Python-lib

2021年9月29日　　10:51

https://docs.python.org/zh-cn/3/library/index.html

内置函数

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

abs(x)

返回一个数的绝对值。 参数可以是整数、浮点数或任何实现了 __abs__() 的对象。 如果参数是一个复数，则返回它的模。

all(iterable)

如果 iterable 的所有元素均为真值（或可迭代对象为空）则返回 True。等价于:

```
def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True
```

any(iterable)

如果 iterable 的任一元素为真值则返回 True。 如果可迭代对象为空，返回 False。等价于:

```
def any(iterable):
    for element in iterable:
        if element:
            return True
    return False
```

ascii(object)

与 repr() 类似，返回一个包含对象的可打印表示形式的字符串，但是使用 \x、\u 和 \U 对 repr() 返回的字符串中非 ASCII 编码的字符进行转义。 生成的字符串和 Python 2 的 repr() 返回的结果相似。

bin(x)

将一个整数转变为一个前缀为“0b”的二进制字符串。结果是一个合法的 Python 表达式。如果 x 不是 Python 的 int 对象，那它需要定义 __index__() 方法返回一个整数。

```
>>>bin(3)
'0b11'
>>>bin(-10)
'-0b1010'
```

。。自然数，不是 反码。。。反码没有 - 。。。。

如果不一定需要前缀“0b”，还可以使用如下的方法。

```
>>>format(14, '#b'), format(14, 'b')
('0b1110', '1110')
>>>f'{14:#b}', f'{14:b}'
('0b1110', '1110')
```

class bool([x])

返回一个布尔值，True 或者 False。 x 使用标准的 真值测试过程 来转换。如果 x 是假的或者被省略，返回 False；其他情况返回 True。bool 类是 int 的子类。

其他类不能继承自它。它只有 False 和 True 两个实例（参见 布尔值）。

在 3.7 版更改: x 现在只能作为位置参数。

breakpoint(*args, **kws)

此函数会在调用时将你陷入调试器中。具体来说，它调用 sys.breakpointhook() ，直接传递 args 和 kws 。默认情况下， sys.breakpointhook() 调用 pdb.set_trace() 且没有参数。在这种情况下，它纯粹是一个便利函数，因此您不必显式导入 pdb 且键入尽可能少的代码即可进入调试器。

3.7 新版功能.

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

返回一个新的 bytes 数组。 bytearray 类是一个可变序列，包含范围为 0 <= x < 256 的整数。它有可变序列大部分常见的方法，见 可变序列类型 的描述；同时有 bytes 类型的大部分方法

可选形参 source 可以用不同的方式来初始化数组：

如果是一个 string，您必须提供 encoding 参数（errors 参数仍是可选的）；bytearray() 会使用 str.encode() 方法来将 string 转变成 bytes。

如果是一个 integer，会初始化大小为该数字的数组，并使用 null 字节填充。

如果是一个遵循 缓冲区接口 的对象，该对象的只读缓冲区将被用来初始化字节数组。

如果是一个 iterable 可迭代对象，它的元素的范围必须是 0 <= x < 256 的整数，它会被用作数组的初始内容。

如果没有实参，则创建大小为 0 的数组。

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

返回一个新的“bytes”对象， 是一个不可变序列，包含范围为 0 <= x < 256 的整

数。bytes 是 bytearray 的不可变版本 - 它有其中不改变序列的方法和相同的索引、切片操作。
因此，构造函数的实参和 bytearray() 相同。
字节对象还可以用字面值创建，参见 字符串与字节串字面值。

callable(object)
如果参数 object 是可调用的就返回 True，否则返回 False。 如果返回 True，调用仍可能失败，但如果返回 False，则调用 object 将肯定不会成功。 请注意类是可调用的（调用类将返回一个新的实例）；如果实例所属的类有 __call__() 则它就是可调用的。
3.2 新版功能: 这个函数一开始在 Python 3.0 被移除了，但在 Python 3.2 被重新加入。

chr(i)
返回 Unicode 码位为整数 i 的字符的字符串格式。例如，chr(97) 返回字符串 'a'，chr(8364) 返回字符串 '€'。这是 ord() 的逆函数。
实参的合法范围是 0 到 1,114,111（16 进制表示是 0x10FFFF）。如果 i 超过这个范围，会触发 ValueError 异常。

@classmethod
把一个方法封装成类方法。
一个类方法把类自己作为第一个实参，就像一个实例方法把实例自己作为第一个实参。请用以下习惯来声明类方法:

```
class C:
    @classmethod
    def f(cls, arg1, arg2, ...): ...
```

@classmethod 这样的形式称为函数的 decorator -- 详情参阅 函数定义。
类方法的调用可以在类上进行（例如 C.f()）也可以在实例上进行（例如 C().f()）。 其所属类以外的类实例会被忽略。 如果类方法在其所属类的派生类上调用，则该派生类对象会被作为隐含的第一个参数被传入。
类方法与 C++ 或 Java 中的静态方法不同。 如果你需要后者，请参阅本节中的 staticmethod()。 有关类方法的更多信息，请参阅 标准类型层级结构。

compile(source, filename, mode, flags=0, dont_inherit=False, optimize=-1)
将 source 编译成代码或 AST 对象。代码对象可以被 exec() 或 eval() 执行。source 可以是常规的字符串、字节字符串，或者 AST 对象。参见 ast 模块的文档了解如何使用 AST 对象。
filename 实参需要是代码读取的文件名；如果代码不需要从文件中读取，可以传入一些可辨识的值（经常会使用 '<string>'）。
mode 实参指定了编译代码必须用的模式。如果 source 是语句序列，可以是 'exec'；如果是单一表达式，可以是 'eval'；如果是单个交互式语句，可以是 'single'。（在最后一种情况下，如果表达式执行结果不是 None 将会被打印出来。）
可选参数 flags 和 dont_inherit 控制应当激活哪个 编译器选项 以及应当允许哪个 future 特性。

class complex([real[, imag]])
返回值为 real + imag*1j 的复数，或将字符串或数字转换为复数。
如果第一个形参是字符串，则它被解释为一个复数，并且函数调用时必须没有第二个形参。第二个形参不能是字符串。每个实参都可以是任意的数值类型（包括复数）。如果省略了 imag，则默认值为零，构造函数会像 int 和 float 一样进行数值转换。如果两个

实参都省略，则返回 0j。
对于一个普通 Python 对象 x，complex(x) 会委托给 x.__complex__()。 如果 __complex__() 未定义则将回退至 __float__()。 如果 __float__() 未定义则将回退至 __index__()。
当从字符串转换时，字符串在 + 或 - 的周围必须不能有空格。例如 complex('1+2j') 是合法的，但 complex('1 + 2j') 会触发 ValueError 异常。

delattr(object, name)
setattr() 相关的函数。实参是一个对象和一个字符串。该字符串必须是对象的某个属性。如果对象允许，该函数将删除指定的属性。例如 delattr(x, 'foobar') 等价于 del x.foobar 。

class dict(**kwarg)
class dict(mapping, **kwarg)
class dict(iterable, **kwarg)
创建一个新的字典。dict 对象是一个字典类。参见 dict 和 映射类型 --- dict 了解这个类。
其他容器类型，请参见内置的 list、set 和 tuple 类，以及 collections 模块。

dir([object])
如果没有实参，则返回当前本地作用域中的名称列表。如果有实参，它会尝试返回该对象的有效属性列表。
如果对象有一个名为 __dir__() 的方法，那么该方法将被调用，并且必须返回一个属性列表。这允许实现自定义 __getattr__() 或 __getattribute__() 函数的对象能够自定义 dir() 来报告它们的属性。
如果对象不提供 __dir__()，这个函数会尝试从对象已定义的 __dict__ 属性和类型对象收集信息。结果列表并不总是完整的，如果对象有自定义 __getattr__()，那结果可能不准确。
默认的 dir() 机制对不同类型的对象行为不同，它会试图返回最相关而不是最全的信息：
如果对象是模块对象，则列表包含模块的属性名称。
如果对象是类型或类对象，则列表包含它们的属性名称，并且递归查找所有基类的属性。
否则，列表包含对象的属性名称，它的类属性名称，并且递归查找它的类的所有基类的属性。

divmod(a, b)
它将两个（非复数）数字作为实参，并在执行整数除法时返回一对商和余数。
对于浮点数，结果是 (q, a % b) ，q 通常是 math.floor(a / b) 但可能会比 1 小。在任何情况下， q * b + a % b 和 a 基本相等；如果 a % b 非零，它的符号和 b 一样，并且 0 <= abs(a % b) < abs(b) 。

enumerate(iterable, start=0)
返回一个枚举对象。iterable 必须是一个序列，或 iterator，或其他支持迭代的对象。enumerate() 返回的迭代器的 __next__() 方法返回一个元组，里面包含一个计数值（从 start 开始，默认为 0）和通过迭代 iterable 获得的值。
```
>>>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')]
等价于:
def enumerate(sequence, start=0):
    n = start
    for elem in sequence:
        yield n, elem
        n += 1
```

eval(expression[, globals[, locals]])

实参是一个字符串，以及可选的 globals 和 locals。globals 实参必须是一个字典。locals 可以是任何映射对象。

expression 参数会作为一个 Python 表达式（从技术上说是一个条件列表）被解析并求值，并使用 globals 和 locals 字典作为全局和局部命名空间。 如果 globals 字典存在且不包含以 __builtins__ 为键的值，则会在解析 expression 之前插入以此为键的对内置模块 builtins 的引用。 这意味着 expression 通常具有对标准 builtins 模块的完全访问权限且受限的环境会被传播。 如果省略 locals 字典则其默认值为 globals 字典。 如果两个字典同时省略，则表达式执行时会使用 eval() 被调用的环境中的 globals 和 locals。 请注意，eval() 并没有对外围环境下的 (非局部) 嵌套作用域 的访问权限。

exec(object[, globals[, locals]])

这个函数支持动态执行 Python 代码。 object 必须是字符串或者代码对象。 如果是字符串，那么该字符串将被解析为一系列 Python 语句并执行（除非发生语法错误）。 如果是代码对象，它将被直接执行。 在任何情况下，被执行的代码都应当是有效的文件输入（见参考手册中的“文件输入”一节）。 请注意即使在传递给 exec() 函数的代码的上下文中，nonlocal, yield 和 return 语句也不能在函数定义以外使用。 该函数的返回值是 None。

filter(function, iterable)

用 iterable 中函数 function 返回真的那些元素，构建一个新的迭代器。iterable 可以是一个序列，一个支持迭代的容器，或一个迭代器。如果 function 是 None ，则会假设它是一个身份函数，即 iterable 中所有返回假的元素会被移除。

filter(function, iterable) 相当于一个生成器表达式，当 function 不是 None 的时候为 (item for item in iterable if function(item))；function 是 None 的时候为 (item for item in iterable if item) 。

class float([x])

返回从数字或字符串 x 生成的浮点数。

如果实参是字符串，则它必须是包含十进制数字的字符串，字符串前面可以有符号，之前也可以有空格。可选的符号有 '+' 和 '-' ； '+' 对创建的值没有影响。实参也可以是 NaN（非数字）、正负无穷大的字符串。

format(value[, format_spec])

将 value 转换为 format_spec 控制的“格式化”表示。format_spec 的解释取决于 value 实参的类型，但是大多数内置类型使用标准格式化语法：格式规格迷你语言。

调用 format(value, format_spec) 会转换成 type(value).__format__(value, format_spec) ，所以实例字典中的 __format__() 方法将不会调用。

class frozenset([iterable])

返回一个新的 frozenset 对象，它包含可选参数 iterable 中的元素。 frozenset 是一个内置的类。

getattr(object, name[, default])

返回对象命名属性的值。name 必须是字符串。如果该字符串是对象的属性之一，则返回该属性的值。例如， getattr(x, 'foobar') 等同于 x.foobar。如果指定的属性不存在，且提供了 default 值，则返回它，否则触发 AttributeError。

注解：由于 私有名称混合 发生在编译时，因此必须 手动混合私有属性（以两个下划线打头的属性）名称以使使用 getattr() 来提取它。

globals()

返回表示当前全局符号表的字典。这总是当前模块的字典（在函数或方法中，不是调用它的模块，而是定义它的模块）。

hasattr(object, name)

该实参是一个对象和一个字符串。如果字符串是对象的属性之一的名称，则返回 True，否则返回 False。（此功能是通过调用 getattr(object, name) 看是否有 AttributeError 异常来实现的。）

hash(object)

返回该对象的哈希值（如果它有的话）。哈希值是整数。它们在字典查找元素时用来快速比较字典的键。相同大小的数字变量有相同的哈希值（即使它们类型不同，如 1 和 1.0）。

如果对象实现了自己的 __hash__() 方法，请注意，hash() 根据机器的字长来截断返回值。另请参阅 __hash__()。

help([object])

启动内置的帮助系统（此函数主要在交互式中使用）。如果没有实参，解释器控制台里会启动交互式帮助系统。如果实参是一个字符串，则在模块、函数、类、方法、关键字或文档主题中搜索该字符串，并在控制台上打印帮助信息。如果实参是其他任意对象，则会生成该对象的帮助页。

hex(x)

将整数转换为以“0x”为前缀的小写十六进制字符串。如果 x 不是 Python int 对象，则必须定义返回整数的 __index__() 方法。

```
>>>hex(255)
'0xff'
>>>hex(-42)
'-0x2a'
```

如果要将整数转换为大写或小写的十六进制字符串，并可选择有无“0x”前缀，则可以使用如下方法

```
>>>'%#x' % 255, '%x' % 255, '%X' % 255
('0xff', 'ff', 'FF')
>>>format(255, '#x'), format(255, 'x'), format(255, 'X')
('0xff', 'ff', 'FF')
>>>f'{255:#x}', f'{255:x}', f'{255:X}'
('0xff', 'ff', 'FF')
```

如果要获取浮点数的十六进制字符串形式，请使用 float.hex() 方法。

id(object)

返回对象的“标识值”。该值是一个整数，在此对象的生命周期中保证是唯一且恒定的。两个生命期不重叠的对象可能具有相同的 id() 值。

input([prompt])

如果存在 prompt 实参，则将其写入标准输出，末尾不带换行符。接下来，该函数从输入中读取一行，将其转换为字符串（除了末尾的换行符）并返回。当读取到 EOF 时，则触发 EOFError。

class int([x])
class int(x, base=10)

返回一个基于数字或字符串 x 构造的整数对象，或者在未给出参数时返回 0。 如果 x 定义了 __int__()，int(x) 将返回 x.__int__()。 如果 x 定义了 __index__()，它将返回 x.__index__()。 如果 x 定义了 __trunc__()，它将返回 x.__trunc__()。 对于浮点数，它将向零舍入。

认的 base 为 10 ，允许的进制有 0、2-36。2、8、16 进制的数字可以在代码中用 0b/0B 、 0o/0O 、 0x/0X 前缀来表示。

isinstance(object, classinfo)

如果参数 object 是参数 classinfo 的实例或者是其（直接、间接或 虚拟）子类则返回 True。如果 object 不是给定类型的对象，函数将总是返回 False。 如果 classinfo 是类型对象元组（或由其他此类元组递归组成的元组），那么如果 object 是其中任何一个类型的实例就返回 True。

issubclass(class, classinfo)

Return True if class is a subclass (direct, indirect or virtual) of classinfo.

。。。还有其他的描述，和上面一样， 只不过上面是instance ，这里是class。

iter(object[, sentinel])

返回一个 iterator 对象。根据是否存在第二个实参，第一个实参的解释是非常不同的。如果没有第二个实参，object 必须是支持迭代协议（有 __iter__() 方法）的集合对象，或必须支持序列协议（有 __getitem__() 方法，且数字参数从 0 开始）。如果它不支持这些协议，会触发 TypeError。如果有第二个实参 sentinel，那么 object 必须是可调用的对象。这种情况下生成的迭代器，每次迭代调用它的 __next__() 方法时都会不带实参地调用 object；如果返回的结果是 sentinel 则触发 StopIteration，否则返回调用结果。

适合 iter() 的第二种形式的应用之一是构建块读取器。 例如，从二进制数据库文件中读取固定宽度的块，直至到达文件的末尾:

```
from functools import partial
with open('mydata.db', 'rb') as f:
    for block in iter(partial(f.read, 64), b''):
        process_block(block)
```

len(s)

返回对象的长度（元素个数）。实参可以是序列（如 string、bytes、tuple、list 或 range 等）或集合（如 dictionary、set 或 frozen set 等）。

class list([iterable])

虽然被称为函数，list 实际上是一种可变序列类型

locals()

更新并返回表示当前本地符号表的字典。 在函数代码块但不是类代码块中调用 locals() 时将返回自由变量。 请注意在模块层级上，locals() 和 globals() 是同一个字典。

map(function, iterable, ...)

返回一个将 function 应用于 iterable 中每一项并输出其结果的迭代器。 如果传入了额外的 iterable 参数，function 必须接受相同个数的实参并被应用于从所有可迭代对象中并行获取的项。 当有多个可迭代对象时，最短的可迭代对象耗尽则整个迭代就将结束。

max(iterable, *[, key, default])
max(arg1, arg2, *args[, key])

返回可迭代对象中最大的元素，或者返回两个及以上实参中最大的。
如果只提供了一个位置参数，它必须是非空 iterable，返回可迭代对象中最大的元素；
如果提供了两个及以上的位置参数，则返回最大的位置参数。

class memoryview(object)

返回由给定实参创建的“内存视图”对象。

min(iterable, *[, key, default])
min(arg1, arg2, *args[, key])

返回可迭代对象中最小的元素，或者返回两个及以上实参中最小的。
如果只提供了一个位置参数，它必须是 iterable，返回可迭代对象中最小的元素；如果提供了两个及以上的位置参数，则返回最小的位置参数。

next(iterator[, default])

通过调用 iterator 的 __next__() 方法获取下一个元素。如果迭代器耗尽，则返回给定的 default，如果没有默认值则触发 StopIteration。

class object

返回一个没有特征的新对象。object 是所有类的基类。它具有所有 Python 类实例的通用方法。这个函数不接受任何实参。
由于 object 没有 __dict__，因此无法将任意属性赋给 object 的实例。

oct(x)

将一个整数转变为一个前缀为“0o”的八进制字符串。结果是一个合法的 Python 表达式。如果 x 不是 Python 的 int 对象，那它需要定义 __index__() 方法返回一个整数

```
oct(8)
'0o10'
oct(-56)
'-0o70'

'%#o' % 10, '%o' % 10
('0o12', '12')
format(10, '#o'), format(10, 'o')
('0o12', '12')
f'{10:#o}', f'{10:o}'
```

('0o12', '12')

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

打开 file 并返回对应的 file object。 如果该文件不能被打开，则引发 OSError。 请参阅 读写文件 获取此函数的更多用法示例。

file 是一个 path-like object，表示将要打开的文件的路径（绝对路径或者当前工作目录的相对路径），也可以是要被封装的整数类型文件描述符。（如果是文件描述符，它会随着返回的 I/O 对象关闭而关闭，除非 closefd 被设为 False 。）

mode 是一个可选字符串，用于指定打开文件的模式。默认值是 'r' ，这意味着它以文本模式打开并读取。

字符	意义
'r'	读取（默认）
'w'	写入，并先截断文件
'x'	排它性创建，如果文件已存在则失败
'a'	写入，如果文件存在则在末尾追加
'b'	二进制模式
't'	文本模式（默认）
'+'	打开用于更新（读取与写入）

默认模式为 'r' (打开用于读取文本，与 'rt' 同义)。 模式 'w+' 与 'w+b' 将打开文件并清空内容。 模式 'r+' 与 'r+b' 将打开文件并不清空内容。

此外还允许使用一个模式字符 'U'，该字符已不再具有任何效果，并被视为已弃用。

buffering 是一个可选的整数，用于设置缓冲策略。传递0以切换缓冲关闭（仅允许在二进制模式下），1选择行缓冲（仅在文本模式下可用），并且>1的整数以指示固定大小的块缓冲区的大小（以字节为单位）。如果没有给出 buffering 参数，则默认缓冲策略的工作方式如下:

二进制文件以固定大小的块进行缓冲；使用启发式方法选择缓冲区的大小，尝试确定底层设备的“块大小”或使用 io.DEFAULT_BUFFER_SIZE。在许多系统上，缓冲区的长度通常为4096或8192字节。

“交互式”文本文件（ isatty() 返回 True 的文件）使用行缓冲。其他文本文件使用上述策略用于二进制文件。

encoding 是用于解码或编码文件的编码的名称。这应该只在文本模式下使用。默认编码是依赖于平台的（不 管 locale.getpreferredencoding() 返回何值），但可以使用任何 Python支持的 text encoding 。

errors 是一个可选的字符串参数，用于指定如何处理编码和解码错误 - 这不能在二进制模式下使用。可以使用各种标准错误处理程序（列在 错误处理方案 ），但是使用 codecs.register_error() 注册的任何错误处理名称也是有效的。标准名称包括:

如果存在编码错误，'strict' 会引发 ValueError 异常。 默认值 None 具有相同的效果。

'ignore' 忽略错误。请注意，忽略编码错误可能会导致数据丢失。

'replace' 会将替换标记（例如 '?' ）插入有错误数据的地方。

'surrogateescape' 将把任何不正确的字节表示为 U+DC80 至 U+DCFF 范围内的下方替代码位。 当在写入数据时使用 surrogateescape 错误处理句柄时这些替代码位会被转回到相同的字节。 这适用于处理具有未知编码格式的文件。

只有在写入文件时才支持 'xmlcharrefreplace'。编码不支持的字符将替换为相应的XML字符引用 &#nnn;。

'backslashreplace' 用Python的反向转义序列替换格式错误的数据。

'namereplace' （也只在编写时支持）用 \N{...} 转义序列替换不支持的字符。

newline 控制 universal newlines 模式如何生效（它仅适用于文本模式）。它可以是 None，''，'\n'，'\r' 和 '\r\n'。它的工作原理:

从流中读取输入时，如果 newline 为 None，则启用通用换行模式。输入中的行可以以 '\n'，'\r' 或 '\r\n' 结尾，这些行被翻译成 '\n' 在返回呼叫者之前。如果它是 ''，则启用通用换行模式，但行结尾将返回给调用者未翻译。如果它具有任何其他合法值，则输入行仅由给定字符串终止，并且行结尾将返回给未调用的调用者。

将输出写入流时，如果 newline 为 None，则写入的任何 '\n' 字符都将转换为系统默认行分隔符 os.linesep。如果 newline 是 '' 或 '\n'，则不进行翻译。如果 newline 是任何其他合法值，则写入的任何 '\n' 字符将被转换为给定的字符串。

使用 os.open() 函数的 dir_fd 的形参，从给定的目录中用相对路径打开文件:

```
>>> 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
```

open() 函数所返回的 file object 类型取决于所用模式。 当使用 open() 以文本模式 ('w', 'r', 'wt', 'rt' 等) 打开文件时，它将返回 io.TextIOBase (特别是 io.TextIOWrapper) 的一个子类。 当使用缓冲以二进制模式打开文件时，返回的类是 io.BufferedIOBase 的一个子类。 具体的类会有多种：在只读的二进制模式下，它将返回 io.BufferedReader；在写入二进制和追加二进制模式下，它将返回 io.BufferedWriter，而在读/写模式下，它将返回 io.BufferedRandom。 当禁用缓冲时，则会返回原始流，即 io.RawIOBase 的一个子类 io.FileIO。

ord(c)

对表示单个 Unicode 字符的字符串，返回代表它 Unicode 码点的整数。例如 ord('a') 返回整数 97， ord('€') （欧元符号）返回 8364 。这是 chr() 的逆函数。

pow(base, exp[, mod])

返回 base 的 exp 次幂；如果 mod 存在，则返回 base 的 exp 次幂对 mod 取余（比 pow(base, exp) % mod 更高效）。 两参数形式 pow(base, exp) 等价于乘方运算符: base**exp。

The arguments must have numeric types. With mixed operand types, the coercion

rules for binary arithmetic operators apply. For 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, pow(10, 2) returns 100, but pow(10, -2) returns 0.01.

。。参数必须是数值类型，混合使用的话，都转为float，然后计算。

对于 int 操作数 base 和 exp，如果给出 mod，则 mod 必须为整数类型并且 mod 必须不为零。 如果给出 mod 并且 exp 为负值，则 base 必须相对于 mod 不可整除。 在这种情况下，将会返回 pow(inv_base, -exp, mod)，其中 inv_base 为 base 的倒数对 mod 取余。

下面的例子是 38 的倒数对 97 取余:

```
>>> pow(38, -1, mod=97)
23
>>> 23 * 38 % 97 == 1
True
```

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

将 objects 打印到 file 指定的文本流，以 sep 分隔并在末尾加上 end。 sep, end, file 和 flush 如果存在，它们必须以关键字参数的形式给出。

所有非关键字参数都会被转换为字符串，就像是执行了 str() 一样，并会被写入到流，以 sep 且在末尾加上 end。 sep 和 end 都必须为字符串；它们也可以为 None，这意味着使用默认值。 如果没有给出 objects，则 print() 将只写入 end。

file 参数必须是一个具有 write(string) 方法的对象；如果参数不存在或为 None，则将使用 sys.stdout。 由于要打印的参数会被转换为文本字符串，因此 print() 不能用于二进制模式的文件对象。 对于这些对象，应改用 file.write(...)。

输出是否被缓存通常决定于 file，但如果 flush 关键字参数为真值，流会被强制刷新。

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

返回 property 属性。

fget 是获取属性值的函数。 fset 是用于设置属性值的函数。 fdel 是用于删除属性值的函数。并且 doc 为属性对象创建文档字符串。

一个典型的用法是定义一个托管属性 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.")
```

如果 c 是 C 的实例，c.x 将调用getter，c.x = value 将调用setter， del c.x 将调

用deleter。

如果给出，doc 将成为该 property 属性的文档字符串。 否则该 property 将拷贝 fget 的文档字符串（如果存在）。

这令使用 property() 作为 decorator 来创建只读的特征属性可以很容易地实现:

```
class Parrot:
    def __init__(self):
        self._voltage = 100000

    @property
    def voltage(self):
        """Get the current voltage."""
        return self._voltage
```

以上 @property 装饰器会将 voltage() 方法转化为一个具有相同名称的只读属性的 "getter"，并将 voltage 的文档字符串设置为 "Get the current voltage."

特征属性对象具有 getter, setter 以及 deleter 方法，它们可用作装饰器来创建该特征属性的副本，并将相应的访问函数设为所装饰的函数。 这最好是用一个例子来解释:

```
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
```

。。。喵喵喵。。。

上述代码与第一个例子完全等价。 注意一定要给附加函数与原始的特征属性相同的名称 (在本例中为 x。)

返回的特征属性对象同样具有与构造器参数相对应的属性 fget, fset 和 fdel。

class range(stop)

class range(start, stop[, step])

虽然被称为函数，但 range 实际上是一个不可变的序列类型

repr(object)

返回包含一个对象的可打印表示形式的字符串。 对于许多类型来说，该函数会尝试返回的字符串将会与该对象被传递给 eval() 时所生成的对象具有相同的值，在其他情况下表

示形式会是一个括在尖括号中的字符串，其中包含对象类型的名称与通常包括对象名称和地址的附加信息。 类可以通过定义 __repr__() 方法来控制此函数为它的实例所返回的内容。

reversed(seq)
返回一个反向的 iterator。 seq 必须是一个具有 __reversed__() 方法的对象或者是支持该序列协议（具有从 0 开始的整数类型参数的 __len__() 方法和 __getitem__() 方法）。

round(number[, ndigits])
返回 number 舍入到小数点后 ndigits 位精度的值。 如果 ndigits 被省略或为 None，则返回最接近输入值的整数。
对于支持 round() 的内置类型，值会被舍入到最接近的 10 的负 ndigits 次幂的倍数；如果与两个倍数的距离相等，则选择偶数（因此，round(0.5) 和 round(-0.5) 均为 0 而 round(1.5) 为 2）。 任何整数值都可作为有效的 ndigits（正数、零或负数）。 如果 ndigits 被省略或为 None 则返回值将为整数。 否则返回值与 number 的类型相同。

class set([iterable])
返回一个新的 set 对象，可以选择带有从 iterable 获取的元素。 set 是一个内置类型。

setattr(object, name, value)
此函数与 getattr() 两相对应。 其参数为一个对象、一个字符串和一个任意值。 字符串指定一个现有属性或者新增属性。 函数会将值赋给该属性，只要对象允许这种操作。
由于 私有名称混合 发生在编译时，因此必须手动混合私有属性（以两个下划线打头的属性）名称以便使用 setattr() 来设置它。

class slice(stop)
class slice(start, stop[, step])
切片对象也会在使用扩展索引语法时被生成。 例如: a[start:stop:step] 或 a[start:stop, i]。

sorted(iterable, *, key=None, reverse=False)
根据 iterable 中的项返回一个新的已排序列表。
key 指定带有单个参数的函数，用于从 iterable 的每个元素中提取用于比较的键 （例如 key=str.lower）。 默认值为 None（直接比较元素）。
reverse 为一个布尔值。 如果设为 True，则每个列表元素将按反向顺序比较进行排序。
使用 functools.cmp_to_key() 可将老式的 cmp 函数转换为 key 函数。
内置的 sorted() 确保是稳定的。 如果一个排序确保不会改变比较结果相等的元素的相对顺序就称其为稳定的

@staticmethod
将方法转换为静态方法。
静态方法不会接收隐式的第一个参数。要声明一个静态方法，请使用此语法
```
class C:
    @staticmethod
    def f(arg1, arg2, ...): ...
```

@staticmethod 这样的形式称为函数的 decorator

静态方法的调用可以在类上进行（例如 C.f()）也可以在实例上进行（例如 C().f()）。Python中的静态方法与Java或C ++中的静态方法类似。另请参阅 classmethod() ，用于创建备用类构造函数的变体。

像所有装饰器一样，也可以像常规函数一样调用 staticmethod ，并对其结果执行某些操作。比如某些情况下需要从类主体引用函数并且您希望避免自动转换为实例方法。对于这些情况，请使用此语法:

```
class C:
    builtin_open = staticmethod(open)
```

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

返回一个 str 版本的 object 。

sum(iterable, /, start=0)

从 start 开始自左向右对 iterable 的项求和并返回总计值。 iterable 的项通常为数字，而 start 值则不允许为字符串。

对某些用例来说，存在 sum() 的更好替代。 拼接字符串序列的更好更快方式是调用 ''.join(sequence)。 要以扩展精度对浮点值求和，请参阅 math.fsum()。 要拼接一系列可迭代对象，请考虑使用 itertools.chain()。

super([type[, object-or-type]])

返回一个代理对象，它会将方法调用委托给 type 的父类或兄弟类。 这对于访问已在类中被重载的继承方法很有用。

super 有两个典型用例。 在具有单继承的类层级结构中，super 可用来引用父类而不必显式地指定它们的名称，从而令代码更易维护。 这种用法与其他编程语言中 super 的用法非常相似。

第二个用例是在动态执行环境中支持协作多重继承。 此用例为 Python 所独有而不存在于静态编码语言或仅支持单继承的语言当中。 这使用实现“菱形图”成为可能，即有多个基类实现相同的方法。 好的设计强制要求这样的方法在每个情况下都具有相同的调用签名（因为调用顺序是在运行时确定的，也因为这个顺序要适应类层级结构的更改，还因为这个顺序可能包括在运行时之前未知的兄弟类）。

对于以上两个用例，典型的超类调用看起来是这样的:

```
class C(B):
    def method(self, arg):
        super().method(arg)    # This does the same thing as:
                               # super(C, self).method(arg)
```

除了方法查找之外，super() 也可用于属性查找。

class tuple([iterable])

虽然被称为函数，但 tuple 实际上是一个不可变的序列类型

class type(object)
class type(name, bases, dict, **kwds)

传入一个参数时，返回 object 的类型。 返回值是一个 type 对象，通常与 object.__class__ 所返回的对象相同。
推荐使用 isinstance() 内置函数来检测对象的类型，因为它会考虑子类的情况。

传入三个参数时，返回一个新的 type 对象。 这在本质上是 class 语句的一种动态形式，name 字符串即类名并会成为 __name__ 属性；bases 元组包含基类并会成为 __bases__ 属性；如果为空则会添加所有类的终极基类 object。 dict 字典包含类主体的属性和方法定义；它在成为 __dict__ 属性之前可能会被拷贝或包装。 下面两条语句会创建相同的 type 对象:

```
class X:
    a = 1

X = type('X', (), dict(a=1))
```

vars([object])

返回模块、类、实例或任何其它具有 __dict__ 属性的对象的 __dict__ 属性。
模块和实例这样的对象具有可更新的 __dict__ 属性；但是，其它对象的 __dict__ 属性可能会设为限制写入
不带参数时，vars() 的行为类似 locals()。 请注意，locals 字典仅对于读取起作用，因为对 locals 字典的更新会被忽略。
如果指定了一个对象但它没有 __dict__ 属性（例如，当它所属的类定义了 __slots__ 属性时）则会引发 TypeError 异常。

zip(*iterables)

创建一个聚合了来自每个可迭代对象中的元素的迭代器。
返回一个元组的迭代器，其中的第 i 个元组包含来自每个参数序列或可迭代对象的第 i 个元素。 当所输入可迭代对象中最短的一个被耗尽时，迭代器将停止迭代。 当只有一个可迭代对象参数时，它将返回一个单元组的迭代器。 不带参数时，它将返回一个空迭代器。
相当于:

```
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)
```

函数会保证可迭代对象按从左至右的顺序被求值。 使得可以通过 zip(*[iter(s)]*n) 这样的惯用形式将一系列数据聚类为长度为 n 的分组。 这将重复 同样的 迭代器 n 次，以便每个输出的元组具有第 n 次调用该迭代器的结果。 它的作用效果就是将输入拆分为长度为 n 的数据块。

当你不用关心较长可迭代对象末尾不匹配的值时，则 zip() 只须使用长度不相等的输入

即可。 如果那些值很重要，则应改用 itertools.zip_longest()。

zip() 与 * 运算符相结合可以用来拆解一个列表:

```
>>> 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
```

。。怎么没有unzip 。。。我记得好像有的啊。

__import__(name, globals=None, locals=None, fromlist=(), level=0)

与 importlib.import_module() 不同，这是一个日常 Python 编程中不需要用到的高级函数

此函数会由 import 语句发起调用。 它可以被替换（通过导入 builtins 模块并赋值给 builtins.__import__）以便修改 import 语句的语义，但是 强烈 不建议这样做，因为使用导入钩子（参见 PEP 302）通常更容易实现同样的目标，并且不会导致代码问题，因为许多代码都会假定所用的是默认实现。 同样也不建议直接使用 __import__() 而应该用 importlib.import_module()。

内置常量

False	bool 类型的假值。
True	bool 类型的真值
None	NoneType 类型的唯一值。 None 经常用于表示缺少值，当因为默认参数未传递给函数时
NotImplemented	双目运算特殊方法（如 __eq__(), __lt__(), __add__(), __rsub__() 等）应返回的特殊值，用于表示运算没有针对其他类型的实现；也可由原地双目运算特殊方法（如 __imul__(), __iand__() 等）出于同样的目的而返回。 它不应被作为布尔值来解读。
Ellipsis	与省略号文字字面 “...” 相同。 特殊值主要与用户定义的容器数据类型的扩展切片语法结合使用。
__debug_	如果 Python 没有以 -O 选项启动，则此常量为真值。 另请参见 assert 语句

_	

由 site 模块添加的常量
site 模块（在启动期间自动导入，除非给出 -S 命令行选项）将几个常量添加到内置命名空间。 它们对交互式解释器 shell 很有用，并且不应在程序中使用。
quit
exit
copyright
credits
license

。。。。。md，太多了，用的时候网上查吧，下面是部分。

https://docs.python.org/zh-cn/3/library/stdtypes.html
内置类型

not 的优先级比非布尔运算符低，因此 not a == b 会被解读为 not (a == b) 而 a == not b 会引发语法错误。

x < y <= z 等价于 x < y and y <= z，前者的不同之处在于 y 只被求值一次（但在两种情况下当 x < y 结果为假值时 z 都不会被求值）。

整数，浮点数 和 复数。

浮点数通常使用 C 中的 double 来实现；有关你的程序运行所在机器上浮点数的精度和内部表示法可在 sys.float_info 中查看。

一个复数 z 中提取这两个部分，可使用 z.real 和 z.imag

二元算术运算符的操作数有不同数值类型时，"较窄"类型的操作数会拓宽到另一个操作数的类型，其中整数比浮点数窄，浮点数比复数窄。

所有 numbers.Real 类型 (int 和 float) 还包括下列运算:

运算	结果
math.trunc(x)	x 截断为 Integral
round(x[, n])	x 舍入到 n 位小数，半数值会舍入到偶数。 如果省略 n，则默认为 0。
math.floor(x)	<= x 的最大 Integral
math.ceil(x)	>= x 的最小 Integral

有关更多的数字运算请参阅 math 和 cmath 模块。

https://docs.python.org/zh-cn/3/library/math.html#module-math
https://docs.python.org/zh-cn/3/library/cmath.html#module-cmath
。。c是 complex， 复数

二进制按位运算的优先级全都低于数字运算，但又高于比较运算；一元运算 ~ 具有与其他一元算术运算 (+ and -) 相同的优先级。

运算	结果
x | y	x 和 y 按位 或
x ^ y	x 和 y 按位 异或
x & y	x 和 y 按位 与
x << n	x 左移 n 位
x >> n	x 右移 n 位
~x	x 逐位取反

整数类型的附加方法
int.bit_length()
返回以二进制表示一个整数所需要的位数，不包括符号位和前面的零
```
>>> n = -37
>>> bin(n)
'-0b100101'
>>> n.bit_length()
6
等价于:
def bit_length(self):
    s = bin(self)       # binary representation:  bin(-37) --> '-0b100101'
    s = s.lstrip('-0b') # remove leading zeros and minus sign
    return len(s)       # len('100101') --> 6
```

int.to_bytes(length, byteorder, *, signed=False)
返回表示一个整数的字节数组。
```
(1024).to_bytes(2, byteorder='big')
b'\x04\x00'
(1024).to_bytes(10, byteorder='big')
b'\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00'
(-1024).to_bytes(10, byteorder='big', signed=True)
b'\xff\xff\xff\xff\xff\xff\xff\xff\xfc\x00'
x = 1000
x.to_bytes((x.bit_length() + 7) // 8, byteorder='little')
b'\xe8\x03'
```

classmethod int.from_bytes(bytes, byteorder, *, signed=False)
返回由给定字节数组所表示的整数。
```
int.from_bytes(b'\x00\x10', byteorder='big')
16
```

```
int.from_bytes(b'\x00\x10', byteorder='little')
4096
int.from_bytes(b'\xfc\x00', byteorder='big', signed=True)
-1024
int.from_bytes(b'\xfc\x00', byteorder='big', signed=False)
64512
int.from_bytes([255, 0, 0], byteorder='big')
16711680
```

int.as_integer_ratio()

返回一对整数，其比率正好等于原整数并且分母为正数。 整数的比率总是用这个整数本身作为分子，1 作为分母。

浮点类型的附加方法

float.as_integer_ratio()

返回一对整数，其比率正好等于原浮点数并且分母为正数。 无穷大会引发 OverflowError 而 NaN 则会引发 ValueError。

float.is_integer()

float.hex()

classmethod float.fromhex(s)

数字类型的哈希运算

如果 x = m / n 是一个非负的有理数且 n 不可被 P 整除，则定义 hash(x) 为 m * invmod(n, P) % P，其中 invmod(n, P) 是对 n 模 P 取反。

如果 x = m / n 是一个非负的有理数且 n 可被 P 整除（但 m 不能）则 n 不能对 P 降模，以上规则不适用；在此情况下则定义 hash(x) 为常数值 sys.hash_info.inf。

如果 x = m / n 是一个负的有理数则定义 hash(x) 为 -hash(-x)。 如果结果哈希值为 -1 则将其替换为 -2。

特定值 sys.hash_info.inf, -sys.hash_info.inf 和 sys.hash_info.nan 被用作正无穷、负无穷和空值（所分别对应的）哈希值。 （所有可哈希的空值都具有相同的哈希值。）

对于一个 complex 值 z，会通过计算 hash(z.real) + sys.hash_info.imag * hash(z.imag) 将实部和虚部的哈希值结合起来，并进行降模 2**sys.hash_info.width 以使其处于 range(-2**(sys.hash_info.width - 1), 2**(sys.hash_info.width - 1)) 范围之内。 同样地，如果结果为 -1 则将其替换为 -2。

```
import sys, math

def hash_fraction(m, n):
    """Compute the hash of a rational number m / n.

    Assumes m and n are integers, with n positive.
    Equivalent to hash(fractions.Fraction(m, n)).
```

```
    """
    P = sys.hash_info.modulus
    # Remove common factors of P.  (Unnecessary if m and n already coprime.)
    while m % P == n % P == 0:
        m, n = m // P, n // P

    if n % P == 0:
        hash_value = sys.hash_info.inf
    else:
        # Fermat's Little Theorem: pow(n, P-1, P) is 1, so
        # pow(n, P-2, P) gives the inverse of n modulo P.
        hash_value = (abs(m) % P) * pow(n, P - 2, P) % P
    if m < 0:
        hash_value = -hash_value
    if hash_value == -1:
        hash_value = -2
    return hash_value

def hash_float(x):
    """Compute the hash of a float x."""

    if math.isnan(x):
        return sys.hash_info.nan
    elif math.isinf(x):
        return sys.hash_info.inf if x > 0 else -sys.hash_info.inf
    else:
        return hash_fraction(*x.as_integer_ratio())

def hash_complex(z):
    """Compute the hash of a complex number z."""

    hash_value = hash_float(z.real) + sys.hash_info.imag * hash_float(z.imag)
    # do a signed reduction modulo 2**sys.hash_info.width
    M = 2**(sys.hash_info.width - 1)
    hash_value = (hash_value & (M - 1)) - (hash_value & M)
    if hash_value == -1:
        hash_value = -2
    return hash_value
```

迭代器类型

```
container.__iter__()
iterator.__iter__()
iterator.__next__()
```

生成器类型

序列类型 --- list, tuple, range

通用序列操作
大多数序列类型，包括可变类型和不可变类型都支持下表中的操作。
此表按优先级升序列出了序列操作。 在表格中，s 和 t 是具有相同类型的序列，n, i, j 和 k 是整数而 x 是任何满足 s 所规定的类型和值限制的任意对象。
in 和 not in 操作具有与比较操作相同的优先级。 + (拼接) 和 * (重复) 操作具有与对应数值运算相同的优先级

运算	结果
x in s	如果 s 中的某项等于 x 则结果为 True，否则为 False
x not in s	如果 s 中的某项等于 x 则结果为 False，否则为 True
s + t	s 与 t 相拼接
s * n 或 n * s	相当于 s 与自身进行 n 次拼接
s[i]	s 的第 i 项，起始为 0
s[i:j]	s 从 i 到 j 的切片
s[i:j:k]	s 从 i 到 j 步长为 k 的切片
len(s)	s 的长度
min(s)	s 的最小项
max(s)	s 的最大项
s.index(x[, i[, j]])	x 在 s 中首次出现项的索引号（索引号在 i 或其后且在 j 之前）
s.count(x)	x 在 s 中出现的总次数

不可变序列类型

可变序列类型

s[i] = x	将 s 的第 i 项替换为 x
s[i:j] = t	将 s 从 i 到 j 的切片替换为可迭代对象 t 的内容
del s[i:j]	等同于 s[i:j] = []
s[i:j:k] = t	将 s[i:j:k] 的元素替换为 t 的元素
del s[i:j:k]	从列表中移除 s[i:j:k] 的元素
s.append(x)	将 x 添加到序列的末尾 (等同于 s[len(s):len(s)] = [x])
s.clear()	从 s 中移除所有项 (等同于 del s[:])
s.copy()	创建 s 的浅拷贝 (等同于 s[:])
s.extend(t) 或 s += t	用 t 的内容扩展 s (基本上等同于 s[len(s):len(s)] = t)
s *= n	使用 s 的内容重复 n 次来对其进行更新
s.insert(i, x)	在由 i 给出的索引位置将 x 插入 s (等同于 s[i:i] = [x])
s.pop() or s.pop(i)	提取在 i 位置上的项，并将其从 s 中移除

s.remove(x)	删除 s 中第一个 s[i] 等于 x 的项目。
s.reverse()	就地将列表中的元素逆序。

列表
列表是可变序列，通常用于存放同类项目的集合（其中精确的相似程度将根据应用而变化）

class list([iterable])
    可以用多种方式构建列表：
        使用一对方括号来表示空列表: []
        使用方括号，其中的项以逗号分隔: [a], [a, b, c]
        使用列表推导式: [x for x in iterable]
        使用类型的构造器: list() 或 list(iterable)

构造器将构造一个列表，其中的项与 iterable 中的项具有相同的的值与顺序。 iterable 可以是序列、支持迭代的容器或其它可迭代对象。 如果 iterable 已经是一个列表，将创建并返回其副本，类似于 iterable[:]。 例如，list('abc') 返回 ['a', 'b', 'c'] 而 list((1, 2, 3)) 返回 [1, 2, 3]。 如果没有给出参数，构造器将创建一个空列表 []。

列表实现了所有 一般 和 可变 序列的操作。 列表还额外提供了以下方法：
    sort(*, key=None, reverse=False)
        此方法会对列表进行原地排序，只使用 < 来进行各项间比较。 异常不会被屏蔽

元组
元组是不可变序列，通常用于储存异构数据的多项集（例如由 enumerate() 内置函数所产生的二元组）。 元组也被用于需要同构数据的不可变序列的情况（例如允许存储到 set 或 dict 的实例）。
class tuple([iterable])
    可以用多种方式构建元组：
        使用一对圆括号来表示空元组: ()
        使用一个后缀的逗号来表示单元组: a, 或 (a,)
        使用以逗号分隔的多个项: a, b, c or (a, b, c)
        使用内置的 tuple(): tuple() 或 tuple(iterable)

tuple('abc') 返回 ('a', 'b', 'c') 而 tuple([1, 2, 3]) 返回 (1, 2, 3)。

请注意决定生成元组的其实是逗号而不是圆括号。 圆括号只是可选的，生成空元组或需要避免语法歧义的情况除外。 例如，f(a, b, c) 是在调用函数时附带三个参数，而 f((a, b, c)) 则是在调用函数时附带一个三元组。

range 对象
range 类型表示不可变的数字序列，通常用于在 for 循环中循环指定的次数

class range(stop)

class range(start, stop[, step])

range 构造器的参数必须为整数（可以是内置的 int 或任何实现了 __index__ 特殊方法的对象）。 如果省略 step 参数，其默认值为 1。 如果省略 start 参数，其默认值为 0，如果 step 为零则会引发 ValueError。

如果 step 为正值，确定 range r 内容的公式为 r[i] = start + step*i 其中 i >= 0 且 r[i] < stop。

如果 step 为负值，确定 range 内容的公式仍然为 r[i] = start + step*i，但限制条件改为 i >= 0 且 r[i] > stop.

文本序列类型 --- str

单引号: '允许包含有 "双" 引号'
双引号: "允许包含有 '单' 引号"。
三重引号: '''三重单引号''', """三重双引号"""