter is a string, it will be interpreted as a complex number and the function must be called without a second parameter. The second parameter can never be a string. Each argument may be any numeric type (including complex). If imag is omitted, it defaults to zero and the constructor serves as a numeric conversion like [int](https://docs.python.org/3/library/functions.html#int) and [float](https://docs.python.org/3/library/functions.html#float). If both arguments are omitted, returns 0j.

**Note**

When converting from a string, the string must not contain whitespace around the central + or - operator. For example,complex('1+2j') is fine, but complex('1 + 2j') raises [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError).

The complex type is described in [Numeric Types — int, float, complex](https://docs.python.org/3/library/stdtypes.html#typesnumeric).

*Changed in version 3.6:*Grouping digits with underscores as in code literals is allowed.

**delattr**(object, name)

This is a relative of [setattr()](https://docs.python.org/3/library/functions.html#setattr). The arguments are an object and a string. The string must be the name of one of the object’s attributes. The function deletes the named attribute, provided the object allows it. For example, delattr(x, 'foobar') is equivalent to del x.foobar.

class**dict**(\*\*kwarg)

class**dict**(mapping, \*\*kwarg)

class**dict**(iterable, \*\*kwarg)

Create a new dictionary. The [dict](https://docs.python.org/3/library/stdtypes.html#dict) object is the dictionary class. See [dict](https://docs.python.org/3/library/stdtypes.html#dict) and [Mapping Types — dict](https://docs.python.org/3/library/stdtypes.html#typesmapping) for documentation about this class.

For other containers see the built-in [list](https://docs.python.org/3/library/stdtypes.html#list), [set](https://docs.python.org/3/library/stdtypes.html#set), and [tuple](https://docs.python.org/3/library/stdtypes.html#tuple) classes, as well as the [collections](https://docs.python.org/3/library/collections.html#module-collections) module.

**dir**([object])

Without arguments, return the list of names in the current local scope. With an argument, attempt to return a list of valid attributes for that object.

If the object has a method named [\_\_dir\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__dir__), this method will be called and must return the list of attributes. This allows objects that implement a custom [\_\_getattr\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getattr__) or [\_\_getattribute\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getattribute__) function to customize the way [dir()](https://docs.python.org/3/library/functions.html#dir) reports their attributes.

If the object does not provide [\_\_dir\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__dir__), the function tries its best to gather information from the object’s [\_\_dict\_\_](https://docs.python.org/3/library/stdtypes.html#object.__dict__) attribute, if defined, and from its type object. The resulting list is not necessarily complete, and may be inaccurate when the object has a custom [\_\_getattr\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getattr__).

The default [dir()](https://docs.python.org/3/library/functions.html#dir) mechanism behaves differently with different types of objects, as it attempts to produce the most relevant, rather than complete, information:

* If the object is a module object, the list contains the names of the module’s attributes.
* If the object is a type or class object, the list contains the names of its attributes, and recursively of the attributes of its bases.
* Otherwise, the list contains the object’s attributes’ names, the names of its class’s attributes, and recursively of the attributes of its class’s base classes.

The resulting list is sorted alphabetically. For example:

>>>

**>>> import** **struct**

**>>>** dir() *# show the names in the module namespace*

['\_\_builtins\_\_', '\_\_name\_\_', 'struct']

**>>>** dir(struct) *# show the names in the struct module*

['Struct', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_',

'\_\_initializing\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_',

'\_clearcache', 'calcsize', 'error', 'pack', 'pack\_into',

'unpack', 'unpack\_from']

**>>> class** **Shape**:

**...**  **def** \_\_dir\_\_(self):

**...**  **return** ['area', 'perimeter', 'location']

**>>>** s = Shape()

**>>>** dir(s)

['area', 'location', 'perimeter']

**Note**

Because [dir()](https://docs.python.org/3/library/functions.html#dir) is supplied primarily as a convenience for use at an interactive prompt, it tries to supply an interesting set of names more than it tries to supply a rigorously or consistently defined set of names, and its detailed behavior may change across releases. For example, metaclass attributes are not in the result list when the argument is a class.

**divmod**(a, b)

Take two (non complex) numbers as arguments and return a pair of numbers consisting of their quotient and remainder when using integer division. With mixed operand types, the rules for binary arithmetic operators apply. For integers, the result is the same as (a // b, a % b). For floating point numbers the result is (q, a % b), where q is usually math.floor(a / b) but may be 1 less than that. In any case q \* b + a % b is very close to a, if a % b is non-zero it has the same sign as b, and 0 <= abs(a % b) < abs(b).

**enumerate**(iterable, start=0)

Return an enumerate object. iterable must be a sequence, an [iterator](https://docs.python.org/3/glossary.html#term-iterator), or some other object which supports iteration. The [\_\_next\_\_()](https://docs.python.org/3/library/stdtypes.html#iterator.__next__) method of the iterator returned by [enumerate()](https://docs.python.org/3/library/functions.html#enumerate) returns a tuple containing a count (from start which defaults to 0) and the values obtained from iterating overiterable.

>>>

**>>>** seasons = ['Spring', 'Summer', 'Fall', 'Winter']

**>>>** list(enumerate(seasons))

[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]

**>>>** list(enumerate(seasons, start=1))

[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]

Equivalent to:

**def** enumerate(sequence, start=0):

n = start

**for** elem **in** sequence:

**yield** n, elem

n += 1

**eval**(expression, globals=None, locals=None)

The arguments are a string and optional globals and locals. If provided, globals must be a dictionary. If provided, locals can be any mapping object.

The expression argument is parsed and evaluated as a Python expression (technically speaking, a condition list) using the globals and localsdictionaries as global and local namespace. If the globals dictionary is present and lacks ‘\_\_builtins\_\_’, the current globals are copied into globalsbefore expression is parsed. This means that expression normally has full access to the standard [builtins](https://docs.python.org/3/library/builtins.html#module-builtins) module and restricted environments are propagated. If the locals dictionary is omitted it defaults to the globals dictionary. If both dictionaries are omitted, the expression is executed in the environment where [eval()](https://docs.python.org/3/library/functions.html#eval) is called. The return value is the result of the evaluated expression. Syntax errors are reported as exceptions. Example:

>>>

**>>>** x = 1

**>>>** eval('x+1')

2

This function can also be used to execute arbitrary code objects (such as those created by [compile()](https://docs.python.org/3/library/functions.html#compile)). In this case pass a code object instead of a string. If the code object has been compiled with 'exec' as the mode argument, [eval()](https://docs.python.org/3/library/functions.html#eval)‘s return value will be None.

Hints: dynamic execution of statements is supported by the [exec()](https://docs.python.org/3/library/functions.html#exec) function. The [globals()](https://docs.python.org/3/library/functions.html#globals) and [locals()](https://docs.python.org/3/library/functions.html#locals) functions returns the current global and local dictionary, respectively, which may be useful to pass around for use by [eval()](https://docs.python.org/3/library/functions.html#eval) or [exec()](https://docs.python.org/3/library/functions.html#exec).

See [ast.literal\_eval()](https://docs.python.org/3/library/ast.html#ast.literal_eval) for a function that can safely evaluate strings with expressions containing only literals.

**exec**(object[, globals[, locals]])

This function supports dynamic execution of Python code. object must be either a string or a code object. If it is a string, the string is parsed as a suite of Python statements which is then executed (unless a syntax error occurs). [[1]](https://docs.python.org/3/library/functions.html#id2) If it is a code object, it is simply executed. In all cases, the code that’s executed is expected to be valid as file input (see the section “File input” in the Reference Manual). Be aware that the [return](https://docs.python.org/3/reference/simple_stmts.html#return) and[yield](https://docs.python.org/3/reference/simple_stmts.html#yield) statements may not be used outside of function definitions even within the context of code passed to the [exec()](https://docs.python.org/3/library/functions.html#exec) function. The return value is None.

In all cases, if the optional parts are omitted, the code is executed in the current scope. If only globals is provided, it must be a dictionary, which will be used for both the global and the local variables. If globals and locals are given, they are used for the global and local variables, respectively. If provided, locals can be any mapping object. Remember that at module level, globals and locals are the same dictionary. If exec gets two separate objects as globals and locals, the code will be executed as if it were embedded in a class definition.

If the globals dictionary does not contain a value for the key \_\_builtins\_\_, a reference to the dictionary of the built-in module [builtins](https://docs.python.org/3/library/builtins.html#module-builtins) is inserted under that key. That way you can control what builtins are available to the executed code by inserting your own \_\_builtins\_\_ dictionary intoglobals before passing it to [exec()](https://docs.python.org/3/library/functions.html#exec).

**Note**

The built-in functions [globals()](https://docs.python.org/3/library/functions.html#globals) and [locals()](https://docs.python.org/3/library/functions.html#locals) return the current global and local dictionary, respectively, which may be useful to pass around for use as the second and third argument to [exec()](https://docs.python.org/3/library/functions.html#exec).

**Note**

The default locals act as described for function [locals()](https://docs.python.org/3/library/functions.html#locals) below: modifications to the default locals dictionary should not be attempted. Pass an explicit locals dictionary if you need to see effects of the code on locals after function [exec()](https://docs.python.org/3/library/functions.html#exec) returns.

**filter**(function, iterable)

Construct an iterator from those elements of iterable for which function returns true. iterable may be either a sequence, a container which supports iteration, or an iterator. If function is None, the identity function is assumed, that is, all elements of iterable that are false are removed.

Note that filter(function, iterable) is equivalent to the generator expression (item for item in iterable if function(item)) if function is not None and (item for item in iterable if item) if function is None.

See [itertools.filterfalse()](https://docs.python.org/3/library/itertools.html#itertools.filterfalse) for the complementary function that returns elements of iterable for which function returns false.

class**float**([x])

Return a floating point number constructed from a number or string x.

If the argument is a string, it should contain a decimal number, optionally preceded by a sign, and optionally embedded in whitespace. The optional sign may be '+' or '-'; a '+' sign has no effect on the value produced. The argument may also be a string representing a NaN (not-a-number), or a positive or negative infinity. More precisely, the input must conform to the following grammar after leading and trailing whitespace characters are removed:

**sign**  ::= "+" | "-"

**infinity**  ::= "Infinity" | "inf"

**nan**  ::= "nan"

**numeric\_value**  ::= [floatnumber](https://docs.python.org/3/reference/lexical_analysis.html#grammar-token-floatnumber) | [infinity](https://docs.python.org/3/library/functions.html#grammar-token-infinity) | [nan](https://docs.python.org/3/library/functions.html#grammar-token-nan)

**numeric\_string** ::= [[sign](https://docs.python.org/3/library/functions.html#grammar-token-sign)] [numeric\_value](https://docs.python.org/3/library/functions.html#grammar-token-numeric_value)

Here floatnumber is the form of a Python floating-point literal, described in [Floating point literals](https://docs.python.org/3/reference/lexical_analysis.html#floating). Case is not significant, so, for example, “inf”, “Inf”, “INFINITY” and “iNfINity” are all acceptable spellings for positive infinity.

Otherwise, if the argument is an integer or a floating point number, a floating point number with the same value (within Python’s floating point precision) is returned. If the argument is outside the range of a Python float, an [OverflowError](https://docs.python.org/3/library/exceptions.html#OverflowError) will be raised.

For a general Python object x, float(x) delegates to x.\_\_float\_\_().

If no argument is given, 0.0 is returned.

Examples:

>>>

**>>>** float('+1.23')

1.23

**>>>** float(' -12345**\n**')

-12345.0

**>>>** float('1e-003')

0.001

**>>>** float('+1E6')

1000000.0

**>>>** float('-Infinity')

-inf

The float type is described in [Numeric Types — int, float, complex](https://docs.python.org/3/library/stdtypes.html#typesnumeric).

*Changed in version 3.6:*Grouping digits with underscores as in code literals is allowed.

**format**(value[, format\_spec])

Convert a value to a “formatted” representation, as controlled by format\_spec. The interpretation of format\_spec will depend on the type of thevalue argument, however there is a standard formatting syntax that is used by most built-in types: [Format Specification Mini-Language](https://docs.python.org/3/library/string.html#formatspec).

The default format\_spec is an empty string which usually gives the same effect as calling [str(value)](https://docs.python.org/3/library/stdtypes.html#str).

A call to format(value, format\_spec) is translated to type(value).\_\_format\_\_(value, format\_spec) which bypasses the instance dictionary when searching for the value’s [\_\_format\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__format__) method. A [TypeError](https://docs.python.org/3/library/exceptions.html#TypeError) exception is raised if the method search reaches [object](https://docs.python.org/3/library/functions.html#object) and the format\_spec is non-empty, or if either the format\_spec or the return value are not strings.

*Changed in version 3.4:*object().\_\_format\_\_(format\_spec) raises [TypeError](https://docs.python.org/3/library/exceptions.html#TypeError) if format\_spec is not an empty string.

class**frozenset**([iterable])

Return a new [frozenset](https://docs.python.org/3/library/stdtypes.html#frozenset) object, optionally with elements taken from iterable. frozenset is a built-in class. See [frozenset](https://docs.python.org/3/library/stdtypes.html#frozenset) and [Set Types — set, frozenset](https://docs.python.org/3/library/stdtypes.html#types-set) for documentation about this class.

For other containers see the built-in [set](https://docs.python.org/3/library/stdtypes.html#set), [list](https://docs.python.org/3/library/stdtypes.html#list), [tuple](https://docs.python.org/3/library/stdtypes.html#tuple), and [dict](https://docs.python.org/3/library/stdtypes.html#dict) classes, as well as the [collections](https://docs.python.org/3/library/collections.html#module-collections) module.

**getattr**(object, name[, default])

Return the value of the named attribute of object. name must be a string. If the string is the name of one of the object’s attributes, the result is the value of that attribute. For example, getattr(x, 'foobar') is equivalent to x.foobar. If the named attribute does not exist, default is returned if provided, otherwise [AttributeError](https://docs.python.org/3/library/exceptions.html#AttributeError) is raised.

**globals**()

Return a dictionary representing the current global symbol table. This is always the dictionary of the current module (inside a function or method, this is the module where it is defined, not the module from which it is called).

**hasattr**(object, name)

The arguments are an object and a string. The result is True if the string is the name of one of the object’s attributes, False if not. (This is implemented by calling getattr(object, name) and seeing whether it raises an [AttributeError](https://docs.python.org/3/library/exceptions.html#AttributeError) or not.)

**hash**(object)

Return the hash value of the object (if it has one). Hash values are integers. They are used to quickly compare dictionary keys during a dictionary lookup. Numeric values that compare equal have the same hash value (even if they are of different types, as is the case for 1 and 1.0).

**Note**

For objects with custom [\_\_hash\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__hash__) methods, note that [hash()](https://docs.python.org/3/library/functions.html#hash) truncates the return value based on the bit width of the host machine. See [\_\_hash\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__hash__) for details.

**help**([object])

Invoke the built-in help system. (This function is intended for interactive use.) If no argument is given, the interactive help system starts on the interpreter console. If the argument is a string, then the string is looked up as the name of a module, function, class, method, keyword, or documentation topic, and a help page is printed on the console. If the argument is any other kind of object, a help page on the object is generated.

This function is added to the built-in namespace by the [site](https://docs.python.org/3/library/site.html#module-site) module.

*Changed in version 3.4:*Changes to [pydoc](https://docs.python.org/3/library/pydoc.html#module-pydoc) and [inspect](https://docs.python.org/3/library/inspect.html#module-inspect) mean that the reported signatures for callables are now more comprehensive and consistent.

**hex**(x)

Convert an integer number to a lowercase hexadecimal string prefixed with “0x”, for example:

>>>

**>>>** hex(255)

'0xff'

**>>>** hex(-42)

'-0x2a'

If x is not a Python [int](https://docs.python.org/3/library/functions.html#int) object, it has to define an \_\_index\_\_() method that returns an integer.

See also [int()](https://docs.python.org/3/library/functions.html#int) for converting a hexadecimal string to an integer using a base of 16.

**Note**

To obtain a hexadecimal string representation for a float, use the [float.hex()](https://docs.python.org/3/library/stdtypes.html#float.hex) method.

**id**(object)

Return the “identity” of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same [id()](https://docs.python.org/3/library/functions.html#id) value.

**CPython implementation detail:** This is the address of the object in memory.

**input**([prompt])

If the prompt argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, [EOFError](https://docs.python.org/3/library/exceptions.html#EOFError) is raised. Example:

>>>

**>>>** s = input('--> ')

--> Monty Python's Flying Circus

**>>>** s

"Monty Python's Flying Circus"

If the [readline](https://docs.python.org/3/library/readline.html#module-readline) module was loaded, then [input()](https://docs.python.org/3/library/functions.html#input) will use it to provide elaborate line editing and history features.

class**int**(x=0)

class**int**(x, base=10)

Return an integer object constructed from a number or string x, or return 0 if no arguments are given. If x is a number, return [x.\_\_int\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__int__). For floating point numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string, [bytes](https://docs.python.org/3/library/functions.html#bytes), or [bytearray](https://docs.python.org/3/library/functions.html#bytearray) instance representing an [integer literal](https://docs.python.org/3/reference/lexical_analysis.html#integers) in radix base. Optionally, the literal can be preceded by + or - (with no space in between) and surrounded by whitespace. A base-n literal consists of the digits 0 to n-1, with a to z (or A to Z) having values 10 to 35. The default base is 10. The allowed values are 0 and 2–36. Base-2, -8, and -16 literals can be optionally prefixed with 0b/0B, 0o/0O, or 0x/0X, as with integer literals in code. Base 0 means to interpret exactly as a code literal, so that the actual base is 2, 8, 10, or 16, and so that int('010', 0) is not legal, while int('010') is, as well as int('010', 8).

The integer type is described in [Numeric Types — int, float, complex](https://docs.python.org/3/library/stdtypes.html#typesnumeric).

*Changed in version 3.4:*If base is not an instance of [int](https://docs.python.org/3/library/functions.html#int) and the base object has a [base.\_\_index\_\_](https://docs.python.org/3/reference/datamodel.html#object.__index__) method, that method is called to obtain an integer for the base. Previous versions used [base.\_\_int\_\_](https://docs.python.org/3/reference/datamodel.html#object.__int__) instead of [base.\_\_index\_\_](https://docs.python.org/3/reference/datamodel.html#object.__index__).

*Changed in version 3.6:*Grouping digits with underscores as in code literals is allowed.

**isinstance**(object, classinfo)

Return true if the object argument is an instance of the classinfo argument, or of a (direct, indirect or [virtual](https://docs.python.org/3/glossary.html#term-abstract-base-class)) subclass thereof. If object is not an object of the given type, the function always returns false. If classinfo is a tuple of type objects (or recursively, other such tuples), return true ifobject is an instance of any of the types. If classinfo is not a type or tuple of types and such tuples, a [TypeError](https://docs.python.org/3/library/exceptions.html#TypeError) exception is raised.

**issubclass**(class, classinfo)

Return true if class is a subclass (direct, indirect or [virtual](https://docs.python.org/3/glossary.html#term-abstract-base-class)) of classinfo. A class is considered a subclass of itself. classinfo may be a tuple of class objects, in which case every entry in classinfo will be checked. In any other case, a [TypeError](https://docs.python.org/3/library/exceptions.html#TypeError) exception is raised.

**iter**(object[, sentinel])

Return an [iterator](https://docs.python.org/3/glossary.html#term-iterator) object. The first argument is interpreted very differently depending on the presence of the second argument. Without a second argument, object must be a collection object which supports the iteration protocol (the [\_\_iter\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__iter__) method), or it must support the sequence protocol (the [\_\_getitem\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getitem__) method with integer arguments starting at 0). If it does not support either of those protocols, [TypeError](https://docs.python.org/3/library/exceptions.html#TypeError) is raised. If the second argument, sentinel, is given, then object must be a callable object. The iterator created in this case will call object with no arguments for each call to its [\_\_next\_\_()](https://docs.python.org/3/library/stdtypes.html#iterator.__next__) method; if the value returned is equal to sentinel, [StopIteration](https://docs.python.org/3/library/exceptions.html#StopIteration) will be raised, otherwise the value will be returned.

See also [Iterator Types](https://docs.python.org/3/library/stdtypes.html#typeiter).

One useful application of the second form of [iter()](https://docs.python.org/3/library/functions.html#iter) is to read lines of a file until a certain line is reached. The following example reads a file until the [readline()](https://docs.python.org/3/library/io.html#io.TextIOBase.readline) method returns an empty string:

**with** open('mydata.txt') **as** fp:

**for** line **in** iter(fp.readline, ''):

process\_line(line)

**len**(s)

Return the length (the number of items) of an object. The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).

class**list**([iterable])

Rather than being a function, [list](https://docs.python.org/3/library/stdtypes.html#list) is actually a mutable sequence type, as documented in [Lists](https://docs.python.org/3/library/stdtypes.html#typesseq-list) and [Sequence Types — list, tuple, range](https://docs.python.org/3/library/stdtypes.html#typesseq).

**locals**()

Update and return a dictionary representing the current local symbol table. Free variables are returned by [locals()](https://docs.python.org/3/library/functions.html#locals) when it is called in function blocks, but not in class blocks.

**Note**

The contents of this dictionary should not be modified; changes may not affect the values of local and free variables used by the interpreter.

**map**(function, iterable, ...)

Return an iterator that applies function to every item of iterable, yielding the results. If additional iterable arguments are passed, function must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted. For cases where the function inputs are already arranged into argument tuples, see [itertools.starmap()](https://docs.python.org/3/library/itertools.html#itertools.starmap).

**max**(iterable, \*[, key, default])

**max**(arg1, arg2, \*args[, key])

Return the largest item in an iterable or the largest of two or more arguments.

If one positional argument is provided, it should be an [iterable](https://docs.python.org/3/glossary.html#term-iterable). The largest item in the iterable is returned. If two or more positional arguments are provided, the largest of the positional arguments is returned.

There are two optional keyword-only arguments. The key argument specifies a one-argument ordering function like that used for [list.sort()](https://docs.python.org/3/library/stdtypes.html#list.sort). The default argument specifies an object to return if the provided iterable is empty. If the iterable is empty and default is not provided, a[ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) is raised.

If multiple items are maximal, the function returns the first one encountered. This is consistent with other sort-stability preserving tools such assorted(iterable, key=keyfunc, reverse=True)[0] and heapq.nlargest(1, iterable, key=keyfunc).

*New in version 3.4:*The default keyword-only argument.

**memoryview**(obj)

Return a “memory view” object created from the given argument. See [Memory Views](https://docs.python.org/3/library/stdtypes.html#typememoryview) for more information.

**min**(iterable, \*[, key, default])

**min**(arg1, arg2, \*args[, key])

Return the smallest item in an iterable or the smallest of two or more arguments.

If one positional argument is provided, it should be an [iterable](https://docs.python.org/3/glossary.html#term-iterable). The smallest item in the iterable is returned. If two or more positional arguments are provided, the smallest of the positional arguments is returned.

There are two optional keyword-only arguments. The key argument specifies a one-argument ordering function like that used for [list.sort()](https://docs.python.org/3/library/stdtypes.html#list.sort). The default argument specifies an object to return if the provided iterable is empty. If the iterable is empty and default is not provided, a[ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) is raised.

If multiple items are minimal, the function returns the first one encountered. This is consistent with other sort-stability preserving tools such assorted(iterable, key=keyfunc)[0] and heapq.nsmallest(1, iterable, key=keyfunc).

*New in version 3.4:*The default keyword-only argument.

**next**(iterator[, default])

Retrieve the next item from the iterator by calling its [\_\_next\_\_()](https://docs.python.org/3/library/stdtypes.html#iterator.__next__) method. If default is given, it is returned if the iterator is exhausted, otherwise[StopIteration](https://docs.python.org/3/library/exceptions.html#StopIteration) is raised.

class**object**

Return a new featureless object. [object](https://docs.python.org/3/library/functions.html#object) is a base for all classes. It has the methods that are common to all instances of Python classes. This function does not accept any arguments.

**Note**

[object](https://docs.python.org/3/library/functions.html#object) does not have a [\_\_dict\_\_](https://docs.python.org/3/library/stdtypes.html#object.__dict__), so you can’t assign arbitrary attributes to an instance of the [object](https://docs.python.org/3/library/functions.html#object) class.

**oct**(x)

Convert an integer number to an octal string. The result is a valid Python expression. If x is not a Python [int](https://docs.python.org/3/library/functions.html#int) object, it has to define an[\_\_index\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__index__) method that returns an integer.

**open**(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

Open file and return a corresponding [file object](https://docs.python.org/3/glossary.html#term-file-object). If the file cannot be opened, an [OSError](https://docs.python.org/3/library/exceptions.html#OSError) is raised.

file is a [path-like object](https://docs.python.org/3/glossary.html#term-path-like-object) giving the pathname (absolute or relative to the current working directory) of the file to be opened or an integer file descriptor of the file to be wrapped. (If a file descriptor is given, it is closed when the returned I/O object is closed, unless closefd is set to False.)

mode is an optional string that specifies the mode in which the file is opened. It defaults to 'r' which means open for reading in text mode. Other common values are 'w' for writing (truncating the file if it already exists), 'x' for exclusive creation and 'a' for appending (which on some Unix systems, means that all writes append to the end of the file regardless of the current seek position). In text mode, if encoding is not specified the encoding used is platform dependent: locale.getpreferredencoding(False) is called to get the current locale encoding. (For reading and writing raw bytes use binary mode and leave encoding unspecified.) The available modes are:

| **Character** | **Meaning** |
| --- | --- |
| 'r' | open for reading (default) |
| 'w' | open for writing, truncating the file first |
| 'x' | open for exclusive creation, failing if the file already exists |
| 'a' | open for writing, appending to the end of the file if it exists |
| 'b' | binary mode |
| 't' | text mode (default) |
| '+' | open a disk file for updating (reading and writing) |
| 'U' | [universal newlines](https://docs.python.org/3/glossary.html#term-universal-newlines) mode (deprecated) |

The default mode is 'r' (open for reading text, synonym of 'rt'). For binary read-write access, the mode 'w+b' opens and truncates the file to 0 bytes. 'r+b' opens the file without truncation.

As mentioned in the [Overview](https://docs.python.org/3/library/io.html#io-overview), Python distinguishes between binary and text I/O. Files opened in binary mode (including 'b' in the modeargument) return contents as [bytes](https://docs.python.org/3/library/functions.html#bytes) objects without any decoding. In text mode (the default, or when 't' is included in the mode argument), the contents of the file are returned as [str](https://docs.python.org/3/library/stdtypes.html#str), the bytes having been first decoded using a platform-dependent encoding or using the specified encodingif given.

**Note**

Python doesn’t depend on the underlying operating system’s notion of text files; all the processing is done by Python itself, and is therefore platform-independent.

buffering is an optional integer used to set the buffering policy. Pass 0 to switch buffering off (only allowed in binary mode), 1 to select line buffering (only usable in text mode), and an integer > 1 to indicate the size in bytes of a fixed-size chunk buffer. When no buffering argument is given, the default buffering policy works as follows:

* Binary files are buffered in fixed-size chunks; the size of the buffer is chosen using a heuristic trying to determine the underlying device’s “block size” and falling back on [io.DEFAULT\_BUFFER\_SIZE](https://docs.python.org/3/library/io.html#io.DEFAULT_BUFFER_SIZE). On many systems, the buffer will typically be 4096 or 8192 bytes long.
* “Interactive” text files (files for which [isatty()](https://docs.python.org/3/library/io.html#io.IOBase.isatty) returns True) use line buffering. Other text files use the policy described above for binary files.

encoding is the name of the encoding used to decode or encode the file. This should only be used in text mode. The default encoding is platform dependent (whatever [locale.getpreferredencoding()](https://docs.python.org/3/library/locale.html#locale.getpreferredencoding) returns), but any [text encoding](https://docs.python.org/3/glossary.html#term-text-encoding) supported by Python can be used. See the [codecs](https://docs.python.org/3/library/codecs.html#module-codecs) module for the list of supported encodings.

errors is an optional string that specifies how encoding and decoding errors are to be handled—this cannot be used in binary mode. A variety of standard error handlers are available (listed under [Error Handlers](https://docs.python.org/3/library/codecs.html#error-handlers)), though any error handling name that has been registered with[codecs.register\_error()](https://docs.python.org/3/library/codecs.html#codecs.register_error) is also valid. The standard names include:

* 'strict' to raise a [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) exception if there is an encoding error. The default value of None has the same effect.
* 'ignore' ignores errors. Note that ignoring encoding errors can lead to data loss.
* 'replace' causes a replacement marker (such as '?') to be inserted where there is malformed data.
* 'surrogateescape' will represent any incorrect bytes as code points in the Unicode Private Use Area ranging from U+DC80 to U+DCFF. These private code points will then be turned back into the same bytes when the surrogateescape error handler is used when writing data. This is useful for processing files in an unknown encoding.
* 'xmlcharrefreplace' is only supported when writing to a file. Characters not supported by the encoding are replaced with the appropriate XML character reference &#nnn;.
* 'backslashreplace' replaces malformed data by Python’s backslashed escape sequences.
* 'namereplace' (also only supported when writing) replaces unsupported characters with \N{...} escape sequences.

newline controls how [universal newlines](https://docs.python.org/3/glossary.html#term-universal-newlines) mode works (it only applies to text mode). It can be None, '', '\n', '\r', and '\r\n'. It works as follows:

* When reading input from the stream, if newline is None, universal newlines mode is enabled. Lines in the input can end in '\n', '\r', or'\r\n', and these are translated into '\n' before being returned to the caller. If it is '', universal newlines mode is enabled, but line endings are returned to the caller untranslated. If it has any of the other legal values, input lines are only terminated by the given string, and the line ending is returned to the caller untranslated.
* When writing output to the stream, if newline is None, any '\n' characters written are translated to the system default line separator,[os.linesep](https://docs.python.org/3/library/os.html#os.linesep). If newline is '' or '\n', no translation takes place. If newline is any of the other legal values, any '\n' characters written are translated to the given string.

If closefd is False and a file descriptor rather than a filename was given, the underlying file descriptor will be kept open when the file is closed. If a filename is given closefd must be True (the default) otherwise an error will be raised.

A custom opener can be used by passing a callable as opener. The underlying file descriptor for the file object is then obtained by calling openerwith (file, flags). opener must return an open file descriptor (passing [os.open](https://docs.python.org/3/library/os.html#os.open) as opener results in functionality similar to passing None).

The newly created file is [non-inheritable](https://docs.python.org/3/library/os.html#fd-inheritance).

The following example uses the [dir\_fd](https://docs.python.org/3/library/os.html#dir-fd) parameter of the [os.open()](https://docs.python.org/3/library/os.html#os.open) function to open a file relative to a given directory:

>>>

**>>> import** **os**

**>>>** dir\_fd = os.open('somedir', os.O\_RDONLY)

**>>> def** opener(path, flags):

**...**  **return** os.open(path, flags, dir\_fd=dir\_fd)

**...**

**>>> with** open('spamspam.txt', 'w', opener=opener) **as** f:

**...**  print('This will be written to somedir/spamspam.txt', file=f)

**...**

**>>>** os.close(dir\_fd) *# don't leak a file descriptor*

The type of [file object](https://docs.python.org/3/glossary.html#term-file-object) returned by the [open()](https://docs.python.org/3/library/functions.html#open) function depends on the mode. When [open()](https://docs.python.org/3/library/functions.html#open) is used to open a file in a text mode ('w', 'r', 'wt','rt', etc.), it returns a subclass of [io.TextIOBase](https://docs.python.org/3/library/io.html#io.TextIOBase) (specifically [io.TextIOWrapper](https://docs.python.org/3/library/io.html#io.TextIOWrapper)). When used to open a file in a binary mode with buffering, the returned class is a subclass of [io.BufferedIOBase](https://docs.python.org/3/library/io.html#io.BufferedIOBase). The exact class varies: in read binary mode, it returns an [io.BufferedReader](https://docs.python.org/3/library/io.html#io.BufferedReader); in write binary and append binary modes, it returns an [io.BufferedWriter](https://docs.python.org/3/library/io.html#io.BufferedWriter), and in read/write mode, it returns an [io.BufferedRandom](https://docs.python.org/3/library/io.html#io.BufferedRandom). When buffering is disabled, the raw stream, a subclass of [io.RawIOBase](https://docs.python.org/3/library/io.html#io.RawIOBase), [io.FileIO](https://docs.python.org/3/library/io.html#io.FileIO), is returned.

See also the file handling modules, such as, [fileinput](https://docs.python.org/3/library/fileinput.html#module-fileinput), [io](https://docs.python.org/3/library/io.html#module-io) (where [open()](https://docs.python.org/3/library/functions.html#open) is declared), [os](https://docs.python.org/3/library/os.html#module-os), [os.path](https://docs.python.org/3/library/os.path.html#module-os.path), [tempfile](https://docs.python.org/3/library/tempfile.html#module-tempfile), and [shutil](https://docs.python.org/3/library/shutil.html#module-shutil).

*Changed in version 3.3:*

* The opener parameter was added.
* The 'x' mode was added.
* [IOError](https://docs.python.org/3/library/exceptions.html#IOError) used to be raised, it is now an alias of [OSError](https://docs.python.org/3/library/exceptions.html#OSError).
* [FileExistsError](https://docs.python.org/3/library/exceptions.html#FileExistsError) is now raised if the file opened in exclusive creation mode ('x') already exists.

*Changed in version 3.4:*

* The file is now non-inheritable.

*Deprecated since version 3.4, will be removed in version 4.0:*The 'U' mode.

*Changed in version 3.5:*

* If the system call is interrupted and the signal handler does not raise an exception, the function now retries the system call instead of raising an [InterruptedError](https://docs.python.org/3/library/exceptions.html#InterruptedError) exception (see [**PEP 475**](https://www.python.org/dev/peps/pep-0475) for the rationale).
* The 'namereplace' error handler was added.

*Changed in version 3.6:*

* Support added to accept objects implementing [os.PathLike](https://docs.python.org/3/library/os.html#os.PathLike).
* On Windows, opening a console buffer may return a subclass of [io.RawIOBase](https://docs.python.org/3/library/io.html#io.RawIOBase) other than [io.FileIO](https://docs.python.org/3/library/io.html#io.FileIO).

**ord**(c)

Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. For example,ord('a') returns the integer 97 and ord('€') (Euro sign) returns 8364. This is the inverse of [chr()](https://docs.python.org/3/library/functions.html#chr).

**pow**(x, y[, z])

Return x to the power y; if z is present, return x to the power y, modulo z (computed more efficiently than pow(x, y) % z). The two-argument formpow(x, y) is equivalent to using the power operator: x\*\*y.

The arguments must have numeric types. With mixed operand types, the coercion rules for binary arithmetic operators apply. For [int](https://docs.python.org/3/library/functions.html#int) operands, the result has the same type as the operands (after coercion) unless the second argument is negative; in that case, all arguments are converted to float and a float result is delivered. For example, 10\*\*2 returns 100, but 10\*\*-2 returns 0.01. If the second argument is negative, the third argument must be omitted. If z is present, x and y must be of integer types, and y must be non-negative.

**print**(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

Print objects to the text stream file, separated by sep and followed by end. sep, end, file and flush, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like [str()](https://docs.python.org/3/library/stdtypes.html#str) does and written to the stream, separated by sep and followed by end. Both sepand end must be strings; they can also be None, which means to use the default values. If no objects are given, [print()](https://docs.python.org/3/library/functions.html#print) will just write end.

The file argument must be an object with a write(string) method; if it is not present or None, [sys.stdout](https://docs.python.org/3/library/sys.html#sys.stdout) will be used. Since printed arguments are converted to text strings, [print()](https://docs.python.org/3/library/functions.html#print) cannot be used with binary mode file objects. For these, use file.write(...) instead.

Whether output is buffered is usually determined by file, but if the flush keyword argument is true, the stream is forcibly flushed.

*Changed in version 3.3:*Added the flush keyword argument.

class**property**(fget=None, fset=None, fdel=None, doc=None)

Return a property attribute.

fget is a function for getting an attribute value. fset is a function for setting an attribute value. fdel is a function for deleting an attribute value. Anddoc creates a docstring for the attribute.

A typical use is to define a managed attribute x:

**class** **C**:

**def** \_\_init\_\_(self):

self.\_x = **None**

**def** getx(self):

**return** self.\_x

**def** setx(self, value):

self.\_x = value

**def** delx(self):

**del** self.\_x

x = property(getx, setx, delx, "I'm the 'x' property.")

If c is an instance of C, c.x will invoke the getter, c.x = value will invoke the setter and del c.x the deleter.

If given, doc will be the docstring of the property attribute. Otherwise, the property will copy fget‘s docstring (if it exists). This makes it possible to create read-only properties easily using [property()](https://docs.python.org/3/library/functions.html#property) as a [decorator](https://docs.python.org/3/glossary.html#term-decorator):

**class** **Parrot**:

**def** \_\_init\_\_(self):

self.\_voltage = 100000

**@property**

**def** voltage(self):

*"""Get the current voltage."""*

**return** self.\_voltage

The @property decorator turns the voltage() method into a “getter” for a read-only attribute with the same name, and it sets the docstring forvoltage to “Get the current voltage.”

A property object has getter, setter, and deleter methods usable as decorators that create a copy of the property with the corresponding accessor function set to the decorated function. This is best explained with an example:

**class** **C**:

**def** \_\_init\_\_(self):

self.\_x = **None**

**@property**

**def** x(self):

*"""I'm the 'x' property."""*

**return** self.\_x

**@x**.setter

**def** x(self, value):

self.\_x = value

**@x**.deleter

**def** x(self):

**del** self.\_x

This code is exactly equivalent to the first example. Be sure to give the additional functions the same name as the original property (x in this case.)

The returned property object also has the attributes fget, fset, and fdel corresponding to the constructor arguments.

*Changed in version 3.5:*The docstrings of property objects are now writeable.

**range**(stop)

**range**(start, stop[, step])

Rather than being a function, [range](https://docs.python.org/3/library/stdtypes.html#range) is actually an immutable sequence type, as documented in [Ranges](https://docs.python.org/3/library/stdtypes.html#typesseq-range) and [Sequence Types — list, tuple, range](https://docs.python.org/3/library/stdtypes.html#typesseq).

**repr**(object)

Return a string containing a printable representation of an object. For many types, this function makes an attempt to return a string that would yield an object with the same value when passed to [eval()](https://docs.python.org/3/library/functions.html#eval), otherwise the representation is a string enclosed in angle brackets that contains the name of the type of the object together with additional information often including the name and address of the object. A class can control what this function returns for its instances by defining a [\_\_repr\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__repr__) method.

**reversed**(seq)

Return a reverse [iterator](https://docs.python.org/3/glossary.html#term-iterator). seq must be an object which has a [\_\_reversed\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__reversed__) method or supports the sequence protocol (the [\_\_len\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__len__) method and the [\_\_getitem\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getitem__) method with integer arguments starting at 0).

**round**(number[, ndigits])

Return number rounded to ndigits precision after the decimal point. If ndigits is omitted or is None, it returns the nearest integer to its input.

For the built-in types supporting [round()](https://docs.python.org/3/library/functions.html#round), values are rounded to the closest multiple of 10 to the power minus ndigits; if two multiples are equally close, rounding is done toward the even choice (so, for example, both round(0.5) and round(-0.5) are 0, and round(1.5) is 2). Any integer value is valid for ndigits (positive, zero, or negative). The return value is an integer if called with one argument, otherwise of the same type as number.

For a general Python object number, round(number, ndigits) delegates to number.\_\_round\_\_(ndigits).

**Note**

The behavior of [round()](https://docs.python.org/3/library/functions.html#round) for floats can be surprising: for example, round(2.675, 2) gives 2.67 instead of the expected 2.68. This is not a bug: it’s a result of the fact that most decimal fractions can’t be represented exactly as a float. See [Floating Point Arithmetic: Issues and Limitations](https://docs.python.org/3/tutorial/floatingpoint.html#tut-fp-issues) for more information.

class**set**([iterable])

Return a new [set](https://docs.python.org/3/library/stdtypes.html#set) object, optionally with elements taken from iterable. set is a built-in class. See [set](https://docs.python.org/3/library/stdtypes.html#set) and [Set Types — set, frozenset](https://docs.python.org/3/library/stdtypes.html#types-set) for documentation about this class.

For other containers see the built-in [frozenset](https://docs.python.org/3/library/stdtypes.html#frozenset), [list](https://docs.python.org/3/library/stdtypes.html#list), [tuple](https://docs.python.org/3/library/stdtypes.html#tuple), and [dict](https://docs.python.org/3/library/stdtypes.html#dict) classes, as well as the [collections](https://docs.python.org/3/library/collections.html#module-collections) module.

**setattr**(object, name, value)

This is the counterpart of [getattr()](https://docs.python.org/3/library/functions.html#getattr). The arguments are an object, a string and an arbitrary value. The string may name an existing attribute or a new attribute. The function assigns the value to the attribute, provided the object allows it. For example, setattr(x, 'foobar', 123) is equivalent to x.foobar = 123.

class**slice**(stop)

class**slice**(start, stop[, step])

Return a [slice](https://docs.python.org/3/glossary.html#term-slice) object representing the set of indices specified by range(start, stop, step). The start and step arguments default to None. Slice objects have read-only data attributes start, stop and step which merely return the argument values (or their default). They have no other explicit functionality; however they are used by Numerical Python and other third party extensions. Slice objects are also generated when extended indexing syntax is used. For example: a[start:stop:step] or a[start:stop, i]. See [itertools.islice()](https://docs.python.org/3/library/itertools.html#itertools.islice) for an alternate version that returns an iterator.

**sorted**(iterable[, key][, reverse])

Return a new sorted list from the items in iterable.

Has two optional arguments which must be specified as keyword arguments.

key specifies a function of one argument that is used to extract a comparison key from each list element: key=str.lower. The default value is None(compare the elements directly).

reverse is a boolean value. If set to True, then the list elements are sorted as if each comparison were reversed.

Use [functools.cmp\_to\_key()](https://docs.python.org/3/library/functools.html#functools.cmp_to_key) to convert an old-style cmp function to a key function.

The built-in [sorted()](https://docs.python.org/3/library/functions.html#sorted) function is guaranteed to be stable. A sort is stable if it guarantees not to change the relative order of elements that compare equal — this is helpful for sorting in multiple passes (for example, sort by department, then by salary grade).

For sorting examples and a brief sorting tutorial, see [Sorting HOW TO](https://docs.python.org/3/howto/sorting.html#sortinghowto).

**staticmethod**(function)

Return a static method for function.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

**class** **C**:

**@staticmethod**

**def** f(arg1, arg2, ...): ...

The @staticmethod form is a function [decorator](https://docs.python.org/3/glossary.html#term-decorator) – see the description of function definitions in [Function definitions](https://docs.python.org/3/reference/compound_stmts.html#function) for details.

It can be called either on the class (such as C.f()) or on an instance (such as C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. Also see [classmethod()](https://docs.python.org/3/library/functions.html#classmethod) for a variant that is useful for creating alternate class constructors.

For more information on static methods, consult the documentation on the standard type hierarchy in [The standard type hierarchy](https://docs.python.org/3/reference/datamodel.html#types).

class**str**(object='')

class**str**(object=b'', encoding='utf-8', errors='strict')

Return a [str](https://docs.python.org/3/library/stdtypes.html#str) version of object. See [str()](https://docs.python.org/3/library/stdtypes.html#str) for details.

str is the built-in string [class](https://docs.python.org/3/glossary.html#term-class). For general information about strings, see [Text Sequence Type — str](https://docs.python.org/3/library/stdtypes.html#textseq).

**sum**(iterable[, start])

Sums start and the items of an iterable from left to right and returns the total. start defaults to 0. The iterable‘s items are normally numbers, and the start value is not allowed to be a string.

For some use cases, there are good alternatives to [sum()](https://docs.python.org/3/library/functions.html#sum). The preferred, fast way to concatenate a sequence of strings is by calling''.join(sequence). To add floating point values with extended precision, see [math.fsum()](https://docs.python.org/3/library/math.html#math.fsum). To concatenate a series of iterables, consider using[itertools.chain()](https://docs.python.org/3/library/itertools.html#itertools.chain).

**super**([type[, object-or-type]])

Return a proxy object that delegates method calls to a parent or sibling class of type. This is useful for accessing inherited methods that have been overridden in a class. The search order is same as that used by [getattr()](https://docs.python.org/3/library/functions.html#getattr) except that the type itself is skipped.

The [\_\_mro\_\_](https://docs.python.org/3/library/stdtypes.html#class.__mro__) attribute of the type lists the method resolution search order used by both [getattr()](https://docs.python.org/3/library/functions.html#getattr) and [super()](https://docs.python.org/3/library/functions.html#super). The attribute is dynamic and can change whenever the inheritance hierarchy is updated.

If the second argument is omitted, the super object returned is unbound. If the second argument is an object, isinstance(obj, type) must be true. If the second argument is a type, issubclass(type2, type) must be true (this is useful for classmethods).

There are two typical use cases for super. In a class hierarchy with single inheritance, super can be used to refer to parent classes without naming them explicitly, thus making the code more maintainable. This use closely parallels the use of super in other programming languages.

The second use case is to support cooperative multiple inheritance in a dynamic execution environment. This use case is unique to Python and is not found in statically compiled languages or languages that only support single inheritance. This makes it possible to implement “diamond diagrams” where multiple base classes implement the same method. Good design dictates that this method have the same calling signature in every case (because the order of calls is determined at runtime, because that order adapts to changes in the class hierarchy, and because that order can include sibling classes that are unknown prior to runtime).

For both use cases, a typical superclass call looks like this:

**class** **C**(B):

**def** method(self, arg):

super().method(arg) *# This does the same thing as:*

*# super(C, self).method(arg)*

Note that [super()](https://docs.python.org/3/library/functions.html#super) is implemented as part of the binding process for explicit dotted attribute lookups such as super().\_\_getitem\_\_(name). It does so by implementing its own [\_\_getattribute\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__getattribute__) method for searching classes in a predictable order that supports cooperative multiple inheritance. Accordingly, [super()](https://docs.python.org/3/library/functions.html#super) is undefined for implicit lookups using statements or operators such as super()[name].

Also note that, aside from the zero argument form, [super()](https://docs.python.org/3/library/functions.html#super) is not limited to use inside methods. The two argument form specifies the arguments exactly and makes the appropriate references. The zero argument form only works inside a class definition, as the compiler fills in the necessary details to correctly retrieve the class being defined, as well as accessing the current instance for ordinary methods.

For practical suggestions on how to design cooperative classes using [super()](https://docs.python.org/3/library/functions.html#super), see [guide to using super()](https://rhettinger.wordpress.com/2011/05/26/super-considered-super/).

**tuple**([iterable])

Rather than being a function, [tuple](https://docs.python.org/3/library/stdtypes.html#tuple) is actually an immutable sequence type, as documented in [Tuples](https://docs.python.org/3/library/stdtypes.html#typesseq-tuple) and [Sequence Types — list, tuple, range](https://docs.python.org/3/library/stdtypes.html#typesseq).

class**type**(object)

class**type**(name, bases, dict)

With one argument, return the type of an object. The return value is a type object and generally the same object as returned by [object.\_\_class\_\_](https://docs.python.org/3/library/stdtypes.html#instance.__class__).

The [isinstance()](https://docs.python.org/3/library/functions.html#isinstance) built-in function is recommended for testing the type of an object, because it takes subclasses into account.

With three arguments, return a new type object. This is essentially a dynamic form of the [class](https://docs.python.org/3/reference/compound_stmts.html#class) statement. The name string is the class name and becomes the [\_\_name\_\_](https://docs.python.org/3/library/stdtypes.html#definition.__name__) attribute; the bases tuple itemizes the base classes and becomes the [\_\_bases\_\_](https://docs.python.org/3/library/stdtypes.html#class.__bases__) attribute; and the dict dictionary is the namespace containing definitions for class body and is copied to a standard dictionary to become the [\_\_dict\_\_](https://docs.python.org/3/library/stdtypes.html#object.__dict__) attribute. For example, the following two statements create identical [type](https://docs.python.org/3/library/functions.html#type) objects:

>>>

**>>> class** **X**:

**...**  a = 1

**...**

**>>>** X = type('X', (object,), dict(a=1))

See also [Type Objects](https://docs.python.org/3/library/stdtypes.html#bltin-type-objects).

*Changed in version 3.6:*Subclasses of [type](https://docs.python.org/3/library/functions.html#type) which don’t override type.\_\_new\_\_ may no longer use the one-argument form to get the type of an object.

**vars**([object])

Return the [\_\_dict\_\_](https://docs.python.org/3/library/stdtypes.html#object.__dict__) attribute for a module, class, instance, or any other object with a [\_\_dict\_\_](https://docs.python.org/3/library/stdtypes.html#object.__dict__) attribute.

Objects such as modules and instances have an updateable [\_\_dict\_\_](https://docs.python.org/3/library/stdtypes.html#object.__dict__) attribute; however, other objects may have write restrictions on their[\_\_dict\_\_](https://docs.python.org/3/library/stdtypes.html#object.__dict__) attributes (for example, classes use a [types.MappingProxyType](https://docs.python.org/3/library/types.html#types.MappingProxyType) to prevent direct dictionary updates).

Without an argument, [vars()](https://docs.python.org/3/library/functions.html#vars) acts like [locals()](https://docs.python.org/3/library/functions.html#locals). Note, the locals dictionary is only useful for reads since updates to the locals dictionary are ignored.

**zip**(\*iterables)

Make an iterator that aggregates elements from each of the iterables.

Returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. The iterator stops when the shortest input iterable is exhausted. With a single iterable argument, it returns an iterator of 1-tuples. With no arguments, it returns an empty iterator. Equivalent to:

**def** zip(\*iterables):

*# zip('ABCD', 'xy') --> Ax By*

sentinel = object()

iterators = [iter(it) **for** it **in** iterables]

**while** iterators:

result = []

**for** it **in** iterators:

elem = next(it, sentinel)

**if** elem **is** sentinel:

**return**

result.append(elem)

**yield** tuple(result)

The left-to-right evaluation order of the iterables is guaranteed. This makes possible an idiom for clustering a data series into n-length groups using zip(\*[iter(s)]\*n). This repeats the same iterator n times so that each output tuple has the result of n calls to the iterator. This has the effect of dividing the input into n-length chunks.

[zip()](https://docs.python.org/3/library/functions.html#zip) should only be used with unequal length inputs when you don’t care about trailing, unmatched values from the longer iterables. If those values are important, use [itertools.zip\_longest()](https://docs.python.org/3/library/itertools.html#itertools.zip_longest) instead.

[zip()](https://docs.python.org/3/library/functions.html#zip) in conjunction with the \* operator can be used to unzip a list:

>>>

**>>>** x = [1, 2, 3]

**>>>** y = [4, 5, 6]

**>>>** zipped = zip(x, y)

**>>>** list(zipped)

[(1, 4), (2, 5), (3, 6)]

**>>>** x2, y2 = zip(\*zip(x, y))

**>>>** x == list(x2) **and** y == list(y2)

True

**\_\_import\_\_**(name, globals=None, locals=None, fromlist=(), level=0)

**Note**

This is an advanced function that is not needed in everyday Python programming, unlike [importlib.import\_module()](https://docs.python.org/3/library/importlib.html#importlib.import_module).

This function is invoked by the [import](https://docs.python.org/3/reference/simple_stmts.html#import) statement. It can be replaced (by importing the [builtins](https://docs.python.org/3/library/builtins.html#module-builtins) module and assigning to builtins.\_\_import\_\_) in order to change semantics of the [import](https://docs.python.org/3/reference/simple_stmts.html#import) statement, but doing so is **strongly** discouraged as it is usually simpler to use import hooks (see [**PEP 302**](https://www.python.org/dev/peps/pep-0302)) to attain the same goals and does not cause issues with code which assumes the default import implementation is in use. Direct use of[\_\_import\_\_()](https://docs.python.org/3/library/functions.html#__import__) is also discouraged in favor of [importlib.import\_module()](https://docs.python.org/3/library/importlib.html#importlib.import_module).

The function imports the module name, potentially using the given globals and locals to determine how to interpret the name in a package context. The fromlist gives the names of objects or submodules that should be imported from the module given by name. The standard implementation does not use its locals argument at all, and uses its globals only to determine the package context of the [import](https://docs.python.org/3/reference/simple_stmts.html#import) statement.

level specifies whether to use absolute or relative imports. 0 (the default) means only perform absolute imports. Positive values for level indicate the number of parent directories to search relative to the directory of the module calling [\_\_import\_\_()](https://docs.python.org/3/library/functions.html#__import__) (see [**PEP 328**](https://www.python.org/dev/peps/pep-0328) for the details).

When the name variable is of the form package.module, normally, the top-level package (the name up till the first dot) is returned, not the module named by name. However, when a non-empty fromlist argument is given, the module named by name is returned.

For example, the statement import spam results in bytecode resembling the following code:

spam = \_\_import\_\_('spam', globals(), locals(), [], 0)

The statement import spam.ham results in this call:

spam = \_\_import\_\_('spam.ham', globals(), locals(), [], 0)

Note how [\_\_import\_\_()](https://docs.python.org/3/library/functions.html#__import__) returns the toplevel module here because this is the object that is bound to a name by the [import](https://docs.python.org/3/reference/simple_stmts.html#import) statement.

On the other hand, the statement from spam.ham import eggs, sausage as saus results in

\_temp = \_\_import\_\_('spam.ham', globals(), locals(), ['eggs', 'sausage'], 0)

eggs = \_temp.eggs

saus = \_temp.sausage

Here, the spam.ham module is returned from [\_\_import\_\_()](https://docs.python.org/3/library/functions.html#__import__). From this object, the names to import are retrieved and assigned to their respective names.

If you simply want to import a module (potentially within a package) by name, use [importlib.import\_module()](https://docs.python.org/3/library/importlib.html#importlib.import_module).

*Changed in version 3.3:*Negative values for level are no longer supported (which also changes the default value to 0).

**Foot**